A-005

仕様に基づいたペトリネットのトランジション優先発火による On-the-fly モデル検査の効率化

Improve Efficiency of On-the-fly Model Checking of Petri Net using Transition Priority Firing based on Inspection Specification

張江 洋次朗 †　和﨑 克己 ††
Yojiro Harie　Katsumi Wasaki

1 まえがき

ペトリネットは，離散事象システムを表現・解析が可能な数学的手法の一つである [1]．HiPS (Hierarchical Petri net Simulator) は，ペトリネット設計ツールとして筆者らの研究グループによって開発・公開されている [2]．HiPS はモデルの状態遷移について，システムの振る舞いを表す状態空間を生成する機構を有する．また，外部モデル検査ツールと連携ができ，状態空間中から条件を満たすトレースの抽出が可能である．しかし外部ツール上で行うモデル検査は，状態空間の全構築が必須である．このため，全状態空間数が膨大となる，状態爆発を起こすモデルに対して，空間的・時間的観点から外部ツールと連携した検証は困難になる．

状態爆発に対する解決の一つに On-the-fly 法 [3] がある．On-the-fly 法は，状態空間の構築と並列して探索を行うことで，受理列を発見した場合に，すべての状態空間の構築より前に探索および構築を終了させる．On-the-fly 実行を行う検査アルゴリズムは複数あり，ここでは LTL モデル検査において逐次最適な探索アルゴリズムである Nested Depth First Search (Nested DFS) [4] を適用する．

本稿では，On-the-fly 実行における仕様オートマトン生成に関する効率化とともに，状態空間の生成順序に関する戦略について提案する．具体的には，事前に仕様 (時相論理式) から得られる観測したいアクション集合に基づき，ペトリネットの次段マーキングの非決定的選択時，仕様に現れるトランジション発火を優先した状態空間の生成を行う．検査に関する重要な状態は全状態空間中の一部であるため，On-the-fly 検査時には提案手法による観測アクション列の早期生成による効率化が見込まれる．

2 ペトリネット

2.1 P/T net

Carl Adam Petri によって提唱された数学的モデリングツールとして Place/Transition net (P/T net) がある [5][6]．並行的，非同期的，分散的，並列的，非決定的といった性質・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．ペトリネットは，トランジションとプレースというノードからなる 2 部グラフであり，トランジションを発火させプレース内のトークンを消費・生成することで状態遷移を行いモデルの振る舞いをシミュレーションする．ペトリネット内のトークンの様子をマーキングといい，特に初期状態のペトリネット内のトークンの様子を初期マーキングという．一般にペトリネットはネットの構造 $N = (P, T, F, W)$ と初期マーキング M_0 の組 (N, M_0) で表される．

2.2 ペトリネット設計開発ツール HiPS

トークンが何らかのデータを保持できるときカラーペトリネットの設計，解析を行えるツール CPN Tools [7] や，時間ペトリネット・確率ペトリネットの設計解析シミュレーションツール PIPE2 [8] がある．いずれのツールも仕様定義からモデル修正まで網羅的なサポートが行えているとは言い難い．既存のペトリネットツールの記述性，操作性及び再利用性の問題を解決するため筆者らのグループにより開発された，ペトリネット設計ツール HiPS (Hierarchical Petri net Simulator) [2] がある．HiPS は直感的で一般的な操作方法の GUI を持ち，ランダムウォークシミュレーションに対応している．C# と C++ や.NET Framework といった言語・環境で開発され，階層的設計や動的・構造的性質に関する解析機能により詳細な検証が可能である．

2.3 状態空間生成

動的解析について，設計したモデルの振る舞いである状態空間を全て生成し，検証を行う．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，状態空間は Labelled Transition System (LTS) によって表現される．有界な可達グラフ生成アルゴリズムが提案されている [9]．有界な可達グラフ生成アルゴリズムの動作フローを図 1 に示す．初期マーキング M_0 から発火評価を行い，次段マーキングを得る．得られたマーキングが状態空間中に含まれていなければ次段マーキングバッファへの登録を行う．次段マーキングバッファが空になるまで逐次的に発火評価を行い，状態空間の生成を行う．

3 On-the-fly モデル検査

3.1 FLTL モデル検査

命題論理は形式的に厳密に真偽を確定させるため，仕様の表現に利用される．命題論理の一種である線形時相論理 [10] がモデル検査でシステムの満たすべき性質である仕様の記述に用いられる．HiPS におけるモデル検査では，線形時相論理 (Linear Temporal Logic: LTL) を拡張定義した，流動線形時相論理 (Fluent Linear Temporal Logic: LTL) を検査仕様の記述に用いる．LTL は無限長の語を表現可能な有限オートマト

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

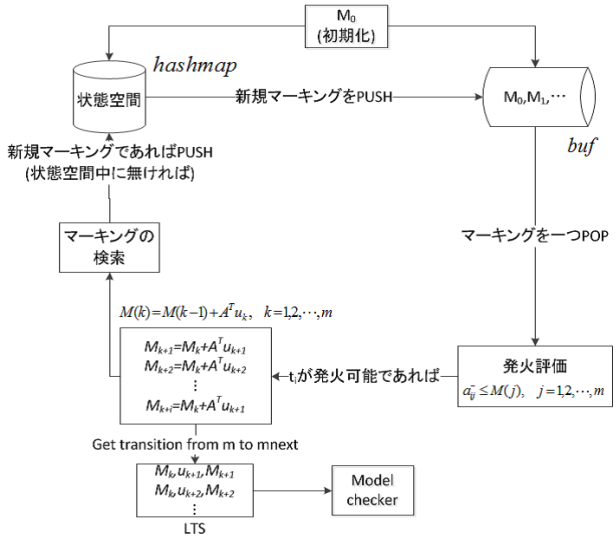

図 1　可達グラフ生成アルゴリズムの動作フロー

ンの一種である Büchi オートマトンへの変換が可能であり [11]，モデル検査ツール SPIN [12] などが採用している．定義**??**に Büchi オートマトンの定義を示す．FLTL を用いた仕様オートマトンの生成法が提案されており [13]，詳細は次節で述べる．

定義 3.1（Büchi オートマトン）

Büchi オートマトン $B = (Q, \Sigma, \delta, q_0, F)$, ここで

Q は，非空な状態の有限集合である．

Σ は，非空なラベルの有限集合である．

$\delta \subseteq Q \times \Sigma \times Q$ は，ラベル付き遷移関係である．

$q_0 \in Q$ は，初期状態である．

$F \subseteq Q$ は，受理状態の集合である．

3.2　イベントに基づく状態空間に対する検査法

LTS はシステムで観察される事象の順序を記述するモデルであるために，事象に関する性質を LTL で記述できることが望ましい．このような性質の記述に，事象によって真理値が定義される述語である流動の概念と，その流動の真偽について定められた LTL である流動 LTL (FLTL) が提案されている [13]．Büchi オートマトンの各遷移は，モデルに現れる状態に関する命題によってラベル付けされている．一方で，FLTL で扱う命題は LTS 上には明示的にラベル付けされていない．よって，FLTL 式より構成した Büchi オートマトンとシステムモデルの両者のラベルの集合は一般に異なるため，モデルより得られる LTS との同期合成が行えない．そこで，Büchi オートマトンを受理状態を含む LTS であるテスタオートマトンへ変換する方法が知られている [13]．

流動は起点となるアクション集合から終点となるアクション集合までのアクション系列について真偽を定めている．流動を等価なオートマトンへ表現することができ，そのオートマトンを fluent オートマトンと呼ぶ．定義 3.2，3.3 に fluent オートマトンと Synchronizer オートマトンの構成をそれぞれ示す．

定義 3.2（fluent オートマトン）

流動 fl を $fl = (I_{fl}, T_{fl}, Initially_{fl})$ とする．

fl についての fluent オートマトン $F_{fl} = (S, A, \delta, s_0)$ を以下のように定める．

$$
\begin{aligned}
Q &= \{q_f, q_t\} \\
A &= I_{fl} \cup T_{fl} \cup 2^{\Phi} \\
\delta &= \{\forall a \in I_{fl} | (q_f, a, q_t), (q_t, a, q_t)\} \\
&\cup \{\forall x \in 2^{\Phi}, fl \not\ni x | (q_f, x, q_f)\} \\
&\cup \{\forall x \in 2^{\Phi}, fl \ni x | (q_t, x, q_t)\}
\end{aligned}
$$

定義 3.3（Synchronizer オートマトン）

Synchronizer オートマトン $Sync_{A_M \Phi} = (Q, A, \delta, q_0)$ を以下のように定める．

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
A &= A_M \cup 2^{\Phi} \\
\delta &= \{\forall a \in A_M | (q_0, a, q_1)\} \\
&\cup \{\forall x \in 2^{\Phi} | (q_1, x, q_0)\}
\end{aligned}
$$

fluent オートマトンと Büchi オートマトン，Synchronizer オートマトンとの同期合成から，言語 2^{Φ} を除去して得たオートマトンであるテスタオートマトン B_{A_M} を次の式 (1) のように構成する．テスタオートマトンは $A_M \subseteq A$ を言語とする無限長の語を扱うことが可能である．

$$B_{A_M} = (F_{fl_1} \parallel \cdots \parallel F_{fl_n} \parallel Sync_{A_M \Phi} \parallel B_{\Phi}) \backslash 2^{\Phi} \quad (1)$$

式 (1) によって得られた同期積オートマトンに存在する 2^{Φ} に関する遷移をすべて除去するために，Synchronizer オートマトンが存在する．

本研究では検査仕様に論理命題として単一のトランジションを対象としているため，構成される fluent オートマトン F_{fl} は全て同一の構造からなる．そのことから，同期合成 $(F_{fl_1} \parallel \cdots \parallel F_{fl_n})$ の構造も流動命題の数 n に応じて，常に同一になる．この事実に着目し，逐次同期合成を行わずに $(F_{fl_1} \parallel \cdots \parallel F_{fl_n})$ を求めるアルゴリズムの考案を図った．全流動命題数を n とすると，全ての流動オートマトン遷移を単純な合成を行うと各 fluent オートマトンは 4 つの遷移を有するため，$\mathcal{O}(4^n)$ の計算コストがかかる．言語 2^{Φ} をビット列によって表現することで，計算の効率化を図った．本ツール実装時には計算量を $\mathcal{O}(n^2)$ とする Algorithm 1 を提案した．

$AList$ と $fluentList$ はそれぞれ A_M と Φ を意味している．構造 $FluentAutomaton$ は，遷移ラベル A_M と 2^{Φ} とそれぞれの行先を組とするマップ構造 Act と $Flue$ を持つ．合成 fluent オートマトンを $FluentAutomaton$ のリストである $fluentAutomatonList$ で表現する．関数 $rPower$ は 2^{Φ} から第一引数を取り除いた集合 (リスト) を返す．Φ のサイズによってはその冪集合である 2^{Φ} は膨大になるため，2^{Φ} を直接計算せずにビット演算のみで部分集合を求めている．関数 $acol$ も同様にビット演算により，要素 i に該当する箇所が否定を示す 2^{Φ}の部分集合を返す．

Algorithm 1 同期合成オートマトン生成法

$FluentAutomaton\ s_0$
$s_0 \rightarrow Flue \rightarrow Add(rPower(true, fluentList), 0)$
$s_0 \rightarrow Act \rightarrow Add(remove(AList, fluentList), 0)$
for each i in fluentList **do**
  $FluentAutomaton\ s$
  $s_0 \rightarrow Act \rightarrow Add(i, i+1)$;
  for each j in fluentList **do**
    $s \rightarrow Act \rightarrow Add(j+1, AList[j])$
  end for
  $int\ n \leftarrow acol(i, fluentList)$
  $s \rightarrow Flue \rightarrow Add(rPower(n, fluentList), i+1)$
  $fluentAutomatonList \rightarrow Add(s)$
end for
$fluentAutomatonList \rightarrow AddFirst(s_0)$

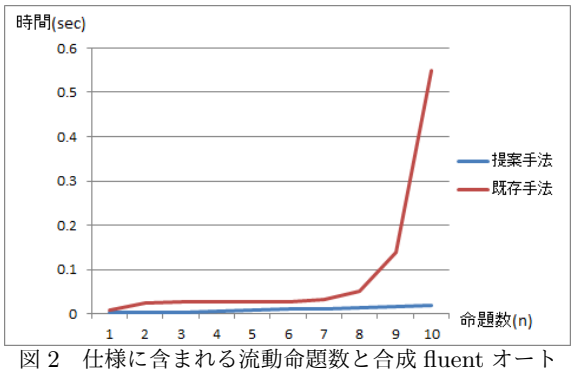

図 2　仕様に含まれる流動命題数と合成 fluent オートマトン生成時間の推移

仕様に含まれる流動の数と合成 fluent オートマトン生成時間の比較を図 2 に示す．横軸は仕様に含まれる流動の数を表しており，縦軸は生成時間に対応している．図 2 では，既存手法では命題数が 8 あたりから急激に増え，命題数が 10 の時点では，0.55 秒かかっている．一方，提案手法では，命題数が 10 の時点で 0.018 秒と $\frac{1}{20}$ 以下にまで抑えられている．この結果から，出現する命題の数が多くなるような複雑な仕様に対して本手法は効果的であると考えられる．

4　優先的発火による状態空間生成法の提案

4.1　優先トランジション

状態空間の生成順序の効率化に関する戦略について提案する．到達可能性といった性質の検証では，クリティカルセッションに入ったかどうかといった，全てのシステムの振る舞いの中の一部分の動作が重要となる．On-the-fly モデル検査では，逐次到達可能な状態空間を受理状態の探索を行うため，受理状態への遷移するパスへの起点となる遷移を含む状態の早期生成によって効率化を図れると言える．前述の状態空間生成アルゴリズムでは，次段マーキングから評価するマーキングの選出は非決定的に行われる．発火評価するマーキングの選出時に，仕様から得られる観測対象とするアクション集合の情報を活用した，状態空間生成アルゴリズムの改良を行った．

観測対象のアクション集合の決定に，検査における優先度 (priority) による順位付けを行う，優先トランジションを導入する．優先度を設定するための指標には，仕様に出現するアクションかどうか，頻出度，仕様オートマトンの受理列解析によるスコアリングなどがある．本研究では，“トランジションが仕様に出現したアクション集合に含まれるかどうか” の二値で分類した．仕様オートマトンの構造によって発火優先度のスコアリングを行えるが，解析コストを考慮し優先トランジション間で優先度の優劣をつけていない．優先トランジションの発火可能性を含むマーキングを検出した場合，その状態以降の部分木を全て生成する．

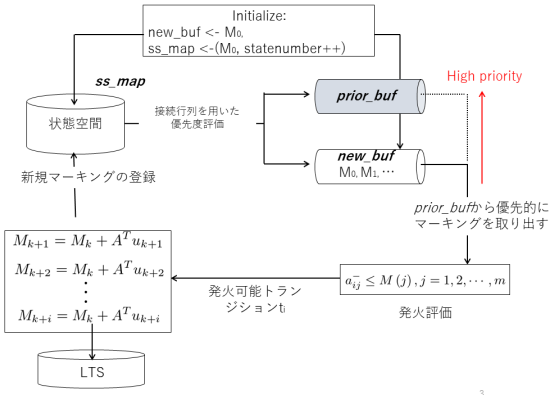

図 3　優先トランジションを用いた状態空間生成アルゴリズムの動作フロー

図 3 は優先トランジションの発火可能性を考慮した状態空間生成アルゴリズムの略図である．既存アルゴリズムにある次段マーキングを格納するバッファとは別に優先トランジションの発火可能性のあるマーキングを格納する $prior\_buf$ を追加した．得られた新規マーキングを次段マーキングへ登録する際に優先度評価を行い，いずれかの次段マーキングバッファへ格納する．$prior\_buf$ から優先的に発火評価処理を行う．優先度評価にはネットの接続行列を用いて，マーキングの優先トランジションの発火可能性を調べる．

4.2　評価

スマートフォンによる遠隔操作オーブンレンジモデルを検査対象とした [14]．図 4 に上記モデルのペトリネットのメインページを示す．このモデルは階層的構造により設計され，複数のユーザ側とオーブンレンジの動作制御部との並列通信を行う．個々のユーザは独立してオーブンレンジの操作が可能で，オーブンレンジの使用状況を確認する ask_status，調理時間や温度といったオーブンレンジの使用に関する設定を送信する set，温めを開始する $start$，停止する $stop$ の 4 つの命令が可能である．サブネットに，ユーザ操作部，内部温度管理部，加熱実行時間管理部，ドアセンサー部，操作設定保存部の 5 つがあり，メインネットである制御部により通信制御及び動作制御を行う．あるユーザが set 命令を送信しオーブンレンジ側でその命令が受理された後の，同一ユーザによる $start$ を保証するために，mutex による動作開始制御を行っている．これを FLTL を用いてこのような仕様を記述すると $\mathbf{G}(user1\_set \Rightarrow \mathbf{F}(user1\_start)) \wedge \mathbf{G}(user2\_set \Rightarrow \mathbf{F}(user2\_start))$ 表現でき，本システムはこれを満たす．

同一ユーザによるリソース占有が起こらないよう

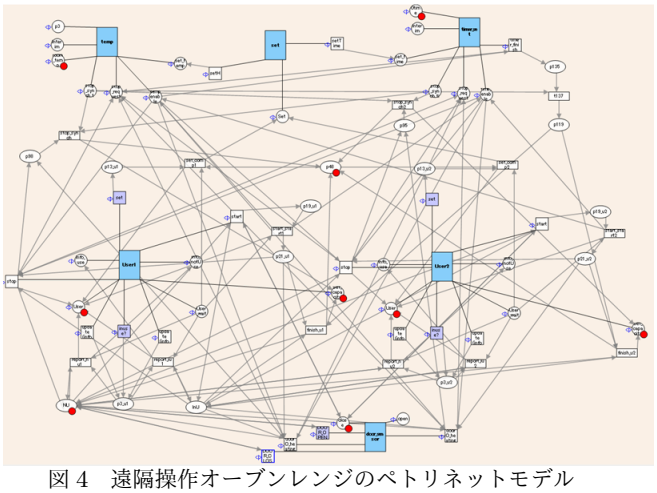

図 4　遠隔操作オーブンレンジのペトリネットモデル

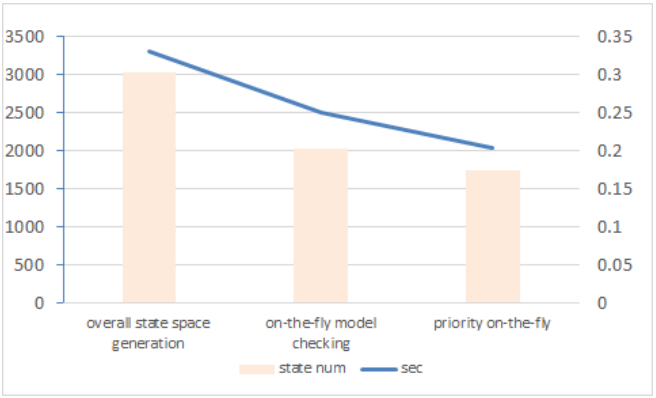

図 5　実行時間比較

な公平性が保障されていることが望ましく，公平性を検証する仕様 $\mathbf{G}(user1\_start \Rightarrow \mathbf{F}user2\_start) \wedge \mathbf{G}(user2\_start \Rightarrow \mathbf{F}user1\_start)$ を与え，検証を行う．本モデルは非有界公平性ネットであることが知られており，この仕様を満たさない．全状態空間生成と，実装済みである既存の On-the-fly モデル検査手法 [15][16] と本提案手法との，実行時間と生成した状態空間数の比較を行った．

表 1　実行環境

CPU	Intel Core(TM) i5-6200U
CPU 周波数	2.3GHz
コア数	2
実装 RAM	8.00GB

全状態数 3,020 と比べ，既存手法は 2,020，提案手法は 1,748 であり，生成数はそれぞれ 34.1%，43.2% 減少する結果となった．実行時間は，既存手法と比較し 19.8% 削減することができた．

5　まとめ

本研究はペトリネット設計ツール HiPS における FLTL モデル検査器の実行時間削減に向けて，いくつかの効率化を図った．遠隔オーブンレンジモデルに対して，状態空間生成時に優先トランジションを導入し仕様に沿った生成順序を踏むことで，既存の On-the-fly 実行と比べ 20% 程効率化が行えた．今回ベンチマークに用いたモデルでは状態数が少なかったが，より大規模モデルに対する本提案手法の評価を行いたい．今後の展望は，提案手法の効率化とより詳細な検証を行うためにマーキング情報を利用したモデル検査の手法の考案である．本手法ではトランジションの優先度評価を接続行列を用いて行っているが，ネットの構造など他のパラメータを考慮した評価法により更なる効率化を図っていく．FLTL モデル検査では，イベントの生起のみに着目しているが，マーキングを状態値とした複合的な仕様記述定義により，詳細な検証が可能になると考える．

参考文献

[1] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541–580, Apr 1989.

[2] 信州大学工学部情報工学科. HiPS : Hierarchical Petri net Simulator. `http://sourceforge.net/projects/hips-tools/`.

[3] Stefan Schwoon and Javier Esparza. A note on on-the-fly verification algorithms. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005 Joint 2005, Notes in Computer Science*, pp. 174–190. Springer, 2005.

[4] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Form. Methods Syst. Des.*, Vol. 1, No. 2-3, pp. 275–288, October 1992.

[5] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[6] Wolfgang Reisig. *Petri Nets: An Introduction*, Vol. 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[7] Michael Westergaard and H.M.W. (Eric) Verbeek. CPN tools. http://cpntools.org/, 2016.

[8] N.J. Dingle and W.J. Knottenbelt. QPN-Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets. http://pipe2.sourceforge.net/, 2016.

[9] 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. 第 12 回情報科学技術フォーラム, FIT2013, pp. 171–174, 2013.

[10] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001.

[11] Paul Gastin and Denis Oddoux. Fast LTL to büchi automata translation. In *Computer Aided Verification, 13th International Conference, CAV 2001, Lecture Notes in Computer Science*, pp. 53–65. Springer, 2001.

[12] Bell Labs. Verifying multi-threaded software with spin. `http://spinroot.com/spin/whatispin.html`.

[13] Dimitra Giannakopoulou and Jeff Magee. Fluent model checking for event-based systems. In *Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003*, pp. 257–266. ACM, 2003.

[14] Yojiro Harie and Katsumi Wasaki. Formal Verification of the Safety Testing for Remote Controlled Consumer Electronics Using the Petri Net Design and Tool: HiPS. In *5th IEEE Global Conference on Consumer Electronics, GCCE2016*, pp. 290–294, 2016.

[15] 張江洋次朗, 和崎克己. ペトリネット検証ツール HiPS における on-the-fly LTL モデル検査器の実装. In *15th Forum on Information Technology, FIT2016*, pp. 109–114, 2016.

[16] Yojiro Harie and Katsumi Wasaki. A Petri Net Design and Verification Platform based on The Scalable and Parallel Architecture: HiPS. In Shahram Latifi, editor, *14th International Conference on Information Technology - New Generations, ITNG 2017, 10-12 April 2017*, pp. 265–273. IEEE Computer Society, 2017.