離散事象システムのモデル化と制御

平石邦彦[1]

平成 12 年 7 月 18 日

[1]北陸先端科学技術大学院大学情報科学研究科

Chapter 1

はじめに

離散事象システム (DES:discrete event system) とは，事象が離散的なタイミングで生起することにより状態が不連続に変化するようなシステムの総称である．特に，内部状態をもち，システムの出力が過去の入力系列に依存するような離散事象システムのことを離散事象動的システム (DEDS:discrete event dynamic system) とよび，主な研究対象となっている．

離散事象システムに分類される対象は，コンピュータ・ソフトウェア／ハードウェア／ネットワーク，通信システム，FMS(flexible manufacturing system)，シーケンス制御 (sequential control)，さらには人間の活動をも含んだグループウェアなどきわめて広範囲にわたっている．これらのシステムのモデル化および解析については従来それぞれの分野で独立に行われてきたが，対象が異なっても扱われる問題が本質的にもつ困難さは共通であるという認識が定着しつつある．

離散事象システムの理論とは，離散事象システムに分類される様々な対象をシステム理論の立場から統一的に扱うことを目的としており，大きく分けて 2 つの流れがある．1 つはシステムの論理的な側面を扱う理論，もう 1 つはシステムの性能評価 (performance evaluation) に関する理論である．

論理モデルに関しては Ramage と Wonham により提唱された形式言語による定式化が最も広く研究されている．この分野における理論研究者の多くは，制御・システム理論をバックグラウンドにもつ人が多く，その内容も連続系の制御理論のアナロジーとして議論されることが多い．一方で，性能評価モデルとしての離散事象システムは待ち行列理論 (queueing theory) をバックグラウンドにしており，その解析手法も実用的なシステムに対してはシミュレーションを用いることが多い．これら 2 つを狭義の離散事象システムの理論と位置付けることができる．

一方で，広義の離散事象システムは，その対象が広いためか，共通のモデル，解

析方法が確立されるには至っていない．理論面では対象を記述するためのモデルごとに議論が行われているが，最近ではそれらのモデルの関係について論じた研究も行われている．応用面では，ソフトウェア工学においてソフトウェアのモデル化という観点からさまざまな研究が行なわれている．さらに，離散事象システムでは状態の変化が個々の事象の生起により生じるということから，動作の並行性，非同期性を考慮しなくてはならない．この点で，理論計算機科学の分野で行われている並行計算モデルの理論とも関係が深い．

本稿ではまず，Ramage と Wonham により提唱された形式言語による離散事象システムの制御理論について述べる．つぎに，離散事象システムを記述するための代表的な 3 つのモデルをとりあげ，それらについて比較を行なう．最後に，設計されたシステムの効率的な解析方法についての述べる．

Chapter 2

スーパバイザ制御

Ramage と Wonham により提唱された**スーパバイザ制御** (supervisory control)[39, 45, 27] について説明する．スーパバイザ制御では，形式言語とオートマトンを用いて離散事象システムを記述するが，基本的概念は連続系の制御理論における状態空間表現のアナロジーとなっている．

2.1 例題

離散事象システムの制御問題を考えるために，Ramage と Wonham らによる例題をまず述べる．猫とネズミが Figure 2.1に示す迷路内にいる．各部屋の間のドアは猫専用 (c_i) とネズミ専用（m_j）に分かれており，矢印に示された向きにしか通れない．猫とネズミのそれぞれの動作は Figure 2.2の 2 つのオートマトンで表現される．制御対象はこの 2 つのオートマトンが合成されたものである（Figure 2.3）[1]．合成されたオートマトンの各状態は合成されるオートマトンの各状態の組になる．

この問題では，ドアの開閉（これは 1 つの事象である）を制御することにより，以下の制御目的を満たし，かつ，できるだけ自由度の大きい動作をさせることが目的である．

[1] オートマトンの合成については，第 3 章でラベル付遷移システムの同期合成という形で厳密に定義する．

制約 1　猫とネズミは同時に同じ部屋に存在してはならない．
（猫がネズミを捕まえてしまう）．

制約 2　猫とネズミは常に初期状態（猫は部屋 2，ネズミは部屋 4）に戻れる．

目的を達成するだけでは不十分なのは，たとえばつぎの事実を見れば明らかであろう．

- すべてのドアを閉じたままにする．これにより，猫とネズミは初期状態から全く移動できないが，制御仕様は満たされる．

すべてのドアの開閉が制御可能ならば，上記の目的を達成する制御が存在することは明らかである．しかし，現実にはつぎのことを考慮する必要がある．

- すべての事象が制御可能とは限らない．あるいは，
- すべての事象を制御しなくても制御目標を達成できるかもしれない．

制御不可能な事象を**不可制御事象** (uncontrollable event)，制御可能な事象を**可制御事象** (controllabel event) という．この例題ではドア c_7の開閉は不可制御事象と仮定される．

一般にスーパバイザ制御ではつぎの問題を扱う．

1. 与えられた制御仕様は達成可能であるかどうかを判定せよ．
2. 制御仕様が達成可能な場合，制御仕様を満たし，かつ，できるだけ自由度の大きい動作をさせる制御を求めよ．
3. 制御仕様が達成不可能な場合，制御仕様を最大限満たすような制御を求めよ．

2.2　制御対象の表現

まず，制御対象 G をつぎのようなオートマトンとして表現する．G を通常，**プラント** (plant) と呼ぶ．

$$G = (E, Q, \delta, q_0, Q_m) \tag{2.1}$$

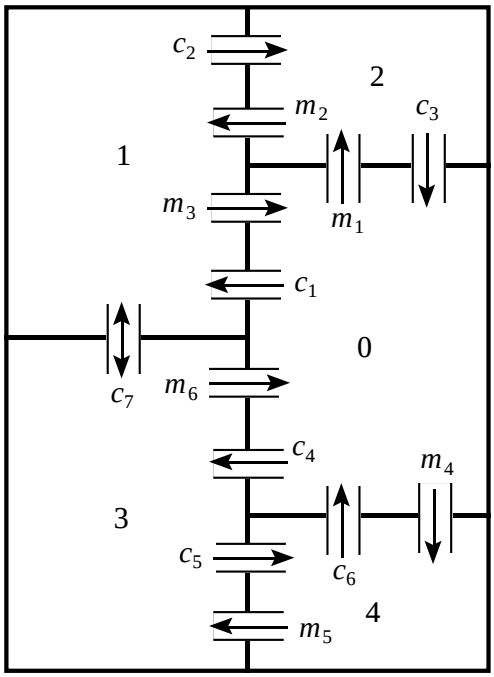

Figure 2.1: 迷路の例題.

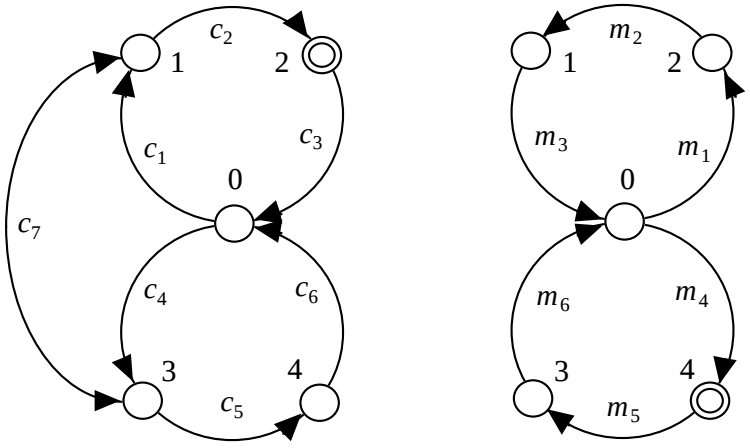

Figure 2.2: 制御対象のオートマトン.

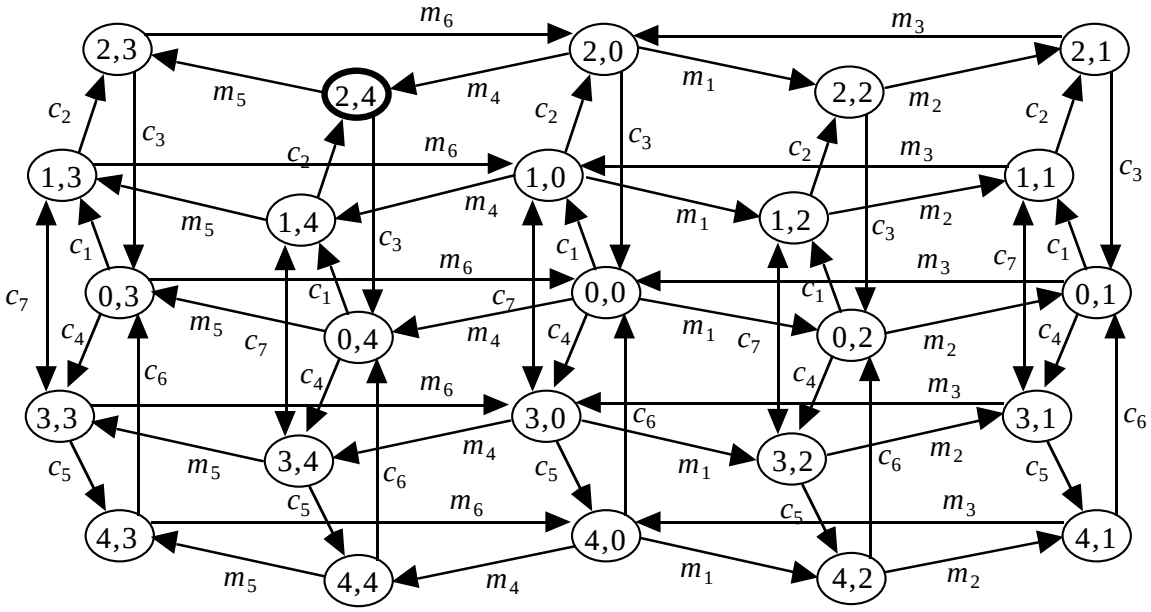

Figure 2.3: オートマトンの合成.

ここで，Eは事象の集合，Q は状態の集合[2]，$\delta : Q \times E \to Q$ は状態遷移関数，$q_0 \in Q$ は初期状態，$Q_m \subset Q$ は目標状態の集合を表す．E^*は E上のすべての有限長の系列の集合を表す．δは事象系列に対する状態遷移関数 $\delta : Q \times E^* \to Q$ に拡張できる．すなわち，空系列を λ とすると，
(i) $\delta(q, \lambda) = q$;
(ii) $\delta(q, \sigma e) = \delta(\delta(q, \sigma), e)$ $(\sigma \in E^*, e \in E)$
により定義される．δ は一般的には部分関数であり $\delta(q, e)!$により$\delta(q, e)$ が定義されていることを表す.

オートマトン G に対し 2 種類の言語を定義する.

$$L(G) = \{\sigma \in E^* \mid \delta(q_0, \sigma)!\} \tag{2.2}$$

$$L_m(G) = \{\sigma \in E^* \mid \delta(q_0, \sigma) \in Q_m\} \tag{2.3}$$

$L(G)$ は接頭語について閉じた (prefix-closed) 言語であり，実行可能な事象系列の集合を表す．$L_m(G)$ は G により受理される言語である.

スーパバイザ制御における制御は状態遷移の制限により行われる．すなわち，事象生起の各ステップにおいてつぎに生起を許可する事象集合 γ （これを**制御パターン**という）を与え，それに属する事象だけが生起できるようにする．これは，可能

[2]一般には無限状態.

な状態遷移はあらかじめ網羅されているという立場である．前述のように，事象は可制御事象と非可制御事象の 2 種類に分類され，E_c, E_u によりそれぞれの集合を表す．すなわち，$E = E_c \cup E_u$, $E_c \cap E_u = \emptyset$ である．非可制御事象は，外部からの入力，故障の発生などに対応し，その発生を抑制できない．これは，各制御パターン γ が常に E_uを含むことで表現される．したがって，可能な制御パターン全体の集合 Γ は

$$\Gamma = \{\gamma \mid E_u \subset \gamma \subset E\} \tag{2.4}$$

と表される．

制御パターンΓにより制御されたプラント G_cはつぎのように表される．

$$G_c = (\Gamma \times E, Q, \delta_c, q_0, Q_m) \tag{2.5}$$

ここで，状態遷移関数 $\delta_c : Q \times \Gamma \times E \to Q$ はつぎのように定義される．

$$\delta_c(q, \gamma, e) = \begin{cases} \delta(q, e) & \delta(q, e)! \wedge e \in \gamma \\ \text{未定義} & \text{それ以外} \end{cases} \tag{2.6}$$

2.3 スーパバイザ

事象を生起させる各ステップにおいて制御パターンを切替える（スイッチング）ことにより制御が行われる．過去の事象系列から制御パターンを作り出す制御器を**スーパバイザ** (supervisor) とよぶ．スーパバイザは形式的には関数 $f : L(G) \to \Gamma$として定義される．スーパバイザ fにより制御されるプラント G は Figure 2.4に示すような閉ループシステムを作り，その挙動は以下に定義される言語 $L(G, f)$ により表される．

1. $\lambda \in L(G, f)$;
2. $\sigma e \in L(G, f) \Leftrightarrow \sigma \in L(G, f) \wedge e \in f(\sigma) \wedge \sigma e \in L(G)$.

また，目標状態に到達する言語は $L_m(G, f) = L_m(G) \cap L(G, f)$ となる．

ここで注意したいのは，スーパバイザによる制御パターンに含まれる事象であっても，$L(G)$ に含まれない，すなわちプラントで生起できない事象は，閉ループシステムにおいても生起しないことである．スーパバイザは事象の生起を許容するだけであり，強制するものではない．

このようなスーパバイザの実装は，関数 fをオートマトン $T = (E, X, \xi, x_0, X)$ および関数 ϕ: $X \to \Gamma$ の対で表現することで行われる．すなわち，各$\sigma \in L(G, f)$

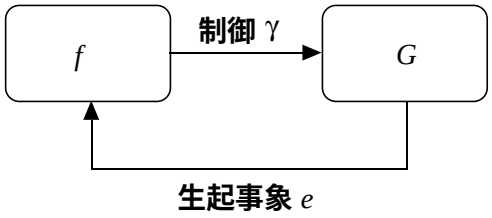

Figure 2.4: 閉ループシステム.

について

$$\phi(\xi(x_0, \sigma)) = f(\sigma) \tag{2.7}$$

を満たすとき，対 $S = \langle T, \phi \rangle$ はスーパバイザ fを**実現する**という．制御理論の用語を用いれば，Tは観測器であり，ϕ はフィードバックである．与えられたスーパバイザ fを実現する $S = \langle T, \phi \rangle$ は一般には一意ではない.

プラント G とスーパバイザ Sが作る閉ループシステムはつぎのような 1 つのオートマトンで表される.

$$S/G_c = (E, X \times Q, \xi \times \delta_c, (x_0, q_0), X \times Q_m) \tag{2.8}$$

ここで，$\xi \times \delta_c((x, q), e) = (\xi(x, e), \delta_c(q, \phi(x), e))$ である．一般に,

$$L(S/G_c) \subset L(G) \tag{2.9}$$

$$L_m(S/G_c) = L(S/G_c) \cap L_m(G) \tag{2.10}$$

が成り立つ.

閉ループシステムが定義されるためには，スーパバイザが**完全**でなければならない．以下の条件が満たされるとき，スーパバイザ $S = \langle T, \phi \rangle$ は完全であるという.

$$\sigma \in L(S/G_c),\ \sigma e \in L(G),\ \text{かつ},\ e \in \phi(\xi(x_0, \sigma))\ \text{ならば，}\ \xi(x_0, \sigma e)!.$$

これは，ある事象が G において生起可能でかつそのときの制御パターンによっても生起が許容されているならば，Tにおいても生起可能であることを意味する.

2.4 可制御性

スーパバイザ制御の枠組では制御とは状態遷移の制限なので，制御仕様は生起可能な言語の部分集合 $K \subset E^*$の形で与えられる．そして，$L(S/G_c) = K$となるスー

パバイザ Sを求めることが目的となる．また，目標状態に到達する言語に関しても，制御目的は $K_m \subset L_m(G)$ で与えられ，$L_m(S/G_c) = K_m$となるスーパバイザ Sを求めることが目的となる．

言語 L の接頭語についての閉包 (prefix closure) を $pr(L)$ で表す．すなわち，

$$pr(L) = \{\sigma \in E^* \mid \exists \nu \in E^* : \sigma\nu \in L\} \tag{2.11}$$

また，2 つの言語 $L_1, L_2 \subset E^*$に対し，それらの**連接** (concatenation) を

$$L_1 L_2 = \{\sigma\nu \mid \sigma \in L_1, \nu \in L_2\} \tag{2.12}$$

により定義する．

$pr(L_m(G)) = L(G)$ のとき G は**ブロックしない** (nonblocking) という．これは，どのような状態遷移を行ってもいずれは目標状態に到達することを意味する．S/G_c がブロックしないシステムのとき，Sはブロックしないスーパバイザであるという．

言語 $K \subset E^*$は以下を満たすとき**可制御** (controllable) であるという．

$$pr(K)E_u \cap L(G) \subset pr(K) \tag{2.13}$$

これは，Kの任意の接頭語σについて，それに続いて不可制御事象 $e \in E_u$が生起する系列 σe がプラント G において生起可能であるならば，それは Kの接頭語になっている（すなわち，許容される動作である）という条件である．許容された動作σの後の不可制御事象 e の発生は妨げることができないので，それもまた許容された動作でなくてはならない．

問題 2.1 以下を示せ．$K \subset L(G)$ が可制御ならば $pr(K)$ も可制御である．逆に，$pr(K)$ が可制御ならば Kは可制御である．

以下の結果が得られている．

定理 2.1 G はプラントとする．このとき，非空の $K \subset L(G)$ について $L(S/G_c) = K$ となる完全なスーパバイザ Sが存在するための必要十分条件は，Kが可制御，かつ，接頭語について閉じている（$K = pr(K)$）ことである．

証明 十分性：言語 Kを生成するオートマトンを $T = (E, X, \xi, x_0, X)$ とおく．すなわち，$L(T) = K$である．このとき，関数 $\phi : X \to \Gamma$ をつぎのように定義する．

$$\phi(x) = \{e \mid \xi(x, e)!\} \cup E_u.$$

スーパバイザ $S = \langle T, \phi \rangle$ に対し，事象系列の長さに関する帰納法を用いて $L(S/G_c) = K$を示す．$K, L(G)$ は接頭語について閉じているので $\lambda \in K \cap L(S/G_c)$ である．いま，$\sigma \in K \cap L(S/G_c)$ となる事象系列 σ を考えると，もし$\sigma e \in K$ならば，$e \in \phi(\xi(x_0, \sigma))$ および $\sigma e \in K \subset L(G)$ が成立するので，$\sigma e \in L(S/G_c)$ である．したがって，

$$K \subset L(S/G_c)$$

となる．逆に $\sigma e \in L(S/G_c)$ とおく．$e \in E_c$ならば，スーパバイザ Sの定義より σe は Tで生起可能であり $\sigma e \in K$である．$e \in E_u$のときは可制御性より $\sigma e \in K$である．したがって，

$$L(S/G_c) \subset K$$

となる．ゆえに，$L(S/G_c) = K$である．Sの完全性は定義より明らか．

必要性：$L(S/G_c) = K$より Kは接頭語について閉じている．いま，Kが可制御でないと仮定すると，Kは空でないので
(i) $\sigma e \in KE_u \cap L(G)$,
(ii) $\sigma e \notin K$
であるような $\sigma \in K$, $e \in E_u$が存在する．ところが，$L(S/G_c) = K$より，$\sigma \in L(T) \cap L(G)$ であり，$e \in E_u$なので $e \in \phi(\xi(x_0, \sigma))$ である．スーパバイザが完全であること，および，(i) より $\sigma e \in L(S/G_c) = K$となるが，これは (ii) に矛盾する．したがって，Kは可制御である． ∎

定理 2.2 G はブロックされないプラントとする．このとき，非空の $K_m \subset L_m(G)$ について $L_m(S/G_c) = K_m$かつ閉ループシステムがブロックされないようなスーパバイザ Sが存在するための必要十分条件は，K_mが可制御，かつ，$pr(K_m) \cap L_m(G) = K_m$である．

証明 必要性：$L_m(S/G_c) = K_m$を満たす完全でブロックしないスーパバイザ Sを考える．$L_m(S/G_c) = K_m$より $pr(K_m) = pr(L_m(S/G_c))$ であり，また，Sはブロックしないことから $pr(L_m(S/G_c)) = L(S/G_c)$ なので，$pr(K_m) = L(S/G_c)$ が得られる．定理 2.1より K_mは可制御である．さらに，

$$\begin{aligned} pr(K_m) \cap L_m(G) &= pr(L_m(S/G_c)) \cap L_m(G) \\ &= L(S/G_c) \cap L_m(G) \\ &= L_m(S/G_c) = K_m \end{aligned}$$

である．

十分性：K_mは可制御なので $pr(K_m) = L(S/G_c)$ となるスーパバイザが S存在する．このとき，

$$\begin{aligned} L_m(S/G_c) &= L(S/G_c) \cap L_m(G) \\ &= pr(K_m) \cap L_m(G) = K_m \end{aligned}$$

が得られる．したがって，Sはブロックしないスーパバイザである． ∎

問題 2.2 スーパバイザが関数 $f : Q \to \Gamma$として定義されるとき，このような制御を状態フィードバックという．状態フィードバックによるスーパバイザが構成できるのは，制御目的 $K \subset E^*$がどのような言語の場合か．

2.5 最大可制御部分言語

制御仕様 Kが可制御でないとき，$L(S/G_c) = pr(K)$ となるスーパバイザは存在しない．このとき，制御仕様を最大限満たすようなスーパバイザを設計するのが自然である．これは，Kの可制御な部分言語[3]の中で（もし存在すれば）最大な言語を実現するスーパバイザを求めることに対応する．

まず，制御仕様 Kに対し，可制御な部分言語全体の集合を $\mathcal{C}(K)$ とおく．

補題 2.3 $\mathcal{C}(K)$ は空でない．また，集合和に関して閉じている．

証明 空集合$\emptyset$ は可制御であることから$\emptyset \in \mathcal{C}(K)$．添字集合 Iに対し，任意の $i \in I$ について $K_i \in \mathcal{C}(K)$ であるとする．このとき，任意の $i \in I$について

$$pr(K_i)E_u \cap L(G) \subset pr(K_i)$$

である．閉包の定義より，

$$pr(\bigcup_{i\in I} K_i) = \bigcup_{i\in I} pr(K_i)$$

なので，

$$\begin{aligned} pr(\textstyle\bigcup_{i\in I} K_i)E_u \cap L(G) &= (\textstyle\bigcup_{i\in I} pr(K_i))\, E_u \cap L(G) \\ &= \textstyle\bigcup_{i\in I}(pr(K_i)E_u \cap L(G)) \\ &\subset \textstyle\bigcup_{i\in I} pr(K_i) \\ &= pr(\textstyle\bigcup_{i\in I} K_i). \end{aligned}$$

したがって，$\bigcup_{i\in I} K_i \in \mathcal{C}(K)$ である． ∎

この補題より**最大可制御部分言語** (supremal controllable sublanguage) の存在が言える．これはつぎのような言語である．

$$K^{\uparrow} = \bigcup_{M\in\mathcal{C}(K)} M \tag{2.14}$$

[3]集合 Kの部分集合のこと．

2.6 最大可制御部分言語を求めるアルゴリズム

$\mathcal{F}(E^*)$ により E上の言語の属を表す．つぎのような写像 $\Omega : \mathcal{F}(E^*) \to \mathcal{F}(E^*)$ を考える．

$$\begin{aligned} \Omega(M) &= K \cap sup\{\Sigma \subset E^* \mid \Sigma = pr(\Sigma),\ pr(\Sigma)E_u \cap L(G) \subset pr(M)\} \\ &= \{\sigma \in K \mid pr(\sigma)E_u \cap L(G) \subset pr(M)\} \end{aligned} \tag{2.15}$$

$\Omega(M)$ は G において非可制御な事象が生起しても $pr(M)$ に属するような事象列の中で Kの要素となるものの集合である．

補題 2.4 $K^\uparrow$は Ω の不動点である．すなわち，$\Omega(K^\uparrow) = K^\uparrow$.

証明 $K^\uparrow$は可制御なので,

$$pr(K^\uparrow) \subset \{\Sigma \subset E^* \mid \Sigma = pr(\Sigma),\ pr(\Sigma)E_u \cap L(G) \subset pr(K^\uparrow)\} = W$$

である．さらに，$K^\uparrow \subset K$より

$$K^\uparrow \subset K \cap pr(K^\uparrow) \subset K \cap W \subset \Omega(K^\uparrow)$$

である．逆に，$\sigma \in \Omega(K^\uparrow)$ とすると，$\sigma \in K, pr(\sigma)E_u \cap L(G) \subset pr(K^\uparrow)$ である．いま，$K' = K^\uparrow \cup \{\sigma\}$ を考えると，$K' \subset K$であり,

$$\begin{aligned} pr(K')E_u \cap L(G) &= (pr(K^\uparrow) \cup \{\sigma\})E_u \cap L(G) \\ &= (pr(K^\uparrow)E_u \cap L(G)) \cup (\{\sigma\}E_u \cap L(G)) \\ &\subset pr(K^\uparrow) \\ &\subset pr(K'). \end{aligned}$$

したがって，$pr(K')$ は可制御である．$K^\uparrow$が最大可制御部分言語であることから $K' = K^\uparrow$である．すなわち，$K^\uparrow \supset \Omega(K^\uparrow)$ である． ∎

補題 2.5 Ωの任意の不動点 Mに対して，$M \subset K^\uparrow$である（すなわち，Mは可制御）.

証明 定義より明らか. ∎

最大可制御部分言語 $K^\uparrow$がΩの不動点となることから，つぎのような $K^\uparrow$の計算法が考えられる．
(i) $K_0 := K$;
(ii) $K_{i+1} := \Omega(K_i)$ $(i = 1, 2, \cdots)$.

補題 2.6 数列 $\{K_i\}$ には極限 $K_\infty := \lim_{i\to\infty} K_i$ が存在し，$K^\uparrow \subset K_\infty$.

証明 $K_1 = \Omega(K_0) \subset K_0 = K$である．また，$\Omega$ の定義より，$K_i \subset K_{i-1}$ならば $\Omega(K_i) \subset \Omega(K_{i-1})$ が成り立つ．したがって，

$$K_0 \supset K_1 \supset K_2 \supset \cdots$$

であり，$K_\infty = \bigcap_i K_i$が存在する．また，$K^\uparrow$はΩ の不動点であり $K^\uparrow \subset K = K_0$なので，

$$K^\uparrow = \Omega(K^\uparrow) \subset K_i \ (i = 0, 1, 2, \cdots)$$

となり $K^\uparrow \subset K_\infty$が得られる． ∎

一般に逆の包含関係 $K_\infty \subset K^\uparrow$は成立しないが，$L(G)$ と Kが正規言語ならば $K_\infty = K^\uparrow$が成立する．いま，正規言語 L を受理する最小状態オートマトンの状態数を $\|M\|$ で表すと以下の結果が得られる．

定理 2.7 [38] Kと $L(G)$ を正規言語とおく．このとき数列 $\{K_i\}$ は有限回で $K^\uparrow$に収束する．$K^\uparrow$もまた正規言語であり，次式が成り立つ．

$$\|K^\uparrow\| \leq \|K\| \cdot \|L(G)\| + 1.$$

2.7 例題の解答

制御仕様を表現する言語を Kを生成するオートマトンは，プラント G から**制約** 1 を満たさない状態，および，**制約** 2 を満たさない状態（初期状態に到達できない状態）を取り除くことで得られる．

K自身は可制御とはならない．最大可制御部分言語はつぎのような言語である．初期状態では猫，ネズミとも移動が許される（c_3, m_5を開く）．もし猫が部屋 2 を出たならばネズミを部屋 4 に隔離する（c_6, m_5を閉じる）．そして猫は他の部屋を自由に移動できる．もしネズミが部屋 4 を出たならば猫を部屋 2 に隔離する．そしてネズミは他の部屋を自由に移動できる．これを実現するスーパバイザを Figure 2.5に示す．

問題 2.3 例題において，G および Kを表現する有限オートマトンを作れ．また，それに基づいて最大可制御言語を求め，それが Figure 2.5のスーパバイザにより実現されることを確認せよ．

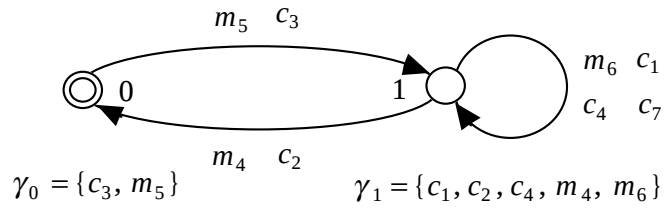

Figure 2.5: スーパバイザ.

2.8 章末問題

問題 2.4 2 台のプリンタ p_1, p_2を 3 台のコンピュータ c_1, c_2, c_3が使用する場合を考える．それぞれのコンピュータは 3 つの状態（空き状態→待ち状態→使用中→空き状態）を繰り返す．以下の制御目的を満足するスーパバイザを設計せよ．

1. 複数のコンピュータが同時に 1 つのプリンタを使用してはいけない（排他制御）.
2. c_1, c_2については先に使用要求を出したコンピュータからプリンタを使用できる.
3. c_3のジョブは最優先で実行される.

Chapter 3

離散事象システムの記述モデル

スーパバイザ制御の理論は離散事象システムに対する制御の枠組の 1 つを与えてはいるが，現実の対象をどのように表現し，制御仕様をどのような形式で記述するか，また，それらの下で制御器をどのように設計するか等，具体的な手順を与えるものではない．本章ではまず例題を示し，そのシステムを具体的な形で記述するための 3 つのモデルについて解説する．

複数のプロセスが同時進行的に相互作用を繰り返しながら処理を実行するようなシステムを**並行システム**という．本章で紹介する 3 つのモデルはいずれも並行システムに対する記述モデルとして提案されたものである．

3.1 例題：踏切問題

例題として取り上げるのは踏切制御システムである（Figure 3.1）[4, 44, 24]．1 本の線路上に踏切がある．列車の通過信号により踏切の上げ下げを制御する制御装置を設計せよという問題である．より詳細に問題を記述すると以下のようになる．

1. 踏切問題は列車，踏切制御装置，遮断機の 3 つの部分から構成される．
2. 列車の通過信号としては，
   - 踏切に接近
   - 踏切を通過

   の 2 つがある．

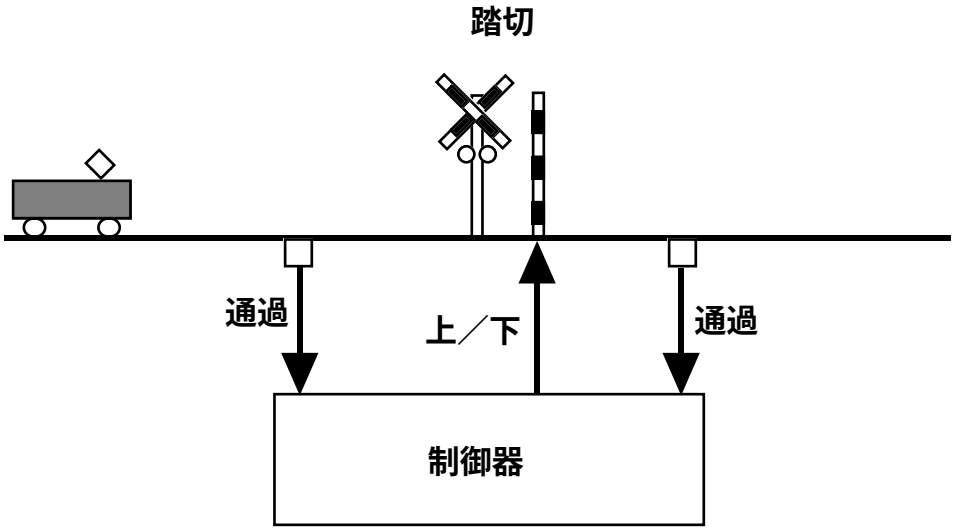

Figure 3.1: 踏切問題.

3. 遮断機は「上」,「下」の 2 つの状態がある．したがって，事象としては「下→上」,「上→下」の 2 つがある．

3.2 ペトリネット

ペトリネット (Petri net) は並行的･非同期的･分散的なシステムを表現するための数学的モデルであり，1960 年代に C.A.Petri により提案された [3, 26, 32, 34, 37, 40, 41].

3.2.1 プレース／トランジションネットの定義

ペトリネットは次の 3 つの側面をもつモデルである．

1. 数学的なモデル
2. 図的なモデル
3. 実行可能なモデル

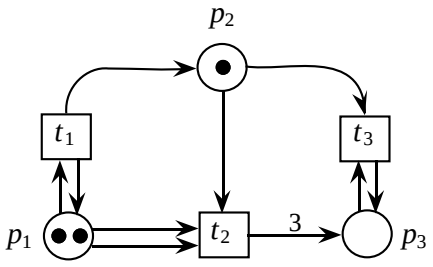

Figure 3.2: プレース／トランジションネット.

すなわち，記述したモデルがそのままシミュレーションにより実行でき，その過程が図的に表現できる点で，ソフトウェア・ツール化に適したモデルである.

ペトリネットの基本となるのは**プレース／トランジションネット** (place/transition net) である．プレース／トランジションネットとは 4 項組 $PN = (P, T, A, M_0)$ である．ここで，P は**プレース** (place) の有限集合，Tは**トランジション** (transition) の有限集合 $(P \cap T = \emptyset)$，$A : (P \times T) \cup (T \times P) \to \mathbb{N}$ はプレースとトランジションの接続を表す写像 ，$M_0 : P \to \mathbb{N}$ は**初期マーキング** (initial marking) である.

プレース／トランジションネットはプレースとトランジションという 2 種類の節点をもつ有向 2 部グラフとして見ることができ，通常，Figure 3.2のように描く．プレースは “◯” で，トランジションは “□” あるいは “|” で，写像 A は**アーク** (arc)“→” により表す．$A(x, y) = n \geq 1$ のとき n 本のアーク（または本数を枝の上に書いた 1 本のアーク）を節点 x から節点 yへ向けて描く．また，初期マーキングは各プレース p 内に $M_0(p)$ 個の “•” （これを**トークン** (token) と呼ぶ）を描くことにより表す. プレースにトークンが存在するとき，そのプレースは**マークされている** (marked) という.

各プレースに存在するトークンの個数を表す写像 $M : P \to \mathbb{N}$ を**マーキング** (marking) と呼ぶ．マーキングはオートマトンにおける状態に対応し，初期値 M_0からトランジションの**発火** (firing) により推移していく．トランジション t はマーキング Mにおいてつぎの条件を満たすときに**発火可能** (enabled, fireable) であるという.

$$\forall p \in P : M(p) \geq A(p, t) \tag{3.1}$$

すなわち，トランジション t は各入力プレース p 上にその間のアークの重みである $A(p, t)$ 個以上のトークンが存在するとき発火可能になる．発火可能なトランジションは発火させることができる．トランジション t の発火により次のマーキング M'に変化する．これを $M \xrightarrow{t} M'$と書く.

$$M'(p) = M(p) + A(t, p) - A(p, t) \quad (p \in P) \tag{3.2}$$

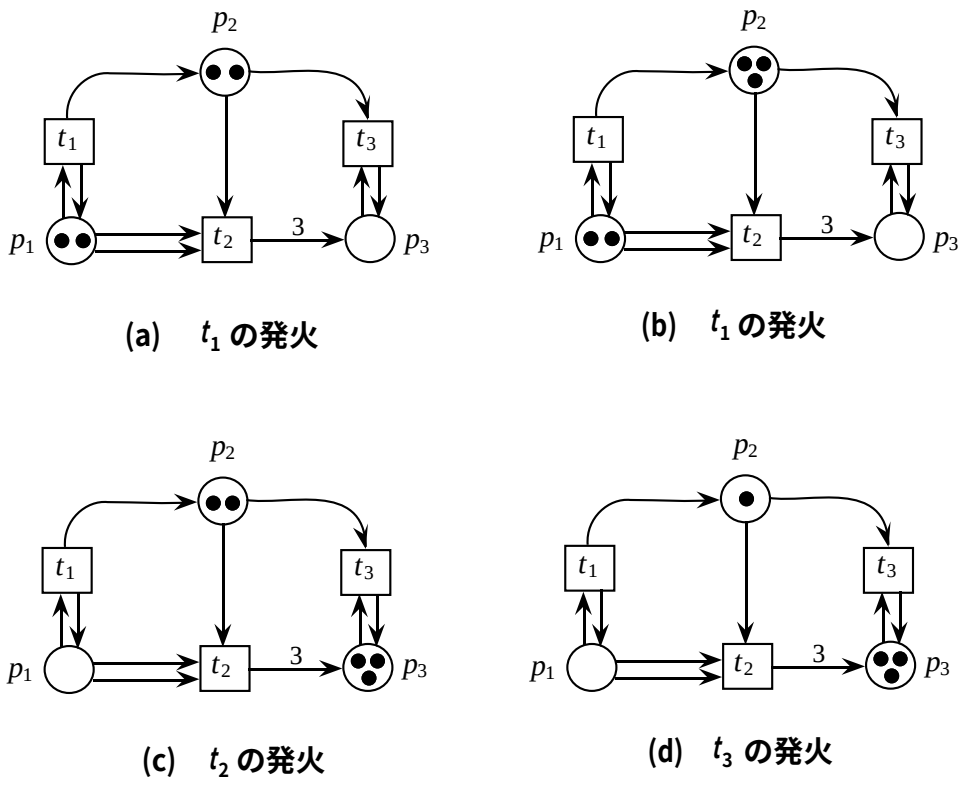

Figure 3.3: トランジションの発火.

トランジションの発火の様子を Figure 3.3に示す.

トランジションを並べた系列を**発火系列** (firing sequence) という．空系列ϵを含む有限の長さの発火系列全体の集合を T^*により表す．発火系列$\sigma = t_1 t_2 \cdots t_n \in T^*$について，$M \stackrel{t_1}{\to} M_1 \stackrel{t_2}{\to} M_2 \cdots M_{n-1} \stackrel{t_n}{\to} M'$が成り立つとき，マーキング Mにおいて σ は発火可能であるといい $M \stackrel{\sigma}{\to} M'$と書く.

1 つのプレースに入り得るトークン数は必ずしも有限ではない．いまマーキング $M \in R(PN)$ において発火系列σが発火可能であり，発火の結果 $M' \geq M \wedge M' \neq M$ であるようなマーキング M'に推移する場合を考えると，発火系列 σの発火はマーキング Mから任意の回数だけ繰り返すことができる．その結果，無限に多くのトークンが生成される.

初期マーキングからどのような発火系列を実行しても 1 つのプレースに入り得るトークン数が有限であるようなペトリネットは**有界** (bounded) であるという．特に，各プレースのトークン数がたかだか 1 であるようなペトリネットは**セーフ** (safe) であるという．また，人為的に 1 つのプレースに入り得るトークン数の上限（これ

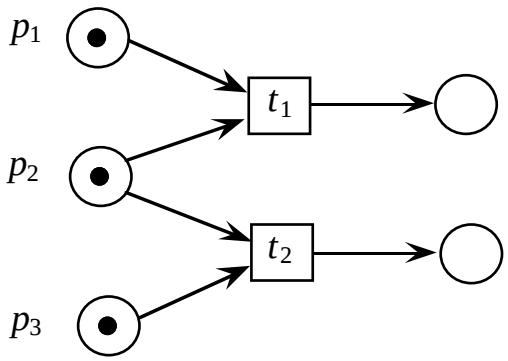

Figure 3.4: 発火の競合.

を容量という）を設けたペトリネットを**有限容量ペトリネット** (finite capacity Petri net) という.

問題 3.1 プレーストランジションネットがセーフであるための条件について考察せよ.

3.2.2 発火の競合

トランジションの発火で注意しておきたいのは，発火可能なトランジションが必ず発火するわけではないことである．定義されるのはトランジションが発火可能となるための条件，および，発火したらマーキングがどう変化するかということだけである．もし「発火可能なトランジションは必ず発火する」という性質を加えると，Figure 3.4に示す**競合** (conflict) の問題が生じる．トランジション t_1と t_2はともに発火可能であるが，もし t_1を発火させると t_2の発火可能性は失われる．逆に，t_2を発火させると t_1の発火可能性は失われる．したがって，t_1と t_2の両方を発火させることはできない.

ペトリネットによるモデル化で特徴的なのは状態の分散性と事象生起の局所性であるといえる．システムの状態は局所的に定義される状態の集まりであるマーキングによって記述され，事象は状態があらかじめ定められた条件を満たしたときに生起可能となる．この条件はシステム全体としての状態ではなく，局所的な状態に依存するものとして定義される．したがって，生起条件が独立な事象はその生起も独立に行うことができる．これにより，並行性・非同期性の表現が可能になっている.

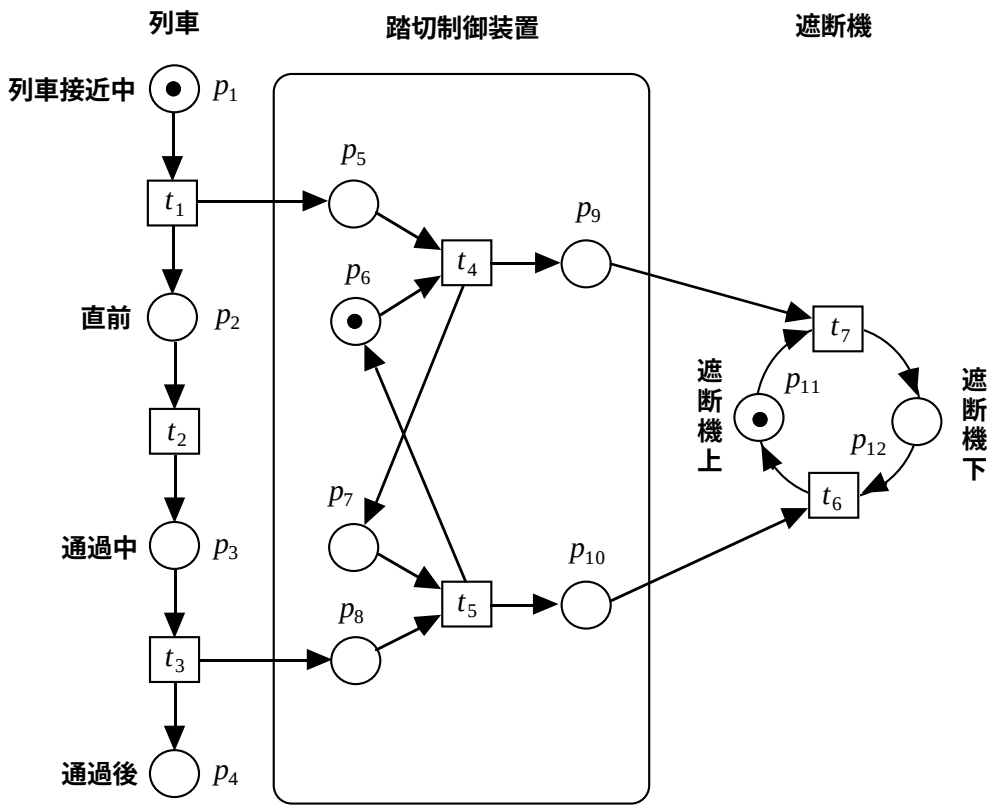

Figure 3.5: 踏切問題のペトリネットによるモデル化.

3.2.3 踏切問題のモデル化

例題をペトリネットによりモデル化するとFigure 3.5のようになる．実際に設計を行なうのは踏切制御システムの部分であるが，システム全体として正しく動作するかを調べるために外部環境（ここでは列車と遮断機）を含めてモデル化している．

踏切制御システムは以下のような動作を行なう．

1. 列車が接近中．遮断機は上.
2. 列車が踏切区間内に進入（t_1の発火）.
3. 踏切制御システムは遮断機を下ろす指令を出す（t_4の発火）.
4. 遮断機は下（t_7の発火）.
5. 列車が踏切を通過（t_3の発火）.

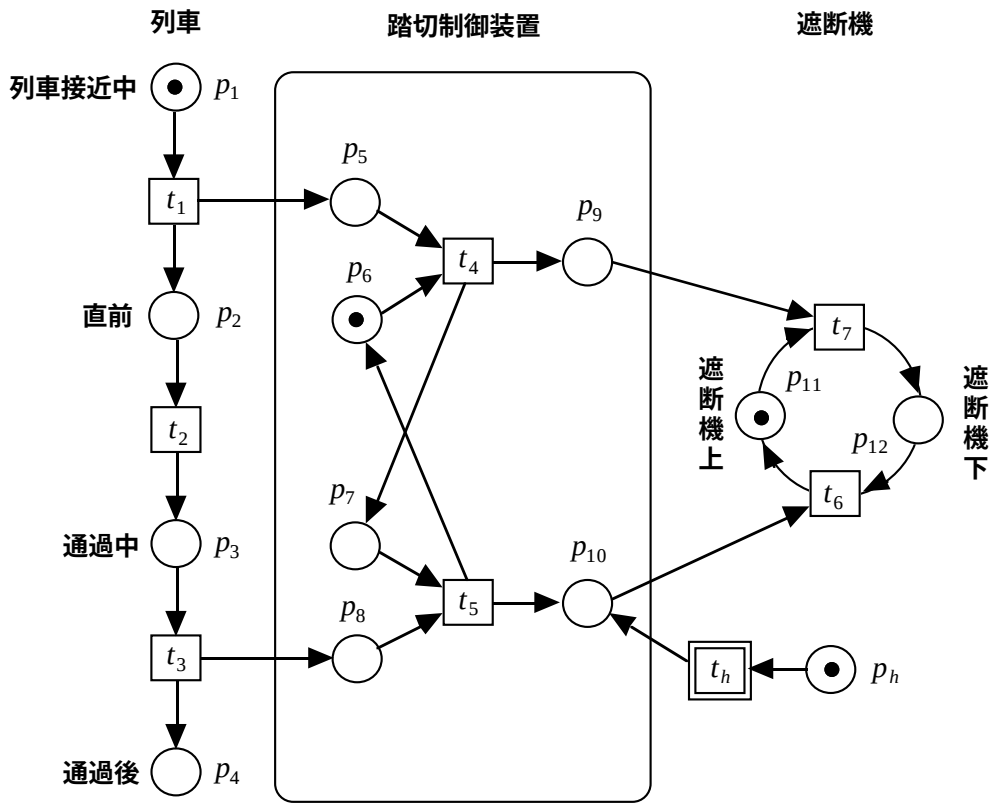

Figure 3.6: 障害を含む踏切問題のペトリネットによるモデル化.

6. 踏切制御システムは遮断機を上げる指令を出す（t_5の発火）.

7. 遮断機は上（t_6の発火）.

問題 3.2 Figure 3.5の踏切制御装置は不完全である．どのような点が不完全かを分析せよ．また，これを修正するにはどのようにすればよいか.

問題 3.3 現実にはモデル化したシステムのある部分の障害や故障も考慮した安全設計が求められる．Figure 3.6は障害の発生に対応するトランジション t_hを追加したモデルである.

(i) t_hはどのような障害の発生に対応しているか.

(ii) 障害に対しても安全であるように踏切制御装置を変更せよ.

3.2.4 到達可能性グラフ

ペトリネットの定義にはそれがどのような動作を行なうかは入っていない．一般にあるモデルが与えられたとき，それがどのような動作をするか[1]は別の数学的構造への写像として与えられる．ペトリネットの場合，その意味は**到達可能性グラフ** (reachability graph) により与えられる．

プレース／トランジションネット $PN = (P, T, A, M_0)$ において，初期マーキング M_0から到達可能なマーキングの集合

$$R(PN) = \{M \in \mathbb{N}^P \mid \exists \sigma \in T^* : M_0 \stackrel{\sigma}{\rightarrow} M\} \tag{3.3}$$

を**到達可能性集合** (reachability set) という．

4 項組 $LTS = (S, E, \Delta, s_0)$ を**ラベル付遷移システム** (labeled transition system) という．ここで，Sは状態の集合，Eは事象ラベルの集合，$\delta \subset S \times E \times S$は遷移関係，$s_0 \in S$は初期状態である．

任意の $(s, e_1, s_1), (s, e_2, s_2) \in \Delta$について $e_1 = e_2$ならば $s_1 = s_2$であるとき，LTSは**決定性** (deterministic) であるという．また，そうでないとき，**非決定性** (nondeterministic) であるという．プレース／トランジションネット PNの動的な挙動は決定性ラベル付遷移システム $LTS_{PN} = (R(PN), T, \delta_{PN}, M_0)$ により記述される．ここで，遷移関係Δ_{PN}は次のように与えられる．

$$\Delta_{PN} = \{(M, t, M') | M, M' \in R(PN), M \stackrel{t}{\rightarrow} M'\} \tag{3.4}$$

LTS_{PN}は各マーキングを節点とし，遷移可能なマーキングどうしを有向枝で結び，枝上に発火させるトランジションを描くことにより，有向グラフとして表現できる．これを PNの到達可能性グラフと呼ぶ．Figure 3.7に示したのは Figure 3.5のネットの到達可能性グラフである．

問題 3.4 Figure 3.2のペトリネットの到達可能性グラフを描け．

問題 3.5 Figure 3.7の到達可能グラフの状態 2 において，2 つの発火系列 t_2t_4, t_4t_2 がともに発火可能である．これはどのような状況に対応しているか．

3.3 状態遷移モデル

状態遷移モデルとは，対象システムの取り得る状態と状態間の遷移で対象システムの挙動を表現したものである．直観的に理解しやすくかつ自然なモデル化である

[1] これをモデルの意味 (semantics) という．

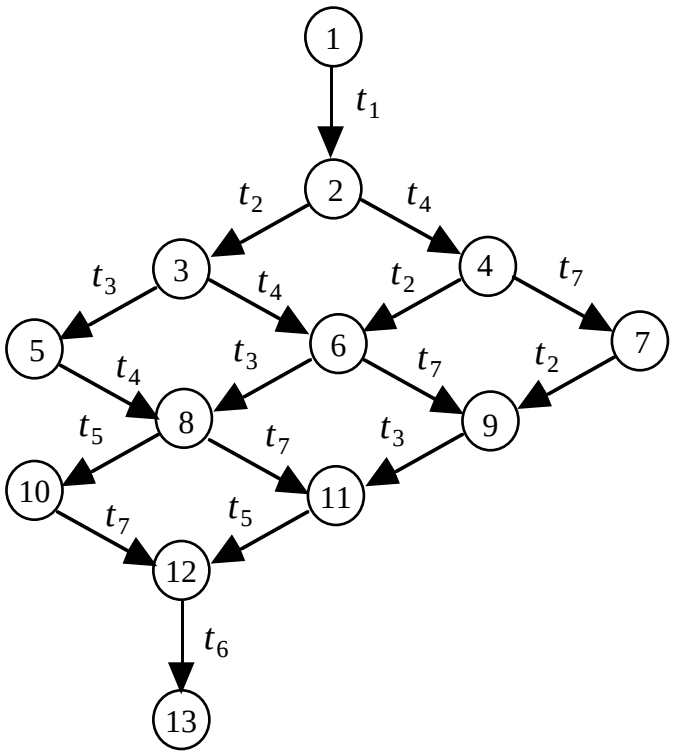

Figure 3.7: 到達可能性グラフ.

ため，システム設計において非常によく用いられている．状態遷移モデルの定式化は 1950 年代の有限状態機械の Moore モデルや Mealy モデルに始まるが，その後状態遷移モデルの定式化には多くの提案があった．

3.3.1 ラベル付遷移システムの同期合成

通常，状態遷移モデルで並行システムを表現する場合，各プロセスの挙動を 1 つの状態遷移グラフで表現し，その遷移条件としてプロセス間の通信を記述する．

いま，2 つのラベル付遷移システム $LTS_i = (S_i, E_i, \Delta_i, s_{0i})$ $(i = 1, 2)$ があったとき，その同期合成 (synchronous composition) はつぎのようなラベル付遷移システム $LTS = (S, E, \Delta, s_0)$ として定義される．これを $LTS_1 || LTS_2$で表す．

1. $E = E_1 \cup E_2$.
2. $s_0 = (s_{01}, s_{02})$.
3. $S \subset S_1 \times S_2$および$\Delta \subset S_1 \times E \times S_2$は以下を満たす最小の集合である．
   - $s_0 \in S$.
   - もし $(s_1, s_2) \in S,\ e \in E_1 - E_2,\ (s_1, e, s_1') \in \Delta_1$ならば，$(s_1', s_2) \in S,\ ((s_1, s_2), e, (s_1', s_2)) \in \Delta$.
   - もし $(s_1, s_2) \in S,\ e \in E_2 - E_1,\ (s_2, e, s_2') \in \Delta_2$ならば，$(s_1, s_2') \in S,\ ((s_1, s_2), e, (s_1, s_2')) \in \Delta$.
   - もし $(s_1, s_2) \in S,\ e \in E_1 \cap E_2,\ (s_1, e, s_1') \in \Delta_1,\ (s_2, e, s_2') \in \Delta_2$ならば，$(s_1', s_2') \in S,\ ((s_1, s_2), e, (s_1', s_2')) \in \Delta$.

同期合成とは，同じ名前の事象は同期して生起させるような合成である．3 個以上のラベル付遷移システムの合成も同様な形で定義できる．

3.3.2 踏切問題のモデル化

Figure 3.8に踏切問題のモデル化を示す．**踏切問題**は 3 つのサブシステム**列車**，**踏切制御装置**，**遮断機**の同期合成で表現される．すなわち，

$$\textbf{踏切問題} = \textbf{列車} \mid\mid \textbf{踏切制御装置} \mid\mid \textbf{遮断機}$$

さらに，**踏切制御装置**は 5 つのラベル付遷移システム C_i $(i = 1, \cdots, 5)$ の同期合成で表現される．

$$\textbf{踏切制御装置} = C_1 || C_2 || C_3 || C_4 || C_5$$

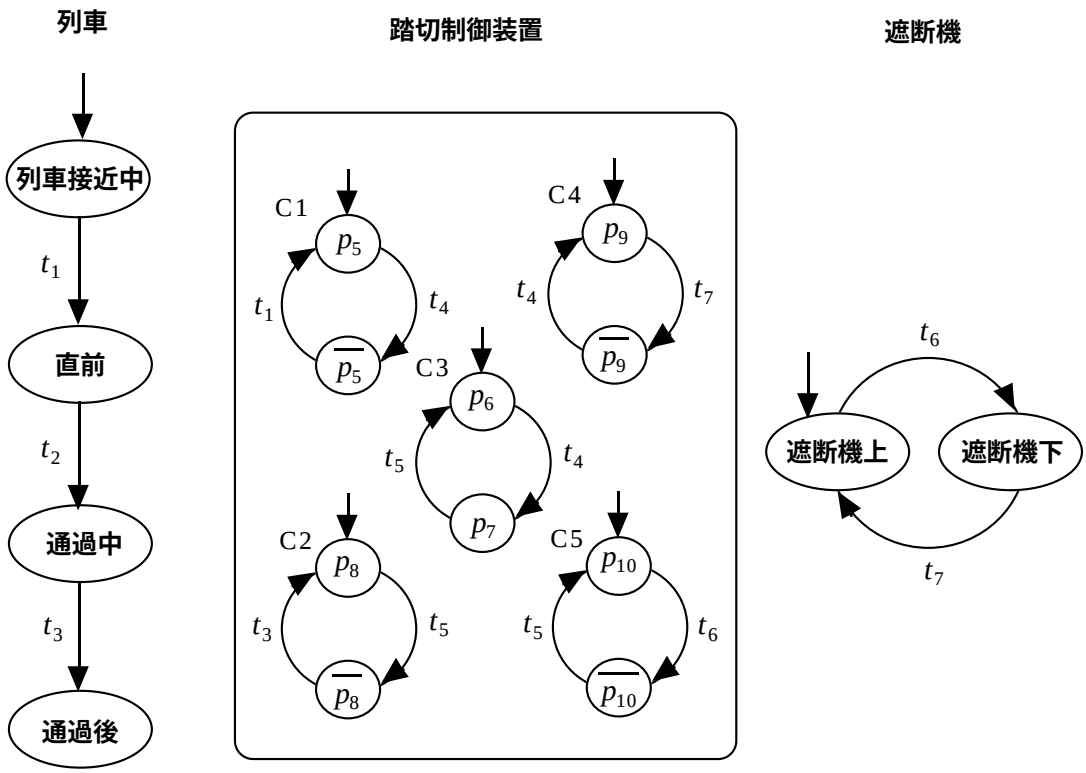

Figure 3.8: 踏切問題の状態遷移モデル.

3.3.3 Statechart

Statechart は並行ソフトウェア設計支援ツールである STATEMATE[17] で使われている状態遷移モデルである．Statechart では状態遷移のラベル（遷移条件および遷移に伴う動作）をつぎの形式で記述する．

event[condition]/action

Figure 3.9に踏切問題の Statechart による記述例を示す．ここで破線で区切られた 3 つの状態遷移モデルは，3 つのプロセスが並行に動くことを表している．各状態遷移モデルは事象（メッセージ）t_1, t_2, t_3, t_4により非同期型の通信を行なっている．たとえば，プロセス**列車**が「接近中」から「直前」に遷移すると，動作として事象 t_1が発行される．事象 t_1によりプロセス**踏切制御装置**の $p_6 \rightarrow p_7$の遷移条件が満たされ，遷移が起こる．また，この遷移に伴う動作として事象 t_4を発行する．それにより，プロセス**遮断機**の状態が「遮断機上」から「遮断機下」へ遷移する．

この他に Statechart では階層的な状態の表現も可能である．Statechart は汎用の状態遷移モデルとしては最も洗練されており，多くの設計手法や CASE ツールで採用されている．

問題 3.6 問題 3.2, 3.3の設計を状態遷移モデルで表現せよ．

3.4 プロセス代数

並列システムを記述するためのモデルとしては，ペトリネットの他に CCS[30] や CSP[22]，ACP[6] などの代数的モデルがよく知られており，これらのモデルを総称して**プロセス代数** (process algebra) と呼ぶ．ペトリネットがオートマトンをベースにしたモデルであるのに対し，プロセス代数はラムダ計算およびプログラミングをベースにしている．ペトリネットが並行性を半順序関係として陽に記述するのに対し，プロセス代数では**インターリービング** (interleaving) を用いて非決定性として記述する．また，プロセス代数は動作を主体に定義されており，システムの状態を直接表現する手段をもたない．

3.4.1 プロセス代数の定義

ここでは CCS 流の定義を与える．まず，以下の集合を定義する．

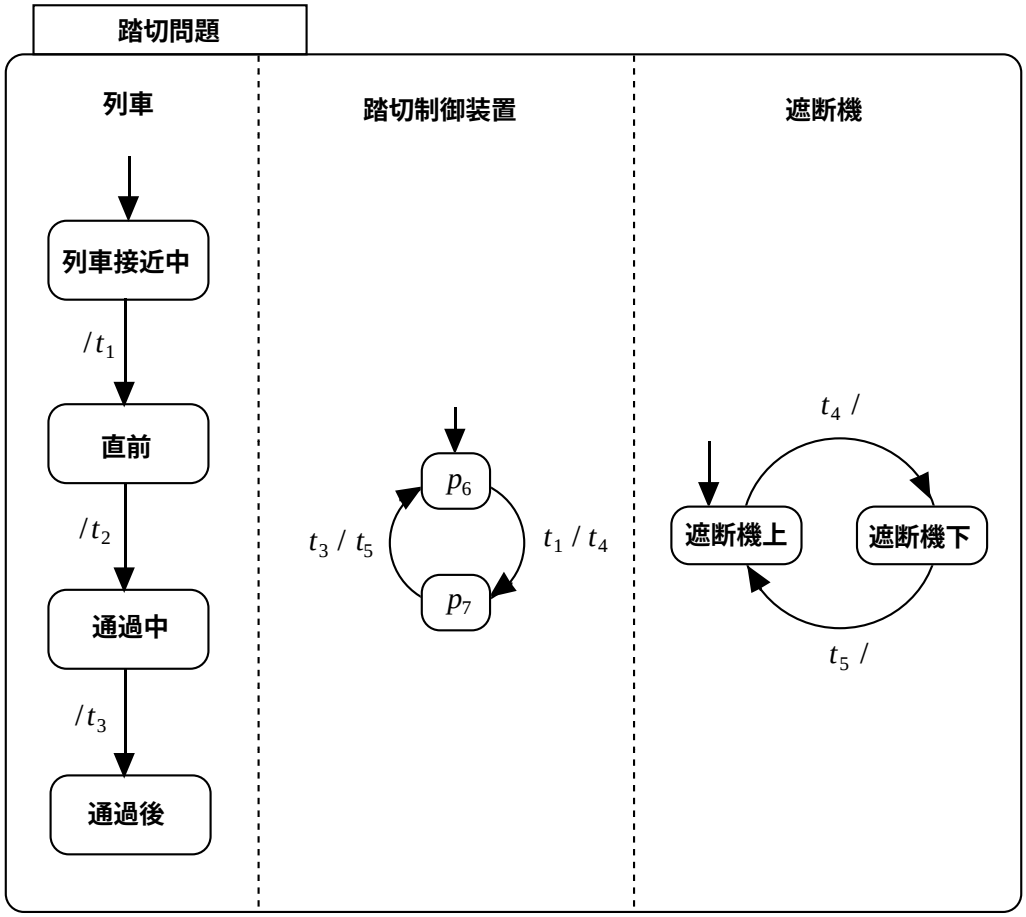

Figure 3.9: 踏切問題の Statechart によるモデル化.

- $\mathcal{A}$：名前の集合.
- $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$：余名の集合．また，$a \in \mathcal{A}$ に対し$\overline{\overline{a}} = a$.
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$：ラベルの集合.
- $Act = \mathcal{L} \cup \{\tau\}$：動作の集合．ここで$\tau$は内部動作とよばれる観測されない特殊な動作.
- $\mathcal{K}$：プロセス定数の集合.
- $\mathcal{X}$：プロセス変数の集合.
- $f : \mathcal{L} \to \mathcal{L}$：リラベリング関数．ただし，$f(\overline{l}) = \overline{f}(l)$ を満たす.

このとき，CCS の動作式の集合 $\mathcal{E}$は以下の式からなる集合として定義される.

- $x \in \mathcal{X}$：プロセス変数.
- $A \in \mathcal{K}$：プロセス定数.
- $\alpha.E$ $(\alpha \in Act,\ E \in \mathcal{E})$：プレフィックス.
- $\sum_{i \in I} E_i$ $(\forall i \in I : E_i \in \mathcal{E})$：和.（2 項の場合は $E_1 + E_2$のように書く）.
- $E|F$ $(E, F \in \mathcal{E})$：合成.
- $E[f]$ $(E \in \mathcal{E},\ f$はラベリング関数$)$：リラベリング.
- $E\backslash L$ $(E \in \mathcal{E},\ L \subset \mathcal{L})$：制限.

プロセス変数を含まない動作式をプロセスとよび，その集合を $\mathcal{P}$で表す．プロセス定数は定義式 $\mathcal{A} \stackrel{def}{=} P$ $(P \in \mathcal{P})$ により意味を与えられるプロセスである．また，停止プロセスは $\mathbf{0} = \sum_{i \in \emptyset} E_i$により与えられる.

3.4.2 プロセス代数の意味

CCS の意味はつぎのラベル付き遷移システムにより与えられる.

$$LTS_{CCS} = (\mathcal{E}, Act, \{\stackrel{\alpha}{\to} \mid \alpha \in Act\}, \varepsilon_0) \tag{3.5}$$

$E \stackrel{\alpha}{\to} E'$は動作式 Eが動作 α を実行することができ，その実行後は動作式 E'のようにふるまうことを表すものとする．動作式の遷移関係$\stackrel{\alpha}{\to}$は以下の推論規則を満たす最小の関係である．（横棒の上が仮定，下が結果，右横が条件を表す．）

$$\mathbf{Act}\frac{}{\alpha.E \stackrel{\alpha}{\to} E} \tag{3.6}$$

$$\mathbf{Sum_j}\frac{E_j \stackrel{\alpha}{\to} E'_j}{\sum_{i\in I} E_i \stackrel{\alpha}{\to} E'_j}(j \in I) \tag{3.7}$$

$$\mathbf{Com_1}\frac{E \stackrel{\alpha}{\to} E'}{E|F \stackrel{\alpha}{\to} E'|F} \tag{3.8}$$

$$\mathbf{Com_2}\frac{F \stackrel{\alpha}{\to} F'}{E|F \stackrel{\alpha}{\to} E'|F} \tag{3.9}$$

$$\mathbf{Com_3}\frac{E \stackrel{l}{\to} E' \quad F \stackrel{\overline{l}}{\to} F'}{E|F \stackrel{\tau}{\to} E'|F} \tag{3.10}$$

$$\mathbf{Res}\frac{E \stackrel{\alpha}{\to} E'}{E\backslash L \stackrel{\alpha}{\to} E'\backslash L}(\alpha, \overline{\alpha} \notin L) \tag{3.11}$$

$$\mathbf{Rel}\frac{E \stackrel{\alpha}{\to} E'}{E[f] \stackrel{f(\alpha)}{\to} E'[f]} \tag{3.12}$$

$$\mathbf{Con}\frac{P \stackrel{\alpha}{\to} P'}{A \stackrel{\alpha}{\to} P'}(A \stackrel{def}{=} P) \tag{3.13}$$

- 規則 **ACT** はプレフィックスに関する規則である．動作式 $\alpha.E$は動作 α の実行後に動作式 Eのようにふるまう．
- 規則 $\mathbf{Sum_j}$は複数のプロセスの中から動作によって 1 つのプロセスを選択することを意味する．たとえば，

  $$a.E_1 + b.E_2 + c.E_3 \stackrel{b}{\to} E_2$$

  である．
- 規則 $\mathbf{Com_{1,2}}$は 2 つのプロセス Eと Fが並行・独立に動作できることを意味する．
- 規則 $\mathbf{Com_3}$は Eと Fの間の通信を表す．通信後は内部動作 τ に変化する（1 対 1 通信）．
- 規則 **Res** は制限に関する規則である．$E\backslash L$ は動作式 Eにおいて L の動作を内部化する（すなわち，外部から観測できなくする）．

- 規則 **Rel** は動作の名前を変更するときに使われる．たとえば，

$$(a.E)[b/a] \xrightarrow{b} E[b/a].$$

- 再帰プロセスはプロセス定数によって記述される．たとえば，

$$E \stackrel{def}{=} a.E + b.\mathbf{0}$$

は無限に動作 a を起こすことができ，動作 b を起こすと停止する．規則 **Con** により，プロセス定数 Eは動作 a を起こすと再びプロセス定数 Eに戻ることが導かれる．すなわち，

$$\mathbf{Act}\frac{}{a.E \xrightarrow{a} E}$$

$$\mathbf{Sum_j}\frac{|}{a.E + b.\mathbf{0} \xrightarrow{a} E}$$

$$\mathbf{Con}\frac{|}{E \xrightarrow{a} E}$$

3.4.3 踏切問題のモデル化

CCS により踏切問題をモデル化するとつぎのようになる [24].

- 外部から観測される動作は，つぎの 4 つである．
  - *approach*: 列車の接近.
  - *crossing*：踏切通過中.
  - *up*：遮断機上げる.
  - *down*：遮断機下げる.
- 列車は踏切に接近したときに信号 sig_1，踏切を通過し終ったときに信号 sig_2を踏切制御装置に送信する．この列車のふるまいはつぎのように表現できる．

$$TRAIN \stackrel{def}{=} approach.\overline{sig_1}.crossing.\overline{sig_2}.\mathbf{0}$$

- 遮断機は踏切制御信号から信号 sig_3を受信すると下がり，信号 sig_4を受信すると上がる．ただし，信号 sig_3と sig_4は交互にしか受信できないとする．この遮断機のふるまいはつぎのように記述される．

$$GATE \stackrel{def}{=} sig_3.down.sig_4.up.GATE$$

- 踏切制御装置 $CTRL$ の内部には，踏切を下げる動作と上げる動作を制御する 2 つの制御装置があり，それらは遮断機の状態を保存している 1 つのスイッチ SW を共有している．スイッチのオン・オフはそれぞれ遮断機の下げ・上げに対応しており，オン・オフは交互にのみ行なえる．踏切を下げるための制御装置 $DOWN$は列車から信号 sig_1を受信するとスイッチをオンにして信号 sig_3 を遮断機に送信する．踏切を上げるための制御装置 UPも同様である．

$$Down \stackrel{def}{=} sig_1.\overline{on}.\overline{sig_3}.DOWN$$
$$UP \stackrel{def}{=} sig_2.\overline{off}.\overline{sig_4}.UP$$
$$SW \stackrel{def}{=} on.off.SW$$
$$CTRL \stackrel{def}{=} (DOWN|UP|SW)\backslash\{on, off\}$$

スイッチのオン・オフは制御装置の内部動作である．

- 最後に全体のふるまいはつぎのように記述できる．

$$TGC \stackrel{def}{=} (TRAIN|GATE|CTRL)\backslash L$$
$$L = \{sig_1, sig_2, sig_3, sig_4\}$$

問題 3.7 TGCのラベル付き遷移システムを作れ．

問題 3.8 問題 3.2, 3.3の設計をプロセス代数で表現せよ．

3.5 章末問題

問題 3.9 踏切問題に対する踏切制御装置をスーパバイザ制御の考えで設計せよ．

問題 3.10 複線にした踏切問題に対する踏切制御装置を設計せよ．

問題 3.11 3 つのモデル（ペトリネット，状態遷移モデル，プロセス代数）を比較せよ．

Chapter 4

並行システムの効率的解析

前章までに述べた離散事象システムの理論および記述モデルを実用的な規模の問題に適用する場合にはいくつかの困難な問題が存在する．それは離散事象システムでは解析すべき多くの問題が組合せ問題となり，計算の複雑さが NP-完全，あるいは NP-hard の問題となることである．特に，システムが並行動作を許す場合，これによりシステムの取り得る状態数が指数関数的に増加する問題（**状態空間爆発**）が生じる．並行性，分散性をもつ離散事象システムでは，サブシステムの状態の組合せで全体の状態空間が生成されてしまうからである．

離散事象システム制御の理論は問題を定式化するための枠組を与えるものであり，問題自体を解く効率的な方法は別に求める必要がある．離散事象システムの理論を現実の大規模な対象に適用するには，このような困難な問題に対しても実用的に有効なアルゴリズムを構築しなければならない．

本章ではモデルを用いて設計されたシステムが正しく動作するかどうかを解析する方法について，ペトリネットを例にして説明する．以下に紹介する手法はいずれも適用できるペトリネットのクラスに制限がないという特徴をもっており，インプリメントも容易である．逆にクラスに対する制限がないことで，ペトリネット特有の性質といったものは使っておらず，他の並行システム記述のためのモデルにも共通に適用可能な手法となっている．

4.1 効率的解析のための方策

ペトリネットの解析に関する理論研究としては，過去においては構造的に限定されたクラスにおける議論が比較的多く行われていた．これらの研究は並行システムの構造的性質を議論するのには有用であったが，ペトリネットをシステムの記述言

語として見た場合，これらの理論的成果が必ずしも実際の問題に適用できるわけではない．より一般的な解析方法，すなわち，以下のような解析方法の研究が重要であると考えられる．

- 対象とするクラスを限定しない．
- 最悪の場合の計算量で手法を評価するのではなく，実際の問題を解析したときの平均的計算量で評価する．

ペトリネットの解析問題をこのような立場で見たとき，解析における問題点はペトリネットに限らず，並行的な動作を許すシステムが共通してもつ問題点であると言える．

たとえば並行動作を許すシステムでは，取り得る状態数がモデルのサイズの指数関数オーダーで増加する状態空間爆発が大きな問題になっているが，ペトリネットの解析においても同様である．逆に言えば，並行システムの解析をペトリネットで行う利点は何かということについて考えなければならない．

これに対してはペトリネットの以下の特徴が解析において有用であると考えられる．

1. ペトリネットは並行システムのもつ基本的性質（並行，同期，非同期，競合）を表現することができる最も単純なモデルの 1 つである．
2. 対象となるシステム全体を 1 つのグラフとして表現するため，グラフ的な構造に基づいた性質の議論が可能である．
3. 構造と状態を明確に区別して記述する．

ペトリネットの解析問題には様々なものが存在するが，それらの多くはその状態空間の探索により解くことができる．例えば，代表的な問題である**到達可能性問題** (reachability problem)（与えられたマーキングが初期マーキングから到達できるか）は状態空間を表すグラフの探索問題として解くことができる．しかしながら，状態空間爆発のため状態空間を直接操作する方法は計算量が膨大なものになってしまう．この問題に対処するための方策として以下のものが提案されている．これらの手法のいくつかはペトリネットだけではなく，並行システムに一般的に適用可能な方法である．

1. **状態空間の縮約**：解析すべき問題に関係する部分だけからなる部分的な状態空間を構成する（**半順序法** [46, 12, 48, 47, 19, 36]）．

2. **状態空間の表現方法の工夫**：グラフとしての対称性を利用する [15]．階層的な表現を用いる [33]．繰り返し部分を折り畳んで表現する [8, 21]．半順序関係を直接表現する (**アンフォールディング** [28, 11, 14, 25])．

3. **ネットレベルでの縮約**：解析すべき性質に関して等価な縮約したペトリネットを作る [7, 31]．

4. **状態空間の表現方法の変換**：BDD の利用 [35, 43, 20]．

5. **アルゴリズムとしての工夫**：状態空間の生成と条件の検証を並行して行う（on-the-fly モデルチェッキング）[47, 36]．

6. **発見的アルゴリズムの利用**：遺伝アルゴリズムによる到達可能性問題の解法 [42]．

状態空間の生成は解析のための一手法であり，それ以外の方法も存在する．たとえば，ペトリネットのいくつかのサブクラスでは状態空間を生成することなく到達可能性を判定する方法も提案されている．しかしながら，そのような方法の多くは対象とするペトリネットのクラスを制限したものが多い．

以下では，比較的新しい解析法である，半順序法，BDD を用いた解析，アンフォールディングの 3 つの手法について述べる．

4.2 半順序法

4.2.1 インターリービング意味論

半順序法 (partial order method) は実行可能な事象の集合上に定義される独立／従属の関係に注目し，互いに独立な事象の生起順序を変えてもそれ以降の状態への到達可能性，および，各事象の実行可能性には影響を与えないという性質を用いている．互いに独立な事象の生起順序を制限することにより，**インターリービング** (interleaving) により生成される状態数の膨張を抑えることができる．

いま，2 つの事象系列 abc と xyが全く独立に生起したとすると，これらの系列は外部からはどのように観測されるであろうか? 確実なのは a は b の前，b は c の前，x は yの前に生起することだけである．a と x のどちらが先に生起するかは観測者に依存して決まる．したがって，観測される可能性のある系列は abc の各文字の間（a の前，c の後も含む）に xyを**はさみこんだ**系列 $xyabc, xaybc, xabyc, xabcy, axybc, \cdots$ である．これらの系列を abc, xyのインターリービングという．

並行性という概念をどうとらえるかについてはいくつかの立場がある．上記の意味モデルでは，事象はすべてラベル付遷移システム上での系列（これは系列に含まれる事象集合に全順序が付いたものである）で表現される．並行性は,「並行な動作はそれらの順序を任意に入れ換えた系列がすべて出現する」ことで表現される．たとえば，事象 a と事象 b が並行に生起することは，系列 ab および系列 ba が共に生起可能なことで表現される．

4.2.2 半順序法の基本的な考え方

半順序法では，ペトリネット PN，および，検証すべき LTS_{PN} の性質 f が与えられたとき，性質 f を調べることが可能な縮約したラベル付き遷移システム（**縮約状態空間**）を求める．このとき，縮約状態空間は PN の並行動作に対するインターリービングの一部分だけを含むように構成される．

通常扱われる多くの性質は，インターリービングのすべてを調べなくても検証可能である．半順序法は，通常の状態空間において出現するインターリービングを制限することにより，より少ない計算量でシステムの動作検証を行おうという方法である．

具体的なアルゴリズムを表 4.1に示す．半順序法は選択的探索であり以下のような形に書ける．得られた探索木により縮約状態空間が作られる．ここで，$en(M)$ はマーキング M において発火可能なトランジションの集合であり，$succ_t(M)$ はマーキング M からのトランジション t による遷移先を表す．

各マーキング M において，選択されたトランジションの集合 $select(M) \subset T$ に属するもののみ発火の対象とする．$succ_t(M)$ がすでにスタックに入っているかどうかをチェックし，入っている場合は縮約を行わない（$working := en(M)$）のは，アルゴリズムが独立なループのみを探索して終了してしまうのを防ぐためである (ignoring の問題 [46, 48])．選択集合 $select(M)$ をどのように決定するかが問題になる．

4.2.3 トランジションの独立性・従属性

以下の条件を満たすとき，2 つのトランジション $t_1, t_2\,(t_1 \neq t_2)$ は**独立** (independent) であるという．

$$M \xrightarrow{t_1} \wedge\, M \xrightarrow{t_2} \Rightarrow\ \exists M' : M \xrightarrow{t_1 t_2} M' \ \wedge\ M \xrightarrow{t_2 t_1} M' \tag{4.1}$$

独立性はトランジションの集合上の反射的，かつ，対称的な関係である．独立性を I で表す．関係 $D = T \times T - I$ を**従属性** (dependence) という．従属性も対称的である．トランジション t に対し，集合 $dep(t) = \{t' \mid (t, t') \in D\}$ を定義する．

Table 4.1: 半順序法のアルゴリズム

Algorithm: *Partial order method*
  procedure $search(M)$
  begin
  $working := select(M) \cap en(M)$;
  if $\forall t \in working : succ_t(M) \in Stack$ **then**
    $working := en(M)$;
  for each $t \in working$ **do**
    begin
    $M' := succ_t(M)$;
    if M' is not in H **then**
      begin
      enter M' in H;
      push M' onto $Stack$;
      $search(M')$;
      end
    end
  pop s from $Stack$;
  end.

begin
$Stack$ is empty; H is empty;
push M_0 onto $Stack$;
$search(M_0)$;
end.

また，$t \notin en(M)$ のとき，集合 $up_M(t) \subset T$をつぎの条件を満たす集合として定義する．

$$M \stackrel{\sigma t}{\rightarrow} \ \wedge \ t \notin T_\sigma \Rightarrow up_M(t) \cap T_\sigma \neq \emptyset. \tag{4.2}$$

ここで T_σは発火系列σに含まれるトランジションの集合である．集合 $up_M(t)$ はトランジション t が生起する以前に発火しなければならないトランジションを少なくとも 1 つは含むような集合である．

従属性 Dにより実行可能なトランジションの系列上に同値関係が定義できる．2 つのトランジションの系列は，隣接する独立なトランジションの順序を入れ替えていくことにより一方が他方に等しくなるならば同値であると定義する．これによりできる同値類をトレース (trace) という [29]．$[\sigma]_D$によりσを含むトレースを表す．

ある状態が到達可能であることを調べるには，そこに至るトレースに含まれる 1 つの系列を調べるだけでよい．そのような系列が必ず縮約状態空間に含まれるよう

に選択集合 $select(M)$ を決定していけばよい.

$dep(t)$ および $up_M(t)$ を求めるのに定義どおりトランジションを発火させて条件を調べる必要はなく，その十分条件となる構文的条件（ペトリネット構造だけから判定可能な条件）を見つければよい．以下はその十分条件の一例である [46].

- $dep(t) = \{t' \in T \mid \exists p \in P : A(p,t') > A(t',p) < A(p,t)\}$.
- $M(p) < A(p,t)$ であるようなプレース p に対し，

$$up_M(t) = \{t' \in T \mid A(t',p) > A(p,t') < A(p,t)\}.$$

4.2.4 デッドロックの検出

どのトランジションも生起できないようなマーキングは**デッド** (dead) であるという．デッドなマーキングが到達可能であることは，システムがデッドロックに陥る可能性があることを意味する．半順序法の重要な成果として，デッドなマーキングを保存するような縮約状態空間の構成がある [46].

各マーキング Mにおいて $select(M)$ として以下を満たす最小の集合を選ぶ.

1. $t \in select(M) \cap en(M) \Rightarrow dep(M) \subset select(M)$.
2. $t \in select(M) - en(M) \Rightarrow up_M(t) \subset select(M)$.
3. $\exists t \in select(M) \cap en(M)$.

このとき，得られた縮約状態空間は通常の状態空間がもつデッドなマーキングをすべて保存することが証明されている.

定理 4.1 半順序法のアルゴリズムにより作られる探索木上の任意の状態 Mについて，もし Mから系列 σ によりデッドな状態 M_dに到達可能ならば，M_dもまた探索木に出現する.

証明 σの長さ $|\sigma|$ に関する帰納法を用いる．$|\sigma| = 0$ のときは明らか．$|\sigma| > 0$ の場合を考える.

1. σ は $select(M)$ のトランジションを少なくとも 1 つ含む.
   もしそうでないと仮定すると，条件 3 より $select(M)$ は発火可能なトランジ

ション t を少なくとも 1 つは含み，さらに条件 1 より，t と従属なトランジション集合 $dep(t)$ はすべて $select(M)$ に含まれる．すなわち，t の発火可能性は σ の実行後も保たれる．これは M_dがデッドであることに矛盾する．

2. σ に含まれる $select(M)$ のトランジションで最初に出現するものを t'とすると，条件 2 より t'は Mで発火可能である．
これはつぎのようにして示せる．$\sigma = \sigma_1 t' \sigma_2$で，$\sigma_1$には $select(M)$ のトランジションは含まれないとする．t'が Mで発火不能ならば，それを発火可能にできるトランジション集合 $up_M(t)$ はすべて $select(M)$ に含まれており，それらは σ_1 には含まれない．これは σ_1 の実行後，t'が依然として発火不能なままであることを意味し，仮定に反する．

3. 条件 1 より t'と従属なトランジションはすべて $select(M)$ に含まれていることから，
$$M \stackrel{t'}{\rightarrow} M' \stackrel{\sigma_1\sigma_2}{\rightarrow} M_d$$
が成り立つ．

$t' \in select(M)$ なので M'は探索木に出現する．帰納法の仮定により M_d も探索木に出現する． ∎

半順序法はデッドロックの検出だけでなく，時相論理で記述されるより一般的な性質に対しても適用できることが示されている [36].

問題 4.1 Figure 3.5のペトリネットを半順序法で探索せよ．

4.3 BDD のペトリネットの解析への応用

4.3.1 BDD とは

BDD(binary decision diagram) はブール関数を 2 分決定木の形で表現する方法であり，VLSI の論理合成等で実用的に用いられている [1, 9, 23, 13]．特に，変数の順序を固定し，冗長な節点まとめた **ROBDD**(reduced ordered BDD) により，パリティ関数などの実用的に重要な関数を効率的に表現することができる．また，各種の論理演算をすべて BDD 上の操作として実現できる（Figure 4.1, 4.2）.

ペトリネットを状態遷移図の生成器として見れば，従来行われていた BDD による解析手法がそのまま適用できる．

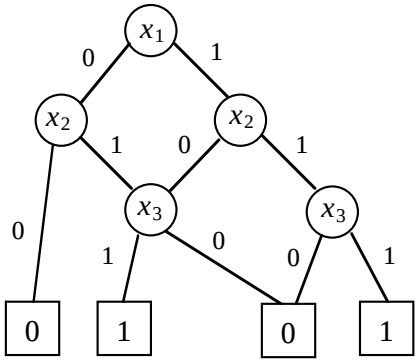

Figure 4.1: ブール関数 $(x_1 \vee x_2) \wedge x_3$を表現する BDD.

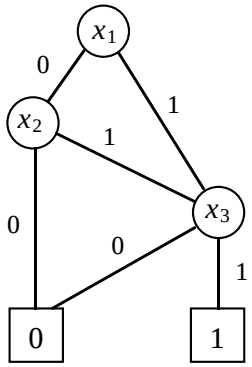

Figure 4.2: ブール関数 $(x_1 \vee x_2) \wedge x_3$を表現する ROBDD.

Table 4.2: 到達可能集合を求めるアルゴリズム

Algorithm *Reachability set*
begin
$R := \{M_0\}$;
do
$R' := R$;
$R := R' \cup \{M' \mid \exists M \in R' : \Delta(M, M') = TRUE\}$
until $R \neq R'$
end.

4.3.2 BDD による到達可能性集合の表現

ペトリネットの到達可能集合を BDD で計算するためには，(i) 各マーキングをブール変数に符号化する方法，および，(ii)1 ステップの状態遷移を表すブール関数 $\Delta(M, M')$（マーキング Mからマーキング M'が 1 つのトランジションの発火により到達可能なときに真，それ以外で偽となる.）を与えればよい．このとき，到達可能集合は表 4.2のアルゴリズムで計算される．

これらの計算をすべて BDD 上の操作として実行する．ペトリネットがセーフならば各プレースは 0,1 の値のみをとるので 1 つのブール変数に対応させることができる．一般の有界ペトリネットの場合は，1 進数符合化，2 進数符合化などいくつかの方法が考えられる．また，変数としては現在のマーキングを表す変数，および，1 ステップの遷移後のマーキングを表す変数の 2 組必要である．ブール関数$\Delta(M, M')$は各トランジションの発火による状態遷移の論理和として得られる．

4.3.3 記号モデルチェッキング

記号モデルチェッキング (simbolic model checking) とは，状態空間を直接生成せず与えられた性質（時相論理式で記述される）が真となる状態を表わす論理式を計算することによりシステムの検証を行う方法である [10]．状態を列挙することなしに，論理式の計算のみで検証を行うことからこの名称が使われている．

ここでは時相論理式として CTL(computation tree logic) を用いた場合について説明する．CTL の論理式は原子命題の集合$\wp$および以下に述べるオペレータから作られる．

1. 状態論理式
    - 原子命題 $p \in \wp$は状態論理式である．

- fおよび gが状態論理式ならば $f \wedge g$および$\neg f$も状態論理式である．
- もし h がパス論理式ならば $E\,h$ および $A\,h$ は状態論理式である．

2. パス論理式

- fおよび gが状態論理式ならば $X\,f$および $f\,U\,g$はパス論理式である．

CTL の意味はラベル付遷移システム上で与えられる．以下，パスとはラベル付遷移システムをグラフとして見たときのパスのことを意味する．

原子命題は各状態に対して真理値が決まる．あるパスに対し，$X\,f$はパス上の 2 番目の状態で fが真となることを，$f\,U\,g$はパス上の状態で gが真となるまでは fが真であることを意味する．また，ある状態が与えられたときに，Eはその状態を始点とするあるパスに対し真となることを意味し，A はすべてのパスに対し真となることを意味する．$f \vee g$は$\neg(\neg f \wedge \neg g)$ により，$f \rightarrow g$は$\neg(f \wedge \neg g)$ により定義される．また，よく用いられる時相オペレータ F(eventually)，G(always) も上記のオペレータを用いて定義できる．$F\,f$はパス上でいつかは fが真，$G\,f$は常に fが真の意味である．

CTL によりペトリネットの様々な性質を記述できる．いま，原子命題を $enabled_t$（トランジション t が発火可能）とすると，たとえば**活性** (liveness)[1]はつぎのように書ける．

$$AGEF\ enabled_t$$

また，デッドなマーキングが存在することは以下のように書ける．

$$EF \bigwedge_{t \in T} \neg enabled_t$$

与えられた CTL 論理式が真となる状態を表す論理式は以下のように計算できる．

1. $EX\ f$

$$EX\ f \equiv \exists M' : [\Delta(M, M')\ \wedge\ f(M')]$$

2. $E[f\ U\ g]$ は以下の式の最小不動点

$$Z = g \vee [f \wedge EX\ Z]$$

3. $EG\ f$は以下の式の最大不動点

$$Z = f \wedge EX\ Z$$

[1]ペトリネット PNは$\forall M \in R(PN) \forall t \in T \exists \sigma \in T^* : M \xrightarrow{\sigma t}$のときライブであるという．

4. $A[f\ U\ g]$, $EF\ f$, $AF\ f$, $AG\ f$, $AX\ f$は上記のオペレータから導かれる．

これらの計算は状態遷移関数$\Delta(M, M')$が求められていればすべてBDD上の操作として実行できる．

4.4 アンフォールディング

ペトリネットの**アンフォールディング** (unfolding) とは，ペトリネットを発火ステップにしたがって展開した**オカレンスネット** (occurrence net) を用いた解析手法である．

4.4.1 オカレンスネット

オカレンスネットは初期マーキングにトークンが存在するプレースをまず作り，発火可能なトランジションのコピー（インスタンス）を順次追加していくことにより得られる．このとき，追加するトランジションの入力プレースはすべて「並行」（それらの間を結ぶパスが存在しない）でなければならない．また，オカレンスネットには一般に同一のトランジションからできたインスタンスが複数回出現する．たとえばFigure 4.3のペトリネットに対するオカレンスネットはFigure 4.4のようになる．

4.4.2 カットオフ点

オカレンスネットをある場所（**カットオフ点**）で切断したとき，それ以前の部分だけからなる部分ネットをもとのペトリネットのアンフォールディングという．カットオフ点をある基準にしたがって選ぶことにより，もとのペトリネットの到達可能なマーキングをすべて含むようなアンフォールディングを得ることができる．

オカレンスネットは有向非循環グラフであるので，トランジションのインスタンス間に半順序関係が定義できる．すなわち，オカレンスネットは半順序関係に基づいた意味論による状態空間の表現であると言える．ペトリネットのアンフォールディングのサイズは多くの場合，もとのペトリネットの多項式サイズで抑えられることが知られている．

カットオフ点を決める基礎となるのは**コンフィグレーション** (configuration) の考え方である．オカレンスネットのトランジションの部分集合C'は以下を満たすと

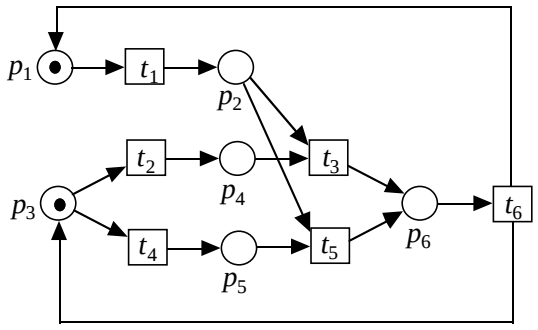

Figure 4.3: ペトリネット.

きコンフィグレーションであるという.

1. 各 $t'_i \in C'$について，C'は $t'_j \preceq t'_i$であるようなトランジション t'_jをすべて含む. ここで，$\preceq$はオカレンスネットによって与えられる半順序である（すなわち，C'は downwards closed）.
2. C'は互いに「競合」（入力プレースを共有）するトランジションを含まない.

Figure 4.4で太枠で描いたトランジション集合はコンフィグレーションである.

トランジション t'_iおよびそれより小さいトランジションからなるコンフィグレーションを t'_iの局所コンフィグレーションといい$\downarrow t'_i$で表す. あるコンフィグレーション C'のトランジションがすべて発火した後のマーキングを C'の最終マーキングとよぶ. t'_iは以下の条件を満たす $t'_j(\neq t'_i)$ が存在するときカットオフ点となることが示されている.

1. $\downarrow t'_i$と$\downarrow t'_j$の最終マーキングが等しい.
2. $|\downarrow t'_i| > |\downarrow t'_j|$.

この基準で得られるものよりも小さいアンフォールディングを得るためのカットオフ点の定義も求められている.

4.5 章末問題

問題 4.2 本章で紹介した効率的解析のための方法は，いずれもその効果を問題のイ

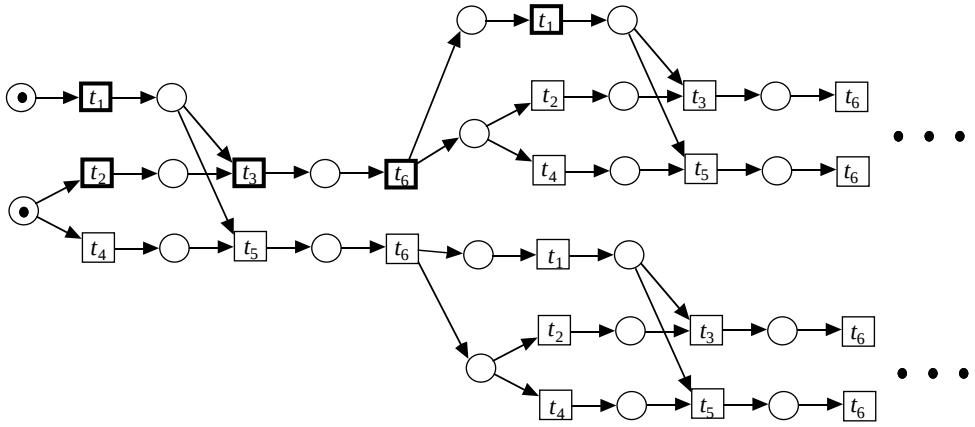

Figure 4.4: オカレンスネット.

ンスタンスとは独立には評価できない（たとえば，探索時間を最悪でも何%削減できるなど）．これはなぜか.

Chapter 5

おわりに

本稿では離散事象システムを表現・制御・解析するための理論について紹介した．この理論は制御ソフトウェア設計等に対して新しい視点を与えるものであり，今後の発展が期待される．また，理論的な発展としては，エージェント指向ソフトウェアに対応できるような拡張が考えられる．

Bibliography

[1] Akers S. B.: Binary decision diagrams, IEEE Trans. Computers, Vol.C27, No.6, pp.509-516 (1978).

[2] 青山幹雄, 平石邦彦, 内平直志: “高水準ペトリネットによるソフトウェア開発方法論”, コンピュータソフトウェア, Vol.11, No.4, pp.3–20 (1994).

[3] 青山幹雄，内平直志，平石邦彦: ペトリネットの理論と実践, 朝倉書店 (1995).

[4] 青山幹雄: 並行ソフトウェア設計のための記述モデルとその実際 - ペトリネット -, 第 8 回回路とシステム軽井沢ワークショップ, pp.251–256 (1995).

[5] Bdouel E., Bernardinello L. and Darondeau P.: “The synthesis problem for elementary net systems is NP-complete”, INRIA Rapport de recherche, No.N-2558 (1995).

[6] Baeten, J. C. M. and W. P. Weijland: Process algebra, Cambridge Tracts in Theoretical Computer Science Vol.18, Cambridge University Press (1990).

[7] Berthelot G.: Checking properties of nets using transformations, Lecture Notes in Computer Science, Vol.222, pp.19–40(1985).

[8] Boigelot B. and Wolper P.: Symbolic verification with periodic sets, Lecture Notes in Computer Science, Vol.818, pp.55–67 (1994).

[9] Bryant R. E.: Graph-based algorithms for boolean function manipulation, IEEE Trans. Computers, Vol.C35, No.8, pp.677–691 (1986).

[10] Burch J. R., Clarke E. M. and McMillan K. L.: Sequential circuit verification using symbolic model checking, Proc. 27th ACM/IEEE Design Automation Conference, pp.46–51 (1990).

[11] Esparza J.: Model checking using net unfoldings, Science of Computer Programming, Vol.23, pp.151–195 (1994).

[12] Godefroid P. and Wolper P.: Using partial orderes for the efficient verification of deadlock freedom and safety properties, Lecture Notes in Computer Science, Vol.575, pp.332–342 (1991).

[13] 藤田昌宏 and Clarke E. M.: “BDD の CAD への応用”, 情報処理, Vol.34, No.5, pp.609–616 (1993).

[14] Hwang C., Kim U. and Lee D.: A concurrency characteristic in unfolding, Proc. ITC-CSCC '96, Vol.2, pp.1212–1217 (1996).

[15] Jensen K.: Coloured Petri nets, basic concepts, analysis methods and practical use – Volume 2, Springer-Verlag (1995).

[16] Kishinevsky M., Cortadella J., Kondrathev A. and Lavagno L.: Synthesis of general Petri nets, 電子情報通信学会技術報告, Vol.96, No.57, CST96-5 (1996).

[17] D. Harel et al., STATEMATE: A Working Environment for the Development of Complex Reactive Systems, Proc. 10th IEEE ICSE (1988).

[18] Hiraishi K.: Some complexity results on transition systems and elementary net systems, Theoretical Computer Science, Vol.135, pp.361–376 (1994).

[19] Hiraishi K.: Reduced state space generation of concurrent systems using weak persistency, IEICE Trans. Vol.E77-A, No.10, pp.1602–1606 (1994).

[20] Hiraishi K. and Nakano M.: On symbolic model checking in Petri nets, IEICE Trans. Vol.E78-A, No.11, pp.1479–1486 (1995).

[21] Hiraishi K.: Reduced state space representation for unbounded vector state spaces, Lecture Notes in Computer Science, Vol.1091, pp.230–248 (1996).

[22] Hoare, C. A. R.: Communicating Sequential Processes, Prentice-Hall (1985).

[23] 石浦菜岐佐: BDD とは, 情報処理, Vol.34, No.5, pp.585–592 (1993).

[24] 磯部祥尚: 並行ソフトウェア設計のための記述モデルとその実際 - プロセス代数 - , 第 8 回回路とシステム軽井沢ワークショップ, pp.263–269 (1995).

[25] Kondratyev A.: Verification of asynchronous systems based on Petri net unfoldings, 電子情報通信学会技術報告, Vol.96, No.57, CST96-4 (1996).

[26] 熊谷貞俊, 薦田憲久: ペトリネットによる離散事象システム論, コロナ社 (1995).

[27] Kumar, R. and V. K. Garg: Modeling and Control of Logical Discrete Event Systems, Kluwer Academic Publishers (1995).

[28] McMillan K. L.: Using unfoldings to avoid the state space explosion problem in the verification of asynchronous circuits, Lecture Notes in Computer Science, Vol.663, pp.164–175 (1992).

[29] Mazurkiewics A.: Trace semantics, Lecture Notes in Computer Science, Vol.255, pp.278–324 (1987).

[30] Milner, R.: Communication and concurrency, Prentice Hall (1989).

[31] Murata T.: Petri nets: properties, analysis and applications, Proceedings of the IEEE, Vol.77, No.4, pp.541–580 (1989).

[32] 村田忠夫: ペトリネットの解析と応用, 近代科学社 (1992).

[33] 村田忠夫，辻孝吉: ペトリネットによる並行処理プログラムの解析手法, 情報処理, Vol.34, No.6, pp.701–709 (1993).

[34] 奥川峻史: ペトリネットの基礎, 共立出版 (1995).

[35] Pastor E., Roig O., Cortadella J. and Badia R.: Petri net analysis using boolean manipulation, Lecture Notes in Computer Science, Vol.815, pp.416–435 (1994).

[36] Peled D.: Combining partial order reductions with on-the-fly model checking, Lecture Notes in Computer Science, Vol.818, pp.377–390 (1994).

[37] Peterson, J. L.: Petri Net Theory and the Modeling of Systems, Prentice-Hall (1981) [市川惇信，小林重信訳：ペトリネット入門, 共立出版 (1984)].

[38] Ramage, P. J. and W. M. Wonham: On the supremal controllable language of a given language, SIAM J. Contr. & Opt., Vol.25, No.3, pp.637-659 (1987).

[39] P. J. G. Ramage and W. M. Whonham: The control of discrete event systems, Proc. IEEE, Vol.77, No.1, 81-98 (1989).

[40] Reisig, W.: Petri Nets: An Introduction, EATCS Monographs on Theoretical Computer Science, Vol.4, Springer-Verlag (1985) [長谷川健介，高橋宏治訳：ペトリネット理論入門, シュプリンガー・フェアラーク東京 (1988)].

[41] 椎塚久雄: 実例ペトリネット, コロナ社 (1992).

[42] Takahashi K., Yamamura M. and Kobayashi S.: A GA approach to solving reachability problems for Petri nets, IEICE Trans. Vol.E79-A, No.11, pp.1774–1780 (1996).

[43] Tiusanen M.: Symbolic, symmetry, and stubborn set searches, Lecture Notes in Computer Science, Vol.815, pp.511–530 (1994).

[44] 内平直志: 並行ソフトウェア設計のための記述モデルとその実際 - 状態遷移モデル -, 第 8 回回路とシステム軽井沢ワークショップ, pp.257–262 (1995).

[45] 潮俊光: 離散事象システムにおける制御問題とスーパバイザ, システム／制御／情報, Vol.34, No.9, 531-538 (1990).

[46] Valmari A.: Stubborn sets for reduced state space generation, Lecture Notes in Computer Science, Vol.483, pp.491–515 (1990).

[47] Valmari A.: On-the-fly verification with stubborn sets, Lecture Notes in Computer Science, Vol.697, pp.397–408 (1993).

[48] Wolper P., Godefroid P. and Pirottin D.: A Tutorial on partial-order methods for the verification of concurrent systems, Computer Aided Verification '93 Tutorial (1993).