6Q-2

Committed-Choice型言語 FLENGのタイプ/モード推論

小中裕喜、田中英彦
(東京大学工学部)

1. はじめに

現在、我々はCommitted-choice型言語FLENG[1][3]を対象として、新しいプログラミング環境を開発している。本稿ではその中でFLENGプログラムのタイプ/モード推論アルゴリズムについて検討する。

一般に論理プログラムにおけるタイプ/モード情報はコンパイラの最適化や、ある種のバグの検出などに有効であることが知られている[4]。FLENGのようなCommitted-choice型言語においても、それらがプログラマやメタシステムに有力な情報を与えるのは同様である[2]。

FLENGはGHCにおけるガード及びゴール間の論理的な関係を排除した言語であり、一方でActive Unify Annotation(AUA)やBind-to-non-variable Annotation(NVA)の導入による記述の簡略化、実行効率化が図られている。

また、FLENGには上位言語として並列オブジェクト指向言語FLENG++[5]、下位言語として、Single Reference Annotation(SRA)などを導入した、FLENGのスーパーセットでもあるFLENG-- が開発されている。本稿ではこれらにFLENGのタイプ/モード推論結果をどのように応用するかについても検討を行う。

2. タイプ/モード推論

2.1 モード

FLENGのプログラムはPrologと同様にいくつかの節の集合として定義される。節のモードはその引数ごとに、またその引数が関数であればその引数に対して定義される。節のヘッド内においては、その節が呼び出されたことによって初めて値が未定義でなくなるものを出力、それ以外を入力とする。節の各ボディゴール内では、そのゴールが呼び出されたことによって値が未定義でなくなるものを出力、それ以外を入力とする。

入力モードの場合、そのゴールの実行によって具体化される部分がなければ'+'、それ以外の場合を'#'、出力モードの場合は'-'、不明の場合は'?'で表す。

述語のモードはその引数ごとに、その述語の定義を構成する節の集合から定まる。また、FLENGにおけるプログラミングでは、述語が双方向に用いられることは少ない。よってこのことをFLENGのモード解析における前提条件と考える。すなわち、

「ある述語を構成する各節の同一引数位置のモードは一定である」

また、同様に

「ある述語を構成する各節の同一引数位置にある同一関数の、同一引数位置のモードは一定である」

という条件も設ける。これによって、例えばリストの各要素のモードは同一である、という結論が導かれるが、通常のプログラミングにおいてこの条件はそれほど強くないと考える。

2.2 タイプ

タイプとしては各述語、節の引数ごとに、タイプ変数を含むHerbrand領域上のものを扱う。ただし、整数については'$int'というアトムとみなす。タイプのユニオンも許す。また、リストのような再帰的な構造を持つものは、例えば次のように扱う。

```
[1,2,3,a,b] => listof('$int';a;b)
```

ここで、';'はユニオンを表す。

2.3 推論アルゴリズム

推論はボトムアップに行っていく。

最初に述語単位で独立にモード付けを行っていく。

まず、各節において

1) 全ての整数を'$int'に変換する。
2) 全ての変数に'?'とモード付けする。
3) ヘッド中で、AUAの範囲外に複数回現われる変数は'+'をモード代入する。
4) ヘッド中でNVAのついた'?'モードの変数は'#'モードにして、NVAを取り除く。
5) ヘッド中で、AUAの範囲外に残る'?'モードの変数で、ボディやAUAの範囲内に参照されていなければ'#'モードにする。
6) ヘッド中でAUAの範囲内にあるアトム、関数、及びヘッド中の他の位置で入力モード付けされている変数に'-'モードをつけて、AUAを取り除く。
7) ヘッド中でモード付けされていないアトムはすべて'+'モードとする。モード付けされていない関数は、引数がすべて'+'モードになれば'+'、それ以外は'#'モードにする。

次に述語単位で、

8) 2.1の条件に合うかどうかをチェックする。その際、'?'モードが具体化されることもある。

```
pred(?X) :- ...        ⇨    pred(#X) :- ...
pred(+Y) :- ...             pred(+Y) :- ...
```

それから節単位に、

9) ヘッドで入力モード付けされた変数がボディにも現われていれば、同一モードを代入する。
10) ボディ中のアトムはすべて'+'モードとする。モード付けされていない関数は、引数がすべて'+'モードになれば'+'、それ以外は'#'モードにする。
11) ボディ中に1回しか現われず、ヘッドにも現われない変数は出力モードとする。

ここまでで各述語及び関数の引数にはなんらかのモード付けがなされている。

次に節ごとに変数のタイプ付けを行う。簡単のため、ここでは述語を構成する各節のボディゴールに現われる述語のタイプはすべてわかっているものとし、概略を示すにとどめる。

12) ボディゴールに対応した述語のタイプ/モードから

Type/Mode Inference on Committed-Choice Language FLENG
Hiroki KONAKA, Hidehiko TANAKA
Univ. of Tokyo

ボディゴールの変数のタイプ／モード付けを行う（タイプ／モードの１方向ユニフィケーション）。
13) ボディゴール間で共有されている変数について、タイプのユニフィケーションを行う。モードに関しては、出力モードに複数回なっていないかをチェックする。また、ヘッドに現われていないのにボディで出力モードにならない変数をチェックする。
14) 各変数のタイプ／モードをもとにヘッド中の各変数のタイプ／モードを決定する。

以上はボトムアップ的処理であったが、これだけではモードを決定するのに不十分である。そこで、
15) 12)の逆で、ボディゴールのタイプ／モードから定義側のヘッドの変数のモード付けを行う（モードの１方向ユニフィケーション）。
という処理を最後に行う。また、トップレベルのゴールがわかっている場合、15) でタイプに関しても同様に行えば、FLENG の実行を最適化するのにより望ましいタイプ情報が得られる。

３．　応用

３．１　バグの発見

タイプ／モード推論中にエラーが生じた場合、その箇所にバグが存在する可能性が高い。また、バグが直接その場所にない場合でも、バグを探索する空間はShapiroのAlgorithmic Debugging のように計算木中ではなく、プログラム中に限られていて有利である。

３．２　ＦＬＥＮＧ＋＋のコンパイル時の利用

FLENG++ の述語定義はメソッド定義とクラス内でのみ参照されるローカル述語定義に分かれる。ローカル述語定義はFLENG で記述される。FLENG++ のプログラムはFLENG へコンパイルされてから実行される。

一般にメソッドとローカル述語の間でオブジェクトの受渡しがあるが、その参照が増える場合それが、プリミティブなオブジェクトかどうかによって、単にコピーするか、マージをはさむかが異なる。またマージをはさむ場合、モードによってはさむ方向がかわる。そこで実行時にこれらをチェックして上記の処理を行うようなコードを生成する必要があるが、タイプ／モード情報がわかっていればその様なコードを排除することが可能になる。

３．３　ＦＬＥＮＧプログラムへのＳＲＡの付加

構造体が単一参照されていることが保証されている場合、その構造体を利用する節でそのセル領域を回収、再利用できるということを明示するのがSRA である。これをタイプ／モード情報を利用して自動的に付加することを考える。

まず、トップレベルのゴールが（参照関係が明らかになる程度に）与えられる場合、
1) 全ての節のAUA を展開する。
2) 全ての節に対して、構造体をタイプとして持つ変数に対してのみ次のことを行う。
  2-1) ボディ中の出力モードの変数で、他に参照されていないものについては、それらが出現するボディゴールに対応した述語の定義節中の対応するヘッドの引数位置にNRマークをつけていく。
  2-2) 新たにNRマークのついた変数がボディ中に１回しか出現しない場合、それらが出現するボディゴールに対応した述語の定義節中の対応するヘッドの引数位置にNRマークをつけていく。
3) 全ての節に対して、構造体をタイプとして持つ変数に対してのみ次のことを行う。
  3-1) ボディ中で入力モードとなっている各変数について、参照回数を数える。ただし、同一ボディゴール中に同一変数が複数回現われる場合、対応する定義節を参照して、実際の参照回数をみる。また、ヘッドでその変数が出現し、しかもNRマークがついていなければ、参照回数に１加える。
  3-2) 上記の参照回数が３を越える変数がある場合、その変数の出現位置にMRマークをつけるとともに、それらが出現するボディゴールに対応した述語の定義節中の対応するヘッドの引数位置にMRマークをつけていく。
  3-3) 新たにMRマークのついたヘッドの変数について、それらが出現するボディゴールに対応した述語の定義節中の対応するヘッドの引数位置にMRマークをつけていく。
4) 全ての節について、ヘッドにMRマークがついていない構造体があれば、SRA をつける。

トップレベルのゴールが未知の場合、そこで参照が増える可能性があるので、この方法はこのままでは使えない。この対策は２つ考えられる。
1) 構造体のコピーを導入する。
2) SRA をつけられる述語定義はコピーして名前を付けかえる。プログラム中でそれを呼び出している部分についても名前をつけかえる。

1)はトップレベルのゴール間での構造体の受渡しに、常にコピーを用いるわけだが、特別扱いが必要となる。

2)は実行時のメモリ消費を抑える分、コード量の増加をともなう。

４．　今後の課題

デバッグの支援はこのシステムの目的の１つであり、そのためのユーザインタフェースを用意する必要がある。

また、オブジェクト指向プログラミングにおけるメッセージ処理について、より正確に表現することも考えたい。

さらにこのシステムの延長として、FLENG++ に対するタイプ／モード推論を核とするプログラミング環境を構築したい。

参考文献

[1] Nilsson,M. and Tanaka,H.,"The Art of Building a Parallel Logic Programming System",Proc. Logic Prog. Conf.'87,Tokyo,Jun.,1987.
[2] 越村,"並列論理型言語のモード解析とその応用",情報処理学会第36回全国大会,5H-3,1988.
[3] 小池,島田,朝海,田中,"高並列エンジン実験環境PIEE上へのFLENGの実装",情報処理学会第36回全国大会,1D-7,1988.
[4] Kanamori,T. and Horiuchi,K.,"Type Inference in Prolog and Its Applications",TR-095,ICOT,1984.
[5] 中村,小中,田中,"Committed-choice型言語 FLENGとその上のユーザ言語 FLENG++",情報処理学会第37回全国大会,2Y-3,1988.