2M-1

動特性とプログラミング意図とのマッチングによるバグ発見支援

菅沼 毅 、 浦 昭二
慶応義塾大学

1。はじめに

通常、プログラマはロジックエラーを次のようにしてデバッグする。 まず、プログラムを実行し制御の流れ・変数値の変化等の実行時情報を収集する。 次にこれを仕様やプログラムとの比較により解析してバグ原因の仮説を立てる。 そして再び情報収集・解析を行い、仮説の正しさ／誤りを調べる。 この手続きによるバグの発見には豊富な経験や技術あるいは勘を要し、多くの試行錯誤を伴うため、初心者には困難である。
そこで、「’エラー’とは、利用者の意図どうりにシステムが機能しない状態. ’バグ’とは、エラーの発生し得る要因のうちソフトウェアに内在する原因.」という定義に立ち返って、エラーが発生した際のバグの発見・真の原因探索の支援方法について考える。

従来の代表的なアプローチに対する批判点のうち本研究の着目点として次の２点を想定している。

ａ）プログラムを静的なものとして捉えていること
ｂ）プログラミングの意図を考慮していないこと

ａ）に関しては、インスツルメンテーションを用いた動特性データの保存により、対象をソースコードだけではなく、プログラムの動作・振舞いとする。
ｂ）に関しては、プログラムに込められた意図を、宣言的な表現を用いて記述することにより、処理の対象に加える。
プログラムへの意図と実際の動作との間に存在する矛盾はプログラムが期待通りに動作していない点であり、すなわちバグである。 よってこの矛盾点を発見・報告することが出来ればデバッグ時の有力な情報に成ると考えられる。

2。基本概念

問題解決において、通常用いられているｐｌａｎ、ｇｏａｌという用語を、次のように定義する。

ｇｏａｌ ：プログラム中において実現されているべき目標を扱い易い抽象レベルで小さく分割したもの。
ｐｌａｎ ：ｇｏａｌを実現するための方策、処理パターン、問題解決計画。

よって、１つのプログラム中には多数のｇｏａｌが存在し、その１つ１つに対して複数個のｐｌａｎが対応し得る。

ｐｌａｎとしてコード・テンプレートを保持しておき、ソースコードとｐｌａｎ群とのマッチングを行う方法に対して、本研究では、動的なｐｌａｎ（プログラムがこのような動作をしていればｇｏａｌが達成されると言えるような、”動きのパターン”）を考えている。 従ってロジック以外の部分を局在化する事が出来るので言語依存性が小さくなっている。 ｇｏａｌは階層的な構造をしており、あるｇｏａｌが別のｇｏａｌの構成要素、つまりｓｕｂｇｏａｌと成ることもある。 これ以上分割不可能な目標をｂａｓｉｃ＿ｇｏａｌと呼ぶ。

ｂａｓｉｃ＿ｇｏａｌ ：多くのｐｌａｎ中で共通に用いられる”部品”であり、”最小単位の目標”を表す。言語依存性の最も大きい部分であり、これをｂａｓｉｃ＿ｇｏａｌとすることにより、ｐｌａｎの適応性・柔軟性が増す。

「ｇｏａｌの構成要素は、ｇｏａｌとｂａｓｉｃ＿ｇｏａｌであり、その種類や組み合せ方を規定したものがｐｌａｎである」と言える。 ｂａｓｉｃ＿ｇｏａｌの例を以下に示す。

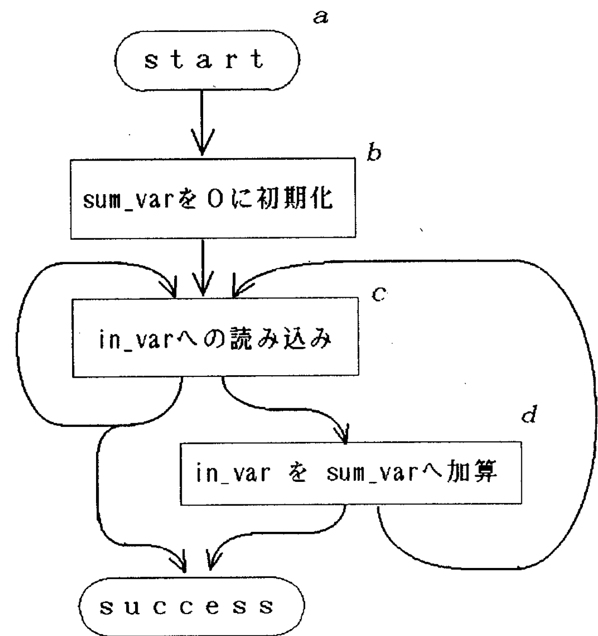
- ある変数への読み込み ： ＩＮ（ var_name ）
- ある変数の値の出力 ： ＯＵＴ（ var_name ）
- 代入 ： ＡＳＳＩＧＮ（ var_name , value ）
- 実行位置に関するアサーション：ＰＯＳ（ line_num ）

3。ｇｏａｌの定義 ＝ ｐｌａｎの記述

システムに蓄えておくｐｌａｎ、つまり初心者が利用する”デバッグ熟練者の知識”の定義の仕方を以下に例示する。

例：入力値のうち、ある条件を満たすものの合計を取る。
＝＞ goal ： read_valid_sum（in_var, sum_var, 条件）

このｇｏａｌを実現するｐｌａｎの一例 （概念的な図）



〈1〉 ｇｏａｌの分割

第一段階として、まずこのｐｌａｎを構成する（つまり, 〈2〉,〈3〉で出現する） ｓｕｂｇｏａｌを列挙する。

Matching Dynamic-Profile with Intention of Programming for finding bugs.
Takeshi SUGANUMA , Syoji URA
KEIO University

```
(a:start)
  b:ASSIGN ( sum_var , 0 )
  c:IN ( in_var )
  d:ASSIGN (sum_var,sum_var(b\/d) +in_var(c))
(z:end of execution)
```

ただし、aとzは必ず存在する。 ---> デフォルト化

〈2〉 subgoal間の制約（アサーション）

次に、subgoal間の遷移において成立しているべき制約条件等をアサーションとして記述する。

assert（c, c:￢（条件））
assert（c, d:条件）
assert（c, z:￢（条件））

〈3〉 遷移表

	b	c	d	z
a	○	E	E	E
b	W	○	E	E
c	E	A	A	A
d	E	○	E	S

ただし
○：正しい遷移、次の検査へ
A：アサーションの成立検査
W：Warning
E：Error
S：Success

全てのアサーションを満たしてzまで到達した場合はSu-ccess、つまりplanが達成された（＝goalが達成された）と判断出来る。

4。goal達成のチェックとバグの発見

動特性の把握は、元のプログラムの動作に影響を与えない様な挿入ルールに従って、データ収集ルーチンの呼び出しを挿入する事により行う。 いかなるgoalへも対応可能なように実行の軌跡・変数値の変化等の情報はすべて収集して、後のマッチングで扱い易いように各種のポインタを持たせたリストの形態として保存しておく。 次に、この動特性とgoalとして宣言的に記述されたプログラミングの意図との整合性のチェックを行うが、その本質的なアルゴリズムは以下の通りである。

[α] 実行軌跡リストより次の実行部分（文）を取り出す
[β] 〈1〉のsubgoalのいずれかが成立しているか否か？
　　不成立 ---> [α]へ
　　成立 -----> [γ]へ
[γ] 成立しているsubgoalに関して〈3〉を調べる
　　○ ---------------> [δ]へ
　　A --- 〈2〉Assertion 成立 --> [δ]
　　　--- 〈2〉Assertion 不成立 -> [ε]
　　W ---------------> [ε]
　　E ---------------> [ε]
　　S -----------> 次のgoalへ
[δ] マッチング履歴を記録してから[α]へ
[ε] エラーやワーニングのメッセージを出して、次の動作の指示を受ける。
　　次の動作
　　・そのまま[α]へ
　　・別のplanのチェックへ
　　・次のgoalのチェックへ
　　・変数値、実行軌跡に関する問い合わせ
　　・終了

goal群から1つずつgoalを取り出して、そのgoalを達成し得るいくつかのplanの達成・非達成を調べる。 いずれか1つのplanが達成されていれば、そのgoalの達成を報告し次のgoalのチェックへ移る。

全てのplanを調べて、いずれも達成されていなければ、整合性の大きいもの、つまりマッチングが進んだplanより「どのgoalが」、「どの条件がクリアされていないために」、「それはソース上のどの部分に該当するのか」等の報告を行う。 そこにバグが存在すればすぐにその修正作業に移れるが、そううまく行かない場合でも、プログラマは示された情報を手がかりとしてその目標（＝goal）をコード化したと言う心当たりのある部分の確認を行えばよいので欠陥探索の対象範囲がかなり絞られることが期待できる。

goalが実現していない原因、つまりバグの真の原因がそれよりもずっと以前やplanの範囲外に存在する場合も考えられるが、動特性を保存しているためその場合には逆方向への実行・状況の再現によりbackwardなバグの探索が強力な武器となるはずである。

5。おわりに

プログラムを静的なものとして扱い、ソースコードとテンプレートとのマッチングを行う方法に比較して以下の点が期待できる。

・plan数の減少
　1つの動作パターンに対して、それを実現するソースコードの記述方法は多数存在するため。
・マッチング精度の向上
　同様に個々の操作に対してもその実現方法は多数存在するため。

本研究は、初心者のために、エキスパートの持つプログラミング技術、デバッグのためのアサーション活用の知識の利用を考えたものである。 より実践的には、学校でのプログラミング教育において、指導者が出題に応じてgoalの定義を行い、学生がそれを利用するという状況が考えられる。

今後はインプリメントと評価が課題であるが、とりあえず限られた範囲の問題のみを扱うことの出来る実験システムを作成してバグの検出能力の定量的な評価を予定している。

6。参考文献

1)E.Soloway and W.L.Johnson,"PROUST:Knowledge-Based Program Understanding",IEEE Trans.on Software Eng. vol.SE-11 No.3 March 1985 pp.267-275
2)R.E.Seviora,"Knowledge-based Program Debugging Sy-stems",IEEE SOFTWARE May 1987 pp.20-32
3)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 1987 NO.10 pp1280-1296
4)石井威望 広瀬通孝 小池英樹、"プログラム読解支援に対する自然言語理解的アプローチ"、情報処理学会・ソフトウェア工学研究会資料 61-3,1988