モデル検査における妥当性確認を目的とした並行システムの表現手法

竹内 亮太郎　粕谷 英人　大久保 弘崇　山本 晋一郎

並行ソフトウェアシステムのモデル検査においては，モデルが正確で十分な精度を備えていることが求められる．本論文は，モデルの理解を視覚的に支援する，有限状態機械の直観的な表現手法を提案する．対象とするモデルは FSP で，表現手法は LTS 表現に基づいている．提案する 4 種類の図法により，LTS 表現の状態爆発を抑制したり，並行システムにしばしば現れる特徴的な状態遷移を明示的に表現することが可能となる．FSP 記述を提案図法により可視化し，モデルの理解支援を対話的に行うツールを作成した．61 個の例題を用いた評価実験により，適切に表示できることを確認した．

In model checking of concurrent software, a model must have enough accuracy. In this paper, we propose intuitive representation of finite state machine supporting visual understanding of models with graph representation. The target model is FSP, and graph representation is based on LTS. The four visualization methods we propose suppress state explosion of LTS representation and emphasize characteristic state transition patterns which are common in concurrent systems. We made a tool that visualizes models by proposed visualization methods to support understanding of models. In experiment, we apply the methods to sixty one examples. As a result, we confirmed that it could visualize them adequately.

1　はじめに

システムの複雑化に伴い，その信頼性を保証する検査技術の要求が高まっている．とりわけ，並行システムは複数の逐次プロセスが互いに影響しているため，その信頼性を確認することは困難である．並行システムに対する網羅性の高いテストケースを作成しようとすると膨大な数となり，実用的な時間で検査をすることはできない．

この問題を解決する手段としてモデル検査 [1] が注目されている．モデル検査は，(1) システムを有限状態機械としてモデル化し，(2) 満たすべき性質を記述し，(3) システムが (2) で与えられた性質を満たすか否かを検証する，という流れで行われる．

システムのモデル化や満たすべき性質の記述は人手で行われる．その中でもシステムを意図どおりにモデル化することは，モデル検査を行う前提であるため重要な作業である．本論文ではシステムのモデルが意図どおりであることを「モデルが妥当である」という．

通常，システムのモデル化と妥当性の確認は，(1) コンポーネント単位のモデルを作成する，(2) 妥当性を確認する，(3) 並行合成後のモデルに対しても妥当性を確認する，という流れで行われる．性質を満たすか否かの検証に失敗する原因が，合成後のモデルが妥当でないという場合があるため，合成後のモデルに対する妥当性確認は必要である．しかし，並行に動作するプロセスの各処理が任意の順番で実行されるため振る舞いを確認することが困難である．

本研究の目的は，モデルの妥当性確認を表示によっ

Visualization Method of Concurrent Systems for Validation in Model Checking.
Ryotaro Takeuchi, 愛知県立大学大学院情報科学研究科, Graduate School of Information Science and Technology, Aichi Prefectural University.
Hideto Kasuya, Hirotaka Ohkubo, Shinichiro Yamamoto, 愛知県立大学情報科学部, School of Information Science and Technology, Aichi Prefectural University.
コンピュータソフトウェア, Vol.28, No.1 (2011), pp.293–299.
[研究論文 (レター)] 2010 年 4 月 30 日受付.

て支援することである．妥当性確認が困難な，システムの並行動作部分の理解を支援するためにモデルの表現手法を提案する．

2 モデル検査と妥当性確認

本章では既存のモデル検査ツールを紹介する．モデル検査における妥当性確認と，それを行うために必要な表現手法を既存ツールの特徴とともに議論する．

2.1 モデル検査における妥当性確認

モデル検査における妥当性確認とは，モデルが作成者の意図どおりであるかを確認することである．モデルが妥当であるかを確認する方法として，モデルの振る舞いをシミュレートすることや，モデルを直観的に理解しやすいグラフとして表示することが有用である．しかし，並行システムにおいては，逐次プロセスのモデルをグラフとして表示するだけでは並行動作部分を理解するには不十分である．並行合成することで得られるシステム全体のモデルをグラフとして適切に表示する必要がある．

並行システムの動作理解の上で重要な特徴として，動作が任意の順で現れるインタリーブと，共有資源の利用で発生するクリティカルセクションがある．

インタリーブ

並行に動作するプロセスの各処理が任意の順番で実行されることである．各逐次プロセスの処理がインタリーブすることで並行システムの動作が理解しにくくなる．そのため各逐次プロセスの動作がインタリーブしている箇所を正確に理解する必要がある．

クリティカルセクション

並行に動作している各プロセスが同時期に共有資源を使用することを禁じられた部分である．クリティカルセクションでは排他制御を行う必要がある．クリティカルセクションはシステムの重点検査箇所であるため把握することは重要である．

2.2 Labelled Transition System Analyzer

LTSA (Labelled Transition System Analyzer) [2] は Jeff Magee らによって開発されたモデル検査ツールである．対象システムは LTSA 専用の言語である FSP (Finite State Process) [3] で記述される．LTSA は FSP で記述したモデルを LTS (Labelled Transition System) として表示する．LTS は状態を表すノードと，ある状態から次の状態への遷移を表すエッジから構成される．エッジには状態遷移が起こるためのアクションがラベルとしてつけられる．LTS は有向グラフと見ることができる．

```
RESOURCE = (acquire->release->RESOURCE).
USER = (printer.acquire->use->
        printer.release->USER).
||PRINTER_SHARE = (alice:USER||bob:USER||
                   {alice,bob}::printer:RESOURCE).
```

図 1 プリンタシステムの FSP

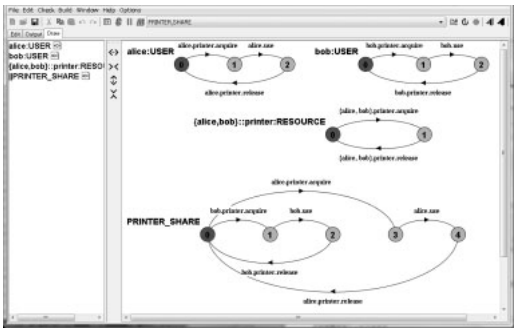

図 2 LTSA

例 1 (プリンタシステム) 2 人のユーザ `alice` と `bob` が 1 台のプリンタを共有するシステムをプリンタシステムとする．プリンタの共有は排他的に制御される．ユーザはプリンタを使用する権利を要求し，権利を確保できたら使用し，終わったらプリンタを解放する．対応する FSP を図 1 に示す．

例 1 の LTSA による表示を図 2 に示す．LTSA はノードを直線上に配置する．上の 2 つのグラフはユーザのモデル，中心のグラフはプリンタのモデル，一番下のグラフはシステム全体のモデルを表す．システム全体のグラフは，各逐次プロセスのモデル (以降，各逐次モデルと表記) を並行合成することで得られる．

LTSA にはモデルの振る舞いをシミュレートするアニメータ機能が備わっている．現在の状態を表すノードと直前のアクションを表すエッジを赤くすることで強調している．実行可能なアクションを選択すると

状態が遷移し，各逐次モデルのグラフ (以降，各逐次グラフと表記) の対応するノードとエッジの色が変わる．しかし，並行モデルのグラフ (以降，並行グラフと表記) はアニメータが機能しない．

2.3 Spin

Spin[4] は Holzamann らによって開発されたモデル検査ツールで，検査対象のシステムは Promela(Process Meta Language) 言語で記述される．作成したモデルの振る舞いをシミュレートする機能がある．Xspin は Spin の GUI フロントエンドで，シミュレーションの進行をシーケンス図で表す．

2.4 UPPAAL

UPPAAL[5] はスウェーデンの Uppsala 大学とデンマークの Aalborg 大学によって共同開発されたモデル検査ツールで，GUI でモデルを作成できる．ノードは状態を表し，エッジは状態から状態への遷移を表す．並行グラフを表示する機能はない．モデルの振る舞いをシミュレートする機能があり，シミュレーションの進行はシーケンス図で表される．

2.5 既存ツールにおける表現の問題点

並行システムの特徴であり理解を妨げる要因として 2.1 節で述べたインタリーブ，クリティカルセクションがあげられる．しかし，既存ツールによるモデルの表現では，上記のパターンを適切に表現できていない．例えば LTSA による表現では，ノードを直線状に並べるため処理の順番を把握しにくく，インタリーブしている遷移が直観的に把握しにくいという問題がある．また実用的なシステムをモデル化すると，インタリーブによって状態数が爆発的に増加し，理解しにくいモデルとなってしまう．UPPAAL や Xspin は並行グラフを表示する機能がないため，システム全体の動作を把握しにくい．そのため，インタリーブしている動作やクリティカルセクションを直観的に把握しにくい．2.1 節で述べた 2 つに加えて，以下の 2 つもシステムの動作理解が困難になる要因である．

プロセスが同じ動作を複数回繰り返す場合

同じ動作を繰り返している箇所を全て表示すると，グラフ全体において繰り返し部分の占める割合が高くなり，グラフ全体が見にくくなるという問題がある．

プロセス中のサブプロセスがモデルの動作を理解する上で重要である場合

モデルを作成する際に，プロセスの概要を表すダイアグラムを記述して，詳細をサブプロセスとして別のダイアグラムに記述するという方法がとられることがある．そのように作成されたモデルを理解するには，大まかな動作を理解した上で詳細な動作へ理解を進めるのがよい．しかし，そのための表示を支援する機能は既存ツールには見つからなかった．

3 妥当性確認のための表現手法

2.5 節で述べたように，既存ツールによるモデルのグラフ表現ではグラフが直観的にわかりにくく，意図とモデルを対比させて妥当性を確認することが難しい．本章では，モデルの妥当性を確認しやすくするための表現手法として，モデルの特定構造に適した表現手法を提案する．状態遷移機械としてモデル化した後のモデルについて考えているため，記述体系から独立になり，各種モデル検査系に適用可能である．

3.1 モデルのグラフの特定構造に適した表現

モデルのグラフで，そのままの表現では理解が困難な特定の構造について，それぞれの構造に適した表現手法を提案する．以下の 4 つの表現手法を考える．

3.1.1 インタリーブパターン

並行に動作している遷移をまとめる表現手法をインタリーブパターンとする．インタリーブパターンの適用箇所は，合成したプロセスの全逐次プロセスが同期している遷移から，次の全逐次プロセスが同期している遷移までの並行動作箇所とする．並行動作箇所に同期している遷移はない．また，初期状態から最初の同期遷移が起こるまでは並行動作箇所なので，インタリーブパターンの適用箇所とする．並行動作箇所で遷移が終わる場合もある．インタリーブパターンの適用例として例 2 をあげる．

例 2 (P_ABC) 逐次プロセス P_A,P_B,P_C によって構成される並行システムを P_ABC とする．P_A,P_B,P_C

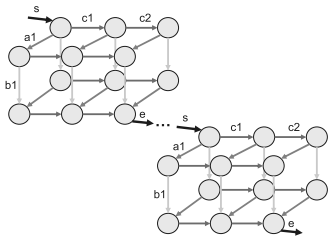

図 3　P_ABC の状態遷移図

```
P_A=(..->s->a1->e->..->s->a1->e->..).
P_B=(..->s->b1->e->..->s->b1->e->..).
P_C=(..->s->c1->c2->e->..->s->c1->c2->e->..).
||P_ABC = (P_A||P_B||P_C).
```

図 4　P_ABC の FSP

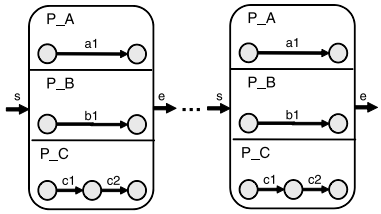

図 5　インタリーブに着目した表示

は，それぞれ並行して起こるアクションとして a1,b1,c1 と c2 を持つ．同期して起こるアクションとして s,e を持つ．P_ABC のインタリーブは，実際に合成した状態空間では図 3 のように 2 箇所となる．FSP を図 4 に示す．‥ は省略を表す．

素朴な図 3 の表現は状態が指数的に増加する．これを抑える為に図 5 のような表現を提案する．並行システムの並行動作箇所を角丸長方形で表し，角丸長方形内を実線で区切って各逐次プロセスの動作を表示する．この表現は UML のステートマシン図を参考にしている．図 5 の表現では P_A と P_B と P_C の組み合わせが並行システムの状態を表す．このように表現することにより，煩雑なグラフとなることを防ぐことができる．

3.1.2　アノテーションパターン

逐次プロセスのアノテーションを合成プロセスに反映する表現手法をアノテーションパターンとする．アノテーションとして逐次プロセスの強調したい状態を付与して合成プロセスに反映することで，合成プロセスのグラフ表現において直観的に逐次プロセスの強調したい状態を把握することができる．クリティカルセクションなどで有用である．アノテーションパターンの適用例として例 1 のプリンタシステムをあげる．

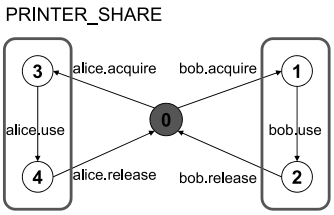

図 6　クリティカルセクションに着目した表示

例 1 を LTSA で表示すると図 2 となり，一番下のグラフがシステム全体のモデルを表す．クリティカルセクションであるプリンタを使用している箇所を把握しづらい表示である．アノテーションとして共有資源であるプリンタを使用している状態を付与し，それを合成プロセスに反映した表示が図 6 である．アノテーションとして付与した状態を枠で囲み明示する．図 6 では共有資源を要求するアクションと解放するアクション，クリティカルセクション部のアクションを把握しやすくなっている．

モデル作成者は元々，図 6 のようにモデルを考えているが，既存ツールではクリティカルセクションを考慮した表示をしない．そのためクリティカルセクションがわかりにくい．アノテーションとして共有資源を使用している状態を付与してクリティカルセクションを把握しやすくしている．状態 0 は共有リソースがすべて解放されている状態なので囲まれていない．クリティカルセクションが複数ある場合はクリティカルセクション毎に異なる枠の色にすることで区別できる．

3.1.3　繰り返しパターン

同じ入出力の遷移を繰り返している箇所に着目する表現手法を繰り返しパターンとする．同じ遷移の繰り返しを表示すると，グラフ全体において繰り返し部分の占める割合が高くなってしまう．そのため，表示スペースが狭くなり，グラフが見にくくなる．そこで繰り返している部分を畳み込んで表示することによ

り，繰り返し処理している部分を把握しやすくする．また，畳み込むことにより，繰り返している箇所以外の表示スペースが広くなり，グラフが見やすくなる．

初期状態のノードから順番に見ていき，同じ入出力の遷移をするノードが 2 つ以上連続していたら繰り返し箇所とし，終端ノード，もしくは初期状態になったら終了する．以上のようにグラフの構造から繰り返している部分を判断できる．

3.1.4 階層構造パターン

プロセス中のサブプロセスに着目する表現手法を階層構造パターンとする．大規模なモデルを作成する際に，プロセスの概要を表すダイアグラムを記述し，詳細をサブプロセスとして別のダイアグラムに記述することが多い．そのような場合，プロセスの概要と詳細の両方がわかる表示方法が望ましい．LTSA は FSP の記述にサブプロセスを使用できるが LTS に変換するとその情報がなくなる．そこでサブプロセスの情報を表示することでプロセスの大まかな動作が把握しやすくなると考えた．このパターンの適用例として文献 [6]8 章のクルーズコントロールシステムをあげる．

例 3 (クルーズコントロールシステム) クルーズコントロールシステムとは，自動車のアクセルペダルを踏み続けることなくセットした速度を維持するシステムである．クルーズコントロールシステムの動作を以下に示す．

- エンジン動作中に on ボタンが押されると，現在の速度を記録し，その速度を維持する
- アクセル，ブレーキ，off ボタンが押されると，速度維持を中止するが，その設定された速度は記録しておく
- resume ボタンが押されると記録した速度を維持する

このシステムは 5 つのプロセスでモデル化される．ここでそのうちの 1 つである Cruise controller に着目する．Cruise controller には INACTIVE，ACTIVE，CRUISING，STANDBY のサブプロセスがある．Cruise contoller の振る舞いを以下に示し，対応する FSP を図 7 に示す．

- INACTIVE のときにエンジンをかけると，clearSpeed イベントがおき ACTIVE になる
- ACTIVE のときに on ボタンが押されると現在の速度を記録し，速度維持を開始し CRUISING になる
- CRUISING のときに off ボタン，ブレーキ，アクセルが押されると速度調整を中断し STANDBY になる
- 任意のサブプロセスでエンジンをオフにすると INACTIVE になる

```
set DisableActions = {off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,
INACTIVE=(engineOn -> clearSpeed -> ACTIVE
         |DisableActions -> INACTIVE
         ),
ACTIVE  =(engineOff -> INACTIVE
         |on->recordSpeed->enableControl->CRUISING
         |DisableActions -> ACTIVE
         ),
CRUISING=(engineOff -> disableControl -> INACTIVE
         |DisableActions->disableControl->STANDBY
         |on->recordSpeed->enableControl->CRUISING
         ),
STANDBY =(engineOff -> INACTIVE
         |resume -> enableControl -> CRUISING
         |on->recordSpeed->enableControl->CRUISING
         |DisableActions -> STANDBY
         ).
```

図 7 クルーズコントロールシステムの FSP

例 3 の Cruise contoller を LTSA で表示すると図 8 になる．サブプロセスの情報は正当性の検査に必要でないため削除される．よって LTSA ではサブプロセスを把握しにくい．しかしサブプロセスに応じて速度維持が作動するときの判断が変わるので，妥当性を確認する上でサブプロセスを把握することは重要である．サブプロセスを反映させた表示が図 9 である．サブプロセスに対応するノード群を枠で囲むことで，そのノードが属するサブプロセスを明示する．図 8 と比較するとサブプロセスを把握しやすくなっている．

4 実装

本稿で提案する表現手法に基づいた並行システムの直観的な可視化を行うツールを実装した．Java 言語によって実装し，描画ライブラリとして Grappa を利用した．本ツールは FSP を入力とする．JavaCC [7] を用いて FSP の解析器を作成した．本ツールの画面

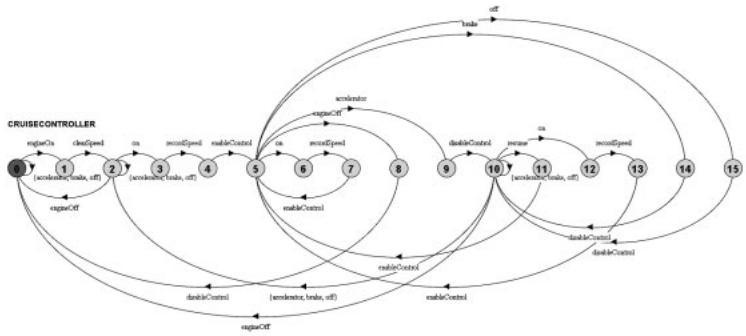

図 8　LTSA による CruiseController の表示

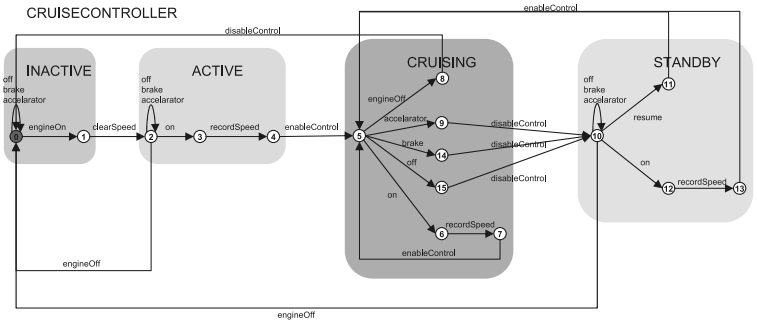

図 9　階層構造に着目した CruiseController の表示

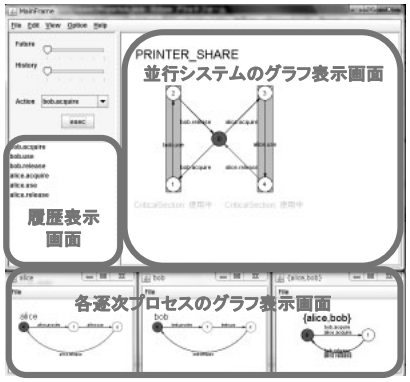

図 10　本ツールの画面構成

構成を図 10 に示す．並行システムのモデルのグラフ表示画面，各逐次プロセスのグラフ表示画面，履歴表示画面で構成される．3.1 節に示したモデルのグラフの特定構造に適した表現を行う．ツールのパターン表示機能に関する詳細は文献 [8] を参照されたい．

表 1　比較結果

評価項目	本ツール	Xspin	UPPAAL	LTSA
シミュレーションが可能	○	○	○	○
各逐次グラフを表示	○	×	○	○
並行グラフを表示	○	×	×	○
特定構造に適した表示が可能	○	×	×	×

5　評価

本ツールと既存ツールについて，モデルの妥当性を検査する次の 4 つの機能を比較した．比較するツールは，2 章で紹介した LTSA，Xspin，UPPAAL とする．比較結果を表 1 に示す．

1. 作成したモデルに対して振る舞いのシミュレーションを行う機能
2. 並行システムを構成する逐次プロセスのモデルをグラフとして表示する機能
3. 並行システムのモデルのグラフを表示する機能
4. 3.1 節で述べたパターンを表示する機能

表 2 適用結果

章	適用数			
	インタリーブ	アノテーション	繰り返し	階層構造
3 章	2	2	0	1
4 章	0	1	2	2
5 章	0	1	5	0
6 章	0	5	0	0
7 章	0	7	0	1
8 章	0	0	0	2
9 章	1	0	0	0
10 章	0	0	5	1
計	3	16	12	7

本ツールは 3 章で提案した特定構造に適した表示を行える．また，モデルの振る舞いをシミュレートすることが可能である．現在の状態と直前のアクションを表すエッジを赤くすることで強調し，モデルの状態遷移を視覚的に把握しやすくしている．並行グラフも色が変わるようにしているので，システム全体の動作も把握しやすい．以上より，LTSA と UPPAAL によって出力されるモデルのグラフより理解しやすく，妥当性の確認を行いやすいと考える．

また本手法は，自動的にパターンを発見して表示を支援する枠組みを提供するため，モデル作成者が意図していない振る舞いを発見しやすくなる．例えばインタリーブパターンでは，モデル作成者が意図せずにインタリーブしている箇所にも適用されるので，意図せずインタリーブしている箇所を発見できる．

文献[6] の 3 章から 10 章までに出てくる例題 61 個への適用結果を表 2 に示す．2 章は逐次プロセスの例しかないため省略とした．全部で 38 箇所にパターンが適用できた．しかし 1 つのグラフに対して複数のパターンが適用できるため，適用数は多くはない．これは非常に簡単な例題が多いことなどが理由に挙げられる．排他制御に関する例題が比較的多かったため，アノテーションパターンの適用数が最も多くなった．

6 おわりに

本研究では，モデル検査において妥当性を確認しやすくするための表現手法を議論した．そして，妥当性を確認しやすくするためにモデルのグラフの特定構造に適した表現方法を提案し，可視化を行うツールを実装した．具体的にはインタリーブやクリティカルセクションなどの特定構造に着目し，それぞれの構造に適した表現を選択することにより，モデルのグラフを直観的に理解しやすくできることを示した．その結果，モデルの妥当性を確認しやすくなったと考える．

本稿では 4 つの構造に特化した表現を提案したが，より多くの構造に着目したり，複数のパターンを組み合わせた場合の表現を考案することで，よりモデルの妥当性を確認しやすくなると考えられる．インタリーブパターンの適用箇所は，すべての逐次プロセスが同期してから再び全ての逐次プロセスが同期するまでとしているが，一部のプロセスが同期するような場合も扱えるようにすることで，より多くのシステムを理解しやすくなる．以上を今後の課題とする．

参 考 文 献

[1] Clarke Jr., E. M., Grumberg, O. and Peled, D. A.: *Model Checking*, MIT Press, 1999.

[2] LTSA - Labelled Transition System Analyzer, http://www.doc.ic.ac.uk/ltsa/.

[3] FSP notation, http://www.doc.ic.ac.uk/~jnm/LTSdocumention/FSP-notation.html.

[4] Holzmann, G. J.: The Model Checker SPIN, *IEEE Trans. Soft. Engin.*, Vol. 23, No. 5 (1997), pp. 279–295.

[5] Bengtsson, J. and Larsson, F.: UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems, DoCS Technical Report Nr 96/67, Uppsala University, ISSN 0283-0574, January 1996.

[6] Magee, J. and Kramer, J.: *Concurrency: State Models & Java Programming*, Wiley, 2008.

[7] JavaCC (JavaCompilerCompiler) Java Parser Generator, https://javacc.dev.java.net/.

[8] 竹内亮太郎, 粕谷英人, 大久保弘崇, 山本晋一郎：モデル検査における妥当性確認を目的とした並行システムの表現手法，ソフトウェア工学の基礎 XVI, 近代科学社, 2009, pp. 119–130.