GHCによる時制論理の証明系の作成

6D-5

高橋和子　金森　直

(株)三菱電機　中央研究所

1. はじめに

Guarded Horn Clauses(GHC)[1]は第5世代プロジェクトの核言語として開発された並列論理型言語である。GHCのプログラムは以下の形をした'|'(ガード)を持つHorn節の集合である。

H :- G1,...,Gn | B1,...,Bm.

我々は命題時制論理の証明系をGHCで作成した。本稿では、並列プログラミングの考え方を中心にその結果を報告する。

2. 命題時制論理

時制論理[2]は通常の一階述語論理を時間の概念を陽に加えることによって拡張したものである。本稿では命題論理に3つの時制オペレータを加えた命題時制論理を考える。各オペレータの持つ意味は以下の通りである。

$\Box P$(always P) : 現在以後Pはずっと真である。

$\Diamond P$(eventually P) : 将来Pが真になる時点がある。

$\circ P$(next P) : 次の時点でPは真である。

例えば $\Diamond\Box\neg P$はある時点以後 $\neg P$ がずっと成立することを、$\Box\Diamond P$ はP が無限回成立することを表す。

3. ω グラフ法

ωグラフは節式と呼ばれる式で各節点がラベル付けされた有向グラフである。命題時制論理の式とωグラフは1対1に対応しており、式F のωグラフにおいて'ωループ'を含む径路はF のモデルに対応する。

ωグラフ法は与えられた命題時制論理の式の恒真性を判定する方法であり、その手続きは以下のとおりである。

(1) 与式を否定する。これをF_0とする。

(2) F_0に対する初期節式を計算する。

(3) 初期節式から始めて節式の展開を繰り返し、ωグラフを生成する。

(4) 生成したωグラフ上で'ωループ' の存在を調べる。存在しなければその時のみ与式は恒真である。

この手続きは決定的であり有限回で終了する。

以下で各段階における並列プログラミングについて述べる。

3.1. 初期節式の計算

まず、与式を否定した式に含まれる $\supset, \equiv$ を除去し、$\neg$を最内側へ移動することにより否定標準形に変換する。初期節式は否定標準形Fに空集合を伴なった形$[F]_{\{\}}$ で表される。$\neg\Diamond\Box\neg P$の初期節式は $[\Box\Diamond P]_{\{\}}$ である。この変換において恒真性は保たれる。更に否定標準形に含まれる$\Diamond G$ の形をした部分式の集合(Eventuality Setと呼ぶ) を求める。これは後にωグラフの節点をタイプ分けするのに使う。これらの処理のアルゴリズムは再帰型の要素を多く含む。例えば remove_implication_and_equivalence のプログラムを考えてみる。

```
negation_normal_form(F,NNF) :-
    remove_implication_and_equivalence(F,G),
    move_not_inwards(G,NNF).
remove_implication_and_equivalence(imply(F,G),A) :-
    remove_implication_and_equivalence(F,F1),
    remove_implication_and_equivalence(G,G1),
    A=or(not(F1),G1).
```

remove_implication_and_equivalence の最初の2つのプロセスはそれぞれF, Gに対する計算を行ない、3番目のプロセスは同時に共有変数Aを通じて move_not_inwards に値を伝播する。ここで他のプロセスが計算を終えていなくても move_not_inwards のhead unification が成功することに注意する。

3.2. ωグラフの生成

ωグラフの各頂点は節式でラベル付けされている。節式は$[F]_H$と表される。但しF は否定標準形の式、Hはh-setと呼ばれる集合であり、$[F]_H$は論理的には Fと等価である。H-setはeventuality($\Diamond$) の実現を保証するために導入されるものであり、eventuality の実現の履歴を表すリストであるが、ここでは詳しい説明は省略する。

F_0を与式を否定した式の否定標準形とする。F_0のωグラフは次のように生成される。

(1) F_0の初期節式に対応する初期節点を作る。

(2) 各節点(Nとする) に対し、対応する節式$[F]_H$を展開する。その結果$[F_1]_{H_1}, ..., [F_m]_{H_m}$が生成されたとする。

(3) 任意の iに対し、$[F_i]_{H_i}$に対応する節点が存在すればNからそれに対して孤を描き、そうでなければ新たに節点を作ってNから孤を描く。

この手続きを繰り返し、すべての節点が展開されればグラフの生成は終了する。

Proof Procedure of Propositional Temporal Logic in GHC

Kazuko TAKAHASHI and Tadashi KANAMORI

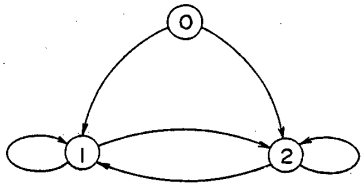
MITSUBISHI Electric Corp.

ここで$[F]_H$の展開というのは、F が現在成立している時、次の時点において成立する可能性のある式の集合を生成する手続きであり、tableau method[3]に基づいて行なわれる。この手続きはPrologでは繰り返し型で自然に記述される。

並列プログラミングにおいては各節点が一つのプロセスに相当し、節点ごとに展開が並列に行なわれているものと考える。この時、異なるプロセスが出力として同じ節式を含むとすると、それが既存のものかどうか判定できない。従って出力をマージする必要がある。以上の考察から、GHCによるωグラフの作成は以下の3つの部分プロセスから成ることがわかる。

(1) graph-manager: node-process の生成・消滅を管理し、また現在の節式のリストを格納してmultiplexerから受け取った式が既存のものか否かを調べる。
(2) multiplexer : すべてのnode-processの出力をマージしてgraph_managerに送る。
(3) node-process : 各節式の展開をする。

$\Box\Diamond P$ のωグラフは以下のようになる。



3.3. ωループの検出

与式を否定した式F_0の Eventuality Setを ES0とする。節式$[F]_H$に対応する節点で H=ES0 のものをωノードと呼び、ωノードから自分自身に還るループをωループと呼ぶ。ωループが存在しなければもとの式は恒真であり存在すれば恒真でない。

ωループの検出は Prolog ではバックトラックのメカニズムを利用して記述されるが、GHCではバックトラックがないため、根本的にアルゴリズムを変更する必要がある。ここでは各節点を入出力チャネルを持つプロセスと見なしグラフをそのネットワークと考える。グラフは以下の4項組のリストで表される。

(NodeNmbr,OutStrm,InStrms,NodeType)

ここで、NodeNmbr は各節点につけられた番号、OutStrm は対応する出力チャネル、InStrms は対応する入力チャネルの集合、NodeType はωノードか否かを表す変数である。例えば$\Box\Diamond P$のωグラフは以下のように表現される。

```
[ (0, X0, [ ],          not-omega),
  (1, X1, [X0,X1,X2], omega),
  (2, X2, [X0,X1,X2], not-omega) ]
```

各ノードは最初に自分のノード番号を出力チャネルに流した後、入力チャネルの集合を出力チャネルにつなぐ。このことによってその節点がつながっているプロセスのネットワークの構造そのものを情報として与える。各ωノードは自分の情報を調べて自分自身につながっているか否かを判定する。また停止フラグとして働く2つの変数 JudgeStopと EndMsgsを導入する。JudgeStopはプロセス間の共有変数であり、あるプロセスがωループを発見すると、stopにinstantiateされて他のすべてのプロセスを止める停止フラグである。EndMsgsからは各プロセスにスロットが切り取られ、各プロセスはチェックを終えるとそれぞれのスロットにendを代入する。すべてのスロットにendが代入されれば ωループなし(ω-loop free)と判定される。

3.4. トップレベルにおける並列化

以下にωグラフ法のトップレベルのインプリメンテーションを示す。ここでconstruct_omega_graph と check_omega_loop_freeness が2つの共有変数 JudgeStop とGraph を持つことに着目する。refute の3つの部分プロセスは並列に走っているので、Graphには部分的に決定されたデータが入ることになる。このデータでcheck_omega_loop_freenessがωループを発見して、JudgeStopをstopに instantiateするとconstruct_omega_graph はその時点で停止し、それ以上グラフの生成を続ける必要はない。

```
prove(F) :- refute(not(F),A), write_answer(F,A).
refute(F,A) :-
    compute_initial_node_formula(F,F0,ES0),
    construct_omega_graphs(JudgeStop,
                           ES0,(F0,[ ]),Graph),
    check_omega_loop_freeness(JudgeStop,Graph,A).
```

4. おわりに

GHCによるωグラフ法の作成について述べた。全体のプログラムは入出力インタフェースを除き約500行になった。今後はこの経験を生かして GHCのプログラミング方法論を発展させたい。

5. 謝 辞

本研究は第5世代計算機プロジェクトの一環として行なわれたものである。このような研究の機会を与えて頂いたICOTの淵一博所長、適切な助言を頂いた古川康一同第1研究室長に感謝致します。

参考文献

[1] Ueda,K., "Guarded Horn Clauses," ICOT TR-103, 1985.
[2] Manna,Z. and A.Pnueli, "Verification of Concurrent Programs, Part1: The Temporal Framework," Stanford TR 81-836,1981.
[3] Wolper,P.L., "Temporal Logic Can Be More Expressive," Proc.22nd IEEE Symposium on Foundation of Computer Science, pp.340-348,1981.
[4] Takahashi,K. and T.Kanamori, "On Parallel Programming Methodology in GHC," Proc. of The Logic Programming Conference'86, pp.135-142,1986.