A-012

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

Realization of “on-the-fly” LTL Model Checker on the Petri Net Verification Tool: HiPS

張江洋次朗 † Yojiro Harie
和崎克己 ‡ Katsumi Wasaki

1. まえがき

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 [1]. ペトリネットモデルに対して，分散アルゴリズムによる情報処理システムの設計・動作検証が行われている [2]．ペトリネットはモデルの振る舞いをトランジションの発火系列として表現でき，トランジションの発火によるモデルの状態の遷移をラベル付き遷移系で記述できる．

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．命題論理は，構文論に基づき体系を定義し，意味論により数学的な意味づけを与える．命題論理の一種である線形時相論理 [3] 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．性質記述において，モデルの振る舞いの仕様における各状態に関しては，その真偽が決定できなくてはならない．そのため，検証性質の適用範囲は振る舞いの仕様記述に依存する．

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 [4]．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．HiPS はモデルの状態遷移について，出力形式を aldebaran 形式とする状態空間を生成する機構をもつ．また，VASY/INRIA によって開発されている CADP[5] の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行える．

本論文は，ペトリネットモデルに対する充足性判定の自動化を目指し，ペトリネット設計ツール HiPS における線形時相論理を用いたモデル検査の機能拡張を提案し，その実装手法を示す．さらに，効率化手法の一つである on-the-fly 検証を導入して，大規模モデルへの実用を図る．

2. ペトリネット

2.1.Place/Transition net

Carl Adam Petri によって提唱された，基本的なペトリネットのことを Place/Transition net (P/T net) という [6][7]．並行的，非同期的，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シミュレートできる．

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

ペトリネットは，プレース，トランジションを頂点とする有向 2 部グラフである．プレース，トランジションに加えて，アークおよびトークンを加えた 4 つの構成要素からなる．ペトリネットの定義は次のようになる．

定義 2.1 (ペトリネットの定義 [1])．ペトリネット $PN = (P, T, F, W, M_0)$ の 5 つ組で定義される．ここで

$P = p_1, p_2, \cdots, p_m$ は，プレースの有限集合．
$T = t_1, t_2, \cdots, t_n$ は，トランジションの有限集合．
$F \subseteq (P \times T) \cup (T \times P)$ は，アークの集合．
$W : F \to 1, 2, 3, \cdots$ は，重み付け関数．
$M_0 : P \to 0, 1, 2, 3, \cdots$ は，初期マーキング．
$P \cap T = \emptyset$ かつ $P \cup T \neq \emptyset$.

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．

多くのシステムの挙動は，システムの状態とその遷移によって記述できる．システムの動的な挙動を解析するために，状態すなわちペトリネットのマーキングを，以下のトランジション発火則に従って得る．

定義 2.2 (接続行列)．P/T ネット (N, M_0) について，$|P| = m, |T| = n$ とする．接続行列 $A = [a_{ij}]$ とは $n \times m$ の整数行列であり，形式的に次のように与えられる．$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$ ここに $a_{ij}^{+} = w(i, j)$ はトランジション i から出力プレース j へのアークの重みであり，$a_{ij}^{-} = w(j, i)$ は入力プレース j からトランジション i へのアークの重みである．

定義 2.3 (発火条件)．P/T ネット (N, M_0) について，$*t = \{p|(p, t) \in F\}$, $t^* = \{p|(t, p) \in F\}$ とする．トランジション $t \in T$ がマーキング M について発火可能であるとは，すべてのプレース $p \in *t$ が $m(p) \geq w(p, t)$ を満たし，かつ，すべてのプレース $p \in t^*$ が $m(p) + w(p, t) \leq k(p)$ を満たすことである．次段マーキング m' へのトランジション t に関する発火とは次のように定義される．$m'(p) = m(p)$ $(p \notin *t \ and \ p \notin t^*)$
もしくは $m(p) - w(p, t)$ $(p \in *t \ and \ p \notin t^*)$
もしくは $m(p) + w(t, p)$ $(p \notin^* t \ and \ p \in t^*)$
もしくは $m(p) - w(p, t) + w(t, p)$ $(p \in^* t \ and \ p \in t^*)$.

可達性はあらゆるシステムのダイナミックな性質を研究する基盤である．発火可能なトランジションの発火は，先述の発火則に従って，マーキングを変化させる．

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

定義 2.4 (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が存在するとき，M_n は M_0 から可達であるという．発火系列は $\sigma = M_0 t_1 M_1 t_2 M_2 \ldots t_n M_n$ と表す．この場合，M_n は σ により M_0 から可達であり，$M_0[\sigma > M_n$ 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 $R(N, M_0)$，あるいは単に $R(M_0)$ と表す．ネット (N, M_0) において，M_0 から始まるすべての発火系列の集合を $L(N, M_0)$，あるいは単に $L(M_0)$ と表す．

ペトリネット (N, M_0) 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, M_0) における可達マーキングの木 (あるいはグラフ) 表現を得る．特にネットが有界であるとき，この (あるいはグラフ) のことを可達 (あるいはグラフ) と呼ぶ．可達グラフの定義は次の通りである．

定義 2.5 (可達グラフ)．可達グラフは次のようなラベル付けされた有向グラフ $G = (V, E)$ である．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る．つまり，V は $R(M_0)$ と等しい．集合 E は辺集合であり，E の元 $\{M_i, M_j\} \in E$ は，トランジション t_k でラベル付けされており，$M_i[t_k > M_j$ であるような単一のトランジション発火の表現である．

可達グラフにおけるノードを状態，トランジションを状態遷移としてみなすことで，モデルに対する状態空間を考えることができる．得られた状態空間はシステムの網羅的な振る舞いを表現しており，状態空間に対して解析を行うことでシステムの振る舞い検証を行える．

2.2.HiPS

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，ペトリネット設計ツール HiPS (Hierarchical Petri net Simulator)[4] がある．ペトリネットにおける性質は初期マーキングに依存する性質と独立な性質に分類される．前者は動的性質と呼び，後者は構造的性質と呼ぶ．HiPS にはペトリネットの性質に基づいた解析機能が実装されている [8]．

2.3.Labelled Transition System (LTS)

LTS は初期状態からのプロセスモデルの振る舞いを表現するグラフ記述言語である．プロセスによるモデル化において “状態値” とは，“その後実行すべきプロセスラベルの列” に過ぎず，プロセスの action による遷移，つまり状態間のラベル付きトランジション (Labelled Transition) の働きが重要となっている．次に LTS の定義を示す．

定義 2.6 (Labelled Transition System (LTS))．Labelled Transition System (LTS) は (S, A, δ, s_0) で構成される，ここで
S: 空でない状態の有限集合
A: 空でないトランジションラベル (振る舞い) の有限集合
$\delta \subseteq S \times A \cup \{\tau\} \times S$: 状態遷移関係 (アーク)
$s_0 \in S$: 初期状態
τ は内部遷移を意味する．

2.4. 生成アルゴリズム

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．状態空間は Labelled Transition System (LTS) による表現で行う．有界な可達グラフ生成アルゴリズムが提案されている [8]．状態空間生成器は，シングルスレッドとマルチスレッドに対する 2 種類のアルゴリズムに基づき実装されている [8]．本研究では初期状態から幅優先探査 (BFS) で逐次生成される状態グラフを必要とするため，シングルスレッドによる状態空間生成を対象とした．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．得られた次段マーキングが状態空間中に含まれていなければマーキングバッファへの登録を行い，終端状態でない新規マーキングに対して逐次的に発火評価を行っていくことで，状態空間の生成を行う．

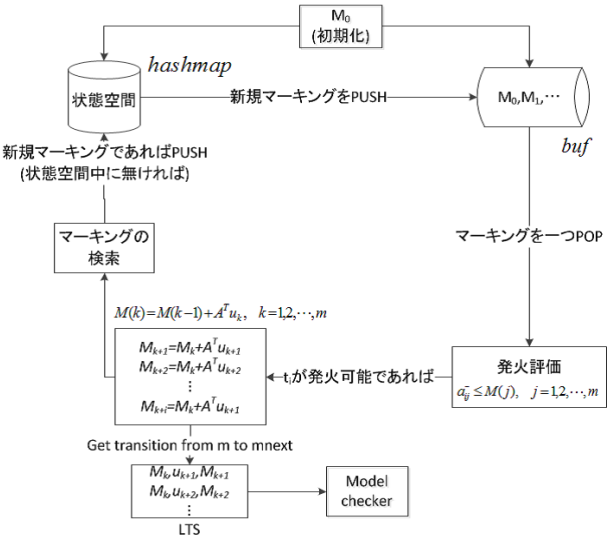


図 1: 可達グラフ生成アルゴリズムの動作フロー

3.LTL モデル検査

本稿では，オートマトンに基づく線形時相論理 (Lnear Temporal Logic: LTL) についてのモデル検査法を前提とする．この方法は，無限長の語を表現可能な有限オートマトンの一種である Büchi オートマトンを基礎とする．システム M_A・仕様 M_φ の双方を Büchi オートマトンで表現し，検証の問題を言語の包含関係として定式化する [9]．LTL を用いたモデル検査は，任意の LTL 式は等価な言語を受理する Büchi オートマトンに変換が可能であるという事実より，以下のようにして行う．

1. φ と等価な Büchi オートマトンに変換する．
2. 得られた Büchi オートマトンと検査対象モデルであるシステムオートマトンとの同期合成オートマトンを求める．

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

3. 同期合成オートマトンを有向グラフとみなし，受理状態とみなすパスを初期状態から網羅的にグラフ探索アルゴリズムを用いて検知を行う．

同期合成オートマトンが空でない場合，受理列が具体的な反例となる．図 2 は簡易的な自動販売機のペトリネットモデルである．このモデルに対して，コインを受理した後にいずれ在庫が補充されるという仕様を与えるとする．トランジション $accept$ はコインの受理を意味しており，トランジション $refill$ は在庫の補充に関するアクションを意味している．求められる安全性についての仕様を式 $\mathbf{G}\ (accept \Rightarrow \mathbf{F}\ refill)$ で与える．次節以降で，事象に基づくシステムにおける状態に関する仕様記述への適用手法について述べる．

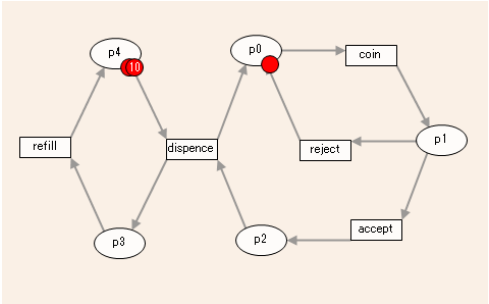


図 2: 自動販売機を表現するペトリネットモデル (在庫 =10)

3.1.on-the-fly モデル検査

モデル検査は全ての状態空間に対する網羅的な検査を行う．検査手法は状態空間生成と状態空間中から反例の検出を行うステップに分けられる．システムの全状態空間生成を行い，得られた全状態空間に探査アルゴリズムによる反例検知を行うようなモデル検査を exhausted モデル検査という．

モデルの規模が大きくなると，状態数が爆発的に増え処理時間が膨大になるという問題がある．大規模なモデルに対して，exhausted モデル検査では，全ての状態空間生成の完了より後に検査を行うために，処理時間が膨大になる．この問題に対する状態空間探索の効率的な手法として，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．ここでは探索アルゴリズムである Nested Depth First Search (Nested DFS) を適用する．

3.2.Nested Depth First Search

Nested Depth First Search (Nested DFS)[10] は，LTL 検査において逐次最適な探索アルゴリズムである．Büchi のオートマトンの受理条件が，受理状態の無限回通過であり，Nested DFS では二重に DFS を受理判定を実施する．はじめの DFS で，初期状態から到達可能な受理状態の探索を行う．受理状態を発見した場合，二番目の DFS でその受理状態から走査をはじめ，閉路の探索を行う．Nested DFS は，受理サイクルが存在する場合，全状態空間を構築する前に探索が終了することが知られている．

図 3 を探査手順の具体例として示す．最初の DFS として，初期状態である S_0 から受理状態の探査を開始し，受理状態 A_0 を発見するまで探査を行う．状態 A_0 を受理状態として検知すると，二番目の DFS プロセスを呼び出し，A_0 へ到達するパスの探索を行う．受理状態へ到達するパスを検知すると，そのパスが与えられた命題に対する具体的な反例となる．図中の状態 A_1 も受理状態ではあるものの，A_1 自身へ至るパスが存在しないために，反例の候補とはならない．

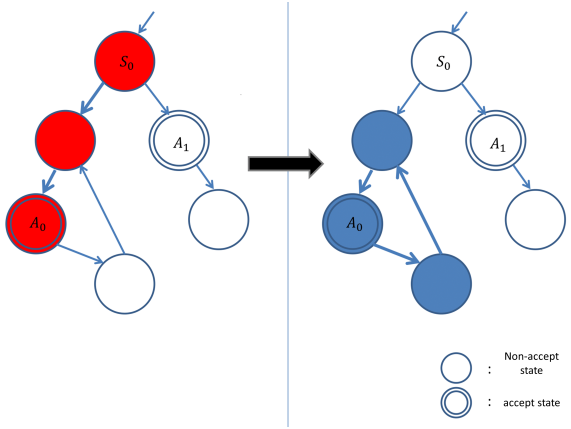


図 3: 同期合成オートマトンに対する Nested DFS を適用した受理状態の走査手順

4.LTS に対するモデル検査

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．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 [11]．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もつ 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 [11]．テスタオートマトンの遷移に関するラベルは対象のモデルに現れる事象で記述されている．

図 4 に検査プロセスの流れを示す．入力フォームより得られた FLTL 式を Büchi オートマトンへ変換を行う．得られた Büchi オートマトンについてテスタオートマトンへ変換する．テスタオートマトンと状態空間を同期合成して得られた LTS について受理列の探査をする．満たすべき性質の否定を式として与えているた

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

め，受理列を検知した場合には，その受理列を反例として結果を示す．

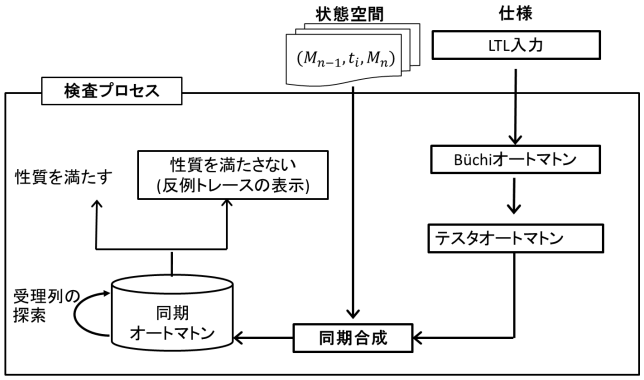


図 4: on-the-fly LTL モデル検査の略式図

4.1. テスタオートマトン

前述の通り，ラベル付が異なるため LTS に対して Büchi オートマトンを合成することはできない．そこで，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．Φ は流動の全集合であり，A はアクション全体の集合である．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．

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

fluent オートマトンはある状態から流動の有無で判別される 2 状態で構成されており，一方の状態は対象の流動を含み，もう一方の状態は流動を含まない．初期状態は流動の初期値により決定され，流動中の開始事象，終了事象により遷移を定めている．つまり，fluent オートマトンは対象の流動の真偽値を 2^{Φ} によるラベル付された自己遷移によって表現している．fluent オートマトンは 2^{Φ} についてラベル付けされる流動に関する自己遷移をもつことから，システムオートマトンとの同期合成 MF も流動に関する自己遷移をもつ．一方，Büchi オートマトンのもつそのすべての遷移は 2^{Φ} についてラベル付けがなされている．性質に関する LTS を作成する工程で 2^{Φ} に関するラベル付を内部遷移へ変換し，さらに内部遷移に関する縮約を行う．上記の各オートマトンのもつ 2^{Φ} に関する遷移を同期合成オートマトンからすべて除去するために，Synchronizer オートマトンが存在する．FLTL 式 $\neg\mathbf{G}\ (accept \Rightarrow \mathbf{F}\ refill)$ より得られるテスタオートマトンを図 5 に示す．

5.HiPS ツールへの実装

5.1.FLTL モデル検査フォーム

モデル検査を HiPS ツール上で実現するために，検査する性質の記述をツール上で行えることが望ましい．キャンバス上の各要素にデザインエントリとして

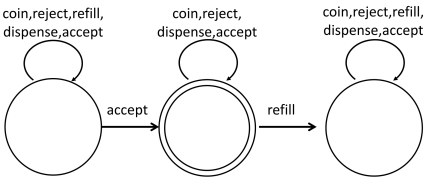


図 5: FLTL 式 $\neg\mathbf{G}\ (accept \Rightarrow \mathbf{F}\ refill)$ を変換して得られるテスタオートマトン

の情報を付与し，FLTL 式の命題として利用するためのユーザインターフェース (UI) を作成した [12]．このフォーム上には，FLTL で用いられる作用素 ([](Global), <>(Future)，X(Next)，U(Until) などの時間演算子と，!，∨，∧ などの古典的な論理演算子) を示すボタンを配置した．ひとつひとつのトランジション毎に関する性質を対象とするために，記述する FLTL に出現する流動は常に単一のアクション (トランジション) によって構成される．任意のトランジション t に関する流動 Fl_t を，単に t というラベル付で記述を行う．

5.2. 対象トランジション表示機能

モデル規模の増大に伴い，ペトリネット内のインスタンス数が増えるために，監視対象のトランジションがイベントとして何を意味しているのかを直感的に理解することが困難になる．そのため，事象に関する性質を記述する際に，設計したモデルの要素 id を参照する方法では，FLTL 式入力の記述コストが高くなるという問題がある．その一例として図 6 を示す．

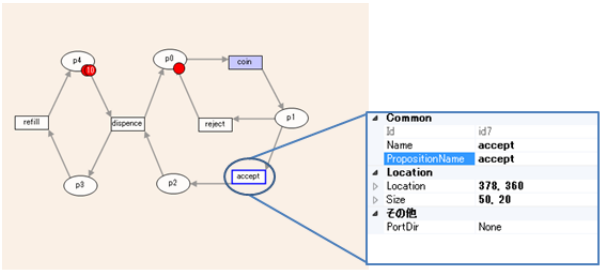


図 6: HiPS ツールにおけるモデル内インスタンスのサブページ表示．自販機モデル中のトランジション *accept* を参照している．

記述コストを下げるため，特定のトランジションを入力フォーム上に自動生成を行い，FLTL 式入力の記述支援を行う．そこで，サブページの要素として監視対象であるかを規定する項目“PropositionName”を付与した．対象とするトランジションのこの項目にネット上から入力があった場合，トランジションの id 等のデータを入力フォーム上にテーブル表示する．表示されたトランジション毎に入力ボタンが付随しており，フォーム上の FLTL 演算子ボタンと併せて記述に利用するこ

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

とで，FLTL 式の記述が容易に行える．図 7 に HiPS に実装した FLTL 入力フォーム画面を示す．図中フォーム上の最上段の式入力部には，自動販売機モデルに対する補充に関するプロパティ$\neg([](accept \rightarrow <> refill))$を与えている．

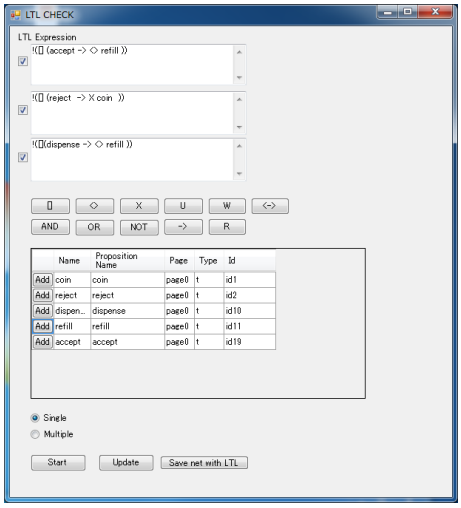


図 7: FLTL 入力フォーム

5.3.LTL2BA

任意の LTL 式は等価な言語を受理する Büchi オートマトンに変換が可能である．LTL 式から等価な Never-Claim への変換を行うツールとして LTL2BA[13] がある．本研究では，C 言語で記述されている LTL2BA を C++へ書き換えた上，HiPS ツールへ組み込みを行った．HiPS への実装のために，Büchi オートマトンとして得られる構造と遷移名について一つにまとめて，C++のコンテナクラスである vector を用いた新しい構造への変換を行った．モデルに対する検査性質の充足性判定を行う場合には，満たすべき性質を表現した論理式の否定形を入力として与える点に注意する必要がある．

5.4.IPC チャンネル

プロセス間通信 (IPC:interprocess communication) はコンピュータの動作において複数のプロセス間でデータをやりとりするための仕組みのことである [14]．メッセージの送受信やスレッドセーフな同期処理や，共有メモリによる情報の共有化などが可能になる．on-the-fly モデル検査の実装に際して，既に HiPS ツール上に実装されている状態空間の生成部と本研究で設計したモデル検査プロセスとの並列処理を行う．モデル検査プロセスと既に実装済みである状態空間生成プロセス間で，IPC チャンネルを用いたプロセス間通信によって on-the-fly 検査を実現した．

5.5. 検証例 :ABP

交番ビットプロトコル (Alternating Bit Protocol: ABP) とは，転送誤りを起こすような片方向通信を行うネットワークプロトコルである [15]．片方向のメッセージ転送を行う際，送信側はメッセージにそのメッセージ番号を交番ビットとして付与する．受信側はメッセージに付与された交番ビットと，受信側が期待した番号との比較を行うことで，受信成功あるいは受信誤りを送信側へ伝え，送信側からの再送処理を行う．

次の図 8 (a) から (d) は ABP を記述した階層型ペトリネットモデルである．このモデルの全状態空間は 94 であり，その状態空間は有界である．

メッセージの送受信が安全に行われたことを確認する仕様として FLTL 式 $(PUT \Rightarrow \mathbf{F}\ GET)$ を与える．反例として，$id14(PUT) \Rightarrow^* [L1\_id0(send)\_id1(put0) \Rightarrow id1.port0(in0) \Rightarrow L1\_id1(medium)\_id11(error0) \Rightarrow id1.port4(error) \Rightarrow L1\_id3(receive)\_id19(error) \Rightarrow id2.port2(in1) \Rightarrow id2.port3(out1) \Rightarrow L1\_id0(send)\_id34(resend0)]$ が得られる (*[] 内をループする)．これは通信路においてメッセージのエラーが起き，再送処理が行わる無限ループ (ライブロック) を常に繰り返すというものである．

6. まとめ

エントリしたペトリネットモデルにおいては再送回数のカウント上限や，送受信イベントのタイムアウト機能は実現されていないため，検知した反例パスは妥当である．HiPS ツールは外部ツールを利用したモデル検査を行っていたが，全状態空間の生成を行う必要があるために大規模なモデルの検査が困難であった．本研究によって，on-the-fly なモデル検査器の導入による空間生成と同時に逐次的な検査を行えるようになった．モデル検査の対象となるトランジションを自動的に記述フォーム上へ情報を出力することで，仕様記述支援を行った．

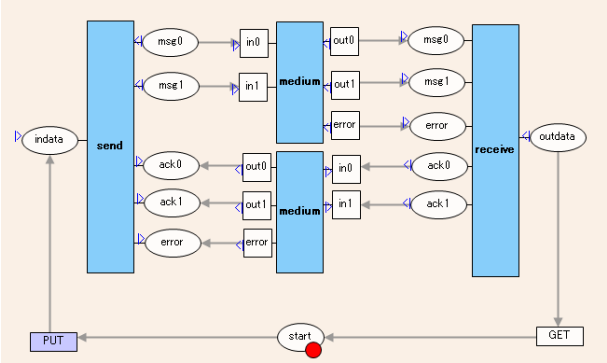
今後は作成したモデル検査器の性能検証を行い，実行速度の向上，メモリ効率の改善に努めたい．システムと仕様に関するオートマトンの同期合成部がボトルネックになるため，仕様オートマトンの縮約機構を実装したい．性能検証と併せて本研究以外の対象モデルの設計・検査を行う．回路モデルや分散システムといった複雑な振る舞いをするモデルに対する検証を行う．

モデル検査への機能の追加として，仕様記述の簡略化とモデル修正のサポートが挙げられる．検証性質にはいくつかの典型的なパターンがあり, 頻繁に用いられる式のリポジトリが行われている．代表的な検証性質をマクロ化した，SpecPattern[16] が考案されており，仕様入力機能への導入を行いたい．モデル修正のための反例分析に，反例の可視化を行うことは重要である．ネットの初期マーキングからトランジションの発火によるトークンの遷移として反例トレースを GUI 上でアニメーションのように確認できる，ガイド付き反例トレースの実行機能を HiPS ツールに追加したい．

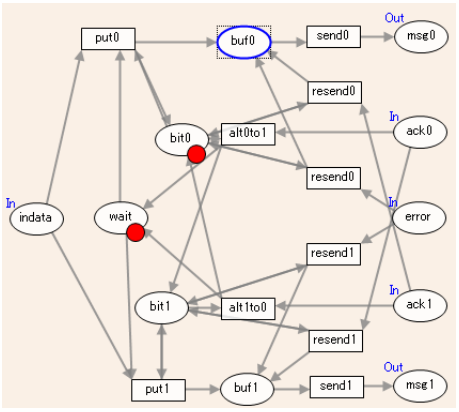
参考文献

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

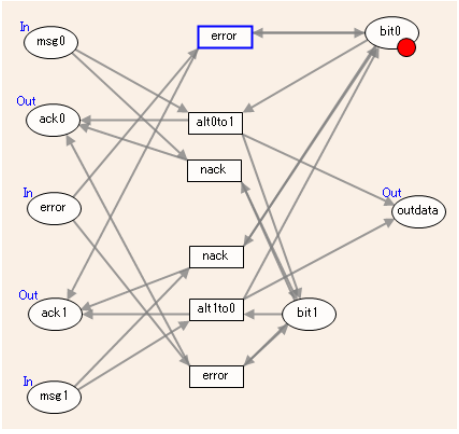
Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.



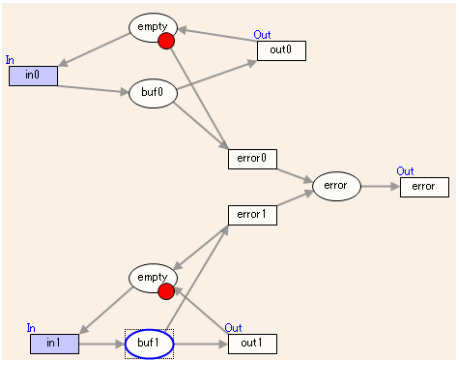
(a) ABP プロトコル



(b) 送信部プロセス (send)



(c) 受信部プロセス (receive)



(d) 通信路 (medium) 上り・下りで共通

図 8: 階層型ペトリネットによる交番ビットプロトコル (ABP) のモデル

[2] 山口真之介, 和崎克己, 師玉康成. 拡張ペトリネットを用いた情報収集の為の分散アルゴリズムの設計. 信学技報 (MSS), 第 109 巻.

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

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

[5] VASY/INRIA. CADP toolbox. `http://cadp.inria.fr/`.

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

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

[8] 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.

[9] 中島震. SPIN モデル検査: 検証モデリング技法. 近代科学社, 2008.

[10] 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, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Computer Science*, pp. 174–190. Springer, 2005.

[11] 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 , 2003*, pp. 257–266. ACM, 2003.

[12] 張江洋次朗, 和崎克己. ペトリネット設計検証ツール HiPS における on-the-fly LTL モデル検査器. 第 14 回情報科学技術フォーラム, FIT2015, pp. 139–142, 2015.

[13] Paul Gastin. LTL2BA. `http://www.lsv.ens-cachan.fr/gastin/ltl2ba/`.

[14] W.R. Stevens. *Unix Network Programming*, Vol. 2. Prentice Hall, 1999.

[15] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, New York, NY, USA, 2nd edition, 2001.

[16] SAnToS laboratory. Spec Patterns. `http://patterns.projects.cis.ksu.edu/`.

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.