■7 群（コンピュータ-ソフトウェア）- 1 編（ソフトウェア基礎）

3 章 モデル検査

（執筆者：中島　震）[2009 年 6 月 受領]

■概要■

形式検証は高信頼システム開発の基礎技術であり精力的に研究が進められている．そのなかで，モデル検査法は，自動検証の技術として，産業界からの期待が大きい．モデル検査法の技術概要については，本章 2-4 節モデル検査（総論）にまとめられており，また，理論的な側面に関する良い教科書が出版されている[1]．ここでは，応用の観点からモデル検査法の発展と現状を概観する[2]．

1981 年に基本的な考え方を提案して以来，E.M. Clarke 教授の CPU グループは記号モデル検査ツール SMV を論理回路や分散基盤プロトコルの検証に適用してモデル検査法の効果を実証した．特に，IEEE 標準 Futurebus+仕様の不具合を指摘したことが，形式検証分野の外部にも大きなインパクトを与えた．また，AT&T ベル研（当時）ではオートマトン理論に基づくモデル検査ツールの研究が進められた．SPIN は通信プロトコルや分散アルゴリズムの検証で，COSPAN は論理回路の検証で有効性を示した．検証対象を表現した状態空間を削減する抽象化の方法，ツールの高速化法に関する研究が進んだことによって実用規模のシステムを自動検証することが可能になった．

1990 年代半ばから，SMV や SPIN といった完成度の高いツールが公開されて多くのプロジェクトで利用できるようになり，モデル検査法の適用対象がソフトウェア工学の分野に広がった．状態遷移の考え方を採用した要求仕様，分散システムのソフトウェア・アーキテクチャあるいは UML ステートダイアグラムの検証などにモデル検査法が適用された．また，組込みソフトウェアはリアクティブ性が重要な性質であり状態遷移の考え方と相性が良い．モデル検査法を用いた自動検証への期待が高まり，リアルタイム性を扱える時間オートマトンや連続系を含むハイブリッドオートマトンの研究につながった．

21 世紀に入った頃から，C 言語や Java プログラムの自動検査を目的とするモダン・モデル検査の研究が盛んになってきた．ソースプログラムの自動的な抽象化が技術の核となる．抽象化に基づく系統的な検証法の枠組みとして CEGAR 法が考案され，これを採用することで実用プログラムの自動検査に成功した．また，実用上，検証よりは不具合発見を目的とする有界モデル検査法が有用であることが認識され，現在，いくつかの自動検査ツール製品に組み込まれている．更に，設計仕様からプログラムのテストケースを自動生成するモデル駆動テスティング基本アルゴリズムに応用する研究などが進んでいる．

モデル検査法は，状態遷移の考え方，時相論理，グラフアルゴリズムなどからなる総合的な技術であり，理論研究から工学的なツール開発までを含むバランスの良い研究成果の集大成である．その計算機科学への貢献から，2007 年度 ACM チューリング賞が，E.M. Clarke 博士，E.A. Emerson 博士，J. Sifakis 博士の 3 名に授与された．四半世紀を経た現在，モデル検査法は高信頼ソフトウェア開発で必要とされる自動検査の標準的な基盤技術になっているといってよい．

【本章の構成】

3-1 節では抽象化の基本的な考え方を解説する．3-2 節と 3-3 節はリアルタイム性や離散事

象と連続事象の取扱いなど，組込みシステムの検証で重要な時間オートマトン並びにハイブリッドオートマトンの考え方を説明する．最後に，3-4 節で実用的なモデル検査ツールが採用している高速化技法を紹介する．

■参考文献

1) C. Baier and J.-P. Katoen, “Principles of Model Checking,” MIT Press, 2008.
2) 中島震, “モデル検査法のソフトウェアデザイン検証への応用,” 日本ソフトウェア科学会 コンピュータソフトウェア, vol.23, no.2, pp.72-86, 2006.

■7 群 - 1 編 - 3 章

3-1 抽象化

（執筆者：高橋孝一・関澤俊弦）[2008 年 8 月 受領]

モデル検査における抽象化（abstraction）[1, 2] は，状態数の多いモデルから，より状態数の少ないモデルを構築する技術である．モデル検査で検証を行うためには，次の二つの要件が必要となる．

- 検証したい性質に関して本質的に必要な部分はモデル化されていなければならない．
- モデルの状態数は十分に少なく，モデル検査が現実的な時間内に終了する．

システムの振る舞いを忠実にモデル化した場合，モデル化には検証したい性質と無関係な部分も含んでしまうため，状態数が多くなり状態爆発が起こりやすい．そのため，無関係な部分を削除してモデルを構築する抽象化は重要な鍵となっている．

抽象化は，検証したい性質を考慮して手動で行われることが多かったが，検証対象の広がりとともに，系統的で自動化可能な抽象化技術が望まれている．例えば，プログラムを検証対象とした場合，プログラムから機械的に構築したモデルは，非常に多くの状態をもつため抽象化が必須である．

本節では，一般論を紹介した後，プログラムを対象にした抽象化の代表的な方法であるデータマッピングと述語抽象化を紹介する．

3-1-1 抽象化の一般論

本節では，モデル検査における抽象化に必要な条件を主に述べる．抽象化される前のモデルを具体モデル，抽象化したモデルを抽象モデルと呼ぶ．抽象化の目的はモデル検査を現実的な時間内に終了させることであるから，抽象モデルは小さい方が望ましい．しかし，単に小さいだけでは意味がなく，抽象モデルで検証した結果が具体モデルでも適用できなければならない．これを抽象化の健全性（soundness）という．定式化すると，抽象モデル A が具体モデル C に対し，性質 φ について健全（sound）であるとは，

$$A \models \varphi \implies C \models \varphi \tag{3·1}$$

が成り立つことである．あらゆる性質に対して健全で，なおかつモデル検査に適する万能な抽象モデルは存在しない．したがって，抽象モデルを用いて検証するときには，検証したい性質ごとに抽象化を行う必要があり，系統的な抽象化技法が重要となる．

多くのモデル検査では，モデルとして遷移系（transition system）が用いられている．例えば CTL モデル検査や LTL モデル検査のモデルであるクリプキ構造も遷移系である．したがって，モデル検査の抽象化では遷移系の抽象化が中心的な役割を果たす．

遷移系を抽象化する方法として，複数の同等な状態を一つの状態とみなすことによって，より小さい遷移系を構築する抽象化がある．例として，図 3·1(a) のような交差点の信号機システムを考える．東西/南北の信号機の状態は赤 (r/R)，青 (g/G)，黄 (y/Y) を取るとすると，信号機システムを表す遷移系は図 3·1(b) となる． 今，東西の信号機の状態が赤 (s) かそうでないか (d) のみを考える．すると，東西の信号の青と黄の状態を同一視してよく，更に南

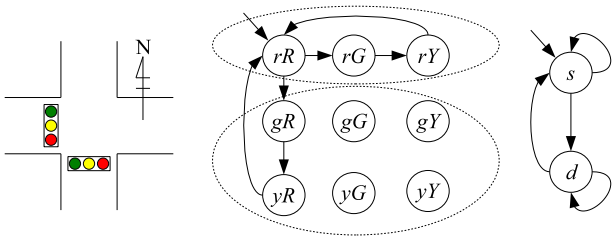


図 3·1　信号機システムの遷移系の抽象化

北の信号機の状態の違いも無視してよい．すなわち，図 3·1(b) の破線で囲まれた状態を同等な状態とみなし，一つの抽象状態とする．抽象モデルを構築するためには，抽象状態間の遷移も決定する必要がある．抽象状態間の遷移は，具体遷移系の模倣（simulation）となるように構成する．模倣は以下のように定義される．遷移系 T, T' に対し，以下を満たす関係 $Q \subseteq T \times T'$ を T から T' への模倣関係（simulation relation）と呼ぶ．

> T において二つの状態 s_1, s_2 の間に遷移があり，$(s_1, s_1') \in Q$ ならば，$(s_2, s_2') \in Q$ となる s_2' が存在し，s_1', s_2' の間に遷移が存在する．かつ，任意の T の初期状態 s_0 と T' の初期状態 s_0' に対して，$(s_0, s_0') \in Q$ となっている．

T から T' への模倣関係が存在するとき，T' は T の模倣であるという．

信号機システムの例では，抽象遷移は図 3·1(c) のようになる．具体遷移系には状態 gR から状態 yR への遷移があり，どちらの状態も抽象状態 d に抽象化されるため，d から d への遷移も存在することになる．

遷移系 T の模倣になっている遷移系 T' は，到達可能性に関して上からの近似（over approximation）になっている．すなわち，T で到達可能な状態は，T' においても対応する状態に到達可能である．したがって，抽象遷移系でモデル検査した結果，到達不可能であることが分かれば，具体遷移系でも到達不可能であることが分かる．

上の議論は，時相論理モデル検査に対して以下のように定式化できる．クリプキ構造 $A = (\overline{S}, \overline{R}, \overline{S_0}, \overline{L})$ がクリプキ構造 $C = (S, R, S_0, L)$ の抽象化になっているとは，抽象化写像 $\beta : S \to \overline{S}$ が存在し，次の条件を満たすときである．

$$(a, a') \in \overline{R} \Longleftarrow \exists c, c' \in S\ [\ a = \beta(c) \wedge a' = \beta(c') \wedge (c, c') \in R] \tag{3·2}$$

$$a \in \overline{S_0} \Longleftarrow \exists c \in S\ [a = \beta(c) \wedge c \in S_0] \tag{3·3}$$

$$L(c) = L(c') \Longleftarrow \beta(c) = \beta(c') \tag{3·4}$$

条件式 (3·2) (3·3) は，$\{(c, \beta(c)) \mid c \in S\}$ が模倣関係となることを要請している．条件式 (3·4) は，同じ原子命題が成り立つ状態どうししか同じ抽象状態とみなしてはいけないことを述べ

ている．

条件式 (3･4) を満たす抽象化写像 β が与えられた場合，$\overline{R}, \overline{S_0}$ をそれぞれ条件式 (3･2)，条件式 (3･3) のように定義することによって抽象クリプキ構造を構築することができる．ただし，このように定義すると，抽象遷移 $\overline{R}$ の計算が，時間がかかるものだったり，決定不能である場合さえある．実際の抽象化では，高速で，なおかつ条件式 (3･2) を満たす抽象遷移 $\overline{R}$ の構築が重要である．

時相論理モデル検査の抽象化に関して，次の定理が成り立つ．

> クリプキ構造 C に対してクリプキ構造 A が抽象化になっているならば，任意の ACTL 式 φ について健全である．また，任意の LTL 式 φ についても健全である．すなわち式 (3･1) が成り立つ．

ACTL 式とは，CTL 式のうち，否定記号は原子論理式の直前にしか現れず，パス限量子 $\mathtt{E}$ をもたないものである．安全性や活性の多くが ACTL 式によって記述可能である．

この定理から，クリプキ構造 C が ACTL で表現できる安全性などの性質 φ を満たすことを確かめるためには，抽象クリプキ構造 A を構築し，$A \models \varphi$ をモデル検査によって確かめればよいことがわかる．

$A \models \varphi$ を検証した結果，A が φ を満たさないことが分かった場合にも，次のような考察が可能である．モデル検査では A が φ を満たさない証拠である反例として A の状態遷移列が得られる．得られた反例を解析することによって，様々なことが分かる場合がある．先の信号機システムの例において，「いずれ必ず東西の信号が赤以外になる」という ACTL 式 $\mathtt{AF}d$ が成り立つか調べることを考える．抽象クリプキ構造をモデル検査すると，反例として $s \to s \to s \to \cdots$ という実行列が得られる．この列を解析すると，具体クリプキ構造にも $rR \to rG \to rY \to rR \to \cdots$ という実行列が存在し，$\mathtt{AF}d$ は具体クリプキ構造でも成り立たないことが分かる．このような反例を実反例（actual counterexample）と呼ぶ．次に，「いつでも，その後いずれ必ず東西の信号が赤になる」という LTL 式 $\mathtt{GF}s$ が成り立つか調べることを考える．この場合も，$s \to d \to d \to d \to \cdots$ という反例が得られる．しかし，今度は具体クリプキ構造にはこの列に対応した実行列が存在しないことが分かる．このような反例を偽反例（spurious counterexample）と呼ぶ．偽反例の出現は，抽象化が適切ではなかったことを示している．更に，偽反例は抽象化の見直しの手がかりを与える場合が多い．

偽反例は，抽象モデルの抽象度が高ければ高いほど，すなわち，より多くの状態を同等とみなすほど，出現しやすくなる．しかし，抽象度が低ければ状態爆発によりモデル検査が終了しにくくなる．モデル検査における抽象化では，いかに適度な抽象度の抽象化を行うかが，非常に重要なポイントである．

適度な抽象化を機械的に求める方法として，反例による抽象化洗練（CounterExample Guided Abstraction Refinement: CEGAR）[3] が提案されている．CEGAR では，まず最初に抽象度の高い健全な抽象モデル A_1 を構築する．もし $A_1 \models \varphi$ ならば具体モデルは φ を満たすことが分かる．一方，反例が得られた場合，それを解析する．もし実反例ならば具体モデルは φ を満たさないことが分かる．偽反例ならば，その偽反例が現れないように抽象度を一つ下げて抽象化を行い，A_2 を構築する．この手続きを繰り返すことによって，次第に適度な抽象度の抽象モデルを構築していく．

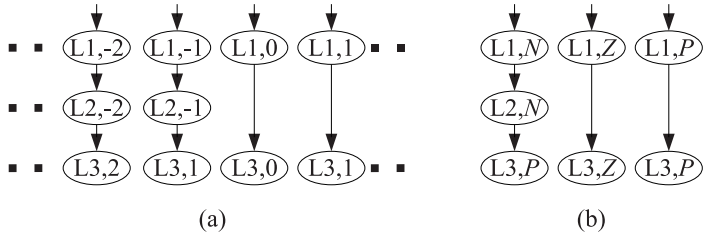


図 3･2　データマッピングによる抽象化

3-1-2 データマッピング

データマッピング（data mapping）[4]は，主に手続き的なプログラムのモデル検査に用いられる抽象化である．プログラムの振る舞いは，プログラムの実行箇所を示すプログラムカウンタの値と，プログラム中の各変数の値の組を状態として，遷移系と考えることができる．したがって，遷移系の状態集合は

$$PC \times D_1 \times D_2 \times \cdots \times D_n$$

となる．ここで，PC はプログラムカウンタの取り得る値の集合，D_i はプログラムに現れる各変数 x_i の取り得る値の集合（データドメイン）である．プログラムでは変数の値は，int（多くの場合 -2^{31} から $2^{31}-1$ までの値）や複雑なデータ構造の値を取り得るため，状態集合が非常に大きくなる．

データマッピングでは，各変数に対し，抽象データドメイン $\overline{D_i}$ と写像 $h_i : D_i \to \overline{D_i}$ を指定し，抽象化写像 β を次のように定める．

$$\beta(pc, d_1, \ldots, d_n) = (pc, h_1(d_1), \ldots, h_n(d_n)) \tag{3･5}$$

この β を用いて前節の一般論で述べた方法により抽象クリプキ構造を構築する．ただし，β は条件式 (3･4) を満たしていなければならない．

例として，int 型の変数 `x` の絶対値を計算する次のプログラムを考える．

```
L1: if (x<0) {
L2:     x = -x; }
L3: assert(x>=0);
```

このプログラムでは $PC = \{\mathtt{L1}, \mathtt{L2}, \mathtt{L3}\}$ である．変数 `x` は int 型の値を取るため，具体遷移系は図 3･2(a) のように巨大になる． そこで，抽象データドメインを値が正かゼロか負かという三つの値の集合 $\{N, Z, P\}$ とし，写像 h は自然なものとする．検証したいことは，`L3` において $\mathtt{x} \geq 0$ が成り立つかどうかなので，原子命題は $\{pc = \mathtt{L3}, \mathtt{x} \geq 0\}$ の二つである．抽象化写像が条件式 (3･4) を満たしていることは明らかである．また，抽象遷移の構築も容易であり，抽象遷移系は図 3･2(b) になる．検証したい性質は ACTL 式では $\mathtt{AG}(pc = \mathtt{L3} \implies \mathtt{x} \geq 0)$ と書ける．抽象遷移系の上でこの式を検証することにより，常に成り立つことが分かる．

頻繁に用いられる抽象データドメインと写像を列挙する．

- 一点に潰す関数．$\overline{D_i} = \{*\}$, $h_i(d) = *$．その変数の値が検証する性質に影響を与えない場合に用いる．検証結果に影響を与えない変数を削除する cone of influence reduction や，その変数の出てくる文を削除するプログラムスライシングに対応する．

- 抽象化しない関数．$\overline{D_i} = D_i$, $h_i(d) = d$．D_i が比較的小さく，すべての値に意味がある場合に用いられる．

- 範囲を限定する関数．$\overline{D_i} = \{small, m_0, r_0, m_1, r_1, \ldots, m_n, large\}$,

$$h(d) = \begin{cases} small & (d < m_0) \\ m_i & (d = m_i) \\ r_i & (m_i < d < m_{i+1}) \\ large & (m_n < d) \end{cases}$$

  D_i が順序集合の場合に，ある値に等しいか，より大きいか，より小さいかによって抽象化する．特にゼロを基準にした N, Z, P の三つの値を用いる場合が多い．

- ある値 m を法とした剰余を計算する関数．$\overline{D_i} = \{0, 1, \ldots, m-1\}$, $h_i(d) = d \mod m$．整数型の変数に対して用いられる．

3-1-3 述語抽象化

述語抽象化（predicate abstraction）[5]は，与えられた有限個の述語に対して，その真偽値の組合せが同じになる状態どうしを同等と見なすことによって抽象化を実現する方法である．

プログラムの状態は $C = PC \times D_1 \times \cdots \times D_n$ の要素 $(pc, d_1, \ldots, d_n)$ で表現される．今，C 上の m 個の述語 $\varphi_1(pc, d_1, \ldots, d_n), \ldots, \varphi_m(pc, d_1, \ldots, d_n)$ が与えられたとする．ただし，これらは原子命題 AP をすべて含んでいるとする：

$$AP \subseteq \{\varphi_1, \ldots, \varphi_m\} \tag{3･6}$$

これら m 個の述語の真偽値が等しい状態を同等とみなすのが述語抽象化である．抽象化写像 β を次のように定める．

$$\beta(pc, d_1, \ldots, d_n) = (pc, \varphi_1(pc, d_1, \ldots, d_n), \ldots, \varphi_m(pc, d_1, \ldots, d_n)) \tag{3･7}$$

すると，値域は $PC \times Bool \times \cdots \times Bool$ である．状態数は $|PC| \cdot 2^m$ となり，データドメインには依存せず述語の個数に依存する．

抽象状態間の遷移 $\overline{R}$ は，条件式 (3･2) を満たすように構築する必要がある．そのために，最弱事前条件（weakest precondition）*という概念を準備する．一般に，実行文 s と条件 ψ に対し，条件 μ がその事前条件 †であるとは，μ が成り立っている状態で実行文 s を実行す

* 最弱事前条件は最弱前条件とも呼ばれる．
† 事前条件は前条件とも呼ばれる．

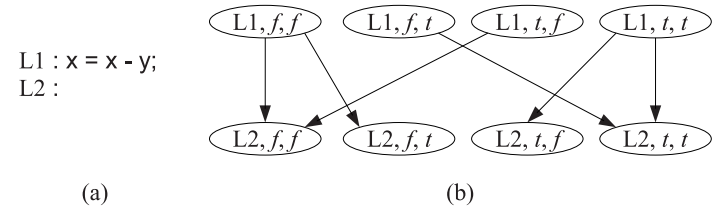


図 3·3　プログラム断片の述語抽象化

ると，実行後は必ず ψ が成り立つことをいう．事前条件のなかで最も弱いものを最弱事前条件といい，$\mathrm{wp}(s,\psi)$ と書く．条件 μ が事前条件ならば，$\mu \implies \mathrm{wp}(s,\psi)$ は恒真である．例えば，実行文 x=x-1 に関する条件 $\mathtt{x} \geq 0$ の最弱事前条件は $\mathtt{x} \geq 1$ である．実行文が代入文 x=e の場合には条件 ψ の最弱事前条件は，ψ のなかの x を e に置き換えることによって得られる．

例を用いて抽象遷移の計算を示す．今，二つの整数型の変数 x と y をもつプログラム P に対し，二つの述語 $\varphi_1 = \mathtt{x}+\mathtt{y} \geq 0$ と $\varphi_2 = \mathtt{x} \geq 0$ を使った述語抽象化を考える． P に，図 3·3(a) に示すコードが現れるとして，この部分に対応する遷移関係を計算する．例として，抽象状態 $a' = (\mathrm{L2}, t, t)$ を考え，$(\mathrm{L1}, b_1, b_2)$（ただし，$b_1, b_2 \in \{t, f\}$）のかたちの各状態 a に対して $(a, a') \in \overline{R}$ となるかどうかを決定したい．抽象状態 a' は，具体遷移系では，L2 の位置で，条件 $\mathtt{x}+\mathtt{y} \geq 0 \wedge \mathtt{x} \geq 0$ を満たす状態に対応する．この条件の実行文 x=x-y に関する最弱事前条件は，x を x-y に置き換えて得られる

$$\mathtt{x} \geq 0 \wedge \mathtt{x}-\mathtt{y} \geq 0 \tag{3·8}$$

である．抽象状態 $a = (\mathrm{L1}, b_1, b_2)$ から a' への遷移があるかどうかは，a が表す条件と条件式 (3·8) との連言が充足可能であることと同値である．例えば，$a_{ff} = (\mathrm{L1}, f, f)$ が表す条件は $\mathtt{x}+\mathtt{y} < 0 \wedge \mathtt{x} < 0$ であるが，これと条件式 (3·8) との連言 $\mathtt{x}+\mathtt{y} < 0 \wedge \mathtt{x} < 0 \wedge \mathtt{x} \geq 0 \wedge \mathtt{x}-\mathtt{y} \geq 0$ は，充足可能でない．したがって，$(a_{ff}, a') \notin \overline{R}$ である．一方，抽象状態 $a_{ft} = (\mathrm{L1}, f, t)$ が表す条件 $\mathtt{x}+\mathtt{y} < 0 \wedge \mathtt{x} \geq 0$ と条件式 (3·8) との連言 $\mathtt{x}+\mathtt{y} < 0 \wedge \mathtt{x} \geq 0 \wedge \mathtt{x} \geq 0 \wedge \mathtt{x}-\mathtt{y} \geq 0$ は，例えば $(\mathtt{x}, \mathtt{y}) = (1, -2)$ によって充足可能である．このようにして，すべての抽象状態の組合せについて充足可能性を調べることによって，遷移関係を決定することができる．このようにして得られた，対応する部分の抽象遷移系を図 3·3(b) に示す．

以上で見たように，遷移関係の決定には充足可能性の判定が必要となる．述語抽象化の実装では，定理証明器などを充足可能性判定に用いる場合が多い．また，定理証明器の呼出し回数は，使用する述語の数に対して指数関数的に増大するため，呼出し回数を抑えるための最適化が必要となる．

なお，データマッピングは，述語抽象化の特別な場合と考えることができる．変数 x_i に対するデータマッピング関数 $h_i : D_i \to \overline{D_i}$ $(i = 1, \ldots, n)$ による抽象化は，$|\overline{D_i}|$ 個の述語 $h_i(d_i) = \overline{d}$ $(\overline{d} \in \overline{D_i})$ による抽象化と考えることができるためである．

述語抽象化は CEGAR を実行するのに適している．なぜならば，述語抽象化の CEGAR では，反例を手がかりとして，述語を順次追加することにより，抽象度を下げることができるからである．また述語を必要に応じて局所的に追加する遅延述語抽象化（lazy abstraction）[6]

も考案されている.

■参考文献

1) D.E. Long, “Model Checking, Abstraction, and Compositional Reasoning,” Ph.D. thesis, Carnegie Mellon University, 1993.
2) 田辺良則, 高井利憲, 高橋孝一, “抽象化を用いた検証ツール,” 日本ソフトウェア科学会 コンピュータソフトウェア, vol.22, no.1, pp.2-44, 2005.
3) E.M. Clarke, “Counterexample-Guided Abstraction Refinement,” TIME, p.7, IEEE Computer Society, 2003.
4) E.M. Clarke, O. Grumberg, and D.E. Long, “Model Checking and Abstraction,” ACM-TOPLAS, vol.16, no.5, pp.1512-1542, 1994.
5) S. Graf and H. Saïdi, “Construction of Abstract State Graphs with PVS,” Computer Aided Verification, Proceedings of 9th International Conference (CAV’97), 1997.
6) T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, “Lazy Abstraction,” ACM SIGPLAN-SIGACT Conference on Principles of Programming Languages (POPL’02), pp.58-70, 2002.

■7 群 - 1 編 - 3 章

3-2 時間オートマトン

（執筆者：岡野浩三）[2009 年 1 月 受領]

3-2-1 時間オートマトンの例

時間オートマトン[1)−4)]の検証ツール UPPAAL[5)]を用いたシステムの記述例を概観する．例として UPPAAL のツールパッケージに同梱されている 4 名の兵士による橋の脱出問題を用いる．この脱出問題は，橋の片一方にいる 4 名の兵士が橋の反対側に全員渡り切る時間を求める問題である．真夜中のため橋を渡るためにトーチが必要とされる．トーチは一つしかなく．また橋の上にはたかだか 2 名の兵士しか同時に載ることはできない．兵士が橋を渡り切るのに要する時間は個々の兵士によって異なる．2 名の兵士が一つのトーチを共用して同時に渡る場合は遅い方の兵士に合わせるものとする．

図 3･4 左は兵士 1 名の振る舞いを表した時間オートマトンである．

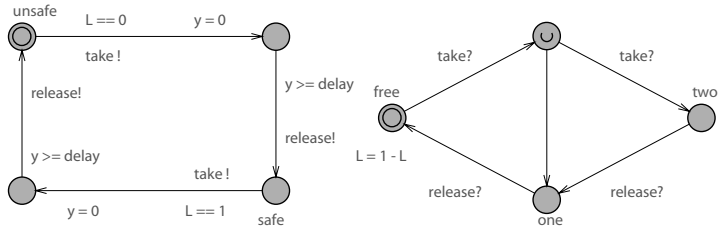


図 3･4　兵士とトーチの振る舞いを表す時間オートマトン

unsafe とラベルされた状態（ロケーション）が初期状態（二重丸で表現されている）である．変数 L はトーチの位置を表す．$L = 0$ のときトーチは unsafe 側にあり，$L = 1$ のときトーチは safe 側にあることを意味する．兵士が unsafe 側に位置し，トーチも unsafe 側にある（$L == 0$）場合，兵士は take という動作を起こし，次ロケーションに移動する．この際，クロック y を 0 にセットしている．遷移は瞬時に起こる．このロケーションにいる状態でクロックの値が delay 以上になったとき（y >=delay），兵士は release という動作を起こし，ロケーション safe に移動する．ロケーション safe から unsafe に移動する際も同様の動作が記述されている．動作 take, release はトーチをもつ，解放するという動作に対応する．このように，状態遷移にクロックの値に関する条件を不等式のかたちで与えることができる．

図 3･4 右はトーチの振る舞いを表した時間オートマトンである．同様に free とラベルづけされた初期ロケーションから動作 take によりロケーション one あるいは，two に移行できることが記述されている．これらのロケーションはそれぞれ，トーチが 1 名あるいは 2 名の兵士にもたれていることにそれぞれ対応する．トーチの中央上のロケーションには U がラベルづけされてる．これは UPPAAL 固有の拡張であり，このロケーションに入った瞬時に次の実行可能な遷移を実行することを要請している．release で free に戻るタイミングで $L = 1 - L$ の式により L の値が更新される．値 0,1 を交互に切替えすることを意味しており，トーチの位置の移動に対応している．

クロック以外にも範囲が限定された離散値をもつ変数や演算を UPPAAL では許している．

これも UPPAAL の時間オートマトンの拡張である．

UPPAAL ではこれらの時間オートマトンをテンプレートとみなし，具体的にパラメータに値を入れることにより，具体的なインスタンスとなる時間オートマトンを定義することができる．兵士の時間オートマトンのパラメータ delay に値 5,10,20,25 を入れることにより，4 名の兵士を表す時間オートマトンを得ることができる．UPPAAL ではクロック変数や，その他の変数はローカル，グローバルいずれの範囲において定義することができる．この例題ではクロック y は各兵士の時間オートマトンに固有のものであるからローカル変数とし，一方，変数 L はグローバルな唯一の変数と定めると題意に適する．

複数の時間オートマトンは同一の動作で同期して遷移することができる．UPPAAL では!,?を用い，プロセス代数的に可能な動作の同期を限定することができる．

以上 4 名の兵士を表す時間オートマトン，一つのトーチを表す時間オートマトンを UPPAAL のツールに与えると，これらの時間オートマトンを図示し，それらのオートマトンの動きを逐次，あるいは自動的にシミュレートし，動きを観察することができる．

一般に，モデル検査では入力としてモデルと検査式を受け取り，モデルの上で検査式が成立するか否かを判定する．

以降では，より標準的な時間オートマトンの定義と諸性質について述べたのち，検査式について簡単に紹介し，モデル検査アルゴリズムとデータ構造を簡単に紹介する．

3-2-2 時間オートマトンの構文

ここでは文献 6, 7) で紹介されている標準的な時間オートマトンを基本にして述べる．

定義 1 (C 上のクロック制約式) クロックの有限集合 C 上のクロック制約式 g の構文を以下のように与える．

$$g ::= x < c \mid x \le c \mid x > c \mid x \ge c \mid\ x - y < c \mid x - y \le c \mid x - y > c \mid x - y \ge c \mid g \wedge g$$

ここで $x, y \in C$, c は整数定数リテラルとする．$c(C)$ を C 上のクロック制約式からなる集合とする．

定義によっては 2 変数の差分不等式をクロック制約とみなさないことがある．

定義 2 (時間オートマトン（timed automaton）) 時間オートマトン $\mathscr{A}$ は 6 項組 (A, L, l_0, C, I, T) で与えられる．ここで

A : アクションの有限集合;
L : ロケーションの有限集合;
$l_0 \in L$:初期ロケーション;
C : クロックの有限集合;
$I : L \to c(C)$: ロケーションからインバリアントへの写像;
T : 遷移集合.
遷移集合 T は $T \subset L \times A \times c(C) \times 2^C \times L$ を満たす．

遷移 $t = (l_1, a, g, r, l_2) \in T$ を $l_1 \xrightarrow{a,g,r} l_2$ と表記する．g をガード，r をリセットクロックと呼ぶ．

定義 3 (クロックの評価関数（clock assigment）) $\nu : C \to \mathbb{R}_{\geq 0}$ なる ν をクロックの評価関数と呼ぶ.
$d \in \mathbb{R}_{\geq 0}$ に対して $(\nu + d)$ を $x \in C$ なるすべての x について $\nu(x) + d$ と定義する.
$R \subseteq C$ とする.

$$\nu[R \mapsto d](x) = \begin{cases} d & if\ x \in R \\ \nu(x) & otherwise \end{cases}$$

と定義する．$g \in c(C)$ に対して $g(\nu)$ を ν の各クロックの評価のもとでの，g の評価（真または偽）と定義する.
また，0^C で $x \in C$ なる各クロック x に対する値を 0 とするクロック評価関数を表すとする.
ν のすべてからなる集合を $N : C \to \mathbb{R}_{\geq 0}$ とする.

3-2-3 時間オートマトンの意味

定義 4 (時間オートマトンの意味) 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ に対して
$\mathscr{A}$ の状態集合を $S = L \times N$ とする.
$\mathscr{A}$ の初期状態は $(l_0, 0^C) \in S$ で与えられる.
状態遷移 $l_1 \xrightarrow{a,g,r} l_2$ $(\in T)$ に対し，次の二つの遷移が定義される.

$$\frac{l_1 \xrightarrow{a,g,r} l_2, g(\nu), I(l_2)(\nu[r \mapsto 0])}{(l_1, \nu) \overset{a}{\Rightarrow} (l_2, \nu[r \mapsto 0])}, \quad \frac{\forall d' \leq d \quad I(l_1)(\nu + d')}{(l_1, \nu) \overset{d}{\Rightarrow} (l_1, \nu + d)}$$

前者をアクション遷移（action transition），後者を時間遷移（delay transition）と呼ぶ.

アクション遷移は以下を意味する．l_1 から l_2 からに遷移 $\xrightarrow{a,g,r}$ があり，現在のクロック評価 ν のもとでガード式を満たし $(g(\nu))$，クロックリセットを施した後 $\nu[r \mapsto 0]$ で l_2 でのインバリアント $I(l_2)$ を満たすならば，対応するアクション遷移が存在する．一方時間遷移は，指定された時間 d の間は同一ロケーションに留まるが，その間インバリアントを満たし続けることを要請している．d として任意の非負実数を取ること，及び，時間遷移は各クロック一様に値を増加させることに注意.

定義 5 (時間オートマトンの意味モデル) 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ に対して上記の意味づけのもとで，初期状態から始まる，（状態数無限の）状態遷移系を定義できる．これを $\mathscr{T}(\mathscr{A})$ で表記する.

インバリアントの概念は文献 4) で導入され，可能なアクション遷移がある場合にロケーションに留まり続けさせずそれを選択させることを，インバリアントを記述することにより，表すことができる.

一般に時間オートマトンを考えるときは，ゼノンのパラドックスなど不自然な動作を回避するためいくつかの性質を仮定する.

定義 6 (時間発散) $\mathscr{T}(\mathscr{A})$ の遷移系列 τ に対して，τ 中のアクション遷移については 0，時間

遷移については d を割り当て，これらの総和として τ の経過時間 $ET(\tau)$ を定義できる．無限遷移系列 τ が時間発散であるとは $ET(\tau) = \infty$ であることをいう．

$Paths_{div}(s)$ を s から始まる遷移系列で時間発散であるものの集合とする．

定義 7 (timelock) $\mathscr{T}(\mathscr{A})$ の状態 s が *timelock* であるとは $Paths_{div}(s) = \emptyset$ であることをいう．

時間オートマトンは一般に，timelock フリーであることが望まれる．

定義 8 (Zenoness) $\mathscr{T}(\mathscr{A})$ の無限遷移系列 τ が *Zeno* であるとは τ が時間発散でなくかつ τ が無限のイベントをもつことをいう．

時間オートマトンは一般に，Zeno であるパスをもたないこと（nonzenoness）が望まれる．
図 3･5 左の時間オートマトンに対する意味モデルの一部を図 3･5 右に与える．

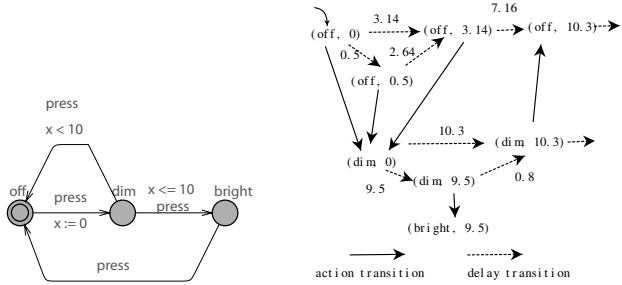


図 3･5　時間オートマトンの例 ライトの時間オートマトンとその意味モデルの一部

3-2-4 時間オートマトンの並列合成

二つの時間オートマトン $\mathscr{A}_1 = (A_1, L_1, l_{1\,0}, C_1, I_1, T_1)$, $\mathscr{A}_2 = (A_2, L_2, l_{2\,0}, C_2, I_2, T_2)$ が与えられたとき，この二つの時間オートマトンの合成動作を表す時間オートマトンとして以下のオートマトンを定義できる．この時間オートマトンを二つのオートマトンの並列合成と呼び，$\mathscr{A}_1 \parallel \mathscr{A}_2$ で表記する．

定義 9 (並列合成（Parallel Composition）) 一般性を失うことなく，二つのオートマトンのクロック集合は互いに素 $C_1 \cap C_2 = \emptyset$ としておく．$\mathscr{A}_1 \parallel \mathscr{A}_2 = (A_1 \cup A_2, L_1 \times L_2, l_{1\,0} \times l_{2\,0}, C_1 \cup C_2, I, T)$ と定義する，ここで $I(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$ であり，遷移 T は以下を満たすように構成される．

- $a \in A_1 \cup A_2$ に対しては，$l_1^1 \xrightarrow{a,g_1,r_1} l_2^1 (\in T_1)$, $l_1^2 \xrightarrow{a,g_2,r_2} l_2^2 (\in T_2)$ なる二つの遷移に対して，$(l_1^1, l_1^2) \xrightarrow{a,g_1 \wedge g_2, r_1 \cup r_2} (l_2^1, l_2^2)$ を構成する．
- $a \in A_1 - A_2$ に対しては，$l_1^1 \xrightarrow{a,g_1,r_1} l_2^1 (\in T_1)$ なる遷移と $l \in L_2$ なる各 l に対して，$(l_1^1, l) \xrightarrow{a,g_1,r_1} (l_2^1, l)$ を構成する．

- $a \in A_2 - A_1$ に対しては，$l_1^2 \overset{a,g_2,r_2}{\longrightarrow} l_2^2 (\in T_2)$ なる遷移と $l \in L_1$ なる各 l に対して，$(l, l_1^2) \overset{a,g_2,r_2}{\longrightarrow} (l, l_2^2)$ を構成する．

3-2-5 時間制約つき時相論理

ここでは，時間に関する表現を CTL（Computation Tree Logic）の時相演算子に取り込んだ MTL（Metric Temporal Logic）[9]と TCTL（Timed Computation Tree Logic）[10]についてごく簡単に述べる．

（1）Metric Temporal Logic

MTL は CTL の各時相演算子 $\mathbb{X}, \mathbb{U}, \mathbb{G}, \mathbb{F}$ にインターバルと呼ばれるパラメータ I を導入し，$\mathbb{X}_I, \mathbb{U}_I, \mathbb{G}_I, \mathbb{F}_I$ という表記を許す．インターバル I は $\geq c_1, > c_1$ などで表記され，それぞれ $[0, c_1], [c_2, \infty]$ を表す．d の法での合同式 $\equiv_d c$ も使われることがある．

MTL 式 f の意味は TPTL 式への変換を通じて与えられる[9]．MTL 式のモデル検査は **EXPSPACE**-complete であることが知られている[9]．

（2）Timed Computation Tree Logic

Timed Computation Tree Logic（TCTL）は以下の構文をもつ．

- $p \in AP$ であれば p は TCL 式;
- α をクロック制約式とすれば α は TCL 式;
- f が式であれば $p, \neg f, f \vee g, f \wedge g$ は TCL 式;
- f, g が TCL 式であれば $\mathbb{A}(f \mathbb{U}^J g), \mathbb{E}(f \mathbb{U}^J g)$ は TCL 式.

ここで J はインターバル制約であり，$[n, m], [n, m), (n, m], (n, m)$ のかたちをもつ．ここで n, m は $n \leq m$ を満たす有理数であり，m の代わりに ∞ も許す．

TCTL の意味は，もともと時間グラフ（timed graph）という時間オートマトンに近い構造で与えられてた．ここでは時間オートマトンの意味モデル $\mathscr{T}(\mathscr{A})$ を介して意味を与える．$s = (l, v)$ を $\mathscr{T}(\mathscr{A})$ の状態とする．

- $s \models$true;
- $s \models p$ iff $p \in L(l)$;
- $s \models \alpha$ iff $v \models \alpha$;
- $s \models \neg f$ iff $s \not\models f$;
- $s \models f \wedge g$ iff $s \models f$ かつ $s \models g$;
- $s \models \mathbb{A}\phi$ iff すべての $\pi \in Paths_{div}(s)$ について $\pi \models \phi$;
- $s \models \mathbb{E}\phi$ iff ある $\pi \in Paths_{div}(s)$ について $\pi \models \phi$;

時間発散パス $\pi \in s_0 \overset{d_0}{\Rightarrow} s_1 \overset{d_1}{\Rightarrow} \cdots$ について

- $\pi \models f\mathbb{U}^J g$ iff ある $i \geq 0$ について $d \in [0, d_i]$ が存在して $(s_i{+}d \models g$ かつ $\sum_{k=0}^{i-1} d_k{+}d \in J)$, かつ，$j \leq i$ なる任意の j と任意の $d' \in [0, d_j]$ について $(s_j + d' \models f \vee g$ かつ $\sum_{k=0}^{j-1} d_k + d' \leq \sum_{k=0}^{i-1} d_k + d)$

TCTL のモデル検査に対する **PSPACE** なアルゴリズムが知られている．

3-2-6 モデル検査のためのデータ表現

この節の内容は主に，文献 5) による．時間領域をどのように有限かつ効率的に計算機のメモリ空間に保持するかがここでの主題である．

(1) Clock Region

与えられた一つの時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ に対してクロックリージョン $CR(\mathscr{A})$ を一つ定めることができる[2]．$\mathscr{A}$ のクロック集合 C について $x \in C$ なる各クロック x に対し，c_x を g, I において x の比較式に使われている最大の定数とする．$k(x)$ をクロック x からある非負整数への関数とする．

$a \in \mathbb{R}_{\geq 0}$ に対して，$fr(a)$ を a の小数部，$\lfloor a \rfloor$ を a の整数部とする．定義より $a = \lfloor a \rfloor + fr(a)$ が成立する．クロックの評価関数間に以下の等価関係 $\simeq_k$ を導入する．

定義 10 $\nu \simeq_k \nu'$ *iff*

- すべての $x \in C$ に対して，$\nu(x) \geq k(x)$ かつ $\nu'(x) \geq k(x)$ が成り立つか，または $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ が成り立つ*;*
- $\nu(x) \leq k(x)$ かつ $\nu(y) \leq k(y)$ を満たすすべての $x, y \in C$ に対して，$fr(\nu(x)) \leq fr(\nu(y))$ *iff* $fr(\nu'(x)) \leq fr(\nu'(y))$*;*
- $\nu(x) \leq k(x)$ を満たすすべての $x \in C$ に対して，$fr(\nu(x)) = 0$ *iff* $fr(\nu'(x)) = 0$*;*

$k(x)$ として c_x をとるとき $\nu \simeq \nu'$ とする．

定義 11 (クロックリージョン（clock region）) 与えられた一つの時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ に対して上述の等価関係 $\simeq$ による $|C|$-次元空間の分割をクロックリージョン（あるいは単にリージョン）と呼ぶ．

図 3･6 のクロックリージョンは 12 個の点，30 個の線分，18 個の面からなる．

$CR(\mathscr{A})$ の要素（である一つのリージョン）を $\simeq$ による同値類分割の意味で $[u]$ で表記する．

クロックリージョンの数は次の値で抑えられる．$|C|! \cdot 2^{|C|} \cdot \prod_{x \in C}(2c_x + 2)$.

(2) リージョングラフ（Region Graph）

与えられた時間オートマトンからリージョンを構成し，そのリージョンを用いて時間オートマトンと同じ動きをもつリージョングラフ（リージョンオートマトン）を構成できる*.

* 時間遷移を ϵ 遷移として構成する定義[6]もある．

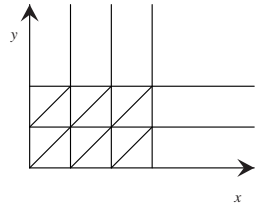


図 3·6　$c_x = 3, c_y = 2$ の場合のクロックリージョンの例

定義 12 (リージョングラフ) 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ のリージョングラフ $\mathscr{R}(\mathscr{A}) = (A, L_r, l_{r\,0}, T_r)$ は以下のように与えられる．
$L_r \subset L \times CR(\mathscr{A})$;
$l_{r\,0} = (l_0, \vec{0})$;
$T_r \subset L_r \times L_r$.
L_r の要素は $(l, [u])$ で表現される．
T_r の要素 t は以下のように定義される．
$(l, [u]) \longrightarrow (l', [u'])$ iff $(l, \omega) \xrightarrow{a} (l', \omega') \in \mathscr{T}(\mathscr{A})$ が存在し，$\omega \in [u]$ かつ $\omega' \in [u']$ を満たす．
$(l, [u]) \longrightarrow (l, [u'])$ iff $(l, \omega) \xrightarrow{d} (l, \omega') \in \mathscr{T}(\mathscr{A})$ が存在し，$\omega \in [u]$ かつ $\omega' \in [u']$ を満たす．

リージョングラフは有限である．リージョンを用いて，到達性解析，時間なし言語の包含問題，時間付き双模倣関係の問題が解決できる．

(3) ゾーン（Zone）

実際のモデル検査においてはリージョングラフを直接扱わず，もう少し効率の良いデータ表現を用いるのが普通である．これ以降，クロック制約式の定数を整数に限定する．

本質的にクロックの値が非負であることに注意するとクロック制約式は以下のかたちで表すことができる．

$x_0 = 0 \bigwedge_{0 \le i \ne j \le n} x_i - x_j \sim c_{i,j}$

ここで $\sim$ は $\le$ または $<$ であり，$c_{i,j}$ は非負整数である．また x_0 は 1 クロックの制約式も 2 クロックの差分不等式として表せるよう 0 定数を表すために便宜上導入されている．

このかたちの表現式をゾーン（あるいはクロックゾーン）（Clock Zone）[4] と呼ぶ．

更に，ゾーンの代わりに，クロック数が n のときに $(n+1) \times (n+1)$ の行列を考え，この行列の要素として $c_{i,j}, \sim$ の 2 項組を与えると DBM （Difference Bound Matrix）となる．

定義 13 (ゾーングラフ（Zone Graph）) 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ のゾーングラフは以下のように与えられる．

- (l_0, D^0) は初期ノードである．ここで D^0 はすべてのクロック $x(\in C)$ に対して，$x = 0$ の制約を表すゾーン;
- 遷移 $(l, D) \rightsquigarrow (l, D^{\Uparrow} \wedge I(l))$ が存在する．ここで $D^{\Uparrow} = \{u + d \mid u \in D, d \in \mathbb{R}_{\ge 0}\}$;
- 遷移 $(l, D) \rightsquigarrow (l', r(D \wedge g) \wedge I(l'))$ が $l_1 \xrightarrow{a,g,r} l'$ に対して存在する．

直後に述べるように D をゾーンとするとき，$D^{\Uparrow}$, $r(D)$ もゾーンである．図 3･7 に図 3･5 に対応するリージョングラフとゾーングラフの例を与える．

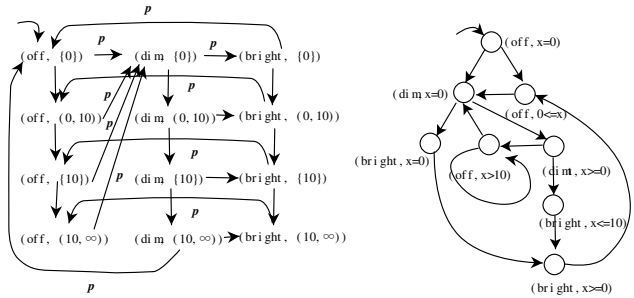


図 3･7　図 3･5 に対応するリージョングラフとゾーングラフ

残念ながら，ゾーングラフは有限にならない可能性がある．

ゾーングラフの無限性を解決するために次の二つの手法が知られている．クロック制約式として $x-y \sim c$ のような 2 変数の式が現れないもの（diagonal-free）については k-Normalization, 2 変数の式が現れるものについては k,g-Normalization である．いずれもクロックの値が k を超えたときに，その値の違いを無視することによりゾーングラフのノードを有限に抑えている．一般に k として時間オートマトンの制約式中に現れる最大の定数を取る．

これらの詳細は文献[5]を参照のこと．

前述のように一般にゾーンは DBM として表現される．次に DBM の定義とその上の操作について簡単に述べる．

3-2-7 DBM（Difference Bound Matrix）

任意のゾーンは常に 0 を表す擬似クロック変数 $x_0 = 0$ を導入し，以下のように表現できた．$\bigwedge_{0\le i\ne j\le n} x_i - x_j \sim c_{i,j}$，ここで ~ は ≤ または < であり，$c_{i,j}$ は非負整数である．

ゾーン D を表現する DBM は $(n+1)\times(n+1)$ の行列として表現される．この行列の i 行 j 列を D_{ij} で表す．D_{ij} は以下のステップで与えることができる．

1. $x_i - x_j \sim c_{i,j} \in D$ に対して，$D_{ij} = (c_{i,j}, \sim)$;
2. 各 $x_i - x_j$ 式について，この式が D に現れていなければ $D_{ij} = \infty$;
3. 各 i について $D_{0i} = (0, \le)$. これは $x_0 - x_i \le 0$ を意味する;
4. 各 i について $D_{ii} = (0, \le)$. これは $x_i - x_i \le 0$ を意味する．

DBM の例として制約 (3･9) を満たすゾーンを考える．

$$x - 0 < 20 \wedge y - 0 \le 20 \wedge x - y \le -10 \wedge y - x \le 10 \wedge 0 - z < 5. \qquad (3\cdot9)$$

制約 (3･9) を表現した DBM を (3･10) で与える．

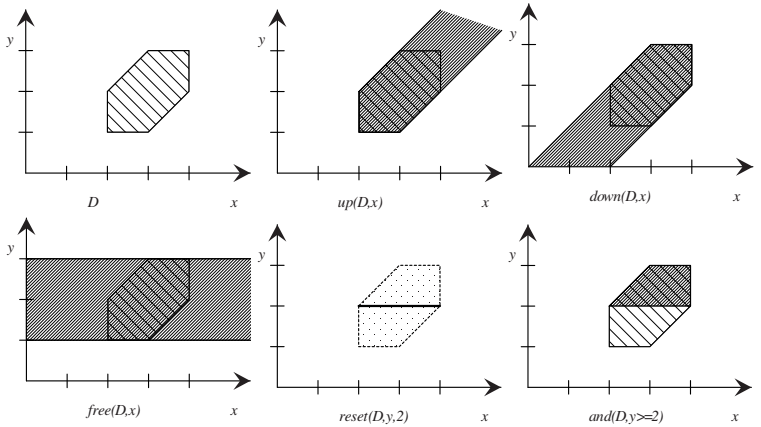


図 3･8　DBM **の基本演算の一部**

$$D = \begin{pmatrix} (0,\leq) & (0,\leq) & (0,\leq) & (5,<) \\ (20,<) & (0,\leq) & (-10,\leq) & \infty \\ (20,\leq) & (10,\leq) & (0,\leq) & \infty \\ \infty & \infty & \infty & (0,\leq) \end{pmatrix} \tag{3･10}$$

(1) DBM の演算

DBM 上でモデル検査に必要ないくつかの演算が定義できる．

(a) 標準形（Canonical Form）

与えられた DBM に対してそのグラフ表現 $G = (V, E)$ を考えることができる．V は各変数，E は D_{ij} が $(n, \sim)$ であるときに x_i から x_j への有向辺として与えられ D_{ij} をそのラベルにもつ．Floyd-Warshall の最短経路問題のアルゴリズムを適用することにより，標準形のグラフ表現を得ることができる．

(b) 最簡化（Minimal Constraint System）

$x - y < 3, y - z < 4, x - z < 7$ の制約式において最後の制約式は最初の二つから推移律で導出可能な意味で冗長である．重み付きグラフに変換することにより冗長な制約式を取り除くアルゴリズムが存在する．

(c) DBM の基本演算

DBM の基本演算は二つのカテゴリに区分できる．性質判定用の演算と変形である．後者は主に時間遷移に伴う時間領域の変遷を演算する．

性質判定には **Consistent**，**Relation**，**Satisfied** などがあり，DBM のデータ構造を利用した効率のよい演算が存在する．

図 3･8 に変形操作を含む一部の基本演算の適用の様子を示す．

DBM の Normalization 操作も同様に DBM 上の記号操作として行える．

3-2-8 モデル検査の実際

UPPAAL では TCTL のサブセットである $\mathbb{AG}f, \mathbb{AF}f, \mathbb{EG}f, \mathbb{EF}f, \mathbb{AG}(f \rightarrow \mathbb{AF}f)$ のみ扱う．ここで f は時相演算子を含まない論理式である．

DBM による解析では基本的に到達解析が可能であるので，$\mathbb{AG}f, \mathbb{EF}f, \mathbb{AG}(f \rightarrow \mathbb{AF}g)$ のような安全性（safety property）は検証可能となる．一方 $\mathbb{AF}f, \mathbb{EG}f$ のような時間束縛のない活性（unbound livness property）は本質的にループの判定となるが，計算コストは大きくなる．また，これらの性質は一般的な検証では用いられることが少ない．

一方，5 単位時間で望ましい状況にたどり着けるといった時間束縛のある活性（bounded livness property）はテストオートマトンを併用することで検証可能となる．

UPPAAL ではほかに次の拡張を行っている．上限のある整数変数，ブールなどの変数の導入; 各変数のスコープ設定や共有; 特別な意味（Urgent，Committed）をもつロケーションの導入．

イベント遷移は，基本的に同一のイベントがラベルづけされている遷移のなかですべてのガード条件をみたす遷移可能な遷移が選択される．ただし，UPPAAL では更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．

図 3･4 の時間オートマトンに関する性質検証を以下のように行うことができる．ツールに対して，$\mathbb{FF}$ Viking1.safe という式でモデル検査を行うと真であるという結果が即返ってくる．この式は「兵士 1 がいつかはロケーション safe に到達する．」ということを意味している．

更に $\mathbb{FF}$ Viking1.safe ∧ Viking2.safe ∧ Viking3.safe ∧ Viking4.safe ∧ time ≤60 という式を問い合わせることもできる．「(いつかは）すべての兵士が safe な状態でかつ経過時間が 60 以内という状況に達することは可能か」ということを意味している．

時間オートマトンの検証ツールは UPPAAL のほかに Kronos が知られている．

DBM 以外のデータ構造として，CDD（clock-difference diagram）, CRD（clock-restriction diagram）, DDD（difference decision diagram）など複数のデータ構造が提案されている[8]．UPPAAL 4.0 では CDD が用いられている．

■参考文献

1) R. Alur and D.L. Dill, “Automata For Modeling Real-Time Systems,” LNCS, vol.443, pp.322-335, 1990.
2) R. Alur and D. L. Dill, “A theory of timed automata,” J. Theoretical Comput. Sci., vol.126, no.2, pp.183-235, 1994.
3) K. Cerans, “Decidability of Bisimulation Equivalences for Parallel Timer Processes,” LNCS, vol.663, pp.302-315, 1992.
4) T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, “Symbolic Model Checking for Real-time Systems,” Inform. Comput., vol.111, no.2, pp.193-244, 1994.
5) J. Bengtsson, and W. Yi, “Timed Automata: Semantics, Algorithms and Tools,” LNCS, vol.3098, pp.87-124, 2004.
6) E.M. Clarke, O. Grumberg, and D.A. Peled, “Model Checking,” MIT Press, 2000.
7) C. Baier and J.-P. Katoen, “Principles of Model Checking,” MIT Press, 2008.
8) F. Wang, “Formal verification of timed systems: a survey and perspective,” Proc. of the IEEE, vol.92, no.8, pp.1283-1305, 2004.

9) R. Alur and D.L. Dill, “Real-time logics: complexity and expressiveness,” Inform. Comput., Vol.104, pp.35-77, 1993.
10) R. Alur, C. Courcoubetis, and D.L. Dill, “Model-checking in dense real-time,” Inform. Comput., vol.104, pp.2-34, 1993.
11) UPPAAL, http://www.uppaal.com/

■7 群 - 1 編 - 3 章

3-3 ハイブリッドオートマトン

（執筆者：山根　智）[2008 年 6 月 受領]

3-3-1 ハイブリッドシステム

ハイブリッドシステムは組込みシステムの重要なモデルである．組込みシステムは，並行動作するソフトウェア及び物理的に連続的な環境と相互作用するソフトウェアの集まりである．制御理論は，物理的なプロセスの最適なパフォーマンスを保証するために，制御法則を設計することに着目しているが，並行性と通信には着目していない．一方，コンピュータ科学は，環境の物理的な性質を抽象化しているので，組込まれた機器の最適なパフォーマンスを保証できない．組込みシステムはセンサ，アクチュエータや制御ソフトウェアなどからなり，離散動作と連続動作が混在するリアクティブシステムであるハイブリッドシステム（hybrid system）[1] とみなすのがベストである．ハイブリッドシステムは制御理論とコンピュータ科学のアプローチを統合化しており，組込みシステムのモデル化と仕様記述には自然なものである．

ハイブリッドシステムの様々なセマンティックモデルや形式的検証手法が研究されている．世界で最初に提案されたハイブリッドシステムのセマンティックモデルは 1992 年の O. Maler らのフェーズ遷移システム[2] であり，その演繹的検証の公理系も開発されている．また，1993 年に，R. Alur らはハイブリッドオートマトン（hybrid automaton）[3] とそのモデル検査手法を開発して，更に，線形ハイブリッドオートマトンの記号モデル検査ツール HyTech[4, 5] を開発している．それと同時に，T.A. Henzinger らはハイブリッド時相論理[6] とその演繹的検証の公理系を開発している．驚くべきことに，これらのすべてのモデルはハイブリッドシステムを部分的連続関数（piecewise continuous function）としてモデル化している．すなわち，連続区間はフェーズに対応しており，システムの制御が固定されており，システムの変数は制御法則に従って連続的に変化する．一方，不連続点はシステムの状態の離散的変化に対応しており，制御法則の変化とともにシステムの制御の変化を意味している．結局，このモデルでは，離散的変化は経過時間ゼロの状態遷移を意味して，時間の経過はフェーズにおいて起きることを意味する．このモデルはリアルタイムシステムのモデル[7, 8] の自然な拡張であり，ハイブリッドシステムの形式的検証はリアルタイムシステムの形式的検証の自然な拡張として発展している．

これらのモデルのなかで組込みシステムの仕様記述言語として極めて重要なクラスは R. Alur らのハイブリッドオートマトン[3] であり，効率的なモデル検査ツールが構築されている，線形ハイブリッドオートマトン[3]，検証問題が計算可能な矩形（rectangular）ハイブリッドオートマトン[9] やプリエンプティブスケジューリングを仕様記述できるストップウォッチオートマトン[4] なども重要である．

3-3-2 ハイブリッドオートマトンの定義

実用上からは，R. Alur らのハイブリッドオートマトンとその自動検証[3] は極めて重要であるので，以下に詳しく説明する．

定義 1 (ハイブリッドオートマトン) ハイブリッドオートマトンは，
$\mathbf{A} = (X, V, flow, inv, init, E, jump, \Sigma, syn)$ の九つ組で定義される．ここで，

1. $X = \{x_1, x_2, \ldots, x_n\}$ は実数値の変数である．

2. V は制御モードの有限集合である．なお，制御モードは通常のオートマトンの有限状態に対応する．

3. ラベリング関数 $flow$ は各制御モード $v \in V$ にフロー条件を割り付ける．フロー条件 $flow(v)$ は $X \cup \dot{X}$ のなかの変数に関する述語であり，連続動作の定義を表す．例えば，$flow(v) = (\dot{x} = -x + 5)$ である．ここで，$\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ であり，$\dot{x}_i = dx_i/dt (1 \leq i \leq n)$ は $\dot{x}_i$ の時間に関する一次導関数である．

4. ラベリング関数 inv は各制御モード $v \in V$ に不変条件を割り付ける．不変条件 $inv(v)$ は X のなかの変数の述語である．

5. ラベリング関数 $init$ は各制御モード $v \in V$ に初期条件を割り付ける．初期条件 $init(v)$ は X のなかの変数の述語である．

6. E は制御スイッチの有限多重集合であり，離散動作を表す．各制御スイッチ (v, v') はソースのモード $v \in V$ とターゲットのモード $v' \in V$ との間の有向な枝である．

7. ラベリング関数 $jump$ は各制御スイッチ $e \in E$ にジャンプ条件を割り付ける．ジャンプ条件 $jump(e)$ は $X \cup X'$ の中の変数の述語である．ここで，$X' = \{x_1', \ldots, x_n'\}$ である．なお，x_i は制御スイッチ前の変数 x_i の値を表し，x_i' は制御スイッチ後の変数 x_i の値を表す．ジャンプ条件 $jump(e)$ は制御スイッチ前後の変数の値を関係づける．

8. Σ はイベントの有限集合である．

9. ラベリング関数 syn は各制御スイッチ $e \in E$ にイベントを割り付ける．

ハイブリッドオートマトン $\mathbf{A}$ の状態 $(v, \mathbf{a})$ は制御モード $v \in V$ とベクタ $\mathbf{a} = (a_1, \ldots, a_n)$ の順序対である．ここで，a_i は各変数 x_i の値を表す．もし各変数 x_i が値 a_i に置換されたとき述語 $inv(v)$ が true ならば，$\mathbf{A}$ の状態 $(v, \mathbf{a})$ は **admissible** である．もし各変数 x_i が値 a_i に置換されたとき述語 $init(v)$ が true ならば，$\mathbf{A}$ の状態 $(v, \mathbf{a})$ は初期状態である．二つの **admissible** な状態を $q = (v, \mathbf{a})$ と $q' = (v', \mathbf{a}')$ とする．もし各変数 x_i が値 a_i に置換されて各変数 x_i' が値 a_i' に置換されたとき述語 $jump(e)$ が true であるような $e \in E$ が存在するならば，順序対 (q, q') は**ジャンプ**である．もし $v = v'$ で $\delta \in \mathbf{R}_{\geq 0}$（フローの持続時間）と微分可能な関数 $\rho : [0, \delta] \to \mathbf{R}^n$（フローのカーブ）が以下の三つの条件を満たすならば，順序対 (q, q') は**フロー**である:

1. $\rho(0) = \mathbf{a}$ かつ $\rho(\delta) = \mathbf{a}'$.

2. すべての $t \in (0, \delta)$ に対して，状態 $(v, \rho(t))$ は **admissible** である．

3. $\dot{\rho} : [0, \delta] \to \mathbf{R}^n$ は ρ の微分係数である．すべての $t \in (0, \delta)$ に対して，各 x_i はベクタ $\rho(t)$ の i 座標で置換され，各 $\dot{x}_i$ がベクタ $\dot{\rho}(t)$ の i 座標で置換されたときに，述語

$flow(v)$ は true になる．

ハイブリッドオートマトン **A** の軌跡は **admissible** な状態 q_j の有限列 $q_0, q_1, \ldots, q_k$ であり，q_0 は **A** の初期状態であり，連続した状態の対 (q_j, q_{j+1}) は **A** の**ジャンプ**か**フロー**である．**A** のある状態が **A** の軌跡の最後の状態ならば，その状態は**到達可能（reachable）**である．

例題 2 (ハイブリッドオートマトンの例) 簡単なハイブリッドオートマトンの例を示す[10]．ハイブリッドオートマトンで温度自動調節器を仕様記述しており，二つの制御モード *on* と off からなる．制御モード *on* には，フロー条件 $\dot{x} = -x + 5$，不変条件 $1 \le x \le 3$ が割り当てられている．また，ジャンプ条件は $x = 3$ 及び $x = 1$ である．

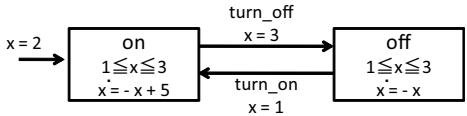


図 3·9　温度自動調節器のハイブリッドオートマトン

安全性〔本編 2 章 2-4-1(a) 参照〕は望ましくない事象がシステムの動作の間に起きないということを意味する．すなわち，望ましくない事象が起こっていない状態を安全な状態であると定義し，すべての到達可能な状態が安全であるとき，そのシステムは安全性を満たすと定義する．したがって，安全性検証には一般的に到達可能な状態の集合を計算することが必要となる．ハイブリッドオートマトンの検証においては，状態の表明（state assertion）を用いて安全性を仕様記述する．ハイブリッドオートマトン **A** の状態の表明 φ は各制御モード $v \in V$ に（X の変数の）述語 $\varphi(v)$ を割り付ける関数である．各変数 x_i が値 a_i に置換したときに述語 $\varphi(v)$ が true になるならば，状態 $(v, \mathbf{a})$ は φ-**状態**（φ-state）であるという．**A** の不変条件 *inv* は状態の表明の例であり，*inv*-状態かつそれらのみが **admissible** である．同様に初期条件 *init* も状態の表明の例であり，*init*-状態かつそれらのみが初期状態である．望ましくない事象が起こっていることを表す表明を $unsafe$ とすると，すべての到達可能な状態がいずれも $unsafe$-状態でなければ，**A** は安全性を満たす．

3-3-3 線形ハイブリッドオートマトンの形式的検証

(1) 到達状態の計算方法

ハイブリッドオートマトン **A** が（状態の表明 $unsafe$ により記述された）安全性を満たすかどうかチェックするために，もう一つの状態の表明 *reach* を計算する．なお，*reach* は到達可能な状態において true である．このとき，状態の表明 *reach* と $unsafe$ の両方が true である状態があるかどうかをチェックする．もしそういう状態があれば安全性を満たさないことになり，もしなければ安全性を満たすことになる．

以下に，状態の表明 *reach* を計算する方法を定義する．状態の表明 φ に対して，φ-状態のジャンプとフローの後継において true となる状態の表明 $Post(\varphi)$ を導入する．なお，ジャンプの後継は次の制御モードへの離散動作を与えて，フローの後継は時間経過によるフローの変化（連続動作）を与える．つまり，この *Post* は後継の状態の表明を計算する演算子で

ある．$(q, q\prime)$ が **A** のジャンプまたはフローであるような φ-状態 q が存在するときに限り，$Post(\varphi)$ は状態 $q\prime$ に対して true である．初期状態の状態の表明 φ_0 に対して，その後継の状態の表明は $\varphi_1 = Post(\varphi_0)$ と表現できて，到達可能な状態の表明は，ある自然数 k に対して，$\varphi_k = Post^k(\varphi_0)$ と表現できる．しかし，この $Post(\varphi)$ の計算には，以下の二つの問題がある．

1. 状態の表明 φ に対して，$Post(\varphi)$ を計算できる必要があるが，一般のハイブリッドオートマトンでは効率的に $Post(\varphi)$ を計算できない．しかし，制限された線形ハイブリッドオートマトンに対しては効率的に計算できる．

2. $Post$ の繰り返し計算は有限回数で収束することが望まれるが，一般のハイブリッドオートマトンでは収束しない．収束が保証されている実用的な部分クラスとしては時間オートマトンしか知られていない．

(2) 線形ハイブリッドオートマトン

一般のハイブリッドオートマトンは表現能力が非常に高い一方，自動検証が困難である．線形ハイブリッドオートマトンと呼ばれる部分クラスに対して効率的な自動検証が可能である．

まず，用語を定義する．原子線形述語は有理数の係数をもつ変数の線形結合と有理数との不等式（例えば，$3x_1 - x_2 + 7x_5 \leq \frac{3}{4}$）である．凸線形述語は原子線形述語の論理積である．線形述語は凸線形述語の論理和である．ハイブリッドオートマトン **A** が以下の二つの条件を満たすならば，**A** は線形ハイブリッドオートマトンである．

1. すべての制御モード $v \in V$ に対して，フロー条件 $flow(v)$，不変条件 $inv(v)$ と初期条件 $init(v)$ は凸線形述語である．すべての制御スイッチ $e \in E$ に対して，ジャンプ条件は $jump(e)$ は凸線形述語である．

2. すべての制御モード $v \in V$ に対して，フロー条件 $flow(v)$ は $\dot{X}$ の述語であり，X は含まれない．これは非常に重要な制限であり，線形ハイブリッドオートマトンでは $\dot{x} = x$ などのフロー条件を許さない．いくつかの重要なオートマトンが特定のフロー条件をもつ線形ハイブリッドオートマトンとして定義できる．例えば，フロー条件 $\dot{x} = 1$ である部分クラスは時間オートマトン，$\dot{x} = 0$ または $\dot{x} = 1$ はストップウオッチオートマトン，ある定数 ϵ に対して $\dot{x} \in [1 - \epsilon, 1 + \epsilon]$ は矩形ハイブリッドオートマトンである．

(3) 非線形ハイブリッドオートマトンの線形化

図 3･9 のハイブリッドオートマトンは $\dot{x} = -x$ のフロー条件があるので，線形ハイブリッドオートマトンではない．非線形ハイブリッドオートマトンを直接効率的に自動検証するのは困難であるので，非線形ハイブリッドオートマトンから線形ハイブリッドオートマトンへ変換する二つの手法，クロック変換（clock translation）[11] と線形フェーズポートレート近似（linear phase-portrait approximation）[12] を紹介する．現実のハイブリッドシステムの多くは線形ハイブリッドオートマトンによりモデル化できないことが知られているので，非線形ハイブリッドオートマトンを線形ハイブリッドオートマトンに変換して自動検証して，元の非線形ハイブリッドオートマトンの正当性を保証する技術は非常に重要である．

(a) クロック変換（clock translation）[11]

クロック変換のアイディアは，変数の値は過去の値とそれからの経過時間で決定されると

いうことである．ハイブリッドオートマトン **A** が以下の二つの条件を満たすならば，変数 x_i はクロック変換可能である．

1. 各フロー条件 $flow(v)$ において，x_i と $\dot{x}_i$ のすべての出現は形式 $\dot{x}_i = g_i^v(x_i)$ の論理積である．ここで，$g_i^v(x_i) : \mathbf{R} \to \mathbf{R}$ は積分可能関数である．各不変条件，各初期条件と各ジャンプ条件において，x_i と $\dot{x}_i$ のすべての出現は形式 $x_i\prime = x_i$ または $x_i \sim c$ または $x_i\prime \sim c$ である．ここで，$\sim \in \{<, \leq, >, \geq\}$ であり，c は有理数の定数である．

2. すべての制御モード $v \in V$ に対して，初期条件 $init(v)$ ならば $x_i = c$ である．すべての制御スイッチ $(v, v\prime)$ に対して，(1)$g_i^v = g_i^{v\prime}$ かつ $jump(v, v\prime)$ ならば $x_i\prime = x_i$，または，(2)$jump(v, v\prime)$ ならば $x_i\prime = c$，である．ここで，c は有理数の定数である．

この二つの条件下で，x_i は，(i) 定数が割り当てられてからの時間，と (ii) その定数の値，によって決定される．ゆえに，クロック変換可能な変数 x_i 上のすべての不変条件，初期条件とジャンプ条件はクロック t_i^x 上の条件に変換される．ここで，クロック t_i^x は x_i が定数に再割り当てされるたびに再スタートされる．x_i が異なる定数に再割り当てされる場合，もし必要ならば制御モードを複製する．

図 3･9 の温度自動調節器のハイブリッドオートマトンの変数 x はクロック変換できる．クロック変換後の線形ハイブリッドオートマトンを図 3･10 に示す．

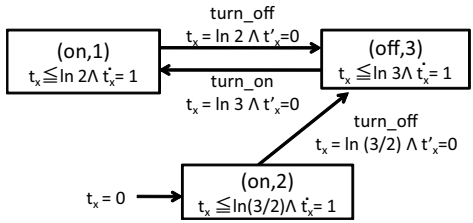


図 3･10　ハイブリッドオートマトンのクロック変換

制御モード on に遷移したときに変数 x に割り当てられる二つの値に対応して，制御モード on は二つの制御モードに分割される．x は $x(t) = -3e^{-t} + 5$ であり，初期値 2 である．$ln(3/2)$ 秒経過すると，温度 3 度 $((off, 3))$ になる．制御モード on の不変条件 $x \leq 3$ は制御モード $(on, 2)$ の $t_x \leq ln(3/2)$ に変換される．制御スイッチ (on, off) のジャンプ条件 $x = 3 \wedge x\prime = x$ は制御スイッチ $((on, 2), (off, 3))$ のジャンプ条件 $t_x = ln(3/2) \wedge t_x\prime = 0$ に変換される．変換されたジャンプ条件はクロック t_x をリセットする．制御モード on に入るときはいつも変数 x は値 1 をもち，値 3 に到達する前の 2 分間は $x(t) = -4e^{-t} + 5$ に従う．クロック変換は元の非線形ハイブリッドオートマトンの軌跡を保存して，変換後のオートマトンは元のオートマトンと時間双模倣な関係にあるので，この変換は TCTL（Timed CTL）で表現できる検証性質を保存する[13]．

(b) 線形フェーズポートレート近似（linear phase-portrait approximation）[12]

線形フェーズポートレート近似のアイディアはより弱い線形条件を用いて，非線形なフロー

条件，不変性条件，初期条件，ジャンプ条件を緩めるということである．具体的には，p ならば p' であるような線形な述語 p' で各非線形な述語 p を置換する．例えば，図 3･11 の線形ハイブリッドオートマトンは図 3･9 の線形フェーズポートレート近似である．

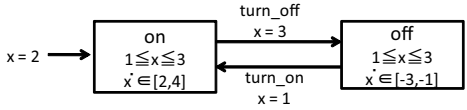


図 3･11　オートマトンの線形フェーズポートレート近似

図 3･9 において，制御モード on で不変条件 $1 \le x \le 3$ とフロー条件 $\dot{x} = -x + 5$ が成り立つならば，x の第一階微分係数は 2 から 4 の範囲にある．ゆえに，非線形なフロー条件 $\dot{x} = -x + 5$ は線形なフロー条件 $\dot{x} \in [2,4]$ に近似される．同様に，制御モード off で非線形なフロー条件 $\dot{x} = -x$ は線形なフロー条件 $\dot{x} \in [-3,-1]$ に近似される．線形フェーズポートレート近似は元の非線形なハイブリッドオートマトンに余分な軌跡を追加する，上からの近似〔本章 3-1-1 参照〕である．もし線形フェーズポートレート近似されたオートマトンがある安全性性質を満たせば，元のオートマトンもその性質を満たすことがいえる．しかし，近似されたオートマトンがある安全性性質を満たさなければ，満たさない軌跡を発見して，それが元のオートマトンの正しい軌跡であるかどうかをチェックする．もしその満たさない軌跡が元のオートマトンの正しい軌跡でなければ，近似の精度を上げて，再度，検証する必要がある．

(4) 安全性の検証

図 3･12 の線形ハイブリッドオートマトンに対して，安全性の検証例を示す．

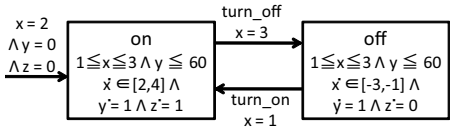


図 3･12　線形ハイブリッドオートマトン

ここで，述語 $y = 60 \wedge z \le (2y/3)$ を両方の制御モード on と off に割り付ける線形な状態の表明を unsafe とする．以降では，述語 p_1 を制御モード on に割り付け，述語 p_2 を制御モード off に割り付ける状態の表明を $\{(on, p_1), (off, p_2)\}$ と表記する．到達可能な状態の計算は初期状態の表明 $\varphi_0 = init = \{(on, x = 2 \wedge y = 0 \wedge z = 0), (off, false)\}$ から開始する．

まず，状態の表明 $\varphi_1 = Post(\varphi_0)$ は以下の二つのステップで計算される．

最初のステップでは，φ_0-状態のすべてのジャンプ後継を見つける．図 3･12 では，on から off への制御スイッチは x が 3 であることが要求されるので，ジャンプ後継は存在しない．

次のステップでは，φ_0-状態のすべてのフロー後継を見つける．このために，以下を用いる．

『状態の表明 φ に対して，述語

$\exists x_1, \ldots, x_n.\ \exists \delta \ge 0.\ \varphi(v) \wedge flow(v) \wedge x_1' = x_1 + \delta \dot{x}_1 \wedge \ldots \wedge x_n' = x_n + \delta \dot{x}_n$

は変数 $x_1\prime, \ldots, x_n\prime$ の値 $a_1, \ldots, a_n$ に対して true である．ただし，$inv(v)$=true とする．

iff

状態 $(v, \mathbf{a})$ は φ が true である状態のフロー後継である．』

線形な述語なので，A. Tarski の実閉体の量化記号消去の定理[14] を用いる．$\varphi = \varphi_0$ と $v = on$ に対して，以下の述語が得られる:

$$\exists x,\ y,\ z.\ \exists \delta \geq 0.\ \exists \dot{x}, \dot{y}, \dot{z}.\ x = 2 \wedge y = 0 \wedge z = 0$$
$$\wedge \dot{x} \in [2,4] \wedge \dot{y} = 1 \wedge \dot{z} = 1 \wedge x\prime = x + \delta\dot{x} \wedge y\prime = y + \delta\dot{y} \wedge z\prime = z + \delta\dot{z}$$
$$= (\exists \delta \geq 0.\ 2 + 2\delta \leq x\prime \leq 2 + 4\delta \wedge y\prime = \delta \wedge z\prime = \delta)$$
$$= (2x\prime + 2 \leq x\prime \leq 4z\prime + 2 \wedge y\prime = z\prime)$$

プライム付きの記号をプライムの付かない記号に名前変えして，制御モード on の不変条件との共通部分をとった後に，述語

$$x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z$$

が得られる．

この述語は初期状態から単一のフローによって到達できる制御モード on をもつ状態を特徴づける．初期状態から単一のフローによって到達できる制御モード off をもつ状態は存在しないので，以下のようになる:

$$\varphi_1 = Post(\varphi_0)$$
$$= \{(on, x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z),\ (off, false)\}$$

次に，$\varphi_2 = Post(\varphi_1)$ を計算する．φ_1-状態のジャンプ後継は以下の状態の表明が true となる状態である:

$$\{(on, false), (off, x = 3 \wedge \tfrac{1}{4} \leq z \leq \tfrac{1}{2} \wedge y = z)\}$$

これは以下の理由による．$x = 3$ のときに on から off への制御スイッチが起こるかもしれないし，x, y や z の値はジャンプによって変化しないし，$2z + 2 \leq 3 \leq 4z + 2$ は $\frac{1}{4} \leq z \leq \frac{1}{2}$ に単純化される．φ_1-状態はフロー後継で閉じている（すなわち，φ_1-状態のすべてのフロー後継は φ_1-状態である）ので，以下のようになる:

$$\varphi_2 = Post(\varphi_1)$$
$$= \{(on, x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z), (off, x = 3 \wedge \tfrac{1}{4} \leq z\tfrac{1}{2} \wedge y = z)\}$$

T.A. Henzinger らは，以上のような計算を記号モデル検査ツール HyTech[10] で行った．HyTech[10] は新しいジャンプ後継とフロー後継が見つからなくなるまで，このような計算を自動で行う．この例では，73 回の繰り返し計算により，（到達可能な状態で true となる）線形な状態の表明 $reach$ を計算して出力した．最後に，安全性の充足性を判断するために，述語 $\exists X.\ \bigvee_{v \in V}(reach(v) \wedge unsafe(v))$ の真偽を判定したら false であった．以上より，$reach$ と $unsafe$ との両方が true となる状態はないので，温度自動調節器は安全性を満たす．

3-3-4 ハイブリッドオートマトンの研究の最新動向

(1) o-minimal ハイブリッドオートマトン

検証アルゴリズムは本質的には到達可能解析アルゴリズムであり，ハイブリッドオートマトンの到達可能解析問題（つまり，ハイブリッドオートマトンの軌跡が望ましくない状態に到達すかどうか）が決定可能（decidable）かどうかは極めて重要である．双模倣関係により，元のハイブリッドオートマトンから有限な商構造を構成することは到達可能解析問題の決定

可能性を示す重要なアプローチであると考えられる．時間オートマトンの場合では，有限な商構造であるリージョングラフを構成して到達可能解析問題の決定可能性を示している[7]．一方，マルチレート時間オートマトンや矩形ハイブリッドオートマトンでは双模倣関係で有限な商構造を構築できないが，時間オートマトンの決定可能性の結果を用いることにより，それらのオートマトンの到達可能解析問題が決定可能であることを示せる[9]．

1999 年に，G. Lafferriere らはモデル理論の o-minimal（order-minimal（順序極小））理論と実閉体の量化記号消去の定理[14] を用いて，双模倣関係により，有限な商構造が構成できるハイブリッドオートマトンのクラスを一般化して，(1) 双模倣関係による有限な商構造の構成により，ハイブリッドオートマトンの決定可能性解析の統一的なフレームワークを構築して，更に，(2) フロー条件が線形微分方程式である，o-minimal ハイブリッドオートマトンの到達可能解析問題が決定可能であることを示した[15]．

(2) ハイブリッドオートマトンの述語抽象化洗練検証

述語抽象化は複雑な無限状態システムから抽象的な有限状態システムを抽出する強力なテクニックである．しかし，抽象的な有限状態システムは元の無限状態システムより多くの動作をもつので，抽象的な有限状態システムの反例は正しくないかもしれない，つまり偽反例である可能性がある．また，R. Kurshan は反例を用いて抽象的なシステムを精錬する手法を開発した[16]．以上を統合化した，反例誘導の抽象化洗練（CounterExample-Guided Abstraction Refinement: CEGAR）の検証パラダイムは，時間オートマトンの検証[17]，C プログラムの検証[18] や記号モデル検査[19] などで使われている．最近，ハイブリッドオートマトン及び確率時間オートマトンを対象として，実数空間上の抽象化述語を用いた反例誘導の抽象化洗練（CounterExample-Guided Abstaction Refinement: CEGAR）の検証手法が R. Alur[20]，E.M. Clarke[21] や山根[22] によって開発されている．

■参考文献

1) O. Maler, “Hybrid Systems and Real-World Computations,” manuscript, 1992.
2) O. Maler, Z. Manna, and A. Pnueli, “From timed to hybrid systems,” LNCS 600, pp.447-484, 1992.
3) R. Alur, C. Courcoubetis, T.A. Henzinger, and P.Ho. Hybrid automata, “An algorithmic approach to the specification and verification of hybrid systems,” LNCS 736, pp.209-229, 1993.
4) R. Alur, T.A. Henzinger, and P.-H. Ho, “Automatic symbolic verification of embedded systems,” RTSS, pp.2-11, 1993.
5) T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, “A user guide to HyTech,” LNCS 1019, pp.41-71, 1995.
6) T.A. Henzinger, Z. Manna, and A. Pnueli, “Towards refining temporal specifications into hybrid systems,” LNCS 736, pp.60-76, 1993.
7) R. Alur and D.L. Dill, “Automata for modeling real-time systems,” LNCS 443, pp.322-335, 1990.
8) T.A. Henzinger, Z. Manna, and A. Pnueli, “Temporal proof methodologies for real-time systems,” POPL, ACM, pp.353-366, 1991.
9) T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya, “What’s decidable about hybrid automata?,” STOC, pp.373-382, 1995.
10) T.A. Henzinger, P.-H.Ho, and H. Wong-Toi, “HyTech: A Model Checker for Hybrid Systems,” Softw. Tools Technol. Trans. 1, pp.110-122, 1997.
11) T.A. Henzinger and P. Ho, “Algorithmic analysis of nonlinear hybrid systems,” LNCS 939, pp.225-238, 1995.

12) T.A. Henzinger and H. Wong-Toi, “Linear phase-portrait approximations for nonlinear hybrid systems,” LNCS 1066, pp.377-388, 1996.
13) T.A. Henzinger, “The theory of hybrid automata,” LICS, pp.278-292, 1996.
14) A. Tarski, “A Decision Method for Elementary Algebra and Geometry,” Manuscript, Santa Monica, CA, RAND Corp., 1948.
15) G. Lafferriere, G.J. Pappas, and S. Yovine, “A New Class of Decidable Hybrid Systems,” LNCS 1569, pp.137-151, 1999.
16) R. Kurshan, “Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach,” Princeton University Press, 1994.
17) R. Alur, A.Itai, R.Kurshan, M.Yannakakis, “Timing verification by successive approximation,” Inf. Comput., vol.118, no.1, pp.142-157, 1995.
18) T. Ball and S.K. Rajamani, “Bebop: A Symbolic Model Checker for Boolean Programs,” LNCS 1885, pp.113-130, 2000.
19) E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-Guided Abstraction Refinement,” LNCS 1855, pp.154-169, 2000.
20) R. Alur, T. Dang, and F. Ivancic, “Predicate abstraction for reachability analysis of hybrid systems,” ACM Trans. Embedded Comput. Syst., vol.5, no.1, pp.152-199, 2006.
21) E.M. Clarke, A. Fehnker, Z. Han, B.H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, “Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems,” Int. J. Found. Comput. Sci., vol.14, no.4, pp.583-604, 2003.
22) 駒形, 森下, 山根, “述語抽象化とその洗練による確率時間オートマトンの到達可能性解析手法,” 電子情報通信学会技術報告 CST2008-5, pp.1-6, 2008.

■7 群 - 1 編 - 3 章

3-4 高速化技法

（執筆者：土屋達弘）[2008 年 6 月 受領]

モデル検査の普及は，**状態爆発問題**（state explosion problem）に対処するための技術の発展に負うところが大きい．空間探索手法の改良により，現在では通常のパーソナルコンピュータでも，状態数が億のオーダの問題でも検証が可能になっている．本節では主要な高速化技法について概説する．

例として，以下の相互排除プログラムを実行する並行プロセス P_0, P_1 を考える．

```
1: while (1) {
2:   while (test-and-set(lock)) {};
3:   lock = 0; }
```

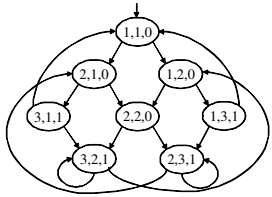


図 3･13　簡単な並行プログラムと状態遷移グラフ

各プロセス $P_i(i = 0, 1)$ がどの行を実行しているかを $pc_i \in \{1, 2, 3\}$ で表すと，$pc_i = 3$ の場合が危険領域である．相互排除は，共有変数 *lock* と，その値に対するビジーウェイトによって実現されている．初期状態において *lock* の値は 0 とする．この並行システムの状態は $(pc_0, pc_1, lock)$ の値の組合せで表現できるので，その動作は図 3･13 の状態遷移グラフで表される．

3-4-1 半順序簡約

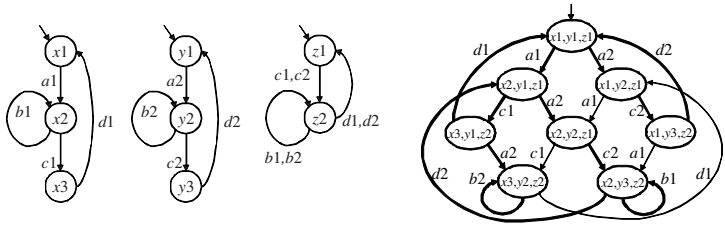
半順序簡約（partial order reduction）は，今や並行システムのモデル検査における標準的な高速化技法となっている．特に，よく知られたモデル検査器 SPIN は半順序簡約を実装しており，極めて大きな状態空間を有すシステムの検証を実現している[5]．

並行システムの検証では，検証したい性質によっては，異なる構成要素の相互関係のない複数の遷移は，それらがどういう順番で起こったとしても，性質の真偽に無関係な場合がある．したがって，検査していない動作でも，性質の真偽が一致することがあらかじめ分かっている別の動作の検査が行われているのであれば，その検査をスキップすることが可能となる．

半順序簡約は，状態空間探索を行う際に，すべての状態を探索するのではなく，検証に十分な動作の検査を保証しつつ，不要な状態空間の探索を避けるよう考慮する遷移を選択する手法である．探索を省略できる状態空間は，どのような性質を検証するかによって変わってくる．ここでは，比較的単純な，デッドロックの検証における半順序簡約について述べる[4]．

並行システムの各構成要素がオートマトンによって表現されており，システム全体の動作は，それらのオートマトンから得られる積オートマトンで表される場合を考える．例えば，図 3･13 の並行プログラムの動作は，図 3･14(a) の三つのオートマトンによってモデル化できる．これらのオートマトンからは図 3･14(b) に示す積オートマトンが得られる．

オートマトンの各遷移は，状態 s，入力記号 a，次状態 s' の三つ組 (s, a, s') で表される．



(a) **オートマトン**

(b) **積オートマトン**

図 3･14　オートマトンによる動作表現

今，構成要素を表す各オートマトン i における遷移集合を $\Delta_i (1 \le i \le n)$，積オートマトンの遷移集合を Δ とする．また，積オートマトンの状態 $\mathbf{s} = (s_1, \cdots, s_n)$ を，$\mathbf{s} = \{s_1, \cdots, s_n\}$ のように，対応する各オートマトンの状態の集合として表わす．このとき，積オートマトンにおける遷移 $t = (\mathbf{s}, a, \mathbf{s}')$ について，${}^\bullet t$, $t^\bullet$, ${}^\bullet t^\bullet$, を以下のように定める．

$$ {}^\bullet t = \{s_i \in \mathbf{s} : (s_i, a, s_i') \in \Delta_i\}, \quad t^\bullet = \{s_i' \in \mathbf{s}' : (s_i, a, s_i') \in \Delta_i\}, \quad {}^\bullet t^\bullet = {}^\bullet t \cup t^\bullet $$

以降，${}^\bullet t = {}^\bullet t'$ かつ $t^\bullet = t'^\bullet$ であるような $t, t' \in \Delta$ すべてをまとめて，一つの遷移と呼ぶ．そして，二つの遷移 t_1, t_2 について，${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$ が成り立つ場合，これらの遷移は互いに独立であると定義する．例えば，$a1$ と $a2$ に対応する遷移は，${}^\bullet a1 = \{x1\}, a1^\bullet = \{x2\}, {}^\bullet a2 = \{y1\}, a2^\bullet = \{y2\}$ であり，互いに独立である．

初期状態から実行可能な遷移の有限系列を考えたとき，その系列の上で連続する二つの遷移 t_1, t_2 が互いに独立な場合，それらを入れ替えて t_2, t_1 の順とした系列も実行可能であり，かつ，両系列でこれら二つの遷移を実行した直後の状態が一致する．これは以下のように簡単に示すことができる．

二つの遷移系列について，これらの遷移による状態の変化をそれぞれ $s_{i-1} \xrightarrow{t_1} s_i \xrightarrow{t_2} s_{i+1}$, $s_{i-1} \xrightarrow{t_2} s_i' \xrightarrow{t_1} s_{i+1}'$ と表す．このとき，$s_i = (s_{i-1} \backslash {}^\bullet t_1) \cup t_1^\bullet$, $s_{i+1} = (s_i \backslash {}^\bullet t_2) \cup t_2^\bullet$ より，$s_{i+1} = (((s_{i-1} \backslash {}^\bullet t_1) \cup t_1^\bullet) \backslash {}^\bullet t_2) \cup t_2^\bullet$ となる．また，同様に $s_{i+1}' = (((s_{i-1} \backslash {}^\bullet t_2) \cup t_2^\bullet) \backslash {}^\bullet t_1) \cup t_1^\bullet$ である．t_1, t_2 は独立，すなわち，${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$ だから，結局 $s_{i+1} = s_{i+1}'$ となる．

この性質から，隣り合う独立した遷移を何度入れ替えても，遷移系列を最後まで実行した結果到達する最終状態は変化しないことがいえる．したがって，デッドロック状態，あるいは、ある 1 グローバル状態への到達可能性を検証するのには，独立した遷移を入れ替えて得られる複数の遷移系列の中の一つを調べれば十分であることが分かる．

通常のモデル検査では，初期状態から実行可能な遷移を一遷移ずつ考慮して状態探索を行う．半順序簡約では，上記のように，検証する性質の真偽が一致することがあらかじめ分かっている異なる遷移系列を多重に探索しないよう，各状態において実行可能な遷移の一部のみを選択して探索を行う．図 3･14(b) において線を強調している状態と遷移は，文献 4) のアルゴリズムによって実際に探索される部分を示している．この例は小さいため，結局すべての到達可能な状態が探索されているが，問題の規模が大きくなると探索を省略できる部分が増

え，検証が高速化される．

より一般的な性質の検証に関しては，たとえば，LTL_x 式や CTL*式の一部の真偽値を保存することが可能な遷移の選択方法が開発されている[8, 3]．LTL_x は LTL から次状態を表す X オペレータを除いたもの，CTL*は LTL と CTL とを真に含む時相論理のクラスである．

3-4-2 対称性の利用

類似した複数の要素から構成される場合など，システムが**対称性**（symmetry）を有することがある．例えば，図 3･13 の例のように，同じアルゴリズムを実行する二つの並行プロセス P_0, P_1 からなるシステムを考える．このとき，検証したい性質によっては，あるシステム状態と，そのシステム状態から P_0 と P_1 の状態を入れ替えて得られる対称的なシステム状態とを「同値」と見なすことができる．同値な複数の状態を一つの状態として扱うことで，検証の高速化が実現できる．このような手法を実装した代表的なモデル検査ツールに Murϕ がある[6]．

形式的には，対称性はクリプキ構造（もしくは，状態遷移グラフ）上の**自己同型**（automorphism）として表現される．状態集合を S，初期状態集合を $S0$，遷移関係を R，原子命題の状態へのラベルづけ関数を L としたとき，クリプキ構造 $M = (S, S0, R, L)$ 上の自己同型 h を，状態集合 S から状態集合 S への全単射で，以下の性質を満たすものとする．

1) $(s, s') \in R$ かつその場合のみ $(h(s), h(s')) \in R$.

2) $s \in S0$ かつその場合のみ $h(s) \in S0$.

3) 原子命題 p と状態 $s \in S$ について，$p \in L(s)$（つまり s で p が成り立つ）の場合，かつその場合のみ $p \in L(h(s))$.

簡単にいえば，自己同型とは構造が変わらないような状態の置換である．定義より，恒等写像も自己同型であり，また，自己同型の逆写像も自己同型である．したがって，写像の合成を積と定義することで，自己同型の集合は代数上の群を構成する．このような自己同形群を G とし，S の 2 状態 s_1, s_2 について，$s_1 = h(s_2)$ となる $h \in G$ が存在する場合，かつその場合のみ成り立つような関係を考えると，この関係は同値関係であることが分かる．

同値な状態の集合である同値類を 1 状態とみなすことで，検証に必要な元の構造の情報を保存した，より小さい構造を得ることができる．このような構造を**商構造**（quotient structure）と呼ぶ．状態 $s \in S$ が含まれる同値類を $\theta(s)$ とすると，商構造 $M_G = (S_G, S0_G, R_G, L_G)$ は次のように定義される．

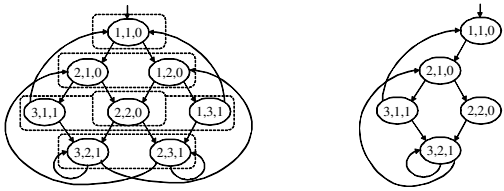
$$S_G = \{\theta(s) \mid s \in S\}, \quad S0_G = \{\theta(s) \mid s \in S0\},$$
$$R_G = \{(\theta(s_1), \theta(s_2)) \mid (s_1, s_2) \in R\}, \quad L_G(\theta(s)) = L(s)$$

こうして得られる商構造の各状態での CTL*式の真偽値は，元のクリプキ構造上の対応する状態におけるそれと一致することがいえる．これは，クリプキ構造と商構造が双模倣関係（bisimulation relation）となることから導ける．したがって，商構造の探索によって検証が可能なことが分かる．

このように対称性はクリプキ構造に関する自己同型として捉えられるが，実際の検証においては，同じ動作をするプロセスの集合など，それらの状態を要素間で置き換えても同値なグローバル状態が得られることがあらかじめ分かっているような構成要素を与えることで，対称性を指定する．

図 3･13 の例では，プロセス P_0 と P_1 がそのような構成要素となる．原子命題を $pc_0 = 3 \wedge pc_1 = 3$ のみとしたとき，$h(pc_0, pc_1, lock) = (pc_1, pc_0, lock)$ という二つのプロセスの状態を入れ替える状態から状態への写像 h は，クリプキ構造上の自己同型となる．また，h と恒等写像からなる自己同型の集合は群となる．この群から得られる同値類を図 3･15(a) に，商構造を図 3･15(b) に示す．

商構造を構成する際，商構造の各状態には対応する同値類の代表元を割り当てる．このとき，元のクリプキ構造の探索範囲は，代表元と代表元から 1 回の遷移で行ける状態のみで十分である（代表元から代表元に遷移があるとは限らないことに注意）．図 3･15(b) では，pc_0 の値が大きい状態を代表元としている．また，同値類で最初に探索した状態を代表元とする手法もある[9]．



(a) **同値類**　　(b) **商構造**

図 3･15　対称性の利用による状態削減

3-4-3 記号モデル検査

半順序間約と対称性の利用という二つの手法は，状態探索において，探索の不要な部分を特定し，高速化を図っていた．**記号モデル検査**（symbolic model checking）は，状態空間自体の表現を工夫することで，高速化を達成する手法である[7]．より詳しくいえば，状態や遷移の一つ一つを明示的に区別して扱うのではなく，状態の集合や遷移の集合を数式によって**記号的に**(symbolically) 表現し，数式に関する演算によってモデル検査を行う．状態グラフが莫大な規模となるようなシステムでも，非常にコンパクトな記号表現が得られる場合が多く，その場合，記号モデル検査により大幅な検証の高速化が実現できる．

一般に，システムはいくつかの変数で表され，その状態はこれらの変数の値で表現される．このような変数をここでは状態変数と呼ぶ．$x_1, x_2, \cdots, x_n$ を状態変数としたとき，どのような状態集合 $\mathcal{S}$ も，状態変数上のブール値を有する式 $f_{\mathcal{S}}$ によって次のように表現できる．

$$f_{\mathcal{S}} = \text{true} \Longleftrightarrow (x_1, x_2, \cdots, x_n) \in \mathcal{S}$$

例えば, 図 3･13 の例では, システムの状態は 3 変数 $pc_0, pc_1, lock$ で表され, 初期状態は $(1, 1, 0)$ なので，初期状態の集合は $I \triangleq pc_0 = 1 \wedge pc_1 = 1 \wedge lock = 0$ と表現できる．同様に相互排除が成り立つ状態すべての集合は（到達不可能な状態も含めて）$MUX \triangleq \neg(pc_0 = 3 \wedge pc_1 = 3)$ と表される．

遷移関係 R は，状態変数と次状態での状態変数上の式として表現できる．x'_i を x_i に対応する次状態における状態変数とすると，遷移関係は以下の条件を満たす式 T で表される．

$$T = \mathsf{true} \Longleftrightarrow ((x_1, x_2, \cdots, x_n), (x'_1, x'_2, \cdots, x'_n)) \in R$$

例として，図 3･13 の遷移関係の記号表現を以下に示す．

$$T \triangleq \left\{ \begin{array}{ll} & pc_0 = 1 \wedge pc'_0 = 2 \wedge pc'_1 = pc_1 \wedge lock' = lock \\ \vee & pc_0 = 2 \wedge lock = 1 \wedge pc'_0 = 2 \wedge pc'_1 = pc_1 \wedge lock' = lock \\ \vee & pc_0 = 2 \wedge lock = 0 \wedge pc'_0 = 3 \wedge pc'_1 = pc_1 \wedge lock' = 1 \\ \vee & pc_0 = 3 \wedge pc'_0 = 1 \wedge pc'_1 = pc_1 \wedge lock' = 0 \\ \vee & pc_1 = 1 \wedge pc'_1 = 3 \wedge pc'_0 = pc_0 \wedge lock' = lock \\ \vee & pc_1 = 2 \wedge lock = 1 \wedge pc'_1 = 2 \wedge pc'_0 = pc_0 \wedge lock' = lock \\ \vee & pc_1 = 2 \wedge lock = 0 \wedge pc'_1 = 3 \wedge pc'_0 = pc_0 \wedge lock' = 1 \\ \vee & pc_1 = 3 \wedge pc'_1 = 1 \wedge pc'_0 = pc_0 \wedge lock' = 0 \\ \vee & \neg(pc_0 = 1 \vee pc_0 = 2 \wedge lock = 1 \vee pc_0 = 2 \wedge lock = 0 \vee pc_0 = 3 \\ & \quad \vee pc_1 = 1 \vee pc_1 = 2 \wedge lock = 1 \vee pc_1 = 2 \wedge lock = 0 \vee pc_1 = 3) \\ & \wedge pc'_0 = pc_0 \wedge pc'_1 = pc_1 \wedge lock' = lock \end{array} \right.$$

最後の項は，ほかの状態に遷移できない状態 s については，$T(s, s') = \mathsf{true} \Leftrightarrow s = s'$ が成り立つように，言い換えると次状態で状態が変化しないことを表している．ここで重要なことは，この式 T は，状態を探索することなしに，システムの記述から直接導かれる点である．

状態探索は，状態集合や遷移関係を記号的に表現した数式の演算によって行われる．その基本となる演算は，**像計算**（image computation），及び**逆像計算**（preimage computation）である．像計算と逆像計算は，それぞれ，ある状態集合から 1 回の遷移で到達できる状態の集合，及び，ある状態集合に 1 回の遷移で到達できる状態の集合を求める演算であり，結果的に幅優先探索を実現していることになる．例えば，I で表される初期状態集合から像計算を繰り返すことで，到達可能状態の集合を得ることができる．

典型的な記号モデル検査では，状態集合や遷移関係をブール式で表し，ブール式を表現・操作するためのデータ構造として**二分決定グラフ**（binary decision diagram: BDD）を用いる．BDD を用いた記号モデル検査ツールとしては SMV が著名である[7)]．

また，整数変数を状態変数とし，整数上の線形制約式を用いる記号モデル検査手法もある．状態変数の定義域が大きい場合，BDD を用いる手法よりも高速に検証が可能な場合があることが報告されている[2)]．

3-4-4 有界モデル検査

有界モデル検査（bounded model checking）[1)] は，記号表現を用いるという点で記号モデル検査の一種といえるが，先に説明した手法とは空間探索手法が全く異なる．具体的には，有界モデル検査では充足可能性判定（SAT）を利用して探索を行う．有界という名が示すとおり，この手法では初期状態から一定の回数の遷移を考慮してモデル検査を行う．以下，この回数を k で表す．

例として，ある性質 P が成り立つ状態への到達可能性の検証について考える．P は状態変数上の式として与えられているとすると，以下の式の充足可能性判定を行うことで検証が可能となる．

$$I(0) \wedge T(0,1) \wedge \cdots \wedge T(k-1,k) \wedge \bigvee_{0 \le i \le k} P(i)$$

ただし，状態変数上の式 f に対し，$f(i)$ は式 f 中の各状態変数 x を x^i に置き換えて得られる式，$T(i,i+1)$ は，T の状態変数 x を x^i に，次状態の状態変数 x' を x^{i+1} に置き換えて得られる式とする．この結果，上の式は，各状態変数 x に対応した k 個の変数 $x^0, x^1, \cdots, x^k$ を含むことになり，これらの変数への値の割当によって k 状態を表すことができる．以下，$(x_1^i, x_2^i, \cdots, x_n^i)$ によって表される状態を，s^i $(i = 0, 1, \cdots, k)$ で表す．

上の式の変数への値の割当によって，最後の $P(i)$ の論理和を除いた部分が真になる必要十分条件は，s^0 が初期状態であり，かつすべての $0 \le i \le k-1$ に対して，状態 s^i から状態 s^{i+1} へ遷移可能であることである．また $\bigvee_{0 \le i \le k} P(i)$ は $s^0, \cdots, s^k$ のいずれかの状態で P が成り立つ場合かつその場合のみ真となる．よって，式全体が充足可能である（真と評価され得る）必要十分条件は，初期状態から k 回以下の遷移で P が成り立つ状態への到達が可能なことである．

したがって，充足可能性判定により充足可能と分かれば，到達可能性が結論できる．しかし，充足不能の場合，初期状態から $k+1$ 回以上の遷移によって P が成り立つ状態に到達する可能性は排除できない．有界モデル検査という呼称は，このように与えられた遷移回数内で性質が成り立つこと，もしくは，成り立たないことを示すという手法の特性に由来する．判定する式を工夫することで，到達可能性だけでなく，一般の LTL 式について検証が可能である．

通常の記号モデル検査と同様，多くの場合，ブール式による記号表現が用いられる．ブール式の充足可能性判定問題はもっとも著名な NP 完全問題であるが，有効なヒューリスティックが多く開発され，近年では非常に大きな式に対しても判定が可能になっている．

■参考文献

1) A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu, “Symbolic model checking without BDDs,” In Proc. of Tools and Algorithms for the Analysis and Construction of Systems (TACAS’99), LNCS 1579, pp.193-207, 1999.
2) T. Bultan, “BDD vs. constraint-based model checking: An experimental evaluation for asynchronous concurrent systems,” In Proc. of Tools and Algorithms for the Analysis and Construction of Systems (TACAS’00), LNCS 1785, pp.441-455, 2000.
3) R. Gerth, R. Kuiper, D. Peled, and W. Penczek, “A partial order approach to branching time logic model checking,” Information and Computation, vol.150, no.2, pp.132-152, 1999.
4) P. Godefroid and P. Wolper, “Using partial orders for the efficient verification of deadlock freedom and safety properties,” Formal Methods in System Design, vol.2, no.2, pp.149-164, 1993.
5) G.J. Holzmann, “The SPIN Model Checker,” Addison-Wesley Longman Publishing Co., Inc., 2004.
6) C.N. Ip and D.L. Dill, “Better verification through symmetry,” Formal Methods in System Design, vol.9, nos.1-2, pp.41-75, 1996.
7) K.L. McMillan, “Symbolic Model Checking,” Kluwer Academic, 1993.

8) D. Peled, “Combining partial order reductions with on-the-fly model-checking,” In Proc. of 6th Conf. on Computer Aided Verification (CAV’94), pp.377-390, 1994.
9) A.P. Sistla, V. Gyuris, and E.A. Emerson, “SMC: A symmetry-based model checker for verification of safety and liveness properties,” ACM Trans. Software Engineering and Methodology, vol.9, no.2, pp.133-166, 2000.