A-014

ペトリネット状態空間生成器への高効率ハッシュマップの適用

Application to High Efficiency Hash-map for State Space Generator of Petri Net

古市 隼汰 †　　和崎 克己 ††
Syunta Furuichi　Katsumi Wasaki

1　はじめに

近年のシステムでは情報量の増大や負荷分散の観点から非同期動作や並列動作などが主流となっている．しかし，エラー検出時に誤りが現れにくく，これらを対象とした検証は一般に難しいとされている．これらシステムの検証に用いられる方法としてペトリネット [1][2] でのモデル化がある．

ペトリネットは事象発生の並列性，非同期制，非決定性を有する離散事象システムの振る舞いを表す数学モデルであり，グラフィックツール，シミュレーションツール，及び数学的方法論の 3 つの機能を同時に持っている．グラフィックツールとしてシステム構造を可視的な表現で記述し，ペトリネットの中でトークンを使用することによりシステムの並列事象をシミュレーションできる．また数学的ツールとしてシステムの挙動を方程式や行列式を用いる事でモデリングが可能である．

このペトリネットの記述，シミュレーション，解析をより容易にするため，筆者らはペトリネット設計解析援用ツール HiPS[3] の開発リリースを行っている．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．

システム全体の振る舞いを確認，解析する上でシステムの取り得る状態遷移を出力する必要がある．これは状態空間生成器 [4] として HiPS に実装されており，網羅的な状態空間を探索する．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．これは同時に解析時間とメモリ消費量の増大を伴う．そのため有限なリソースの中，膨大な状態空間を高速に生成する必要がある．本研究では並列処理を用いた可達グラフ生成アルゴリズムと 4bitINT 及びハッシュマップを用いたメモリ管理手法について提案する．

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．状態を表すプレース，事象を表すトランジション，関係を表すアーク，リソースを表すトークンなどから構成される重み付き有向二部グラフであり，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，記述するツールとして有用である．システム構造の可視的な表現手段としての利用だけでなく，トークンを使用する事によりシステムの振る舞いを動的に解析することができる．

† 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.

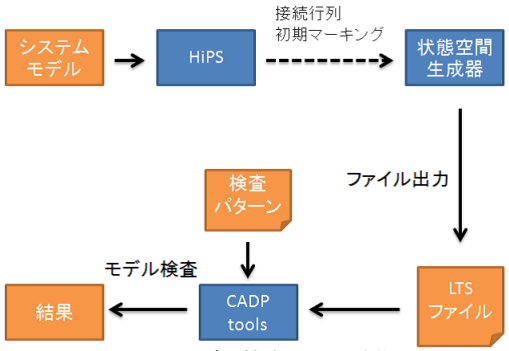

図 1　モデル検査器との連携

2.1　マーキング

ペトリネットにおいて，あるプレース $p \in P$ に対し，非負整数 k が割り当てられたとき，プレース p は k 個のトークンでマーキングされていると言い，この時トークンはプレース p 内の k 個の点として図示される．ペトリネットは，マーキングによりシステムの状態を表現し，m 個のプレースからなるペトリネット全体のマーキングは m 次元ベクトル M で表される．特にマーキングの初期状態を初期マーキング (initial marking)M_0 と呼び，マーキングはトランジションの発火により遷移する．

2.2　可達性

マーキング M_0 からマーキング M_n へ至る発火系列 $\sigma = M_0 t_1 M_1 t_2 M_2 \ldots t_n M_n$ 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．

2.3　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．

2.4　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．

2.4.1 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．

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

2.4.2 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．

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

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

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，我々の研究室ではペトリネット設計ツール HiPS(Hierarchical Petri net Simulator) の開発を行っている．HiPS は，直感的かつ一般的な操作方法の GUI を持ち，階層構造化に対応している．また、作成したモデルのシミュレーションにより、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，モデルの動的解析構造的解析が可能である．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．階層化機能と各種解析機能を同時に用いる事で，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言う特徴を持つ反面，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現れ，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．状態空間生成器は，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モデルの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機能である．本研究では，この状態空間生成器をより効率よく，より高速に状態生成することを目指す．

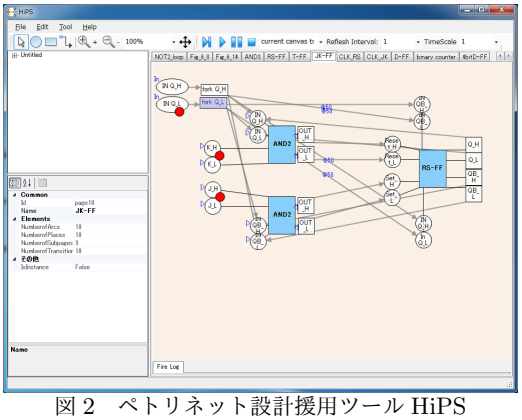

図 2 ペトリネット設計援用ツール HiPS

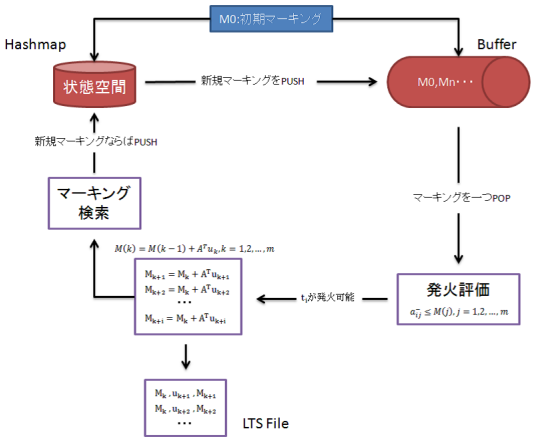

図 3 シングルスレッド版状態生成器の処理フロー

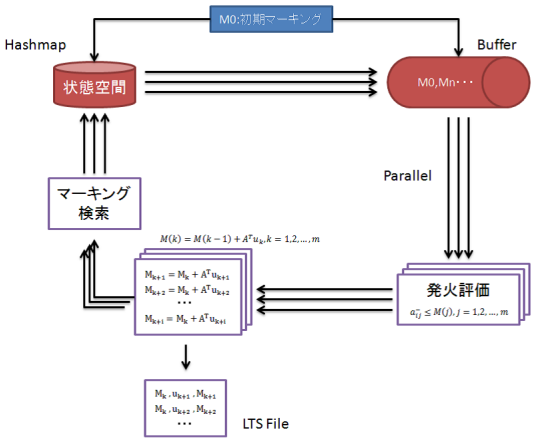

図 4 マルチスレッド版状態生成器の処理フロー

4 状態空間生成器と並列処理化

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．状態生成は初期マーキング M_0 から，マーキングと後ろ向き接続行列を基に，発火可能なトランジションを順次評価し次状態のマーキング生成，更に発火評価の後，次状態のマーキング生成を繰り返し，M_0 から可達な全てのマーキングを求める．図 3 に現在導入

されているシングルスレッド版状態空間生成器の処理フローを示す．本研究では，Intel より提供されているマルチスレッド対応の C++ テンプレートライブラリである IntelTBB(Intel Threading Building Blocks)[5] を用いて，状態生成状態空間からの状態番号検索 LTS コンテナへの追加の処理を並列動作させることで生成時間の短縮を図った．

5　4bitINT の導入とダブルハッシュ構造化

従来の状態空間生成器ではマーキングに用いられるトークン数を標準 API の 8bitINT を用いて表現していた．これはセーフなネットにおいて状態遷移を表す場合，殆どのプレースにおいてオートマトンの状態遷移のように 1 つのトークンがシーケンス的に動いていくだけであるという特徴があるからである．つまり，複数個有るプレース全てが容量の概念を持つ変数として扱う事は殆ど無く，シーケンス的な動きに耐えうる 1，0 を表現すれば良い．容量の概念が必要なプレースはごく一部のプレースのみである．そこで INT 型を用いるよりもより少ないメモリ容量でマーキングを表現でき，生成状態数の向上に繋がる 8bitINT が用いられて来た．しかし，更なる状態数の生成が望まれる．そこで今回 4bitINT を考案してより多くの状態数を生成できるよう改良を試みた．また，データ格納は汎用用途である boost::unorderedMap 型 [6] を用いていた．これをペトリネットのマーキング管理に特化したダブルハッシュに置き換えることでより高速な挿入，参照を試みた．

5.1　4bitINT の導入とダブルハッシュ構造化

有限なペトリネットの条件下において極端なトークン数の増大は見受けられないことが多い．そのため殆どのプレースには一桁程度のトークン数しか入らず，更に一部プレースのみしか数の変化を起こさないためトークン数を表現するために確保されたメモリが無駄になってしまうことがある．ここで，増減を伴うビットのみを可変ビットとし動的なビット幅の管理を行う事で極限まで状態空間を圧縮する事が可能になる．しかし，理想的なビット幅の算出は困難であり，さらに動的リンクに伴うオーバーヘッドや参照に用いられるメモリが無駄となってしまう．また，個々のプレースに対するメモリ管理もまた困難となる．そこで新たに 4bitINT 型を作成し以前の半分のビット数でトークン数を表現することを試みた．これは構造体として，4bit の非負整数を 8 個セットした物を最小単位として扱う事でトークン数を 4bit で表現したという物である．これにより表現可能なトークン数は 16(2 の 4 乗) と減ってしまうものの，以前の半分のメモリでマーキングが確保でき理論上 2 倍の状態数を確保できることになる．

5.2　ダブルハッシュ構造

状態空間生成において，生成されたマーキングが新たなマーキングであるかそうでないかの確認を常に行うことでマーキングに対してナンバリングを行う．これはすでに挿入された状態空間からのマーキングの検索，挿入を行うことで実現している．この検索行程が状態空間生成において時間が掛かる一番の要因となっているため改善の余地があった．

現在状態確保に用いられているのは Boost ライブラリに含まれる unorderdMap コンテナである．これは汎用用途で用いられるコンテナであり，今回これをペトリネットの特性に合わせて高速な検索ができるよう自作コンテナに置き換え高速化を試みた．状態空間生成器はマーキングが一定値以下で遷移する有界なネットに対して適用が可能である．この有界なネットの条件下での発火に伴うマーキングの変化として極端なトークンの増加は考えにくい．また，1 つのトランジションの発火に対するプレース内のトークン数の変化として全てのトークン数が変化することは無くアークで接続された 2 つないし 3 つ程のプレースのトークン数のみの変化が主である．つまり，トランジションの発火に対して全体としては殆ど値の変化が無く，ごく一部の値の変化のみが見受けられる特徴がある．この特徴を考慮して，高速かつ微妙な値の変化で振り分け可能なハッシュを 2 重構造にし，図 6 に示すようなダブルハッシュをコンテナとして扱う事で高速化を試みた．

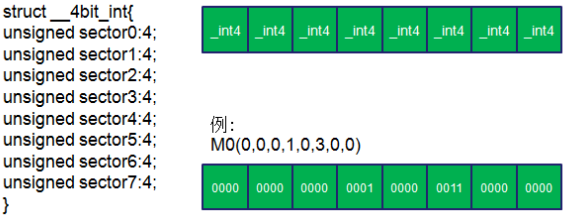

図 5　4bitINT

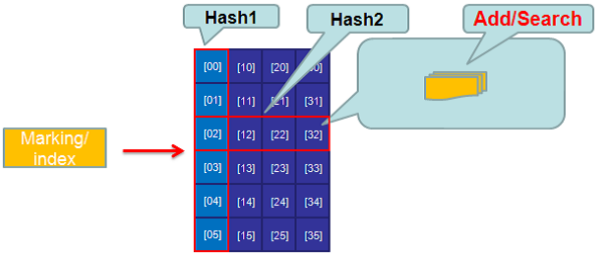

図 6　DoubleHash

6　実装と検証結果

提案したアルゴリズムを評価するために下記の環境でのテストを行った．OS：Windows7 Professional，CPU：Intel Core i5 4200M 2.5GHz，メモリ：8GB である．

6.1　並列化状態生成

実装した可達グラフ生成器の性能計測のため，図 7 に示すネットをモジュールとしたネットを計測対象とした．このモジュールは入力プレースに 1 つのトークンを置くことで 10 の状態を取り得るモデルである．このモデルを HiPS の持つ階層化機能によってモジュール化し，ネット上に独立して配置する．各モジュールは入出力の外部ポートを持ち，この入力ポートに対しそれぞれ 1 つずつのトークンを入力し並列動作させる．これにより各モジュールの状態数の積による状態を作成する事で膨大な状態数を持つ図 8 のネットを作成する．このネッ

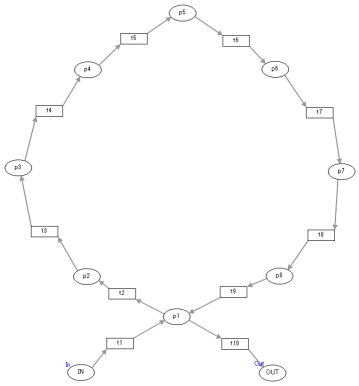

図 7　ベースモジュール (Module k)

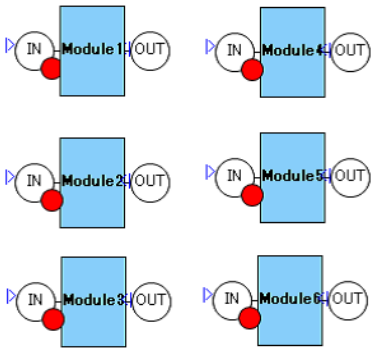

図 8　生成対象ネット (Module を階層化したもの)

state	transition	branch	time(sec)	speed(states/sec)	memory(MB)
10	10	1	0.00055	18,623	80.00
100	200	2	0.00118	85,109	78.67
1,000	3,000	3	0.00876	124,144	79.67
10,000	40,000	4	0.07613	132,149	86.33
100,000	500,000	5	0.90526	110,595	156.33
1,000,000	6,000,000	6	12.06710	83,086	986.00

図 9　シングルスレッドモードにおける実行性能

state	transition	branch	time(sec)	speed(states/sec)	memory(MB)
10	10	1	0.00100	10,289	79.33
100	200	2	0.00207	48,518	79.33
1,000	3,000	3	0.01807	60,497	81.33
10,000	40,000	4	0.07330	137,096	90.67
100,000	500,000	5	0.77640	129,006	191.00
1,000,000	6,000,000	6	10.16849	98,425	1,390.67

図 10　マルチスレッドモードにおける実行性能

トに対して以前の生成手法である Boost ライブラリを用いたシングルスレッドモードと，今回の提案手法である IntelTBB ライブラリを用いたマルチスレッドモードにて状態生成，LTS ファイル出力を行い，それぞれのネット規模に基づく実行時間 (sec)・生成速度 (states/sec)・メモリ使用量を測定し，比較を行った．それぞれ結果を表 9，表 10 に示す．

各結果より，本生成器は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分かった．また，同じ状態数での実行時間を比べた時に，以前よりも短時間で状態生成を行える事が分かる．これにより，以前よりも高効率での状態生成ができている事が分かる．一方で状態数が 1000 よりも小さい場合，以前のシングルスレッドモードで動作させた方が高速であることが分かった．これは，マルチスレッド向けコンテナはシングルスレッド向けのコンテナに比べてオーバーヘッドの割合が大きいことが原因と思われる．また，マルチスレッドモードでのメモリ使用量はシングルスレッドモードに比べて 1.2 倍程度の増加が見られた．

6.2　4bitINT とダブルハッシュ

実装した 4bitINT とダブルハッシュの性能計測のため，以前の生成手法である 8bitINT と Boost ライブラリのコンテナを用いたシングルスレッドモードと，今回の提案手法である 4bitINT とダブルハッシュコンテナを用いたモードにて状態生成を行い，それぞれのネット規模に基づく実行時間 (sec)・生成速度 (states/sec)・メモリ使用量を測定し，比較を行った．

全般として使用メモリにおいて，以前の生成器よりも若干の省メモリ化が見受けられた．しかし，生成時間においては以前よりも大幅に増大してしまい，規模の増大とともに際立った結果となった．これは，マーキングが等価であるかの比較のために都度全てのマーキングの確認を行う事と，ダブルハッシュでの衝突の際にチェイン法を用いているため，規模増大に伴い同一マーキングの検索に必要とする時間も増大した事が原因であると考えられる．

7　まとめと今後の課題

今回 IntelTBB を用いた高速化では並列化を用いたことで従来の生成器よりも大幅な高速化が図られた．4bitINT でのマーキング表現ならびにダブルハッシュによるメモリ管理では目立った省メモリ化が見られなかった．これは，4bitINT をダブルハッシュで管理するに当たってのクラスの多段構造や各マーキングを格納する際のアドレス用メモリの消費が原因であると考えられる．また実行時間の増大においては，挿入検索処理で従来の unorderdMap コンテナよりも大幅な時間のロスが起きてしまっていることが考えられる．

今後はこの点を考慮してメモリの一律確保によるアドレス用メモリの削減や，ブルームフィルタなどによる外部インデックスを用いて，より高速な検索アルゴリズムの導入を検討している．また，並列処理時のメモリ量増加を考慮し今回のシングルスレッド向き 4bitINT に加え，並列処理での状態生成に合わせた 4bitINT の導入を考えていく．

参考文献

[1] Tadao Murata, “Petri Net: Properties, Analysis and Applications”, Proceedings of the IEEE, Vol.77, No.4, 1989.
[2] 村田忠夫, ペトリネットの解析と応用, 近代科学社, 1992.
[3] HiPS : Hierarchical Petri net Simulator, http://sourceforge.net/projects/hips-tools/
[4] 太田淳也, 和崎克己, “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”, FIT2013 (第 12 回情報科学技術フォーラム), 4A-5, 2013.
[5] Intel TBB Libraries, https://software.intel.com/en-us/intel-tbb
[6] Boost C++ Libraries, http://www.boost.org/