GHCプログラムのモード解析

3G－1

上田 和紀
(財)新世代コンピュータ技術開発機構

森田 正雄
(株)三菱総合研究所

1. はじめに

GHCをはじめとする並列論理型言語では，プロセス間通信路(ストリーム)をリストによって実現し，単一化によって通信を行なう．これは概念の単純さや柔軟さという点からは大変すぐれているが，効率的な実現が困難ではないかという批判も一方にある．つまり，他の並列言語と比べて，次の点で不利であるというわけである：

(1) 普通の実現技法では，単一化のたびに情報の流れの向きと，失敗の可能性とを検査しなければならない．
(2) 普通の実現技法では，ストリーム通信のたびにconsを行なうことになる．

本稿では，これらのオーバヘッドを除去するための，プログラムのモード解析手法について述べる．この手法を応用した最適化技法とその効果については，[3][4]を参照してほしい．

2. モード解析の目的

我々のモード解析の目的は，どのゴールが共有変数のどの部分を具体化する(またはしない)かを推論することである．これは，ゴールの実行前後の引数の具体化状態を推論するDebrayの体系[1]と目的を異にしている．

モード体系の設計にあたっては，単一化による通信のもたらす柔軟性を損なわないことが重要である．つまり，双方向通信や，ストリームのストリームも扱いたい．だが，静的解析のための体系であるから，制約も伴う．実際，我々の体系は，多くのGHCプログラムが，次の仮定を満たしているか，満たすように書き換えられるということに基いている：

(1) 共有変数の具体化は，競争的にではなく，協調的に行なわれる．つまり，変数が複数回出現する場合，そのうちの一つだけが，主関数記号を決めることのできる出力出現であり，他は入力出現である．
(2) ある変数の出現が入力か出力かは，その出現とトップレベルの述語記号の間に存在する祖先の関数・述語記号によって決まる．つまり，述語の引数のモードは一意的であるが，関数の引数のモードは，その関数がどこに出現するかに依存する．ただし，単一化の組込述語 '=' は(局所的にモードが決められるので)，呼出しごとにモードが異なってもよい．

(1)は，ボディの単一化ゴールが，必ず未定義変数への代入であるということも意味する．

Mode Analysis of GHC Programs
Kazunori Ueda[1], Masao Morita[2]
[1]Institute for New Generation Computer Technology
[2]Mitsubishi Research Institute

3. モード体系

述語記号の集合を $Pred$，関数記号の集合を Fun，アトムの集合を $Atom$，項の集合を $Term$ と書く．述語記号 p が n_p 引数のとき，N_p で集合 $\{1,\ldots,n_p\}$ を表わす．関数記号 f についても N_f を同様に定める．さらに，**パス**の集合 P_t(項用)と P_a(アトム用)を，次のように定める：

$$P_t = (\sum_{f\in Fun} N_f)^*, \quad P_a = (\sum_{p\in Pred} N_p)\times P_t.$$

P_t の要素は $\langle f_1,j_1\rangle\ldots\langle f_n,j_n\rangle$ と，P_a の要素は $\langle p,i\rangle p'$ $(p'\in P_t)$ と書く．パスは，項やアトムの部分項を指定するためのものである．つまり，各項 t について，関数 $\bar{t}: P_t \to Term$ を次のように定める(アトムについても同様)：

$$\begin{cases} \bar{t}(\epsilon) = t; \\ \bar{t}(\langle f,j\rangle p') = \begin{cases} \overline{t_j}(p'), & t \text{ が } f(t_1,\ldots,t_n) \text{ の形のとき;} \\ \bot, & \text{それ以外.} \end{cases} \end{cases}$$

モードは，次のような集合 M の要素である：

$$M = P_a \to \{in, out\},$$

ここで，M の値域について，$in \neq out$ を仮定する．

モード m の意味は次の通りである．あるゴールが，ある時点で $a\in Atom$ という形をしていたとし，$p\in P_a$ を，$\bar{a}(p)$ が未定義変数であるようなパスとする．このとき，

(a) $m(p) = in$ は，$\bar{a}(p)$ が他の項(変数を含む)に書き換えられることがないことを意味し，
(b) $m(p) = out$ は，このゴールが，$\bar{a}(p)$ の値を待って中断することがないことを意味する．

4. モード解析

モード解析の目的は，与えられたプログラムに対して可能(feasible)なモード，つまり後述の制約をすべて満たすモードを求めることである．

解析を簡単にするために，プログラムは[5]の方法にしたがって正規化しておく．正規化したプログラムは，次の性質を満たす：

(1) ガードには単一化ゴールはない．
(2) 節のボディの単一化ゴールの集合は，$v_1 = t_1$, …, $v_n = t_n$ という形をしている．ここで，
- $v_1,\ldots,v_n$ は，頭部に現れる相異なる変数であり，
- $v_1,\ldots,v_n$ は，$t_1,\ldots,t_n$ やボディの他のゴールに現れず，
- もし t_i が変数ならば，それは頭部にも現れる．

また，述語 '=' のオーバーローディングを扱うために，プログラム中のすべての '=' の呼出しを，'$=_1$', '$=_2$', … と添字によって区別する．

さて，プログラムがその可能なモード m に課す制約は次の通りである：

(1) ある述語 q がパス p を調べていれば，$m(p) = in$ である．ここで，q がパス p を調べるとは，
  (1a) $\overline{h}(p)$ が非変数であるような頭部 h をもつ (q の) 節がある
  (1b) 頭部 h をもつある節と，p のある接頭辞 (prefix) $p' \in P_a$ について，$\overline{h}(p')$ が二回以上出現する
  (1c) 頭部 h をもつある節と，p のある接頭辞 $p' \in P_a$ について，$\overline{h}(p')$ がガードゴールに出現する
  のいずれか一つ以上が成り立つことである．(条件 (1c) は，ガードゴールの種類によっては弱めるべきであるが，詳細は省く．)

(2) ボディの単一化ゴール $t_1 =_k t_2$ のふたつの引数は，逆のモードをもつ．すなわち，

$$\forall p \in P_t\big(m(\langle =_k, 1\rangle p) \neq m(\langle =_k, 2\rangle p)\big).$$

(3) あるボディ・ゴール a について，部分項 $\overline{a}(p)$ が変数でなければ，$m(p) = in$ である．

(4) ある節に，変数 v が，n 回出現するとする．ただし，頭部における2回目以降の出現と，ガードでの出現は除外して数える．i 回目の出現は，アトム a_i のパス p_i に現れるとする．各 i について，モード $m_i \in M$ を次のように定める：

$$\begin{cases} \forall p \in P_a\big(m_i(p) = m(p)\big), & a_i \text{ がボディ・ゴール；} \\ \forall p \in P_a\big(m_i(p) \neq m(p)\big), & a_i \text{ が頭部.} \end{cases}$$

このとき，次の制約が成り立つ：

$$\forall p \in P_t \exists i \leq n\big(m_i(p_i p) = out \wedge \forall j \leq n(j \neq i \rightarrow m_j(p_j p) = in)\big).$$

これは，v のインスタンスの各関数記号は，v の出現のうちのただ一つが決定するということである．i は p に依存して決まることに注意．m_i を定義したのは，変数の頭部での出現とボディでの出現を統一的に扱うためである．つまり，節の中から頭部を眺めたときのモードと，呼出し側から眺めたときのモードは，ちょうど逆になるわけである．

5. 例と考察

次のプログラムを考える：

```
t(M,S) :- M=:=0 | S=[].
t(M,S) :- M=\=0 |
  S=[push(M),pop(N)|S1], N1:=N-1, t(N1,S1).

s([],          _     ) :- true | true.
s([push(X)|S],D      ) :- true | s(S,[X|D]).
s([pop(X)|S], [Y|D1]) :- true | X=Y, s(S,D1).
```

$m(\langle \mathtt{t}, i\rangle p)$ を $t_i(p)$，$m(\langle \mathtt{s}, i\rangle p)$ と $s_i(p)$ を略記することにすると，t から得られる制約は次のものを含む ('.' は非空リストの関数記号)：

$$t_1(\epsilon) = in,\ t_2(\epsilon) = out,\ t_2(\langle ., 1\rangle) = out,$$
$$t_2(\langle ., 2\rangle) = out,\ t_2(\langle ., 2\rangle\langle ., 1\rangle) = out,$$
$$\forall p \in P_t\big(t_2(\langle ., 2\rangle\langle ., 2\rangle p) = t_2(p)\big),$$
$$t_2(\langle ., 1\rangle\langle \mathtt{push}, 1\rangle) = out,\ t_2(\langle ., 2\rangle\langle ., 1\rangle\langle \mathtt{pop}, 1\rangle) = in.$$

また s からは次のようなものが得られる：

$$s_1(\epsilon) = in,\ s_1(\langle ., 1\rangle) = in,$$
$$\forall p \in P_t\big(s_1(\langle ., 2\rangle p) = s_1(p)\big),$$
$$s_2(\epsilon) = in,\ \forall p \in P_t\big(s_2(\langle ., 2\rangle p) = s_2(p)\big),$$
$$\forall p \in P_t\big(s_2(\langle ., 1\rangle p) = s_1(\langle ., 1\rangle\langle \mathtt{push}, 1\rangle p)\big),$$
$$\forall p \in P_t\big(s_2(\langle ., 1\rangle p) \neq s_1(\langle ., 1\rangle\langle \mathtt{pop}, 1\rangle p)\big).$$

$s_1(\langle ., 1\rangle\langle \mathtt{push}, 1\rangle)$，$s_1(\langle ., 1\rangle\langle \mathtt{pop}, 1\rangle)$，および $s_2(\langle ., 1\rangle)$ の具体値は，s だけからはきまらない．だが，ゴール t(10,S) がゴール s(S,[]) を駆動するという情報があれば，$s_1(\langle ., 1\rangle\langle \mathtt{push}, 1\rangle)$ と $s_2(\langle ., 1\rangle)$ は in に，$s_1(\langle ., 1\rangle\langle \mathtt{pop}, 1\rangle)$ は out に定まる．一般に，可能なモードは一意には決まらないが，コード生成に関係するパスのモードが決まれば問題はない．このためには，外界とのインタフェース (トップレベルのゴールや入出力プロセスの引数) のモードについての表明が一般に必要となる．

モード解析に成功したプログラムが，2節の仮定 (1) や，3節に述べた性質 (a)(b) を満たすことは容易にわかる．また，解析が成功すれば，オカー・チェック以外の原因では単一化が失敗しないことが保証される．これは重要なことで，このような解析がないと，Strand [2] のように，ボディの単一化を代入 (v := t) の形で書かせることにしても，動的な検査を省略できない．

紹介した解析手法は，抽象解釈などによる大域的データフロー解析ではなく，個々のプログラム節が課す局所的制約に基づいている．したがって，分割コンパイルとの親和性もよい．この場合，できるだけ多くのことをコンパイル時に決定するには，大域的述語のモードを宣言すればよい．宣言した情報の一貫性は，リンク時に検査される．この検査も，ここに示したモード解析の枠組の中で行なうことができる．

参考文献

[1] Debray, S. A. Static Inference of Modes and Data Dependencies in Logic Programs. *ACM TOPLAS*, Vol. 11, No. 3 (1989), pp. 418–450.

[2] Foster, I. and Taylor, S. Strand: A Practical Parallel Programming Language. In *Proc. 1989 North American Conf. on Logic Programming*, MIT Press, 1989, pp. 497–512.

[3] 森田正雄，上田和紀： GHC プログラムの最適化. ロジックプログラミングコンファレンス '89, ICOT, 1989, pp. 203–214.

[4] 森田正雄，上田和紀： GHC のメッセージ指向の処理方式. 本予稿集.

[5] Ueda, K. and Furukawa, K. Transformation Rules for GHC Programs. In *Proc. Int. Conf. on FGCS'88*, ICOT, 1988, pp. 582–591.