ミニ特集 [解説]
安心・安全システム構築のための形式検証技術

時間オートマトンによる実時間システムの形式的検証

中 田 明 夫*

*広島市立大学 大学院情報科学研究科 〒 731-3194 広島市安佐南区大塚東 3-4-1
*Graduate School of Information Sciences, Hiroshima City University, 3-4-1, Ozuka-higashi, Asaminami-ku, Hiroshima 731-3194, Japan
*E-mail: nakata@hiroshima-cu.ac.jp

キーワード：**形式的検証** (Formal Verification), **実時間システム** (Real-time Systems), **時間オートマトン** (Timed Automata), **並列合成** (Parallel Composition), **モデル検査** (Model Checking)
JL 0008/03/4208–0685 ©2003 SICE

1. はじめに

近年，航空機・自動車や FA 機器から携帯電話・情報家電など広範囲な分野でコンピュータシステムが使用されてきている．それらのシステムの多くは設計誤りが人命にかかわったり多大な損害につながるなどのため，高い信頼性が要求される．一方で，これらのシステムの多くは定められた時間制約内に動作を行うことを要求される実時間システムとなっている．高信頼なシステムの設計開発技術の一つとしてモデル検査[1] などの形式的検証技術が近年注目を集めており，実時間システムを対象としたモデル検査技術およびツールもいくつか提案されてきている．実時間システムのモデルとしては時間オートマトン[2] が良く用いられている．

時間オートマトンは，有限オートマトン[3] の拡張であり，システムを離散的なイベントと連続的な時間経過の両方による状態遷移で記述するモデルである．時間オートマトンでは連続的な時間経過を扱うため，一般にクロックによる同期のない物理的な制御対象との相互作用を自然な形で記述可能である．また，離散イベント間の時間制約を記述可能であり，時間的な挙動と離散的な状態遷移が相互に影響しあう動作を記述できる．さらに，イベントと時間のみに着目し，他の観測量，制御量や具体的な入出力データなどを捨象するため，効率の良いモデル検査が可能で，実用に耐えるツールも近年整備されてきている．

本稿では，時間オートマトンによる実時間システムの振る舞いのモデル化手法，および，安全性などの性質の形式的検証技術の概要について，技術の詳細な内容には立ち入らず，例題とツールを用いた検証の実例を交えて紹介する．

2. 時間オートマトン

2.1 定義

時間オートマトンの定義を図 1 を用いて説明する．時間オートマトンは有限個の**クロック変数**，有限個の**ロケーション**，ロケーション間の**遷移**，および，**初期ロケーション**の指定，から構成される．

クロック変数はシステムの連続的な状態である時間経過を記憶する変数であり，一般に一つの時間オートマトンにおいて複数個のクロック変数を持つことができる．各クロック変数は一般に非負実数値を取り，初期値はゼロであり，時間経過と共に同じレートで連続的に値が増加する変数である．各クロック変数は，後述のクロック変数の更新によってそれぞれ独立にゼロにリセットすることができる．図 1 では，x と y がクロック変数である．

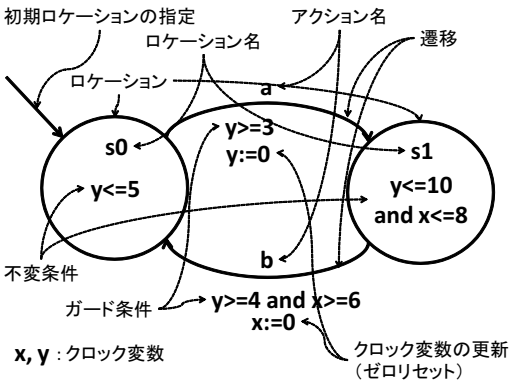

図 1 時間オートマトン
時間オートマトンはクロック変数，ロケーション，不変条件，ロケーション間の遷移，および，初期ロケーションの指定，から構成される．クロック変数は時間の経過と共に連続的に値が増加する実数型変数である．各ロケーションはロケーション名および不変条件という属性を持つ．不変条件はそのロケーションに滞在中にクロック変数が満たすべき条件を表す．各遷移は，アクション名，ガード条件，および，クロック変数の更新という属性を持つ．ガード条件は遷移を実行可能となるための各クロック変数の現在値に対する条件式である．クロック変数の更新は，遷移を実行後にゼロにリセットするクロック変数を指定するものである．

ロケーションはシステムの離散的な状態（制御状態）を表しており，有限オートマトンにおけるいわゆる「状態」に対応する．ロケーションを特定するために各ロケーションに名前 (**ロケーション名**) をつけることができる(注1)．各ロケーションには**不変条件**を付与することが出来る．不変条件はそのロケーションに滞在している間に必ず成り立つべきクロック変数に関する条件であり，一般にクロック変数に対する上限制約 (「ある整数以下」，または,「ある整数未満」の形の条件) の論理積で記述する．不変条件で制約されないクロック変数に関してはそのロケーションに滞在中

(注1) ロケーション名は必ずしもつける必要は無いが，後述のモデル検査においてロケーションを指定する目的で用いる．

に任意の値が可能であるとする．また不変条件を指定しない場合はすべてのクロック変数が任意の値を取りうるとする．図 1 では，丸がロケーションを表し，丸の中に記載された s0,s1 がロケーション名である．また，ロケーション s0 の不変条件として y<=5, ロケーション s1 の不変条件として y<=10 and x<=8 が指定されている．

ロケーション間の遷移は，(遷移元ロケーション，アクション名，ガード条件，クロック変数の更新，遷移先ロケーション) の 5 つ組で構成される．**アクション名**はその遷移が表す離散イベントを特定する名前である．**ガード条件**は，クロック変数と整数の大小比較(注2)およびそれらの論理積で記述可能であり，クロック変数の現在の値がガード条件を満たすときに遷移が実行可能となることを意味する(注3)．**クロック変数の更新**は，遷移を実行時にどのクロック変数をゼロにリセットするかを指定したものであり，一般に一つの遷移に対して複数のクロック変数を指定することが出来る．図 1 では，ロケーションの間に引かれた実線矢印が遷移を表し，矢印の元が遷移元ロケーション，矢印の先が遷移先ロケーションである．また各矢印の脇にアクション名，ガード条件，クロック変数の更新を記載している．クロック変数の更新（ゼロリセット）は，各クロック変数に 0 を代入する文 (x:=0 など) で表現している．図 1 では一つのクロック変数しか更新していないが，一般には複数のクロック変数を同時に更新することもできる．その場合は図においてはカンマで区切って記述することにする (x:=0,y:=0 など)．

時間オートマトンは**初期ロケーション**において各クロック変数がすべてゼロである状態から動作を開始する．初期ロケーションの指定は図 1 では，左上からロケーションに入る実線矢印が初期ロケーションの指定を表す．

2.2 動作意味

時間オートマトンの正確な動作は，時間オートマトンの「実際の」状態を考えることで理解することが出来る．時間オートマトンの実際の状態は，ロケーションと各クロック変数の現在の値の組で表現される(注4)．例えば図 1 の時間オートマトンの実際の状態遷移は図 2 のように表すことができる．図 2 において，例えば (s0,[x=0,y=0]) は，ロケーション s0 において，クロック変数 x と y が共に 0 である状態を表す．

実際の状態遷移は時間経過による連続的状態遷移とアクションによる離散的状態遷移の 2 種類に分類される．

時間経過による遷移は，一つのロケーションに滞在中の遷移であり，ロケーションは変化せず，すべてのクロック変数は同じ値だけ連続的に増加する．ただし，不変条件を満たす範囲内でしか時間経過は許されない．例えば，図 1 においてロケーション s0 の不変条件は y<=5 であり，[x=3.2,y=3.2] というクロック変数の値はこの不変条件を満たす．従って，図 2 に示すように，状態 (s0,[x=0,y=0]) から，3.2 単位時間経過によって状態 (s0,[x=3.2,y=3.2]) に遷移できる．なお，図 2 では 3.2 単位時間経過という一つの具体的な時間経過しか記していないが，実際には不変条件を満たすような任意の非負実数値による時間経過遷移が無数に存在する．

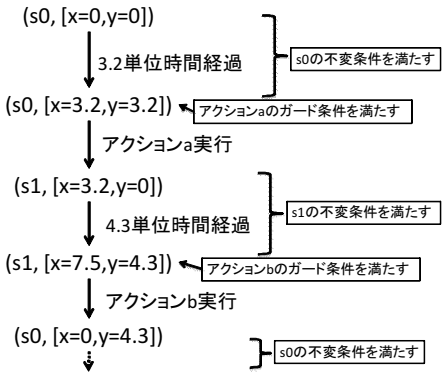

図 2　時間オートマトンの動作意味
時間オートマトンの実際の状態はロケーションと各クロック変数の現在値の組で表現される．時間経過による遷移は，一つのロケーションに滞在中の遷移であり，ロケーションは変化せず，すべてのクロック変数は同じ値だけ連続的に増加する．ただし，不変条件を満たす範囲内でしか時間経過は許されない．アクションによる状態遷移は，時間は経過せず瞬時にロケーションを変化させる遷移であり，現在のロケーションで実行可能な遷移のガード条件を満たすときに実行可能である．アクションによる遷移先状態では，クロック変数の更新で指定されたものに関しては値がゼロにリセットされる．

アクションによる状態遷移は，時間は経過せず瞬時にロケーションを変化させる遷移であり，現在のロケーションで実行可能な遷移のガード条件を満たすときに実行可能である．アクションによる遷移先状態では，クロック変数の更新で指定されたものに関しては値がゼロにリセットされる．また，遷移先での各クロック変数の値は遷移先ロケーションの不変条件を満たす必要がある．遷移後に不変条件を満たさないような遷移は最初から許されない．従って，遷移の実行可能性の判断のためには，ガード条件に加えて，クロック変数の更新後に遷移先の不変条件を満たすか否かもチェックする必要がある．例えば，図 1 においてアクション a による遷移のガード条件は y>=3 であり，[x=3.2,y=3.2] というクロック変数の値はこのガード条件を満たす．また，この遷移のクロック変数の更新として y:=0 が指定されており，この更新を適用後のクロック変数の値 [x=3.2,y=0] は，遷移後

(注2) 一般には，2 つのクロック変数の差と整数との大小比較も記述可能である．
(注3) 正確には，後述するようにガード条件に加えて，クロック変数の更新の実行後の各クロック変数の値が遷移先ロケーションの不変条件を満足しなければ，その遷移を実行できない．
(注4) 先の定義で制御状態を「状態」と呼ばず「ロケーション」と呼ぶのは，このことが理由である．

のロケーション s1 の不変条件 [y<=10 and x<=8] を満たす．従って，図 2 に示すように，状態 (s0,[x=3.2,y=3.2]) から，アクション a によって状態 (s1,[x=3.2,y=0]) に遷移できる．

同様に考えることで，図 1 の時間オートマトンは，状態 (s1,[x=3.2,y=0]) から 4.3 時間経過しアクション b を実行し，ロケーション s0 に戻るという一連の遷移も可能である (図 2)．ここで注意してほしいのは，同じロケーションでもクロック変数の値が異なれば異なる状態になる点である．例えば図 2 において，アクション b 実行後に遷移する状態は (s0,[x=0,y=4.3]) であって，初期状態 (s0,[x=0,y=0]) とはクロック変数 y の値が異なるため，以後の可能な動作系列が異なる可能性がある．

2.3 並列合成

制御システムに限らず，一般のコンピュータシステムは複数のサブシステムが相互に通信する並行/分散システムとして構成されることが多い．時間オートマトンについても，複数のサブシステムを記述した時間オートマトンの並列合成として記述することにより，大きなシステム全体の動作を簡潔に分かりやすく表現することができる．

時間オートマトンの並列合成はプロセス代数[4] と同様の定義に基づく．まず，並行に動作する複数の時間オートマトンの組に対して，各時間オートマトンのロケーションの組を，並列合成された時間オートマトンのロケーションとする．これは有限オートマトンにおける積オートマトンの構成法と同じ考え方である．次に，各時間オートマトン間の通信を表現するために次の概念を導入する．2 つ以上の時間オートマトンに共通に現れるアクション名があれば，そのアクションを**同期アクション**と呼び，さもなければ**非同期アクション**と呼ぶ．

非同期アクションに関しては，並行に動作する各時間オートマトンにおいて独立に実行，すなわち，どれが先に実行されてもかまわない（任意の順序が可能）とする．

一方，同期アクションに関しては，そのアクションを持つすべての時間オートマトンが同時に実行可能であるときのみ，そのアクションが実行可能である（すなわち，同期実行しなければならない）とする．

例えば，図 3 左側の 2 つの時間オートマトンにおいては，a と b が非同期アクション，c が同期アクションである．これらの時間オートマトンの並列合成は図 3 右側のようになる．ここでロケーション q0 ではアクション c が実行可能であるが，s0 では実行できないので，並列合成後のロケーション (s0,q0) ではアクション c による遷移はない．しかし，非同期アクション a の実行後のロケーション (s1,q0) では同期アクション c がロケーション s1 と q0 の両方で実行可能であるため，c による遷移が存在する．

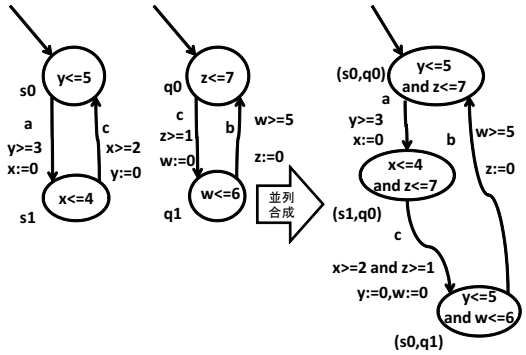

図 3　並列合成
図左側の 2 つの時間オートマトンの並列合成は右の時間オートマトンのようになる．アクション a および b は非同期アクション，アクション c は同期アクションである．同期アクション c はロケーション (s1,q0) で初めて実行可能となる．

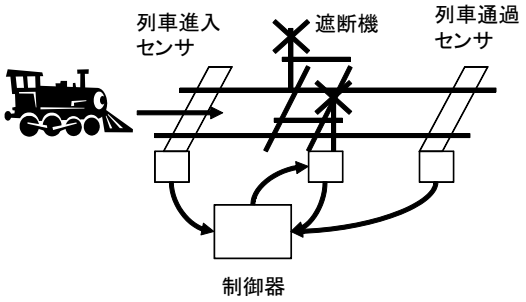

図 4　踏み切り制御システムの構成
線路上の踏み切りの前後にはセンサが取り付けてあり，踏み切りへの列車の進入および通過を検知することができる．遮断機を制御する制御器は，センサからの列車検知情報に基づいて，遮断機に対して開閉の命令を出す．

2.4 記述例：踏み切り制御システム

時間オートマトンによるシステムの記述例として，図 4 のような踏み切り制御システムを取り上げる．簡単のため，列車は左から右への一方向にのみ通過するものとする．線路上の踏み切りの前後にはセンサが取り付けてあり，踏み切りへの列車の進入および通過を検知することができる．遮断機を制御する制御器は，センサからの列車検知情報に基づいて，遮断機に対して開閉の命令を出すものとする．

このような踏み切り制御システムの時間オートマトンによるモデル化を考える．

まず，列車の動作は図 5 の時間オートマトン TRAIN で記述できる．列車はまず初期ロケーションの far(遠くにいる) におり，踏み切りに接近し，列車進入センサで検知されることで，ロケーション approaching(接近中) にアクション approach で遷移する．このとき，クロック変数 x をリセットし，列車が進入してからの時間を計測する．ロケーション approaching, crossing,leaving の各不変条件 x<=7 は，列車は接近 (approach) から 7 単位時間以内

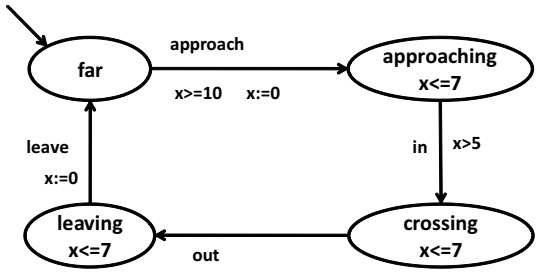

図 5　列車のモデル TRAIN
列車の動作は図の時間オートマトン TRAIN で記述できる．ロケーションは，far(遠くにいる), approaching(接近中), crossing(踏み切り通過中),leaving(離脱中) の 4 つで，アクションは approach(接近列車進入センサで検知), in(踏み切りに進入), out(踏み切りを通過), leave(列車通過センサで検知) の 4 つである．列車は接近から踏み切り内に進入するまで 5 単位時間超かかり，接近から 7 単位時間以内に通過センサで検知される．次の列車が再び接近するまでには 10 単位時間以上の時間間隔がある．

に踏み切りを離脱 (leave) することを表している．ロケーション approaching からさらに列車が進行すると，踏み切り内に進入 (in) し，ロケーション crossing(踏み切り通過中) に遷移する．アクション in のガード条件 x>5 によって，列車は踏み切りに接近してから 5 単位時間超の時間経過後に踏み切りに進入することを表している．ロケーション crossing において列車が踏み切り外に出ると (out), ロケーション leaving(離脱中) に遷移する．ロケーション leaving において，さらに列車が踏み切りから離れて列車通過センサに検知されると (leave), ロケーション far に戻る．このとき，クロック変数 x を再度リセットし，列車が踏み切りを通過してから次の列車が踏み切りに到着するまでの時間を計測する．アクション approach のガード条件 x>=10 によって，前の列車の通過から少なくとも 10 単位時間以上の間隔をおいて次の列車が進入することを表している．

遮断機の動作は図 6 の時間オートマトン GATE で記述できる．初期ロケーションは open(開いている) であり，アクション lower により closing(閉めている) に遷移すると同時に，クロック変数 y および w をリセットする．ロケーション closing では不変条件 y<=2 が指定されており，lower から 2 単位時間以内に，次のアクション down を実行し，ロケーション close(閉まっている) に遷移しなければならないことが指定されている．down のガード条件には y>=1 が指定されているので，合わせて「lower から 1 単位時間以上，2 単位時間以内に down を実行する」ことが指定されている．これは，遮断機を閉め始めてから完全に閉まるまでの時間が 1 単位時間以上，2 単位時間以内であることを表している．ロケーション close からは raise によってロケーション opening(開けている) に遷移し，クロック変数

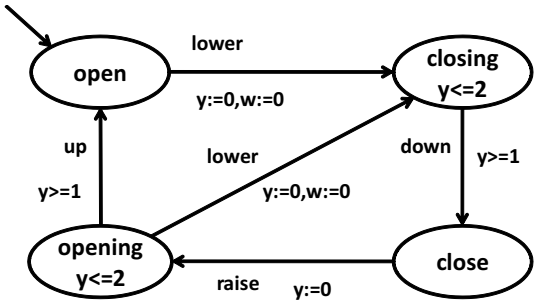

図 6　遮断機のモデル GATE
遮断機の動作は図の時間オートマトン GATE で記述できる．ロケーションは open(開いている), closing(閉めている)，close(閉まっている), opening(開けている) の 4つで，アクションは lower(閉め始める)，down(完全に閉まる)，raise(開け始める), up(完全に開く) の 4 つである．閉め始めてから完全に閉まるまで 1 単位時間以上 2 単位時間以内，開け始めてから完全に開くまで 1 単位時間以上 2 単位時間以内の時間がそれぞれかかる．

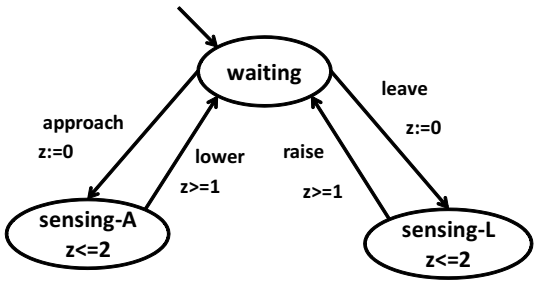

図 7　制御器のモデル CONTROLLER
制御器の動作は図の時間オートマトン CONTROLLER で記述できる．ロケーションは waiting(入力待ち状態), sensing-A(列車進入検知) sensing-L(列車通過検知) の 3 つである．approach,leave は列車進入センサおよび列車通過センサから制御器に対して検知信号が入力されるイベントを表す．lower,raise はそれぞれ遮断機を閉める/開けるという命令を制御器から遮断機に対して送信するイベントを表す．制御器が列車進入を検知してから遮断機を閉める命令を送信するまで 1 単位時間以上 2 単位時間以内，制御器が列車通過を検知してから遮断機を開ける命令を送信するまで 1 単位時間以上 2 単位時間以内の時間がそれぞれかかることが指定されている．

y をリセットする．次のロケーション opening および次のアクション up には同様に y に関する不変条件とガード条件が指定されており，遮断機を開け始めて (raise) から完全に開く (up) までの時間が 1 単位時間以上，2 単位時間以内であることを指定している．なお，遮断機を開けている途中でも閉めることができるように，ロケーション opening からもアクション lower が実行できるようにしている．

制御器の動作は図 7 の時間オートマトン CONTROLLER で記述できる．図 7 において approach,leave は前述の列車のモデルとの同期アクションであり，それぞれ列車進入センサおよび列車通過センサから制御器に対して検知信号が入力されるイベントを表す．また，lower,raise は前述の遮

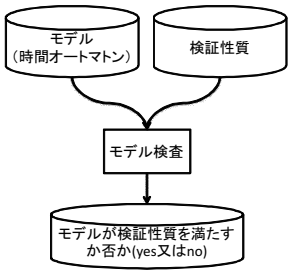

図 8　時間オートマトンのモデル検査問題
時間オートマトンのモデル検査問題とは，時間オートマトンで記述されたモデルと検証性質が入力されたとき，モデルが検証性質を満たすか否かを出力として求める問題である．

断機のモデルとの同期アクションであり，それぞれ遮断機を閉める/開けるという命令を制御器から遮断機に対して送信するイベントを表す．初期ロケーションは `waiting` であり，列車進入を検知すると (`approach`)，`sensing-A` に遷移し，クロック変数 `z` をリセットする．ロケーション `sensing-A` では `approach` から 1 単位時間以上 2 単位時間以内に遮断機に対して閉めろと命令し (`lower`)，`waiting` に戻る．同様に，列車通過を検知すると (`leave`)，`sensing-L` に遷移し，クロック変数 `z` をリセットする．ロケーション `sensing-L` では `leave` から 1 単位時間以上 2 単位時間以内に遮断機に対して開けろと命令し (`raise`)，`waiting` に戻る．

踏み切り制御システム全体のモデルは，列車，遮断機，制御器の動作を表す 3 つの時間オートマトンを並列合成することで得られる．

3. モデル検査

時間オートマトンのモデル検査問題は検証対象システムのモデル（時間オートマトン）と検証性質を入力としてとり，モデルが検証性質を満たすか否か (yes 又は no) を出力として求める問題である (図 8)．

検証性質は，システムの正しさを形式的に定義するものであり，大きく分けて**安全性**と**活性**の 2 種類に分類される．

安全性 (safety) は「システムが決して悪い状態に到達しない」という性質である．悪い状態の例としては，デッドロック (すなわち，システムが何も動作できなくなって停止した状態) や，資源の競合などが挙げられる．踏み切り制御システムの例では,「遮断機が完全に閉まっていない状態で列車が踏み切りに進入することはない」という性質が安全性に該当する．

活性 (liveness) は「システムがいつかは良い状態に到達する」という性質である．安全性のみを保証するだけではシステムが何も役に立つ動作を行わない可能性があるため，活性も検証する必要がある．踏み切り制御システムの例では,「いつかは遮断機が開く（開かずの踏み切りにならない）」という性質が活性に該当する．

検証性質の形式的な記述には，時相論理 (temporal logic) [1] が良く用いられる．ここでは時間オートマトンのモデル検査ツール UPPAAL[5] で検証性質の記述に使用されている時相論理 CTL(Computation Tree Logic)[1] のサブセットによる記述法を紹介する．

CTL は**状態述語** (state predicate) および**時相演算子** (temporal operator) から構成される．状態述語はモデルの任意の状態の集合を指定する述語であり，ロケーション名，クロック変数に関する条件式，および，これらを通常の論理演算子 (論理和，論理積，論理否定など) を用いて任意に組み合わせた式である．ロケーション名は，時間オートマトンが指定したロケーションに居る時に真となる述語である．モデルが複数の時間オートマトンの並列合成である場合は，その要素である個々の時間オートマトンが指定したロケーションに滞在している間は真となる．並列合成された時間オートマトンのロケーション名やクロック変数を指定する場合は，どの時間オートマトンのものかを明確にするために，それが属する時間オートマトンの名前を接頭語として付与することにする．例えば，踏み切り制御システムの例において，

```
TRAIN.crossing and not GATE.close
```

は，列車のモデル `TRAIN` がロケーション `crossing` に滞在していて，かつ，遮断機のモデル `GATE` がロケーション `close` に滞在していない状態で真となる状態述語である．また，

```
GATE.open and GATE.w<=12
```

は，`GATE` がロケーション `open` に滞在していて，かつ，`GATE` のクロック変数 `w` が 12 以下であるとき真となる状態述語である．

時相演算子は，`A[] f` (必ず常に `f` が成り立つ, necessarily always), `A<> f` (必ずいつかは `f` が成り立つ, necessarily eventually), `E[] f` (`f` が常に成り立つ可能性がある, possibly always), `E<> f` (いつかは `f` が成り立つ可能性がある, possibly eventually) の 4 つがある (図 9)．ここで `f` は任意の状態述語である．`A[] f` は，いかなる動作系列においても常に `f` が成り立っているならば真となる．`A<> f` は，いかなる動作系列においてもいつかは `f` が成り立つならば真となる．`E[] f` は，ある可能な動作系列において常に `f` が成り立っているならば真となる．`E<> f` は，ある可能な動作系列においていつかは `f` が成り立つならば真となる．

安全性や活性は状態述語と時相演算子を組み合わせることで記述可能である．例えば，踏み切り制御システムの例において，悪い状態 (危険な状態) の集合は，先の例で示した `TRAIN.crossing and not GATE.close`（列車が踏み切りを通過中なのに遮断機が閉まっていない）という述語

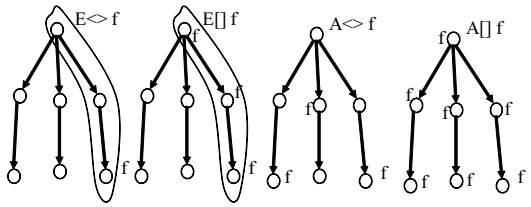

図 9　時相演算子の意味
A[] f は，いかなる動作系列においても常に f が成り立っているならば真となる．A<> f は，いかなる動作系列においてもいつかは f が成り立つならば真となる．E[] f は，ある可能な動作系列において常に f が成り立っているならば真となる．E<> f は，ある可能な動作系列においていつかは f が成り立つならば真となる．

で表現できる．安全性は，悪い状態に決して到達しないことであるから，先の時相演算子 A[] を用いると，踏み切り制御システムの安全性は

```
A[] (not (TRAIN.crossing and not
GATE.close))
```

と記述できる．カッコ内の状態述語は論理演算子 imply(「ならば」) を用いて，

```
A[] (TRAIN.crossing imply GATE.close)
```

と簡単化できる．

一方，踏み切り制御システムの活性は,「必ずいつかは遮断機が開く」と表現できる．この性質は時相演算子 A<>を用いて

```
A<> GATE.open
```

と表現できる．しかしこの性質では遮断機が 1 回だけ開いてそれ以降は閉じたままという（実は良くない）動作を許容してしまう．この問題の解決には,「任意の遮断機が閉まった状態以降に必ずいつかは遮断機が開く」という性質に修正すればよい．このことは，時相演算子 A[] と A<>を入れ子に用いて，

```
A[] (GATE.close imply (A<> GATE.open))
```

と表現できる．なお，ツール UPPAAL では時相演算子を入れ子で用いることができないため，上記の式をそのまま入力することはできないが，同じことを表現する時相演算子-->が用意されており，これを用いて

```
GATE.close --> GATE.open
```

と表現することができる．

以上の検証性質は通常のオートマトンのモデル検査においても利用可能な性質であるが，時間オートマトンのモデル検査においては，時間制約を考慮した性質を検証することもできる．例として，踏み切り制御システムにおいて,「遮断機が閉じてから 12 単位時間以内に開く」という性質を考える．これは時間制約付きの活性である．このことを表現するためには，遮断機が閉じてからの時間を計測するクロック変数が別途必要となる．図 6 の時間オートマトン GATE にはその目的のためにクロック変数 w をあらかじめ導入している．w に関する条件式を用いると，この性質は

```
GATE.close --> (GATE.open and w<=12)
```

と表現することができる．

4．検証例

踏み切り制御システムを例にとり，モデル検査による検証の簡単な実例を紹介する．検証ツールとしては UPPAAL 4.0.8(注5) を用いる．モデルとしては，先に紹介した時間オートマトン TRAIN, GATE, CONTROLLER の並列合成，検証性質としては先に示した安全性 A[] (TRAIN.crossing imply GATE.close) と活性 GATE.close --> GATE.open に加え，「デッドロック無し」(UPPAAL では A[] not deadlock と記述される) を考える．

紙面の制約により UPPAAL の詳細な利用法については省略する．興味のある読者は http://www.uppaal.com/ の Documentation のページに掲載されているチュートリアルを参照されたい．チュートリアルの日本語訳も同ページに掲載されている．

UPPAAL で検証するため，時間オートマトン TRAIN, GATE, CONTROLLER の記述に次のような変更を加えている．まず，UPPAAL では同期アクションをチャネル (channel) と呼んでおり，チャネルによる通信を行うためには，チャネル名を宣言する必要がある．chan 宣言により，各同期アクション approach,leave,lower,raise をチャネルとして宣言した．また，UPPAAL では各同期アクションをチャネル c に対する入力アクション c?および出力アクション c!のいずれかに分類して記述する必要がある．そこで TRAIN のアクション approach,leave をそれぞれ出力アクション approach!,leave!とし，対応する CONTROLLER のアクション approach,leave を入力アクション approach?,leave?とした．同様に GATE のアクション lower,raise をそれぞれ入力アクション lower?,raise?とし，対応する CONTROLLER のアクション lower,raise を出力アクション lower!,raise!とした．さらに，非同期アクションは UPPAAL ではすべて無名の遷移とする必要があるので，非同期アクション in,out,down,up をすべて無名の遷移に変更した．

以上のような修正を加えたモデルと検証性質を UPPAAL に入力し，検証した結果，デッドロック無しで，安全性，活性とも満たされることが確認できた．

(注5) UPPAAL は教育・非商用目的および個人での使用に限り，http://www.uppaal.com/ から無料でダウンロード可能である．

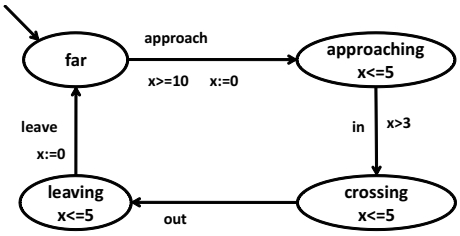

図 10　速い列車のモデル FASTTRAIN

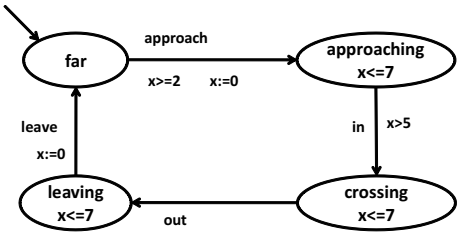

図 11　到着間隔の短い列車のモデル FREQUENTTRAIN

次に，モデルのロケーションと遷移の構造は一切変えず，時間制約のみを変更して検証性質が満たされるか否かの検証を試みてみる．まず，列車が踏み切りを通過する速度が速い場合の検証を行う．速い列車のモデルとしては図 10 を考え，時間オートマトン FASTTRAIN，GATE，CONTROLLER の並列合成に対して検証を試みた．検証の結果，安全性が満たされないことが分かった．同様にして，遮断機のモデル GATE において，遮断機の開閉までにかかる時間をより遅くした場合や，制御器のモデル CONTROLLER において，センサからの入力があってから，遮断機に対して開閉の指令を出すまでの時間を遅くした場合においても，安全性を満たさないことがあることが検証で確認できる．このように，システムの各部の時間制約が相互依存しており，システムの一部が速く (または遅く) なっても全体の安全性に影響するため，時間制約を考慮した検証が必要となる．

時間制約の変更が活性に与える影響を見るために，列車の到着間隔，すなわち，前の列車が通過してから次の列車が進入するまでの間隔を短くしたモデル FREQUENTTRAIN(図 11) を考え，(FREQUENTTRAIN，GATE，CONTROLLER) の組合せで検証を試みた．その結果，安全性は満たされたが活性が満たされない (すなわち，開かずの踏み切りになる) ことが分かった．

このように，一般に実時間システムにおいては，システムの制御構造の設計は同じで，時間制約のみが変化した場合でも，システムが安全性や活性を満たすか否かが変わってしまう可能性がある．このような実時間システムに対しては，時間オートマトンを用いて，システムの具体的な時間パラメータを考慮した検証を行う必要がある．

5．あとがき

本稿では実時間システムの形式的検証技術の一つである時間オートマトンのモデル検査技術に関して概説した．本稿が実時間システムの設計を行う際の一助となれば幸いである．

(2009 年 7 月 15 日受付)

参　考　文　献

1) E. M. Clarke, O. Grumberg and D. A. Peled: “Model Checking”, MIT Press (1999).
2) R. Alur and D. Dill: “A theory of timed automata”, Theoretical Comput. Sci., **126**, pp. 183–235 (1994).
3) M. Sipser: “Introduction to the Theory of Computation”, chapter 1, PWS Publishing Company, 1st edition (1996).
4) J. Magee and J. Kramer: “Concurrency: State Models and Java Programs”, John Wiley & Sons, 2nd edition (2006).
5) K. G. Larsen, P. Pettersson and W. Yi: “UPPAAL in a nutshell”, Int. Journal of Software Tools for Technology Transfer, **1**, 1-2, pp. 134–152 (1997).

［著　者　紹　介］

中田明夫(なかたあきお) 君(非会員)

1992 年大阪大学基礎工学部情報工学科卒業．1997 年同大大学院博士課程修了．博士 (工学)．同年広島市立大学情報科学部助手．2000 年大阪大学大学院基礎工学研究科助手．2002 年同大大学院情報科学研究科助教授．2007 年広島市立大学大学院情報科学研究科教授，現在に至る．実時間システム，並行分散システムの設計手法および形式的検証法に関する研究に従事．情報処理学会，IEEE 各会員．