6R-6

動的解析と静的解析を融合した Ｐｒｏｌｏｇプログラムのデバッグ

御宿哲也　永田守男
(慶應義塾大学)

1．はじめに

プログラミング言語 Prolog は、記述力が高い反面、Lisp などに比べて言語としての歴史が浅いため、プログラミング環境としての整備が遅れているといわれている。しかし、E.Y. Shapiro が、ユーザとの対話によってバグを同定するアルゴリズム [1] を考案してから、Prolog 言語の特徴に着目したデバッガの研究が進んでいる。この手法はバグ箇所の同定という点では汎用的であるが、実際のプログラミングを考慮にいれていないので、同定されたバグに対する修正案を提示したり、修正のヒントを示すことに対しては弱い。

一方、対象とする言語は異なるが、Soloway らは、プログラミングに関する知識に主体に置いたデバッグ方式を研究している [3]。この手法はプログラミングに関する知識とはいえアルゴリズムに限られているため、アルゴリズムが分かっている場合にしか適用できないという点で汎用性に欠ける。

そこで、本研究では、前者の手法に知識としてプログラマーが用いていると思われるスキーマを組み込むことによりプログラムの静的な解析を強化し、バグの説明とその修正のヒントを与えることが可能なデバッギング・システムを考案する。

2．従来の研究

基本的に、デバッグのプロセスは二段階に分けることができる。第一段階は、プログラム中のバグの箇所を同定することであり、第二段階は、そのバグを訂正することである。

また、Prolog のバグの種類は、３つあり、(1) 間違った答を返す (2) 予期せぬ失敗 (3) 無限ループ に分類できる。

アルゴリズミック・デバッギングの基本的な考え方は、計算木に注目することである。計算結果の真偽や計算順序の制約はプログラマーに入力してもらい、計算木をトップダウンにトレースして一つ一つのステップが思いどうりに行われているかをチェックしてバグ箇所を同定する。一方、バグの訂正は、バグのステップを削除し、帰納推論アルゴリズムを用いて正しいと思われるステップを生成し、それを追加することで行われる。

Soloway らの知識をベースとしたデバッギング方式は、正解のプログラムやそのアルゴリズムを抽象化したプランを知識としてシステムがもち、デバッグされるプログラムの多様性を考慮にいれながらバグを同定する。バグは正解のプログラムと等価であることが証明不能であったり、バグ・ルールによって書換え可能であることによって同定される。一方、バグの訂正は、正解プログラムやプランをバグ箇所に埋め込むことで達成される。

3．本研究のアプローチ

本研究の基本方針は、アルゴリズミック・デバッギングのようなプログラミング言語の意味に基礎を置いた動的な手法とプログラミングに関する知識を利用してデバッグを行うような静的な手法の融合を行うことである。

前者の手法は言語の意味に中心があるためバグ箇所の同定という点では汎用的であるが、修正に関しては複雑なプログラムには対応できない。

一方、後者の手法でアルゴリズムよりもさらに抽象度をあげた定型的なパターンを知識として利用すれば、前者のアプローチではできなかった誤りの説明や修正のヒントの提示は可能になる。

本研究では、対象とするプログラムとしてリスト処理などを行う再帰プログラムを考えている。リストなどのデータ構造は再帰的に定義されているので、それらを処理するプログラムの型は限られる。これらの型がプログラミングの際の定型的なパターンとなり、これに当てはまらないものはバグの可能性が高いと考えられる。これらの分類は第４節で行う。

また、バグの原因として (1) 条件の間違い (強すぎる・弱すぎる)、(2) サブゴールの呼び出し方の間違い、(3) 間違ったサブゴールの呼び出しが考えられる。第５節では、(1) の条件の間違いに焦点を絞って議論を進めていく。

4．再帰プログラムの型

Prolog は論理型言語なので入出力に対するはっきりした区別はないが、プログラムを書くときにはプログラマーは入出力を意識するので、ここでも入出力モードを考慮してプログラムを分析する。すると、値を返す手続き型と真偽を決定する述語型にプログラムは大きく分類することができる。さらに終了条件の違い (Map/While・And/Or) と分岐の有無 (Total/Selective) によって表１のように細かく分類される。

Debugging of Prolog Programs
Based on Dynamic and Static Analysis
Tetsuya MISHUKU and Morio NAGATA
Keio University

Procedure		Predicate	
Map	Total	And	Total
	Selective		Selective
While	Total	Or	Total
	Selective		Selective

表1　再帰プログラムの条件を考慮した分類

ここで、Map(And) と While(Or) の違いは、前者がデータ構造をすべて探索するのに対して後者はある条件が満たされるまで探索するという点である。また、Total と Selective の違いは、前者が再帰呼び出しに制約がないのに対して後者は制約付であるという点である。

つぎに本研究で対象とするプログラムが処理するデータ構造に関する考察を行う。ここでは、データ構造として次のような3つの再帰的に定義された構造を考える。

(1) 線形構造 (ex. 線形リスト、自然数)
(2) 分岐が一定な木構造 (ex. ﾌﾙｰﾘｽﾄ、二分木)
(3) 分岐が一定でない木構造 (ex. ﾌﾙｰﾘｽﾄ、項)

すると、プログラムの定型的なパターンとして、条件を考慮した分類からは 8個のパターン、データ構造からは 3 つあり、組合せとしては 24 個のパターンが存在することになる。

5．条件の誤りとその検出方法

条件の誤りの検出は二段階に分けて行われる。第一段階は、節に含まれる条件の無矛盾性 (consistency) と冗長性 (redundancy) のチェックである。もし条件が矛盾していれば、その節は常に成功しないのでバグの一因となる。また、条件が冗長な場合は、バグになる場合、ならない場合がある。たとえば次の節を考えてみる。

```
upper_letter(C) :- C>64,C>91.          % C>64,C<91
```

この節は本当は % にあるようにしたかったのに不等号の向きが逆である。これでは C > 91 だけしかきいていない。このように条件が冗長な場合でも、バグの可能性があると考えられるのである。

第二段階は、手続きの定義に含まれる節の条件の冗長性、網羅性 (exhaustiveness) [3] である。冗長性に関しては、次のような手続きを考えてみよう。

```
p(X,Y,Z) :- intersect(X,Y),....
p(X,Y,Z) :- subset(X,Y),....
```

ここで、intersect(X,Y) は集合 X と Y に積集合が存在するかを調べ、subset(X,Y) は X が Y の部分集合であるかを調べるとする。部分集合であれば、当然積集合は存在するので、二番目の節は選択されない。部分集合ではないが交わりだけをもつ場合を考えたいのであれば、このままでは条件が弱すぎるので、部分集合ではないという条件を付け加えなければならない。

網羅性に関しては次のような手続きを考えてみる。

```
q(X,Y,Z) :- X>Y,...
q(X,Y,Z) :- X<Y,...
```

この場合、X と Y が等しい場合の節が存在しないので、たとえば、q(3,3,W) は計算できずに失敗してしまう。もし、必要なければこのままでよいが成功させたいときは X=Y の場合の節を追加しなければならない。

ここで、条件 C1,C2,...,Cn の冗長性、無矛盾性、網羅性についてまとめる。

冗長性：　Ci ⊆ Cj (i ≠ j)
矛盾性：　Ci ⊆ ¬Cj (i ≠ j)
網羅性：　C1 ∨ C2 ∨... ∨ Cn が真

これらの三つの性質を定理証明の技術を用いてチェックすることで、条件が満たされないところはバグの可能性があると考えられる。

このように条件を静的に解析することによって、疑わしい箇所を特定することができるが、それは可能性にすぎない。デバッグ中にユーザのモデルをえることによってはじめてバグであることが確かめられる。そのとき静的解析情報がバグの修正のヒントとなり得る。これが本研究の目的である、静的な手法と動的な手法の融合である。動的な手法だけでもバグの箇所は特定できるが、静的な手法があってはじめてそのバグの説明が可能となり、修正の役に立つのである。

6．おわりに

現在、条件の誤りに関するバグを検出し修正するシステムを試作中であり、今後は、さきに挙げた残りの二つの原因を処理する方法を考案していくつもりである。二番目の「サブゴールの呼び出し方の間違い」は正しいトレースから適切な一般化を行えば修正が可能なのではないと考えている。三番目の「間違ったサブゴールの呼び出し」は部品合成 [4] や仕様 [5] を用いる必要があるのではないかと思われる。

参考文献

1) Shapiro,E.Y. : Algorithmic Program Debugging, MIT Press (1983)
2) Johnson,W.L. and Soloway,E. : Intention-Based Diagnosis of Programming Errors, Proc of AAAI-84, pp. 162-168 (1984)
3) Nagata, Morio : An Approach to Construction of Functional Programs, Journal of Information Processing, Vol. 5 No. 4, pp. 231-238 (1982)
4) 小泉、永田 : トレースと設計戦略を用いた Prolog プログラムのトップダウンな合成方法, 第 39 回情報処理学会全国大会 (予定) (1989)
5) 深谷、永田 : Prolog プログラムのデバッグのための検証と説明, 第 39 回情報処理学会全国大会 (予定) (1989)