ICOT Technical Memorandom: TM-0973

TM-0973

ペトリネットで表現された GHCプログラムの実行履歴

中島　俊介、長谷川　晴朗（沖）

November, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

ペトリネットで表現されたGHCプログラムの実行履歴

中島俊介　　長谷川晴朗

沖通信システム株式会社　　沖電気工業株式会社

本稿では，FGHCプログラムの実行過程から，その並列性を抽象する方法を示す．リダクションをトランジションに，ゴールと変数をプレースに対応させることで，プログラムの実行過程をペトリネットでモデル化する．そのようなペトリネットを求めるため，リダクション時に削除・追加されるゴールと，参照・設定される変数を調べるプログラム変換を与える．これによって得られた情報は，プログラマがスケジューリングを指定したり，アルゴリズムを比較する際に有用である．

The Modeling of Parallelism in GHC using Petri Nets

Shunsuke Nakajima　　Haruo Hasegawa

Oki Telecommunication Systems Co., Ltd.　　Oki Electric Industry Co., Ltd.

4-11-22, Shibaura, Minato-ku, Tokyo 108, Japan

This paper presents a method of abstracting the parallelism in FGHC execution. Petri nets are used to model the behavior of programs. In this modeling, a reduction corresponds to a transition, and a goal or a variable to a place. The program translation is defined in order to abstract such Petri nets from given programs. The translated programs report the deleted/added goals and the referred/substituted variables in every reduction. The parallelism represented by Petri nets is useful in case that a programmer decides the scheduling or compares the algorithms.

1 はじめに

並列処理プログラミングを難しくしている原因の一つに, プロセス・スケジューリングの問題がある. 我々は, 並列論理型言語 FGHC (Flat Guarded Horn Clauses)[1, 2] を対象に, この問題に対する支援を目指している. FGHC におけるプロセス (ゴール) は, 細粒度かつ頻繁に通信し合うので, スケジューリングが非常に難しくなっている.

本稿では, 与えられた FGHC プログラムの実行過程からその並列性を抽象する方法を示す. これによって得られた情報は, プログラマがスケジューリングを指定したり, アルゴリズムを比較する際に有用である.

本稿は, 5 つの章から構成されている. 2 章では, ペトリネットによるプログラムの実行過程のモデル化について, その基本的な考え方を説明する. 3 章では, そのようなペトリネットを求めるためのプログラム変換を定義する. 4 章では, 実際の適用例を紹介する. 5 章では, 本稿のまとめを述べ, 今後の課題を示す.

2 実行過程のモデル化

以降の説明を簡単にするため, あらかじめ正規化されたプログラムだけを扱う. プログラム節の正規形は, 次のように定義される [3].

定義 1 プログラム節の正規形とは, 次のような形式の節のことである.

$$H \ :\!\!-\ G \mid U \cup B$$

ここで, H, G および B は単一化を除くアトムの集合であり, $|H| = 1$, $|G|, |B| \geq 0$ である. また, U は単一化の集合

$$v_1{=}t_1, \cdots, v_n{=}t_n \quad (n \geq 0)$$

であり, 次の条件を満たす.

- $v_1, \cdots, v_n$ は相異なる変数である
- 各 v_i は H 中に現れ, $t_1, \cdots, t_n$ や B 中に現れない
- ある t_i が変数ならば, t_i は H 中に現れる

□

定義 1 における H, G, U, B の元をそれぞれヘッド, ガード・ゴール, 出力単一化, ボディ・ゴールと呼ぶことにする (出力単一化をボディ・ゴールに含めないことに注意されたい).

例 1 次の節は正規形になっている.

```
filter(N,[I|A],C) :- I mod N =\= 0 | C=[I|B], filter(N,A,B)
```

□

正規化されたプログラム節 (H :- G | $U \cup B$) を用いてリダクションが行われた時, 実際には次のような処理が行われる.

(1) ヘッド単一化と G 中のゴールを実行するため, ゴール d 中の変数 $r_1, \cdots, r_m$ の値を (具体化を待って) 参照する

(2) ゴール d をカレント・ゴール列から削除する

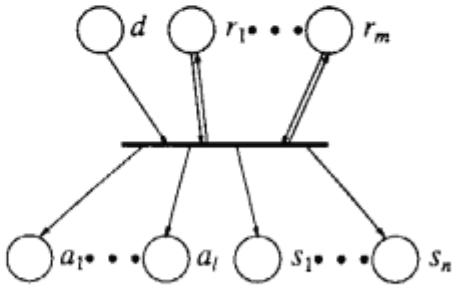
(3) U 中の出力単一化を実行し, d 中の変数 $s_1, \cdots, s_n$ の値を設定する (出力単一化の実行は決して中断しないので, リダクション時に実行されるものと考える)

(4) B 中のゴール $a_1, \cdots, a_l$ をカレント・ゴール列に追加する

このようなリダクションの実行を, ペトリネットを用いて次のようにモデル化する.

定義 2 以下に示すような対応に基づいて, リダクションの実行をトランジションの発火としてモデル化する.

リダクション ⇒ トランジション
削除ゴール d ⇒ 入力プレース
参照変数 $r_1, \cdots, r_m$ ⇒ 入出力プレース
追加ゴール $a_1, \cdots, a_l$ ⇒ 出力プレース
設定変数 $s_1, \cdots, s_n$ ⇒ 出力プレース



□

従って, プログラム実行時の全リダクションについて, 削除・追加されるゴールと参照・設定される変数を調べれば, 実行過程全体をペトリネットでモデル化できる. このようにして得られたペトリネットは, 各リダクションの実行順序に関する制約を表している.

3 プログラム変換の定義

2 つのプログラム変換を定義する. それぞれ変換されたプログラムを実行すると, 削除・追加されるゴール, 参照・設定される変数がリダクション毎にレポートされる.

3.1 削除・追加ゴールのレポート

このプログラム変換は, 正規化されたプログラム節の各々に対して, 次に示す 2 ステップの変換を施すことによって行われる.

(1) ヘッドとボディ・ゴールの各々をその ID 付きアトム (後述) で置き換える

(2) レポート・ストリームに対応する引数をヘッドとボディ・ゴールの各々に加え, レポート・メッセージ (後述) の出力を行う単一化を加える

以下, 各ステップについて説明する.

ステップ 1

プログラムの実行開始から終了までの間に生成されるゴールのひとつひとつを区別するため, ゴール毎にユニークな名前 — ゴール ID を付与する. ID 付きアトムはこの目的で導入される形式であり, 次のように定義される.

定義 3 アトム a に対する ID 付きアトム $\dot{a}$ は,

- $a = p(t_1, \cdots, t_n)$, $(n \geq 0)$ ならば, $\dot{a} = p(v, t_1, \cdots, t_n)$

である. ただし, v は未定義変数である. □

定義 3 において, v が未定義変数であることから, ゴール $\dot{a}$ が起動される度に v に対して新たな変数が生成される. 実行中に生成される変数は処理系によってユニークに区別されるので, この変数がゴール ID の役割を果たす.

ステップ 2

リダクション毎のレポートは削除・追加されるゴールの ID を用いて行う. そこで, メッセージとして次のような形式を用いる.

$$\mathrm{g}(d, [a_1, \cdots, a_l]) \quad (l \geq 0)$$

ここで, d は削除されるゴールの ID, $a_1, \cdots, a_l$ は追加されるゴールの ID である. そのような ID を持つゴールの集合は次のように定義される.

定義 4 正規化されたプログラム節 (H :- G | $U \cup B$) を用いてリダクションが行われる場合について考える. ここで, H, G, U, B はそれぞれヘッド, ガード・ゴール, 出力単一化, ボディ・ゴールの集合であり, ヘッドとボディ・ゴールの形式は ID 付きアトムになっているものとする. このとき, カレント・ゴール列から削除されるゴールの集合は H であり, カレント・ゴール列に追加されるゴールの集合は B である. □

例 2 例 1 の節に対して, ステップ 1 および 2 を適用した結果を次に示す.

```
filter(G,N,[I|A],C,R1,R3) :- I mod N =\= 0 |
        R1=[g(G,[Gf])|R2], C=[I|B], filter(Gf,N,A,B,R2,R3)
```

述語 filter の第 1 引数がゴール ID になっていて, 第 5, 6 引数が差分リストを用いたレポート・ストリームになっている. また, 変数 R1 に対する単一化により, 削除されるゴールに対する ID G と追加されるゴールに対する ID Gf がレポートされる. □

3.2 参照・設定変数のレポート

このプログラム変換は, 正規化されたプログラム節の各々に対して, 次に示す 3 ステップの変換を施すことによって行われる.

(1) ヘッド, 出力単一化およびボディ・ゴール中に現れる項の各々をその ID 付き項 (後述) で置き換える

(2) 同じ変数を値とする ID 付き項に対して同じ ID を付与するための単一化 (後述) をボディに加え, その後, 再び正規化を行う

(3) レポート・ストリームに対応する引数をヘッドとボディ・ゴールの各々に加え, レポート・メッセージ (後述) の出力を行う単一化を加える

以下, 各ステップについて説明する.

ステップ1

プログラムの実行開始から終了までの間に生成される変数や,構成される項のひとつひとつを区別するため,項毎にユニークな名前 — 項 ID を付与する. ID 付き項はこの目的で導入される形式であり,次のように定義される.

定義 5 項 t に対する ID 付き項 $\dot{t}$ は,

- t が変数または定数ならば, $\dot{t} = (v,t)$
- $t = f(t_1,\cdots,t_n)$, $(n \geq 1)$ かつ各 $\dot{t_i} = (v_i,t_i)$ ならば, $\dot{t} = (\{v,v_1,\cdots,v_n\},t)$

である. ただし, v は未定義変数である. □

ID 付き項 (v,t) または $(\{v,v_1,\cdots,v_n\},t)$, $(n \geq 1)$ に対して, v をその ID, t をその値と呼ぶことにする. また, 定義5 によると, t の ID 付き項 $\dot{t}$ の値が t 自身になっている点に注目されたい. (詳しい説明は省略するが) この形式は, ヘッドの引数をその ID 付き項で置き換えても,デッドロックすることがないように考慮されたものになっている.

ここで, ステップ 2, 3 の説明に備え, いくつかの定義を行っておく.

定義 6 A をすべての引数が ID 付き項になっているアトムの集合とする. このとき, A 中に現れる ID 付き項の集合 $Term(A)$ は, 次の条件を満たす最小の集合である.

- $p(t_1,\cdots,t_n) \in A$, $(n \geq 0)$ ならば, $t_1,\cdots,t_n \in Term(A)$
- $(\{v,v_1,\cdots,v_n\},f(t_1,\cdots,t_n)) \in Term(A)$, $(n \geq 1)$ ならば, $(v_1,t_1),\cdots,(v_n,t_n) \in Term(A)$

□

定義 7 A をアトムの集合とする (引数が ID 付き項になっていてもよい). このとき, A 中に現れる変数の集合を $Var(A)$ で表す. □

ステップ2

リダクションの実行により,親ゴール中の変数や新たに生成された変数は,いくつかの子ゴールに引き継がれていく. ちょうどそれに対応するように,同じ変数を値とする ID 付き項の ID も子ゴールに引き継がれなければならない. そのために必要な項 ID の単一化は,次のように定義される.

定義 8 正規化されたプログラム節 $(H$:- G | $U \cup B)$ について考える. ここで, H, G, U, B はそれぞれヘッド, ガード・ゴール, 出力単一化, ボディ・ゴールの集合であり, ヘッド, 出力単一化およびボディ・ゴール中のすべての引数は ID 付き項になっているものとする. このとき, 同じ変数を値とする ID 付き項に同じ ID を付与するための単一化は,

$$\{v{=}w \mid (v,t),(w,t) \in Term(H \cup U \cup B) \text{ かつ } t \in Var(H \cup U \cup B)\}$$

である. □

ステップ3

リダクション毎のレポートは参照・設定される変数のIDを用いて行う．そこで，メッセージとして次のような形式を用いる．

$$\mathrm{v}([r_1,\cdots,r_m],[s_1,\cdots,s_n]) \quad (m,n \geq 0)$$

ここで，$r_1,\cdots,r_m$ は参照される変数のID，$s_1,\cdots,s_n$ は設定される変数のIDである．そのようなIDを持つ変数の集合は次のように定義される．

定義 9 正規化されたプログラム節 $(H$:- G | $U \cup B)$ を用いてリダクションが行われる場合について考える．ここで，H, G, U, B はそれぞれヘッド，ガード・ゴール，出力単一化，ボディ・ゴールの集合であり，ヘッド，出力単一化およびボディ・ゴール中のすべての引数はID付き項になっているものとする．このとき，値が参照される変数の集合は，

$$\{(v,t) \in Term(H) \mid t \notin Var(H) \text{ または } t \in Var(G)\}$$

であり，値が設定される変数の集合は，

$$\{(v,t) \in Term(U \cup B) \mid t \notin Var(U \cup B) \text{ または } t \in Var(G)\}$$

である．□

例 3 例1の節に対して，ステップ1から3を適用した結果を次に示す．

```
filter((TN,N),({Tn1,TI,TA},[I|A]),(TC,C),R1,R3) :- I mod N =\= 0 |
        R1=[v([Tn1,TI,TN],[Tn2])|R2], (TC,C)=({Tn2,TI,TB},[I|B]),
        filter((TN,N),(TA,A),(TB,B),R2,R3).
```

述語 filter の第1-3引数がID付き項で置き換えられており，第4,5引数が差分リストを用いたレポート・ストリームになっている．変数R1に対する単一化により，参照される変数に対するID Tn1, TI, TNと，設定される変数に対するID Tn2がレポートされる．□

4 実際の適用例

実際には，削除・追加ゴールと参照・設定変数の両方を同時にレポートするような変換を行う．このとき，レポート・メッセージとして，次のような形式を用いる．

$$\mathrm{r}(d,[a_1,\cdots,a_l],[r_1,\cdots,r_m],[s_1,\cdots,s_n]) \quad (l,m,n \geq 0)$$

ここで，d はカレント・ゴール列から削除されるゴールのID，$a_1,\cdots,a_l$ はカレント・ゴール列に追加されるゴールのIDである．また，$r_1,\cdots,r_m$ は値が参照される変数のID，$s_1,\cdots,s_n$ は値が設定される変数のIDである．

変換例

素数を生成するプログラムを次に示す. 起動ゴールとなる述語 primes は, 第1引数に正の整数 n を指定すると, 第2引数に n 以下の素数のリストを返す.

```
primes(N,B) :- true | gen(1,N,A), sift(A,B).

gen(I,N,A) :- I>=N | A=[].
gen(I,N,A) :- I<N, J:=I+1 | A=[J|B], gen(J,N,B).

sift([],A) :- true | A=[].
sift([I|A],D) :- true | D=[I|C], filter(I,A,B), sift(B,C).

filter(N,[],A) :- true | A=[].
filter(N,[I|A],B) :- I mod N =:= 0 | filter(N,A,B).
filter(N,[I|A],C) :- I mod N =\= 0 | C=[I|B], filter(N,A,B).
```

素数生成プログラムに対する変換結果を次に示す (紙幅の都合により, 述語 sift の第2節と述語 filter の第3節のみを示す).

```
sift(G, ({Tn1,TI,TA},[I|A]), (TD,D), R1,R4) :- true |
    R1=[ r(G,[Gf,Gs],[Tn1],[Tn2]) |R2],
    (TD,D) = ({Tn2,TI,TC},[I|C]),
    filter(Gf, (TI,I), (TA,A), (TB,B), R2,R3),
    sift(Gs, (TB,B), (TC,C), R3,R4).

filter(G, (TN,N), ({Tn1,TI,TA},[I|A]), (TC,C), R1,R3) :-
    I mod N =\= 0 |
    R1=[ r(G,[Gf],[Tn1,TI,TN],[Tn2]) |R2],
    (TC,C) = ({Tn2,TI,TB},[I|B]),
    filter(Gf, (TN,N), (TA,A), (TB,B), R2,R3).
```

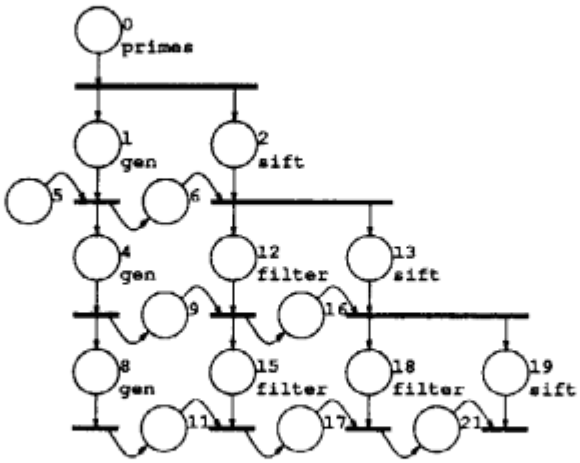
実行例

変換後のプログラムの実行結果を次に示す. ただし, ID は $\_i$, (i は整数) という形式で表示されている. また, レポート・メッセージの各々について, その時にリダクションされたゴールをコメントとして記した.

```
| ?- primes(G, (T3,3), (TA,A), R,[]).
R = [ r( _0,[ _1, _2],[          ],[     _3]),  % primes(3,[2,3])
      r( _1,[     _4],[     _3, _5],[ _6, _7]),  % gen(1,3,[2,3])
      r( _4,[     _8],[     _7, _5],[ _9,_10]),  % gen(2,3,[3])
      r( _8,[       ],[    _10, _5],[    _11]),  % gen(3,3,[])
      r( _2,[_12,_13],[         _6],[    _14]),  % sift([2,3],[2,3])
      r(_12,[    _15],[ _9,_10, _7],[    _16]),  % filter(2,[3],[3])
      r(_15,[       ],[        _11],[    _17]),  % filter(2,[],[])
      r(_13,[_18,_19],[        _16],[    _20]),  % sift([3],[3])
      r(_18,[       ],[        _17],[    _21]),  % filter(3,[],[])
      r(_19,[       ],[        _21],[    _22]) ] % sift([],[])
```

並列性の抽象

得られたレポート・メッセージに対応するペトリネットを次に示す. (ただし, リダクション順序に影響しない冗長なプレースとアークは省略した).



5 おわりに

本稿では, 与えられたFGHCプログラムの実行過程からその並列性を抽象する方法を示した. リダクションをトランジションに, ゴールと変数をプレースに対応させることで, プログラムの実行過程をペトリネットでモデル化した. 実際にそのようなペトリネットを求めるため, リダクション時に削除・追加されるゴールと, 参照・設定される変数を調べるプログラム変換を与えた.

しかし, 本稿で示した方法は完全なものではなく, 次のようなプログラムに対しては, リダクション順序に関する全ての制約が求まらない.

- ヘッドに同じ変数が複数回現れる節がある
- ヘッド中の変数どうしの出力単一化が行われる

前者はプログラム変換に関する問題であり, 後者はモデル化のしかたに関わる問題である. 現在, これらの問題について検討を続けている.

謝辞 本研究は第五世代コンピュータ・プロジェクトの一環として行っている. 日頃御指導頂くICOT研究部長代理長谷川隆三氏に感謝します.

参考文献

[1] Ueda, K.: Guarded Horn Clauses: Tech. Report TR-103, ICOT, 1985.

[2] 淵 一博 (監修), 古川康一, 溝口文雄 (共編): 並列論理型言語 GHC とその応用, 共立出版, 1987.

[3] Ueda, K., Furukawa, K.: Transformation Rules for GHC programs, Proc. of FGCS'88, ICOT, 1988.

[4] Peterson, J. L.: Petri Net Theory and the Modeling of Systems, Prentice-Hall, Inc., 1981.