並列論理型言語のモード解析とその応用

越村 三幸
新世代コンピュータ技術開発機構研究所

1 はじめに

論理プログラムのモード情報は、Prologにおけるコンパイラの最適化やGHC[4]プログラムの部分計算機[1]にとって有効であることが知られている。

並列論理型言語では、共有変数を通信チャネルとみなしてアクティブなプロセスを結合したプロセスのネットワークを意識してプログラミングすることが多い[3]。

そして、チャネルの結合間違いなどからバグが発生する。入力プロセスのみに繋がれたチャネルが存在する場合、デッドロックの可能性がある。また複数の出力プロセスに繋がれたチャネルが存在する場合、フェイルする可能性がある。このようなバグは単純なミス[1]から引き起こされることが多い。

並列論理型言語のプログラムのモード情報によって、プロセス間の依存関係を明らかにすることが出来る。したがって、モード解析によってチャネルの結合間違いといったバグの検出が出来ることになる。

本論文では並列論理型言語としてGHCを取り上げ、GHCプログラムのモード情報がデッドロックを含むある種のバグの検出に有効であることを示す。

2 モード解析

一般にPrologプログラムには方向性がない。ある引数は入力として用いられるかもしれないし、出力として用いられるかもしれない。したがって、Prologプログラムにおいてモード解析はトップダウン的に行われる。

一方、通常のGHCプログラムには方向性がある。つまり入力引数と出力引数は固定しているのが普通である。したがって、GHCプログラムにおいてモード解析はボトムアップ的に行うことが可能である。

本節では、GHCプログラムのモード[2]付けの定義をし、それに基づいてプログラムのクラスの定義[2]をする。

モードとして +(入力)、−(出力)、?(不定)を考える。

節 $H :- G_1,\ldots,G_m \mid B_1,\ldots,B_n.\ (m \geq 0,\ n \geq 0)$ のHをヘッド部、$G_1,\ldots,G_m$をガード部、$B_1,\ldots,B_n$をボディ部と呼ぶ。

定義 1 (節のモード) 述語 P の各節の各引数Arg_iのモードを以下のように定義する。ボディ部とガード部に現れる述語 Qのモードは既知のものとする。但し、Qの第 j 引数のモードが Pのモードに依存して決定する[3]場合、Qの第 j 引数のモードは不定とする。

Arg_iが変数以外の時、入力。

Arg_iが変数 Xの時、次の規則に従う。

- Xがヘッド部に 2回以上現れる時は入力。
- Xがガード部又はボディ部の入力引数位置に現れ、ボディ部の出力引数位置に現れない時は入力。
- Xがガード部の出力引数位置に現れ、 ボディ部の出力引数位置に現れない時は入力。
- Xがヘッド部に 1回しか現れず、ボディ部の出力引数位置に現れ ガード部に現れず、さらにボディ部の入力引数位置に現れない時は出力。
- 上記以外は不定。

定義 2 (リテラル間関係 $\prec$) 節の右辺のリテラル(Q_i)間の 2項関係$\prec$を次のように定める。

$Q_i \overset{X}{\prec} Q_j \overset{def}{\Longleftrightarrow} Q_i$と$Q_j$は論理変数 Xを共有し Xは、Q_iの出力引数位置に現れQ_jの入力引数位置に現れる。

2つのリテラル Pと Qが$P \overset{X}{\prec} Q$ の関係にある時、 Xに関して Qは Pに依存しているという。

Pに含まれる全ての変数 Xに対して $Q \overset{X}{\prec} P$となるような Qが存在しない時、 Pは$\prec$に関して極小であるという。またPに含まれる全ての変数 Xに対して $P \overset{X}{\prec} Q$となるような Qが存在しない時、 Pは$\prec$に関して極大であるという。

リテラルは、実行時におけるゴールに対応している。プログラム実行時におけるゴール群も関係$\prec$によって関係付けられる。そして、実行時に$\prec$のチェインに着目してゴール群を観察することによってデッドロックを検出することが可能である[5]。またデッドロックが発生した場合、$\prec$に関して極小なゴールの定義にバグが存在する可能性が高いと思われる。

定義 3 (良い節定義) 節 CLが良い節である $\overset{def}{\Longleftrightarrow}$ 以下のことが全て成り立つ。

1. CL中に各変数は、 2回以上出現する。
2. ヘッド部に現れない変数は、入力引数位置または出力引数位置のみに 現れることはない。
3. $\prec$に関して極小なリテラルについて 入力引数位置に変数 Xが出現する場合、次のことが成り立つ。 Xは他のリテラル中またはヘッド部の引数Arg_iに出現し、 ヘッド部に出現する場合Arg_iのモードは入力である。
4. $\prec$に関して極大なガード部のリテラルについて、 出力引数位置または不定引数位置変数 Xが出現する場合、次のことが成り立つ。 出力引数位置に現れ

[1] スペルミス、引数の位置の間違いなど
[2] 組込み述語のモードは既知のものとする
[3] 再帰呼出しの場合

Mode Analysis for GHC Programs and Its Application
Miyuki KOSHIMURA (ICOT Research Center)

る変数Xは他のリテラル中または ヘッド部の引数Arg_iに出現し、 ヘッド部に出現する場合Arg_iのモードは入力である。 不定引数位置に現れる変数Xは、他のリテラル中に現れるか、 ヘッド部の引数Arg_iにも出現し、 Arg_iのモードは入力または不定である。

5. $\prec$に関して極大なボディ部のリテラルは、 出力引数または不定引数を持ち次のことが成り立つ。 出力引数位置に現れる変数Xは他のボディ部のリテラル中または ヘッド部の引数Arg_iに出現し、 ヘッド部に出現する場合Arg_iのモードは出力である。 不定引数位置に現れる変数Xは、他のリテラル中に現れるか、 ヘッド部の引数Arg_iにも出現し、 Arg_iのモードは出力または不定である。

定義 4 (述語のモード) n個の節$CL_i(1 \leq i \leq n)$からなる述語pの第j引数のモードp^jを次のように定義する。但し、CL_iの第j引数のモードをCL_i^jとし、モードの集合M_jを$M_j = \{CL_i^j | 1 \leq i \leq n\}$とする。

$$p^j = \begin{cases} + & M_j = \{+,?\}\text{または、}\{+\}\text{の時} \\ - & M_j = \{-,?\}\text{または、}\{-\}\text{の時} \\ ? & \text{その他} \end{cases}$$

また$M_j = \{+,-\}$若しくは$\{+,-,?\}$の時モードが矛盾したという。

上記述語のモードの定義は強過ぎるかも知れないが、実用上は問題ないと考えている。

定義 5 (良い述語定義) 述語pのモードが矛盾しない時、pの定義は良いという。

例 1 (append プログラム) appendプログラム：

```
append(    [],Y,Z) :- true | Z = Y.
append([A|X],Y,Z) :- true |
   Z = [A|Zs], append(X,Y,Zs).
```

の第1節のモードはappend(+,?,?)で、第2節のモードはappend(+,?,-) である。そして、述語appendのモードは、append(+,?,-)となる。

例 2 次のような双方向性を持つ述語は、良い述語定義ではない。

```
p(a,B) :- true | B = b.
p(A,b) :- true | A = a.
```

3 応用

良い節定義、良い述語定義ではない節並びに述語に対し警告を発することによって、バグを検出出来ることがある。素数生成プログラム[4] を例にとって、2で述べたモード解析のバグ検出への有効性を示す。プログラムを以下に示す。

```
primes(Max, Ps) :- true |
   gen(2,Max,Ns), sift(Ns, Ps).
gen(N,Max,Ns) :- N >  Max | Ns = [].
gen(N,Max,Ns) :- N <= Max | Ns = [N|Ns1],
   N1 := N + 1, gen(N1,Max,Ns1).
sift(    [],Zs) :- true | Zs = [].
sift([P|Xs],Zs) :- true | Zs = [P|Zs1],
   filter(P,Xs,Ys), sift(Ys,Zs1).
filter(_,    [],Ys) :- true | Ys = [].
filter(P,[X|Xs],Ys) :- X mod P =:= 0 |
                  filter(P,Xs,Ys).
filter(P,[X|Xs],Ys) :- X mod P =\= 0 |
   Ys = [X|Ys1], filter(P,Xs,Ys1).
```

このプログラムをモード解析すると、primes(+,-), gen(+,+,-), sift(+,-), filter(+,+,-)となる。

3.1 バグが検出出来る例

述語 sift の第1節の定義を間違えて

```
sift([],[]) :- true | true.
```

にしたとする。この時第1節のモードは sift(+,+) 第2節のモードは sift(+,-) となり、モードが矛盾する。

また、述語 primes の定義を間違えて

```
primes(Max, Ps) :- true |
   gen(2,Ns,Max), sift(Ns, Ps).
```

の様にしたとする。この時変数 Ns は入力引数位置にしか現れず、良い節定義とはならない。

3.2 バグが検出出来ない例

述語 sift の定義を次のように間違えた場合、この方法ではバグは検出されない。

```
sift([P|Xs],Zs) :- true |
   filter(P,Xs,Ys), sift(Ys,Zs).
```

4 問題点と今後の課題

ここで述べたモード解析について次のような問題点が分かっている。

- 差分リストの扱い： 解析出来ない[4]。 トップダウン的アプローチが必要となるだろう。
- 未完成メッセージの扱い： 十分な情報が得られない。 タイプ推定を行う必要があると思われる。

今後、モード解析プログラムを試作し実際のプログラムで評価を行う予定である。またそのプログラムは部分計算機の自動化のための有効な部品になると考えられる。

参考文献

[1] K. Furukawa, A. Okumura, and M. Murakami. UNFOLDING RULES FOR GHC PROGRAMS In 日本ソフトウェア科学会第4会大会論文集, 1987.

[2] U.S. Reddy. Transformation of Logic Programs into Functional Programs. In *Proc. of '84 SLP*, 1984.

[3] S. Takagi. *A Collection of KL1 Programs - Part 1 -*. TM 311, ICOT, 1987.

[4] K. Ueda. *Guarded Horn Clauses*. TR 103, ICOT, 1985.

[5] K. Yamato and T. Nishimura. Run Time Detection of Possibility for Deadlock or Starvation in Parallel Program. 1982.

[4] ? と推定してしまう