6R-7

Ｐｒｏｌｏｇプログラムのデバッグのための検証と説明

深谷哲司　永田守男
（慶應義塾大学理工学部）

１．はじめに

現在のプログラム開発環境は、様々な形態で発展しつつある。本稿では、開発環境のうちデバッグに着目した研究について述べる。そのなかでもプログラムに関する知識を利用したデバッグについて考えている。そうした研究は、これまでにも多く行われている。有名なシステムとして、トップダウン的な手法としてＰＲＯＵＳＴ[1]、ボトムアップ的な手法としてＰＵＤＳＹ[2]がある。これらは、いくつかのプランもしくはアルゴリズムリズムに対するテンプレートのような物を用意し、この枠組みに入り得るものについては対応が可能であった。この枠組みをできる限り外すことが本研究の焦点である。

そのために、本研究では、与えられた入出力仕様（制約）がプログラム上で破棄されていないかを検証し、さらにその説明を行うことによってデバッグに役立たせる方法を提案する。

過去の行われたプログラムの検証の興味は、そのプログラムが正しいか、誤っているかという点にだけあったが、入出力に関する制約がなぜ満たされていないかという説明機能が加えられたならば、十分にデバッグ支援になり得るのではないかと考えたからである。

２．方針

本研究では、入出力仕様の検証に注目している。また、Prolog言語でデータ構造がリストである場合を対象としている。

検証および説明のために、言語(Prolog)、公理系（数式、集合）、リストに関する知識を独立のモジュールとして持ち、それらを統合し、検証（推論）を進めるルールをこれらと別のモジュールに持たせた。

全体の流れは以下のようになっている。

１）入出力仕様

入力と出力の間に制約を与える。基本的な制約（ex:入力と出力の長さが等しい）と各要素の意味を持たせるような限定表現の組合せから成り立つ。現状では、合わせて２０程度用意している。

２）検証対象の設定

入出力のうちから、どの制約に着目し検証を進めるかを決定する。他の制約に付いては、基本的には破棄されていないことを仮定する。すなわち積極的には誤りを同定しない。

３）利用可能知識の統合

対象となる制約の検証ごとの戦略に合わせて知識の統合をおこなう。

利用する知識とは、大きく分類すると次の３つである。各知識は、モジュールの単位でおさめ、各知識を統合するルールはモジュール間参照関係により表現する。

①言語に関する知識
- 再帰のパターン知識
  - 単純再帰
  - 条件付き単純再帰
  - 相互再帰
  - バイナリ再帰

  終了条件（リストの型）、数式に関する知識より再帰回数等の決定が可能。

⋮

②リストに関する知識
- 長さに関する知識

  数値計算に関する知識にリストの結合等の知識ををマッピングすることにより長さの計算についての知識が得られる。
- 基本リスト処理知識

  car部、cdr部、結合（削除）

⋮

③数式、集合等に関する公理系
- 変数から変数への制約伝幡のための知識
- 再帰の条件式をまとめる集合に関する知識[3]

⋮

４）検証

検証方法の概要は以下のようになる。

検証の対象となる制約についての戦略ルールを用いる。

各変数（要素、リスト）に制約を付加する。その変

Verification and Explanation for Debugging Prolog Programs

Tetsuji FUKAYA and Morio NAGATA

Keio University

数の持つ制約をサブプログラムまで伝幡させる。サブプログラムから得られる変数からも制約の伝幡を受ける。プログラムを最後まで検証し終えたときに、制約が破棄されている場合、それまでの検証過程と、守られるべき制約と、守られている制約を提示する。これらの差からプログラムの訂正法を推論する。

3．例題

例をもとに進める。

入出力仕様（入出力制約）　入力と出力のリストの長さが等しい。

```
quicksort([X|Xs],Ys) :-
        partition(X,Xs,Littles,Bigs),
        quicksort(Littles,Ls),
        quicksort(Bigs,Bs),
        append(Ls,Bs,Ys).
quicksort([],[]).

partition(X,[Y|Ys],[X|Ls],Bs) :-
        X > Y,
        partition(X,Ys,Ls,Bs).
partition(X,[Y|Ys],Ls,[X|Bs]) :-
        X =< Y,
        partition(X,Ys,Ls,Bs).
partition(_,[],[],[]).
```

これはクイックソートの例である。これは、入力と出力のリストの長さが等しいという制約が損なわれている。これの検証のしかたと、説明するために必要となる知識を次に示す。

トッププログラム上での制約の意味

Ys = [X|Xs]　－　①

サブプログラムからの制約の伝幡

（appendからの伝幡）

Ys = Ls + Bs　－　②

再帰呼び出し中での制約

Ls = Littles

Bs = Bigs　　－　③

①＋②　[X|Xs] = Ls + Bs　－　④

③＋④　[X|Xs] = Littles + Bigs　－　⑤

ここまでで[X|Xs] = Littles + Bigs を満たせばよいことが得られる。以後サブプログラムからの検証。

partitionは、P←Q，Pの条件付き単純再帰の型に入り、tail部(cdr部)を　再帰呼び出しするtail recursion である。　－　ⓐ

partitionの２つのプログラムの差分をとる。

[Y|Ls],Bs ← 条件式　X > Y

Ls,[Y|Bs] ← 条件式　X =< Y

数式と集合の常識より、２つの条件の関係は、相反し、２つの条件で１となり、すべての条件が網羅されている。　－　ⓑ

終了条件より、空リストに条件により[Y|Ys]の要素が振り分けられる。　－　ⓒ

ⓐ＋ⓑ＋ⓒ　[Y|Ys]の長さの回数再帰が繰り返される。

Littles + Bigs = Xs　－　ⓓ

⑤とⓓより、矛盾が生じる。制約が破られ、この説明から要素Xが不足していることを推論できる。

この１つの例からもわかるように、一般的な知識と呼ばれる物をいかに構造的に、因果関係を考慮し、推論に適した形で蓄えられるかが、非常に問題となるところである。

Prolog言語を扱うことにより、リストの再帰処理を扱うことが多くなるという特色から、リストと再帰の関係、さらに、条件式の矛盾、十分性を得るために数式、集合との常識も推論に適した形でまとめあげたものが必要となる。

現在、入出力仕様としては、２０近くの制約の組合せとなっている。各制約を検証するためのアプローチの知識、即ち、先に述べたどのようなな常識を合わせて推論を進めるかを研究中である。

4．今後の課題

現在の入出力仕様では、すべての意図が記述できるとは限らない。記述性を向上させることが問題である。

サブプログラムとの制約の伝幡の方法がまだ明確でないため、部品合成をおこうための手法を利用することも考えている。[4]

対象とするプログラムが正しいと仮定し、制約と矛盾を導き出すために複数の誤りが同時に発生した場合の仮説の設定が困難である。

参考文献

[1]W.L.Johnson and E.Soloway:
"PROUST:Knowledge-Based Program Understanding",
IEEE Trans. on SOFTWARE ENGINEERING,
vol.SE-11, no.3, March 1985, pp.267-275

[2]F.J.Lukey:
"Understanding and debugging programs",
Int. J. Man-Machine Studies 12, pp.189-202, 1989

[3]御宿、永田：”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” 第３９回情報処理学会全国大会（予定） 1989

[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