OUKA
The University of Osaka
Institutional Knowledge Archive

Title	上位設計におけるシステムの振る舞い検証技術
Author(s)	岡野, 浩三
Citation	システム制御情報学会誌. 2008, 52(9), p. 328-333
Version Type	VoR
URL	https://hdl.handle.net/11094/51213
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

システム/制御/情報, Vol. 52, No. 9, pp. 328–333, 2008

「システム設計のための形式手法の基礎と応用特集号」

解　説

上位設計におけるシステムの振る舞い検証技術

岡野　浩三*

1. はじめに

システム開発では上位設計における振る舞い性質の検証を行っておくことが設計の後戻りを少なくする意味で大事である．モデル検査は状態遷移系としてモデル化されたシステムと期待される（あるいは期待されない）振る舞い性質記述の二つを入力とし，前者の振る舞いが後者を満たすか否かを総当りで調べる形式的手法である．状態表現の簡約化や，状態走査の高速化の研究が進み，近年ではシステム上位設計における検証技術として実用的観点から注目を浴びている．

また，近年 SysML (System Model Language)[1] が注目を浴びている．SysML は UML2 をもとに拡張を行い，物理系，物理デバイス，ソフトウェアシステムからなる混合システムを表現することを目指している．これらの拡張により各ブロック間に流れる信号や物理的フローなどに関する制約や関係をよりわかりやすく提示できるようになっている．システムの振る舞いを記述する振る舞い図については UML2 と同様に四つの図 (diagram) をサポートしている．とりわけ Harel の Statechart にもとづく状態遷移図はサブシステムの内部の振る舞いを記述するのに適しており，状態の階層化，並列など表現能力も高い．

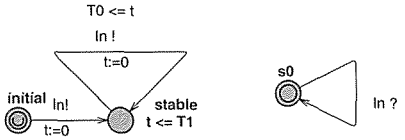
状態遷移図を用いていくつかのサブシステムの振る舞いを記述したときに，このシステムの入出力がブロックのポートに指定した制約条件等を満たすかどうかを設計の早い段階でチェックできることが望ましい．

以降，本稿では，振る舞いモデルの形式検証技術，とりわけ，時間オートマトンのモデル検査と確率オートマトンのモデル検査について述べる．

2. 時間オートマトン

時間オートマトン [3–5] は有限個のクロックを付加したオートマトンである．時間オートマトン用のモデル検査ツールは複数知られているが，ここでは UPPAAL[2] について述べる．

第1図に時間オートマトンの例をあげる．これは二つの時間オートマトンから構成されているが，左の時間オートマトン（以降 Producer とよぶ）が本質である．Producer はクロック t を一つもち，時間制約式が遷移や状態（ロケーションとよばれる）に与えられている．前者をガードとよび，後者をインバリアントとよぶ．**initial** と名づけられているロケーションが初期ロケーション（二重丸で初期ロケーションを意味する）である．



第 1 図　時間オートマトン

T1 の値を 25，T0 の値を 24 とすると，この時間オートマトンの動作は簡単にはつぎのようにみなせる．イベント (またはアクションともよぶ) $In!$ を発生させ，その際クロック t を 0 リセットする．その後は前のイベント発生時刻からちょうど 24 単位時間経過したのち，25 単位時間に達するまでのどこかの任意の時刻でイベント $In!$ を発生させることを繰り返す．

以降，もう少し詳しく説明する．時間オートマトンの意味モデルは状態遷移系として与えられる．初期ロケーションに位置し，各クロックの値を 0 とした状態を意味モデルの初期状態とみなす．意味モデルでの遷移は時間遷移 (delay transition) と，アクション遷移 (action transition) の二つがある．時間遷移はロケーションの滞在を意味する．ロケーションに与えられたインバリアントを満たす範囲で，各クロックに一様かつ稠密な時間遷移を許している．**initial** では特別なインバリアント[1]U(Urgent) が与えられており，**initial** での時間滞在が許されないことを意味している．したがって Producer は初期状態から瞬時に実行可能なアクション遷移（この例では **initial** から **stable** の遷移）を実行することとなる．一般に，アクション遷移は瞬時に実行される．アクション遷移が実行できる条件はガードを満たすこと，遷移先のロケーションにおけるインバリアントを満たすこと，および遷移のイベントが同期可能であることである．例の遷移はガードがないので実行可能となり，イベント $In!$ が発生する．

* 大阪大学 大学院情報科学研究科

Key Words: SysML, model checking, timed automaton, probabilistic model.

[1] 正確にはインバリアントで時間制約が与えられるのではなく，このロケーションの出力遷移に時間制約 $t \leq 0$ が与えられる，と解釈する．

このとき，クロックtが値0にリセットされ**stable**に移る．**stable**に付加されたインバリアント$t \leq T1$が満たされる間，滞在することができる（この間，時間遷移がおこり，クロックtの値が増加していく）．**stable**からはセルフループの遷移があり，インバリアントとガードを満たす任意の時点で瞬時にイベント$In!$を発生させ，そのつど，クロックtをリセットさせることを以降繰り返す．

一般に複数の遷移が実行可能なときは非決定的に選択できる．時間遷移にともなって実行可能なアクション遷移が変化することもある．一般に，あるロケーションに滞在したときに，インバリアントを満たす範囲で実行可能なアクション遷移がない場合はデッドロックとなる．

なお，インバリアントが与えられてないロケーション（すなわちいつまでも滞在可能なロケーション）においても，出次遷移が定義されていない場合はデッドロックとされる．

UPPAALでは複数の時間オートマトンを並列合成し，複雑なシステムを記述することができる．並列合成されたシステムの初期状態は各オートマトンの初期状態で定義される．時間遷移はすべてのクロックに一様に適用される．一方，イベント遷移は，基本的に同一のイベントがラベル付けされている遷移の中ですべてのガード条件を満たす遷移可能な遷移が選択される．ただし，UPPAALではさらに強い制約をおいており，CCSのように！が付けられたイベントは？と付けられた同名のイベントとしか同期できない．またこの制約を満たす可能な遷移の組合せのうち!,？のペアの遷移が非決定的に選択される．上述の説明ではガード条件さえ合えば，イベント$In!$が発生可能であるかのように説明したが，実際は右の時間オートマトンに対応する遷移があり，無条件かつ任意の時刻で実行可能にしているため，この二つの並列合成が全体として，先ほどの説明にあうように動作する．

なお，UPPAAL独自の拡張により，クロック以外に有限の値域をもつ整数変数などをガード式に用いたり，遷移において簡単な計算をさせたりすることができる．また，これらのクロックや変数のスコープをローカル，グローバルに設定することができる．さらに，イベントによる同期とグローバルに設定された変数の共有を用いて時間オートマトン間で柔軟なデータ交換を行うシステムを記述することができる．

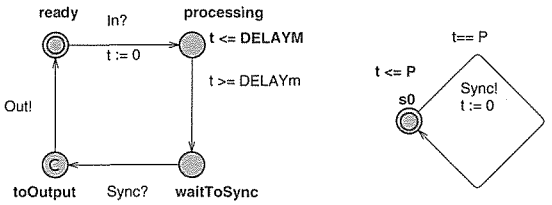
時間遷移は各クロックの値を同時に任意の正実数の値で増加させることに注意されたし．したがって，時間オートマトンの意味モデルの状態はロケーションと各クロックの値の組で表現することができるが，意味モデルの状態（や遷移）は一般に無限に存在することになる．モデル検査では制約条件式の特徴を活かし，無限状態遷移系を等価な有限状態遷移系でとらえ直し扱っている．また，さまざまな効率化の工夫を行っている．

2.1 記述例

簡単な例題を通じてこの時間オートマトンの検証の実際についてやや詳しく述べる．

第1図のコンポーネント（Producer）において，T0とT1がある値Tに近い場合は，ほぼ一定の周期TでイベントInを繰り返し発生させるとみなせる．そのイベントを受けて，やや長い処理時間Lを要したのち別のイベントOutを発生させるコンポーネント（Consumer）を考える．一般に一つのProducerに対しL/T個以上のConsumerをパイプライン式に配置することにより，滞りなく，各イベントの処理ができると期待される．問題をやや複雑にするために，Outの発生時刻は別のコンポーネントSynchronizerの発生させる周期Pのイベント$Sync$に同期して出力されることとする．すなわち処理時間Lを終えた後の最初の$Sync$のタイミングでOutが出力される．このとき，各パラメータT, L, P, Consumerの個数などの設計値により，イベントOutのthroughput, jitter, Inとのlatency(これらの定義は後述) などが変わり得る．

第2図にConsumer, Synchronizerの各時間オートマトンを示す．



第2図　Consumerと Synchronizer

ConsumerはイベントInを受けてロケーション**processing**に遷移する．この状態で少なくとも$DELAYm$単位時間，多くとも$DELAYM$単位時間滞在したのち，ロケーション**waitToSync**に遷移する．ここで$Sync$を受信すればただちにOutを送信し，初期ロケーションに遷移する．このように複数の遷移を優先的，連続的かつ瞬時に動作させることをロケーションにC(Committed)をつけることにより要請している．これもUPPAAL独自の拡張である．

Synchronizerは周期Pでイベント$Sync$を発生させる．Consumerが複数あるとしても一般に，定期的に発生する$Sync$をそのタイミングで受信する相手が常にいるとは限らない．解決方法として，無条件に$Sync$と同期する時間オートマトンRecを新たに導入し，大域変数にセマフォを設定し，全体をコントロールする方法，同様にRecを導入した上でプロセスの優先度を設定する方法，など複数あるが，ここではRecは導入せず，ブロードキャストチャネルを用いる方法を使う．$Sync$をブロードキャストチャネルに指定すると！と指定されたイベントは？と指定された0個以上の複数のイベントと可能ならば同時に遷移を行う．すなわち，同期するイベントが

なくてもデッドロックにはならない．

システムのあるモジュールが一定の間隔で同一信号を出力しつづけることなどシステムの（周期的な）時間性質を早期の上位設計で検証，保証することは有用である．このような時間性質はtimeliness QoSとよばれる．

timeliness QoSとして以下の例をあげることができる．以下では，イベントxのi回目の発生時刻をx_iとする．

- L単位時間にk回以上のイベントxが発生する．
  (throughput) $\forall n: |x_{n+k}-x_n| \leq L$
- T単位時間間隔，誤差dでイベントxが発生する．
  (jitter) $\forall n: T-d \leq |x_{n+1}-x_n| \leq T+d$
- 二つのイベントx,yについて発生時刻の差はL単位時間以内である．
  (latency) $\forall n: |y_n-x_n| \leq L$

2.2 テストオートマトン

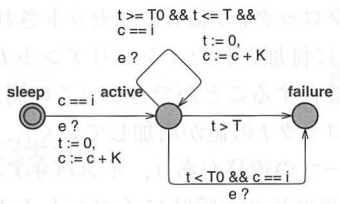
UPPAALでは時間オートマトンの並列合成として与えられたシステムに対して，与えられた検査式が成り立つか否かをモデル検査できる．UPPAALではかなり限定されたTCTL式しか扱えないが，それでも「システムがどのように振舞ったとしても悪い状況pにならないこと (安全性)(A□ not p)」,「システムがいつかは指定された状況pにたどり着くこと (活性)(A◇ p)」,「システムがある指定された状況pにたどり着く可能性があること (到着可能性) (E◇ p)」などが検証できる．pとしては各プロセスのクロック値に関する制約や各プロセスが位置するロケーションに関する表現が記述できる．

残念ながらtimelinessの性質はUPPAALの検査式では表現が難しいと思われる．一般に複雑な性質はテストオートマトンを用いると解決できることがある [6]．検査対象システムをSとし，テストオートマトンをTとするとT,Sの並列合成$T||S$がデッドロックを起こすとき，性質を満たさないと規定することができる．したがってモデル検査でデッドロック判定(検査式はA□ not deadlock) を行うことによりこれらの性質をシステムSが持つか否かを判定できる．

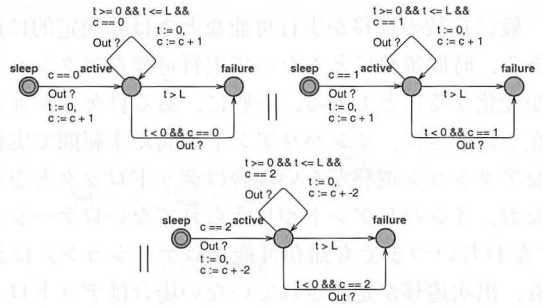
第3図の時間オートマトンのパラメータを$T0,T,i,K$の四つとし，以降この時間オートマトンを$TAd(T0,T,i,K)$で表す．

第4図のテストオートマトンTATは$TAd(T0,T,i,K)$のパラメータを変えた三つの時間オートマトンの並列合成$TAd(0,T,0,1)||TAd(0,T,1,1)||TAd(0,T,2,-2)$として構成されており，(throughput) $\forall n: |Out_{n+3}-Out_n| < T$を判定する．

TATは以下のように動作する．最初，整数カウンタcの値は0である．このときイベントOutが発生すると最左の時間オートマトン$TAd(0,T,0,1)$が起動し，ガード$c==0$を満たすのでロケーションを**sleep**から**active**に変化させる．このときカウンタを一つインクリメントさせ，同時にローカルクロックt_0を値0にリセット



第3図　イベントeの監視をするテストオートマトンTAd



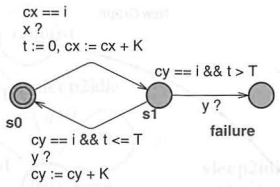
第4図　throughput $K=3$のテストオートマトン

する．この時点でさらにOutが発生したとする．こんどは二つ目の$TAd(0,T,1,1)$が起動し，ロケーションを**sleep**から**active**に変化させる．同時にカウンタcを一つインクリメントする．三つ目の信号が発生したときは$TAd(0,T,2,-2)$が起動し，同様の動作を行うが，カウンタの値は$c:=c-2$で0にセットする．これ以降，i番目の信号Outが$i-3$番目の信号Outの発生時刻から区間$[0,T]$の間に発生する限り，各時間オートマトンが順次，**active**から**active**への遷移を繰り返すこととなる．信号の発生時刻がこの条件を満たさないとき，いずれかの時間オートマトンが**failure**ロケーションに移行し，それにともなって他の時間オートマトンも**failure**ロケーションに移行する．最終的にシステム全体がデッドロックに陥り，このスループットが満たされないことをデッドロック判定で検証できる．

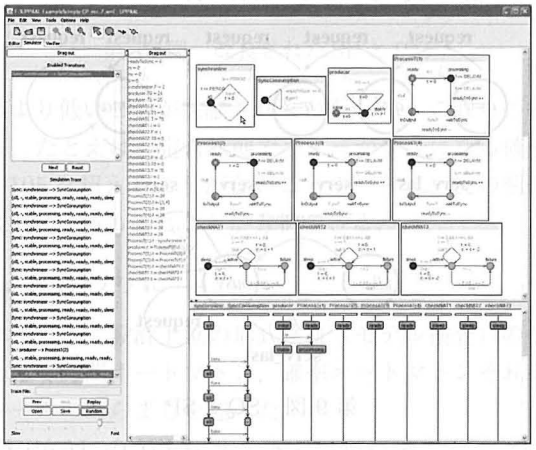
同様に一つの$TAd(T-d,T+d,0,1)$をこのパラメータでインスタンス化すると(Jitter) $\forall n: T-d \leq |x_{n+1}-x_n| \leq T+d$のテストオートマトンとなる．

第5図の時間オートマトンTALはlatencyを観測するための時間オートマトンである．throughputと同様に，複数合成して用いる．T,K,iをパラメータとしており，Tは計測したいlatencyの値を表し，iはTALの識別子に用いる．クロックtは各オートマトン固有のものであり，信号x,y間の遅延を計測するのに用いる．KはTAdと同様に，これらの時間オートマトンをサイクリックに再利用するコントロールに用いる．カウンタcx,cyはグローバルに共用され，現在いくつのイベントが計測の対象になっているかを表す．TALをパラメータ込みで$TAL(i,L,K)$で表すことにする．

予想されるlatencyがMで信号の発生間隔がおよそTのとき，TALを少なくともM/T個用意し，以下のよ



第 5 図　latency のテストオートマトン



第 6 図　UPPAAL によるシミュレーションの様子

うに並列合成する必要がある．

$TAL(1,L,0) \,||\, TAL(1,L,1) \,||\, \cdots \,||\, TAL(1,L,n-2) \,||\, TAL(-(n-2),L,n-1)$

2.3　検証例

Producer, Syncronizer のプロセスを一つ，Consumer のプロセスを四つ，その他，イベント Out とフリーで同期するプロセスをパラメータ ($T0$=24; $T1$=25; $DELAYm$=79; $DELAYM$=80; P=2) で並列に動作させる．このとき，UPPAAL において，シミュレーションで動作の進行を視覚的に確かめることができるし，検証結果として deadlock に陥らないことも検証できる（第 6 図）．

この状況で Consumer のプロセスを三つに減らすと，Producer の信号を受けるプロセス数の絶対数が足りず，進行不能（deadlock）になることも容易に確かめられる．

また，ブロードキャストを用いたため，$Sync$ に同期して，複数の Consumer が同時に Out を出す可能性が生じるが，そういうことは指定パラメータのもとでは生じないことも次の検証式で検証できる．

$$A\Box(\forall i : id_t \forall j : id_t ProcessT(i).waitToSync \wedge ProcessT(j).waitToSync \rightarrow i == j)$$

この式は二つの相異なる ProcessT(i) が同時に waitToSync に入らないことを意味している．

イベント Out とフリーで同期するプロセスの代わりに，各 TA を入れると throughput, jitter, latecy の判定ができる．deadlock に陥らない各パラメータの値および，標準的な PC で行った検証時間を第 1 表にまとめる．

第 1 表　パラメータ表

timeliness QoS	parameters	検証時間 [s]
throughput	$L=78, K=3$	60
jitter	$T0=22, T=29$	5
latency	$L=82, K=5$	10

3.　確率オートマトン

確率的モデルを対象としたモデル検査は，ランダムな振る舞いをもつシステムのパフォーマンスや信頼性，安定性を見積もるための形式的手法である．本節では，本手法に用いられる確率的モデル検査ツール PRISM[7] (以降，PRISM とする) について簡単に述べる．

PRISM は離散時間マルコフ鎖 (DTMC)，連続時間マルコフ鎖 (CTMC)，マルコフ決定過程 (MDP) を状態遷移系としてとれる．DTMC は遷移確率つきの有限オートマトンとみなせる．クロック変数などの概念はないが 1 状態遷移をクロックによる時間遷移とみなすことができる（MDP は本質的に DTCM にアクションの概念と非決定的選択を追加した派生とみなせる）．一方，CTMC では指数分布のパラメータを状態遷移に与える（時間要素は遷移の遅延時間として意味づけされる）．なお，時間オートマトンに直接確率を導入したモデル PTA[9] も提案されているが，現時点では計算効率の問題等もあり，ツールではサポートされていない．

PRISM の WEB サイト [8] のチュートリアルから，DTMC の例として自己安定分散アルゴリズム，CTMC の例としてパワーコントローラについて簡単に触れる．

その前に性質記述の言語 PCTL (Probabilistic Computation Logic), CSL (Continuous Stochastic Logic) についてごく簡単に紹介する．

3.1　PCTL と CSL

これらはともに CTL をもとに確率に関する表現を追加した言語である．ともにほぼ同じ構文を有するが，解釈モデルとして PCTL は DTMC と MDP を，一方，CSL は CTMC のモデルをそれぞれ用いる．

PCTL(CSL) 式は $P_{\geq p}$ pathprop, $S_{\geq p}$ pathprop などの形をしている．前者は，たとえば，パス式 pathprop が真である確率が p 以上であるかを表す式である．$\geq$ 以外の比較式も用いることができる．P,S オペレータはそれぞれ確率，定常状態の確率を意味する．後者は CTMC のみで用いられる．他に，期待値（利益）に関する R オペレータも用意されている．P,S,R はいずれも，クエリ形を持っており $P_{=?}$ などでツールに対して，値を問うこともできる．MDP に対しては $P_{=?}$ の代わりに，$P_{\max?}$ $P_{\min?}$ が用いられる．

パス式 pathprop は CTL 式のように X, U, U_I, F, G オペレータを許している．

PCTL のパス式の集合は可測であることが証明されている．基本的には CTL の拡張であるので，これらの式の

```
dtmc
// module for process 1
module process1
  x1 : [0..1] init 1;

  [step] (x1=x7) -> 0.5:(x1'=0) + 0.5:(x1'=1);
  [step] !(x1=x7)-> (x1'=x7);
endmodule

// add further processes through renaming
module process2=process1[x1=x2, x7=x1] endmodule
module process3=process1[x1=x3, x7=x2] endmodule
module process4=process1[x1=x4, x7=x3] endmodule
module process5=process1[x1=x5, x7=x4] endmodule
module process6=process1[x1=x6, x7=x5] endmodule
module process7=process1[x1=x7, x7=x6] endmodule

// costs - 1 in every state
rewards
  true : 1;
endrewards
```

第7図 Hermanの自己安定アルゴリズム(ノード7)

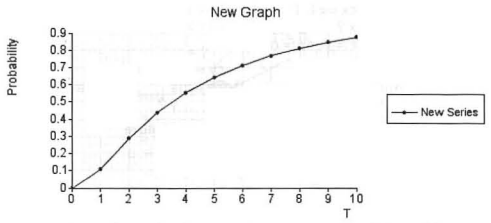
真偽判定や，クエリに対する値をもとめるアルゴリズムが存在する．同様にCSLのモデル検査アルゴリズムも示されており，これらがPRISMのツールで使われている．

3.2 自己安定分散アルゴリズム

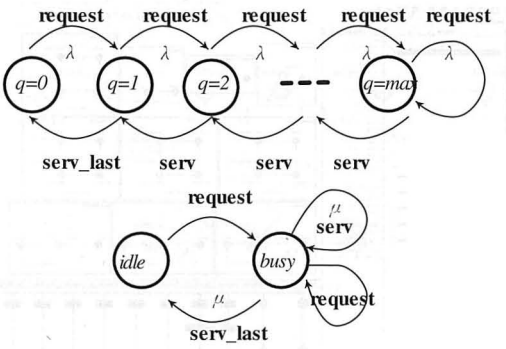
DTMCの例としてHermanの自己安定分散アルゴリズム[11]の例が示されている．このアルゴリズムは確率を使った自己安定アルゴリズムである．複数のノードがリング構成をなし，各ノードは各ラウンドで0, 1のコイントスを繰り返す．安定状態は一つのノードだけが自身の左のノードと同じ値を持つこととしている．各ノードは以下のような振る舞いをもつ．i番目のラウンドで左隣と同じ値である場合，$i+1$番目のラウンドでコイントスをする．そうでなければ，$i+1$番目のラウンドの値はi番目のラウンドの左隣の値に更新する．1番目のラウンドの初期値は各ノードとも1でスタートする．

第7図はCTMCによる7ノードのHermanアルゴリズムの記述である．一つのノード(process1)の振る舞いを記述した後，他の6ノードは変数名の名前変えをcall by nameで行い，定義している．process1の記述ではアクションstepが発生したときの動作が二つ定義されている．たとえば，前者では条件$x1 = x7$が成り立つとき実行される．実行動作は確率が付加され，変数$x1$の次状態での値が規定されている．＋は動作の選択を表す．

PRISMのプロパティータブ上で,「いつかは確率1で安定状態にたどり着くこと($P_{\geq 1}(trueU\text{“}stable\text{”})$)」や,「10ラウンド以内で安定状態にたどり着く確率が0.5以上ある($P_{>0.5}(trueU_{\leq 10}\text{“}stable\text{”})$)」ものの，1ではないことや,「安定状態にたどり着くまでのラウンド数の期待値($R_{=?}(F\text{“}stable\text{”})$) (結果は約5.49409)」などをPCTL式を使い解析することができる．第8図のようにグラフを得ることもできる．



第8図 PRISMの解析で得られたグラフ



第9図 SQとSP

3.3 パワーコントローラ

富士通のハードディスクの電源管理システムをモデル化している文献[10]の例題を紹介する．Service Queue (SQ)，Service Provider (SP)，Power Manager (PM)の三つのモジュールからなる．まずPMなしの簡単なモデルでみていく．SQはキュー長q_{max}平均到着率λの典型的な待ち行列でモデル化される．SPも単位時間あたりの平均処理数μを遷移のラベルにもつ状態遷移系としてモデル化される(第9図)．

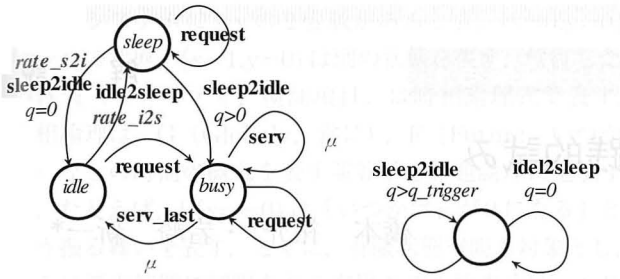
各モジュールの遷移にはアクションがラベル付けされており，各モジュールで同一アクションによる同期がとられる．なお，各モジュールは第7図と同様にテキストベースで定義される．

この二つのモジュールで典型的な待ち行列を構成しており，これだけでも様々な解析が可能である．

チュートリアルではキュー長を20，到着率を1/0.72[1/s]，処理率を1/0.008[1/s]としたときに,「状態遷移の様子」,「初期のキューを0から開始したときに，キュー長が1を超えないこと」,「初期のキュー長を20にしても定常状態ではキュー長1以下に落ち着くこと」などがツールで確認できることが示されている．

つぎにPMモジュールを追加する．PMはキュー長を監視し，それが閾値を超えた場合はSPを*sleep*から*idle*に遷移させるコマンドを発行し，0のときは逆方向に遷移させるコマンドを発行させる．SPではそれに対応した遷移を設け，各コマンドの処理時間に相当する遅延時間を与えている（第10図）．

このモデルに対して，PRISMのツールでは以下の事項がCSL式を用いて検査できる.「指定された時刻間隔におけるキューがフルになる確率を表すグラフの提示」,「定常状態においてキューがフルになる確率が指定され



第 10 図　改良された SP と PM

た値より低いことが保証されるか否か?」．これらの結果から，たとえば，定常状態でキューがフルになる確率が 0.00105 を超えることがないことや，キュー長の期待値が 3.198 であることなどが分析できる．

4. おわりに

本稿では，設計上位の形式検証（モデル検査）のツールとして時間オートマトン，確率オートマトンそれぞれのツールのあらましを述べた．

UPPAAL の WEB サイトでは適用例として，時間制約が問題となるプロトコルやバス制御，Lego MINDSTROMS^{TM} Systems を用いたシステム制御の例題などがあげられている．また SaveCCM[12] との連携も提案されている．SaveCCM はコンポーネントベース開発（CBD）の技法として提案されており，そのコンポーネントの記述モデルではコンポーネントのポートの定義を与え，コンポーネントの振る舞いを時間オートマトンで与えている．SaveCCM を用いたシステム設計をサポートする Eclipse 上の IDE も存在する．

一方，PRISM を用いたケーススタディは，確率分散アルゴリズム，通信プロトコル，セキュリティ，生体内作用，高信頼システム，ゲーム理論など多岐にわたる．これらは PRISM の WEB サイトにまとめられている．PTA を用いたケーススタディとして Firewire，CSMA/CD などがあげられている．

PRISM の欠点としてモデル検査で満たされない場合に反例が出力されないことがあげられる．UPPAAL はこの場合に反例としてそのような実行系列を得ることができる．また，PRISM にけおるモデルの入力はプロセス代数的な構文でテキストベースであるのに対し，UPPAAL は図入力ができ，直感的にはわかりやすい．UPPAAL はこの 10 数年間の例題適用実績もあり，いくつかのサブプロジェクトが展開され，実用観点からも有用性が期待できるが，商用利用には制限がある．一方で PRISM はソースコードが公開されている．研究が進み，ツールの開発も進めばシステムの抽象レベルの振る舞いを解析，検証するツールとして今後の発展が期待できる．

謝　辞

原稿執筆の機会を頂いた宮本俊幸氏（大阪大学），中島震氏（NII）に感謝いたします．

（2008 年 4 月 28 日受付）

参 考 文 献

[1] The official OMG SysML site, http://www.omgsysml.org/
[2] J. Bengtsson and W. Yi: Timed automata: semantics, algorithms and tools, *Lecture Notes on Concurrency and Petri Nets* (W. Reisig and G. Rozenberg (eds.)), *LNCS*, Vol. 3098, pp. 87–124 (2004)
[3] R. Alur and D. L. Dill: Automata for modeling real-time systems, *LNCS*, Vol. 443, pp. 322–335 (1990)
[4] R. Alur and D. L. Dill: A theory of timed automata; *Journal of Theoretical Computer Science*, Vol. 126, No. 2, pp. 183–235 (1994)
[5] T. A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine: Symbolic model checking for real-time systems; *LICS*, Vol. 92, pp. 394–406 (1992)
[6] B. Bordbar and K. Okano: Verification of Timeliness QoS Properties in Multimedia Systems; *Proc. of the ICFEM 2003, LNCS*, Vol. 2885, pp. 523–540 (2003)
[7] J. Rutten, M. Kwiatkowska, G. Norman and D. Parker: Mathematical techniques for analyzing concurrent and probabilistic systems; Vol. 23 of *CRM Monograph Series*, American Mathematical Society (2004)
[8] PRISM web site, http://www.prismmodelchecker.org/
[9] M. Kwiatkowska, G. Norman, J. Sproston and F. Wang: Symbolic model checking for probabilistic timed automata; *Information and Computation*, Vol. 205, No. 7, pp. 1027–1077 (2007)
[10] Q. Qiu and M. Pedram: Dynamic power management based on continuous-time Markov decision processes; *Proc. of Design Automation Conference*, pp. 555–561 (1999)
[11] T. Herman: Probabilistic Self-stabilization; *Information Processing Letters*, Vol. 35, No. 2, pp. 63–67 (1990)
[12] J. Carlson, J. Håkansson and P. Pettersson: SaveCCM: an analysable component model for real-time systems; *Proc. of FACS'05, Electronic Notes in Theoretical Computer Science*, Vol. 160, pp. 127–140 (2005)

著 者 略 歴

岡野　浩三（おかの　こうぞう）

1967 年 6 月 12 日生．1993 年 3 月大阪大学大学院基礎工学研究科物理系専攻博士後期課程退学（1995 年，博士（工学））．同年 4 月同大学助手，2004 年 4 月同大学大学院情報科学研究科助教授となり現在，准教授．形式的手法の研究に従事．電子情報通信学会，情報処理学会などの会員．