3G－6 並列論理型言語Flengのマルチウインドウ・デバッガHyperDEBU

舘村 純一, 小池 汎平, 田中 英彦
{tatemura,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部*

1 はじめに

論理型プログラムのデバッギングにはトレーサのような操作的意味に基づくデバッギングとアルゴリズミックデバッギングのような宣言的意味に基づくデバッギングがある。しかしGHCなどの並列論理型言語では、これらをそのまま適用するのは問題がある。これは、論理型言語に対して並列実行のための機能を拡張したため、プログラムの実行のモデルが従来のものでは不十分となるからである。このため並列論理型言語のデバッガを構築するにはそれに適した実行モデルはどんなものかを考え、さらにその実行モデルをいかにユーザに示すかについて考慮しなければならない。

Flengは我々の研究室で開発されたCommitted-Choice型言語(以下CCLと呼ぶ)であるが、これを対象としたデバッガであるDEBUはプログラムの実行の様子を通信し合うプロセスとしてモデル化する。現在、我々はプログラムの実行のモデルをより分かり易くユーザに示すために、ウインドウ環境を利用したHyperDEBUを開発中である。このプログラムはx-windowインタフェスを持つflengl疑似並列処理系であるxfleng [1]を用いて作成されている。ここではこのHyperDEBUの概要について述べる。

2 Flengの実行のモデル化

2.1 CCLのデバッグの問題点

逐次論理型言語におけるデバッギングに比べてCCLのデバッギングは以下のような問題がある。

- 操作的デバッギング
  並列論理型プログラムの実行では、複数のゴールが互いに同期を取り合いながら並列にリダクションされていくので、実行を直線的に追っていくことはできない。
- 宣言的デバッギング
  CCLではホーン節にガードの概念を加えており、純粋な論理型言語ではなくなっている。よってCCLの宣言的意味は従来の宣言的意味の中に操作的な要素を加えたようなものになる。並列論理型言語における宣言的意味を入出力の集合で表すことがあるが、入出力の集合が同じものでも非決定的なプログラムと合わせると結果が違ってくる可能性があり、入出力の因果関係も考慮しなければならない[2]。そこで、入出力因果関係がわかるように実行をモデル化する必要がある。

*HyperDEBU : Multi-window Debugger for Fleng programs
Junichi TATEMURA, Hanpei KOIKE, Hidehiko TANAKA,
the University of Tokyo

2.2 プロセス型実行モデル

並列論理型言語ではゴール一つをプロセスとみなすことがあるが、ここでは一つのゴールだけでなく、そのゴールから生成される全てのゴール(サブゴール)を含めて一つのプロセスと考える。プロセスの動作の様子は外部からはプロセスの入出力としてとらえられる。

ここで、プログラム中で実行されるあるゴールGと、そこから生成されるゴールからなる集合を実体とするプロセスを、「Gに対応するプロセス」ということにする。これは外部から見た場合以下のように表現できる。

$$\langle G_{skel},\ I,\ O,\ S,\ G_{ins} \rangle$$

G_{skel}はGの*skeletal predicate*で、その引数(変数)が外部との通信の窓口になる。Sはプロセスの状態(terminate / suspend / active)を表す。G_{ins}はゴールの*instance*である。

I,Oは*Input*/*Output*であり、これがプロセスの入出力を表す。G_{skel}の変数がプロセスの外部及び内部からどのように具体化されていくかを示したものである。定義節はプロセスとサブプロセスの関係を規定するもので、その入出力の関係も決められる。これによりプロセスの入出力は再帰的に定義できる[3]。Flengでガードの役割をする部分はヘッドの部分だけであるから、引数に書かれた変数でない項が入出力の因果関係を決める要因となる。例えば、

```
append([H|X],Y,Z) :- Z = [H|Z1], append(X,Y,Z1).
append([],Y,Z):- Z = Y.
```

というプログラムで、append([1],[2],X)が実行された場合、このゴールに対応するプロセスの第三引数の出力は以下のようになる。

```
Gskel: append(A,B,C)
out(C): cond(A = [D|E])
          |-C = [D|F]
            |-cond(E = [])
               |-F = B
```

このcondという部分が入出力の因果関係を表す部分である。

3 実行モデルとウインドウ

ここで述べたような「プロセス」をデバッガに適用する場合、それをユーザにいかに見せるか、いかに扱わせるかが問題となってくる。そこで、プロセスの特徴・見え方を挙げてみる。

- 入出力
  プロセスの実行の様子を外側から抽象化して観察するもので、複数のプロセスが並列に存在してそれらの通信する様子を見るといったことができる。

- 計算木
  一つのプロセスは一つの計算木に対応する。そのサブツリーはプロセスのサブプロセスに対応する。計算木は、プロセスがどのようなサブプロセスに分割されたかを示すものであり、あるプロセスからそのサブプロセスにプログラマの注視点を移動したいとき、ゴールがどのようにリダクションされたかをたどりたい時に使える。
- ゴールの集合
  プロセスの実体はゴールの集合である。実行中に存在するゴールを個々に扱うのでなく、その中の一部を一まとまりにして、そこに注目する、あるいは逆に抽象化してしまうということができる。

このように階層的構造を持ち、互いに関係する複数のものを同時に取り扱うものを表示する場合はウインドウ環境を活用するのが必要不可欠と思われる。そもそも並列プログラムの実行過程は多次元的な情報であるから、これを一次元的なインタフェースで取り扱うのは限界がある。一方、従来直線的にしか表されなかったテキストを多次元的に表現するハイパー・テキストの研究が盛んである。並列プログラムのデバッグにもこのようなアプローチを導入していくべきであろう。

ここで、一つのプロセスには一つのウインドウを割り当てる(以下では「プロセス・ウインドウ」と呼ぶ)。一つのプロセス・ウインドウは図1のようになる。上に述べたプロセスの特徴からウインドウの主な構成は以下のようになる。

- **トップレベルのゴール**: 引数ごとに、入出力のサブウインドウ、インスタンスのサブウインドウが開く。
- **TREE サブウインドウ**: サブプロセスの様子を調べ、必要ならばサブプロセスのウインドウを開く。
- **GOALS サブウインドウ**: ゴールを一つずつ扱いたい時に開く。ゴール・キューのようなイメージであり、マウスで一つを選んで操作する。
- **その他**: メッセージ用のウインドウ、操作用のボタンなど。

これらを開いた様子は図2のようになる。

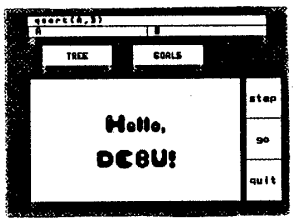

図1: プロセス・ウインドウ

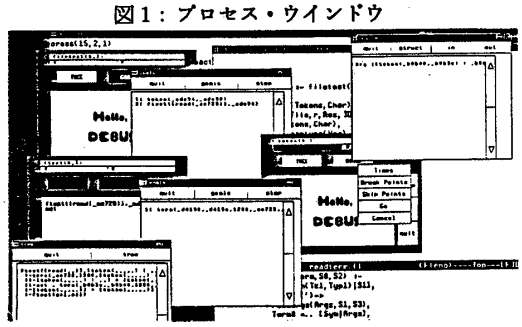

図2: HyperDEBU の概観

4 デバッグ手法

4.1 トレーサ

従来のトレースはゴールの実行を追っていくものだけだったが、HyperDEBU ではそれだけでなく、プロセスの入出力を追っていくというアプローチも可能である。トレースにおいてプログラムを実行させるやり方には、(1) プロセス(ウインドウ)に対してブレークポイントをつけてリダクションする (2) ゴール(サブウインドウ)に対してゴールを選択しながらリダクションする という二通りの方法がある。

4.2 アルゴリズミックデバッギング

Committed-Choice 型言語の宣言的デバッグを行なう場合、入出力因果関係を考慮しなければならず、そのためユーザにインスタンスでなくプロセスを示して正否を問う手法を提案した[3]。ただアルゴリズミックにプロセスを提示するのではなく、入出力やインスタンスで間違っている部分をユーザーが指摘し、その情報を用いて次のプロセスを選んでウインドウを開くような機能をつける予定である。

5 実装上の問題

5.1 使用メモリの問題

HyperDEBU では実行の結果を計算木の形で記憶するが、実行のすべてを記憶するとメモリが膨大になる。しかし計算木が必要になる部分はデバッグ時に注目したい部分であり、それ以外の部分は木を記憶する必要はなく、不可分なプロセスとして抽象化できる。

5.2 実行速度(計算量)の問題

現在 HyperDEBU は Fleng インタプリタ上でメタインタプリタを用いて実現されている。これは実行に問題があるが、対策として、(1) デバッグに必要のない部分はそのまま実行してしまう (2) メタプログラムをプログラム変換する (3) コンパイル時にデバッグ用のコードを埋め込む等が考えられる。

6 おわりに

現在はまだ HyperDEBU は試作段階だが、今後はより機能を充実させて実用的な形で完成させたい。なお、本研究は文部省特別推進研究 No.6265002 による。

参考文献

[1] 勝亦, 小池, 田中 : "並列オブジェクト指向言語 Fleng++ によるユーザインタフェースの構築", 第 40 回情報処理全国大会.

[2] Takeuchi, A.: *A Semantic Model of Gurded Horn Clauses* Technical Report, ICOT, 1987.

[3] 舘村, 田中 : "並列論理型言語 FLENG のデバッガ", Logic Programming Conference '89, 1989.