Cプログラムから設計仕様情報を導出するプログラム理解システム

3N－6

神保　至†[1]　佐藤徳幸†[2]　大熊義嗣‡

†情報処理振興事業協会(IPA) 技術センター

‡東京電機大学 理工学研究科

1　はじめに

企業のソフトウェア開発部門では、新規ソフトウェアの開発と比較して、既存ソフトウェアの保守作業工数が増大する傾向にある。これは、そのソフトウェアの不完全な仕様書しか残っていないために、多くの時間がそのプログラムの理解作業に費やされていることが一因である。

そのために、最近注目を集めているのがリバースエンジニアリングである。しかし従来のリバースエンジニアリング・ツールの多くは、プログラムに対して構造的な分析のみを行ない、制御フロー図やデータフロー図などを出力するものであった。すなわち、保守作業自体を実施するのに有用な情報が得られる反面、そのプログラムが全体としてあるいは部分的に、どのような機能を果たすプログラムであるか(仕様情報)はわからなかった。このような仕様情報を出力するためには、プログラムの意味的な理解を行なわなければならない。

そこで本稿では、プログラミング教育を主な目的とした、アルゴリズムに基づくプログラム理解手法[1]をリバースエンジニアリングに応用し、C言語プログラムのソースコードから設計仕様書レベルの情報を導出するプログラム理解システムについて述べる。

2　リバースエンジニアリングのためのプログラム理解手法

コンピュータにプログラムを理解させるということは、一定の構造(モデル)で表現された知識を使ってプログラムの意味を推論するということである。

プログラミング教育を目的としたプログラム理解では、理解対象のプログラムのサイズが比較的小さいために、プログラミング・プラン[2]のような粒度の小さい知識に基づいた推論が行なわれる。しかし、リバースエンジニアリングの対象となるような大きなプログラムでは、プランの組合せが膨大な数になるために、プランに基づく推論は適用が困難である。このような場合には、プランよりも粒度の大きい知識であるアルゴリズムに基づいた推論が有効となる。

ただし、プログラミング教育のためのプログラム理解と異なり、リバースエンジニアリングのためのプログラム理解では、使用されているアルゴリズムをあらかじめ特定することはできない。したがって、使用アルゴリズムを限定するために業務に関する知識が必要となる。この知識には、アルゴリズム・パターンの他に、使用されるデータ名称やそれらの制約条件なども含まれる。

また、この業務知識によるアルゴリズムの限定が不十分であり、1つに特定できない場合には、プログラムの部分的な情報から全体のアルゴリズムを特定しなければならない。プログラミング教育のためのプログラム理解のように、プログラム中で使用されるアルゴリズムがあらかじめ1つに特定できているならば、そのアルゴリズムに基づいてトップダウン的な推論を行なうだけでよいが、1つに特定できない場合にはボトムアップ的な推論も組み合わせて行なう必要がある。すなわち、ボトムアップ推論により全体のアルゴリズムの仮説を生成し、トップダウン推論によりその仮説を検証するという推論過程が繰り返されることになる。

3　プログラム理解システムプロトタイプ

2で述べたプログラム理解手法の有効性を検証するために、現在本研究プロジェクトでは、Cプログラムを対象としたプログラム理解システムプロトタイプを実装中である。本プロトタイプは主に、正規化処理部、推論処理部と知識ベースから構成される。

3.1　正規化処理部

推論処理は、主にプログラミング知識とのパターンマッチングによって行なうが、リバースエンジニア

A Program Understanding System which Derives Design Specification from C Program

†JIMBO Itaru (e-mail jimbo@stc.ipa.go.jp)
Software Technology Center,
Information-technology Promotion Agency, Japan
3-1-38 Shiba-kouen Minato-ku TOKYO 105 ,JAPAN

[1](株)三菱総合研究所より出向中

[2](株)日本コンピュータ研究所より出向中

リングを対象とした場合、そのプログラムの多くは複数の人間によって度重なる修正や機能拡張が行なわれてきたはずである。そのため、多種多様なコーディング・スタイルが使用されている可能性が高い。しかし、想定されるすべてのパターンを事前に用意しておくことは不可能である。そこでパターンマッチングによる推論処理の前に、ソースコードをできる限り統一した形式に変換しておくこと(正規化)が有効である。

ここでの正規化は、プログラムの意味やアルゴリズムを変更することのないように、構文的な処理に限定する。具体的には、カンマ演算子の処理、forとwhileの統一などを行ない、LISP処理のための中間言語リスト(以下、単に中間言語リストとする)に置き換える(図1)。

```
for (i = 0; i < 10; ++i) {
    x += i;
}
        ↓
(ASN |i| (CONST 0))
(?LOOP (< |i| (CONST 10))
    (NIL ((ASN |x| (+ |x| |i|))
          ($++ |i|))))
```

図 1: 正規化処理の例

3.2 推論処理部と知識ベース

推論処理部は、正規化処理部で生成された中間言語リストを入力として、プログラミング知識・業務知識・変数知識とのパターンマッチングを行なう。これらの各知識は、汎用フレーム型知識工学環境ZERO[3]を利用して、階層型フレーム構造を持つ知識ベースとして構築されている。

まず最初に、プログラム中で使われている変数名や特徴的な操作・処理に着目し、業務知識を利用することにより、そのプログラムのアルゴリズムの限定を行なう。続いてプログラミング知識を利用して、ボトムアップ推論とトップダウン推論を繰り返し、アルゴリズムを特定する。ボトムアップ推論では、各フレームの part-of 関係を参照することにより、上位フレームの仮説を生成する。トップダウン推論では、has-part 関係とパターンを参照し、パターンマッチングを行なうことでその仮説を検証する(図2)。最後に変数知識を利用して、使用されている変数の同定を行なう。

推論結果は、知識ベースから生成されたインスタンスフレームの階層構造となるため、これらのフレームの説明を統合することにより、仕様情報を導出する。

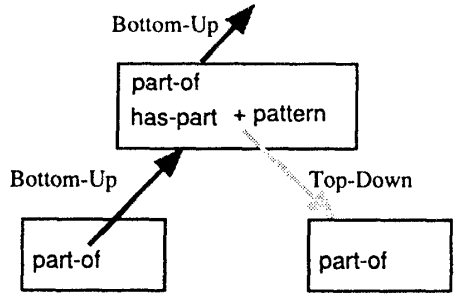

図 2: ボトムアップ推論とトップダウン推論

4 おわりに

本稿では、アルゴリズムに基づいたプログラム理解手法のリバースエンジニアリングへの応用と、そのシステムプロトタイプについて述べた。

ここで述べたような、知識とのパターンマッチングによる推論では、知識のパターン表現が重要な意味を持つ。すなわち、パターン表現を自由度の高いものにすれば、マッチングの成功確率は向上し、持つべき知識の数も少なく抑えられるが、理解の精度は曖昧なものになる。逆にパターン表現の自由度を低くすれば厳密な理解を行なえるが、少しでもパターン表現の異なる知識を別個に持たせなければならない。

今後は、このトレードオフを考慮するとともに、推論効率についても検討しながら本プロトタイプを完成させ、最終的な評価を行なう予定である。

謝辞

本研究は、東京電機大学理工学部経営工学科上野研究室で開発された、プログラム理解システムALPUSを基盤として、実用的なプログラム理解システムの構築を目指して進められている。本研究プロジェクトを進めるにあたり、貴重な御意見・御指導を頂いた、東京電機大学理工学部の上野教授、コンサルティング委員／ワーキング委員の方々、IPAの古宮特別研究員、並びに関係者の皆様に深く感謝致します。

参考文献

[1] 上野晴樹. 知的プログラミング環境−プログラム理解を中心に−. 情報処理, Vol. 28, No. 10, pp. 1280-1296, Oct 1987.

[2] W.L. Johnson. "Understanding and Debugging Novice Programs". *Artificial Intelligence*, Vol. 42, pp. 51-97, 1990.

[3] 今井健志, 伊藤治之, 吉村貞徳, 上野晴樹. "汎用フレーム・システムZERO−その概要とユーザ・インタフェースについて". 電子情報通信学会 人工知能と知識処理研究会, No. AI-87-22, pp. 35-42, 1987.