4E−2　木属性文法とGUI生成系を利用したデバッガの作成

太田 行紀　田淵 聡†　脇田 建　佐々 政孝
東京工業大学 理学部 情報科学科　†(旧)筑波大学 情報学類　(現)ソニー

1　はじめに

プログラム開発環境においてはデバッガの存在が欠かせない．しかし，それを始めから作成するためには多大な時間と労力を要する．本稿は，インタプリタの内部情報を利用することにより，インタプリタ型のデバッガ作成に要する労力を劇的に減少することができることを報告する [3].

われわれは言語処理系を記述する統一的な枠組として属性文法を使ってきた．これまでに，さまざまなクラスの属性文法を用いることでコンパイラの構文解析，意味解析，最適化，コード生成の各フェイズが記述できることを確認したほか，インタプリタやグラフィカルユーザインタフェイスの生成にも利用できることが分かった [1].

このインタプリタに対して，文を実行する前にフックをかけ，コルーチン的に協調動作するデバッガフロントエンドを呼び出すように変更することで，文字端末用のデバッガが実現できた．今回作成した PL/0 言語のデバッガの記述に要するコードは従来のデバッガに比べ行数比でわずか 1/65 にすぎない．

さらに，すでにわれわれが開発した属性文法に基づくグラフィカルユーザインタフェイス (GUI) 生成系を利用し，デバッガの GUI も作成した．これにより，xdbx とほぼ同等の機能を持ったものが実現できた．

2　研究背景

われわれは，プログラミング言語処理系を記述する際に属性文法という統一した枠組を用いて記述することを長期的目標とし，Rie と Jun とよばれる二つの属性文法による生成系を開発した (図 1)[1]. Rie は構文解析器，意味解析器などのコンパイラフロントエンドの記述から効率的な解析器Ⓐを生成することを目的とする．Jun は Rie よりも広いクラスの属性文法を解釈し，最適化器Ⓑ，コード生成器Ⓒなどのコンパイラバックエンド，さらに，インタプリタⒹを生成することができる．

ここで特徴的なのは，Rie によるフロントエンドが生成した抽象構文木の形の中間木をそれぞれのバックエンドのフェイズで共有できるため，フェイズ間のインタフェイスをプログラムする必要がなくなっていることである．

3　デバッガ

実現したデバッガは，xdbx 風の外部仕様をもつウインドウ上のデバッガである．

この構成は大きく次の二つに分かれる．

1. インタプリタを利用した文字端末用デバッガ

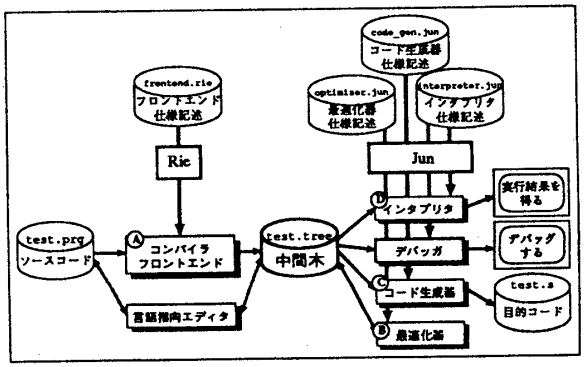

図 1: フロントエンド，バックエンド生成系

これは，dbx に相当するものである．コマンドの例をいくつか挙げると，cont,step,stop at,trace,display 等である．

2. 1. への GUI の付加

以下，4, 5 章でそれぞれについて述べる．

4　文字端末用デバッガ

以下この章では文字端末用デバッガを単にデバッガと呼ぶ．

デバッガはインタプリタとデバッガフロントエンドの二つの部分がコルーチン的に協調動作することで実現されている．インタプリタは通常のインタプリタにデバッガフロントエンドとの通信インタフェイスを与えたものである．デバッガフロントエンドはデバッグ情報を管理し，ユーザが入力するデバッグコマンドにしたがってデバッガの状態を更新し，デバッグ情報の表示やブレイクポイントの設定などをする．

ここで用いるインタプリタは Jun に -debug というデバッグ用のオプションをつけて生成した中間木の解釈器である．このインタプリタは，文に相当する中間木のノードの属性を評価する時にデバッガに制御を渡す，ということ以外はデバッグオプションなしの通常のインタプリタと全く同じ動作をする．

以下に生成されたインタプリタのプログラムの一部を示す．

```
(DEFUN C_STM.EXECUTE (TREE C_STM.PREACT)
  (PROGN (DBG TREE C_STM.PREACT) ---------------- *
    (CASE (CAR TREE)
      (NULL_STM C_STM.PREACT)
                :
      (STM_S (C_STM.EXECUTE (NTH 2 TREE)
        (C_STM.EXECUTE (NTH 1 TREE) C_STM.PREACT))))))
```

リスト中'*'のついた行はデバッグオプションをつけない場合には出力されない．これがオプションの有無による唯一の違いである．ここで，DBG とは，デバッガフロントエンド本体である．

インタプリタはデバッグしているプログラムを中間木の形で保存し，実行時情報として関数の呼びだしスタックおよび変数束縛を持っている．

インタプリタは文の実行の直前にデバッガフロントエンドに制御を渡す．リスト中 3 行目以降が文実行部であり，この様子が良く分かる．

Implementation of a Debugger Based on a Tree Attributed Grammar and a GUI Generation System
Y. Oota, S. Tabuchi, K. Wakita, and M. Sassa
Tokyo Institute of Technology & University of Tsukuba

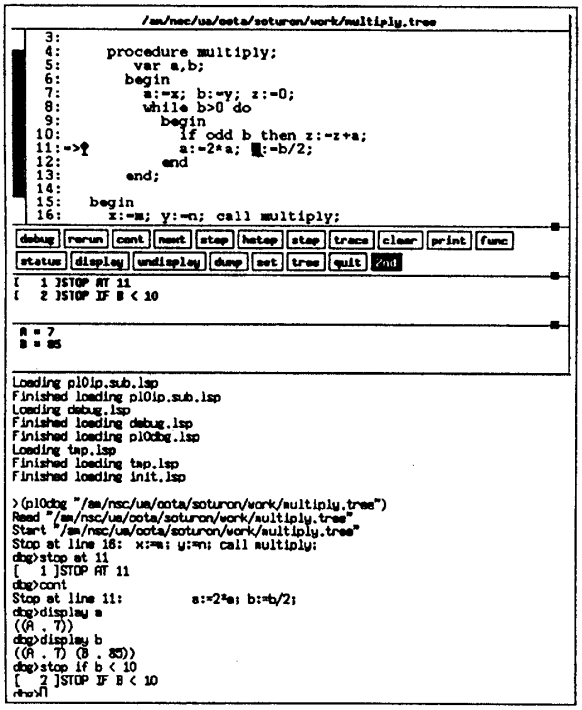

図 2: Xwindow 版デバッガの実行画面

デバッガフロントエンドはユーザが指定する(条件つき)停止の指定や変数表示のための情報，ソースレベルデバッグのためのソースプログラムなどを管理している．

デバッガフロントエンドは，インタプリタから制御を渡されると，停止のための情報に基づき，その実行位置で停止すべきかどうかを判定する．停止する場合は，変数表示の情報を基に，必要な変数の値を表示した後，ユーザの入力を受けつけ，それにしたがってデバッグ情報を更新したり変数値等を表示したりする．ユーザが実行コマンド(cont,step等)を入力すると，インタプリタに制御を渡しステップ実行するという処理を繰り返す．

5 wing による GUI の付加

wing とは本研究室で開発された GUI 生成系であり，属性文法風の GUI 仕様記述と C 言語による補助関数から，既存の文字端末用アプリケーションに対する GUI フロントエンドを生成する．文字端末用のアプリケーションプログラムとの通信は kterm という端末エミュレーションプログラムを利用して行なわれている [2].

前章で述べた文字端末用デバッガにも，この wing を利用することにより，デバッガ自身をほとんど変更することなしに GUI を付加できた．(図 2)

作成されたものは，ほぼ xdbx と同程度の機能を持つ．

GUI を付加することによって，ほとんどのコマンドは，マウスの操作だけで実行できるようになり，操作性が向上した．マウスでは操作しにくいコマンドに対しては，従来通りキーボードから入力すれば良い．

また，常にソースプログラムが見えているので，プログラムの実行位置が把握できるだけでなく，ブレイクポイントとの対応もとりやすい．

6 評価

- 実行速度

表 1: pl0dbg と gdb の記述量の比較

	pl0dbg			gdb
	フロントエンド	1,129		
デバッガ本体	インタプリタ	411	C	133,803
	デバッガフロントエンド	451		
小計		1,991		133,803
GUI	wing	800	C	12,023
合計		2,791		145,826

ただ 1,000 回ループするだけのプログラムを PL/0 で書き，デバッガで実行したところ，約 13 秒かかった．C 言語で書いた同様のプログラムと比較したところ約 30,000 倍遅いことがわかった．実行速度が遅い一番大きな原因は，インタプリタをベースとしているためである．

しかし，一般にデバッグにかかる時間には，ユーザの思考時間が一番大きなウエイトを占めているとも言われている．

- 記述量

表 1 に生成されたデバッガ (pl0dbg) と GNU デバッガ gdb との記述量 (行数) の比較を示す．この表より，デバッガ本体で約 1/65 ,合計で約 1/50 の記述量で済んでいることがわかる．

ここで挙げた二つのデバッガは，対象とする言語のレベルや種類に違いがあるものの，機能的にはほぼ同等といえる．

今回作成したデバッガは実行速度が遅く，対象とする言語は PL/0 のみと制限つきではあるが，記述量や開発期間が約 1 カ月 (デバッガフロントエンド自体は約 2 週間) であることを考慮に入れると，プロトタイプとしては，充分役に立つものだと考えられる．

7 おわりに

木属性文法によるインタプリタの記述があれば，容易にデバッガを作成できるということがわかった．

しかし，現段階では，フロントエンドから受けとる中間木の中には，関数のネストの情報や各関数の中で宣言されている変数の情報が含まれていない．そのため，宣言されていない変数を display したり，関数に入った時に止まる (stop in) という命令で，定義されていない関数を指定した時などに，エラーを出力することができない．この問題は，これらの情報を中間木に記号表として持たせれば，解決するであろう．

また，前章で述べた通り，このデバッガはインタプリタをベースとして動作しているため，実行速度が非常に遅い．今後は，実行形式のプログラムをベースとするデバッガについても考えていきたい．

参考文献

[1] M. Sassa. Rie and Jun:Towards the Generation of all Compiler Phases. In Proc. 3rd Int. Workshop on Compiler Compilers, LNCS Vol. 477, Springer. 1991. pp. 56 - 70.

[2] 金子正俊 他. 属性文法に基づくグラフィカルユーザインタフェース生成系とその評価, 第 46 回全国大会論文集, 1993.

[3] 太田行紀. 木属性文法と GUI 生成系を利用したデバッガの作成, 東京工業大学理学部情報科学科卒業論文, 1993.