ソフトウェアモデル論（2013 年度）

桑原 寛明
情報理工学部 情報システム学科
kuwabara@cs.ritsumei.ac.jp
http://www.ritsumei.ac.jp/~hkuwa/class/2013/model/

目次

第 1 章

はじめに

本講義では，有限オートマトン，チューリング機械，命題論理，モデル検査，の 4 つの話題を取り上げる．これらは，ソフトウェアあるいは計算機が行う動作（すなわち “計算”）を形式的，数学的にとらえモデル化し理解するための理論と技術である．本講義の目標は，モデル検査の基礎を理解しソフトウェア検証に応用できることであるが，その過程を通して，計算機科学の理論的基礎であるオートマトン理論や数理論理学に対する理解を深めたい．

モデル検査は，システムの動作を表す有限状態モデルを網羅的に探索することで，システムが与えられた性質を満たすか判定する検証手法である．他の形式検証技術に対し，モデル検査は完全に自動化することが可能である，システムが与えられた性質を満たさないと判定した場合にその具体的な例（反例）を示すことができる，という利点がある．多くの場合，システムを表す有限状態モデルとして Kripke（クリプキ）構造やオートマトンが利用され，性質の表現には時相論理が利用される．

本講義では，モデル検査を取り上げる前にその準備として，モデル検査も含め計算機科学全般の基盤であるオートマトン理論から**有限オートマトン**，数理論理学から**命題論理**について解説する．そして，有限オートマトンの拡張として Kripke 構造を，命題論理の拡張として時相論理を導入し，モデル検査の基礎を解説する．さらに，モデル検査の応用として，逐次プログラムにはない難しさを持つ**並行プログラム**のモデル検査を取り上げる．

有限オートマトンは現在の計算機の動作を表現できる形式モデルの一つである．計算機は “計算” を行う機械であるため，有限オートマトンは “計算” を表していると考えることもできるが，有限オートマトンの表現能力は制限されている．“計算” を表現するより一般的な形式モデルとして**チューリング機械**がある．チューリング機械は計算可能性や計算量を考える際に重要な役割を果たしており，現在の計算機は原理的には（そして計算能力も）チューリング機械と同等である．本講義では，チューリング機械の基礎とチューリング機械でも計算できない問題についても簡単に解説する．

第 2 章

数学的準備

2.1　集合

集合と記法

集合 (set) とは，互いに区別することができる “もの” の集まりを指す．集合として集められた “もの” をその集合の**元**あるいは**要素**と呼ぶ．集合は同じ要素を高々 1 つしか含まないことに注意する．ある “もの” を x と表す時，x が集合 S の要素であることを $x \in S$ と書く．

集合は要素を指定すれば決定される．例えば，3 つの要素 a, b, c を持つと決めればそのような集合は一意に定まる．この集合を $\{a, b, c\}$ と書いて表す．要素を記述する順序に意味はなく，$\{a, b, c\}$ と $\{c, b, a\}$ は同じ集合を表す．$\{x_1, \ldots, x_n\}$ のように要素を省略して記述することもある．$\{a, b, c\}$ のように要素の数が有限である集合を**有限集合**と呼ぶ．一方，要素の数が無限である集合を**無限集合**と呼ぶ．例えば，すべての自然数の集合は無限集合である．無限集合を $\{x_1, x_2, \ldots\}$ のように書いて表す．有限集合，無限集合に関わらず，$\{x \mid \cdots\}$ のように書いて条件 $\cdots$ を満たす要素を集めた集合を表す．例えば，$\{x \mid x$ は整数 $\}$ のように書いて整数の集合を表すことができる．

等しい集合

2 つの集合 S, T があって，S が含む要素と T が含む要素が同じである時，S と T は等しい．このことを $S = T$ と書く．

空集合

要素を 1 つも含まない集合を**空集合** (empty set) と呼び，$\emptyset$ と書く．$\{\}$ と書く場合もある．$\emptyset = \emptyset$ であるので，空集合はただ 1 つだけ存在する．

部分集合

2 つの集合 S, T があって，S の要素がすべて T にも含まれている場合，S は T の**部分集合** (subset) であるといい，$S \subseteq T$ と書く．$S \subseteq T$ であり同時に $S \neq T$ である場合，S は T の真部分集合であるという．真部分集合であることを強調したい場合，$S \subset T$ や $S \subsetneq T$ のように書く．任意の集合 S に対して $S \subseteq S$ である．$S = T$ であることと，$S \subseteq T$ かつ $T \subseteq S$ であることは同じである．空集合は任意の集合の部分集合である．

つまり，任意の集合 S に対して $\emptyset \subseteq S$ である．

和集合

2 つの集合 S,T があって，S の要素と T の要素をすべて集めて新しい集合を求めることを集合の**和**，得られる集合を**和集合** (union) といい，$S \cup T$ と書く．$S \cup T = \{x \,|\, x \in S \text{ または } x \in T\}$ である．

積集合

2 つの集合 S,T があって，S と T の両方に含まれる要素のみを集めて新しい集合を求めることを集合の**積**，得られる集合を**積集合** (intersection) といい，$S \cap T$ と書く．$S \cap T = \{x \,|\, x \in S \text{ かつ } x \in T\}$ である．

差集合

2 つの集合 S,T があって，S の要素の中から T にも含まれる要素を除いて新しい集合を求めることを集合の**差**，得られる集合を**差集合** (difference set) といい，$S - T$ あるいは $S \backslash T$ と書く．$S - T = \{x \,|\, x \in S \text{ かつ } x \notin T\}$ である．

補集合

集合 S があって，S に含まれない要素を集めた集合を S の**補集合** (complement) といい，$\overline{S}$ と書く．$\overline{S} = \{x \,|\, x \notin S\}$ である．考えている対象の全体の集合 U が決まっていれば，$S \subseteq U$ および $\overline{S} = U - S$ である．

べき集合

集合 S があって，S の部分集合をすべて集めた集合を S の**べき集合** (power set) といい，2^S と書く．$2^S = \{T \,|\, T \subseteq S\}$ である．例えば，$S = \{a,b,c\}$ とすると $2^S = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$ である．

ベン図

和集合，積集合，差集合および補集合を図示する方法として**ベン図**がある．2 つの集合の場合を図 2.1 に示す．

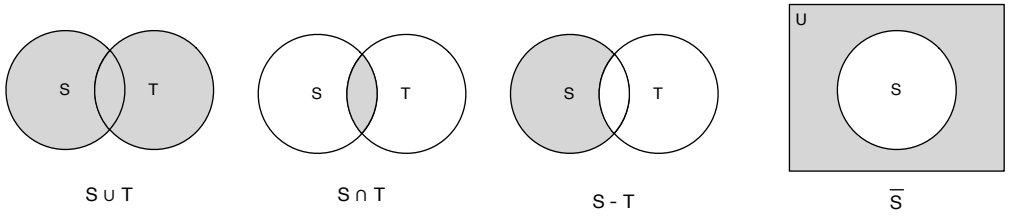


図 2.1　ベン図

直積

2 つの集合 S,T があって，S の要素 s と T の要素 t を s,t の順に順序付けた組を**順序対**と呼び，(s,t) と書く．順序が重要であり，$s \neq t$ ならば (s,t) と (t,s) は異なることに注意する．また，S の要素と T の要素からなる順序対の集合を S と T の**直積** (direct product, Cartesian product) と呼び，$S \times T$ と書く．$S \times T = \{(s,t) \,|\, s \in S \text{ かつ } t \in T\}$ である．例えば，$S = \{a,b\}, T = \{1,2\}$ とすると $S \times T = \{(a,1),(a,2),(b,1),(b,2)\}$ である．直積も順序が重要であり，$S \neq T$ ならば $S \times T$ と $T \times S$ は異なる．

練習問題 2.1 $S = \{a,b,c,d\}$，$T = \{c,d,e,f\}$ とする．$S \cup T$，$S \cap T$，$S - T$，$T - S$，$S \times T$，$T \times S$ をそれぞれ求めよ．

2.2 関係

関係と記法

「1 は 2 より小さい」といった数の大小関係のように，“もの” と “もの” の間に何らかの関係が存在する場合があるが，この関係の概念を一般的に定義する．直観的には，関係が存在する “もの” と “もの” の組合せをすべて列挙すればよい．ここで，大小関係のように組合せの順序が重要な関係もあるため，組合せは順序対とする．

2 つの集合 S,T があって，直積 $S \times T$ の部分集合を S から T への**関係** (relation) という．特に，$S \times S$ の部分集合を S 上の関係という．例えば，$S = \{1,2,3\}$ とすると，数の大小に関する S 上の関係 $<$ は $\{(1,2),(1,3),(2,3)\}$ である．R が S から T への関係であり $(s,t) \in R$ であることを sRt とか $R(s,t)$ などと表す場合もある．

性質

関係は様々な性質を持つが，特に重要な性質を挙げる．以下では，R は S 上の関係であり，$x,y,z \in S$ とする．

- 任意の $s \in S$ に対して sRs である時，R は**反射的** (reflexive) であるという．
- xRy ならば yRx である時，R は**対称的** (symmetric) であるという．
- xRy かつ yRx ならば $x = y$ である時，R は**反対称的** (anti-symmetric) であるという．
- xRy かつ yRz ならば xRz である時，R は**推移的** (transitive) であるという．

順序関係

関係が反射的かつ反対称的かつ推移的である時，その関係は**順序関係** (order) であるという．集合 S 上の関係 R が順序関係であり，S の任意の 2 つの要素 s,s' に対して sRs' あるいは $s'Rs$ である時，R を**全順序関係** (total order) と呼ぶ．全順序でない順序関係を特に**半順序関係** (partial order) と呼ぶ．半順序関係が定義された集合を**半順序集合**，全順序関係が定義された集合を**全順序集合**と呼ぶ．

同値関係

関係が反射的かつ対称的かつ推移的である時，その関係は**同値関係** (equivalent relation) である．

合成と閉包

S から T への関係 R_1 と T から U への関係 R_2 に対して，$\{(s,u) \mid$ ある $t \in T$ があって sR_1t かつ $tR_2u\}$ を R_1 と R_2 の**合成**といい，$R_1 \circ R_2$ と書く．直観的には，s から R_1, R_2 の順に関係をたどって u に到達できることを表す．

n を自然数[*1]，R を S 上の関係，$s_0, s_n \in S$ とする．この時，

$$\{(s_0, s_n) \mid \text{ある } s_1, \ldots, s_{n-1} \in S \text{ が存在して } 0 \leq i < n \text{ なる } i \text{ に対して } s_i R s_{i+1}\}$$

を R の**反射推移閉包** (reflexive transitive closure) といい，R^* と書く．$n = 0$ や $n = 1$ の場合も反射推移閉包に含まれることに注意する．直観的には，$(s, s') \in R^*$ ならば s から関係 R を 0 回以上たどって s' に到達できることを意味する．任意の関係 R について，その閉包 R^* は反射的かつ推移的であり，$R \subseteq R^*$ である．もし R が反射的かつ推移的であれば $R = R^*$ である．

練習問題 2.2 $S = \{1, 2, 3\}$ とし，以下の S 上の関係それぞれについて，反射性，対称性，反対称性，推移性のうち満たすものをすべて挙げよ．

1. $\{(1,2), (1,3), (2,3)\}$（つまり $<$）
2. $\{(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)\}$（つまり $\leq$）
3. $\{(1,1), (2,2), (3,3)\}$（つまり $=$）

練習問題 2.3 $S = \{1, 2, 3, 4\}$ とし，S 上の関係 R を $R = \{(1,2), (2,3), (3,4)\}$ とする．R の反射推移閉包を求めよ．

2.3 関数

関数と記法

集合 S の各要素に対して集合 T の要素を 1 つ対応付ける規則 f を**関数** (function) あるいは**写像** (mapping) と呼ぶ．$f : S \to T$ と書いて f が S の要素に T の要素を対応付ける関数であることを表す．$s \in S$ が f によって $t \in T$ に対応付けられる時，このことを $f(s) = t$ と書く．

定義域と値域

関数 $f : S \to T$ において，S を f の**定義域** (domain) という．f の定義域を $\mathrm{dom}(f)$ と書く場合もある．また，$T' = \{f(s) \mid s \in S\}$ を f の**値域** (range) という．一般に，$T' \subseteq T$ である．

[*1] 0 以上の整数

$S' \subseteq S$ とする．この時，$f : S \to T$ に対して $f' : S' \to T$ を S から T への**部分関数** (partial function) という．部分関数に対して，f を特に**全関数** (total function) という．

単射，全射，全単射

$s, s' \in S$ とする．関数 $f : S \to T$ において，$s \neq s'$ ならば $f(s) \neq f(s')$ である時，f は**単射** (injection) であるという．f が単射ならば，f によって S の各要素に対して異なる T の要素が対応付けられる．f の値域が T に等しい時，f は**全射** (surjection) であるという．関数が単射かつ全射である時，その関数は**全単射** (bijection) であるという．

合成

2 つの関数 $f : S \to T$ および $g : T \to U$ があって，$s \in S$ に対して $g(f(s))$ を対応付ける関数 $h : S \to U$ を f と g の**合成**といい，$g \circ f$ と書く．

第 3 章

有限オートマトンと正規表現

3.1　順序機械と状態遷移

何かが入力されると何かを出力する機械を考える．ただし，機械の内部はどうなっているかわからないブラックボックスとする．このような機械は，今までの入力には関係なく同じ入力に対しては常に同じ出力を行う機械と，今までの入力によって次の入力に対する出力が変化する機械の 2 種類にわけられる．前者の機械では次の出力を次の入力だけから求めることができる．一方，後者の機械では次の出力を求めるために今までの入力で決まり次の出力に影響する “何か” を覚えておく必要がある．例えば，ジュースや切符の自動販売機はどの紙幣や硬貨が何枚投入されたか覚えておかなければならない．100 円玉が投入された時に商品のランプを点灯するかはそれまでに投入された金額によるからである．このように，ある入力に対する出力がそれまでの入力によって変化する機械を**順序機械**と呼ぶ．

自動販売機の例では，10 円玉が 1 枚投入された場合と 100 円玉が 1 枚投入された場合では内部的に何かが異なっていなければならない．この “何か” を**状態**と呼ぶ．つまり，「投入金額が 10 円の状態」と「投入金額が 100 円の状態」があり，それらは異なる状態とみなされる．もし 200 円の商品を販売しているのであれば，「投入金額が 10 円の状態」で 100 円玉が 1 枚投入されても商品のランプは点灯させないが，「投入金額が 100 円の状態」で 100 円玉が 1 枚投入されると商品のランプを点灯させる．すなわち，順序機械の出力はその時の状態と入力によって決定される．

「投入金額が 10 円の状態」で 100 円玉が 1 枚投入されたり「投入金額が 100 円の状態」で 10 円玉が 1 枚投入されると「投入金額が 110 円の状態」に変化する．また，「投入金額が 100 円の状態」でもう 1 枚 100 円玉が投入されると「投入金額が 200 円の状態」に変化する．このように，順序機械が入力によってある状態から別の状態へと移行することを**状態遷移**という．図 3.1 のような状態遷移を表す図を**状態遷移図**と呼ぶ．

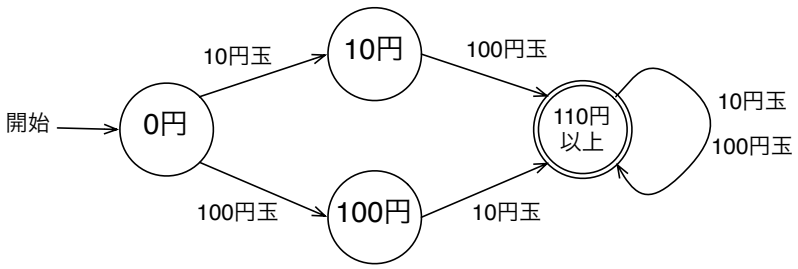


図 3.1　状態遷移図の例

このような状態遷移を伴う動作を一般的に表現するモデルを**オートマトン**といい，オートマトンを用いることで順序機械を統一的に記述できる．オートマトンは計算機が実行する “計算” を抽象的に表現するためにも利用される．オートマトンの概念図を図 3.2 に示す．オートマトンの入出力は有限個の種類の記号を使って表される．テープは入力元および出力先となる記憶媒体であり，1 つのマスに 1 つの記号が記録される．テープは必要なだけ十分に長いとする．ヘッドは入出力を行うテープのマスを指すものであり，ヘッドが指すマスに対して記号の読み書きが行われる．ヘッドは左右に 1 マスずつ移動することができる．制御部は機械の現在の状態を記憶しており，現在の状態とヘッドが指すマスから読み込んだ記号によって遷移先の状態，テープに書き込む記号，ヘッドの移動方向を決定する．テープの読み書きと制御部の状態遷移を繰り返しながら最終的に停止した時，もし制御部の状態が**最終状態**（または**受理状態**）と呼ばれる特別な状態に到達しているならば，かつその時に限り入力が**受理**されたとする．

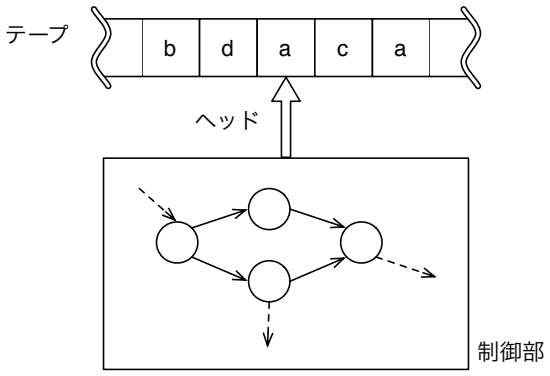


図 3.2　オートマトンの概念図

図 3.2 のテープの使用方法に制限を加えることで能力の異なる様々なオートマトンを構成することが可能であり，能力の低い方から高い方へ順に

- 有限オートマトン
- プッシュダウンオートマトン
- 線形有界（拘束）オートマトン
- チューリング機械

と呼ばれるオートマトンがある．以下では**有限オートマトン**について説明する．

3.2　決定性有限オートマトン

有限オートマトンは図 3.2 のテープの使用方法に以下の制限を加えたオートマトンである．

- 読み出しのみで，書き込みは禁止
- 1 マス読んだらヘッドを必ず右へ 1 マス進める

初め，テープには入力となる記号列が書き込まれており，左端のマスから読み込みを開始する．有限オートマトンの 1 回の動作は，

1. テープからヘッドが指すマスの記号を読む
2. 読んだ記号に応じて状態を遷移する
3. ヘッドを 1 マス右へ進める

の 3 ステップである．テープに書き込まれた記号列をすべて読んだ後に最終状態に到達していれば，入力となった記号列は受理される．

有限オートマトンは以下のように定義できる．

定義 3.1 (有限オートマトン) 有限オートマトン M は 5 項組 $(Q, \Sigma, \delta, q_0, F)$ で定義される．ここで，Q は状態の有限集合，Σ は入力記号の有限集合（アルファベット），δ は状態遷移関数，q_0 は初期状態，F は最終状態の集合である．ただし，$Q \neq \emptyset$，$q_0 \in Q$，$F \subseteq Q$ とする．δ は $Q \times \Sigma \to Q$ として定義される関数であり，状態 $q \in Q$ と記号 $a \in \Sigma$ に対して $\delta(q, a)$ は状態 q で記号 a を読んだ場合の遷移先の状態を与える． □

状態遷移関数は全関数とは限らないことに注意する．つまり，状態と記号の組み合わせによっては遷移先の状態が存在しない可能性がある．定義 3.1 の有限オートマトンは，各状態においてある記号による遷移先の状態が一意に決まるため**決定性有限オートマトン** (DFA: Deterministic Finite Automaton) とも呼ばれる．一方，一意に決まらない有限オートマトンは**非決定性有限オートマトン** (NFA: Non-deterministic Finite Automaton) と呼ばれる．非決定性有限オートマトンについては後で説明する．

例 3.2 以下のように制限された自動販売機の動作を有限オートマトンで表現することを考える．

- 150 円と 200 円の切符を販売する
- 50 円玉と 100 円玉のみが使用できる
- 投入可能金額は 200 円まで
- 投入金額に応じて購入可能な切符のランプを点灯する
- 釣り銭を返却する機能はない

状態を投入金額によって区別すると，0 円，50 円，100 円，150 円，200 円の 5 つの状態にわけられる．投入金額が x 円の状態を S_x と書くことにする．また，1 種類以上の切符が購入可能な状態を最終状態とする．この時，自動販売機の動作は以下の有限オートマトンによって表すことができる．

$$
\begin{aligned}
&Q : \{S_0, S_{50}, S_{100}, S_{150}, S_{200}\} \\
&\Sigma : \{\text{50 円玉, 100 円玉, 150 円切符, 200 円切符 }\} \\
&\delta : (S_0, \text{50 円玉}) \to S_{50},\ (S_0, \text{100 円玉}) \to S_{100}, \\
&\quad (S_{50}, \text{50 円玉}) \to S_{100},\ (S_{50}, \text{100 円玉}) \to S_{150}, \\
&\quad (S_{100}, \text{50 円玉}) \to S_{150},\ (S_{100}, \text{100 円玉}) \to S_{200}, \\
&\quad (S_{150}, \text{50 円玉}) \to S_{200},\ (S_{150}, \text{150 円切符}) \to S_0, \\
&\quad (S_{200}, \text{150 円切符}) \to S_{50},\ (S_{200}, \text{200 円切符}) \to S_0 \\
&q_0 : S_0 \\
&F : \{S_{150}, S_{200}\}
\end{aligned}
$$

状態遷移関数は図 3.3 のような状態遷移図や表 3.1 のような状態遷移表を使って表すこともできる． □

練習問題 3.3 例 3.2 の有限オートマトンを 300 円の切符が扱えるように拡張せよ．

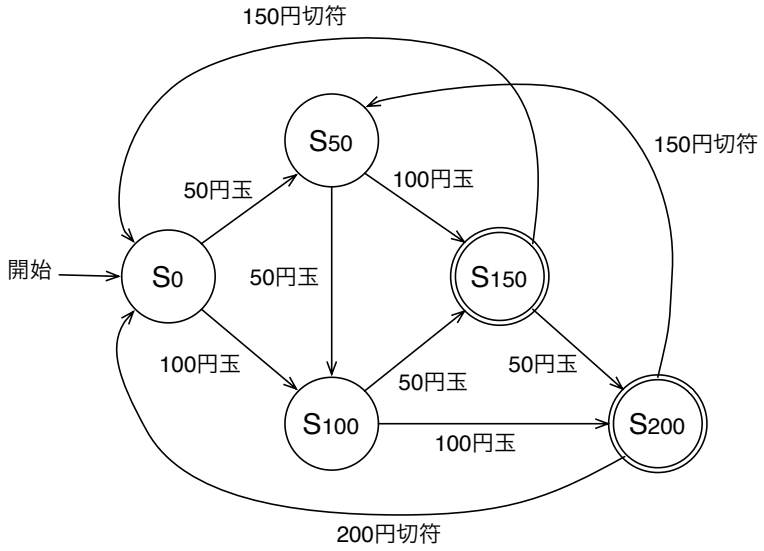


図 3.3　自動販売機の状態遷移図

表 3.1　自動販売機の状態遷移表

	50 円玉	100 円玉	150 円切符	200 円切符
S_0	S_{50}	S_{100}	-	-
S_{50}	S_{100}	S_{150}	-	-
S_{100}	S_{150}	S_{200}	-	-
S_{150}	S_{200}	-	S_0	-
S_{200}	-	-	S_{50}	S_0

記号列を**語**と呼ぶ．記号の集合 Σ に含まれる記号から作られる語の集合を Σ^* と書く．例えば，$\Sigma = \{a, b\}$ とすると，$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ である．ここで ε は空列（長さが 0 の記号列）を表す．語の長さは語に含まれる記号の数である．語の集合を**言語**と呼ぶ．

有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ に対して，ある状態からある語に従って遷移を繰り返し最終的に到達する状態を与える関数 $\hat{\delta} : Q \times \Sigma^* \to Q$ を次のように定義する．

1. 任意の $q \in Q$ に対して $\hat{\delta}(q, \varepsilon) = q$
2. 任意の $q \in Q$，任意の語 $\boldsymbol{w} \in \Sigma^*$，任意の記号 $a \in \Sigma$ に対して $\hat{\delta}(q, \boldsymbol{w}a) = \delta(\hat{\delta}(q, \boldsymbol{w}), a)$

長さが 1 の記号列に対して $\hat{\delta}$ は δ と同じ結果を与えるので，以下では $\hat{\delta}$ も単に δ と書く．

有限オートマトン M において，$\delta(q_0, \boldsymbol{w})$ が F の要素であるならば，語 $\boldsymbol{w}$ は M によって受理されるという．M によって受理されるすべての語の集合を M の**受理言語**と呼び，$L(M)$ と書く．つまり，$L(M) = \{\boldsymbol{w} \mid \boldsymbol{w} \in \Sigma^* \wedge \delta(q_0, \boldsymbol{w}) \in F\}$ である．2 つの有限オートマトン M_1 と M_2 についてそれらの受理言語が等しい時，つまり $L(M_1) = L(M_2)$ の時 M_1 と M_2 は等しいという．

例 3.4 以下の有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ は整数を表す文字列を受理する有限オートマトンであ

る[*1].

$Q : \{S_0, S_1, S_2, S_3\}$
$\Sigma : \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
δ : 図 3.4
$q_0 : S_0$
$F : \{S_2, S_3\}$

□

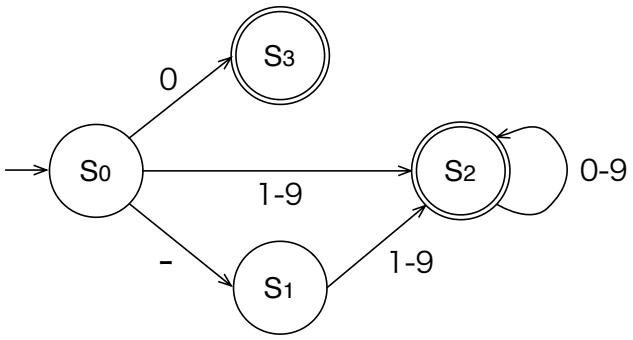


図 3.4　整数を表す文字列を受理する有限オートマトン

練習問題 3.5 例 3.4 の有限オートマトンを小数も受理できるように拡張せよ．

3.3　非決定性有限オートマトン

非決定性有限オートマトンは，ある状態から同じ記号で遷移する先の状態が複数存在する有限オートマトンであり，以下のように定義できる．

定義 3.6 (非決定性有限オートマトン) 非決定性有限オートマトン M は 5 項組 $(Q, \Sigma, \delta, q_0, F)$ で定義される．ここで，Q，Σ，q_0，F は定義 3.1 と同じである．状態遷移関数 δ は $Q \times \Sigma \to 2^Q$ として定義される．状態 $q \in Q$ と記号 $a \in \Sigma$ に対して $\delta(q, a)$ は状態 q で記号 a を読んだ場合に遷移先となる状態の集合を与える． □

非決定性有限オートマトンでは一つの記号列に対して複数の遷移系列が存在する可能性がある．このような時，最終状態に到達して終了する遷移系列が一つでもあればその記号列は受理されるとする．

[*1] 図 3.4 中の “1-9” のラベルが付いた遷移は，1 から 9 のラベルが付いた 9 本の遷移を 1 本にまとめて表したものである．“0-9” も同様．

例 3.7 以下に与える有限オートマトンは非決定性有限オートマトンである．

$$
\begin{aligned}
&Q : \{S_0, S_1, S_2\} \\
&\Sigma : \{a, b\} \\
&\delta : (S_0, a) \to \{S_0, S_1\},\ (S_0, b) \to \{S_0\},\ (S_1, a) \to \emptyset,\ (S_1, b) \to \{S_2\}, \\
&\quad (S_2, a) \to \emptyset,\ (S_2, b) \to \emptyset \\
&q_0 : S_0 \\
&F : \{S_2\}
\end{aligned}
$$

状態遷移図は図 3.5 のようになる．例えば，状態 S_0 において a が入力されると遷移先は S_0 と S_1 の 2 通りがある．入力される記号列が ab の場合，可能な遷移は $S_0 \to S_0 \to S_0$ と $S_0 \to S_1 \to S_2$ の 2 通りであり，後者が最終状態で終了しているので ab は受理される． □

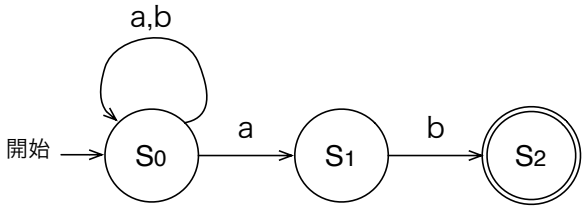


図 3.5　非決定性有限オートマトンの例

練習問題 3.8 例 3.7 の非決定性有限オートマトンが受理する言語の特徴を一言で述べよ．

練習問題 3.9 例 3.7 の非決定性有限オートマトンと同じ言語を受理する決定性有限オートマトンを作れ．

3.4　DFA と NFA の等価性

決定性有限オートマトンは非決定性有限オートマトンの特殊な場合であるので，非決定性有限オートマトンの方が表現能力が高いように思うかもしれない．しかし，実際には任意の非決定性有限オートマトンに対して等しい決定性有限オートマトンが存在する．

任意の非決定性有限オートマトン $M_N = (Q_N, \Sigma, \delta_N, q_0^N, F_N)$ に対し，$L(M_N) = L(M_D)$ である決定性有限オートマトン $M_D = (Q_D, \Sigma, \delta_D, q_0^D, F_D)$ を次のようにして構成できる．まず M_N の状態集合の部分集合を M_D の状態，つまり

$$Q_D = 2^{Q_N}$$

とする．M_D の初期状態を M_N の初期状態のみからなる集合，つまり

$$q_0^D = \{q_0^N\}$$

とする．Q_D の要素のうち F_N の要素を一つでも含むものを終了状態として，

$$F_D = \{q^D \mid q^D \in Q_D \wedge q^D \cap F_N \neq \emptyset\}$$

とする．状態遷移関数は，$q_1^N, \ldots, q_i^N \in Q_N$，$a \in \Sigma$ に対して

$$\delta_D(\{q_1^N, \ldots, q_i^N\}, a) = \bigcup_{j \in \{1,\ldots,i\}} \delta_N(q_j^N, a)$$

と定義する．状態 q^D から記号 a で遷移する先の状態は，q^D に含まれる M_N の状態から a で遷移できる状態の和集合である．この和集合は Q_N の部分集合であるため Q_D の元でもある．

例 3.10 例 3.7 の非決定性有限オートマトンと等しい決定性有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ は以下のように得られる．

$$\begin{aligned}
&Q : \{\emptyset, \{S_0\}, \{S_1\}, \{S_2\}, \{S_0, S_1\}, \{S_0, S_2\}, \{S_1, S_2\}, \{S_0, S_1, S_2\}\} \\
&\Sigma : \{a, b\} \\
&\delta : (\emptyset, a) \to \emptyset,\ (\emptyset, b) \to \emptyset,\ (\{S_0\}, a) \to \{S_0, S_1\},\ (\{S_0\}, b) \to \{S_0\}, \\
&\quad (\{S_1\}, a) \to \emptyset,\ (\{S_1\}, b) \to \{S_2\},\ (\{S_2\}, a) \to \emptyset,\ (\{S_2\}, b) \to \emptyset, \\
&\quad (\{S_0, S_1\}, a) \to \{S_0, S_1\},\ (\{S_0, S_1\}, b) \to \{S_0, S_2\}, \\
&\quad (\{S_0, S_2\}, a) \to \{S_0, S_1\},\ (\{S_0, S_2\}, b) \to \{S_0\}, \\
&\quad (\{S_1, S_2\}, a) \to \emptyset,\ (\{S_1, S_2\}, b) \to \{S_2\}, \\
&\quad (\{S_0, S_1, S_2\}, a) \to \{S_0, S_1\},\ (\{S_0, S_1, S_2\}, b) \to \{S_0, S_2\} \\
&q_0 : \{S_0\} \\
&F : \{\{S_2\}, \{S_0, S_2\}, \{S_1, S_2\}, \{S_0, S_1, S_2\}\}
\end{aligned}$$

元の非決定性有限オートマトンでは，S_0 から a による遷移先が S_0 と S_1 の 2 通りであったが，決定性有限オートマトン M では $\{S_0\}$ から a による遷移先が $\{S_0, S_1\}$ の一通りになっていることに注意する．

初期状態から到達できない状態は省略しても影響がないため，M を簡単化して $M' = (Q', \Sigma, \delta', q_0, F')$ が得られる．

$$\begin{aligned}
&Q' : \{\{S_0\}, \{S_0, S_1\}, \{S_0, S_2\}\} \\
&\delta' : (\{S_0\}, a) \to \{S_0, S_1\},\ (\{S_0\}, b) \to \{S_0\}, \\
&\quad (\{S_0, S_1\}, a) \to \{S_0, S_1\},\ (\{S_0, S_1\}, b) \to \{S_0, S_2\}, \\
&\quad (\{S_0, S_2\}, a) \to \{S_0, S_1\},\ (\{S_0, S_2\}, b) \to \{S_0\} \\
&F' : \{\{S_0, S_2\}\}
\end{aligned}$$

Q' は初期状態から到達可能な状態のみの集合になっている． □

練習問題 3.11 例 3.10 の有限オートマトン M と M' の状態遷移図を描き，M および M' が決定性有限オートマトンであること，M' は M の初期状態から到達不可能な状態を省略して得られることを確認せよ．

練習問題 3.12 例 3.10 の有限オートマトン M' が受理する言語の特徴を一言で述べよ．練習問題 3.8 の答えと比較せよ．

練習問題 3.13 練習問題 3.9 で作った決定性有限オートマトンと例 3.10 の決定性有限オートマトンを比較せよ．

3.5 正規表現

どのような言語に対しても，それを受理するような有限オートマトンが存在するわけではない．例えば，

$$L = \{(ab)^n \mid n \text{ は } 0 \text{ 以上の整数 }\} = \{\varepsilon, ab, abab, ababab, \cdots\}$$

なる言語 L を受理する有限オートマトンは図 3.6 のように存在するが，

$$L' = \{a^n b^n \mid n \text{ は } 0 \text{ 以上の整数 }\} = \{\varepsilon, ab, aabb, aaabbb, \cdots\}$$

なる言語 L' を受理する有限オートマトンは存在しない[*2]．

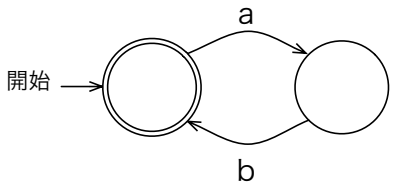


図 3.6　$\{(ab)^n \mid n$ は 0 以上の整数 $\}$ を受理する有限オートマトンの例

有限オートマトンの受理言語は**正規表現** (RE: Regular Expression) によって表すことができる．本節では正規表現について説明する．以降の節で，任意の正規表現に対してそれが表す言語を受理言語とする有限オートマトンを構成する方法と，任意の有限オートマトンに対しその受理言語を正規表現で表す方法を説明する．

正規表現の定義を以下に示す．

定義 3.14 (正規表現) 記号の集合 Σ 上の正規表現を次のように定義する．

1. 空列 ε，空集合 $\emptyset$，記号 $a \in \Sigma$ は正規表現である．
2. P, Q が正規表現ならば $(P + Q)$，$(P \cdot Q)$，(P^*) は正規表現である．
3. 上記の 2 つの規則を有限回適用して生成されるもののみが正規表現である．　□

ε と $\emptyset$ は異なることに注意する．ε は長さが 0 の語（つまり空列 ε）を受理する有限オートマトン[*3]の受理言語を表し，$\emptyset$ は何も受理しない有限オートマトン[*4]の受理言語を表す．$+$ は選択，$\cdot$ は連接，$*$ は繰り返し（閉包）を表す．結合の優先順位を

$$(1)\ *\quad (2)\ \cdot\quad (3)\ +$$

として括弧を適宜省略する．連接の $\cdot$ を省略して $a \cdot b$ を ab のように書くこともある．ε の連接も省略して $\varepsilon a = a\varepsilon = a$ とする．

定義 3.15 記号の集合 Σ 上の正規表現が表す語の集合を次のように定義する．

[*2] プッシュダウンオートマトンであれば受理できる

[*3] 初期状態が最終状態でもある有限オートマトン

[*4] 例えば，最終状態への遷移がない有限オートマトン

1. 正規表現 ε は集合 $\{\varepsilon\}$ を表す.
2. 正規表現 $\emptyset$ は空集合 $\emptyset$ を表す.
3. 正規表現 a は集合 $\{a\}$ を表す.
4. 正規表現 P, Q が表す語の集合を $\mathcal{P}, \mathcal{Q}$ とすると,
   - 正規表現 $P + Q$ は集合 $\mathcal{P} \cup \mathcal{Q} = \{w \mid w \in \mathcal{P} \vee w \in \mathcal{Q}\}$
   - 正規表現 $P \cdot Q$ は集合 $\mathcal{P} \cdot \mathcal{Q} = \{v \cdot w \mid v \in \mathcal{P} \wedge w \in \mathcal{Q}\}$
   - 正規表現 P^* は集合 $\bigcup_{n \geq 0} \mathcal{P}^{(n)}$

   を表す.

ここで，$v \cdot w$ は語 v に語 w を連結して得られる語を表す．また，$\bigcup_{n \geq 0} \mathcal{P}^{(n)}$ は $\{\varepsilon\} \cup \mathcal{P} \cup \mathcal{P} \cdot \mathcal{P} \cup \cdots$ の意味である．$\mathcal{P}^{(n)}$ は n 個の $\mathcal{P}$ の連接であり，$\mathcal{P}^{(0)} = \{\varepsilon\}$，$\mathcal{P}^{(n+1)} = \mathcal{P} \cdot \mathcal{P}^{(n)}$ と定義される． □

例 3.16 例 3.7 の非決定性有限オートマトンが受理する言語の正規表現は $(a + b)^* ab$ である． □

練習問題 3.17 $\Sigma = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ として，Σ^* に含まれる整数を表す文字列を表現する正規表現を書け.

練習問題 3.18 $\Sigma = \{., -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ として，Σ^* に含まれる小数を表す文字列を表現する正規表現を書け.

3.6 正規表現から有限オートマトンへの変換

任意の正規表現に対して，それが表す言語を受理言語とする有限オートマトンを構成することができる．具体的には，変換後の有限オートマトンを $(Q, \Sigma, \delta, q_0, F)$ とし，以下のように変換規則を正規表現の構造に関して再帰的に定義する．この方法で得られる有限オートマトンは非決定性有限オートマトンであるが，3.4 節の方法を併用すれば決定性有限オートマトンを得ることができる.

- 空列 ε：

  $Q : \{q\}$　　$\Sigma : \{\varepsilon\}$　　$\delta : -$　　$q_0 : q$　　$F : \{q\}$

- 空集合 $\emptyset$：

  $Q : \{q\}$　　$\Sigma : \emptyset$　　$\delta : -$　　$q_0 : q$　　$F : \emptyset$

- 記号 a：

  $Q : \{q, q'\}$　　$\Sigma : \{a\}$　　$\delta : (q, a) \to \{q'\}$　　$q_0 : q$　　$F : \{q'\}$

- 選択 $A + B$：
  A, B を変換して得られる有限オートマトンをそれぞれ $(Q_A, \Sigma_A, \delta_A, q_A, F_A), (Q_B, \Sigma_B, \delta_B, q_B, F_B)$ と

する．

$$
\begin{aligned}
&Q : Q_A \cup Q_B \cup \{q_{new}\} \\
&\Sigma : \Sigma_A \cup \Sigma_B \\
&\delta : \text{任意の } a \in \Sigma \text{ に対して } (q,a) \to \begin{cases} \delta_A(q_A,a) \cup \delta_B(q_B,a) & \text{if } q = q_{new} \\ \delta_A(q,a) & \text{if } q \in Q_A \\ \delta_B(q,a) & \text{if } q \in Q_B \end{cases} \\
&q_0 : q_{new} \\
&F : \begin{cases} F_A \cup F_B \cup \{q_{new}\} & \text{if } q_A \in F_A \vee q_B \in F_B \\ F_A \cup F_B & \text{otherwise} \end{cases}
\end{aligned}
$$

- 連接 $A \cdot B$：
  A, B を変換して得られる有限オートマトンをそれぞれ $(Q_A, \Sigma_A, \delta_A, q_A, F_A), (Q_B, \Sigma_B, \delta_B, q_B, F_B)$ とする．

$$
\begin{aligned}
&Q : Q_A \cup Q_B \\
&\Sigma : \Sigma_A \cup \Sigma_B \\
&\delta : \text{任意の } a \in \Sigma \text{ に対して } (q,a) \to \begin{cases} \delta_A(q,a) & \text{if } q \in Q_A - F_A \\ \delta_B(q_B,a) & \text{if } q \in F_A \\ \delta_B(q,a) & \text{if } q \in Q_B \end{cases} \\
&q_0 : q_A \\
&F : \begin{cases} F_A \cup F_B & \text{if } q_B \in F_B \\ F_B & \text{if } q_B \notin F_B \end{cases}
\end{aligned}
$$

- 閉包 A^*：
  A を変換して得られる有限オートマトンをそれぞれ $(Q_A, \Sigma_A, \delta_A, q_A, F_A)$ とする．

$$
\begin{aligned}
&Q : Q_A \\
&\Sigma : \Sigma_A \\
&\delta : \text{任意の } a \in \Sigma \text{ に対して } (q,a) \to \begin{cases} \delta(q,a) & \text{if } q \in Q_A - F_A \\ \delta(q_A,a) & \text{if } q \in F_A \end{cases} \\
&q_0 : q_A \\
&F : F_A \cup \{q_A\}
\end{aligned}
$$

正規表現 A, B を変換して得られる有限オートマトンをそれぞれ M_A, M_B とする．選択 $A + B$ の場合，直観的には M_A の初期状態と M_B の初期状態を同一視できればよい．そこで，新しい初期状態 q_{new} を導入し，M_A と M_B それぞれの初期状態からの遷移と同じ遷移を q_{new} からの遷移に加える．M_A と M_B のいずれか一方か両方で初期状態が最終状態でもあるならば q_{new} を最終状態に追加する．連接 $A \cdot B$ の場合，M_A の最終状態を M_B の初期状態とみなせばよい．そこで，M_B の初期状態からの遷移と同じ遷移を M_A の最終状態からの遷移に追加する．もし M_B の初期状態が最終状態ならば M_A の最終状態も全体の最終状態とする．閉包 A^* の場合，M_A の最終状態を初期状態とみなせば繰り返しを表現できる．そこで，初期状態からの遷移と同じ遷移を最終状態からの遷移として新たに追加する．閉包は空列も表現するため，初期状態を最終状態に追加する．

図 3.7 に状態遷移図を用いて変換規則を示す．点線で表される遷移が存在する場合に太線の遷移が追加される．

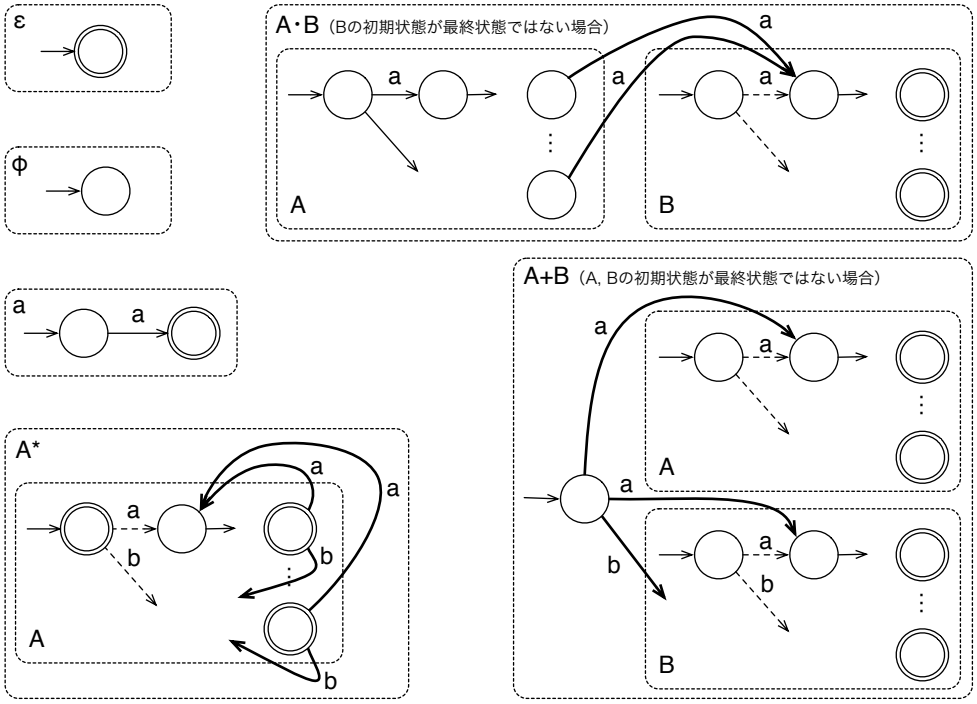


図 3.7　正規表現から有限オートマトンへの変換

例 3.19 この変換規則によって得られる，正規表現 $(a+b)^*ab$ を受理言語とする非決定性有限オートマトンを図 3.8 の左側に示す．ただし，初期状態から到達不能な状態は除去されている．この有限オートマトンの左側 3 つの状態を同一視して簡単化すると，図の右側の有限オートマトンが得られる．　□

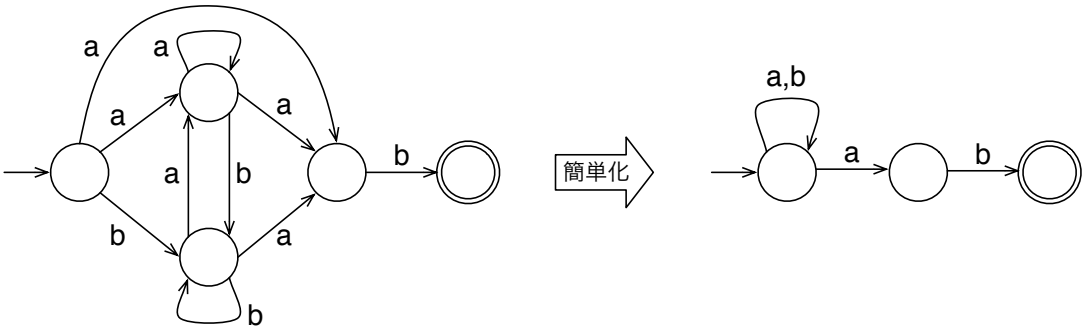


図 3.8　正規表現 $(a+b)^*ab$ を受理言語とする有限オートマトン

練習問題 3.20 例 3.19 を自分でやってみよ．

練習問題 3.21 練習問題 3.17 の正規表現を受理言語とする有限オートマトンを作れ．

練習問題 3.22 練習問題 3.18 の正規表現を受理言語とする有限オートマトンを作れ．

3.7 有限オートマトンから正規表現への変換

任意の決定性有限オートマトンに対して，その受理言語を正規表現で表すことができる．決定性有限オートマトンからその受理言語の正規表現を得るための手続きを以下に示す．非決定性有限オートマトンについては，3.4 節の方法で決定性有限オートマトンに変換すればよい．

決定性有限オートマトン M を $M = (Q, \Sigma, \delta, q_0, F)$ とする．ただし，$Q = \{1, \ldots, n\}$，$q_0 = 1$ とする．ここで，記号列の集合 R_{ij}^k を次のように定義する．

$$R_{ij}^0 = \begin{cases} \{a \mid \delta(i,a) = j\} & \text{if } i \neq j \\ \{a \mid \delta(i,a) = j\} \cup \{\varepsilon\} & \text{if } i = j \end{cases}$$
$$R_{ij}^k = R_{ik}^{k-1} \cdot (R_{kk}^{k-1})^* \cdot R_{kj}^{k-1} \cup R_{ij}^{k-1} \quad (k \geq 1)$$

記号列の集合 S, T に対し，$S \cdot T = \{s \cdot t \mid s \in S \wedge t \in T\}$，$S^* = \emptyset \cup S \cup S \cdot S \cup \ldots$ である．R_{ij}^k は状態 i から j へ途中で k 以下の状態のみを通過して遷移できる入力記号列の集合を表す．両端の i や j は k より大きくてもよい．状態数は n であるので R_{ij}^n が状態 i から j へ遷移する入力記号列全体を表す．

R_{ij}^0 は状態 i から j へ直接遷移できる入力記号列の集合である．$i = j$ の場合は遷移しない入力記号 ε も含まれることに注意する．R_{ij}^k の定義は以下のように考えるとよい．k 以下の状態を通過する遷移は，状態 k を通過する遷移としない遷移にわけられる．$R_{ik}^{k-1} \cdot (R_{kk}^{k-1})^* \cdot R_{kj}^{k-1}$ は状態 k を通過する遷移の入力記号列の集合を表す．R_{ik}^{k-1} は状態 i を出発して $k-1$ 以下の状態を通過して初めて状態 k に到達するまでの入力記号列の集合を表し，R_{kk}^{k-1} は状態 k から再び k に到達するまで入力記号列の集合を表し，R_{kj}^{k-1} は最後に状態 k を出発して j に到達するまでの入力記号列の集合を表す．状態 k を通過しない場合は $k-1$ 以下の状態のみを通過することになるため，その場合の入力記号列の集合は R_{ij}^{k-1} と表される．

n 以下の任意の i, j, k に対して R_{ij}^k が正規表現によって表されることを k に関する帰納法によって証明する．

基底段階 $k = 0$ の場合，正規表現として $i \neq j$ ならば $a_1 + \cdots + a_l$，$i = j$ ならば $a_1 + \cdots + a_l + \varepsilon$ を考えればよい．ここで，$\{a_1, \ldots, a_l\}$ を $\delta(i,a) = j$ を満たす a の集合とする．$\delta(i,a) = j$ なる a が存在しない場合は，$i \neq j$ ならば $\emptyset$，$i = j$ ならば ε とする．

帰納段階 帰納法の仮定より，任意の l, m について R_{lm}^{k-1} を表す正規表現 r_{lm}^{k-1} が存在する．そこで，R_{ij}^k の正規表現として $(r_{ik}^{k-1})(r_{kk}^{k-1})^*(r_{kj}^{k-1}) + r_{ij}^{k-1}$ をとればよい．

M において初期状態から最終状態へと遷移する入力記号列の集合（つまり受理言語）は $\bigcup_{f \in F} R_{1f}^n$ と表される．よって，$F = \{f_1, \ldots, f_l\}$ とすると受理言語の正規表現は $r_{1f_1}^n + \cdots + r_{1f_l}^n$ として得られる．

例 3.23 決定性有限オートマトン $M = (\{q_1, q_2, q_3\}, \{a, b\}, \delta, q_1, \{q_3\})$ の受理言語の正規表現は次のようにして得られる．ここで，遷移関数 δ は

$$\delta : (q_1, a) \to q_2,\ (q_1, b) \to q_1,\ (q_2, a) \to q_2,\ (q_2, b) \to q_3,\ (q_3, a) \to q_2,\ (q_3, b) \to q_1$$

のように定義されるとする．

M の状態数は 3 で最終状態は q_3 だけなので，記号列集合 R^3_{13} に対応する正規表現 r^3_{13} を求めればよい．r^3_{13} を求めると

$$
\begin{aligned}
r^1_{12} &= r^0_{11}(r^0_{11})^* r^0_{12} + r^0_{12} \\
&= (b+\varepsilon)(b+\varepsilon)^* a + a \\
&= b^* a \\
r^1_{22} &= r^0_{21}(r^0_{11})^* r^0_{12} + r^0_{22} \\
&= a + \varepsilon \\
r^1_{23} &= r^0_{21}(r^0_{11})^* r^0_{13} + r^0_{23} \\
&= b \\
r^1_{13} &= r^0_{11}(r^0_{11})^* r^0_{13} + r^0_{13} \\
&= \emptyset \\
r^2_{13} &= r^1_{12}(r^1_{22})^* r^1_{23} + r^1_{13} \\
&= (b^* a)(a+\varepsilon)^* b + \emptyset \\
&= b^* a a^* b \\
r^3_{13} &= r^2_{13}(r^2_{33})^* r^2_{33} + r^2_{13} \\
&= b^* a a^* b (b^* a a^* b + \varepsilon)^* (b^* a a^* b + \varepsilon) + b^* a a^* b \\
&= (b^* a a^* b)^* b^* a a^* b
\end{aligned}
\qquad
\begin{aligned}
r^1_{32} &= r^0_{31}(r^0_{11})^* r^0_{12} + r^0_{32} \\
&= b(b+\varepsilon)^* a + a \\
&= b^* a \\
\\
\\
\\
r^1_{33} &= r^0_{31}(r^0_{11})^* r^0_{13} + r^0_{33} \\
&= \varepsilon \\
r^2_{33} &= r^1_{32}(r^1_{22})^* r^1_{23} + r^1_{33} \\
&= (b^* a)(a+\varepsilon)^* b + \varepsilon \\
&= b^* a a^* b + \varepsilon
\end{aligned}
$$

となり，$(b^* a a^* b)^* b^* a a^* b$ が M の受理言語の正規表現である． □

練習問題 3.24 例 3.4 の有限オートマトンの受理言語を表す正規表現を作れ．

練習問題 3.25 練習問題 3.5 の有限オートマトンの受理言語を表す正規表現を作れ．

練習問題 3.26 状態数が 3 程度の有限オートマトンを適当に定義し，その受理言語を表す正規表現を作れ．

3.8 反復補題

3.5 節で有限オートマトンで受理できる言語とできない言語の例を示した．では，ある言語を与えられた時にその言語が適当な有限オートマトンで受理できるかどうか判定する方法はあるだろうか．その答えは肯定的に与えられるが，本節ではその判定に有用な**反復補題**（ポンピング定理）を説明する．

有限オートマトンの状態数は有限であるため，もし受理される語の長さが状態数よりも大きければ，その語による初期状態から最終状態までの遷移のどこかに複数回到達する状態が存在し，そのためにループが生じているはずである．この時，一つのループに着目すると状態遷移図は図 3.9 のようになる．中央の状態が複数回到達する状態である．

図 3.9 から直観的に明らかであるが，ループを通過しない遷移やループを 2 回以上通過する遷移となるような語も同じ有限オートマトンで受理される．つまり，ある有限オートマトンが受理する語の中に十分に長い語が含まれる場合，その語の初めの方にある部分記号列を何度も繰り返すことで，受理される言語をいくらでも長くすることができる．このことを正確に述べたのが反復補題である．

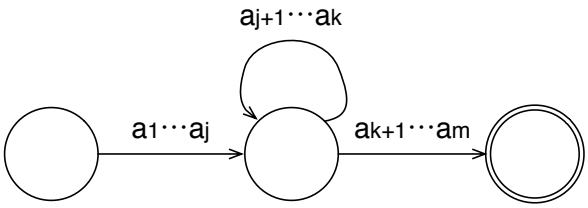


図 3.9　ループを含む状態遷移図

補題 3.27 (反復補題) 正規言語[*5] L に対して以下を満たす定数 n が存在する．すなわち，$\boldsymbol{z} \in L$ として $|\boldsymbol{z}| \geq n$ ならば，$|\boldsymbol{uv}| \leq n$ および $|\boldsymbol{v}| \geq 1$ を満たす語 $\boldsymbol{u}$，$\boldsymbol{v}$ と適当な語 $\boldsymbol{w}$ を用いて $\boldsymbol{z} = \boldsymbol{uvw}$ とできて，さらに 0 以上の任意の i について $\boldsymbol{uv}^i\boldsymbol{w} \in L$ である．ここで，n は L を受理する状態数が最小の有限オートマトンの状態数より大きくない． □

反復補題から，正規言語 L が十分に長い語を含むならば，その語をもとにして作られる $\boldsymbol{uv}^i\boldsymbol{w}$ の形の語も L に含まれることがわかる．ただし，L に含まれる十分に長い語がすべて $\boldsymbol{uv}^i\boldsymbol{w}$ の形をしているわけではないことに注意する．

反復補題を利用して，与えられた言語 L が正規言語か否か判定することができる．適当な n に対して $|\boldsymbol{z}| \geq n$ なる語 $\boldsymbol{z} \in L$ を見つけ，$\boldsymbol{z} = \boldsymbol{uvw}$，$|\boldsymbol{uv}| \leq n$，$|\boldsymbol{v}| \geq 1$ となる適当な語 $\boldsymbol{u}$，$\boldsymbol{v}$，$\boldsymbol{w}$ に対して $\boldsymbol{uv}^i\boldsymbol{w} \notin L$ となるような i が存在すれば L は正規言語ではない．すなわち，正規言語は反復補題を満たし，反復補題を満たさない言語は正規言語ではない．

例 3.28 3.5 節で挙げた言語

$$L' = \{a^l b^l \mid l \text{ は } 0 \text{ 以上の整数 }\} = \{\varepsilon, ab, aabb, aaabbb, \cdots\}$$

が正規言語でないことは次のように示すことができる．

L' が正規言語であると仮定すると反復補題が成り立つ．つまり，適当な定数 n が存在して，$|a^l b^l| \geq n$ となる l を選べば，反復補題の条件を満たす適当な語 $\boldsymbol{u}$，$\boldsymbol{v}$，$\boldsymbol{w}$ を用いて $a^l b^l = \boldsymbol{uvw}$ と分解できる．また，$\boldsymbol{uv}^2\boldsymbol{w} \in L'$ である．ここで，$a^l b^l$ の分解の方法は以下のいずれかである．

(i)　$\boldsymbol{uvw} = a^i a^j a^{k_1} b^{k_2}$　$(j \geq 1,\ k_1 \geq 0,\ i + j + k_1 = k_2 = l)$
(ii)　$\boldsymbol{uvw} = a^i a^{j_1} b^{j_2} b^k$　$(j_1 \geq 1,\ j_2 \geq 1,\ i + j_1 = j_2 + k = l)$
(iii)　$\boldsymbol{uvw} = a^{i_1} b^{i_2} b^j b^k$　$(j \geq 1,\ i_2 \geq 0,\ i_1 = i_2 + j + k = l)$

(i) の場合，$\boldsymbol{uv}^2\boldsymbol{w} = a^i a^{2j} a^{k_1} b^{k_2}$ であるが，$j \geq 1$ かつ $i+j+k_1 = k_2$ より $i+2j+k_1 \neq k_2$ ゆえ $a^i a^{2j} a^{k_1} b^{k_2} \notin L'$．これは $\boldsymbol{uv}^2\boldsymbol{w} \in L'$ に矛盾する．(iii) の場合も同様である．(ii) の場合，$\boldsymbol{uv}^2\boldsymbol{w} = a^i a^{j_1} b^{j_2} a^{j_1} b^{j_2} b^k$ であるが，これは明らかに L' の語ではない．以上より，いずれの場合も $\boldsymbol{uv}^2\boldsymbol{w} \in L'$ に矛盾するため L' は正規言語ではない． □

[*5] 有限オートマトンで受理可能な言語

第 4 章

チューリング機械

本章については，別途資料を配付する．

第 5 章

命題論理

5.1　数理論理学

人間にとって，何かしらの命題（言明）を表明したり，得られた命題から別の命題を推論したりすることは日常的な行為であり，半ば無意識のうちに行っている．命題は，真であるか偽であるか決めることができる「何か」を述べたものである．通常，自らが命題を表明する際はその命題が真であると信じており，他の人が表明した命題に対してはその真偽を評価する．

命題は独立して存在するだけでなく，一方の命題が真であれば必ず他方も真である，のように命題同士で関係し合うこともある．例えば，命題の集合

1. 人間ならば死ぬ
2. ソクラテスは人間である
3. ソクラテスは死ぬ

を考えると，1. と 2. の命題が真であれば 3. の命題は必ず真である．別の集合

1. 人間ならば死ぬ
2. ソクラテスは人間である
3. ソクラテスは死なない

を考えると，3 つの命題が同時に真になることはない．このような命題同士の関係は直観的には理解できる．

論理学は，命題の集まりについて，ある命題（の真偽）が他の命題（の真偽）にどのように影響を与えるか，命題間の関連を系統的に調べる学問である．そして，その道具として数学における形式手法，記号的手法を利用し，論理学を数学的に扱おうとするのが**数理論理学**である．

論理は計算機科学の基礎となっている．計算機が扱う情報は 0 と 1 の組み合わせであり，情報の処理は 0 と 1 に対する処理の組み合わせである．0 と 1 をそれぞれ偽と真とみなせば，0 と 1 に対する処理は「かつ」，「または」，「ならば」，「…でない」のいずれかかそれらの組み合わせとして表すことができる．結局は真偽の組み合わせの計算であり，すなわちそれは論理である．論理の具体的な応用例としては

- 論理回路，順序回路
- （論理）プログラミング
- モデル検査

- 定理自動証明

などがある．

論理には，命題の関係のみを扱う**命題論理** (propositional logic)，対象と述語を扱う**述語論理** (predicate logic)，可能性や必然性を扱う**様相論理** (modal logic)，時系列を考慮する**時相論理** (temporal logic) などがある．本章では，もっとも基本的な命題論理について解説する．

5.2 命題

命題 (proposition) は内容の真偽が確定できる文のことである．例えば

- 5 は 2 より大きい
- 1 と 10 は等しい
- ポチは犬であるかまたは猫である

などは命題である．一方，

- 明日は晴れますか？
- 明日は晴れるといいなあ
- この定理は証明できるかもしれない
- この定理を証明しなさい

といった疑問，希望，命令などの文は命題ではない．

5.3 論理式

命題論理は，命題の真偽に関する論証を最も基本的な命題の真偽と命題の組み合わせ構造のみに着目して行う体系である．最も基本的な命題と命題の組み合わせ構造を記号を用いて表し，記号列に対する操作として機械的に論証を行う．命題を表す記号列を**論理式** (logical formula) と呼ぶ．論証を行う際は，命題の具体的な内容は無視する．

論理式には以下の記号が用いられる．

- 論理記号： $\wedge, \vee, \rightarrow, \neg$
- 非論理記号： $p, q, r, \ldots$
- 補助記号： $(,)$

論理記号（論理演算子，logical operator，結合子，logical connective）は既存の命題を組み合わせて新しい命題を構成する記号である．非論理記号（**命題変数**，proposition variable）は命題，特に最も基本的な命題（**原子命題**，atomic proposition）を表す記号である．以降では，p, q, r などの小文字アルファベットや p_0, p_1, p_2 のように添字付きの小文字アルファベットを用いて命題変数を表す．

定義 5.1 (論理式) 論理式を以下のように帰納的に定義する．

1. 命題変数 p は論理式である．この形の論理式を基本論理式，原子論理式と呼ぶ．

2. P, Q が論理式ならば
   - $(P \wedge Q)$
   - $(P \vee Q)$
   - $(\neg P)$
   - $(P \to Q)$

   は論理式である．
3. 1. と 2. で論理式であるものだけが論理式である． □

$(P \wedge Q)$ は P と Q の連言 (conjunction) であり「P かつ Q」と読む．$(P \vee Q)$ は P と Q の選言 (disjunction) であり「P または Q」と読む．$(\neg P)$ は P の否定 (negation) であり「P でない」と読む．$(P \to Q)$ は P と Q の含意 (implication) であり「P ならば Q」と読む．

論理記号の間の優先度を

$$(1)\ \neg \quad (2)\ \wedge \quad (3)\ \vee \quad (4)\ \to$$

と決める．また，$\wedge$ と $\vee$ は左結合，$\to$ は右結合であるとする．こうすることで，一部の括弧を省略して論理式を記述できる．例えば，$p \wedge \neg q \wedge r \vee s$ は $(((p \wedge (\neg q)) \wedge r) \vee s)$ の意味である．以降では，P, Q, R などの大文字アルファベットや P_0, P_1, P_2 のような添字付きの大文字アルファベットを用いて論理式を表す．

論理式の**意味**（真偽値，真理値）は，原子命題を表す命題変数の真理値と論理記号の意味によって定まる．命題変数に真理値 **true**（真）あるいは **false**（偽）を割り当てる関数を**解釈**と呼ぶ．

定義 5.2 (解釈) Σ を命題変数の集合とする．解釈 I は Σ に含まれる各命題変数 p に真理値 **true** あるいは **false** を割り当てる関数

$$I : \Sigma \to \{\mathbf{true}, \mathbf{false}\}$$

である． □

命題変数 $p \in \Sigma$ に対し $p \in \mathrm{dom}(I)$ ならば $I(p) = \mathbf{true}$ または $I(p) = \mathbf{false}$ である．

論理記号 $\neg, \wedge, \vee, \to$ の意味はそれぞれ表 5.1 のように定義される真理値関数 Not, And, Or, Imp として定義できる．ここで，真理値関数は真理値集合 $\{\mathbf{true}, \mathbf{false}\}$ 上の関数である．

表 5.1　真理値関数

P	$\mathrm{Not}(P)$
true	**false**
false	**true**

P	Q	$\mathrm{And}(P,Q)$	$\mathrm{Or}(P,Q)$	$\mathrm{Imp}(P,Q)$
true	**true**	**true**	**true**	**true**
true	**false**	**false**	**true**	**false**
false	**true**	**false**	**true**	**true**
false	**false**	**false**	**false**	**true**

解釈と 4 つの真理値関数を用いて論理式の意味を定義する．

定義 5.3 (論理式の意味) Σ を命題変数の集合とする．解釈 I のもとでの論理式 P の真理値 $V_I(P)$ を

$$V_I(P) = \begin{cases} I(p) & P \text{ が命題変数 } p \in \Sigma \text{ の場合} \\ \mathrm{Not}(V_I(Q)) & P \text{ が } \neg Q \text{ の場合} \\ \mathrm{And}(V_I(Q), V_I(R)) & P \text{ が } Q \wedge R \text{ の場合} \\ \mathrm{Or}(V_I(Q), V_I(R)) & P \text{ が } Q \vee R \text{ の場合} \\ \mathrm{Imp}(V_I(Q), V_I(R)) & P \text{ が } Q \to R \text{ の場合} \end{cases}$$

と定義する． □

以降では，$V_I(P)$ を $I(P)$ と記述する場合がある．

5.4 論理的帰結

命題同士の関係の一つに**論理的帰結**がある．命題の集合 Φ と命題 P に対して，Φ に含まれるすべての命題が真である場合に必ず P が真になる時，P は Φ の論理的帰結であるという．例えば，r は $\{p \to q, q \to r, p\}$ の論理的帰結である．命題の集合が与えられた時にそれが論理的帰結の関係にあることを証明することは，命題論理における基本的な問題の一つである．

定義 5.4 (モデル) Φ を論理式の集合，I を解釈とする．Φ に含まれるすべての論理式 $P \in \Phi$ に対して $I(P) = \mathbf{true}$ である時，かつその時に限り I は Φ の**モデル** (model) であるといい，$\models_I \Phi$ と書く．特に $\Phi = \{P\}$ である時は $\models_I P$ とも書く． □

定義 5.5 (論理的帰結) Φ を論理式の集合，P を論理式とする．すべての解釈 I に対して $\models_I \Phi$ ならば $\models_I P$ である時，かつその時に限り P は Φ の**論理的帰結** (logical consequence) であるといい，$\Phi \models P$ と書く．特に $\Phi = \emptyset$ である時は $\models P$ とも書く． □

$\{P_1, \ldots, P_n\} \models Q$ について集合の括弧を省略して $P_1, \ldots, P_n \models Q$ と書くこともある．論理的帰結と含意の関係として $P \models Q \iff \models P \to Q$ が成り立つ．

命題 5.6 Φ を論理式の集合，P, Q を論理式とする．

1. $P \models Q \iff \models P \to Q$
2. $\Phi \cup \{P\} \models Q \iff \Phi \models P \to Q$（1. の一般化） □

定義 5.7 (トートロジー) P を論理式とする．すべての解釈 I に対して $\models_I P$ である時，かつその時に限り P は**トートロジー** (tautology) または**恒真式**であるという． □

P がトートロジーであることと $\models P$ は同値である．

定義 5.8 (充足可能) 論理式の集合 Φ（または論理式 P）に対してモデルが存在する時 Φ（または P）は**充足可能** (satisfiable) である．そのようなモデルが存在しなければ Φ（または P）は**充足不能** (unsatisfiable) である． □

定義 5.9 充足不能な論理式を**恒偽式**という．P を論理式とし，すべての解釈 I に対して $\not\models_I P$ ならば P は恒偽式である． □

与えられた論理式 P がトートロジーか否か判定するには，その論理式に出現する命題変数の集合に関するすべての解釈について $\models_I P$ であるか調べればよい．可能な解釈の数は有限であるため調べあげることは可能である．しかし，もし命題変数が 10 個あれば解釈は $2^{10} = 1024$ 通りあるため，可能な解釈のすべてについて $\models_I P$ であることを調べるのは効率が悪い．同様に，論理式 P が論理式の集合 Φ の論理的帰結であることも，すべての解釈について $\models_I \Phi$ ならば $\models_I P$ であることを調べれば判定できるが，やはり効率が悪い．しかし，以降で解説する証明系を利用すると効率よく判定することができる．

練習問題 5.10 $\{p \to q, q \to r, p\}$ のモデルを 1 つ示せ．

練習問題 5.11 r が $\{p \to q, q \to r, p\}$ の論理的帰結であることを確認せよ．

練習問題 5.12 命題 5.6 を証明せよ．(1. のヒント：$P \models Q \implies \models P \to Q$ と $\models P \to Q \implies P \models Q$ を示せばよい．)

練習問題 5.13 $p \to (q \to p)$ がトートロジーであることを確認せよ．

5.5 証明系

証明系は，論理式が論理式集合の論理的帰結であることを，論理式（の列）に対する機械的な操作のみによって調べる方法である．この時，論理式を構成する記号列のみに着目し，論理式の意味は一切考慮しないという特徴がある．

証明系には大きく分けて演繹系と反駁系の 2 種類がある．演繹系は，前提となる論理式集合から結論の論理式が導かれるまで推論を繰り返す証明系である．$\mathcal{LK}$ や以下で取り上げる**自然演繹** (natural deduction) などがある．反駁系は，前提の論理式集合に結論の否定を加えて推論を進め，否定的な結果が得られたら証明が成功としたとする証明系であり，分解証明系などがある．

演繹系において，推論は規則に従って行われる．演繹系は，どのような論理式（集合）からどのような論理式を導き出すことができるか定めた規則の集合として定義される．この規則に従ってある論理式（集合）から新たな論理式を得ることが推論である．規則は推論規則と呼ばれており，規則群に含まれる推論規則の違いによって $\mathcal{LK}$ や自然演繹といった異なる証明系が構成される．

前提（前件，premise）と呼ばれる論理式集合 $\{P_1, \ldots, P_n\}$ と，**結論**（後件，conclusion）と呼ばれる論理式 Q があるとする．$\{P_1, \ldots, P_n\}$ から推論を開始し，繰り返し推論を行って Q を得ることができれば

$$\{P_1, \ldots, P_n\} \vdash Q$$

あるいは集合の括弧を省略して

$$P_1, \ldots, P_n \vdash Q$$

と書く．この形の式を**シークェント**と呼び，推論を繰り返す過程を**証明**と呼ぶ．シークェントによっては証明が一通りでないことに注意する．特に証明系 PS によってシークェントが証明できることを明示する場合は $P_1, \ldots, P_n \vdash_{PS} Q$ と書く．

証明系は論理式が論理式集合の論理的帰結であることを調べるための手段として使えることが意図されている．そのため，証明系には

1. 証明できたことは必ず正しいこと

2. 正しいことは必ず証明できること

の 2 つが期待される．この場合の「正しいこと」は「論理的帰結であること」である．1. を証明系の**健全性** (soundness) といい，2. を証明系の**完全性** (completeness) という．

一般的に証明系の健全性と完全性は次のように定義される．

定義 5.14 (健全性) 証明系 PS は，任意の論理式集合 Φ と論理式 P に対し

$$\Phi \vdash_{PS} P \quad \text{ならば} \quad \Phi \models P$$

である時，かつその時に限り**健全**である． □

定義 5.15 (完全性) 証明系 PS は，任意の論理式集合 Φ と論理式 P に対し

$$\Phi \models P \quad \text{ならば} \quad \Phi \vdash_{PS} P$$

である時，かつその時に限り**完全**である． □

証明系が健全でないとは，その証明系で証明できた事柄が正しくない場合があるということである．例えば，$p \to \neg q, p \models q$ ではないため，$p \to \neg q, p \vdash q$ が証明できてしまう証明系は健全ではない．このような証明系は証明系のそもそもの目的に反している．また，証明系が完全でないとは，正しいにもかかわらず証明できない事柄があるということである．例えば，$p \to \neg q, p \vdash \neg q$ が証明できない証明系は完全ではない．このように健全でない証明系や完全でない証明系は正しく証明が行えないケースがあって有用性に欠けるといえる．健全性と完全性は証明系の重要な性質である．

5.6 自然演繹

自然演繹と呼ばれる証明系の推論規則を説明する．以降では自然演繹のみを扱うため，$P_1, \ldots, P_n \vdash Q$ は自然演繹で証明できることを表す $P_1, \ldots, P_n \vdash_{ND} Q$ の意味であるとする．

連言に関する推論規則は

$$\frac{P \quad Q}{P \wedge Q} \wedge\mathrm{i} \qquad \frac{P \wedge Q}{P} \wedge\mathrm{e}_1 \qquad \frac{P \wedge Q}{Q} \wedge\mathrm{e}_2$$

である．$\wedge$i は $\wedge$-導入 (and-introduction) であり，P と Q が成り立っているならば，その事実から $P \wedge Q$ が成り立つことを推論できることを表している．$\wedge\mathrm{e}_1$ と $\wedge\mathrm{e}_2$ は $\wedge$-除去 (and-elimination) であり，$P \wedge Q$ が成り立っているならば，その事実から P が成り立つこと ($\wedge\mathrm{e}_1$)，および Q が成り立つこと ($\wedge\mathrm{e}_2$) を推論できることを表す．

例 5.16 $p \wedge q \vdash q \wedge p$ は以下のように証明できる．

$$\dfrac{\dfrac{p \wedge q}{q} \wedge\mathrm{e}_2 \quad \dfrac{p \wedge q}{p} \wedge\mathrm{e}_1}{q \wedge p} \wedge\mathrm{i}$$

$\wedge$-除去により q と p がそれぞれ推論できるので，さらに $\wedge$-導入によって $q \wedge p$ が推論できる． □

シークェントの具体的な証明は例 5.16 のように木構造として表現される．証明を表す木構造を**証明木**と呼ぶこともある．証明木の節点は論理式でラベル付けされており，根は結論の論理式，葉は前提の論理式集合に

含まれる論理式である．また，証明木のすべての部分木について，親節点の論理式は子節点の論理式に推論規則を適用して得られる論理式である．適用した推論規則の名前を枝に記すこともある．

二重否定 (double negation) に関する推論規則は

$$\frac{P}{\neg\neg P}\,\neg\neg\mathrm{i} \qquad \frac{\neg\neg P}{P}\,\neg\neg\mathrm{e}$$

である．否定の否定が元と同じであるというのは直観的に納得できる[*1]．¬¬-導入は他の推論規則を用いて導出することが可能である．

含意に関する推論規則は

$$\frac{\begin{array}{c}[P]\\ \vdots\\ Q\end{array}}{P \to Q}\,\to\mathrm{i} \qquad \frac{P \quad P \to Q}{Q}\,\to\mathrm{e}$$

である．→-導入は，P を前提として Q が成り立っていれば，そこから $P \to Q$ が成り立つと推論できるということを述べている．$[P]$ は，Q の証明に必要な前提 P を $P \to Q$ の導出と同時にもはや前提として扱わないことを表し，$[\cdot]$ を付す操作を打ち消し (discharge) と呼ぶ．以下では，打ち消された（$[\cdot]$ で囲まれた）論理式を仮定の論理式と呼ぶ場合もある．→-除去はいわゆる**三段論法**である．

例 5.17 $p \to (q \to r) \vdash p \wedge q \to r$ は以下のように証明できる．

$$\dfrac{\dfrac{\dfrac{[p \wedge q]}{q}\,\wedge\mathrm{e}_2 \quad \dfrac{\dfrac{[p \wedge q]}{p}\,\wedge\mathrm{e}_1 \qquad p \to (q \to r)}{q \to r}\,\to\mathrm{e}}{r}\,\to\mathrm{e}}{p \wedge q \to r}\,\to\mathrm{i}$$

□

証明木の葉は打ち消された論理式の場合もある．その場合，すべての打ち消された論理式に対応するように，打ち消しを行う →-導入などの推論規則が適用されていなければならない．また，上の例では $p \to (q \to r)$ と $p \wedge q$ から r を推論しているが，この例のように打ち消された論理式が複数の葉に出現してもよい．

例 5.18 $p \wedge q \to r \vdash p \to (q \to r)$ は以下のように証明できる．

$$\dfrac{\dfrac{\dfrac{\dfrac{[p]_1 \quad [q]_2}{p \wedge q}\,\wedge\mathrm{i} \qquad p \wedge q \to r}{r}\,\to\mathrm{e}}{q \to r}\,\to\mathrm{i},2}{p \to (q \to r)}\,\to\mathrm{i},1$$

□

[*1] ここでは二重否定の除去（¬¬-除去）を認める**古典論理**を取り上げて説明するが，二重否定の除去を認めない**直観主義論理**もある．直観主義論理ではすぐ後で導入する背理法や排中律が成り立たない．

この例では，打ち消された論理式が 2 種類ありそれぞれが異なる推論規則の適用に対応している．そこで，それぞれがどの推論規則の適用に対応するか明示するため，$[p]_1$ のように打ち消された論理式に添字を付け，$\to \mathrm{i}, 1$ のように適用した推論規則名と一緒に打ち消された論理式の添字を示す．

例 5.17 と例 5.18 から $p \to (q \to r) \vdash p \wedge q \to r$ および $p \wedge q \to r \vdash p \to (q \to r)$ であることがわかる．この 2 つをまとめて $p \wedge q \to r \dashv\vdash p \to (q \to r)$ と書くこともある．

選言に関する推論規則は

$$\frac{P}{P \vee Q}\vee\mathrm{i}_1 \qquad \frac{Q}{P \vee Q}\vee\mathrm{i}_2 \qquad \frac{\begin{array}{ccc} & [P] & [Q] \\ P \vee Q & \vdots & \vdots \\ & R & R \end{array}}{R}\vee\mathrm{e}$$

である．$\vee$-導入は，P が成り立っていれば任意の論理式 Q に対して $P \vee Q$ および $Q \vee P$ が推論できることを表している．$\vee$-除去は少し複雑である．$P \vee Q$ が成り立っている時は P と Q のいずれか一方かあるいは両方が成り立っている．そのため，もし P が成り立つと仮定した場合と Q が成り立つと仮定した場合で同じ R を証明できれば，$P \vee Q$ が成り立つ場合にも R が成り立つことがわかる．このことを述べたのが $\vee$-除去である．

例 5.19 $p \vee q \vdash q \vee p$ は以下のように証明できる．

$$\frac{p \vee q \qquad \dfrac{[p]}{q \vee p}\vee\mathrm{i}_2 \qquad \dfrac{[q]}{q \vee p}\vee\mathrm{i}_1}{q \vee p}\vee\mathrm{e}$$

□

否定に関する推論規則は

$$\frac{\bot}{P}\bot\mathrm{e} \qquad \frac{\begin{array}{c}[P] \\ \vdots \\ \bot\end{array}}{\neg P}\neg\mathrm{i} \qquad \frac{P \quad \neg P}{\bot}\neg\mathrm{e}$$

である．$\bot$ は恒偽式を表す記号である．矛盾ともいう．p を適当な命題変数として $p \wedge \neg p$ とみなしてもよい．$\bot$-除去は矛盾からはどのようなことでも推論できることを述べている．$\neg$-導入は P を仮定して矛盾が導かれるのであれば P ではなく $\neg P$ が成り立つことを表している．背理法と非常によく似ており，後で示すように $\neg$-導入を用いて背理法を証明することができる．$\neg$-除去は P とその否定の $\neg P$ が成り立つ時が矛盾であることを表す．

例 5.20 $\neg\neg$-導入は $\neg$-除去と $\neg$-導入を用いて

$$\frac{\dfrac{P \quad [\neg P]}{\bot}\neg\mathrm{e}}{\neg\neg P}\neg\mathrm{i}$$

のように導出できる． □

自然演繹の基本的な推論規則は $\wedge\mathrm{i}, \wedge\mathrm{e}_1, \wedge\mathrm{e}_2, \vee\mathrm{i}_1, \vee\mathrm{i}_2, \vee\mathrm{e}, \to \mathrm{i}, \to \mathrm{e}, \neg\mathrm{i}, \neg\mathrm{e}, \bot\mathrm{e}, \neg\neg\mathrm{e}$ の 12 個であるが，ここで証明に便利な規則をいくつか導入する．これらの規則はいずれも $\neg\neg$-導入と同様に 12 個の基本的な規則を用いて導出できる．

1 つ目は，後件否定 (modus tollens)

$$\dfrac{P \to Q \quad \neg Q}{\neg P}\ \mathrm{MT}$$

である．後件否定の意味は含意の意味を考えれば自然である．この規則は以下のように基本的な規則から導出できる．

$$\dfrac{\dfrac{\dfrac{[P] \quad P \to Q}{Q}\to \mathrm{e} \qquad \neg Q}{\bot}\neg \mathrm{e}}{\neg P}\neg \mathrm{i}$$

2 つ目は，背理法 (proof by contradiction)

$$\dfrac{\begin{array}{c}[\neg P]\\ \vdots\\ \bot\end{array}}{P}\ \mathrm{PBC}$$

である．¬-導入に非常によく似ている．この規則は以下のように導出できる．

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}[\neg P]\\ \vdots\\ \bot\end{array}}{\neg\neg P}\neg \mathrm{i}}{P}\neg\neg \mathrm{e}$$

3 つ目は，排中律 (the law of excluded middle)

$$\dfrac{}{P \vee \neg P}\ \mathrm{LEM}$$

である．排中律は，任意の論理式 P は **true** か **false** のいずれかであることを述べたものである．以下のように他の規則から導出できる．

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg(P \vee \neg P)]_3 \quad \dfrac{[P]_1}{P \vee \neg P}\vee \mathrm{i}_1}{\bot}\neg \mathrm{e}}{\neg P}\neg \mathrm{i},1 \qquad \dfrac{\dfrac{[\neg(P \vee \neg P)]_3 \quad \dfrac{[\neg P]_2}{P \vee \neg P}\vee \mathrm{i}_2}{\bot}\neg \mathrm{e}}{P}\mathrm{PBC},2}{\bot}\neg \mathrm{e}}{P \vee \neg P}\mathrm{PBC},3$$

練習問題 5.21 以下を証明せよ．

1. $p \wedge (q \wedge r) \dashv\vdash (p \wedge q) \wedge r$
2. $p \vee (q \vee r) \dashv\vdash (p \vee q) \vee r$
3. $(p \wedge q) \vee r \dashv\vdash (p \vee r) \wedge (q \vee r)$
4. $(p \vee q) \wedge r \dashv\vdash (p \wedge r) \vee (q \wedge r)$
5. $p \wedge (p \vee q) \dashv\vdash p$

6. $p \vee (p \wedge q) \dashv\vdash p$
7. $p \to q \dashv\vdash \neg p \vee q$
8. $p \to q \dashv\vdash \neg q \to \neg p$
9. $\neg(p \wedge q) \dashv\vdash \neg p \vee \neg q$
10. $\neg(p \vee q) \dashv\vdash \neg p \wedge \neg q$
11. $p \wedge q \dashv\vdash \neg(\neg p \vee \neg q)$
12. $p \vee q \dashv\vdash \neg(\neg p \wedge \neg q)$
13. $p \wedge p \dashv\vdash p$
14. $p \vee p \dashv\vdash p$

練習問題 5.22 以下を証明せよ．

1. $p \vee q, \neg p \vdash q$
2. $\neg p \to p \vdash p$
3. $\vdash p \to (q \to p)$
4. $\vdash (p \to q) \to ((q \to r) \to (p \to r))$

最後に，論理式集合に対する無矛盾性の概念を導入する．後で述べる証明系の無矛盾性と混同しないように注意すること．

定義 5.23 Φ を論理式集合とする．もし Φ から $\bot$ が導出できる時，つまり $\Phi \vdash \bot$ である時，かつその時に限り Φ は**矛盾** (inconsistent) であるという．矛盾でない場合，**無矛盾** (consistent) であるという． □

矛盾あるいは無矛盾な論理式集合は以下の性質を持つ．

命題 5.24 Φ を論理式集合とする．この時，以下の 3 つは互いに必要十分条件である．

1. Φ は矛盾である．
2. 任意の論理式 P に対して $\Phi \vdash P$ である．
3. ある論理式 P に対して $\Phi \vdash P$ かつ $\Phi \vdash \neg P$ である．

証明： (1. $\Rightarrow$ 2.) Φ は矛盾なので $\Phi \vdash \bot$ であり，$\bot$-除去により任意の論理式 P に対して $\Phi \vdash P$ である．(2. $\Rightarrow$ 3.) 自明．(3. $\Rightarrow$ 1.) $\neg$-除去により $\bot$ が導出できるので Φ は矛盾である． □

系 5.25 Φ を論理式集合とする．この時，以下の 2 つは互いに必要十分条件である．

1. Φ は無矛盾である．
2. $\Phi \vdash P$ ではない論理式 P が存在する． □

5.7 自然演繹の健全性

先に述べたように，証明系には健全性と完全性が期待される．そこで，まず自然演繹が健全であること，すなわち自然演繹によって論理式集合 Φ から論理式 P が推論できれば P は Φ の論理的帰結であることを示す．

定理 5.26 (自然演繹の健全性) $P_1, \ldots, P_n, Q$ を論理式とする．もし自然演繹によって $P_1, \ldots, P_n \vdash Q$ が証

明できれば $P_1, \ldots, P_n \models Q$ である．

証明： $P_1, \ldots, P_n \vdash Q$ の証明木の高さに関する帰納法による．

基底段階　証明木の高さが 1 の場合，証明木の形は

$$p$$

である．これは $p \vdash p$ の証明である．推論規則のいずれを適用しても証明木の高さは 2 以上になるため，証明木の高さが 1 になるのは前提と結論が等しい場合に限られる．この時，明らかに $p \models p$ である．

帰納段階　証明木の高さが n 未満の場合に成り立つと仮定し，n の場合を考える．証明木の根にあたる論理式（つまり結論）を導出するために適用した規則によって場合分けする．

- $\wedge$i の場合，$Q = Q_1 \wedge Q_2$ であり Q_1 と Q_2 の証明木が存在する．すなわち，論理式集合 Φ_1, Φ_2 が存在して $\Phi_1 \vdash Q_1$ および $\Phi_2 \vdash Q_2$ である．それぞれの証明木の高さは明らかに n 未満であるため，帰納法の仮定より $\Phi_1 \models Q_1$ および $\Phi_2 \models Q_2$ である．ここで $\Phi = \Phi_1 \cup \Phi_2$ とすると，Φ に含まれるすべての論理式の真理値を **true** とするような任意の解釈 I に対して，$I(Q_1) = I(Q_2) = \mathbf{true}$ であるので $I(Q) = I(Q_1 \wedge Q_2) = \mathbf{true}$ である．よって $\Phi \models Q$ である．
- $\rightarrow$ i の場合，$Q = Q_1 \rightarrow Q_2$ とすると Q_1 を仮定して Q_2 を導出する証明木が存在する．よって，Q_2 を導出する際の Q_1 以外の前提の集合を Φ とすると $\Phi \cup \{Q_1\} \vdash Q_2$ である．Q_2 を導出する証明木の高さは明らかに n 未満であるので，帰納法の仮定より $\Phi \cup \{Q_1\} \models Q_2$ である．ここで，命題 5.6 の 2. より $\Phi \models Q_1 \rightarrow Q_2$ である．
- $\wedge\mathrm{e}_1, \wedge\mathrm{e}_2, \neg\neg\mathrm{e}, \rightarrow \mathrm{e}, \vee\mathrm{i}_1, \vee\mathrm{i}_2, \vee\mathrm{e}, \neg\mathrm{i}, \neg\mathrm{e}$ の場合は省略（練習問題とする）． □

練習問題 5.27 定理 5.26 の証明を完成させよ．

前提が空の場合を考えると，健全性から $\vdash P$ ならば $\models P$ である．そして，この対偶を考えると $\not\models P$ ならば $\not\vdash P$ である．つまり，トートロジーでない命題を自然演繹によって前提なしに証明することはできない．例えば，P を適当な論理式として，$P \wedge \neg P$ は明らかにトートロジーではないため自然演繹では前提なしに証明できない．このことは当たり前のように思うかもしれないが重要である．

定義 5.28 (無矛盾性) 証明系 PS において $\vdash_{PS} P$ かつ $\vdash_{PS} \neg P$ となる論理式 P が存在する時，かつその時に限り PS は**矛盾**であるという．矛盾でない場合，**無矛盾**であるという． □

もし自然演繹が矛盾であるとすると，$\vdash P$ かつ $\vdash \neg P$ である論理式 P が存在し，$\neg$-除去と $\bot$-除去を適用することで任意の論理式 Q を導出できる．つまり，どんな論理式であっても証明できてしまうということであり，このような証明系は役に立たない．自然演繹は無矛盾であり，前提なしに証明できない論理式が存在する．

定理 5.29 (自然演繹の無矛盾性) 自然演繹は無矛盾である．

証明： 自然演繹が矛盾であるとする．この時，$\vdash P$ かつ $\vdash \neg P$ なる論理式 P が存在する．定理 5.26 より $\models P$ かつ $\models \neg P$ であるが，$\neg$ の意味からこれはあり得ない．よって，自然演繹は無矛盾である． □

系 5.30 $\vdash P$ ではない論理式 P が存在する．

証明： 自然演繹は無矛盾なので，ある論理式 P に対して $\vdash P$ と $\vdash \neg P$ の両方が成り立つことはない． □

5.8 自然演繹の完全性

自然演繹は完全でもある．すなわち，論理式 P が論理式集合 Φ の論理的帰結であるならば自然演繹によって Φ から P を推論できる．自然演繹の完全性は，$P_1, \ldots, P_n \models Q$ が成り立つと仮定して，以下の 3 ステップで証明できる．

1. $\models P_1 \to (P_2 \to (\ldots (P_n \to Q) \ldots))$ を示す
2. $\vdash P_1 \to (P_2 \to (\ldots (P_n \to Q) \ldots))$ を示す
3. $P_1, \ldots, P_n \vdash Q$ を示す

補題 5.31 $P_1, \ldots, P_n, Q$ を論理式とする．もし $P_1, \ldots, P_n \models Q$ ならば $\models P_1 \to (P_2 \to (\ldots (P_n \to Q) \ldots))$ である．

証明： $R = P_1 \to (P_2 \to (\ldots (P_n \to Q) \ldots))$ とする．ある解釈 I があって $I(R) = \mathbf{false}$ となるのは，1 から n のすべての i について $I(P_i) = \mathbf{true}$ であり，かつ $I(Q) = \mathbf{false}$ の場合のみである．しかし，$P_1, \ldots, P_n \models Q$ であるためそのような解釈 I は存在しない．よって $\models R$ である． □

補題 5.32 P は論理式であり，命題変数として $m + n$ 個の命題変数 $p_1, \ldots, p_m, q_1, \ldots, q_n$ のみを含むとする．また，解釈 I は，1 から m のすべての i について $I(p_i) = \mathbf{true}$，かつ 1 から n のすべての i について $I(q_i) = \mathbf{false}$ を満たすとする．この時，$\Phi = \{p_1, \ldots, p_m, \neg q_1, \ldots, \neg q_n\}$ とすると

1. $I(P) = \mathbf{true}$ ならば $\Phi \vdash P$
2. $I(P) = \mathbf{false}$ ならば $\Phi \vdash \neg P$

である．

証明： P の構造に関する帰納法による．

- $P = p$ の場合，
  - $I(P) = \mathbf{true}$ ならば，$\Phi = \{p\}$ なので明らかに $\Phi \vdash p$ である．
  - $I(P) = \mathbf{false}$ ならば，$\Phi = \{\neg p\}$ なので明らかに $\Phi \vdash \neg p$ である．
- $P = \neg Q$ の場合，
  - $I(P) = \mathbf{true}$ ならば，$I(Q) = \mathbf{false}$ である．Q に含まれる命題変数の集合を Φ とすると，帰納法の仮定より $\Phi \vdash \neg Q$ であるので，明らかに $\Phi \vdash P$ である．
  - $I(P) = \mathbf{false}$ ならば，$I(Q) = \mathbf{true}$ である．Q に含まれる命題変数の集合を Φ とすると，帰納法の仮定より $\Phi \vdash Q$ であり，$\neg\neg$-導入の規則を適用すると $\Phi \vdash \neg\neg Q$ が得られる．これはすなわち $\Phi \vdash \neg P$ である．

以下，P が $Q_1 \wedge Q_2$，$Q_1 \vee Q_2$，$Q_1 \to Q_2$ の場合を示す．以下では，Φ, Ψ_1, Ψ_2 をそれぞれ P, Q_1, Q_2 に含まれる命題変数の集合とする．明らかに $\Phi = \Psi_1 \cup \Psi_2$ である．

- $P = Q_1 \wedge Q_2$ の場合，
  - $I(P) = \mathbf{true}$ ならば，$I(Q_1) = \mathbf{true}$ かつ $I(Q_2) = \mathbf{true}$ である．帰納法の仮定より $\Psi_1 \vdash Q_1$ および $\Psi_2 \vdash Q_2$．よって $\Phi \vdash Q_1 \wedge Q_2$ である．

– $I(P) = \mathbf{false}$ ならば，$I(Q_1)$ と $I(Q_2)$ のいずれか一方あるいは両方が **false** である．ここで，$I(Q_1) = \mathbf{false}$ および $I(Q_2) = \mathbf{true}$ であるとすると，帰納法の仮定より $\Psi_1 \vdash \neg Q_1$ および $\Psi_2 \vdash Q_2$ であり，$\wedge$-導入を適用して $\Phi \vdash \neg Q_1 \wedge Q_2$ が得られる．あとは $\neg Q_1 \wedge Q_2 \vdash \neg(Q_1 \wedge Q_2)$ を示せばよい（練習問題とする）．$I(Q_1) = \mathbf{true}$ および $I(Q_2) = \mathbf{false}$ の場合も同様（練習問題とする）．$I(Q_1) = \mathbf{false}$ および $I(Q_2) = \mathbf{false}$ の場合は，帰納法の仮定から $\Phi \vdash \neg Q_1 \wedge \neg Q_2$ が得られるので，$\neg Q_1 \wedge \neg Q_2 \vdash \neg(Q_1 \wedge Q_2)$ を示せばよい（練習問題とする）．

- $P = Q_1 \vee Q_2$ の場合，省略（練習問題とする）．
- $P = Q_1 \to Q_2$ の場合，省略（練習問題とする）． □

練習問題 5.33 以下を証明せよ．

1. $\neg p \wedge q \vdash \neg(p \wedge q)$
2. $\neg p \wedge \neg q \vdash \neg(p \wedge q)$

練習問題 5.34 補題 5.32 の証明を完成させよ．

1. $P = Q_1 \wedge Q_2$，$I(Q_1) = \mathbf{true}$，$I(Q_2) = \mathbf{false}$ の場合を証明せよ．
2. $P = Q_1 \vee Q_2$ の場合を証明せよ．
3. $P = Q_1 \to Q_2$ の場合を証明せよ．

補題 5.35 $\models P$ ならば $\vdash P$ である．

証明： P を論理式として $\models P$ とする．P に含まれる命題変数を $p_1, \ldots, p_n$ とする．また，$\hat{p}_1, \ldots, \hat{p}_n$ はそれぞれ p_i か $\neg p_i$ のいずれかを表すとする．$\models P$ であるため，すべての解釈 I に対して $I(P) = \mathbf{true}$ である．$\{\hat{p}_1, \ldots, \hat{p}_n\}$ の各 $\hat{p}_i$ について p_i または $\neg p_i$ の 2 通りの可能性があるため，$\{\hat{p}_1, \ldots, \hat{p}_n\}$ には 2^n 通りの組み合わせがあるが，すべての解釈 I に対して $I(P) = \mathbf{true}$ であることと補題 5.32 よりそのすべての組み合わせについて $\hat{p}_1, \ldots, \hat{p}_n \vdash P$ である（$\hat{p}_i$ として $I(p_i) = \mathbf{true}$ ならば p_i，$I(p_i) = \mathbf{false}$ ならば $\neg p_i$ を選べばよい）．ここで $\hat{p}_n$ に着目すると，$\hat{p}_1, \ldots, \hat{p}_{n-1}, p_n \vdash P$ および $\hat{p}_1, \ldots, \hat{p}_{n-1}, \neg p_n \vdash P$ である．よって，排中律と $\vee$-除去を適用して $\hat{p}_1, \ldots, \hat{p}_{n-1} \vdash P$ が得られる．これを繰り返せば $\vdash P$ が得られる． □

補題 5.36 $P_1, \ldots, P_n, Q$ を論理式とする．もし $\vdash P_1 \to (P_2 \to (\ldots(P_n \to Q)\ldots))$ ならば $P_1, \ldots, P_n \vdash Q$ である．

証明： $\vdash P_1 \to (P_2 \to (\ldots(P_n \to Q)\ldots))$ の証明に P_1 を導入して $\to$-除去規則を適用すれば

$$\dfrac{P_1 \quad P_1 \to (P_2 \to (\ldots(P_n \to Q)\ldots))}{P_2 \to (\ldots(P_n \to Q)\ldots)} \to \mathrm{e}$$

のように $P_1 \vdash P_2 \to (\ldots(P_n \to Q)\ldots)$ が得られる．同様にして $P_2, \ldots, P_n$ を順に前提として導入すれば

$$\dfrac{\dfrac{\dfrac{P_3 \quad \dfrac{P_2 \quad P_2 \to (\ldots(P_n \to Q)\ldots)}{P_3 \to (\ldots(P_n \to Q)\ldots)} \to \mathrm{e}}{\vdots} \to \mathrm{e}}{Q} \to \mathrm{e}$$

のようにして $P_1, \ldots, P_n \vdash Q$ が得られる．つまり，$\vdash P_1 \to (P_2 \to (\ldots(P_n \to Q)\ldots))$ の証明から $P_1, \ldots, P_n \vdash Q$ の証明が構成できる．（正式には n に関する数学的帰納法による．） □

定理 5.37 (自然演繹の完全性) $P_1, \ldots, P_n, Q$ を論理式とする．もし $P_1, \ldots, P_n \models Q$ ならば自然演繹によって $P_1, \ldots, P_n \vdash Q$ を証明できる．

証明： 補題 5.31，5.35，5.36 から容易に導かれる． □

第 6 章

モデル検査

6.1　モデル検査とは

モデル検査は，システムが正しく動作するか検証する手法の 1 つである．システムの動作を状態遷移系（有限状態モデル）として，調べたい性質を論理式として記述し，システムの動作が性質を満たすか否かを状態遷移系をしらみつぶしに調べることで判定する．システムの動作のあらゆる可能性について検査できることが利点の 1 つである．通常はシステムが正しく動作するか確認するためにテストが行われる．しかし，テストではシステムの取り得る動作のいくつかについて確認するだけであり，あらゆる可能性について確認することは現実的には不可能である．すべての可能性を調べ尽くす網羅性がモデル検査の優位性の 1 つである．

状態遷移系が与えられた論理式を満たすか否かの判定は計算機を利用して完全に自動化できる．そのため，状態遷移系によるシステムの記述と論理式による性質の記述が得られればモデル検査が行える．モデル検査では，システムが与えられた性質を満たさない場合に，状態がどのように遷移すると性質を満たさないのか具体的な例（反例）を示すことが可能であり，開発者は反例を分析することで原因を突き止めることができる．これらもモデル検査の利点である．

システムの動作を表現する状態遷移系は，状態の集合と状態間の遷移を決めれば定義される．直観的には，非決定性有限オートマトン $(Q, \Sigma, \delta, q_0, F)$ から入力記号の集合 Σ と最終状態の集合 F を取り除き，遷移関数 δ を $Q \times \Sigma \to 2^Q$ ではなく $Q \to 2^Q$ とみなせばよい．これで，システムが取り得る状態としてどのような状態があり，どの状態からどの状態へと変化する可能性があるか把握することができる．$q_1 = \delta(q_0), q_2 = \delta(q_1), \ldots$ である時，$q_0 q_1 q_2 \ldots$ が 1 つの実行を表しているとみなす．つまり，システムの実行を状態の移り変わりによって表す．状態遷移系から得られるすべての状態遷移列（つまり実行）について与えられた性質を満たすか調べることがモデル検査である．

システムの動作を記述する状態遷移系としては Kripke（クリプキ）構造やオートマトンが，調べたい性質の記述には時相論理が利用されることが多い．そこで，以下では Kripke 構造と時相論理について説明し，その後，性質の具体的な記述，モデル検査のアルゴリズム，並行プログラムのモデル検査について取り上げる．

6.2　Kripke 構造

システムの動作を表現する状態遷移系として **Kripke 構造**を導入する．以下では，PV を命題変数の集合とする．PV に含まれる各命題変数は何らかの命題を表す．

定義 6.1 (Kripke 構造) Kripke 構造 M は 3 項組 (S, R, L) で定義される．ここで，S は状態の有限集合であ

り，$S \neq \emptyset$ とする．$R \subseteq S \times S$ は状態間の遷移を表す関係であり，$R(s, s')$ は状態 s から s' に遷移できることを表す．任意の $s \in S$ に対して $R(s, s')$ なる $s' \in S$ が少なくとも 1 つは存在するとする．L は $S \to 2^{PV}$ として定義されるラベル付け関数であり，任意の $s \in S$ に対して $L(s)$ は状態 s で真となる命題変数の集合を与える． □

状態 s と s' に対して $R(s, s')$ である時，$s \to s'$ と書くこともある．直観的には，システムの状態が s の場合，次の時刻でシステムの状態が s' に遷移すると理解すればよい．s に対して $s \to s'$ なる s' が複数存在する場合は，いずれかを非決定的に選択すると考える．

状態 $s_0 \in S$ から始まる無限長の状態列

$$\sigma = s_0 s_1 s_2 \ldots \qquad \text{ただし，任意の } i \geq 0 \text{ に対して } R(s_i, s_{i+1})$$

を**経路** (path) と呼ぶ．経路 σ の $i\,(\geq 0)$ 番目の状態を $\sigma(i)$ と表す．

例 6.2 スキャナとプリンタを用いてコピーを行うシステムを Kripke 構造で表現することを考える．このシステムは，スキャナ，プリンタの順にロックしてコピーを行い，プリンタ，スキャナの順にロックを解除する．また，コピー前にスキャナをロックした時点でキャンセルできるものとする．

命題変数 p, q について，p は「スキャナをロックしている」，q は「プリンタをロックしている」ことを表すこととすると，以下の Kripke 構造 $\mathrm{COPY} = (S, R, L)$ によって表現できる．

$$\begin{aligned}
&S : \{s_0, s_1, s_2, s_3\} \\
&R : R(s_0, s_1), R(s_1, s_2), R(s_1, s_0), R(s_2, s_3), R(s_3, s_0) \\
&L : L(s_0) = \emptyset \\
&\quad\ L(s_1) = L(s_3) = \{p\} \\
&\quad\ L(s_2) = \{p, q\}
\end{aligned}$$

s_0 が初期状態，s_2 がスキャナとプリンタをロックしてコピーしている状態を表す．COPY の状態遷移図を図 6.1 に示す． □

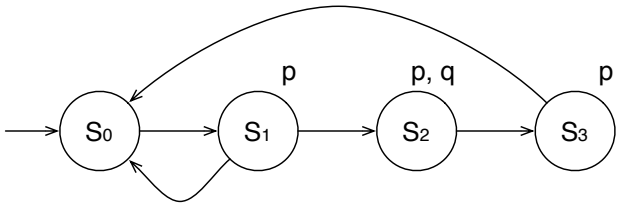


図 6.1　Kripke 構造 COPY

定義 6.1 ではすべての状態に対して遷移先の存在を要求しているため，図 6.2 の上側の状態遷移系のように遷移先のない状態（この例では s_2）を持つ状態遷移系は認められない．しかし，下側の状態遷移系のように s_2 の遷移先としてループを繰り返す状態 s_d が存在するとみなせば，遷移先のない状態を持つ状態遷移系を定義 6.1 の範囲で表現できる．

例 6.3 以下のプログラムを Kripke 構造でモデル化することを考える．Kripke 構造の状態および命題変数の集合の決め方は一通りではなく，モデル検査の対象とする性質に依存する．ここでは，実行中の行番号を状態

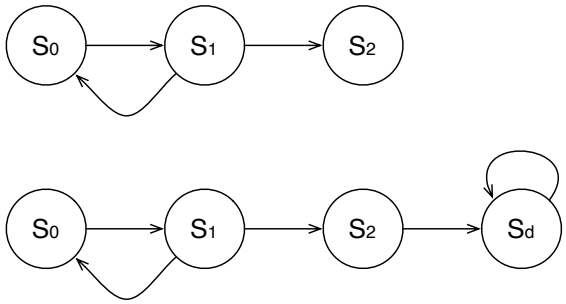


図 6.2　遷移先のない状態を持つ Kripke 構造

とみなした場合と，行番号と n の値の組を状態とみなした場合の Kripke 構造を示す．

```
1: n = 3; fact = 1;
2: while (n > 1) {
3:   fact *= n;
4:   n--;
5: }
6: print(fact);
```

行番号を状態とする場合

以下の Kripke 構造 $Fact_l = (S_l, R_l, L_l)$ によって表現できる．ここで，命題変数 l_i は実行中の行番号が i であること，n_i は変数 n の値が i であること，f_i は fact の値が i であることを表し，$N_{i,\ldots,j} = n_i \vee \cdots \vee n_j$ および $F_{i,\ldots,j} = f_i \vee \cdots \vee f_j$ とする．

$$
\begin{aligned}
&S_l : \{s_1, s_2, s_3, s_4, s_6\} \\
&R_l : R_l(s_1, s_2), R_l(s_2, s_3), R_l(s_2, s_6), R_l(s_3, s_4), R_l(s_4, s_2) \\
&L_l : L_l(s_1) = \{l_1, n_3, f_1\} \qquad L_l(s_2) = \{l_2, N_{3,2,1}, F_{1,3,6}\} \\
&\quad\ L_l(s_3) = \{l_3, N_{3,2}, F_{3,6}\} \qquad L_l(s_4) = \{l_4, N_{2,1}, F_{3,6}\} \\
&\quad\ L_l(s_6) = \{l_6, n_1, f_6\}
\end{aligned}
$$

行番号と n の値の組を状態とする場合

以下の Kripke 構造 $Fact_n = (S_n, R_n, L_n)$ によって表現できる．各命題変数の意味は上と同じである．

$$
\begin{aligned}
&S_n : \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\} \\
&R_n : R_n(s_1, s_2), R_n(s_2, s_3), R_n(s_3, s_4), R_n(s_4, s_5), R_n(s_5, s_6), R_n(s_6, s_7), R_n(s_7, s_8), R_n(s_8, s_9) \\
&L_n : L_n(s_1) = \{l_1, n_3, f_1\} \qquad L_n(s_2) = \{l_2, n_3, f_1\} \\
&\quad\ L_n(s_3) = \{l_3, n_3, f_3\} \qquad L_n(s_4) = \{l_4, n_2, f_3\} \\
&\quad\ L_n(s_5) = \{l_2, n_2, f_3\} \qquad L_n(s_6) = \{l_3, n_2, f_6\} \\
&\quad\ L_n(s_7) = \{l_4, n_1, f_6\} \qquad L_n(s_8) = \{l_2, n_1, f_6\} \\
&\quad\ L_n(s_9) = \{l_6, n_1, f_6\}
\end{aligned}
$$

他にも，n 以外の変数に着目して状態とするモデル化も考えられる．　□

6.3 時相論理

時相論理 (temporal logic) は，命題論理に対して，「ある性質が次の時刻で成り立つ」や「ある性質が将来のいつかの時点で成り立つ」といった時間が関わる性質を記述するための時相演算子を追加した論理体系である．命題論理によって記述できる性質は，システムのある瞬間の状態を対象とする性質のみであるため，例えば「このプログラムはいつか停止する」といった性質を記述することはできない．しかし，システムやプログラムの正しさを検証するためには，動作に伴う状態の変化を考慮することも必要であり，時間の経過に関わる性質を記述するためには時相論理が有効である．

時間をどのように扱うかによって様々な時相論理が考えられるが，ここでは，時間は離散的であり，時間は点であり，未来のみを扱う時相論理を考える．時間が離散的であるとは，各時刻に対して直前および直後の時刻が存在するということである．例えば，1 秒，2 秒，3 秒，と数えるのは時間を離散的にとらえている．連続時間の場合は，どれだけ近い 2 つの時刻の間にも別の時刻が存在すると考える．時間が点であるとは，幅を持たない各瞬間における論理を扱うということである．

ここで取り上げる時相論理は未来に向かう時間の流れを扱うが，時間の流れに伴う状態変化のあらゆる可能性を同時に考慮する**分岐時間**と，時間の流れに伴う状態変化のうちの 1 つの可能性のみを追跡する**線形時間**の 2 種類の扱い方があり，それぞれに対応する時相論理が考えられている．以下では，分岐時間を扱う分岐時相論理である**計算木論理** (**CTL**: Computational Tree Logic) と，線形時間を扱う**線形時相論理** (**LTL**: Linear Temporal Logic) を説明する．時間の経過については，Kripke 構造においてある状態から次の状態へ遷移する時に次の時刻に進むものとする．

6.3.1 CTL

ある状態を開始状態とする経路の集合は，木構造を用いると 1 つにまとめて表現できる．この木構造を**計算木**と呼び，CTL は計算木に対して意味が定義される論理体系である．CTL は，経路演算子と呼ばれる「すべての経路において」を意味する演算子と，「ある経路において」を意味する演算子，および時相演算子と呼ばれる時間に関する性質を表す 4 種類の演算子を持つ．4 種類の時相演算子の意味は，「次の時刻で」，「将来のいつか」，「今後ずっと」，「ある時点まで」である．

初めに，CTL の論理式を定義する．命題論理の論理式に 6 種類の演算子を追加したものになっている．

定義 6.4 (CTL の論理式) CTL の論理式を以下のように帰納的に定義する．

1. 命題変数は状態式である．
2. P, Q が状態式ならば，$(P \wedge Q)$，$(P \vee Q)$，$(\neg P)$，$(P \to Q)$ は状態式である．
3. P, Q が状態式ならば，$\mathbf{X}\,P$，$\mathbf{F}\,P$，$\mathbf{G}\,P$，$P\,\mathbf{U}\,Q$ は経路式である．
4. P が経路式ならば，$\mathbf{A}\,P$，$\mathbf{E}\,P$ は状態式である．
5. 以上が状態式と経路式のすべてであり，状態式が CTL の論理式のすべてである． □

演算子の優先度は，$\mathbf{A}, \mathbf{E}, \mathbf{X}, \mathbf{F}, \mathbf{G}$ は $\neg$ と同じ，$\mathbf{U}$ は $\wedge$ と同じとする．以下，CTL の論理式を CTL 式とも呼ぶ．CTL の論理式は，命題論理の論理式であるか，文法の制限から以下のいずれかの形をした論理式である．ここで，P, Q は適当な CTL 式である．直観的な意味も一緒に示す（図 6.3）．

- $\mathbf{AX}\,P$　今の状態から開始されるすべての経路において，次の状態で P が成り立つ．
- $\mathbf{EX}\,P$　今の状態から開始されるいずれかの経路において，次の状態で P が成り立つ．
- $\mathbf{AF}\,P$　今の状態から開始されるすべての経路において，いつか P が成り立つ．
- $\mathbf{EF}\,P$　今の状態から開始されるいずれかの経路において，いつか P が成り立つ．
- $\mathbf{AG}\,P$　今の状態から開始されるすべての経路において，すべての状態で P が成り立つ．
- $\mathbf{EG}\,P$　今の状態から開始されるいずれかの経路において，すべての状態で P が成り立つ．
- $\mathbf{A}[P\,\mathbf{U}\,Q]$　今の状態から開始されるすべての経路において，いつか Q が成り立ち，初めて Q が成り立つまでのすべての状態で P が成り立つ．
- $\mathbf{E}[P\,\mathbf{U}\,Q]$　今の状態から開始されるいずれかの経路において，いつか Q が成り立ち，初めて Q が成り立つまでのすべての状態で P が成り立つ．

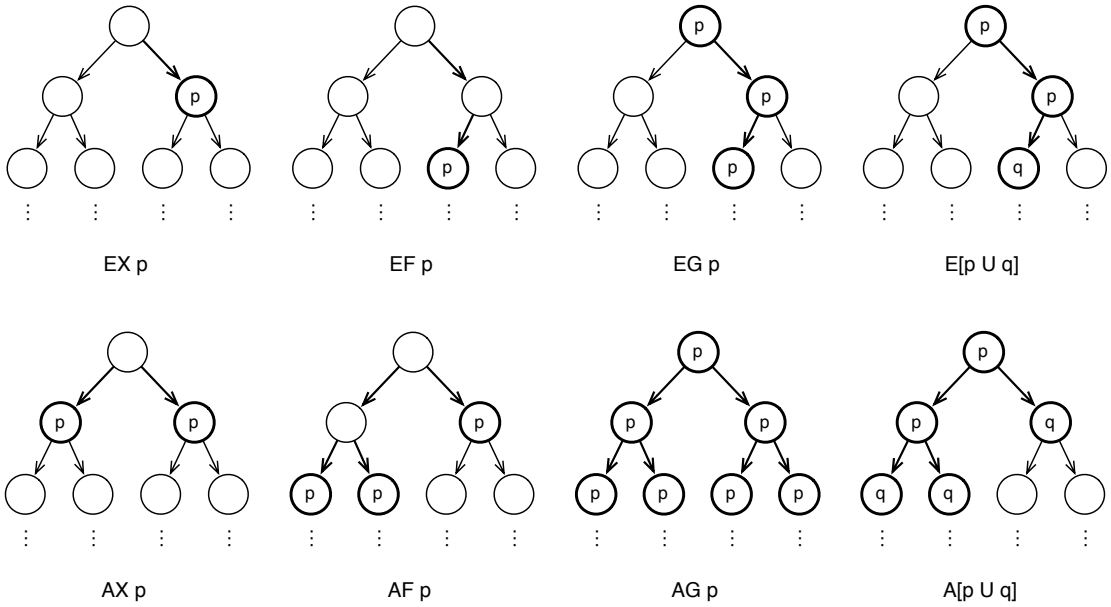


図 6.3　CTL の時相演算子の直観的な意味

このように CTL の時相演算子は 8 種類あるが，否定演算子などと組み合わせて別の演算子を表現することができる．例えば，$\mathbf{EX}, \mathbf{EG}, \mathbf{EU}$ の 3 種類の演算子があれば，以下のように他の 5 種類の演算子が表現できる．ここで，$\top$ は恒真式を表す．

- $\mathbf{AX}\,P = \neg\mathbf{EX}\,(\neg P)$
- $\mathbf{AF}\,P = \neg\mathbf{EG}\,(\neg P)$
- $\mathbf{EF}\,P = \mathbf{E}[\top\,\mathbf{U}\,P]$
- $\mathbf{AG}\,P = \neg\mathbf{EF}\,(\neg P)$
- $\mathbf{A}[P\,\mathbf{U}\,Q] = \neg(\mathbf{EG}\,(\neg Q) \vee \mathbf{E}[\neg Q\,\mathbf{U}\,(\neg P \wedge \neg Q)])$

練習問題 6.5 $\mathbf{EX}, \mathbf{AU}, \mathbf{EU}$ を用いて他の演算子を表現せよ．

Kripke 構造 $M = (S, R, L)$ における状態 $s \in S$ を開始状態とする計算木において CTL 式 P が成り立つこ

とを

$$M, s \models P$$

と書いて表す．これを用いて，先に直観的な説明を与えた時相演算子を含め，CTL の意味を以下のように定義する．ここで，$p \in PV$ である．

$$
\begin{array}{lcl}
M, s \models p & \Leftrightarrow & p \in L(s) \\
M, s \models \neg P & \Leftrightarrow & M, s \not\models P \\
M, s \models P \wedge Q & \Leftrightarrow & M, s \models P \text{ かつ } M, s \models Q \\
M, s \models P \vee Q & \Leftrightarrow & M, s \models P \text{ または } M, s \models Q \\
M, s \models P \rightarrow Q & \Leftrightarrow & M, s \models P \text{ ならば } M, s \models Q \\
M, s \models \mathbf{AX}\, P & \Leftrightarrow & \sigma(0) = s \text{ なるすべての経路 } \sigma \text{ において } M, \sigma(1) \models P \\
M, s \models \mathbf{EX}\, P & \Leftrightarrow & \sigma(0) = s \text{ かつ } M, \sigma(1) \models P \text{ なる経路 } \sigma \text{ が存在する} \\
M, s \models \mathbf{AF}\, P & \Leftrightarrow & \sigma(0) = s \text{ なるすべての経路 } \sigma \text{ においてある } i \geq 0 \text{ が存在して } M, \sigma(i) \models P \\
M, s \models \mathbf{EF}\, P & \Leftrightarrow & \sigma(0) = s \text{ かつ } M, \sigma(i) \models P \text{ なる } i \geq 0 \text{ が存在する経路 } \sigma \text{ が存在する} \\
M, s \models \mathbf{AG}\, P & \Leftrightarrow & \sigma(0) = s \text{ なる任意の経路 } \sigma \text{ において任意の } i \geq 0 \text{ に対して } M, \sigma(i) \models P \\
M, s \models \mathbf{EG}\, P & \Leftrightarrow & \sigma(0) = s \text{ かつ任意の } i \geq 0 \text{ に対して } M, \sigma(i) \models P \text{ なる経路 } \sigma \text{ が存在する} \\
M, s \models \mathbf{A}[P\, \mathbf{U}\, Q] & \Leftrightarrow & \sigma(0) = s \text{ なる任意の経路 } \sigma \text{ においてある } j \geq 0 \text{ が存在して } M, \sigma(j) \models Q \text{ かつ} \\
 & & 0 \leq i \leq j \text{ に対して } M, \sigma(i) \models P \\
M, s \models \mathbf{E}[P\, \mathbf{U}\, Q] & \Leftrightarrow & \sigma(0) = s \text{ かつ，ある } j \geq 0 \text{ が存在して } M, \sigma(j) \models Q \text{ かつ } 0 \leq i \leq j \text{ に対して} \\
 & & M, \sigma(i) \models P \text{ なる経路 } \sigma \text{ が存在する}
\end{array}
$$

例 6.6 例 6.2 の COPY について，

- $\mathrm{COPY}, s_0 \models \mathbf{EF}\,(p \wedge q)$
- $\mathrm{COPY}, s_0 \models \neg\mathbf{AF}\,(p \wedge q)$

である．前者はいつかはスキャナとプリンタをともにロックできること，後者はスキャナとプリンタをともにロックしない場合があることを表している． □

練習問題 6.7 例 6.2 の COPY について，

- $\mathrm{COPY}, s_0 \models \mathbf{AG}\,(q \rightarrow p)$
- $\mathrm{COPY}, s_0 \models \mathbf{EG}\,(\neg q)$

であることを適当な高さの計算木を描いて確認せよ．

6.3.2 LTL

LTL はある状態を開始状態とする 1 つの経路に対して意味が定義される論理体系である．LTL は，CTL と同じ時相演算子を持つ．経路は 1 通りなので経路演算子は存在しない．

LTL の論理式は次のように定義される．

定義 6.8 LTL の論理式を以下のように帰納的に定義する．

1. 命題変数 p は LTL の論理式である．

2. P, Q が論理式ならば $(P \wedge Q)$，$(P \vee Q)$，$(\neg P)$，$(P \to Q)$，$(\mathbf{X}\, P)$，$(\mathbf{F}\, P)$，$(\mathbf{G}\, P)$，$(P\, \mathbf{U}\, Q)$ は LTL の論理式である．
3. 1. と 2. で LTL の論理式であるものだけが LTL の論理式である． □

LTL の意味を CTL と同様に Kripke 構造を用いて定義する．Kripke 構造 $M = (S, R, L)$ の経路 σ の i 番目（先頭を 0 とする）の状態において論理式 P が成り立つことを

$$M, \sigma, i \models P$$

と書いて表す．LTL の意味を以下のように定義する．ここで，$p \in PV$ である．

$$\begin{array}{lcl}
M, \sigma, i \models p & \Leftrightarrow & p \in L(\sigma(i)) \\
M, \sigma, i \models \neg P & \Leftrightarrow & M, \sigma, i \not\models P \\
M, \sigma, i \models P \wedge Q & \Leftrightarrow & M, \sigma, i \models P \text{ かつ } M, \sigma, i \models Q \\
M, \sigma, i \models P \vee Q & \Leftrightarrow & M, \sigma, i \models P \text{ または } M, \sigma, i \models Q \\
M, \sigma, i \models P \to Q & \Leftrightarrow & M, \sigma, i \models P \text{ ならば } M, \sigma, i \models Q \\
M, \sigma, i \models \mathbf{X}\, P & \Leftrightarrow & M, \sigma, i+1 \models P \\
M, \sigma, i \models \mathbf{F}\, P & \Leftrightarrow & \text{ある } j \geq i \text{ が存在して } M, \sigma, j \models P \\
M, \sigma, i \models \mathbf{G}\, P & \Leftrightarrow & \text{任意の } j \geq i \text{ に対して } M, \sigma, j \models P \\
M, \sigma, i \models P\, \mathbf{U}\, Q & \Leftrightarrow & \text{ある } k \geq i \text{ が存在して } M, \sigma, k \models Q \text{ かつ任意の } i \leq j < k \text{ に対して } M, \sigma, j \models P
\end{array}$$

例 6.9 例 6.2 の COPY を再度考える．経路 σ_1 を $s_0 s_1 s_2 s_3$ を繰り返す経路，σ_2 を $s_0 s_1$ を繰り返す経路とする．つまり，$i = 0, 1, 2, \ldots$ に対して $\sigma_1(4i) = s_0$，$\sigma_1(4i+1) = s_1$，$\sigma_1(4i+2) = s_2$，$\sigma_1(4i+3) = s_3$，$\sigma_2(2i) = s_0$，$\sigma_2(2i+1) = s_1$ である．この時，

- $\mathrm{COPY}, \sigma_1, 1 \models \mathbf{X}\, (p \wedge q)$
- $\mathrm{COPY}, \sigma_1, 0 \models \mathbf{G}\, (p \to (\mathbf{F}\, q))$
- $\mathrm{COPY}, \sigma_2, 0 \models \mathbf{G}\, \neg q$

である． □

練習問題 6.10 例 6.2 の COPY について，

- $\mathrm{COPY}, \sigma_1, 3 \models p\, \mathbf{U}\, q$
- $\mathrm{COPY}, \sigma_2, 0 \models \mathbf{F}\, q$

が成り立たないことを確認せよ．

CTL と LTL は表現能力が異なる．例えば，CTL の $\mathbf{AG}\, (P \to \mathbf{EF}\, Q)$ は LTL では表すことができない．一方，LTL の $\mathbf{GF}\, P$ を CTL で表現することはできない．CTL と LTL の双方を含む CTL* という体系もあるが，ここでは省略する．

6.4 モデル検査アルゴリズム

モデル検査は，システムのモデルである Kripke 構造が，満たしたい性質を表す CTL 式や LTL 式のモデルになっているか検査することである．ここでは，モデル検査のアルゴリズムを説明する．特に，Kripke 構造 $M = (S, R, L)$ において状態 $s \in S$ が CTL 式 P を満たしているか，すなわち $M, s \models P$ であるか判定するア

ルゴリズムを示す．CTL 式は演算子として $\neg, \vee, \mathbf{EX}, \mathbf{EG}, \mathbf{EU}$ があればすべて表現できるので，ここではこの 5 種類の演算子のみを扱う．

アルゴリズムの方針として，Kripke 構造 M の各状態 $s \in S$ についてその状態で成り立つ CTL 式の集合を求め，その集合に P が含まれるか否かを判定することとする．この時，各状態において成り立つ CTL 式として適当な CTL 式ではなく P と P の部分式について成り立つか調べる．状態 s において成り立つ CTL 式の集合を $label_M(s)$ と書くことにすると，$M, s \models P$ であるか判定することは $P \in label_M(s)$ か判定することと同じである．そのため，各状態で成り立つ CTL 式の集合を求めることがアルゴリズムの中心となる．以下，M が明らかな場合は単に $label(s)$ と書く．

各状態で成り立つ CTL 式の集合を求めるアルゴリズム $Check(M, P)$ を図 6.4，図 6.5，図 6.6 に示す．ここで，M は Kripke 構造，P は CTL 式，PV は命題変数の集合である．すべての $s \in S$ に対して $label(s)$ の初期値は $\emptyset$ として，P の構造によって場合分けする．直観的には，$Check(M, P)$ は M の各状態に対して P と P の部分式のうちその状態で成り立つ式をラベルとして付加する．

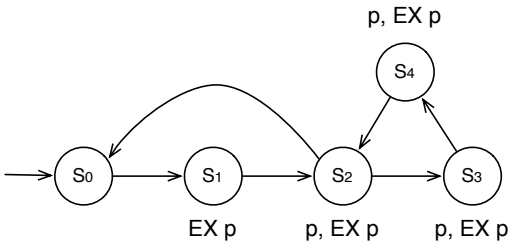
procedure $Check(M, P)$
  case: $P \in PV$
    for all $s \in S$ **do**
      if $P \in L(s)$ **then** $label(s) := label(s) \cup \{P\}$
  case: $P \equiv \neg Q$
    $Check(M, Q)$
    for all $s \in S$ **do**
      if $Q \notin label(s)$ **then** $label(s) := label(s) \cup \{\neg Q\}$
  case: $P \equiv Q_1 \vee Q_2$
    $Check(M, Q_1)$
    $Check(M, Q_2)$
    for all $s \in S$ **do**
      if $Q_1 \in label(s)$ **or** $Q_2 \in label(s)$ **then** $label(s) := label(s) \cup \{Q_1 \vee Q_2\}$
  case: $P \equiv \mathbf{EX}\, Q$
    $Check(M, Q)$
    for all $s \in \{s \mid Q \in label(s)\}$ **do**
      for all s' **such that** $R(s', s)$ **do**
        $label(s') := label(s') \cup \{\mathbf{EX}\, Q\}$
  case: $P \equiv \mathbf{EG}\, Q$
    $CheckEG(M, Q)$
  case: $P \equiv \mathbf{E}[Q_1\, \mathbf{U}\, Q_2]$
    $CheckEU(M, Q_1, Q_2)$
end procedure

図 6.4　Kripke 構造 M の各状態において成り立つ CTL 式 P の部分式を求めるアルゴリズム

procedure $CheckEG(M, Q)$
  $Check(M, Q)$
  $S' := \{s \mid Q \in label(s)\}$
  SCC := $\{C \mid C$ は S'の強連結成分 $\}$
  $T := \bigcup_{C \in \mathrm{SCC}} \{s \mid s \in C\}$
  for all $s \in T$ **do** $label(s) := label(s) \cup \{\mathbf{EG}\, Q\}$
  while $T \neq \emptyset$ **do**
    choose $s \in T$
    $T := T - \{s\}$
    for all $s' \in S'$ **such that** $R(s', s)$ **do**
      if $\mathbf{EG}\, Q \notin label(s')$ **then**
        $label(s') := label(s') \cup \{\mathbf{EG}\, Q\}$
        $T := T \cup \{s'\}$
end procedure

図 6.5　$\mathbf{EG}\, Q$ が成り立つ状態にラベル付けするアルゴリズム

- $P \in PV$ の場合，すべての $s \in S$ について $P \in L(s)$ ならば s にラベル P を付ける．
- $\neg Q$ の場合，Q が成り立つ状態にラベル Q を付けた後，ラベル Q が付いていないすべての状態にラベル $\neg Q$ を付ける．
- $Q_1 \vee Q_2$ の場合，初めに Q_1 が成り立つ状態にラベル Q_1 を，Q_2 が成り立つ状態にラベル Q_2 を付ける．次に，すべての状態について Q_1 か Q_2 の少なくとも一方がラベル付けされていればラベル $Q_1 \vee Q_2$ も付ける．
- $\mathbf{EX}\, Q$ の場合，初めに Q が成り立つ状態にラベル Q を付ける．$\mathbf{EX}\, Q$ は遷移先の状態で Q が成り立つことを表しているので，ラベル Q が付けられた状態に 1 回の遷移で移ることができる状態にラベル $\mathbf{EX}\, Q$ を付ける．

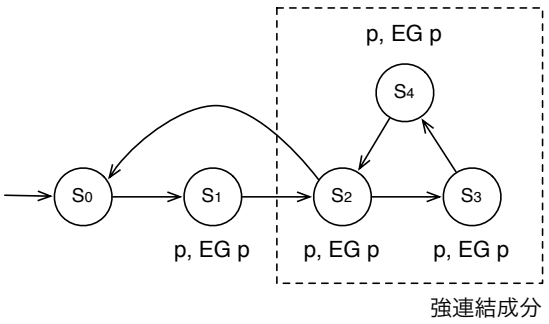


- $\mathbf{EG}\, Q$ の場合，初めに Q が成り立つ状態にラベル Q を付ける．$\mathbf{EG}\, Q$ は現在の状態で Q が成り立っており，さらに現在の状態から遷移を続けてもずっと Q が成り立ち続ける経路があることを意味するので，そのような経路上の状態に $\mathbf{EG}\, Q$ をラベル付ければよい．そこで，まず Q がラベル付けられた状態のみに着目することにして，その中で強連結成分を見つける．強連結成分中の任意の 2 状態間を遷移する間は常に Q が成り立っているため，強連結成分中の各状態に $\mathbf{EG}\, Q$ をラベル付ける．そして，

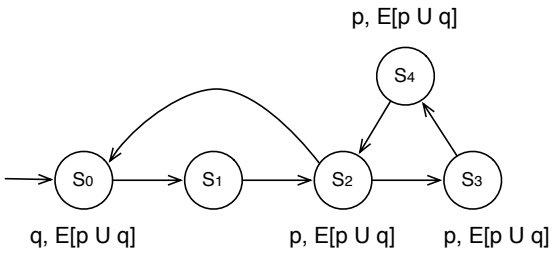
procedure $CheckEU(M, Q_1, Q_2)$
  $Check(M, Q_1)$
  $Check(M, Q_2)$
  $T := \{s \mid Q_2 \in label(s)\}$
  for all $s \in T$ **do** $label(s) := label(s) \cup \{\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2]\}$
  while $T \neq \emptyset$ **do**
    choose $s \in T$
    $T := T - \{s\}$
    for all s' **such that** $R(s', s)$ **do**
      if $\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2] \notin label(s')$ **and** $Q_1 \in label(s')$ **then**
        $label(s') := label(s') \cup \{\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2]\}$
        $T := T \cup \{s'\}$
end procedure

図 6.6　$\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2]$ が成り立つ状態にラベル付けするアルゴリズム

強連結成分中の状態から遷移関係を逆向きに Q がラベル付けされた状態のみをたどって到達可能なすべての状態に $\mathbf{EG}\, Q$ をラベル付ける．



- $\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2]$ の場合，初めに Q_1 が成り立つ状態にラベル Q_1 を，Q_2 が成り立つ状態にラベル Q_2 を付ける．$\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2]$ は Q_2 が成り立つ状態に到達するまでのすべての状態で Q_1 が成り立つことを表す．そこで，Q_2 が成り立つ状態から遷移関係を逆向きにたどって Q_1 が成り立つ状態に $\mathbf{E}[Q_1 \,\mathbf{U}\, Q_2]$ をラベル付ける．逆向きにたどる間は常に Q_1 が成り立っていなければならないことに注意する．



例 6.11 例 6.2 の COPY について $\mathrm{COPY}, s_0 \models \mathbf{EF}\,(p \wedge q)$ であることを示すには，$label_{COPY}(s_0)$ を求めて $\mathbf{EF}\,(p \wedge q) \in label_{COPY}(s_0)$ であるか調べればよい．$\mathbf{EF}\,(p \wedge q) = \mathbf{E}[\top\,\mathbf{U}\,(p \wedge q)] = \mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)]$ であるので $Check(COPY, \mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)])$，つまり $CheckEU(COPY, \top, \neg(\neg p \vee \neg q))$ を計算すればよい．この時，$\top$ はすべての状態で成り立つことに注意する．$Check(COPY, \top)$ および $Check(COPY, \neg(\neg p \vee \neg q))$ の結果として

- $label_{COPY}(s_0) = \{\top, \neg p, \neg q, \neg p \vee \neg q\}$
- $label_{COPY}(s_1) = \{\top, \neg q, \neg p \vee \neg q\}$
- $label_{COPY}(s_2) = \{\top, \neg(\neg p \vee \neg q)\}$
- $label_{COPY}(s_3) = \{\top, \neg q, \neg p \vee \neg q\}$

が得られる．次に，$\mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)]$ がラベル付けられる状態を求めると結果として

- $label_{COPY}(s_0) = \{\top, \neg p, \neg q, \neg p \vee \neg q, \mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)]\}$
- $label_{COPY}(s_1) = \{\top, \neg q, \neg p \vee \neg q, \mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)]\}$
- $label_{COPY}(s_2) = \{\top, \neg(\neg p \vee \neg q), \mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)]\}$
- $label_{COPY}(s_3) = \{\top, \neg q, \neg p \vee \neg q, \mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)]\}$

となり，$\mathbf{E}[\top\,\mathbf{U}\,\neg(\neg p \vee \neg q)] \in label_{COPY}(s_0)$ であるので $\mathrm{COPY}, s_0 \models \mathbf{EF}\,(p \wedge q)$ である． □

練習問題 6.12 例 6.2 の COPY について，

- $\mathrm{COPY}, s_0 \models \neg\mathbf{AF}\,(p \wedge q)$
- $\mathrm{COPY}, s_0 \models \mathbf{AG}\,(q \rightarrow p)$
- $\mathrm{COPY}, s_0 \models \mathbf{EG}\,(\neg q)$

であることを $label_{COPY}(s_0)$ を求めて確認せよ．ここで，$\neg\mathbf{AF}\,(p \wedge q) = \mathbf{EG}\,\neg(p \wedge q) = \mathbf{EG}\,(\neg p \vee \neg q)$，$\mathbf{AG}\,(q \rightarrow p) = \neg\mathbf{EF}\,\neg(\neg q \vee p) = \neg\mathbf{E}[\top\,\mathbf{U}\,\neg(\neg q \vee p)]$ である．

練習問題 6.13 例 6.3 の $Fact_l$ について，

- $Fact_l, s_1 \models \mathbf{EF}\,l_6$
- $Fact_l, s_1 \models \neg\mathbf{AG}\,\mathbf{AF}\,l_6$

であることを確認せよ．また，$Fact_n$ について，

- $Fact_n, s_1 \models \mathbf{AG}\,\mathbf{AF}\,l_6$

であることを確認せよ．

例 6.3 の命題変数 l_6 はプログラムの 6 行目を実行すること，すなわちプログラムが終了することを表しているため，$\mathbf{AG}\,\mathbf{AF}\,l_6$ は「『いつかプログラムが終了する』ことが今後常に成り立つ」という意味である．練習問題 6.13 でみたように，$Fact_l$ の初期状態である s_1 では $\mathbf{AG}\,\mathbf{AF}\,l_6$ が成り立たないが，$Fact_n$ の初期状態である s_1 では成り立つ．検査対象の動作をどのように Kripke 構造で表現するかによって検証できる性質が異なることに注意する．

6.5 性質の記述

モデル検査では，検証したいシステムの性質を CTL などの時相論理の式として記述する必要がある．しかし，システムのあらゆる動作に対して成り立つ（あるいは成り立たない）性質を宣言的に与えることは慣れないと難しい．そこで以下では，システムが満たすべき性質として頻繁に取り上げられる性質と，その時相論理式としての表現について説明する．さらに，性質の自然言語による表現と時相論理式による表現のマッピングを定義する記述パターンを取り上げる．

6.5.1 性質と時相論理式

一般に，システムに対する要求は，システムが実現すべき機能に対する要求（機能要求）と，機能に対する制約や条件を表す要求（非機能要求）に分けられる．機能要求は，入力に対してシステムが期待される出力を行うこととして表現できる．ここでは機能の具体的な内容には立ち入らない．非機能要求としては，性能や使いやすさ，危険な動作をしないこと，予定外の状況が発生しても停止しないこと，あるいは停止すること，など様々な要求が挙げられる．これらの要求を分類しそれぞれに対応する性質を明らかにした上で，各性質を時相論理式によって記述する方法を考えることで，性質を時相論理式として記述する難易度を下げることができる．

システムに期待される典型的な性質として，**活性** (liveness)，**到達可能性** (reachability)，**安全性** (safety) がある．

活性

活性とは，ある条件が成り立てば将来のいつかにある特定の状況が必ず発生すること，を表す性質である．通常，「ある特定の状況」は何か望ましい，発生することが期待されている状況を指す．一般に，入力に対して何らかの出力が期待される機能要求が活性として表現できる．例えば，「料理を注文して待っていれば，いつか料理が出てくる」ことは活性である．

活性の多くは CTL 式として $\mathbf{AG}\,(P \rightarrow \mathbf{AF}\,Q)$ のように表現できる．$P \rightarrow \mathbf{AF}\,Q$ は，P が成り立てば将来のいつかに必ず Q が成り立つことを意味しており，$\mathbf{AG}$ によってそれが常に言えることが表されている．例えば，*order* が料理を注文すること，*served* が料理が出てくることを表すとすると，「料理を注文して待っていれば，いつか料理が出てくる」ことは $\mathbf{AG}\,(\mathit{order} \rightarrow \mathbf{AF}\,\mathit{served})$ と表せる．他の例を以下に示す．

- $\mathbf{AG}\,(\mathit{push\_switch} \rightarrow \mathbf{AF}\,\mathit{power\_on})$
  スイッチを押せば電源が入る
- $\mathbf{AG}\,\mathbf{AF}\,\mathit{green}$
  信号が青に変わる

他に $\mathbf{AG}\,(P \rightarrow \mathbf{E}[P\,\mathbf{U}\,Q])$ も活性を表す CTL 式である．この式は Q が成り立つまで P が成り立ち続けることを表現している．

到達可能性

到達可能性とは，システムが初期状態からある特定の状態に到達する可能性があること，を表す性質である．例えば，「関数 f が呼び出される可能性がある」ことは到達可能性である．また，関数 g の呼び出しが到達可能でなければ，その呼び出しは不要で削除すべきだが残っている，あるいは本来呼び出されるはずだが何らかのロジックが誤っていて呼び出されないようになっている，といった可能性を考えることができる．

到達可能性の多くは CTL 式として $\mathbf{EF}\,P$ のように表現できる．いずれかの経路においていつか P が成り立つことを意味しており，これが到達可能性である．他に，$\mathbf{E}[P\,\mathbf{U}\,Q]$ のように特定の状態に到達するまでの条件を付加することもできる．例を以下に示す．

- $\mathbf{EF}\ \mathit{call\_f}$
  関数 f を呼び出す可能性がある
- $\neg\mathbf{EF}\,(x > n)$
  $x > n$ が成り立つ可能性はない
- $\mathbf{EF}\,(\mathit{lock\_scanner} \wedge \mathit{lock\_printer})$
  スキャナとプリンタをともにロックする可能性がある
- $\mathbf{E}[(i \neq 0)\,\mathbf{U}\ \mathit{call\_f}]$
  $i = 0$ となることなく関数 f が呼び出される可能性がある

安全性

安全性とは，システムがある望ましくない状況に決して陥らないこと，を表す性質である．望ましくない状況の例として，2 つのプロセスが同時に同じ関数を呼び出す，2 つのプロセスが同時に同じメモリ領域に書き込む，デッドロックが発生する，などを挙げることができる．

安全性の多くは CTL 式として $\mathbf{AG}\,\neg P$ のように表現できる．P が望ましくない状況を表しており，いずれの経路のいずれの状態においても P は成り立たないことを意味している．他に，$\neg\mathbf{E}[P\,\mathbf{U}\,Q]$ のような表現もできる．例を以下に示す．

- $\mathbf{AG}\,\neg(\mathit{1\_call\_f} \wedge \mathit{2\_call\_f})$
  2 つのプロセスが同時に関数 f を呼び出すことはない
- $\mathbf{AG}\,\neg \mathit{overflow}$
  メモリのオーバーフローは発生しない
- $\neg\mathbf{E}[\neg \mathit{login}\,\mathbf{U}\,(\mathit{access} \wedge \neg \mathit{login})]$
  ログインしなければデータにはアクセスできない

6.5.2 記述パターン

システムが満たすべき性質を日本語などの自然言語によって表現することは難しくない．しかし，時相論理式によって表現することは慣れていないと難しい[*1]．この難易度を下げるために性質の記述パターンが考えら

[*1] 性質によっては慣れていても難しい

れている．自然言語による性質の表現に頻出する典型的なパターンと，それに対応する時相論理式による表現のマッピングを定義することで，自然言語による性質記述から時相論理式による性質記述を得られるようにするものである．記述パターンでは，システムの実行系列において性質の成否に着目する範囲と，イベントの発生の仕方を分類し，時相論理式によるそれぞれの表現方法をパターン化している．以下で順に説明する．

システムの実行系列において性質の成否に着目する範囲をスコープと呼ぶ．システムの実行中は常に成り立つ性質もあれば，ある状態から別のある状態へ遷移する間だけ成り立つ性質もある．システムの実行においていつ成り立つべきかは性質によって異なっており，スコープはその範囲を指定する．現在は 5 つのスコープが定義されている（図 6.7）．

global　システムの実行系列中のすべての状態を対象とする．
before　指定されたイベントや状態に到達するまでの状態を対象とする．
after　指定されたイベントや状態に到達してからの状態を対象とする．
between　指定されたイベントや状態の間の状態を対象とする．1 つ目のイベントや状態に到達しても 2 つ目のイベントや状態に到達しない場合は対象外とする．
after-until　指定されたイベントや状態の間の状態を対象とする．1 つ目のイベントや状態に到達すれば 2 つ目のイベントや状態に到達しなくても対象とする．

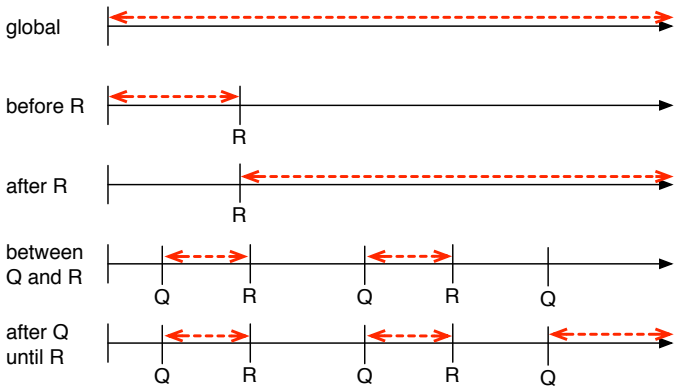


図 6.7　記述パターンのスコープ

例えば，「デッドロックはいかなる時も発生しない」という性質は，「デッドロックが発生しない」という性質に global スコープを適用した性質である．また，「施錠してから解錠するまで扉が開くことはない」という性質は，「扉が開くことはない」という性質に after-until スコープを適用した性質である．将来に渡って解錠されない可能性があるので between スコープではなく after-until スコープになっている．

イベントの発生の仕方は occurrence パターンと order パターンの 2 つに大きく分類される．occurrence パターンは，システムの実行系列において指定されたイベントが発生することや，システムが指定された状態に到達することを表す．スコープを使用して実行系列のどの範囲に着目するか指定する．occurrence パターンはさらに以下の 4 つのパターンに分類できる．

absence　指定されたイベントが決して発生しない，指定された状態に決して到達しないことを表す．
universality　指定されたイベントが常に発生する，指定された状態に常に到達することを表す．
existence　指定されたイベントが発生することがある，指定された状態に到達する可能性があることを表す．

bounded existence　指定されたイベントが k 回発生する可能性がある，指定された状態に k 回到達する可能性があることを表す．バリエーションとして，少なくとも k 回，高々 k 回などがある．

order パターンは，システムの実行系列において特定の順序でイベントが発生することや，特定の順序で状態遷移することを表す．occurrence パターンと同様に，スコープを使用して実行系列の着目する範囲を指定する．order パターンはさらに以下の 4 つのパターンに分類できる．ここで，p, p_i, q, q_i をそれぞれイベントあるいは状態とする．

precedence　イベント q が必ずイベント p に先行して発生する，状態 q には必ず状態 p に先行して到達することを表す．

response　イベント q が必ずイベント p の後で発生する，状態 q には必ず状態 p に到達した後で到達することを表す．

precedence chains　イベント $q_1, \ldots, q_m$ が必ずイベント $p_1, \ldots, p_n$ に先行して発生する，状態 $q_1, \ldots, q_m$ には必ず状態 $p_1, \ldots, p_n$ に先行して到達することを表す．

response chains　イベント $q_1, \ldots, q_m$ が必ずイベント $p_1, \ldots, p_n$ の後で発生する，状態 $q_1, \ldots, q_m$ には必ず状態 $p_1, \ldots, p_n$ に到達した後で到達することを表す．

論理式 P で表される性質の absence を表現した CTL 式を以下に示す．他のパターンと CTL 式のマッピングについては Web サイト[*2] を参照すること．

global	$\mathbf{AG}\,\neg P$
before R	$\neg\mathbf{E}[\neg R\,\mathbf{U}\,(P \wedge \mathbf{EF}\,R \wedge \neg R)]$
after R	$\mathbf{AG}\,(R \rightarrow \mathbf{AG}\,\neg P)$
between Q and R	$\mathbf{AG}\,(Q \wedge \neg R \rightarrow \neg\mathbf{E}[\neg R\,\mathbf{U}\,(P \wedge \mathbf{EF}\,R \wedge \neg R)])$
after Q until R	$\mathbf{AG}\,(Q \wedge \neg R \rightarrow \neg\mathbf{E}[\neg R\,\mathbf{U}\,(P \wedge \neg R)])$

6.6　並行プログラム

本節については，別途資料を配付する．

6.7　並行プログラムのモデル化

並行プログラムの正しさを確認するためにモデル検査を行うには，検査対象の並行プログラムを Kripke 構造で表現する必要がある．並行プログラムは，複数の逐次プロセスが相互作用しながら同時に動作しているプログラムであると考えることができる．そこで，個々の逐次プロセスの状態遷移を定義してそれらを組み合わせることができれば，並行プログラム全体の状態遷移が得られる．そして，各状態に対してその状態で満たされるべき命題を割り当てることで Kripke 構造としてモデル化できる．

モデル化の例として，図 6.8 のような並行プログラムを考える．このプログラムでは，2 つのプロセス P と Q が変数 n を共有している．n の初期値は 0 とする．それぞれのプロセスは 2 行目で共有変数 n に値を書き

[*2] http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml

込むが，2 つのプロセスが同時に書き込むことは禁止したい．本当に禁止できているかモデル検査を用いて確認するために，このプログラムを Kripke 構造によってモデル化する．

プロセス P	プロセス Q
`1: if (n == 0) goto 1;`	`1: if (n == 1) goto 1;`
`2: n = 0; goto 1;`	`2: n = 1; goto 1;`

図 6.8　並行プログラム（ループ版）

初めにプロセス P と Q の状態遷移を考える．ここでは，実行中の行番号 l と変数 n の値の組 (l, n) を状態とする．例えば，状態 $(1, 0)$ は 1 行目を実行中で n の値が 0 であることを表す．プログラムは 2 行，n が取り得る値は 0 または 1 なので，状態数は 4 である．プログラムは 1 行目から実行が始まり，n の初期値は 0 なので，初期状態は $(1, 0)$ である．それぞれのプロセスについて 4 状態間の遷移を考えると図 6.9 のようになる．例えばプロセス P の場合，状態 $(1, 0)$ は 1 行目を n の値が 0 で実行しているということなので，n の値は同じままで 1 行目に進む，つまり状態 $(1, 0)$ に遷移する．この遷移はループになる．状態 $(1, 1)$ であれば 2 行目に進むため状態は $(2, 1)$ に遷移する．状態 $(2, 0)$ であれば，n に 0 を代入して 1 行目に戻るので状態 $(1, 0)$ に遷移する．状態 $(2, 1)$ の場合も同様で状態 $(1, 0)$ に遷移する．プロセス Q の遷移も同様に得られる．

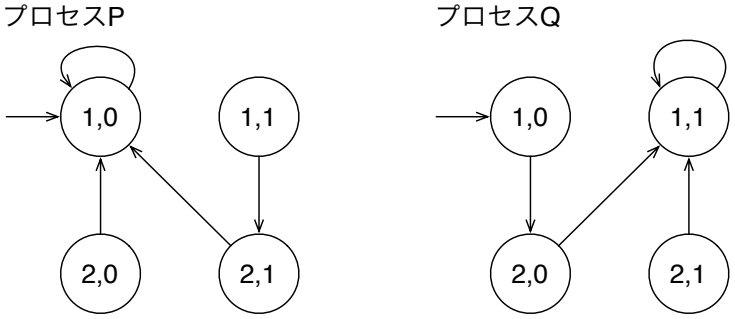


図 6.9　プロセス P と Q の状態遷移

次にプロセス P と Q それぞれの状態遷移を 1 つにまとめて並行プログラム全体の状態遷移を得る．並行動作するプロセスを 1 つにまとめることを**並行合成**というが，合成の方法には同期式と非同期式の 2 種類がある．**同期並行合成**では，すべてのプロセスの状態遷移が同期して実行される（図 6.10 左）．例えば，図 6.9 のプロセス P と Q の状態がともに $(2, 0)$ であるとする．この時，次の遷移で P は状態 $(1, 0)$ へ，Q は状態 $(1, 1)$ へ遷移するが，同期並行合成ではこれらの遷移が同時に発生する．P の状態が $(2, 0)$ のままで Q の状態が $(1, 1)$ へ遷移することはない．**非同期並行合成**では，各プロセスの状態遷移は他のプロセスの状態遷移とは無関係に実行される（図 6.10 右）．通常は 1 回の遷移でいずれか 1 つのプロセスのみが状態遷移するように制限する．個々のプロセスの遷移を互い違いに挟み込むようにして合成することを**インターリーブ**と呼ぶ．

図 6.9 の 2 つのプロセスを同期並行合成して得られる状態遷移を図 6.11 に示す．状態は，プロセス P の行番号 l_P，プロセス Q の行番号 l_Q，共有変数 n の値の組 (l_P, l_Q, n) である．n の初期値は 0 で，いずれのプロセスも 1 行目から実行が始まるので，初期状態は $(1, 1, 0)$ である．状態 $(1, 1, 0)$ では n の値が 0 であるため，プロセス P は 1 行目，Q は 2 行目に進み，状態は $(1, 2, 0)$ に遷移する．状態 $(1, 2, 0)$ でも n は 0 である

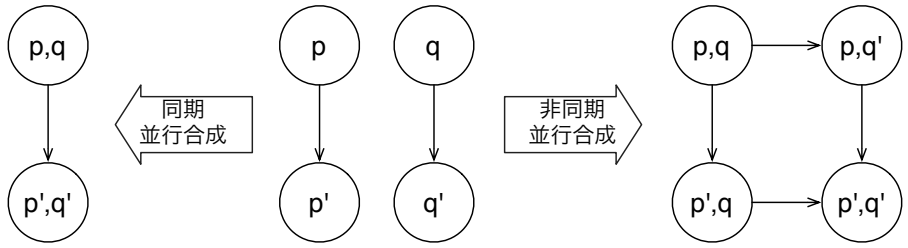


図 6.10　プロセスの並行合成

ため，P は 1 行目に進み，Q は n に 1 を代入して 1 行目に戻るので，状態は $(1,1,1)$ に遷移する．この状態では n が 1 であるので P は 2 行目に進み，Q は 1 行目でループする．よって次の状態は $(2,1,1)$ である．状態 $(2,1,1)$ では P は n に 0 を代入して 1 行目に戻り，Q は 1 行目でループするため，状態 $(1,1,0)$ に遷移する．状態 $(2,2,0)$ や $(2,2,1)$ ではプロセス P と Q の双方で共有変数 n への代入が発生するため，その結果として n の値は 0 または 1 のいずれかになり，遷移先は $(1,1,0)$ と $(1,1,1)$ が非決定的に選択される．

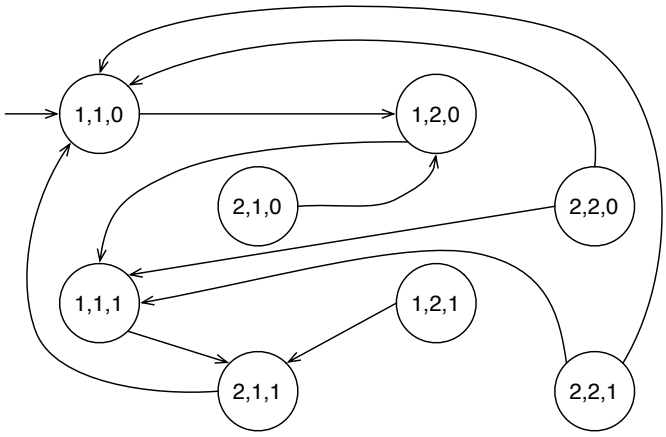


図 6.11　プロセス P と Q の同期並行合成

図 6.9 の 2 つのプロセスを非同期並行合成して得られる状態遷移を図 6.12 に示す．状態は，同期合成の場合と同様にプロセス P の行番号，プロセス Q の行番号，共有変数 n の値の組である．初期状態も同様に $(1,1,0)$ である．各状態について，プロセス P が遷移した場合とプロセス Q が遷移した場合の 2 通りに対応する遷移が存在する．例えば，状態 $(1,1,0)$ において P が遷移した場合，P は n が 0 で 1 行目を実行するので遷移先は $(1,1,0)$ である．一方，同じ状態で Q が遷移した場合，Q は n が 0 で 1 行目を実行するので遷移先は $(1,2,0)$ である．各状態において P と Q のどちらが遷移するかは非決定的に選択される．

練習問題 6.14 図 6.9 の 2 つのプロセスの同期並行合成と非同期並行合成を自分でやってみよ．

プログラムのモデルとして Kripke 構造を得るために，並行合成して得られた各状態に対してその状態で満たされるべき命題を割り当てる必要がある．そこで，次のような命題変数を用意する．$i \in \{1,2\}$ に対して命題変数 P_i は $l_P = i$ を，Q_i は $l_Q = i$ を表すとする．また，$j \in \{0,1\}$ に対して N_j は $n = j$ を表すとする．そして，状態 (p,q,n) に対して命題変数の集合 $\{P_p, Q_q, N_n\}$ を割り当てる．

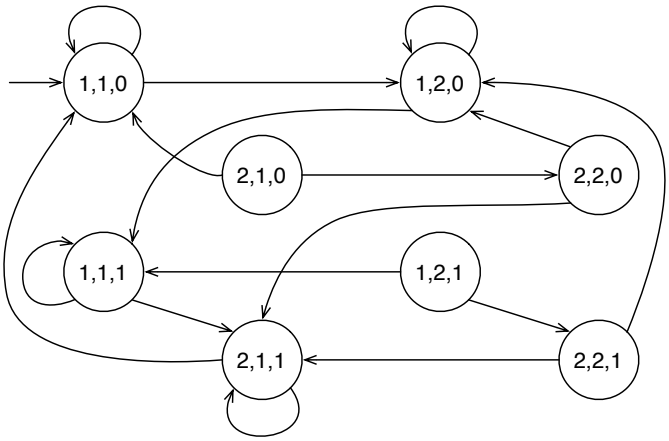


図 6.12　プロセス P と Q の非同期並行合成

図 6.8 の並行プログラムの同期並行合成による Kripke 構造 $M_s = (S, R_s, L)$ は以下のように定義できる．

$$
\begin{aligned}
S &: \{(p,q,n) \mid p,q \in \{1,2\}, n \in \{0,1\}\} \\
R_s &: \{((1,1,0),(1,2,0)),((1,2,0),(1,1,1)),((2,1,0),(1,2,0)),((2,2,0),(1,1,0)),((2,2,0),(1,1,1)), \\
&\quad ((1,1,1),(2,1,1)),((1,2,1),(2,1,1)),((2,1,1),(1,1,0)),((2,2,1),(1,1,0)),((2,2,1),(1,1,1))\} \\
L &: L((p,q,n)) = \{P_p, Q_q, N_n\}
\end{aligned}
$$

一方，非同期並行合成による Kripke 構造 $M_a = (S, R_a, L)$ は以下のように定義できる．S と L は M_s と同じである．

$$
\begin{aligned}
R_a &: \{((1,1,0),(1,1,0)),((1,1,0),(1,2,0)),((1,2,0),(1,2,0)),((1,2,0),(1,1,1)), \\
&\quad ((2,1,0),(1,1,0)),((2,1,0),(2,2,0)),((2,2,0),(1,2,0)),((2,2,0),(2,1,1)), \\
&\quad ((1,1,1),(1,1,1)),((1,1,1),(2,1,1)),((1,2,1),(1,1,1)),((1,2,1),(2,1,1)), \\
&\quad ((2,1,1),(2,1,1)),((2,1,1),(1,1,0)),((2,2,1),(1,2,0)),((2,2,1),(2,1,1))\}
\end{aligned}
$$

図 6.8 の並行プログラムにおいて，2 つのプロセスが共有変数 n に同時に代入するということはプロセス P と Q がともに 2 行目を実行している，すなわち並行プログラムの状態が $l_P = 2$ かつ $l_Q = 2$ が成り立つ状態にあるということである．このような状況を禁止できているかどうかは，Kripke 構造 M_s や M_a において $\{P_2, Q_2\} \subseteq L(s)$ なる状態 s に初期状態から到達できないかどうかを調べればよい．この例の場合は，到達できないことが図 6.11 や図 6.12 の状態遷移図から容易に確かめられる．

練習問題 6.15 Kripke 構造 M_s および M_a において $\{P_2, Q_2\} \subseteq L(s)$ なる状態 s に初期状態から到達できないことを確認せよ．

$\{P_2, Q_2\} \subseteq L(s)$ なる状態に到達しないことは CTL 式を用いると $\mathbf{AG}\,\neg(P_2 \wedge Q_2)$ と表される．そのため，Kripke 構造 M_s や M_a において $\{P_2, Q_2\} \subseteq L(s)$ なる状態 s に初期状態から到達しないことを確認するということは

- $M_s, (1,1,0) \models \mathbf{AG}\,\neg(P_2 \wedge Q_2)$

- $M_a, (1,1,0) \models \mathbf{AG}\,\neg(P_2 \wedge Q_2)$

が成り立つことを示すことと同じである．

練習問題 6.16 $label_{M_s}((1,1,0))$ および $label_{M_a}((1,1,0))$ を求めて

- $M_s, (1,1,0) \models \mathbf{AG}\,\neg(P_2 \wedge Q_2)$
- $M_a, (1,1,0) \models \mathbf{AG}\,\neg(P_2 \wedge Q_2)$

が成り立つことを確認せよ．

図 6.8 の並行プログラムの各プロセスは，1 行目でループによる待機を行っている．ループによる待機では，待機している間もプログラムは動作し続けることになる．一方，スリープのようにプログラムの動作を停止して待機する方法もある．例えば，条件 c が成り立っている間は何もしないで待機することを `sleep(c)` と記述することにすると，図 6.13 のようなプログラムが考えられる．それぞれのプロセスの状態遷移図は図 6.14 のようになる．図 6.9 との違いは状態 $(1,0)$ におけるループ状の遷移がないだけであるが，並行プログラム全体としての動作は大きく異なる．

プロセス P'	プロセス Q'
`1: sleep(n == 0);`	`1: sleep(n == 1);`
`2: n = 0; goto 1;`	`2: n = 1; goto 1;`

図 6.13　並行プログラム（`sleep` 版）

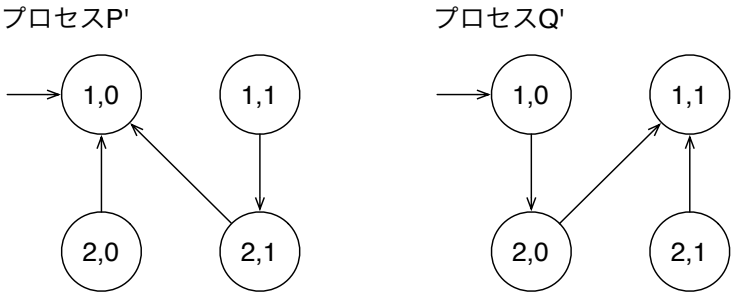


図 6.14　プロセス P' と Q' の状態遷移

図 6.14 の 2 つのプロセスの同期並行合成を図 6.15 に示す．図 6.11 と比較すると，いくつかの遷移がなくなっている．例えば，状態 $(1,1,0)$ では n が 0 であるためプロセス P' は `sleep` を続ける．同期並行合成ではすべてのプロセスが同時に遷移しなければならないが，P' が遷移しないため状態 $(1,1,0)$ では遷移が発生しない．また，図 6.16 に非同期並行合成を示す．非同期並行合成の場合は同時に 1 つのプロセスのみが遷移するとみなすため，図 6.12 との違いはループ状の遷移がないことだけである．ただし，このループ状の遷移の有無が全体の動作に大きく影響していることに注意が必要である．例えば，プログラムがどのように動作しても n の値がいつかは 1 になることがあるか調べると，図 6.12 ではならない場合がある一方で，図 6.16 では n の値はいつかは必ず 1 になる．

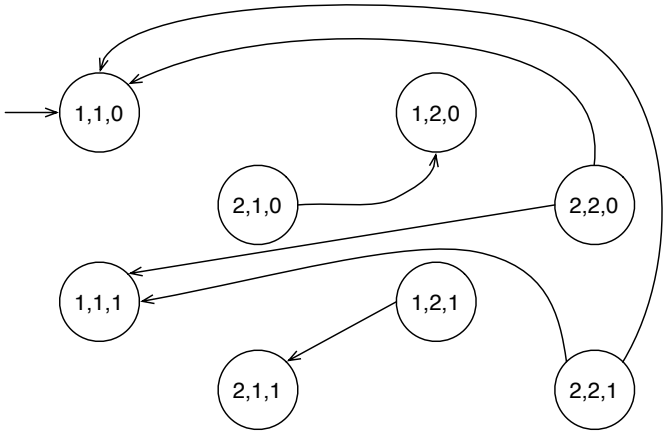


図 6.15　プロセス P' と Q' の同期並行合成

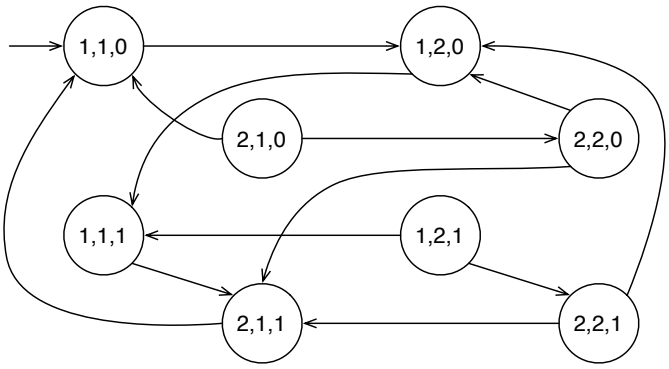


図 6.16　プロセス P' と Q' の非同期並行合成

練習問題 6.17 図 6.12 の状態遷移図において，初期状態から遷移がどのような経路をたどると n の値が 1 にならないか，具体的な経路を 1 つ挙げよ．

練習問題 6.18 図 6.16 の状態遷移図において，初期状態から遷移がどのような経路をたどっても n の値が 1 になることを確認せよ．

練習問題 6.19 「n の値が 1 になる」ことを命題変数 N_1 で表すとする．「すべての経路で n の値がいつかは 1 になる」ことを CTL 式として表せ．

練習問題 6.20 練習問題 6.19 の CTL 式を E で表すとして，$M_a, (1,1,0) \models \neg E$ であることを示せ．

練習問題 6.21 図 6.16 の状態遷移図の遷移関係を書き下せ．

練習問題 6.22 Kripke 構造 $M'_a = (S, R'_a, L)$ を考える．ここで，S と L は M_s や M_a と同じであり，R'_a は図 6.16 の状態遷移図の遷移関係とする（練習問題 6.21）．また，練習問題 6.19 の CTL 式を E で表す．この時，$M'_a, (1,1,0) \models E$ であることを示せ．