属性文法に対するデバッグ方式の構想

7G－7

大久保 琢也　佐々木 晃　脇田 建　佐々 政孝
東京工業大学 理学部 情報科学科

1　はじめに

一般に，属性文法で書かれたソフトウェアのデバッグには独特の難しさがある．これまでの研究では，属性文法による開発環境という点がなおざりにされる傾向があった．本研究では，アルゴリズミック・デバッギング[1]の手法が属性文法に適用できることを示し，これを利用した属性文法記述デバッガの構想を述べる．

2　属性文法記述のデバッグ

属性文法は，簡潔な記述で強力な表現が可能な定式化である．コンパイラ生成の分野などではその有用性が高く評価され，さまざまな応用が研究されている．

しかし，今までの研究では，属性文法の開発環境という点がおろそかにされてきた．属性文法で書かれたソフトウェアは，記述を読んだだけでは評価時の動きを把握しにくいため，そのデバッグは通常のプログラミング言語とは異なる難しさを含んでいる．そこで，本研究では属性文法と同じく宣言的な意味を持つ，Prologについて提案されたアルゴリズミック・デバッギングの手法を，属性文法のデバッグに応用することについて考察する．

3　アルゴリズミック・デバッギングの応用

3.1　アルゴリズミック・デバッギング

アルゴリズミック・デバッギングとは，プログラム実行時に起こる関数の入出力に注目して，バグを含む関数を探すというデバッグ方式である．この方式では，関数の中で他の関数が呼び出される部分が重要な点となる．しかし，属性文法の記述法は一般の言語とはパラダイムが異なり，ユーザが記述する意味関数はその中で他の意味関数を呼び出すことがない．そのため，何らかの工夫が必要になる．

3.2　Syn関数とデバッグアルゴリズム

属性文法では，属性評価を効率良く行なうために，Syn関数というものを考えることがある．Syn関数とは任意の合成属性に対して定義される関数で，入力に，(1) その属性と同じノードに属する継承属性 と，(2) そのノードを根とする部分解析木 をとり，その合成属性の値を返す．Syn関数は，通常の言語の関数と同じように，他のSyn関数を呼び出しながら値を求めるので，アルゴリズミック・デバッギングの手法が適用できる．

A study of debugging method for attribute grammar description
T. Ookubo, A. Sasaki, K. Wakita, and M. Sassa
Tokyo Institute of Technology

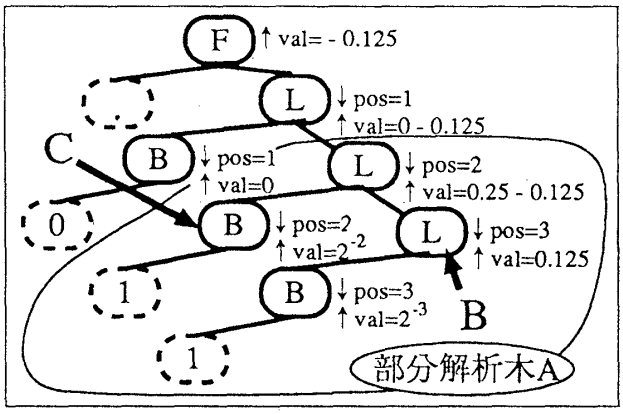

図1 属性つき解析木

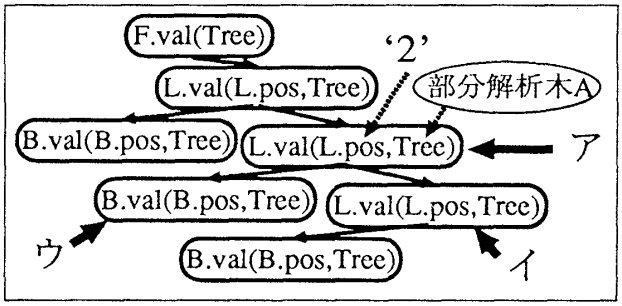

図2 Syn関数の計算木

例として，2進数の小数の値を計算する次の属性文法を考える．ただし，これは一ヶ所誤りを含んでいる．

```
F ::= . L
        { L.pos = 1; F.val = L.val }
L ::= B L
        { L[1].pos = L[0].pos+1;
          B.pos = L[0].pos;
          L[0].val = B.val - L[1].val }
    | B
        { B.pos = L.pos; L.val = B.val }
B ::= 1
        { B.val = 2 ^ -B.pos }
    | 0
        { B.val = 0 }
```

これに対して，.011 という入力を与えると，図1のような属性つき解析木ができる．この時 Syn 関数がそれぞれ他の Syn 関数を呼び出した様子を図2に示す．これをこの文法の計算木と呼ぶ．

求められた属性値が誤っていたとき，デバッガはその値を求めるのに使われた計算木を調べる．この計算木中の任意の Syn 関数 $F.s(i_1, i_2, \ldots, i_n, Tree)$ を調べた時，(1) これが誤った値を返していれば，バグは $F.s$ の

定義の中か，計算木中の $F.s$ 以下の部分木のどこかに少なくとも一つある．(2)$F.s$ が正しい値を返していれば，バグはそれ以外のところに少なくとも一つある．いずれの場合も $F.s$ の点で計算木を分割すれば，バグを含む，より小さい部分木が得られる，この手法を再帰的に用いることにより，バグが存在すれば，バグを含む関数が少なくとも一つ必ず見つかる．これにより，インフォーマルではあるがこのアルゴリズムが完全なことがわかる．

例えば図1において，属性値 F.val = −0.125 は誤った値である．この時，デバッガは次のようにバグを発見する．まず，F.val を計算するのに使われた計算木を調べ，その中から適当な点を選ぶ．ここでは図2の点アに着目したとする．この点では，Syn関数 L.val により L.val(2,[部分解析木A]) = 0.125 という計算が行なわれている．この値は誤っているので，バグは L.val の定義か，計算木中の点アより下のどこかに存在することがわかる．そこで，次にアより下の部分木のどこかの点をとり，同様の検査をする．(以降，解析木中のノードB，ノードCを根とする部分木を，それぞれ部分解析木B，Cと呼ぶ．)点イを調べると L.val(3,[部分解析木B]) = 0.125 という計算が行なわれている．この計算は正しいので，バグは点イの外側にあることがわかる．同様に点ウにおいて，B.val(2,[部分解析木C]) = 0.25 という計算は正しいので，バグは点ウの外側にあることがわかる．以上より，バグは点ア，すなわち L.val の値を定めている部分にあることが突き止められた．

4 さらなる効率化のために

しかし，デバッグの効率という点を考えると，単に上のアルゴリズミック・デバッギングの手法を用いるだけでは十分とはいえない．少なくとも，次の二つの点について工夫が必要であると思われる．

4.1 問合せの回数の減少

一つめの点は，ユーザへの冗長な質問を減らす必要があるということである．この手法でバグを発見するためには，計算木が大きくなったとき，ユーザは相当な数の質問に回答する必要がある．これを軽減するため，ユーザに怪しいと思った値を指摘してもらうという方法を導入する．デバッガはその値に関係のあるところを優先的に調べることにより，冗長な問合せを減らすことができる可能性がある．問合せは，バグを含む可能性のあるところならば，計算木のどこで行なってもよいので，この方法はデバッグ法全体の完全性を損なうことはない．

4.2 問合せの複雑さの軽減

二つめの点は，ユーザに対する問合わせが難解な質問となりうることである．上の例でいえば，ユーザは図3のような問合わせに回答しなくてはならない．ユーザへの問合せの対象となるのは「Syn関数」であるが，この関数はユーザが意識して作ったものではないので，場合によっては難解な質問にならざるを得ない．

この問題を回避するため，ユーザに「わからない」という回答を許すことにする．この回答を受けとった場合，デバッガはユーザがこの問題に答えられるように援助を行なう．具体的には，他のSyn関数の調査を行ない，この問合せのヒントとなる情報を集めてから再度質問を行なったり，ベースとなる言語のデバッガを使って，ユーザにその部分の調査をしてもらうなどの工夫が考えられる．

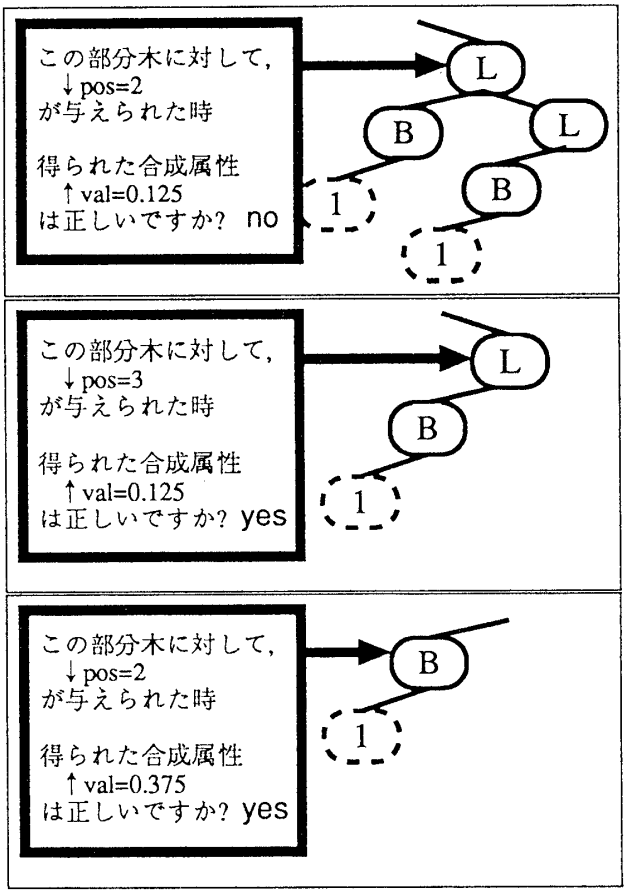

図3 ユーザに対する問合わせの例

5 おわりに

本研究では，アルゴリズミック・デバッギングの手法が属性文法に適用できることを示し，属性文法に対する有効なデバッグ法についての考察を行なった．

現在我々は，Jun という属性評価器生成系の改訂を行なっている．これは有限帰納属性文法というクラスの文法が扱える強力な生成系であるが，その上にかぶせる形で上で述べたようなデバッガの実装を進めている．

今後の課題として，より属性文法に適したデバッグ法の開発と，属性文法のプログラミングが手軽にできるような，統合環境の開発を目指している．

謝辞 議論をしていただいた今泉貴史氏に感謝する．なお，本研究の一部はEAGL事業推進機構および文部省科学研究費の補助を受けた．

参考文献

[1] E. Y. Shapiro. *Algorithmic Program Debugging.* The MIT Press, 1982.