

動作履歴を用いた焦点絞り込みによるプログラム理解

6 J-6

渡辺 竜明

山本 晋一郎

阿草 清滋

名古屋大学大学院工学研究科

愛知県立大学情報科学部

名古屋大学情報メディア教育センター

1 はじめに

プログラムのコードを大雑把に知りたい時、モジュール・関数の構成・機能を解析した上でコードの理解に取り組むが、このとき例外処理など本質的な処理部分以外のコードも読んでいかなくてはならない。またモジュール・関数の動作を理解し、必要な関数に絞ってからコードを読む時、そのコードに影響を与える変数の意味を理解した上で読んでいかなくてはならないが、変数は関数内のみで定義されているわけではないので、実際に関数に絞って理解するのは難しい。そこで本稿ではプログラムの動作履歴を用いることにより、ユーザが目的とする動作に関するコードに絞って理解する手法を提案する。

2 プログラム理解

図1のように現在ソフトウェアライフサイクルにおいて多くを保守が占めている。また、システムの機能や構造についてプログラムコードの他には信頼性のある情報源が存在しないことが少なくなく、そのため保守作業においてプログラム理解に対するコストは保守作業の50%を占める[1]。

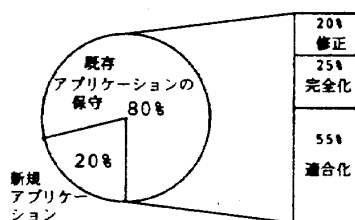


図1: 保守作業の内訳

このことからプログラム理解がいかに重要であるかが伺われる。

2.1 人によるプログラム理解

プログラム理解方法は人により様々であるが、一般に自分が理解したい機能が書かれた箇所を見つけてから、その部分のコードを読んでいく。しかし、この方法では

コードを読む際に、変数の値が何を意味しているのかわからないうえ、どの箇所を読めば理解が早まるかわからず、結局機能の内部を理解するのに時間がかかる。この原因は実際にユーザが理解したいものが何であるかの情報が事前にわかっていないことや変数の定義・参照が必ずしも対象範囲(関数)内にとどまらないためである。本稿では解決策として動作履歴を用いた手法を提案する。履歴を用いることにより余分なコードを読む必要がなくなり、さらに逐次的にプログラムを読んでいくため様々な変数を動作に関連づけて認識していくことができる。欠点としては、動的な情報のためデータ量が莫大になることである。静的情報を用いることによって、この欠点を解決したプログラムフロー理解を提案する。

3 プログラムフロー理解

プログラムフロー理解とは、プログラムの振舞いを重要なデータに着目しながら、シーケンシャルに理解していく方法である。プログラムの静的情報と動的情報を組み合わせることによって実現する(図2)。プログラム理解フローは今、自分が理解しようとしているところが、どのような経緯で、最終的な出力にどのように貢献するのかをふまえたうえで読むことができるので理解が容易である。

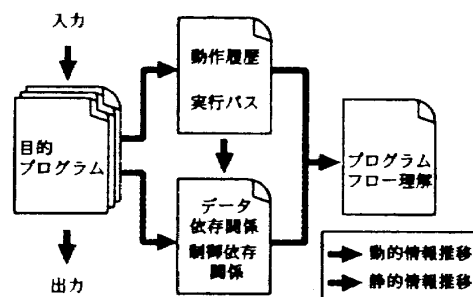


図2: プログラムフロー理解生成プロセス

期待した出力をするためにユーザが入力をするということは、単なる動的情報を得るというだけでなく、ユーザが理解したい動作を意思表示したことを意味する。すなわちプログラムを解釈していくのは最大でもこの入力を通った部分だけであり、特に注目すべきデータは出力に最も関係のあるデータであるといえる。

⁰Tatsuaki Watanabe† Shinichiro Yamamoto†
Kiyoshi Agusa†

⁰†Nagoya University †Aichi Prefectural University

また、動作履歴を用いると不要な情報は削減されるが、同じ動作を100回繰り返すなど、理解にとっては冗長なデータが増大してしまう。そこでデータ依存関係により注目すべきデータに焦点を定め、そのデータを扱うステートメントと他のコードとを区別する。さらにステートメントを必要な部分だけに絞り込むために、区別されたステートメントのうち、理解に不必要な箇所を抽象化する。

4 データ依存関係

まずプログラム理解の焦点となるデータを定める。そのため出力に関係の深いデータが最後に参照された箇所から静的バックスライシングを行なう。これによりデータ依存関係・制御依存関係を取得する。注目データとの関連から他のデータを以下のように分類する。便宜上、あるデータを定義しているデータを定義データとする。定義データは定数を代入されるまで再帰的に求める。

注目データ: 出力に関係の深いデータ及びこの定義データ

条件データ: 注目データの制御依存関係にあるデータ及びその定義データ

注目データを大きく算術型とポインタ型に分ける。集合型である配列型は領域の変化がないので算術型として扱い、構造体型は注目しているメンバによって算術型かポインタ型かを判断する。

注目データの型によって理解を意識する点異なる。算術型であれば値が変更されることに大きな意味を持つが、ポインタ型であれば、その指す領域のサイズと内容が重要になってくる。

5 抽象化

本稿での抽象化とは複数のステートメントを1まとめにして表現することである。抽象化はできる限り関数を軸に行なう。なぜなら関数はある機能としてまとめられたものであり表現することが容易だからである。抽象化するために、まず関数を注目データに従って分類する。

- 注目データを変更する関数
- 条件データを生成している関数

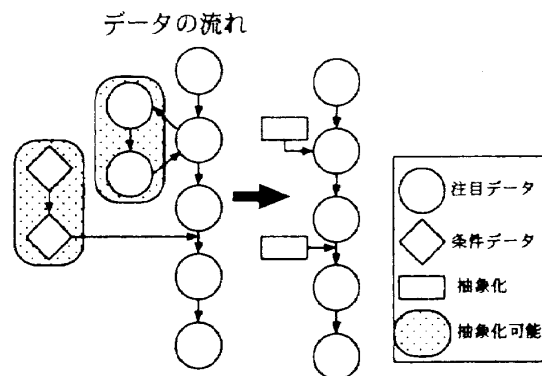


図3: データの抽象化

条件データを生成しているステートメントについては読む必要はない。どのような状態の時にどのような値を返してくるかがわかれば十分であるからである。そのため、条件データに関するステートメントは抽象化する(図3)。

また注目データにおいても、同一操作をする箇所については抽象化した方がプログラム理解フローを読みやすい。

6 おわりに

本稿ではプログラム理解のためにプログラム理解フローを提案した。本手法は動作履歴を用いることによりユーザが本当に理解したいところを把握し、注目データの選定・不必要情報の抽象化によりユーザに理解しやすい形式にして提供する。

動作履歴を使用するため、データ量の削減が大きな焦点となる。本稿では述べなかったが、まだ様々なところで削減することができる。例えば定義が2回続いた場合、1回目の定義には意味がないので省略できる。これは動的に見ていく上で価値のあることである。また、関数単位で抽象化するのではなく、その関数を含むブロックが関数だけに影響を与えているのであれば、ブロックごとまとめることができる。例えばfor文である関数を何度も呼んでいる時、for文を含めた関数を1つのグループとして表現した方が見易い。

参考文献

- [1] プログラム理解システムとその応用に関する調査研究報告書 1996. 情報処理振興事業協会技術センター