2v-07 # アルゴリズミック・デバッギングを用いた属性文法のデバッガ*

池添 洋平　　佐々木 晃　　佐々 政孝
東京工業大学 情報理工学研究科 数理・計算科学専攻

1　はじめに

属性文法は、コンパイラの定式化をはじめとした様々な分野で用いられている。しかし、属性文法で記述したソフトウェアの開発環境、特に、デバッガに関する研究はこれまであまり行われてこなかった。

本研究では、属性文法に対する系統的なデバッグ法として、大久保が提案した属性文法に対するアルゴリズミック・デバッギング [2] および、佐々木が提案したプログラム・スライシングを用いたデバッグ法 [1] を適用したデバッガを実装することによって、これらの有用性を確認する。

さらに、ユーザが実際にデバッグを行う際に有効な情報を提供し、より円滑に作業を進めることができるようにする。

2　アルゴリズミック・デバッギング

アルゴリズミック・デバッギングは元々、論理型言語の関数性に基づいたデバッグ法として提案されたもので、手続き型言語や関数型言語にも応用されている。

アルゴリズミック・デバッギングの動きを順を追って説明する。まず、デバッガは誤った計算の実行結果に基づいて、計算木と呼ばれる木を作りデバッグを開始する。計算木のノードは、プログラム中の関数を表し、その関数の中で呼ばれている関数が子ノードになる。デバッガは計算木に含まれるある関数についての入力と計算結果をユーザに見せて、その関数の挙動が正しいかどうかをユーザに問い合わせ、

- ユーザがその関数を正しいと判断した場合、そのノード以下の部分木にはバグを含まないので、その部分木を計算木から削除する。
- ユーザがその関数を正しくないと判断した場合、そのノード以下の部分木にバグを含むので、その部分木だけを計算木として残す。

このような問い合わせを繰り返し、計算木を小さくして行くことによってバグの場所を特定することができる。

属性文法に対してアルゴリズミック・デバッギングを適用するには、属性を評価する過程で、「関数」のような振る舞いをする部分に着目し、これを計算木のノードとする。

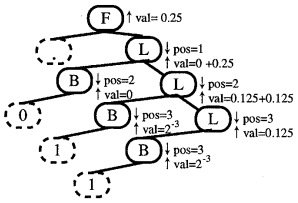

図 1: 属性付構文解析木

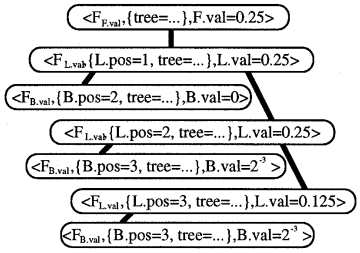

図 2: 計算木

構文解析木のノードの各合成属性の値は、その合成属性が依存する継承属性とそのノード以下の部分木によって一意に定まるので、あるノード N の合成属性 $N.s$ と $N.s$ が依存する継承属性 $N.I_{N.s}$および N を根とする部分木 $tree_N$に対して次のような関数 $F_N.s$ を定義することができる。

$$N.s = F_{N.s}(N.I_{N.s}, tree_N)$$

このような関数を Synth 関数と呼ぶ。この Synth 関数を計算木のノードとしてアルゴリズミック・デバッギングを行うことができる。図 1 および図 2 に属性文法に対する例を示す。図 1 は 2 進小数からその値を計算使用とする属性文法に".011"を入力として与えたときの属性付構文解析木で、図 2 はそれを元に作られた計算木を表している。

ユーザは、Synth 関数の振舞いが意図したものに合致しているかどうかを、Synth 関数の引数となる、継承属性の値と構文解析木の部分木、および Synth 関数の返す合成属性の値との関係をもとに判断する。

*A Debugger for Attribute Grammars using Algorithmic Debugging, Youhei Ikezoe (Yohei.Ikezoe@is.titech.ac.jp), Masataka Sassa, Akira Sasaki, Dept. of Math. Comp. Sci., Tokyo Institute of Technology

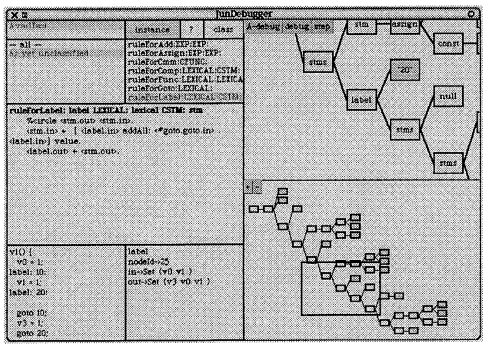

図 3: Aki の実行の様子

3 プログラム・スライシングを用いたデバッグ法

プログラム・スライシング技術とは、プログラム内のある計算に影響を与える計算の集合を求める技術である。最初は、プログラムのデバッグを支援するためのものとして考案されたが、現在ではプログラムのテスト、保守、デバッグ等、広い分野で応用されている。

ある属性に対するスライスは、その属性が、直接間接に依存する属性であり、これは属性依存グラフをもとに得ることができる。

このスライスに対して、分割検証法と呼ばれる方法でデバッグを行う。分割検証法はスライスを 2 つに分割し、分割点に対してフローデータと呼ばれる属性値の集合を定義する。フローデータには、分割点前で計算される属性の中で、分割点後の属性計算でその値が直接用いられるものが含まれる。このフローデータに含まれる属性に対して次のようなことが言える。

- フローデータのある属性の値を正しくないとユーザが判断したときその属性に対するスライスの中にバグを含む。
- フローデータに含まれるすべての属性の値が正しいとユーザが判断したとき分割点より後にバグを含む。

分割検証法では、このようなバグを含む部分に対してスライスの分割、属性の値のユーザへの問い合わせを繰り返すことによってバグの場所を特定することができる。

デバッグ時、ユーザはデバッガの問い合わせに対して構文解析木を見て、属性の値がユーザの意図していた値になっているかどうかによって値の正しさを判断する。

4 デバッガの実装

本研究では 2 章および 3 章で説明したデバッグ方法を用いたデバッガ Aki を実装し、これらのアルゴリズムが属性文法に正しく適用されていることを確かめた。また、実際に属性文法を使ってプログラムを行う人が効率的にデバッグの作業を進めることができるように、属性付構文解析木の表示・対話方式に工夫を施した。

Aki は Jun と呼ばれる実験系を基にしている。Aki を実装するにあたり、Jun の生成する属性評価器がトレースを出すように手を加えた。トレースには属性の値と依存関係の情報が含まれている。Aki は、トレースと構文解析木をもとに計算木を作り、デバッグを開始する。実行の様子は図 3 のようになる。

Aki では、

- 属性付構文解析木の拡大図
- 属性付構文解析木の全体図
- 属性の値

を見てユーザはデバッガの質問に答えることができる。これらとともに、

- 構文解析木をアンパーズして得られるソースプログラム
- 属性の評価規則

を同時に見ることができる。構文解析木のノードをマウスでクリックすると、そのノードに付随する属性の値と評価規則が表示され、そのノードに対応するソースプログラムの部分が強調される。逆に、ソースプログラムの部分をマウスでクリックすると、それに対応するノードが強調され、属性の値、評価規則が表示される。

これによってユーザが現在注目しているノード、属性の値、評価規則、ソースプログラムの間の対応を同時に見ることができ、デバッガの質問に答えやすくなっている。

5 まとめ

本研究では、属性文法に対する系統的なデバッグ法を実装した。

これにより、属性文法でアプリケーションを開発するプログラマは、生成された属性評価器や、入り組んだ属性の依存関係を気にすることなく、デバッガの問い合わせに答えることによってデバッグを行うことができる。

また、グラフィカル・ユーザ・インターフェースによる有用な情報の提供により、ユーザが属性の値が正しいかどうかを直観的に理解できるようになるので、より円滑にデバッグの作業を行うことが可能になる。

今後の課題として、2 つのデバッグ方法を組み合わせた、より効果的なデバッグ方法の研究、デバッガを含めたより実用的な属性文法の開発環境の開発などが挙げられる。

参考文献

[1] 佐々木晃, 脇田建, 佐々政孝. スライスを用いた属性文法記述のデバッグ法. 日本ソフトウェア科学会第 13 回大会論文集, pp. 249–252, 1996.

[2] 大久保琢也, 佐々政孝. 属性文法に対する系統的デバッグ方式. コンピュータソフトウェア, Vol. 13, No. 2, pp. 45–57, 1996.