解　説

IPS

ネット指向パラダイムを求めて

2. ペトリネットによる並行処理プログラムの解析手法†

村田忠夫††　辻　孝吉†††

1. はじめに

ペトリネット (Petri net) は，多くのシステムに適用可能なグラフィックで数学的なモデル化ツールである．並行的 (concurrent)・非同期的 (asynchronous)・分散的 (distributed)・並列的 (parallel)・非決定的 (nondeterministic)・確率的 (stochastic) な動作を特徴とする情報処理システムを，記述・解析するツールとして有力なものである．グラフィックなツールとして，フローチャートやブロックダイヤグラム・ネットワークと同じように，システム構造の可視的な表現手段として，ペトリネットを使用することができる．加えて，ペトリネットの中でトークン (token) を使用することにより，システムの並行的でダイナミックな事象をシミュレートすることができる．一方，数学的なツールとしては，システムの挙動を表現する状態方程式や代数方程式その他の数学モデルを立てることが可能である．すなわち，ペトリネットの特徴は上記の三つのツール（グラフィック視覚的ツール，シミュレーションツール，及び数学的なツール）の機能を同時にもつ，いわゆる“三位一体”の一般的性質をもつことである．したがって，実務に携わる技術者と理論研究者との双方でペトリネットを利用することができる．

具体的にペトリネットで解析できる性質には，可達性 (Reachability)，有界性 (Boundedness)，活性 (Liveness)，可逆性 (Reversibility)，被覆性 (Coverability)，同期距離 (Synchronic Distance)，公平性 (Fairness) などがある．これらの性質を調べることでシステムの構造的性質だけでなく，動的性質を調べることができる[1),2)]．ペトリネットのこれらの性質の解析法は次の4つのグループに分けることができる．1)被覆（可達）木法 (coverability (reachability) tree method)，2)行列方程式法 (matrix equation approach)，3)縮約あるいは分割手法 (reduction or decomposition techniques)，4)ネットの部分構造（Sコンポーネント，ハンドル，ブリッジなど）を調べる手法．最初の方法は，本質的にすべての可達マーキングあるいは被覆マーキングを数え上げるものである．これはネットの全クラスに適用可能だが，状態空間発散 (state-space explosion) の問題のため「小規模な」ネットに限定される．ほかの三つの手法は強力であるが，多くの場合，ペトリネットの特定のサブクラスあるいは特殊な条件のもとでのみ適用可能である．したがって，これらの手法を並行処理プログラム解析のような実用的な応用に適用する場合，解析（アルゴリズム）の最後の「締めくくり」として可達木的手法あるいはシミュレーションに頼る必要がある．

本稿では，Adaプログラムを例に，そのデッドロックの検出にペトリネットを用いる最近の方法を紹介する．一般にそれらの解析の手数は，ネットの規模に対して指数的に増大する．したがって，与えられたネットをそのまま扱うことは，小規模なネットでないかぎり実用上不可能である．そこで，このような問題に対処するために今まで用いられてきた三つの方法（階層的可達グラフ，ネット縮約，インバリアント）のうち，二つについて解説を行う．これにより，並行処理ソフトウェアへペトリネットを応用することの有用性を示すとともに，ペトリネットの一般のシステムへの応用の可能性を示す．

本稿では，ペトリネットに関する定義などはなるべく文献1)，2)と同じものを用いる．また，煩

† Static Analysis of Concurrent Programs by Petri Nets by Tadao MURATA (Univ. of Illinois at Chicago) and Koukichi TSUJI (Department of Electrical and Electronics Engineering, Fukui University).

†† イリノイ大学シカゴ校
††† 福井大学工学部

雑さを避けるため厳密な定義などを省略している場合があるので，詳しく知りたい場合には，該当論文を参照されたい．

2. ペトリネットの階層的可達グラフ

2.1 概　要

本節では，第1番目の方法である“可達グラフを階層的に用いて可達性やプログラムのデッドロックを解析する方法”を，Adaプログラムの例を用いて説明する．この方法では，全体のペトリネットはお互いにトランジションを通じて通信あるいは同期し合う部分ネットの集まりでできていると仮定する．したがって，まず，各部分ネットの可達グラフ (R-graph) を求め（階層表現の最下位レベル），次に，各可達グラフを縮約し，数個（普通2または3個）ごとに結合し，階層表現の上位レベルの可達グラフを得る．このような，縮約と結合を全体のペトリネットを意味する最上位レベルまで繰り返すという方法である[3]．この方法は，Adaプログラムのペトリネットモデルに限らず，有界ペトリネットでモデル化可能なすべての場合に適用可能であることに注意されたい．

まず，可達グラフの結合の例として**図-1**のように二つの可達グラフ RG1 と RG2 を考える．このとき，トランジション t1′ が共通のトランジション（同期トランジション：Synchronizing transition）である．ここで，二つのグラフを t1′ で結合させること (G-Combine) ができる．図-1から，トランジション t1′ を発火させると RG1 では状態 a1 から状態 a3 に変化し，RG2 では状態 b1 から状態 b2 に変化する．したがって，RG1 と RG2 を結合した可達グラフ (RG1, RG2) では**図-2**に示すように状態 (a1, b1) から状態 (a3, b2) へ変化する．ほかのトランジションが発火した場合には，RG1 か RG2 のいずれか一方の状態だけが変化する．このことを考慮して結合された可達グラフ (RG1, RG2) をつくると図-2のようになる．

次に，可達グラフの縮約の例として**図-3**に示す可達グラフ RG を考える．ここで，ほかの可達グラフと同期している同期トランジションは t2 と t5 の二つのみであるとする．このとき，t1 を発火すれば，同期トランジション t2 と t5 を発火させることなく v1 から v2 へ到達する．したがって，状態 v1 と v2 は同期トランジション t2 と t5 を発火させることなく可達であるという意味において同値である．同様に，t3 と t4 を発火すれば，発火の順序によって通過する状態は異なるものの v3 から v6 へ同期トランジション t2 と t5 を発火させることなく到達することが可能である．したがって，4つの状態 v3, v4, v5, v6 は同値である．そこで，v1, v2 を v1′ と

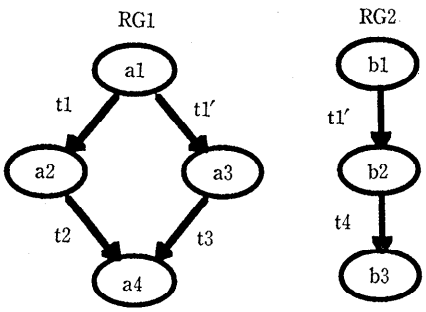


図-1 可達グラフ RG1 と RG2

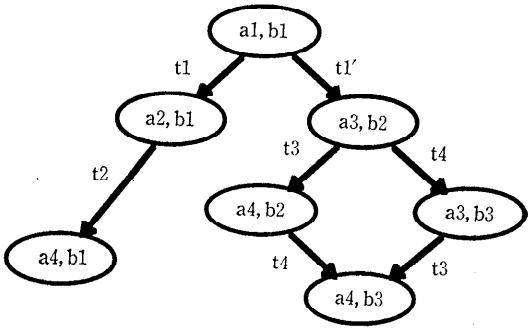


図-2 結合された可達グラフ (RG1, RG2)

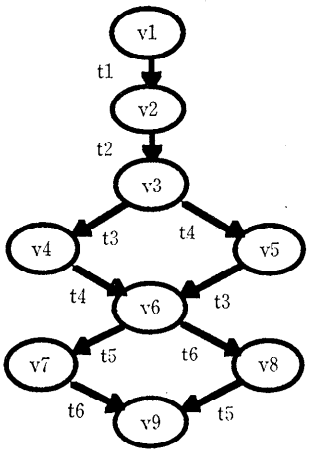


図-3 可達グラフ RG

し，v3，v4，v5，v6 を v2′，v7 を v3′，v8 を v4′，v9 を v5′ とみなせば，**図-4** のように縮約された可達グラフが得られる (G-Compress). このような結合と縮約を繰り返すと最後に一つの可達グラフになる，この最後の可達グラフを根可達グラフ (Root RG) と呼ぶ．根可達グラフは階層表現の木の最上位レベル，根に相当し，与えられた各部分ネットの可達グラフは階層表現の木の最下位レベル，葉に相当する．ここで，根可達グラフは縮約しないものとする．根可達グラフで可達であることが，与えられたペトリネットで可達であることの必要十分条件になっている[3]．さらに，この階層的可達グラフを用いてデッドロックに対する解析も行える．上記の各可達グラフにおいて出力枝をもたない節点 (deadlock vertex) は，一般にプログラムのデッドロックの状態かプログラムの完了を意味するが，ループが縮約された状態を含んでいることがあるので注意すべきである．たとえば，**図-5** において，グラフRをグラフR′に縮約しても，デッドロックに関する情報を失っていない．しかし，**図-6** の場合，t3 を発火しない限り，t2 と t4 を発火し続けることができる．したがって，このような節点は，ほかの節点と区別しなければならない．ペトリネットがデッドロックをもつための必要十分条件は，根可達グラフに deadlock vertex があり，それが下位レベルでループを縮約した節点に相当していないことである．その他の詳しいことは，文献 3) を参照されたい．

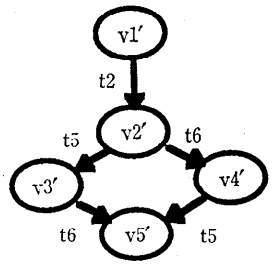


図-4 縮約された可達グラフ

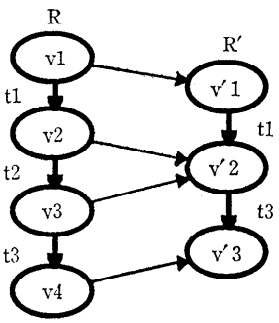


図-5 デッドロックに関する情報を失わない縮約

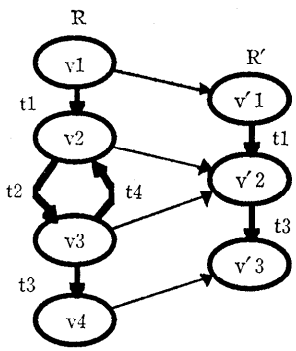


図-6 デッドロックに関する情報を失い得る縮約

```
task body T1 is
begin
    T2.E;
end T1;

task body T2 is
begin
    accept E do
    ...
    end E;
    T3.F;
end T2;

task body T3 is
begin
    accept G do
    ...
    end G;
    accept F do
    ...
    end F;
end T3;

task body T4 is
begin
    T3.G;
end T4;
```

図-7 4つのタスクからなる Ada プログラム例

2.2 階層的可達グラフの応用例

例 1：**図-7** に示す4つのタスクからなる Ada プログラムのペトリネットモデルは**図-8** に示す4つの部分ネット T1，T2，T3，T4 から構成されている．ここで，たとえばタスク T1 とタスク T2 とのランデブは二つの同期トランジション t5 と t6 で表されている．同様にして，ほかの二つのランデブは同期トランジションの二つの組 {t12, t13} と {t10, t11} で表されている．このような同期通信パターンから，**図-9** の木で示すように，T1 と T2 を結合し，T3 と T4 を結合してから，T1+T2 と T3+T4 を結合する．(一般にどのような順序で結合するのが最適であるかという問題は興味深いが未解決である．) まず**図-10** に示すように部分ネット T1 の可達グラフ R_1 とその縮約可達グラフ R_1' を求める．ここで，R_1 の三つの状態 (マーキング) (1)，(2)，(3,

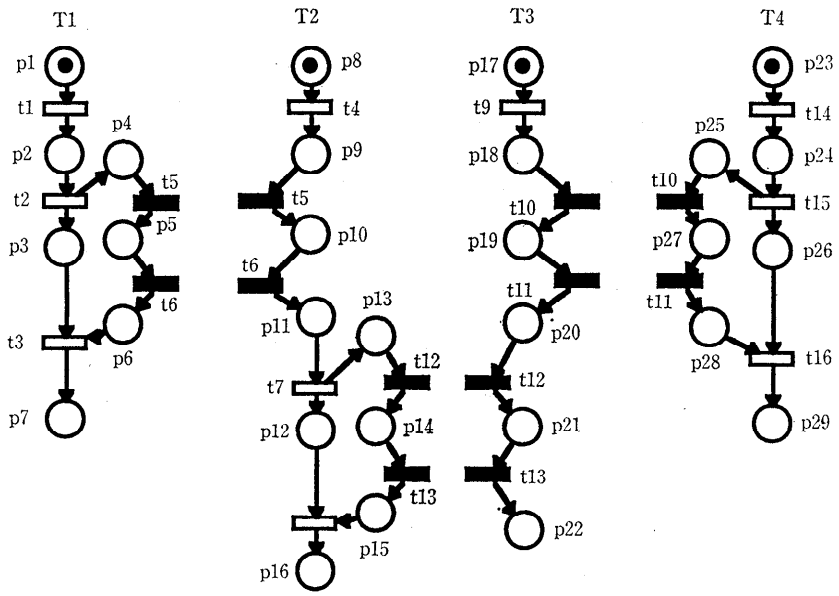


図-8　4つのタスクからなる Ada プログラムのペトリネット

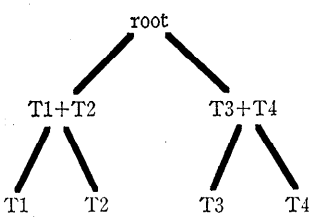


図-9　部分可達グラフの結合順序を示す木

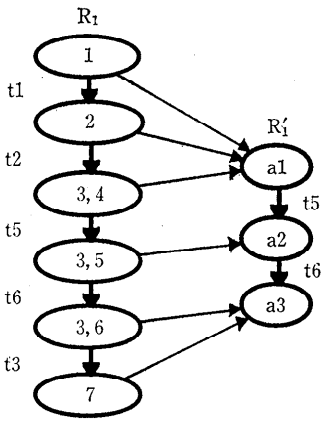


図-10　T1 の可達グラフ R_1 とその縮約可達グラフ R_1'

4) は同期トランジション t5 と t6 に関係なく可達であるから R_1' で一つの状態 a1 に縮約され，R_1 の二つの状態 (3,6)，(7) は R_1' の一つの状態 a3 に縮約される．R_1' では同期トランジション t5 と t6 のみが残っている．同様にして，**図-11**，**図-12**，**図-13** に示すように部分ネット T2，T3，T4 の可達グラフ R_2，R_3，R_4 とその縮約可達グラフ R_2'，R_3'，R_4' を求める．次に，T1 と T2 の縮約可達グラフ R_1' と R_2' を結合すると**図-14** に示す T1+T2 の可達グラフ R_5 が得られ，これを縮約すると R_5' が求まる．ここで，部分ネット T1 と T2 を結合した場合，t5 と t6 は結合ネット T1+T2 の内部トラン

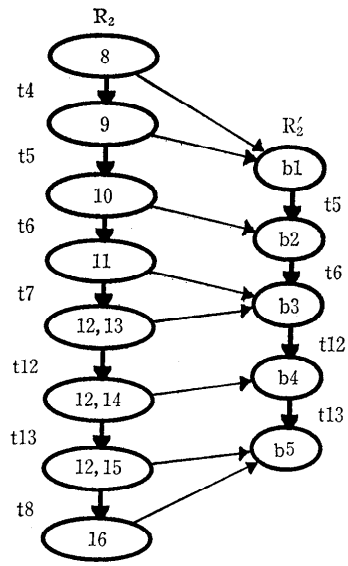


図-11　T2 の可達グラフ R_2 とその縮約可達グラフ R_2'

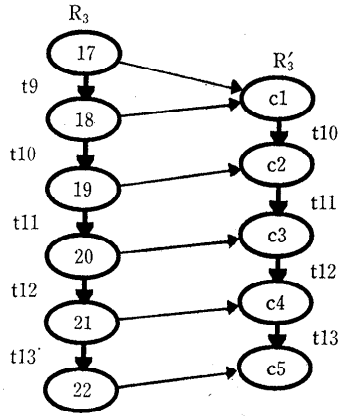


図-12 T3 の可達グラフ R_3 とその縮約可達グラフ R_3'

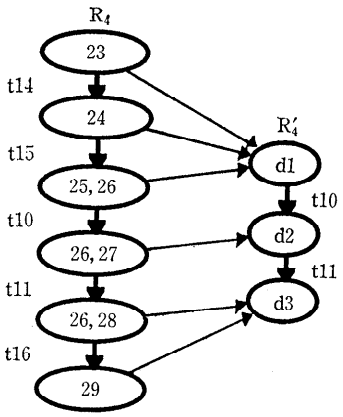


図-13 T4 の可達グラフ R_4 とその縮約可達グラフ R_4'

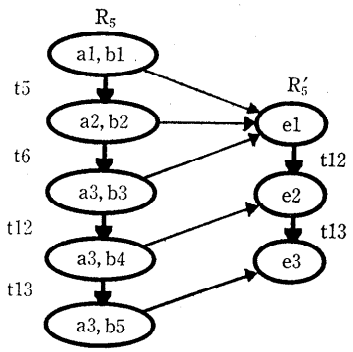


図-14 T1+T2 の可達グラフ R_5 ($R_1'+R_2'$) とその縮約可達グラフ R_5'

ジションとなり，t12 と t13 のみが外部との同期トランジションとなる，図-14 に示すように T1+T2 の縮約可達グラフ R_5' では同期トランジション t12 と t13 のみが残っている．(一般に，図-4 の t6 のように同期トランジション以外のトランジションも縮約後残る場合があること

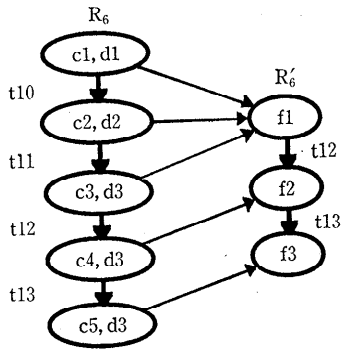


図-15 T3+T4 の可達グラフ R_6 ($R_3'+R_4'$) とその縮約可達グラフ R_6'

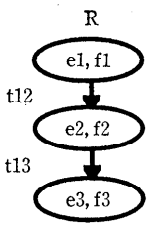


図-16 根可達グラフ R ($R_5'+R_6'$)

に注意.)同様にして，**図-15** に示すように部分ネット T3 と T4 を結合したネットの可達グラフ R_6 とその縮約可達グラフ R_6' が得られる．最後に，T1+T2 の縮約可達グラフ R_5' と T3+T4 の縮約可達グラフ R_6' とを結合すると**図-16** に示す根可達グラフRが得られる．

可達性の解析の例として，目標マーキング (7, 16, 22, 29) が初期マーキング (1, 8, 17, 23) から可達であるかどうか判定するには，根可達グラフRに目標マーキングの「対応マーキング」があるかどうか調べるだけでよい．上記の例では，目標マーキング (7, 16, 22, 29) の対応マーキング (e3, f3) が存在するから可達である．ここで，(e3, f3) が (7, 16, 22, 29) の対応マーキングであることは，((7=a3, 16=b5), (22=c5, 29=d3)) ⇔ (e3, f3) の対応 (⇔) から明らかである．

プログラムのデッドロックを検出する場合はまず根可達グラフ R に出力枝をもたない節点 (deadlock vertex) があるかどうかを判定する．なければプログラムにデッドロックがないし，あれば根可達グラフから階層表現の下位レベルの可達グラフにおいてループで縮約した節点に対応してないかどうかを調べる．上記の例では根可達グラ

```
task body Customer is
  begin
    loop                          c 1→
      Operator. Prepay            t 1→c 2→t 4→c 3→
      Pump. Start                 t 5→c 4→t 6→c 5→
      Pump. Finish                t 7→c 6→t 10→c 7→
      accept Change               t 9→c 8→t 12→
    end loop                      →c 1
  end Customer

task body Pump is
  begin
    loop                          p 1→
      accept Activate             t 2→p 2→t 3→p 3→
      accept Start                t 5→p 4→t 6→p 5→
      accept Finish do            t 7→p 6→
        Operator. Charge          t 8→p 7→t 11→p 8→
      end Finish                  t 10→
    end loop                      →p 1
  end Pump

task body Operator is
  begin
    loop
      select                      o 1→(t 1 or t 8)
        accept Prepay do          t 1→o 2→
          Pump. Activate          t 2→o 3→t 3→o 4→
        end Prepay                t 4→o 1
      or
        accept Charge do          t 8→o 5→
          Customer. Change        t 9→o 6→t 12→o 7→
        end Charge                t 11→
      end select                  →o 1
    end loop
  end Operator
```

図-17　自動ガソリン・スタンドの Ada プログラム（左側）とそのペトリネットモデル（図-18）との対応（右側）

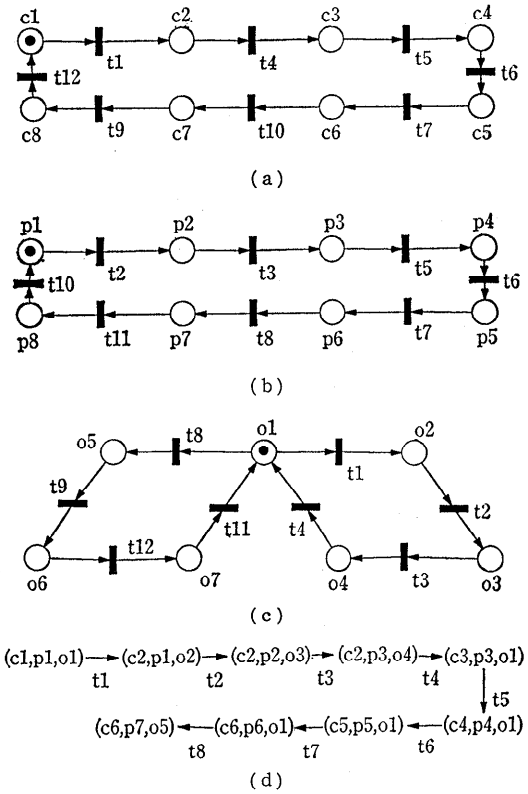


図-18　自動ガソリン・スタンドの Ada プログラムのペトリネットモデル
(a)カスタマ　(b)ポンプ　(c)オペレータ
(d)根可達グラフ

フRに deadlock vertex (e 3, f 3) が存在するが，これは最下位レベル可達グラフのマーキング (7, 16, 22, 29) に対応しておりプログラムの完了を表している．

例 2：もう一つの例として，Ada タスクプログラム解析の分野ではベンチマークとみなされるほどよく例に使われる自動ガソリン・スタンドのプログラム[4]を考える．自動ガソリン・スタンド Ada プログラムは**図-17** の左側に示すように三つのタスク，すなわちカスタマ (Customer)，ポンプ (Pump)，オペレータ (Operator) からなっている．議論を簡単にするため，カスタマ，オペレータを各一人，ポンプ一台とし，ランデブ通信に必要のないステートメントやパラメータは省略する．まず，カスタマがガソリンを買いにガソリン・スタンドにくるとオペレータに少し余分にお金を前払い (Prepay) する．（アメリカの自動ガソリン・スタンドでは料金不払い防止のためにこのように前払い (Prepay) しなければならないところがある．）次にオペレータはポンプを始動 (Activate) し，カスタマはポンプをスタート (Start) し，終ったらポンプ終了 (Finish) を伝える．ポンプはオペレータに料金 (Charge) を知らせ，オペレータはカスタマに前払いと料金の差をおつり (Change) として渡す．以上の三つのタスク間のおのおのの通信は，Ada プログラムでは図-17 の左側に示すようにエントリ・コールとアクセプトの対によるランデブで行われる．この Ada プログラムでは誤ってオペレータのタスクプログラムの中で Customer. Change と end Charge とのステートメントの順序が逆（オペレータがポンプに料金知らせの通知の受取りの確認をする前にカスタマにおつりを渡すこと）になっているため，デッドロックがある．

図-18 (a), (b), (c) に示す三つのペトリネットは，図-17 に示す三つのタスクをモデル化した

ネットで，通信に不必要なものは皆縮約されている．したがって，すべてのトランジションが同期トランジションである．図-17 の三つのタスクプログラムと図-18 の三つのペトリネットの対応を図-17 の右側に示す．たとえば，トランジション t1 は Operator. Prepay と accept Prepay との同期した（同時）開始を表し，トランジション t4 は Operator. Prepay と accept Prepay との同期した（同時）終了を表す．

図-18 の三つのペトリネットは活性安全な状態機械 (state machine) であるから，これらの可達グラフはトランジションをアークのラベルにしたグラフと同形である．（例1と例2のペトリネットは安全であるが，一般に与えられたペトリネットは安全でなくても有界である限り，階層的可達グラフ法が適用可能であることに注意.）これら三つの部分可達グラフを同時に結合すると，図-18(d) に示す根可達グラフを得る．ここで，状態（c6, p7, o5）は，上記のデッドロックである．すなわち，次のような相互待ち状態を表す．プレース c6 にトークンがあるが t10 を発火するためにはその前に図-18(b) の t11 を発火する必要がある．t11 を発火するためにはその前に図-18(c) の t12 を発火する必要がある．t12 を発火するためにはその前に図-18(c) の t9 を発火する必要がある．t9 を発火するためにはその前に図-18(a) の t10 を発火する必要がある．

3. ネット縮約

本章では，デッドロックを検出するための2番目の方法として，“ペトリネットを縮約する”方法[5]について述べる．

まず，一般のペトリネットに適用できる文献 6) の縮約規則の中で，複雑さ (Complexity) がネットの規模に対して線形的に増加する規則を以下に示す．どの縮約規則も活性（デッドロックの情報）を保存する．すなわち，縮約前に(不)活性であったトランジションは縮約後でも（不）活性であるし，縮約によって活性なトランジションが不活性にならないことが証明されている．また，これらの縮約規則は安全性を保存して適用することが可能であり，Ada プログラムのペトリネットモデルに適用可能であることを指摘しておく．

規則1 Post-Fusion of Transitions
規則2 Pre-Fusion of Transitions
規則3 Serial-Fusion of Transitions
規則4 Parallel Redundant Place

さらに，Ada プログラムのペトリネットモデルは安全であり，Ada プログラムの特徴を利用することにより次の縮約規則が導出されている[5]．以

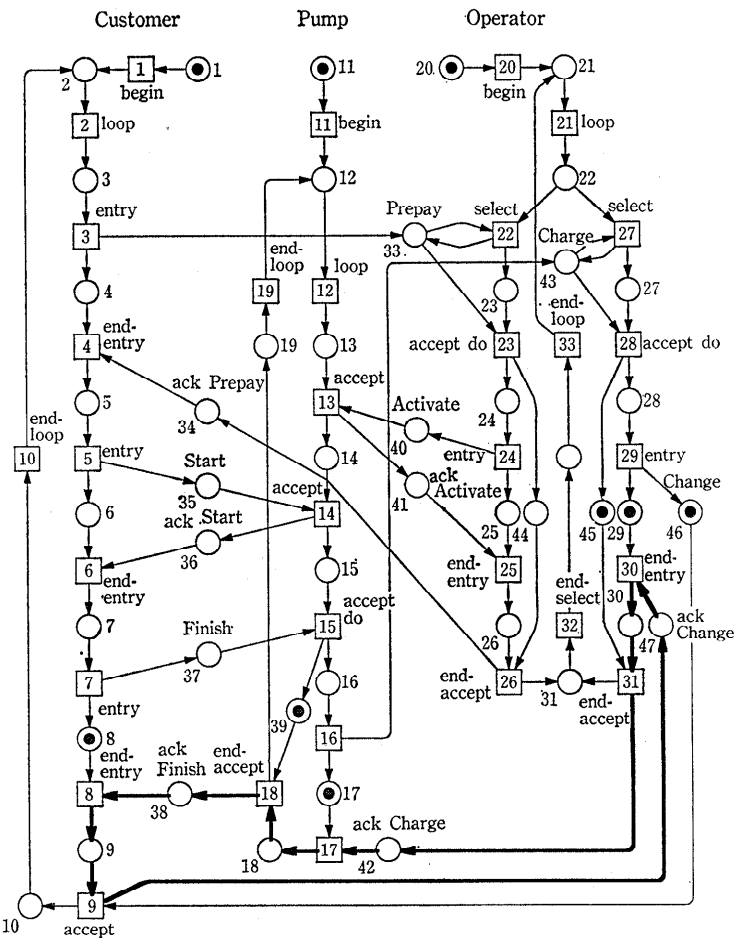


図-19 タスク間の通信もモデル化した詳しい自動ガソリン・スタンド Ada プログラムのペトリネットモデル

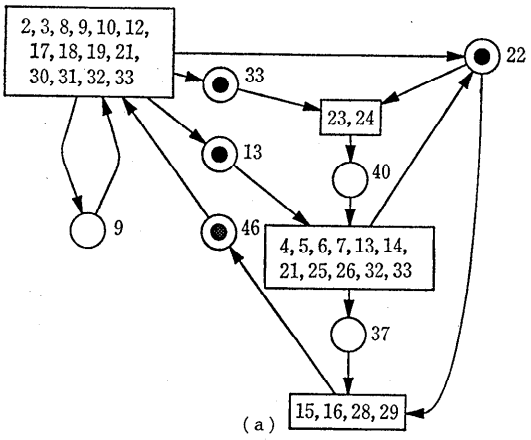


図-20 (a)図-19 のネットを縮約したネットと (b)その可達グラフ

下に示すどの規則もプレースに関する縮約であることを指摘しておく.

規則 5 Removal of Wait-Places
規則 6 Removal of Redundant Accept-Places
規則 7 Removal of Redundant Ex-Places
規則 8 Removal of Reduced Accept-Places
規則 9 Reduction of Selective-Wait Structure
規則10 Removal of Begin-Places

各規則の詳しい説明は，紙数の関係上，文献5) を参照されたい．ここではこれらの縮約規則を適用した例のみ述べる．

例 3: 図-17 の左側に示す自動ガソリン・スタンド Ada プログラムは，文献7)の方法でペトリネットに変換すると**図-19** に示すネットのようになる．このネットはタスク間の通信もモデル化した詳しい自動ガソリン・スタンド Ada プログラムのモデルである．初期マーキング (1, 11, 20) から発火可能な限り発火すると，太線で示した相互待ち状態のデッドロックに到達することが分かる．しかし，このネットの可達グラフの節点は全部で 92 個あり，この 92 個の節点を全部生成することなくデッドロック（出力枝をもたない節点）を検出することはかなり複雑である．このネットを少し前処理 (begin を表すトランジションなどを省略) した後に上記の縮約規則を適用した結果を**図-20**(a)に示す．ここで，長方形は縮約された1個のトランジションを表し，その中の数字は縮約前の複数個のトランジションの番号を示す．この縮約されたネットの可達グラフは**図-20**(b) に示すように (92 個から) 4 個の節点 (状態) に縮約され，デッドロック検出が簡単化されるのが明らかである．上記の縮約規則は計算機上で自動的に適用可能であり，その複雑さは線形かまたは多項式のオーダである[5].

4. むすび

本稿では，並行処理プログラムの一つである Ada プログラムを例にとり，ペトリネットを用いたデッドロック検証について，階層的可達グラフによる解析手法とネット縮約による方法とについて述べた．第3番目の方法である“インバリアントを用いる方法”は，紙数の関係上，割愛した．詳しくは，文献8)を参照されたい．これらの手法の有用性を簡単に説明するために，厳密な定義などは避け，具体的な例題により説明をした．これらペトリネットの解析手法は，並行処理プログラムだけでなく，離散事象システムの範囲に入るものへ広く適用可能であり，応用範囲の広いものである．しかし，ペトリネット理論は完成しているわけではなく，実用に耐えうる解析・設計ツールとして広く一般に使用されるようになるためには，さらに今後の研究が必要である．なお，本稿に関連した並行処理プログラム解析のためのグラフモデルについては，文献9)およびその参考文献を参照されたい．

参考文献

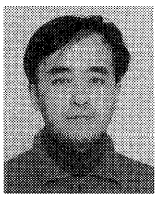
1) Murata, T.: Petri Nets: Properties, Analysis and Applications, Proc. IEEE, Vol. 77, No. 4, pp. 541-580 (1989).
2) 村田忠夫: ペトリネットの解析と応用, p. 203, 近代科学社 (1992).
3) Notomi, M. and Murata, T.: Hierarchically Organized Petri Net State Space for Reachability and Deadlock Analysis, Proc. of IPPS '92, pp. 616-622, IEEE Comp. So. (Mar. 1992).
4) Helmbold, D. and Luckham, D.: Debugging Ada Tasking Programs, IEEE Softw., Vol. 2, No. 2, pp. 47-57 (1985).
5) Tu, S., Shatz, S. M. and Murata, T.: Theory and Application of Petri Net Reduction for Ada-Tasking Deadlock Detection, Report #UIC-EECS 91-15 (1991). Also, an earlier version appeared in Procs. of ICDCS '10, Paris, France, pp. 96-103 (May 1990).
6) Berthelot, G.: Checking Properties of Nets Using Transformations, Rozenberg, G. (ed.),

Advances in Petri Nets 1985, LNCS 222, pp. 19-40, N. Y., Springer-Verlag (1987).
7) Shatz, S. M. and Cheng, W. K.: A Petri Net Framework for Automated Static Analysis of Ada Tasking Behavior, J. Syst. Softw., 8, pp. 343-359 (1988).
8) Murata, T., Shenker, B. and Shatz, S. M.: Detection of Ada Static Deadlocks Using Petri Net Invariants, IEEE Trans. Softw. Eng., Vol. 15, No. 3, pp. 314-326 (1989).
9) Tiusanen, M. and Murata, T.: Graph Models for Static Analysis of Ada Tasking Programs, 情報処理研究会報告, Vol. 92, No. 59, pp. 141-150 (1992), および第9回離散事象システム研究会講演論文集, pp. 25-34 (1992).

(平成5年1月7日受付)

村田 忠夫 (正会員)

1938年生. 東海大卒後, 1962年米国イリノイ大学アーバナ校大学院留学, 1964年修士, 1966年 Ph. D. 修得. 同年より同大学シカゴ校工学部助教授. 現在同校 Senior University Scholar 教授. 1993年7月より一年間, 大阪大学基礎工学部客員教授. IEEE フェロー, 1991年 IEEE Donald G. Fink 賞受賞. 著書「ペトリネットの解析と応用」など.



辻 孝吉

昭和57年福井大学工学部電気工学科卒業. 昭和59年同大大学院修士課程修了. 昭和62年大阪大学大学院後期課程修了. 工学博士. 同年福井大学工学部助手. 平成4年文部省在外研究員として米国イリノイ大学で離散事象システムに関する研究に従事. 現在, ペトリネット, 離散事象システム, システム理論の研究に従事.