A-015

ペトリネット援用ツールを用いたモデル設計とポスト検証ツール向け状態空間生成アルゴリズム

Model Designing using A Petri Net Tool and State Space Generation Algorithm for Post-Verification Tool

太田 淳也 †　和崎 克己 ††
Junya Ohta　Katsumi Wasaki

1　はじめに

非同期回路や非同期に動作する通信プロトコルなど，並列かつ非同期に動作するシステムの設計及び動作検証は，一般に難しいとされている．一方で，事象発生の並列性・非同期性・非決定性を有する離散事象システムの設計及び検証に有用な数学的ツールとしてペトリネット [1] がある．システム設計にペトリネットを用いることは，並列システムの記述や動作解析上有用である．

ペトリネットを用いたシステム設計を行うため，著者らの研究グループはペトリネット設計ツール HiPS(Hierarchical Petri net Simulator)[2] を開発している．ペトリネットベースでの効率的なモデル化を行うために，HiPS は一般的な操作方法の GUI を備え，ネットの階層化機能及びモデルの動作解析機能が実装されている．しかし，実システム規模のモデルに対し動的・構造的な解析を行うと，結果として膨大な数の解析結果が得られ，設計者が観察したい特徴や代表的なトレースを得にくくなる問題がある．そこで設計部分と検証部分を分離し，設計に関しては従来通り HiPS で行い，検証に関しては専用の検証ツールを用いて，詳細な検証を行う．ペトリネットを用いた設計から，モデル検査器による検証までの流れを図 1 に示す．ポスト検証ツールとして CADP toolbox[3] を用いる．モデル検査器には，設計者が特に観察したい振る舞いの条件を記したファイルと共に，モデルの状態空間を入力する必要がある．

ペトリネットにおける状態空間は有界な可達グラフとして与えられるが，モデルの規模に対して状態数が爆発的に増加する状態空間爆発の問題がある．効率的な設計及び検証を行うためには，莫大な状態空間を高速に生成する必要がある．本研究では，ネットの接続行列と初期マーキングから可達グラフ生成を行うアルゴリズムと，生成時における同一の状態値を効率よく検索する手法について提案する．

提案方法の評価を行うため，提案アルゴリズムに基づいた生成器を試作し，状態空間生成性能の計測を行った．対象として様々な規模の状態空間を持つネットを HiPS 上で設計し，実行時間性能等の観点から評価を行った．最後に，HiPS を用いたモデル設計及び検証を行った．モデルの状態空間と記述した検査条件をモデル検査器に入力し，各種の動作検証を行った．

2　ペトリネット

ペトリネット (Petri net) とは，Carl Adam Petri によって提唱された，複数のプロセスからなる離散事象システムをグラフィカルに表現するモデルである．並行性，非同期性，分散的，並列的，非決定的，確率的な動作を特徴とする情報処理システムを，記述・研究するツールとして有用である．システムの構造の可視的な表現手段としての利用だけでなく，ペトリネット内でトークンを使用することにより，システムの振る舞いを動的に解析できる．

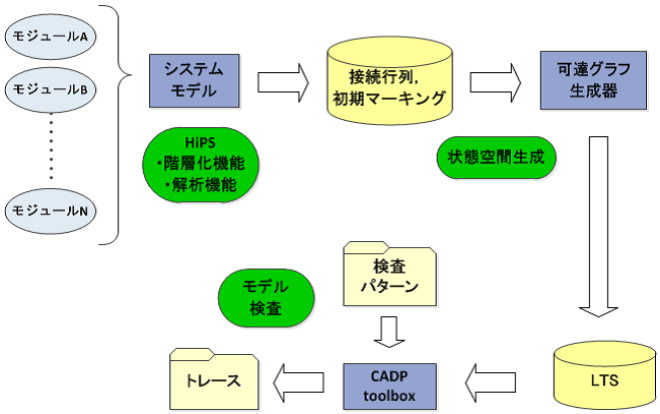

図 1　HiPS とモデル検査器の連携

2.1　可達性

マーキング M_0 をマーキング M_n へ変換する発火系列 $\sigma = M_0 t_1 M_1 t_2 M_2 \ldots t_n M_n$ が存在するとき，M_n は M_0 から可達であるといい，$M_0[\sigma > M_n$ と記す．ネット (N, M_0) において，M_0 から可達なすべてのマーキングの集合を $R(M_0)$ と表す．

2.2　可達グラフ

ペトリネット N において，M_0 から可達マーキングを順次求めていく過程から N の可達木を得る．可達グラフはラベル付けされた有向グラフ $G = (V, E)$ である．ここで，ノードの集合 V は可達木内のすべての異なったマーキングを持つノードの集合であり，$R(M_0)$ と等しくなる．また，アーク集合 E は $M_i[t_k > M_j$ であるような単一のトランジション発火を表現しているトランジション t_k でラベル付けされたアークの集合である．ペトリネットにおける状態空間は可達グラフとして与えられる．

2.3　接続行列と状態方程式

ペトリネットの接続関係を表す行列及び，その行列と発火ベクトルを用いたマーキング状態の遷移を表す式について述べる．

† 信州大学大学院理工学系研究科，Graduate School of Science and Technology，Shinshu University.
†† 信州大学工学部，Faculty of Engineering，Shinshu University.

2.3.1 接続行列

トランジション数 n，プレース数 m であるようなペトリネット N に対して，接続行列 $A = [a_{ij}]$ は n 行 m 列の行列であり，トランジションとプレースの接続関係を表す．接続行列の各成分は $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$ で与えられる．ここで，$a_{ij}^{+} = w(i,j)$ はトランジション i からその出力プレース j へ向かうアークの重みであり，$[a_{ij}^{+}]$ を前向き接続行列という．また，$a_{ij}^{-} = w(j,i)$ はトランジション i の入力プレース j からトランジション i へ向かうアークの重みであり，$[a_{ij}^{-}]$ を後ろ向き接続行列という．さらに，トランジション i の発火可能条件は a_{ij}^{-} を用いて，以下の式で表すことができる．

$$a_{ij}^{-} \leq M(j),\ j = 1, 2, \cdots, m \tag{1}$$

2.3.2 状態方程式

トランジション k の発火によるマーキングの変化は接続行列 A を用いて，以下の式で表される．

$$M_k = M_{k-1} + A^T u_k,\ k = 1, 2, \cdots \tag{2}$$

ここで，マーキング M_k は，m 行 1 列の列ベクトルであり，j 番目の成分はある発火系列 σ における k 番目のトランジション発火直後のプレース j 内のトークン数を表す．また，発火ベクトル u_k は n 行 1 列の列ベクトルであり，発火系列 σ の k 番目のトランジション i の発火を表す．

3 ペトリネット設計援用ツール HiPS

HiPS は，既存のペトリネットツールの記述性，操作性，再利用性の問題を解決するために本学で開発された．図 2 に，HiPS の編集画面を示す．HiPS は直感的かつ一般的な GUI で操作可能であり，ペトリネットの階層化に対応している．また，作成したモデルのシュミレーションにより，挙動の様子を観察できる．さらに，ペトリネットの数学的性質に基づいた解析器が多数実装されており，モデルの動作解析が可能である．代表的な構造的解析器として Fourier-Motzkin 法を応用した高速インバリアント解析器が実装されている [4]．階層化機能と各種解析機能を同時に用いることで，厳密かつボトムアップ的なシステム設計を効率化でき，有用である．

ペトリネットの構造的解析は検証コストが低く，高速に解析可能であるという特徴を持つ反面，実システム規模のモデルに対して解析を行うと，結果として膨大な数の解析結果が得られ，観察したい特徴や代表的なトレースが得られにくくなってしまう問題がある．また，得られたトレースはペトリネット上で再現しづらく，また形式的なトレース結果の保存も困難である．

本研究では，非同期に動作する部分を多く持ち，構造的な解析が適用しにくいモデルにおいて，システム全体の網羅的な振る舞いを状態空間として生成し，観察したいパターンとともに外部のモデル検査器に入力することで，特定のトレースを観察できるようにする．

4 提案する状態空間生成器

本節では，ネットの接続行列 A と初期マーキング M_0 から状態空間生成を行うアルゴリズムと，グラフ生成時における同一マーキングの検索方法について述べる．

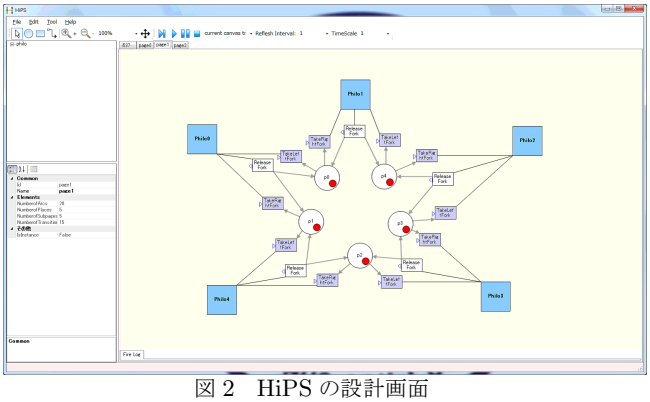

図 2 HiPS の設計画面

4.1 状態空間生成アルゴリズム

従来，HiPS に実装されていた可達グラフ生成器はいくつかの実装上の問題点があり，実行性能が低かった．問題点として，以下のものが挙げられる．

1. 再帰関数を用いた深さ優先探索での実装のため，関数呼び出しのオーバーヘッドや呼び出し回数の限界があること
2. 状態空間を保存するハッシュコンテナにおいて，要素の検索速度が劣化すること
3. マーキング状態の遷移を，ネット上にトークンを展開し，逐一シュミレーションを行うことで表現していること

提案する有界な可達グラフ生成アルゴリズムを Algorithm 1 に示す．また，このアルゴリズムの動作フローを図 3 に示す．

提案方法では，まず問題点 1 について，再帰関数を用いた深さ優先探索から，ハッシュコンテナとキューを用いた探索に変更した．このことで，関数呼び出しのオーバーヘッドや呼び出し回数の制限に依らない実装を実現した．

次に問題点 2 について，可達グラフの生成時には，ある瞬間に生成された状態値と同一の状態値の有無を，その瞬間までに得られている全状態空間から検索する必要がある．このことから，状態空間の生成性能は状態値の検索効率に大きく依存する．よって，生成性能の向上のためには，状態空間爆発による膨大な状態空間においても状態値の検索速度を維持する必要がある．コンテナ内の要素数が膨大であっても，特定の値の検索が比較的高速に行えるハッシュコンテナを使用し，単一のマーキング状態の検索速度を維持した．

さらに問題点 3 に対して，マーキング状態の遷移を逐一シュミレーションする方法から，ネットの初期マーキングと接続行列及び後ろ向き接続行列を，発火可能性の判定式 (1)，状態遷移式 (2) に与え，単純な数値の比較及び計算でマーキング状態の遷移を表現した．

アルゴリズムの詳細な動作を説明する．Algorithm 1 中で，buf は新規に生成されたマーキングを一時的に保存しておくためのコンテナである．また，$hashmap$ は状態空間を保存するためのハッシュコンテナである．まず初期化処理として，初期マーキング M_0 を buf と $hashmap$ に代入する．buf が空でない間，buf から取り出したマーキングに発火可能性の判定式 (1) を適用

Algorithm 1 可達グラフ生成アルゴリズム

```
1: buf ← M_0
2: map ← M_0
3: while buf is not empty do
4:   m ← buf.pop()
5:   for i = 0 to row size of A do
6:     fireble ← true
7:     for j ← 0 to column size of A do
8:       if a⁻_{i,j} > m(j) then
9:         fireble ← false
10:        break
11:      end if
12:    end for
13:    if fireble is true then
14:      mnext ← m + A^T u_i
15:      get transition from m to mnext
16:      if map does not have mnext then
17:        map ← mnext
18:        buf ← mnext
19:      end if
20:    end if
21:  end for
22: end while
```

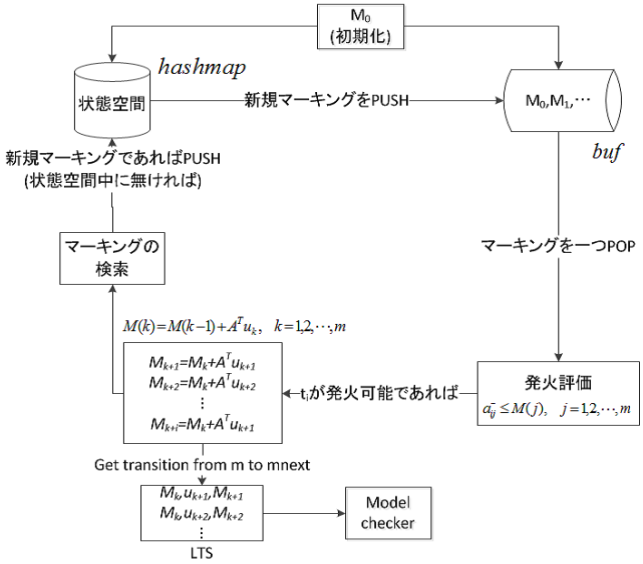

図 3　可達グラフ生成アルゴリズムの動作フロー

し，発火評価を行う．発火評価の結果，トランジション i が発火可能ならば，状態遷移式 (2) を適用し，現在のマーキング m とトランジション u_i から新たな次段可達マーキング $mnext$ を生成する．同時に，m と $mnext$ 間のマーキング間の遷移を得る．可達グラフを生成するために，現在までに得られている状態空間中に $mnext$ と一致するものがあるかどうかを検索する．検索の結果，$mnext$ が $hashmap$ に含まれていなければ，そのマーキングを新規マーキングとして扱い，buf と map に保存し，処理を続行する．

states	transition	branch	time (sec)	speed (states/sec)	memory (MB)
152	356	2.34	0.001	152,000	2
7,502	32,451	4.33	0.134	55,985	5
32,590	148,532	4.56	0.868	37,546	19
367,652	2,325,406	6.33	14.018	26,227	206
1,622,564	10,589,425	6.53	132.669	12,230	1,342
18,015,002	149,974,901	8.33	1,155.930	15,585	12,749

表 1　状態空間生成器の実行性能

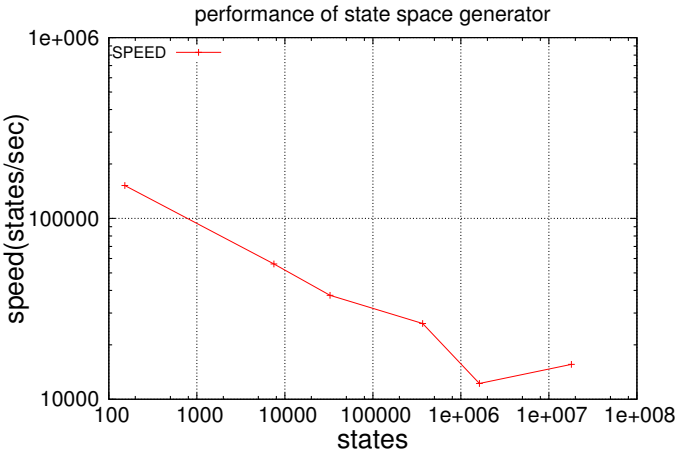

図 4　状態空間生成器の実行速度

4.2　実行性能の計測結果

提案したアルゴリズムを評価するために，以下の生成器を試作した．ハッシュコンテナは，Boost C++ library[5] のハッシュライブラリを利用した．HiPS 上で設計した様々な規模のネットに生成器を適用し，生成能力を計測した．計測を行った項目は，状態数・ブランチ数 (transition/state)・経過時間 (sec)・実行速度 (states/sec)・メモリ (MB)・トランジション数の 5 つである．実行計算機の性能は OS : Windows 7 Professional，CPU : Intel Core i7 870 2.93GHz，メモリ : 16.0GB である．実行性能を表にまとめたものを表 1，実行速度のグラフを図 4 に示す．

結果として，状態数 7,502，平均ブランチ数 4.33 のとき，生成速度は約 56K states/sec であった．また，状態数 367,652，平均ブランチ数 6.33 のとき，生成速度は約 26K states/sec であった．さらに，状態数 18,015,002，平均ブランチ数 8.33 のとき，生成速度は約 16K states/sec であった．以上のことから，本生成器の生成能力は，複雑かつ膨大な状態空間を持つシステムに対しても一定の生成能力を維持することがわかった．

5　モデル検査器を用いた検証例

本節ではペトリネットによるシステム設計とモデル検査器を利用した検証の流れを説明する．

5.1　モデル検査器との連携の流れ

ポスト検証ツール [3] を利用したモデル検査の流れは図 1 に示した通りである．まず，HiPS を用いて下位モジュールの設計を行う．これはシステムを機能ごとに分

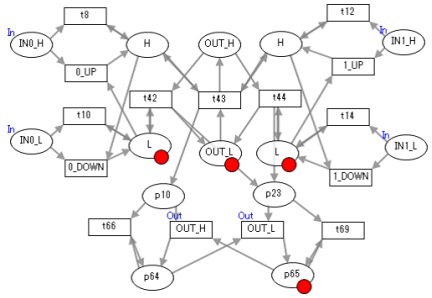

図 5　2 入力 AND ゲートのペトリネットモデル

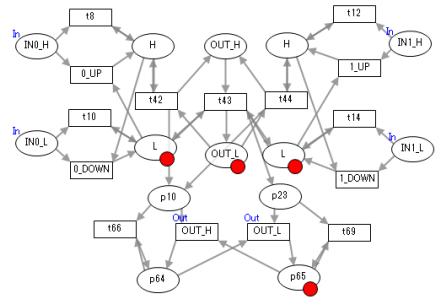

図 6　2 入力 OR ゲートのペトリネットモデル

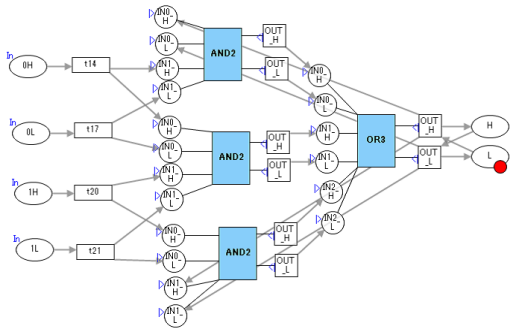

図 7　階層化機能を用いて設計された C-element モデル

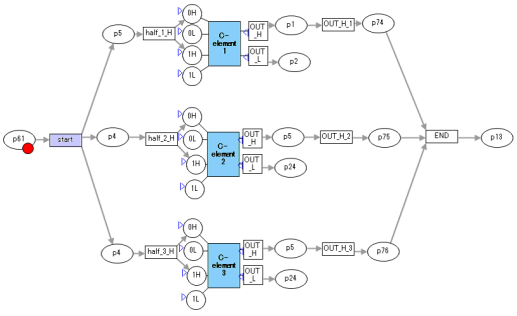

図 8　C-element を並列に接続したモデル

解したときの各機能部品に当たる．このとき機能部品に解析器を適用することで，比較的モデル規模の小さいモジュール段階での動作検証を行い，動作を保証する．次に，HiPS の階層化機能を利用し，モジュールを組み込んだシステム全体の設計を行い，モデル全体を作成する．作成したモデル全体の振る舞いを生成するために，モデルを 4 節で示した状態空間生成器に入力し，LTS(Labeled Transition System) を生成する．生成された状態空間と検査したいパターンを記述したファイルを CADP toolbox のモデル検査プログラム (exihibitor) に与え，観察したいトレースについて検査を行う．

5.2　検証例

モデル検査器を用いた検証例として，C-element(図 7) を作成した．このモデル中では，先に作成しておいた AND ゲート (図 5) と OR ゲート (図 6) を階層化して利用している．C-element モデルを並列に 3 個接続したモデル (図 8) を用意し，可達グラフ生成を行ったところ，状態数 3,629,458, トランジション数 25,498,616, ブランチ数 7.02 の LTS が得られた．生成にかかった時間は，約 481sec，使用メモリ量は 1,990MByte であった．チェックする項目として，最終段の C-element の出力が L から H になるトレースを検査したところ，初期状態から 45 states，44 transition のトレースが検出できた．モデル検査器の実行計算機の性能は OS : Fedora Core 8，CPU : Intel Core i5 M540 2.53GHz，メモリ : 1.0GB であり，検査にかかった時間は，21.166sec であった．

6　まとめと今後

階層化機能と動作解析機能を持つペトリネット設計援用ツール HiPS とモデル検査器を連携させることによって，ネット志向ベースで実規模なシステム検証の可能性を示した．本研究では，HiPS を用いて作成したモデルの詳細な動作検証を行うため，モデルの状態空間を生成し，外部のモデル検査器に入力し検証を行った．また，状態空間生成について，膨大な状態数をもつモデルに対しても高速に LTS 生成を行えるような，ペトリネットにおける有界な可達グラフ生成アルゴリズムを提案した．さらに，提案アルゴリズムに従って可達グラフ生成器を生成し，実効性能を確かめた．

今後は，マルチコアプロセッサを用いた並列プログラミングによる，状態空間生成器の高速化を行う．また，現在の状態空間生成器の問題点として，生成された状態数に従ってメモリの使用量が増加してしまうという問題がある．現在は，一つの状態を要素が整数であるベクトルで表現している．マーキングのデータ型を見直すことや，状態空間を保存しているハッシュコンテナのデータ構造の改良などを行い，メモリの使用量の削減を行う．実行時間性能と使用メモリ量の改良により，より大規模な実システムをより具体的に設計し，詳細な検証を行いたい．

謝辞

本研究の一部は科学研究費（23500174）の助成を受けたものである．

参考文献

[1] T. Murata, “Petri Net: Prooerties, Analysis and Applications.”, Proceedings of the IEEE, Vol.77, No4, pp.541-580, 1989.
[2] HiPS : Hierarchical Petri net Simulator, http://sourceforge.net/projects/hips-tools/
[3] CADP toolbox : http://cadp.inria.fr/
[4] 堀内維作, 松山千尋, 和崎克己, “ペトリネットツール HiPS におけるインバリアント解析機能の回路検証への適用”，平成 22 年度電子情報通信学会信越支部大会, 2D-4, 2010.
[5] Boost C++ Libraries, http://www.boost.org/