

調停者の選出方法を考慮した分散合意アルゴリズムの PROMELA モデルと検証

後藤 亮馬¹ 和崎 克己²

概要：合意問題は分散システムの研究分野において重要な問題の一つであり、非同期分散アルゴリズムにおいては、合意問題が解決されているかを確認することが困難である。本研究では、非同期分散アルゴリズムの正当性を、モデル検査によって検証することを目的として、モデル検査器 SPIN を用いて、非同期分散合意アルゴリズムである Chandra-Toueg アルゴリズムの正当性を検証する。SPIN とは通信プロトコルの検証を目的として提案されたモデル検査ツールであり、使用記述言語 PROMELA で記述された振る舞いを、網羅探索により検査することができる。モデルの作成を行う際には、PROMELA 本来の記述方法では困難であった表現を行うために、グローバルタイムアウト機構などのいくつかの工夫を行った。Chandra-Toueg アルゴリズムで使用されている巡回調停者方式では、調停者が健全か否かを考慮しない。また、実際には調停者変更を行った後、ノード間でのリセット動作などの同期を取るべきである。以上二つの理由から、調停者変更時に健全性の判定機能を追加し、調停者変更後に同期をとる選出方法に変更した。作成した通常モデルと拡張モデルを使用してデッドロック検査を行い、正当性を検証した。

1. はじめに

合意問題は、分散システムの研究分野において、最も重要な問題の一つである [1][2]。特に、非同期システムにおいては、複数のノードが任意の順序、速度で実行されるため、手動でその全ての振る舞いを把握し、合意問題が解決しているかを検査することは非常に困難である。そこで本研究では、非同期分散アルゴリズムの正当性を、モデル検査によって検証することを目的としている。

モデル検査とはシステムの振る舞いをモデル化し、そのモデルを網羅的に探索することで、そのシステムの仕様に含まれないような不正な状態を検出する手法である。本研究ではモデル検査器の一つである SPIN[3][4][5] を用いて、Chandra-Toueg アルゴリズム [6] を検証する。Chandra-Toueg アルゴリズムは、非同期分散合意アルゴリズムである。通信ノードの中から調停者と呼ばれる特殊なノードを選出し、調停者の行動を中心に、それぞれのノードがラウンドと呼ばれる一連の動作を繰り返す間に合意に至る。合意問題において満たすべき条件の内、合意性と妥当性を満たす。

この分野の従来研究としては、南川 [7]、松尾 [8] らの研究例がある。特に、南川らの研究において、非同期分散シ

ステムにおける合意アルゴリズムの正当性検証を目的に、新たな記述言語の提案が行われているが、時間の概念そのものの再現は行われておらず、本来の時間を考慮した検証方法は想定されていない。勝山 [9] による従来研究では、本研究と同じく Chandra-Toueg アルゴリズムのモデル化が行われているが、時間の概念の考慮がなかったために、アルゴリズムとは無関係な回避不可能なデッドロックが検出された。

本報告では、グローバルタイムアウト機構を導入することで、PROMELA の記述では困難であった時間の概念の考慮を可能とし、正確な振る舞いモデル (4.3 節で詳述) の作成を提案する。グローバルタイムアウト機構とは、通常動作するプロセスとは別に特殊なプロセスを作成し、PROMELA の予約語である timeout と atomic を利用して、各ノードに割り当てたタイムアウトカウンタを一律に減らしていくことで、擬似的にタイムアウト処理を再現するものである。

本報告は次の通り構成される。まず第 2 節では SPIN によるモデル検査について述べる。第 3 節では分散合意アルゴリズムについて述べる。第 4 節では Chandra-Toueg アルゴリズムの PROMELA モデルについて述べ、第 5 節では合意アルゴリズムの検証について述べる。最後に第 6 節ではまとめと今後の課題について述べる。

¹ 信州大学大学院理工学系研究科情報工学専攻

² 信州大学工学部情報工学科

2. SPINによるモデル検査

2.1 モデル検査とは

モデル検査は仕様が正しく動くことを検証する方法の一つである。対象となるシステムの仕様の振る舞いをモデルとして記述し、要求される性質を論理式で記述する。モデル検査器によって、初期状態からとりうる状態全てを網羅的に検査することで、要求通りに仕様が振舞っていることを検証する手法である。この検査法は、設計段階から適用することができるので、不具合の早期発見ができ、開発コスト削減にもつながる。いくつかのモデル検査器が存在しているが、本研究ではSPIN(Simple Promela INterpreter)を使用する。

2.2 モデル検査器 SPIN

モデル検査器SPINはAT&Tベル研究所(現:カリフォルニア工科大学NASA ジェット推進研究所)のG.J.Holzmann氏によって、通信プロトコルの検証を目的として提案されたモデル検査ツールである。検査するモデルの振る舞いはPROMELA(PROtocol/PROcess MEta LAnguage)という専用の仕様記述言語を使って記述する。検証を行った結果、誤りがあった場合はその反例が出力される。具体的にSPINで検査できる性質は以下のとおりである。

- 表明
振る舞い記述の中に、その振る舞いにおいて成立すべき条件を、表明として記述する。SPINは表明に記述された条件を検査する。
- End ラベル
プロセスが停止しても良い箇所に、End ラベルを記述する。このラベルが記述された箇所は、正しい停止箇所として扱われる。SPINはこのEnd ラベル以外で、プロセスが終了していないか検査する。
- Progress ラベル
モデルにおいて、頻繁に実行すべき箇所に、Progress ラベルを記述する。SPINはこのラベルが記述された箇所が、有限回しか実行されないことが無いことを検査する。
- Never Claim
モデルにおいて、常に成立してはならない性質を記述する。LTL 式 (Linear Temporal Logic:線形時相論理式) を用いて記述することも可能で、この場合はSPINがLTL 式をNever Claimに自動変換する。SPINはNever Claimに一致する性質が存在しないことを検査する。

2.3 仕様記述言語 PROMELA

PROMELAはC言語を基としたSPINの専用仕様記述言語である。SPINでモデル検査を実行する際には、まずこの言語を使用して、モデルを記述する。並行して動作するプロセスや、非決定的な振る舞いの記述が容易な言語仕様となっている。以下にこの言語において特徴的な、不可分実行について記述する。

不可分実行とは、ある振る舞いを常にまとめて実行することである。不可分実行の記述を使用することで、他のプロセスに行動順を回さず、一つのプロセスだけ行動し続ける、というような振る舞いを表現することができる。PROMELAにおいて不可分に実行したい振る舞いの記述には`atomic`ブロックか`d_step`ブロックを使用する。

```
d_step{ 処理 1; 処理 2; }  
atomic{ 処理 1; 処理 2; }
```

`d_step`ブロックでは、記述された処理1と処理2の間に他の処理を行わず、不可分に実行される。処理内容が実行できないものだった場合、エラーとして検出される。

`atomic`ブロックも、`d_step`ブロックと同じように不可分に実行されるが、処理内容が実行不可能だった場合でもエラーとはならず、実行が停止し、他の処理を実行する。つまり、`d_step`ブロックでは一連の処理を必ず不可分に実行し、`atomic`ブロックでは実行できるところまでを不可分実行する。

3. 分散合意アルゴリズム

3.1 合意問題

合意問題とは、合意に至る際に正常に動作しない故障プロセスが存在していても一つの値に決定できるか、というものである。分散システムにおいて合意問題は、非常に重要な問題の一つである。

プロセスの故障には、クラッシュ故障、オミッション故障、ビザンチン故障の三種類があるが、通常システムに要求される信頼性を考慮した場合、ビザンチン故障が起こる可能性は事実上無視できることが多い。また、ネットワーク上で使用される分散システムを考えた場合、非同期システムがより適切であると考えられている。本研究では、プロセスの故障ケースをクラッシュ故障に限定し、非同期分散システムにおける合意問題を解くアルゴリズムを対象に検証を行う。

3.2 非同期システムにおける合意問題

合意問題において、具体的に満たすべき条件は以下の3つである。

停止性：全ての正常プロセスが一つの値を決定し停止する

合意性：全ての正常プロセスの決定した値は同じである

妥当性：決定した値を提案したプロセスが存在する

非同期システムでは、一つのプロセスのクラッシュ故障を考えた場合ですら、上記の条件全てを満たすアルゴリズムは存在しない。なぜなら、非同期システムでは処理が遅れているプロセスと、故障しているプロセスを見分けることができないからである。図1はタイムアウトによる故障検出を表している。生存メッセージの到着が遅れることで、故障を誤検出している。

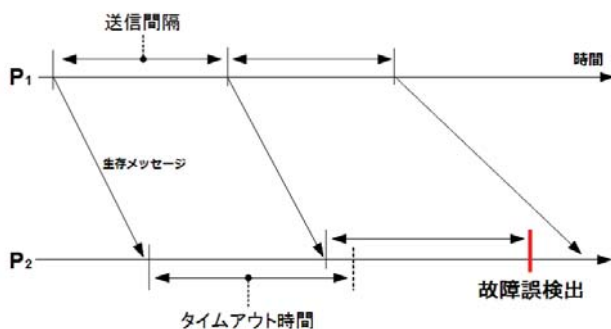


図1 タイムアウト処理による故障誤検出

このように、故障検出が不正確という理由で、停止性を満たすアルゴリズムは存在しない。しかし、故障検出が不正確な場合でも、合意性と妥当性を満たすアルゴリズムは存在する。次節では、そのアルゴリズムについて記述する。

3.3 Chandra-Toueg 分散合意アルゴリズム

このアルゴリズムは、非同期分散システムにおける合意アルゴリズムである。各プロセスは、ラウンドと呼ばれる4ステップから成る一連の動作を繰り返す。ラウンドでは特定のプロセスを調停者として選出し、調停者を基準にしてステップを実行する。調停者の選出方法は巡回調停者という方法を使用する。調停者による高信頼ブロードキャストという全プロセスに送信する特別な通信を受信することで値を決定し、停止する。以下に各ステップの動作と判断を示す。

● STEP1

調停者プロセスに提案値と更新ラウンド番号を送信する。

● STEP2

調停者プロセスは過半数のプロセスからのメッセージを待つ。受信したメッセージの中で、更新ラウンド番号が最も大きい提案値を自分の提案値に設定し、全プロセスに自分の提案値を送信する。調停者プロセス以外は何もしない。

● STEP3

調停者プロセスから値を受信するまで待つ。受信できた場合は、受信した値を自分の提案値に設定すると共に、現在のラウンド番号を更新ラウンド番号として設定し、調停者にackを返信する。受信出来なかった場合はnackを返信し、正常に受信出来なかったことを伝える。

● STEP4

調停者プロセスは過半数のプロセスからの返信を待つ。返信内容にnackが含まれていた場合は、ラウンド番号を一つ増やしてSTEP1へ戻る。返信内容全てがackだった場合は、自身の提案値をプロセス全ての決定値として、高信頼ブロードキャストを使用して全プロセスに通知する。調停者プロセス以外はラウンド番号を一つ増やしてSTEP1へ戻る。

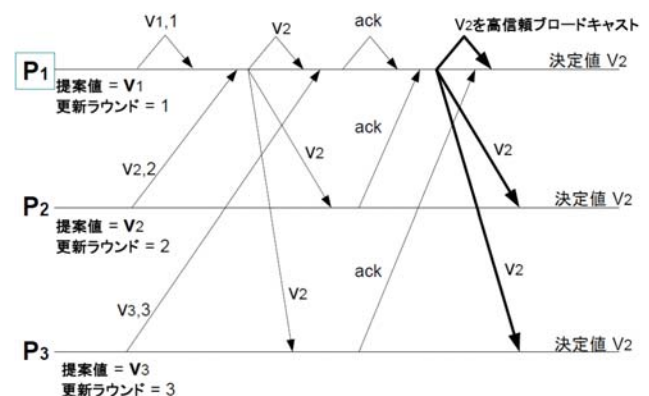


図2 通信遅延が無い場合のアルゴリズムの振る舞い

図2は通信遅延が無い場合の振る舞いを表している。各プロセスは、調停者であるP1に対して、提案値と更新ラウンド番号を送信する。P1は更新ラウンド番号の最も大きい提案値V2を自分の提案値に設定する。過半数のプロセスから、値が集まった時点で、自分の提案値を全てのプロセスに送信する。調停者からの提案値V2を受け取ったプロセスは、提案値を自分の提案値に設定すると共に、更新ラウンド番号を更新し、調停者にackを送信する。調停者P1はackを集計し、過半数のack受信を確認した時点で、高信頼ブロードキャストにて決定値V2を送信する。高信頼ブロードキャストを受け取ったプロセスは決定値V2を保存し、停止する。



4. Chandra-Toueg アルゴリズムの PROMELA モデル

Chandra-Toueg アルゴリズムにおける主体は各ノードである. これをプロセスとして定義する.

引数の int はノード固有の識別番号で、ノード判別に利用する。

Chandra-Toueg アルゴリズムの処理は 1 ラウンド 4 ステップで構成されているので，各ノードで行われる処理として，if 文と goto 文を使って記述する．

各プロセスに `step` という変数を持たせ、この値によって処理を分岐させる。また、高信頼度ブロードキャストを受

調停者の変更時に使用する，巡回調停者方式を再現するマクロを記述する．

```
inline RoundMaster(m){ m=(m+1)%NODES; }
```

各プロセスはモジュロ演算を使用し、調停者番号 (m) と全体のノード数 (NODES) から、次の調停者を決める.

PROMELA でプロセス間の通信を、通信チャネルでつなぐ形で表現するが、この方法では通信経路でのアクシデントを再現できない。そこで、プロセスとプロセスの間に通信経路プロセスを挟み、このプロセスの中に通信遅延などのアクシデントを記述することでこれを解決する。

```
proctype Middle(int id){}
```

通信経路使用権の奪い合いになるのを防ぐために、この通信経路プロセスは各ノード毎に作成する。

通信を行う際のチャネルは、通信の内容によって複数用意する。このチャネルも各プロセス毎に設定する。また、前述の通信経路プロセスを使用する関係上、高信頼ブロードキャストと調停者変更用チャネルを除いて、各チャネルは「送信元から通信経路」、「通信経路から送信先」の2つずつ用意する。

故障プロセスを表現するために retire ラベルを作成し、故障処理を記述する. if 文を使用し、非決定的に故障を発生させる.

故障したプロセスは、高信頼ブロードキャストを受信する以外の処理を行わない。しかし、故障プロセス宛にメッセージが送り続けられた場合、通信チャネルのバッファが溢れてしまうので、メッセージを捨てるという処理を行う。

また、過半数のプロセスが故障すると、アルゴリズムが破綻するので、この場合は、高信頼ブロードキャストを使用して全プロセスに全処理の停止を通知する。

4.3 グローバルタイムアウト機構

STEP3, STEP4 の処理では、プロセスが故障しているかどうかの推測処理が必要となるが、このアルゴリズムには故障プロセスの推測方法についての規定がない。本研究ではタイムアウト処理による、故障検出方法を使用する。しかし、PROMELA の言語仕様では、時間の概念を記述することが困難である。このような表現を行うために、“グローバルタイムアウト機構”を導入する。

グローバルタイムアウト機構は、アルゴリズムとしてのタイムアウトを再現するマクロである。各プロセスにタイムアウトカウンタ (タイムアウト処理発生までの猶予を示す大域変数) を割り当て、各プロセスによって必要なタイムアウトカウンタを設定させる。この変数を、他のプロセスの行動が全てロックした時に、Clock プロセスによって減らすことで時間を表現する。各プロセスは各カウンタが 0 になったときにタイムアウト処理を行う。Clock プロセスがカウンタ減算処理を `atomic` 文を使用して不可分実行として行うことで、減らし方を一律にする。Clock プロセスの記述は以下の通りである。

```
proctype Clock(){
  do
    ::timeout ->
      atomic{
        NODE1 のカウンタをデクリメントする処理
        NODE2 のカウンタをデクリメントする処理
        NODE3 のカウンタをデクリメントする処理
        .
        .
        .
      }
    od;
  }
}
```

タイムアウト処理を利用する場合は、以下のように記述する。

```
timeout_count = 3;
do
  ::(timeout_count == 0) -> タイムアウト処理
  ::(line?message;) -> 通常の処理
od;
```

`timeout_count` に値を代入し、`do` 文で `line` チャネルに

メッセージが届くのを待つ。メッセージ受信を待っている間は、`timeout_count` が減っていく。カウンタが 0 になる前にメッセージが受信できれば通常の処理を行い、メッセージ受信前にカウンタが 0 になった場合はタイムアウト処理を行う。

また、この仕組みは通信経路におけるメッセージ遅延を再現することにも使用する。タイムアウト処理を通信経路プロセスで使用し、必ずタイムアウトになるような記述を行うことで、通信経路でのカウンタ減算分だけメッセージの到着を遅らせる。

```
inline delay(count){
  delay_count = count;
  if
  ::(delay_count == 0) -> skip;
  fi;
}
```

`delay_count` に `count` 分の値を代入し、`if` 文によって `delay_count` が 0 になるまで何もしない状態を作る。`delay_count` が減算される際には、他プロセスのタイムアウトカウンタも同時に減算される。このとき生じるカウンタの差によって、擬似的に遅延を再現する。

4.4 調停者選出のアルゴリズム拡張

Chandra-Toueg アルゴリズムは、巡回調停者方式という方法によって調停者を決めている。しかしこの方法では、調停者が健全かどうかにかかわらず調停者を変更してしまう。また、実際には調停者の変更を行った後は、ノード間でリセット動作等の同期をとるべきである。そこで本研究では、調停者の選出方法を変更し、モデルに調停者が健全かどうかの判定と、選出後のノード間の同期処理を組み込む。調停者の選出方法にはブリーアルゴリズム [10] を使用する。

ブリーアルゴリズムは、複数のプロセスの持つ値から最大の値を選出するアルゴリズムである。各プロセスは自分の `id` より大きなプロセスに Election メッセージを送信する。Election メッセージを受信したプロセスは OK メッセージを返信する。OK メッセージを受信したプロセスは値が決定されるまで受信待機となる。OK メッセージを受信しなかったプロセスは、タイムアウト処理を行い、全てのプロセスに自分の値が選出されたことを伝える。(図 4.5) このアルゴリズムを再現するために、以下のような記述を行う。

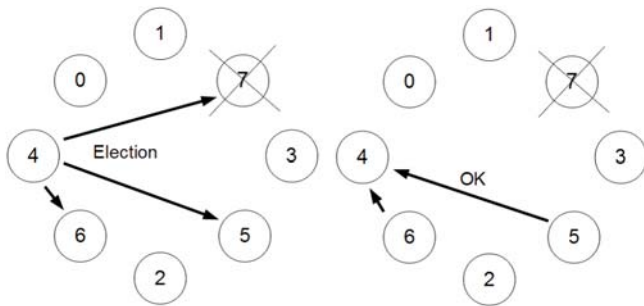


図 4 Election メッセージと返信

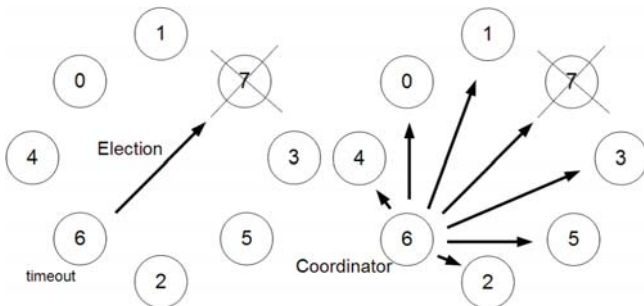


図 5 timeout と Coordinator 宣言

```
inline bully(id){
  自分より大きい id に Election メッセージ送信;
  タイムアウトカウントを設定;
  do
    ::Election メッセージを受信 -> OK メッセージ返信;
    ::OK メッセージを受信 -> Coordinator 通信を待つ;
    ::タイムアウト発生 -> Coordinator 通信を行う;
  od;
}
```

まず、各プロセスは Election メッセージを送信する。次にグローバルタイムアウト機構を利用してタイムアウト処理を行うために、タイムアウトカウントを設定する。OK メッセージを受信した場合は Coordinator 通信を待つ状態になる。Election メッセージを受信した場合は OK メッセージを返信する。タイムアウトが発生した場合は、Coordinator 通信を行い、ブリーアルゴリズムを終了する。Coordinator 通信を受信したプロセスは、次の調停者の id を保存し、ブリーアルゴリズムを終了する。

5. 合意アルゴリズムの検証

5.1 検証内容

作成したモデルを用いて検証する内容は、デッドロック検証である。SPIN を用いてデッドロック検証を行い、アルゴリズムにデッドロックが存在しないことを検証する。また、モデルの全ての振る舞いの最後に、assert 文を使用して全ての値が同じであることを表明する。検証の際にはノード数、タイムアウトカウント、遅延時間などを変更し

ながら検証を行い、各値によって検証結果にどのような変化があるのかを調べる。[A] 巡回調停者方式と [B] ブリーアルゴリズム方式の両方のモデルで検証を行う。

5.2 検証結果と考察

PROMELA で記述した Chandra-Toueg 分散合意アルゴリズムのモデルを使用し、SPIN を用いてデッドロック検査を行った。[A][B] どちらの方式でも、NODE 数は 3~6、タイムアウトカウントは、調停者が NODE 数+2、調停者以外が (NODE 数+2)*2 となっている。その結果得られた検証結果を表 1, 2 にそれぞれ示す。

ノード数	遅延量	深さ	状態数
3	0	607	39,776,795
	0~2	661	54,863,200
	0~3	838	54,952,659
4	0	892	56,360,002
	0~2	954	59,488,654
	0~4	997	59,697,820
5	0	913	67,518,205
	0~3	1,195	63,510,851
	0~5	1,121	63,414,250
6	0	1,147	67,597,519
	0~3	1,127	66,491,342
	0~6	1,017	66,591,643

表 1 [A] 巡回調停者方式での検証結果

ノード数	遅延量	深さ	状態数
3	0	320	36,938,000
	0~2	497	57,543,593
	0~3	819	64,291,401
4	0	431	60,829,611
	0~2	607	61,847,552
	0~4	607	62,290,276
5	0	1,885	66,323,866
	0~3	2,289	65,773,944
	0~5	2,034	67,449,404
6	0	2,214	70,384,165
	0~3	2,139	70,037,429
	0~5	2,039	70,808,413

表 2 [B] ブリーアルゴリズム方式での検証結果 (1)

[A] 巡回調停者方式では、ノード数、遅延量が増えると、深さ、状態数、共に増えている。しかし、NODE 数を増やしたときの状態数の増え方はそれほど多くない。タイムアウト処理によって消滅した状態が多くあるものだと考えられる。ノード数 5、遅延量 0~3 から深さ、状態数の数値が増えなくなっているのは、設定したタイムアウトカウントの値が適切でなかったためだと考えられる。適切なタイムアウトカウントを設定すれば、ノード数 3, 4 と同じような値の推移が見られると考えられる。

[B] ブリーアルゴリズム方式においても巡回調停者方式と同じく、ノード数、遅延量が増えると、深さ、状態数が増えるが、状態数の増え方が巡回調停者方式よりも大きい。これは、ブリーアルゴリズムを組み込むにあたって増やした変数の影響であると考えられる。状態数は変数の変化の仕方では決まるので、変数を増やしたブリーアルゴリズム方式の方が状態数は大きくなっている。また、調停者選出時に同期を取ることで、ノード間の歩調が自然に合うまでの動きが省略されているので、巡回調停者方式に比べて深さが浅い。

ノード数	遅延量	深さ	状態数
3	0~6	818	64,763,682
	2~3	1,703	44,682,285
	3~4	1,517	59,078,144
	4~5	599	63,155,488
	5~6	574	56,565,504

表 3 [B] ブリーアルゴリズム方式での検証結果 (2)

ノード 3 について遅延量を細かく変更したところ、遅延量 2~4 で深さが増大する傾向が見られた。これは設定したタイムアウトカウントに到達するかしないかの遅延が発生したことで、タイムアウト処理と通常処理を繰り返し、深さが増えたものと考えられる。

6. まとめ

仕様記述言語 PROMELA を用いて、Chandra-Toueg 分散合意アルゴリズムをモデル化し、モデル検査器 SPIN を用いて作成したモデルのデッドロック検証を行った。検証の結果、巡回調停者方式、ブリーアルゴリズム方式のいずれもデッドロックは検出されず、正当性を検証することが出来た。モデル化の際には、グローバルタイムアウト機構を作成することで、PROMELA には存在しない時間の概念を表現し、タイムアウト処理を再現した。また、アルゴリズムの拡張として、調停者の選出方法を巡回調停者方式から、ブリーアルゴリズムを使用した選出方法に変更し、調停者変更後は同期処理を行うようにした。

今回はノード数などの値を手作業で変更しながら検証を行ったが、この作業は工数が多くなる。この変更作業を自動化することで、検証データを集めることが容易になると考えられる。

宮本 [11]、坂本 [12] の研究では、UML 図からの PROMELA コードの自動生成を試みている。その中では、本研究で使ったグローバルタイムアウト機構の UML ステートマシン図への形式記述の組み込みが成功している。他方、UML コミュニケーション図において、スケラブルなノード記述に対応した変換が実装できていない。本研究で行ったような、値をスケラブルに記述し、定義

範囲を変化させながら自動検証を行う方法を確立することが今後の課題として挙げられる。

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] 米田友洋, 梶原誠司, 土屋達弘: ディペンダブルシステム—高信頼システム実現のための耐故障・検証・テスト技術—, 共立出版, 2005.
- [2] 谷口秀夫: 分散処理, オーム社, 2005.
- [3] Gerard J.Holzmann: *THE SPIN MODEL CHECKER*, Addison-Wesley, 2004.
- [4] 吉岡信和, 青木利晃, 田原康之: *SPIN* による設計モデル検証, 近代科学社, 2008.
- [5] 中島震: *SPIN* モデル検査, 近代科学社, 2008.
- [6] Tushar Deepak Chandra, Sam Toueg: *Unreliable failure detectors for reliable distributed systems*, Journal of the ACM, 43(2), 225-267, 1996.
- [7] 南川恭洋, 土屋達弘, 菊野亨: 耐故障分散アルゴリズムに対する *PROMELA* モデルの生成, 信学技報, CPSY2008-5, DC2008-5, 25-30, 2008.
- [8] 松尾尚文, 土屋達弘, 菊野亨: モデル検査による分散システムにおける合意アルゴリズムの正当性の検証, 信学技報, DC2005-7, 1-5, 2005.
- [9] 勝山純一, 和崎克己: モデル検査器 *SPIN* を用いた *Chandra-Toueg* 分散合意アルゴリズムのデッドロック検証, 電子情報通信学会 信州大学 Student Branch 論文発表会 講演論文集, 5, 2011.
- [10] Hector Garcia-Molina: *Elections in a Distributed Computing System*, IEEE Trans. on Computers, C-31(1), 48-59, 1982.
- [11] 宮本直樹, 和崎克己: 上流設計からモデル検査プロセスまでの一貫設計検証環境, 信学技報, SWIM2011-19, 7-12, 2011.
- [12] 坂本 統, 和崎克己: *UML* コミュニケーション図に対応する *SPIN* モデル検査向けコード生成器の拡張, 平成 24 年度電気関係学会東海支部連合大会講演論文集, A3-6, 2012.