解　説

リアルタイムシステム

5. ハードリアルタイムシステムの検証†

米田　友洋††

1. はじめに

リアルタイムシステムでは，動作が論理的に正しくなければならないのはもちろんであるが，さらにその動作を正しいタイミングで行わなければならない．そのタイミングに対する要求が非常に強いもの，すなわち，ある処理が定められたデッドラインを越えると非常に大きな損害を被るものをハードリアルタイムシステムと呼ぶ．たとえば，航空管制システムでは，一時的な負荷の増大により処理が間に合わなくなると，飛行機が衝突してしまうかもしれないし，プラント制御システムの場合はプラントの爆発を引き起こすかもしれない．このようなハードリアルタイムシステムを設計する際には，あらゆる場合を想定してタイミングに対する要求（たとえば，デッドライン内にすべての処理を終える）を満足するよう注意しなくてはならない．このため，システムの動作をできるだけ予測可能とすることを始め，さまざまな研究がスケジューリング理論やリアルタイム OS の分野で行われている（本特集「分散リアルタイムシステムのための OS アーキテクチャ」や参考文献 1) 参照）．これらの研究によりハードリアルタイムシステムの質は飛躍的に向上しているが，特に誤りの影響の大きい設計段階では，システムの動作や仕様・要求を形式的に記述し，自動的に検証を行う必要性は大きい[2]．この検証は，静的な情報のみを用いてある程度発見的・近似的に解析する方法と，動的な動作を考慮して厳密に解析する方法に分類できる．前者として，たとえば参考文献 3) をあげることができるが，対象とするシステムがある程度限定されることと，動的な動作を考慮しないため常に悲観的な結果（最悪値）を出す．後者は，いろいろなシステムの解析が可能で，かつ非常に厳密に検証を行えるが，すべての動作を調べるため非常に多くの計算量を必要とするという欠点をもつ．しかし，最近，より少ない計算量で検証を行う方法がいろいろと提案されており，このアプローチも実用的になりつつある．

後者のアプローチを取った場合，多くの形式的検証手法で行われているように，リアルタイムシステムを有限状態機械でモデル化し，満足すべき性質，あるいは要求を時相論理などで形式的に記述することになる．ところが，通常の有限状態機械は，動作にともなう状態の遷移関係を表すものであり，動作にどれだけの時間を要するなどの実時間的制約を記述することはできない．また，時相論理も，次にある動作が起こるとか，いつかある動作が起こるなどの順序関係のみを記述し，何単位時間以内にある動作が起こるなどという実時間関係を記述できないのが普通である．そこで，リアルタイムシステムの検証においては，動作の実時間的制約を記述できるモデルを用い，かつ実時間に関する性質を記述できる論理を用いる必要がある．

本稿では，後者のアプローチによるリアルタイムシステムの検証を，いかに形式的かつ自動的に行うかについて，基本的な技術を解説する．まず **2.** でリアルタイムシステムをモデル化する手法をいくつか述べ，**3.** では実時間性を形式的に表現する方法について述べる．次に **4.** では，例を用いてどのように検証を行うかを具体的に説明し，最後に **5.** で問題点などを述べる．

2. リアルタイムシステムのモデル化

リアルタイムシステムのモデル化手法はいくつか提案されているが，ここでは，時間オートマトン

† Verification of Hard Real-Time Systems by Tomohiro YONEDA (Tokyo Institute of Technology).
†† 東京工業大学工学部情報工学科

(timed automaton)[4),5)], タイムペトリネット (time Petri net)[6),7)], モードチャート (modechart)[2),8)] をやや直感的に紹介する.

(1) 時間オートマトン

時間オートマトンは, 直感的には有限状態オートマトンに任意個のクロックとクロック制約を付加したものである. 各クロックは初期状態で値0を取り, 時間の経過とともに進む. **図-1** に時間オートマトンの一例を示す. s_0, s_1, s_2, s_3 は状態, a, b は入力アルファベット, x はクロック, $reset(x)$ はクロック x のリセット, そして $(x \leq 2)?$, $(x > 2)?$ はクロック制約を表す. 初期状態は s_0 で (何もラベル付けされていない短い矢印により示される), 状態 s_0 で a が入るとクロック x はリセットされ, 状態 s_2 に遷移する. また, 状態 s_2 で b が入ると, クロック制約により, クロック x が2以下ならば s_3 への遷移が起き, クロック x が2より大きいなら s_1 への遷移が起こる. たとえば, この時間オートマトンが時刻0で初期状態 s_0 にあるとき, 時刻2に a が, 時刻2.7に b が, 時刻2.8に a が, 時刻5に b が入ると, 以下のような状態遷移を起こすことになる.

$$\langle s_0, [0]\rangle \xrightarrow[2]{a} \langle s_2, [0]\rangle \xrightarrow[2.7]{b} \langle s_3, [0.7]\rangle \xrightarrow[2.8]{a}$$

$$\langle s_2, [0]\rangle \xrightarrow[5]{b} \langle s_1, [2.2]\rangle \cdots$$

ただし, 状態の後の $[a]$ は, その状態に遷移した時点のクロック x の値を示す. 時間オートマトン☆は受理ファミリ (acceptance family) という状態集合の集合をもち, 上記のような状態遷移系列を考えたときに, その系列中に無限回存在する状態☆☆の集合が受理ファミリの一つの要素 (受理ファミリの要素は状態の集合であることに注意) であるとき, その状態遷移系列を引き起こした入力系列を受理するという. たとえば, 上記の例で, 受理ファミリが $\{\{s_2, s_3\}\}$ ならば, s_2, s_3 のみが無限回現れる系列が受理されるから, 受理される系列では b と a の発生間隔が2より大きい場合は有限回しか生じてはならず, したがって, ある時刻以降は常に b と a の発生間隔は2以下でなくてはならない☆. 受理される入力系列のみをリアルタイムシステムの動作であると考えることが, 時間オートマトンによるリアルタイムシステムのモデル化である.

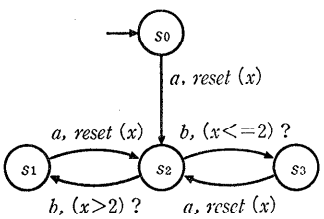


図 1 時間オートマトン

容易に予想できるように, やや大きなシステムを一つの時間オートマトンで表そうとすると, 非常に複雑となる. そこで, いくつかの時間オートマトンの積 (product) を用いて与えられたシステムを表すことが多い. 簡単な踏切制御システムを用いて, どのようにリアルタイムシステムをモデル化するかを説明する. ここで考えるシステムは, 「列車」, 「コントローラ」, 「踏切」からなる. 列車は, 踏切ゾーン (踏切の前後数百メートルの区間) に入って (*approach*) から60秒以上120秒以内に踏切に入る (*in*). 踏切に入った列車は10秒以内に踏切から出 (*out*), その後300秒以上経過した後に踏切ゾーンから出る (*gone*). 踏切ゾーンには1台の列車しか入れない. コントローラは, 列車が踏切ゾーンに入る (*approach*) と2秒以内に遮断機を下げるよう指示し (*lower*), 列車が踏切を出た (*out*) 後5秒以内に遮断機を上げるよう指示する (*raise*). 踏切は, 遮断機を下げるよう指示される (*lower*) と30秒で下げ終え (*down*), 遮断機を上げるよう指示される (*raise*) と40秒で上げ終える (*up*). 列車, コントローラ, 踏切を時間オートマトンでモデル化すると, たとえば M_T, M_G, M_C (**図-2**) のようになる. M_T, M_G, M_C の受理ファミリはそれぞれ $\{\{s_0^T, s_1^T, s_2^T, s_3^T\}\}$, $\{\{s_0^G, s_1^G, s_2^G, s_3^G\}\}$, $\{\{s_0^C, s_1^C, s_2^C, s_3^C\}\}$ である.

今考えているリアルタイムシステムは, これら三つの時間オートマトンの積を表すある一つの時間オートマトンにより表される. 積オートマトンの状態は, 各オートマトンの状態の組で表され (たとえば, 積オートマトンの初期状態は $\langle s_0^T,$

☆ 厳密には timed Muller automaton というクラスの時間オートマトン

☆☆ 非常に直感的には, 無限ループに落ち込んだ場合に, そのループ中の状態が無限回発生すると考えることができる.

☆ $s_2 \to s_3 \to s_2 \to s_1 \to s_2$ を無限に繰り返す状態遷移系列では, 無限回存在する状態の集合は $\{s_1, s_2, s_3\}$ となり, 受理ファミリに含まれないから, この系列は受理されない.

$s_0^C, s_0^G\rangle$)，クロック集合は各オートマトンのクロック集合の和集合となる．M_T と M_C では初期状態で *approach* が起こり得るが，積オートマトンではこれらが同時に起こることが要求される．すなわち，積オートマトンでは，$\langle s_0^T, s_0^C, s_0^G\rangle$ から *approach* により $\langle s_1^T, s_1^C, s_0^G\rangle$ へ遷移する．M_G は *approach* を入力アルファベットとしてもたないので，この入力により M_G に対応する状態は変化しない．また，各遷移にクロック制約やリセットされるクロック集合が付加されていたら，積オートマトンのクロック制約はそれらのクロック制約の論理的な and となり，リセットされるクロックはそれらの和集合となる．積オートマトンの受理ファミリは，各オートマトンの状態への射影がそのオートマトンの受理ファミリとなるような積オートマトンの状態集合の集合である☆．**図-3** に，M_T, M_C, M_G の積オートマトンの一部を示す．ただし，積オートマトンは各オートマトンからまったく機械的に求めることができるため，ユーザは各個別のオートマトンを与えるだけでよい．

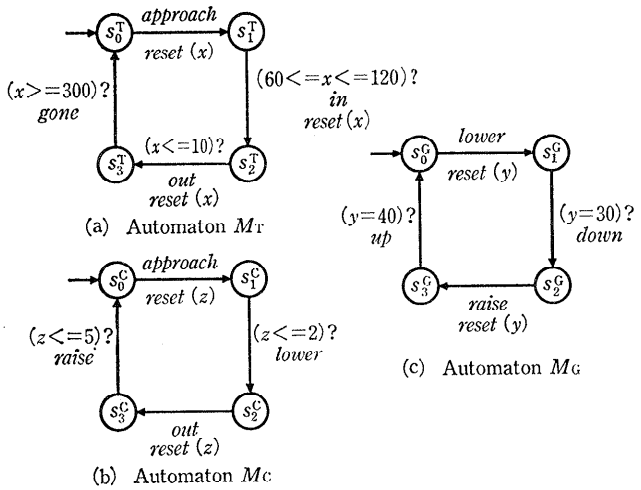


図-2　踏切制御システムを表す時間オートマトン

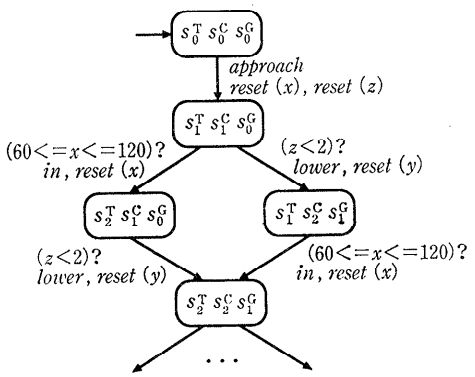


図-3　積オートマトン

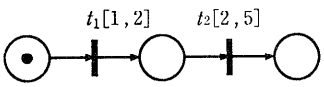


図-4　タイムペトリネット

（2）　タイムペトリネット

タイムペトリネットは，ペトリネットを時間制約をもつように拡張したもので，**図-4** に示すように，プレース（大きな丸），トランジション（太い棒），トークン（黒丸），入出力アーク（矢印）からなる．各トランジションは $[a, b]$ の形式で最小発火時間 (a) と最大発火時間 (b) をもつ．トランジションはその入力プレース（そのトランジションに向かうアークを出すプレース）すべてにトークンが存在するとき発火可能であるという．トランジションの発火とは，入力プレース中のトークンを取り去り，出力プレース（そのトランジションから出るアークを受けるプレース）にトークンを出力することである．タイムペトリネットでは，トランジションは最小発火時間の間発火可能であり続けなければ発火できず，また，最大発火時間を過ぎて発火可能であり続けることはできない．発火可能なトランジションは，上記の条件を満足するかぎり，任意の時刻に発火できる．発火は瞬間的に起こり，発火に要する時間は 0 である．図-4 の例では，トランジション t_2 はトランジション t_1 の発火後，2単位時間以降，5単位時間以内にのみ発火できることになる．このようなタイムペトリネットの発火規則により，容易にリアルタイムシステムの時間幅をもった動作を記述できる．

前記の踏切制御システムをタイムペトリネットで記述すると，**図-5** のようになる．タイムペトリネットでは，複数のトークンを用いることができるため，このように並列動作を含むシステムも，比較的容易に直接記述できる．

（3）　モードチャート

モードチャートは，リアルタイムシステム用に設計されたグラフィカルな仕様記述言語であり，

☆厳密には，各オートマトン中の自己ループを取り除く必要がある場合もある．

システムの動作モードを表す「モード」とそのモード間の遷移関係からなる．モード（四角形で表される）には，

- それ以上内部にモードをもたないプリミティブモード，
- 内部にいくつかのモードをもち，システムは同時にそれらのモードで実行されるパラレルモード，
- 内部にいくつかのモードをもつが，システムは逐次的にそれらの一つのモードをとるシリアルモード

がある．各モードは高々一つのアクションをもち，そのモードに入るときに実行される．モード間の遷移関係は矢印で表され，状態変数や発生イベントによる遷移条件（時間制約を含む）をもつことができる．

前記の踏切制御システムをモードチャートで記述すると**図-6** のようになる．ここで，TRAIN, GATE は状態変数で，アクションによりそれらの値が更新される．たとえば，Action : GATE := DOWN [0, 2] は，CONTROLLER モードの中の gate down というモードに入ったときに実行され，状態変数 GATE の値が DOWN とされ，そのときに要する時間は 0 以上 2 以下であることを示す．モード間の枝には遷移条件が付され，たとえば (TRAIN=NEAR)? は状態変数 TRAIN の値が NEAR のときにこの遷移が起こることを表す．また，TRAIN モード中の crossing モードと passed モード間に付された [0, 10] は，これらの間の遷移が 0 以上10以内に起こることを示す．"train exits crossing" などはコメントである．

以上，三つのモデル化手法を紹介した．モードチャートはリアルタイムシステム用に設計されたもので，ドキュメント性が一番高いといえる．他の二つは理論的解析に向いたモデル化手法であり，特に，時間オートマトンの記述能力は三つの中で最も高いが，小さなシステム以外の記述には向かないであろう．タイムペトリネットは並列動作も容易に記述できるため，中規模程度のシステムの記述も可能であると思われる．

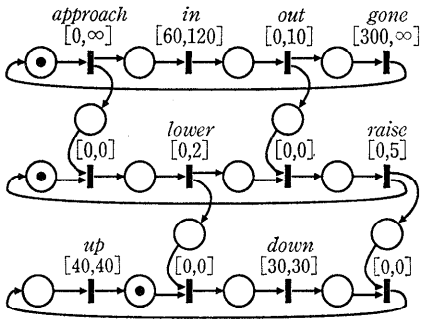


図-5 踏切制御システムを表すタイムペトリネット

3. 検証すべき性質の記述

検証したい実時間関係の記述には，時間オートマトンを用いる方法[5]と実時間論理[9]を用いる方法がある．

（1） 時間オートマトンによる方法

目的の実時間関係はある特定の動作系列として記述できるから，その動作系列を受理する時間オートマトンにより記述することができる☆．たとえば，前章の踏切制御システムにおいて，「列車が踏切内を通過中は必ず遮断機が降りている」という性質は**図-7** の時間オートマトンで記述さ

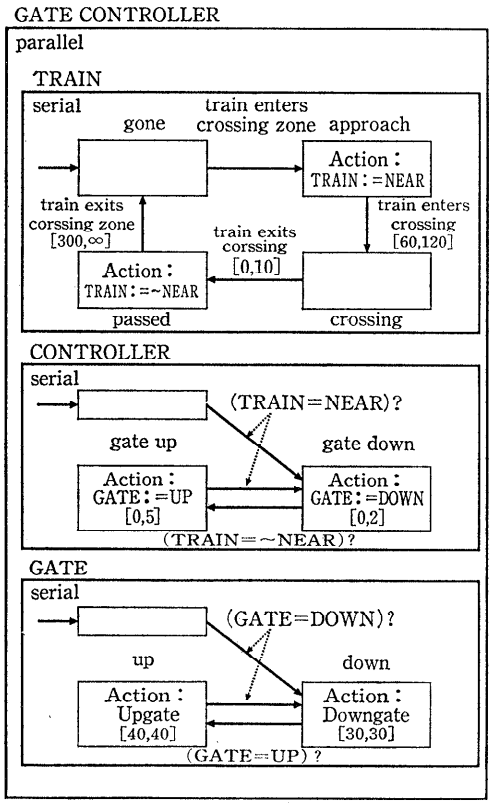


図-6 踏切制御システムを表すモードチャート

☆ 厳密には決定性の timed Muller automaton を用いる．非決定性の時間オートマトンに対する効率のよい検証アルゴリズムは存在しない．

れる．受理ファミリは $\{\{s_a, s_b, s_c\}\}$ である．ここで，"～" は図-2 の各時間オートマトンの入力アルファベット集合の和集合 $\{approach, in, out, gone, lower, raise, down, up\}$ に対する補集合を表す．たとえば，"$\sim in, \sim down$" は $\{approach, out, gone, lower, raise, up\}$ に等しい．これは明らかに，遮断機が降りてから列車が入り，列車が通り過ぎてから遮断機が上がることを意味している*．また，「遮断機は 200 秒以上連続して降りていることはない」という性質は図-8 により表される．受理ファミリは $\{\{s_a, s_b\}\}$ である．

（2） 実時間論理による方法

まず，実時間に関与しない時相論理として CTL[10),11)] のサブセットを簡単に紹介する．図-2 の時間オートマトンにおいて，あるオートマトンが状態 s にあるとき真となるような命題変数 s を考える．すなわち，踏切制御システムの初期状態では三つの命題変数 s_0^T, s_0^G, s_0^G が真である．命題変数は時相論理式である．f, g を時相論理式とするとき，$\neg f$ や $f \vee g$ も時相論理式であり，その意味は通常どおりである．時相演算子としてよく使われるのは EX, EF, AG である．EXf, EFf, AGf は時相論理式であり，その意味は以下のとおりである．

- 状態 s から 1 回の遷移で到達できる少なくとも一つの状態で f が成り立つとき，状態 s において EXf が成り立つ．
- 状態 s から到達できる少なくとも一つの状態で f が成り立つとき，状態 s において EFf が成り立つ．

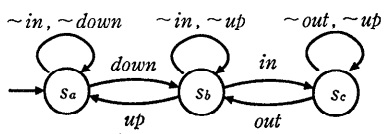


図-7 検証すべき性質（1）

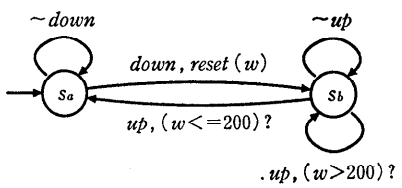


図-8 検証すべき性質（2）

☆ 図-7 で，s_a の "$\sim in$" は，"$down$" よりも先に "in" が起こってはならないことを示すもので，省略できない．s_c の "$\sim up$" も同様である．

- 状態 s から到達できるすべての状態で f が成り立つとき，状態 s において AGf が成り立つ．

上記の例で，列車が踏切内を通過中は，M_T は状態 s_2^T を取るし，遮断機が降りているときは状態 s_2^G あるいは s_3^G を取るから，「列車が踏切内を通過中は必ず遮断機が降りている」は，式 EF $\{s_2^T \wedge \neg(s_2^G \vee s_3^G)\}$ が初期状態で偽となることと等価である．

ところが，これらの時相演算子では「遮断機は 200 秒以上連続して降りていることはない」のような定量的な時間関係を記述できない．そこで，実時間論理 TCTL[12)]☆ では，状態間の時間関係を陽に記述できる EF$_{\leq c}$, EF$_{>c}$, AG$_{\leq c}$, AG$_{>c}$ などの時間演算子を用いる☆☆．EF$_{\leq c}$, AG$_{\leq c}$ の意味は次のとおりである（他の時間演算子の意味も同様）．

- 状態 s から到達できる時間差 c 以下のある状態で f が成り立つとき，状態 s において EF$_{\leq c}f$ が成り立つ．
- 状態 s から到達できる時間差 c 以下のすべての状態で f が成り立つとき，状態 s において AG$_{\leq c}f$ が成り立つ．

これを用いると，その先少なくとも 200 秒連続して遮断機が降りている状態では，AG$_{\leq 200}$ $(s_2^G \vee s_3^G)$ が成り立つといえる．したがって，そのような状態に到達可能でない，すなわち EF $[$AG$_{\leq 200}(s_2^G \vee s_3^G)]$ が初期状態で偽であることと上記の性質は等価となる．

このほか，イベントの発生時刻を陽に参照する論理もある（RTL[13)]，TNL[14)] など）．たとえば，RTL ではアクション a の開始，終了をイベント $a\uparrow$, $a\downarrow$ で表し，その i 番目の発生時刻を @$(a\uparrow, i)$, @$(a\downarrow, i)$ で表す．したがって，図-6 のモードチャートに関して「遮断機は 200 秒以上連続して降りていることはない」という性質は，$\forall i$, @$(Upgate\uparrow, i)$ − @$(Downgate\downarrow, i) < 200$ で表すことができる．

4. 検証の実際

検証とは，モデル化されたシステムが与えられた性質を常に満足するかどうかを調べることである．システムのモデル化の方法，性質の記述方法

☆ 実際には多くのバリエーションがある．参考文献 9) にサーベイが示されている．

☆☆ これも TCTL のサブセットであり，また若干記法も変えてある．

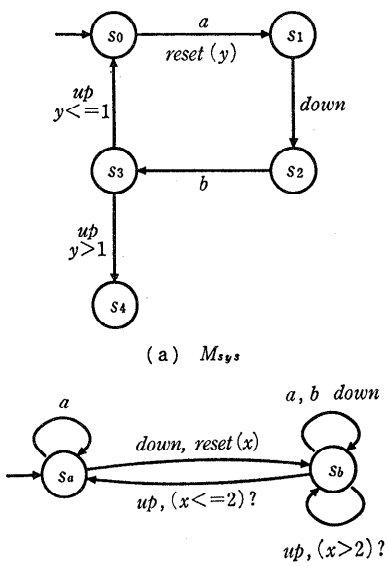
により検証方法も異なるが，ここでは，システムおよび検証すべき性質がともに時間オートマトンで表されている場合について具体的に検証方法を述べる．

今，システムを表す時間オートマトンを M_{sys} (いくつかのオートマトンとして与えられている場合はそれらの積オートマトン)，検証すべき性質を表す時間オートマトンを M_{pr} とする．$L(M_{sys})$ を M_{sys} により受理されるアルファベットの集合（システムの取る動作系列を意味する）を表し，$L(M_{pr})$ を M_{pr} により受理されるアルファベットの集合（検証すべき性質を満足する動作系列を意味する）を表すものとする．上記の検証の定義より，検証が成功することと $L(M_{sys})\subseteq L(M_{pr})$ が成立することは同値である．この包含関係は，実際には $L(M_{sys})\cap\overline{L(M_{pr})}=\emptyset$ が成り立つかどうかにより調べる．ただし，$\overline{L(M_{pr})}$ は $L(M_{pr})$ の補集合である．簡単な例を用いて，具体的な方法を説明する．M_{sys} を**図-9**（a）に示すもの（受理ファミリは $\{\{s_0, s_1, s_2, s_3\}\}$），$M_{pr}$ を**図-9**（b）に示すもの（受理ファミリは $\{\{s_a, s_b\}\}$）とする．この例では検証結果は真である．

まず，M_{pr} の状態の集合を S，受理ファミリを $\mathcal{F}$ とすると，受理ファミリが $2^S-\mathcal{F}$ であること以外 M_{pr} と同じである時間オートマトン M_{pr}^c を考える．ただし，2^S は S のすべての部分集合の集合を表す．このとき，$\overline{L(M_{pr})}=L(M_{pr}^c)$ が成り立つ．今考えている例では，M_{pr}^c は図-9（b）と同じで，受理ファミリとして $\{\{s_a\}, \{s_b\}, \{\}\}$ をもつ．この性質を用いると $L(M_{sys})\cap L(M_{pr}^c)$ を求めればよいわけであるが，受理言語の積集合は積オートマトンの受理言語に等しいことが分かっている．したがって，M_{sys} と M_{pr}^c の積オートマトン（これを M とする）の受理言語が空であるかどうかをしらべればよい．M を**図-10**に示す．M の受理ファミリは，M_{sys} の状態集合への射影が $\{s_0, s_1, s_2, s_3\}$ であり，M_{pr}^c への射影が $\{s_a\}$, $\{s_b\}$, または $\{\}$ となる M の状態集合の集合であるから，

$$\{\{\langle s_0, s_a\rangle, \langle s_1, s_a\rangle, \langle s_2, s_a\rangle, \langle s_3, s_a\rangle\},$$
$$\{\langle s_0, s_b\rangle, \langle s_1, s_b\rangle, \langle s_2, s_b\rangle, \langle s_3, s_b\rangle\}\}$$

となる．この例では，状態 $\langle s_3, s_b\rangle$ において明らかに $x\leq y$ なので，この状態から状態 $\langle s_0, s_b\rangle$ への遷移は起こらない．一方，状態 $\langle s_0, s_a\rangle$ への遷移は起こり得る．したがって，無限回現れる状態，すなわち，ループ中に含まれる状態は $\langle s_0, s_a\rangle, \langle s_1, s_a\rangle, \langle s_2, s_b\rangle, \langle s_3, s_b\rangle$ だけであるから，これらは受理されない．このほかにループに含まれる状態はないから，M の受理言語は空ということになり，これは検証結果が真であることを意味する．この例は非常に簡単なので，時間制約により起こりえない遷移（$\langle s_3, s_b\rangle\rightarrow\langle s_0, s_b\rangle$）を簡単に見つけることができたが，一般には容易ではない．そこで，このような時間関係の判定を自動的に行うために，M の状態とそのとき取り得るクロックの状態を組とし，これを状態とする新たなオート



(a) M_{sys}　(b) M_{pr}

図-9 例

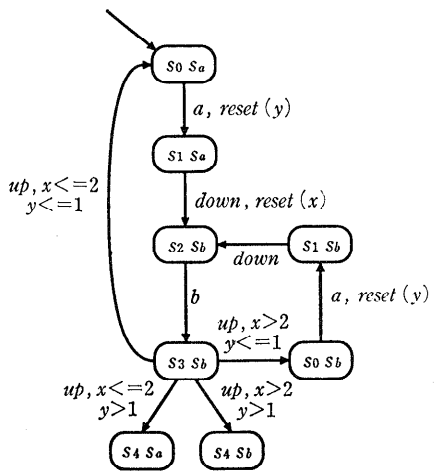


図-10 M_{sys} と M_{pr}^c の積オートマトン

マトン M' を作る．クロックの状態も M' の状態に含まれるから，時間制約が満足されるかどうかは M' の状態により一意に決定できる．したがって，M' はもはや時間制約をもたない．このようなオートマトンの受理言語が空かどうかのチェックは，上記の例から類推できるように，単にループを見つけることに帰着できる．詳細については参考文献 4) などを参照されたい．

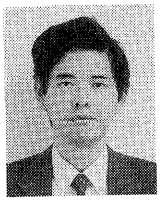
5. おわりに

リアルタイムシステムの検証を形式的かつ自動的に行う方法について，簡単に，しかしできるかぎり具体的に紹介したつもりである．いろいろな検証方法のごく一部ではあるが，どのようにしてリアルタイム性の形式的検証を行うかについて理解していただければ幸いである．前章では非常に基本的な検証方式を述べるにとどまったが，実用的なシステムの検証を行うためには，探索する状態空間を縮小し，検証に必要な計算量を減らす工夫が必要である．リアルタイムシステムの検証に限らず，これは状態空間を探索するタイプの自動検証方式における重要な問題で，現在おもに非リアルタイム性の検証方式において，さまざまな研究が行われている．これらの技術が，リアルタイム性の検証に応用されているものもあるが（たとえば参考文献 14) では，プレース数が数百程度のタイムペトリネットを実用的な計算時間で扱えるようになったことが示されている．），今後，さらに活発な研究が期待される．

参考文献

1) Raijkumar, R.: *Synchronization in Real-Time Systems, A Priority Inheritance Approach*, Kluwer Academic Publishers (1991).
2) Tilborg, A. M. and Koob, G. M. (eds.): *Foundations of Real-Time Computing, Formal Specifications and Methods*, Kluwer Academic Publishers (1991).
3) Leinbaugh, D. W. and Yamini, M. R.: Guaranteed Response Times in a Distributed Hard-Realtime Environment, *IEEE Trans. on Software Eng.*, 12(12), pp. 1139-1144 (1986).
4) Alur, R. and Dill, D.: The Theory of Timed Automata, *LNCS 443*, pp. 322-335 (1990).
5) Alur, R. and Dill, D.: Automata for Modeling Real-Time Systems, *LNCS 600 Real-Time: Theory in Practice*, pp. 45-73 (1992).
6) Merlin, P. and Faber, D. J.: Recoverability of Communication Protocols, *IEEE Trans. on Communication*, COM-24(9) (1976).
7) Berthomieu, B. and Diaz, M.: Modeling and Verification of Time Dependent Systems using Time Petri Nets, *IEEE Trans. on Software Eng.*, 17(3), pp. 259-273 (1991).
8) Jahanian, F. and Stuart, D. A.: A Method for Verifying Properties of Modechart Specification, *Proc. Real-Time Systems Symposium*, pp. 12-21 (1988).
9) Alur, R. and Henzinger, T. A.: Logics and Models of Real Time: A survey, *LNCS 600 Real-Time: Theory in Practice*, pp. 74-106 (1992).
10) Clacke, E. M., Emerson, E. A. and Sistla, A. P.: Automatic Verification of Finite-State Concurrent Systems using Temporal-Logic Specifications, *ACM Trans. on Programming Languages and Systems*, 8(2), pp. 244-263 (1986).
11) Yoeli, M.: *Formal Verification of Hardware Design*, IEEE computer society press (1990).
12) Alur, R., Courcoubetis, C. and Dill, D.: Model-Checking for Real-Time Systems, *Proc. of 5th IEEE LICS* (1990).
13) Jahanian, F. and Mok, A. K.: Safety Analysis of Timing Properties in Real-Time Systems, *IEEE Trans. on Software Eng.*, 12(9), pp. 890-904 (1986).
14) Yoneda, T., Shibayama, A., Schlingloff, H. and Clarke, E. M.: Efficient Verification of Parallel Real-Time Systems, *LNCS 697 Computer Aided Verification*, pp. 321-332 (1993).

(平成5年6月23日受付)



米田 友洋（正会員）

1957 年生．1980 年東京工業大学工学部情報工学科卒業．1985 年同大学院博士課程修了．同年同大学助手．1989 年同講師．1990 年～1991 年カーネギーメロン大学客員研究員，1991 年同助教授となり現在に至る．工学博士．フォールトトレラント設計，形式的設計検証に関する研究に従事．電子情報通信学会，IEEE 各会員．