B-021

タイムアウト機構を有するメッセージ交換プロトコルのUMLモデルとSPINモデル検査

UML modeling and SPIN model checking for a message exchange protocol with timeout behavior

坂本 統 †　後藤 亮馬 †　和崎 克己 ††
Osamu Sakamoto　Ryoma Gotoh　Katsumi Wasaki

1　はじめに

近年，ソフトウェアの仕様を検証する手法として，モデル検査が注目されている [1][2][3]．モデル検査とは，検査の対象となる仕様の振る舞いにおいて，仕様に含まれない不正な状態を，モデルが初期状態から取りうる状態を自動的に網羅することにより調べる技術である．モデル検査を行うことで，設計段階での欠陥を検出することができる．現在利用できるモデル検査ツールは多数存在するが，本研究では SPIN モデル検査器を採用する．SPIN は，分散システムの形式的な検証を行うためのソフトウェアパッケージとして広く用いられている．SPIN でモデル検査を実行するには，検査対象モデルを専用の仕様記述言語である PROMELA(PRocess MEta LAnguage) で記述し，要求仕様を線形時相論理 (Linear Temporal Logic : LTL) 式で記述することで，モデルが要求仕様を満たすかを検証することができる．

上流工程におけるモデルの記述法から，モデル検査器用のプロセスへ自動変換する手法は，広く研究されている．再利用性のある上位記術を行う従来研究として，UML 配置図とステートマシン図，シーケンス図を用いて検査対象システムと要求仕様をモデル化し，これら UML のデザインエントリ全体を，LTL 式及び，PROMELA 言語コードへ自動変換する手法が提案されている [4]．この手法では，自動変換後の PROMELA コードと LTL 式をセットで扱うことで，UML からモデル検査器までの一貫した設計検証が可能となる．本研究では，この UML 自動変換器を配置図に代わりコミュニケーション図に対応させ，ステートマシン図の記述を拡張することで，ノード間メッセージ交換プロトコルの振る舞いに適したモデル記述に対応させている．特に，PROMELA のタイムアウトを利用した時間同期を行う機構を導入することで，待機状態からの時間経過をトリガーとする振る舞いに対応させたことを報告する．

2　SPIN モデル検査器

SPIN は，ベル研究所で G.J.Holzman 博士を中心に開発されているモデル検査ツールである．分散システムの各プロセスを状態機械でモデル化し，プロセス間通信に同期・非同期チャネルを用いて並列動作するシステムの，形式的な検証を行うためのソフトウェアパッケージとして広く用いられている．専用の仕様記述言語 PROMELA は，相互通信プロセスや，非決定的な動作の記述をコードベースで行う．SPIN は，PROMELA

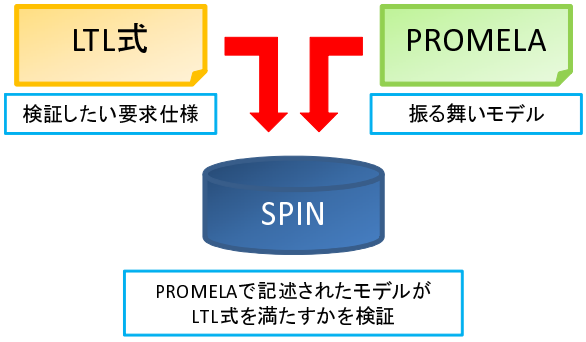

図 1　モデル検査器 SPIN

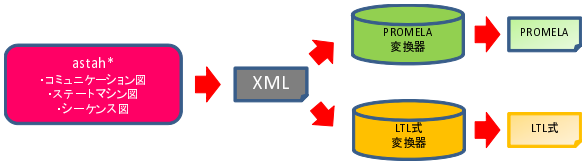

図 2　UML 図から PROMELA コードの自動生成

で記述されたモデルの表明検査，デッドロック検査等の性質を保証できる．

LTL 式は，時相論理式の一種であり，時間の進み方が一直線の性質を記述する．時相論理とは，論理式に時間の経過という概念を加えた論理体型である．「条件 A が最終的に真となる」，「条件 B が真になるまで条件 C は真である」といった将来的な命題を論理式で表すことができる．図 1 は SPIN を用いて検証を実行する流れを示す．SPIN は，LTL 式を NeverClaim というプロセスに変換し，モデルに組み込むことで，要求仕様を満たすかを検証する．

3　UML

UML[5] は，OMG（Object Management Group）により管理されている仕様記述言語であり，グラフィカルな記述で抽象化したシステムのモデルを生成する汎用モデリング言語である．本研究では，メッセージ交換プロトコルをモデル化対象とし，ネットワークの状態とプロトコルの振る舞いを記述するため，コミュニケーション図とステートマシン図を採用している [6]．モデリングツールにはチェンジビジョン社の astah*[7] を用いている．astah*は，デザインエントリ全体を XML 形式のファイルとしてエクスポートする機能を有する．エクスポートされた XML ファイルを，変換器を通じて PROMELA モデルと LTL 式に自動で変換することで，上流設計からモデル検査までの一貫した設計検証を行うことができる (図 2)．

† 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.

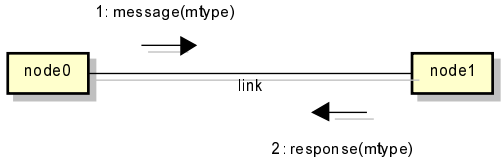

図 3　コミュニケーション図

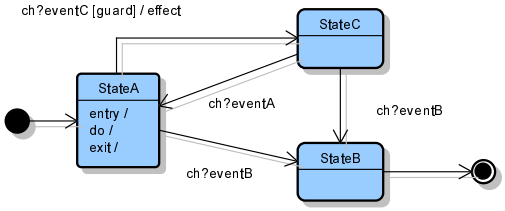

図 4　ステートマシン図

3.1　コミュニケーション図

コミュニケーション図は，オブジェクト間のメッセージフローを表し，オブジェクト間に接続関係があることを明示する (図 3)．従来研究では，配置図を用いて，プロトコルを適用されるノードとその接続関係をモデル化しているが，本研究ではネットワークの記述にコミュニケーション図を用いることで，接続状況に加え，メッセージフローを用いた通信情報の定義を行う．図 3 はコミュニケーション図の記述例である．ライフラインをネットワーク中の各ノードとして扱い，リンク線を繋げることで接続状況をモデル化する．リンク線にはメッセージと呼ばれるラベル付きの有向グラフを記述することができ，メッセージ名，引数，方向を併せて通信情報として定義する．コミュニケーション図において，ノード数や接続情報を任意に変更することで，スケールアウトに対応したネットワーク状態の上位記述を実現する．

3.2　ステートマシン図

ステートマシン図は，モデルの状態の変化を状態遷移として表現する (図 4)．本研究では，コミュニケーション図におけるライフライン (ノード) に関連付けられることで，それぞれのノードの振る舞いとして，ステートマシン図をモデル化する．トランジションに記述されたイベントを受信し，かつ条件 [*guard*] を満たした場合に，/*effect* を実行し，遷移先に状態遷移する．ステートには入場動作 *entry*，実行活動 *do*，退場動作 *exit* を記述でき，メッセージ送信等の処理を記述できる．黒丸シンボルは開始状態を表し，二重丸シンボルは終了状態を表す．遷移可能なステートが複数ある場合は，非決定的に状態遷移する．図 4 は，状態が *StateA* であるとき，通信路 *ch* に *eventB* を受信した場合は *StateB* に遷移し，*eventC* を受信し，かつ *guard* を満たした場合に，*effect* を実行し，*StateC* に遷移することを表す．

4　UML から PROMELA モデルへの変換

本研究で開発している変換器は，Perl 言語で実装されており，astah*でエクスポートされた XML ファイルを基に変換を行う．本項では，UML から PROMELA のモデルの変換手法 [4][6] について述べる．

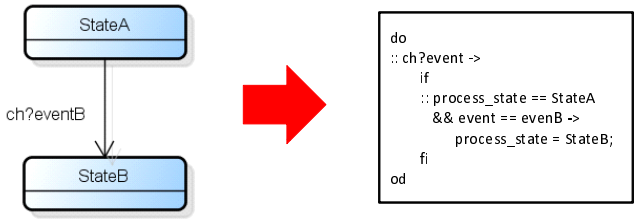

図 5　イベント駆動による状態遷移

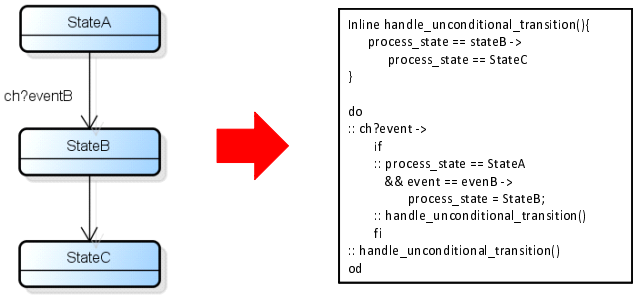

図 6　無条件遷移とイベント駆動による状態遷移

4.1　プロセスと通信用チャネルの定義

PROMELA では，振る舞いをプロセスごとに記述し，プロセス間通信は，*chan* 型の変数で行われる．自動変換後のプロセスと通信用チャネルは，コミュニケーション図で定義される．ライフライン (ノード) 毎にプロセスが生成され，各プロセスの振る舞いは関連付けられたステートマシン図から生成される．

通信用チャネルは，コミュニケーション図におけるメッセージ毎に生成され，リンク線の名前と結合されて定義される．メッセージの引数はチャネルの転送データ型として変換される．図 3 から変換された *chan* 型の変数の定義例を次に示す．PROMELA では，[0] は同期通信を表し，値が 1 以上の場合は非同期通信のバッファ数を表す．

```
chan link_1_message = [0] of {mtype};/*1:msg*/
chan link_2_message = [0] of {mtype};/*2:response*/
```

4.2　プロセスの振る舞いの変換

プロセスの振る舞いは関連付けられたステートマシン図から生成される．各プロセスは，どの状態であるかを保持する状態変数の値と，遷移条件を元に状態遷移を行う状態遷移モデルとして変換される．状態遷移モデルは，イベント受信によるイベント駆動遷移処理部と，イベントとは無関係に遷移する処理を記述する条件遷移処理部の 2 つで構成される．イベント駆動による状態遷移による変換例を図 5 に示す．PROMELA では，*chan* 型変数を用いて，*!*が送信，*?*が受信を表す．*do* 文でイベント受信を待ち，*if* 文で受信したイベントと状態変数の値を判定する．イベント受信に関わらない無条件遷移がある場合は，図 6 に示す形に変換される．*inline* によるマクロを用いて，状態変数の値を判定して遷移する．いずれの場合にも，*effect* と *exit* は状態遷移直前に，*entry* と *do* は状態遷移直後に出力される様に変換される．

5　タイムアウト機構への対応

従来の変換器の拡張機能として，タイムアウト機構を生成するモデル記述に対応させた．タイムアウト機構の概要と実装について述べる．

5.1　タイムアウト機構

メッセージ交換プロトコルの振る舞いの一つとして，待機状態からの時間経過による状態遷移が考えられる．例えば，通信待ち状態からのメッセージ再送要求処理が挙げられる．しかし，PROMELA においては，実時間を表現する手段がないため，擬似的な時間表現として，*timeout* を用いた他プロセスとの同期処理により，時間経過による状態遷移を表現する．PROMELA において，*timeout* は全てのプロセスがロックしている状態で真となることを表す．タイムアウト機構は，*timeout* を用いて，各プロセスが持つタイムアウト機構用カウンタ (タイムアウト処理発生までの猶予を示す大域変数) を進めることで，全プロセスがロックされた際に，最も早くカウントを達成し終えたプロセスの処理を実行させることで，カウンタのセット値の比較によるプロセス間の同期を実現する．

5.2　PROMELA におけるタイムアウト機構の実装

タイムアウト機構を PROMELA で実現するため，次の項目を実装した．

- カウントプロセス
- プロセス毎のカウンタ変数
- カウンタ変数の値の設定処理

timeout によるカウント同期を取るためには，他のプロセス全てのカウンタを同時に進行させるカウントプロセスが必要である．カウントプロセスは，他のプロセス全てがロックした際に，*timeout* を用いて，全てのカウンタ変数を排他制御で同時に進める．カウンタ進行動作は *tick*() としてマクロで定義されており，引数をデクリメントする処理である．各プロセスはカウンタが 0 になったタイミングをトリガーとして処理を実行する．排他制御は，PROMELA では *atomic* で記述される．次にカウンタプロセスを記載する．

```
#define tick(x)     if \
                    :: x != -1 -> x = x - 1; \
                    :: else; \
                    fi;

proctype Clock(){
  do
  :: timeout ->
      atomic{
        tick(process0_counter); /*Process0*/
        tick(process1_counter); /*Process1*/
        ...
      }
  od;
}
```

タイムアウト機構をトランジションのトリガーとする場合は，次の様に記述する．カウンタ変数の値の設定処理や，タイムアウト機構によるトリガー等はマクロで定義されている．*set*() はカウンタ変数に値を設定する．*expire*() は，カウントプロセスによってカウンタ変数が 0 になっている条件を表す．カウンタが 0 になる前でも，条件が満たされれば，他の遷移が実行される．

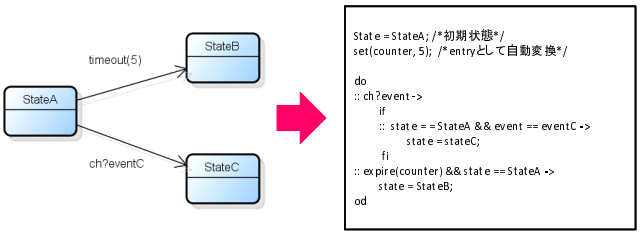

図 7　タイムアウト機構による状態遷移

```
#define set(x,y)    x = y
#define expire(x)   x == 0

set(process_counter, 3); /*値の設定処理*/
do
:: expire(process_counter) -> /*タイムアウト機構遷移*/
:: ch?event -> /*イベント駆動遷移*/
:: handle_unconditional_transition() -> /*無条件遷移*/
od
```

5.3　ステートマシン図における記述と変換

ステートマシン図において，タイムアウト機構による状態遷移を記述するには，*timeout*(*value*) をトリガーとして記述する．引数 *value* には任意の整数値が設定される．カウンタ変数の値の設定処理は，*timeout*(*value*) をトリガーに持つトランジションのソースステートの入場動作として自動に変換される．

図 7 はステートマシン図の記述と変換例である．*StateA* に状態遷移後の入場動作として，*set*() によるカウンタ値の設定処理が実行される．タイムアウト機構によるトリガー *expire*() の引数であるカウンタ変数の値が 0 かつ状態変数値が *StateA* である場合に，*StateB* に遷移することを表している．*expire*() を満たす前に，*eventC* を受信すれば，*StateC* に遷移する．

6　変数を用いた式計算と評価

タイムアウト機構による同期機能を追加することで，時間経過による処理に対応させた．メッセージ交換プロトコルの時間経過処理として，メッセージの送信要求処理等が挙げられる．メッセージ送信要求は，待機状態から特定の時間が経過した場合に送信されるが，要求送信後も反応がない場合は繰り返して再送要求が送られる．この時，再送要求を送る回数は有限回であり，再送要求を送るまでの待機時間は変化させるという振る舞いが考えられる．この振る舞いを表現するために，変数を用いた記述に対応させた．図 8 に記述例を示す．

6.1　変数の定義

ステートマシン図において，コメントを用いることで，変数宣言を行えるように対応させた．図 8 の様にコメントの先頭に変数宣言を示す**declaration**を記述することで，変数を定義することが可能になる．変数の記述法と，定義可能な型は，PROMELA の記述法に準拠する．定義された変数はステートマシン図の中で使用することができる．

6.2　変数を用いた制御

タイムアウト機構の経過時間を，変数を用いた数式として記述可能に対応させた．記述表現は，PROMELA で記述可能な表現に準拠する．図 8 では，引数として

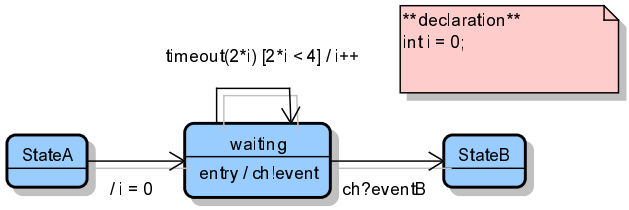

図 8　変数を用いたタイムアウト機構の繰り返し表現

数式 2*i を設定している．変数 i はコメントで定義され，*StateA* から *waiting* に遷移する際に，アクションで i=0 として初期化している．*timout* を有限回繰り返すため，ガード [2*i<4] を用いることで状態遷移条件の制御を行い，アクション i++ で変数 i をインクリメントし，設定時間を変化させている．

6.3　チャネル受信用変数の定義

チャネル受信動作において，受信したイベント以外の情報を基に遷移する振る舞いに対応させるため，コミュニケーション図のメッセージの引数に変数名を定義できるよう対応させた．例えば，ABP プロトコルといった受信情報によって状態が遷移するような振る舞いに対し，変数名を用いることで受信した値の評価が可能になった．

7　モデル記述例

本研究で開発している変換器に対応した UML モデルの例を載せる．

7.1　リングネットワークモデル

リングネットワークのリーダー選出アルゴリズムモデル [8] の記述例を図 9 と図 10 に示す．各ノードはリング上に接続され，片方向に通信可能な状態である．開始ノードは，自らの ID を送信可能な隣接ノードへ送り，ID を受信したノードは，自分の ID と受信 ID の値を比較する．受信 ID が自らの ID の値よりが大きい場合に，ID を保存し，受信 ID を次のノードへ送る．値が小さい場合には，自らの ID を次のノードへ送る．お互いの ID の値が等しい場合は，リーダー選出の合意がとれたものとし，アルゴリズムを終了する．終了した時に保存されている ID がリーダとなる．このアルゴリズムモデルを変換器で変換し，SPIN を用いてデッドロック検証を行ったところ，エラーが発生しないことを確認した．

8　まとめと今後の課題

8.1　まとめ

UML コミュニケーション図とステートマシン図を用いることでスケーラブルな SPIN モデル検査器向けコード生成と検証を行うことが可能になった．また，タイムアウト機構を実装し，変数に対応させることで，ABP プロトコルや，リングネットワークにおける合意アルゴリズム等の複雑なメッセージ交換プロトコルのモデル化に対応可能になり，幅広いシステムの検証を行えるようになった．

8.2　今後の課題

現在の変換器では，ノード毎に個別にステートマシン図を記述しなければならない．例えば，振る舞いは同じ

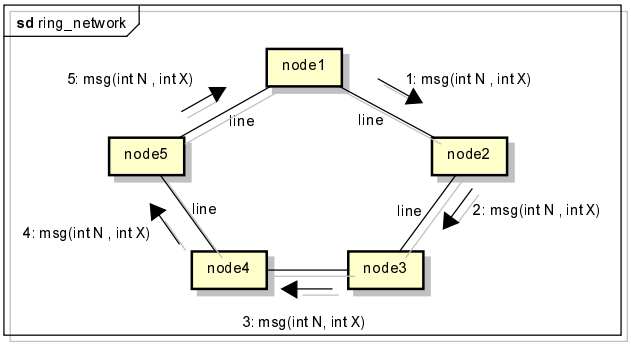

図 9　リングモデル（コミュニケーション図）

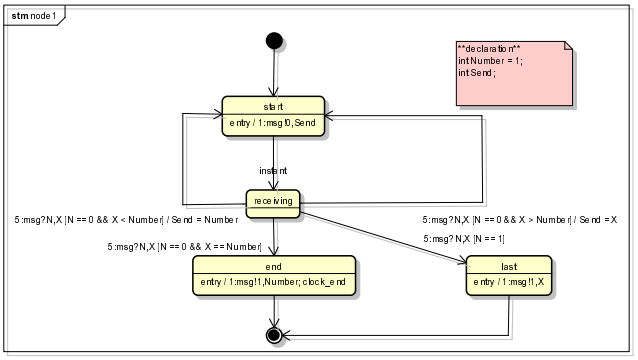

図 10　リングモデル（ステートマシン図）

でも，接続されているノードが異なれば，使用する通信チャネルを変更したステートマシン図を用意しなければ変換を行うことができない．そのため，ステートマシン図を一般化し，メッセージ送受信などの振る舞いを汎用的に記述することで，モデル記述の抽象化を図りたい．現在，通信チャネルの汎用化記述として，ステートマシン図におけるチャネル記述を，コミュニケーション図から読み取ることで自動変換する手法を考えている．

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] Gerard J. Holzmann : “The SPIN Model Checker” ; Addison-Wesley, 2004.
[2] 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, 2009.
[3] 中島震 : “SPIN モデル検査” ; 近代科学社, 2008.
[4] 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), 111(308), 7-12, 2011.
[5] UML Resource Page, The Unified Modeling Language(UML),the Object Management Group(OMG), http://www.uml.org/
[6] 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), 1page(CD-ROM), 2012.
[7] 株式会社チェンジビジョン, http://www.changevision.com/
[8] 谷口秀夫 : “分散処理” ; オーム社, 2005.