モデルの図式表現とその意味: なぜUMLがはやるのか?

玉井哲雄
東京大学 大学院情報学環

1 はじめに

今年 (2001 年) の国際ソフトウェア工学会議 (ICSE2001) は 5 月にカナダのトロントで開催されるが，そのチュートリアル・プログラムを見て驚いた．全部で 22 あるチュートリアルのうち，1/3 にあたる 7 つのタイトルに UML という語が入っている．さらに，タイトルには含まれていないが，10 行足らずの概要説明の中に 3 個所も UML という文字を躍らせているテーマもあって，それを含めると 8/22 という高比率となる．

なんで，これほど UML がもてはやされるのか，正直なところ筆者にはよく分からない．UML は，オブジェクト指向モデルの記述言語ということだが，これまでソフトウェア開発で使われてきた数多くの図式の集大成，というおもむきもある．そして，それらの図表現の中に，過去に培われたモデル化や設計のノウハウが蓄積されているとも言える．オブジェクトモデルの記述言語というよりも，むしろそのようなより広い範囲で用いられる共通言語として，多くの人に受け入れられている，ということかもしれない．

2 グラフによるモデル化

2.1 さまざまなモデルのグラフ表現

多くのモデル化技法では，なんらかの図式表現を用いることが多い．図式にはたとえば次のようなものがある．ここには，UML に入っているものもいないものも挙げている．また，分析時のモデルだけでなく，概要設計やプログラム設計で使われるモデルの図式も含めている．

1. 制御フローを表すもの (フローチャート，PAD, HCP, SPD, など)
2. データフローを表すもの (データフロー図，SADT，HIPO，など)
3. データ構造を表すもの (Jackson 構造図，など)
4. データ関連構造を表すもの (ER 図，意味ネットワーク，オブジェクトモデル図，など)
5. 状態遷移を表すもの (状態遷移図，Statechart, ペトリネット，など)

これらは皆，丸だか四角だか箱といった形が閉じた図形群と，それらを結ぶ折れ線だか点線だか矢印といった線群から構成されるという，共通の特徴を持つ．数学的に言えば，頂点集合と辺集合からなるグラフ構造をとる点が，共通である．

グラフ構造がこれだけよく用いられる理由として，次のような要因が考えられる．

1. 図として視覚化することで「モデル」らしく見える上，描きやすく直観的に分かりやすいこと

2. 対象となる世界の「もの」を頂点で表し,「もの」と「もの」との関係を辺で表すことにより,世界がグラフによって自然に表現されること

3. 「もの」と「もの」との関係には推移律や何らかの推移的な関係が成り立つことが多いが,それはグラフ上の経路の追跡することに対応させて理解でき，また経路探索のアルゴリズムもよく研究されていること

しかし，直観的に分かりやすいことが，恣意的であいまいな使用を助長している面もある．実際，状態遷移図やデータフロー図の概念を学生に説明すると比較的容易に理解するように見えるが，試しに図を描かせてみると，とても状態遷移やデータフローを表していないしろものができてしまうことがよくある．実は，学生ばかりでなく，かなりのベテランのソフトウェア技術者でもそういうケースを見かける．

図としてグラフ構造が愛用されるのは，実はソフトウェアの世界に限らない．新聞，雑誌，教科書，報告書，企画書，論文，発表スライドなどに登場する図の多くは，グラフ構造を表現したものになっている．そして，それらの図の意味は往々にして曖昧である．たとえば，矢印が何を表しているのか，因果関係か，時間関係か，物の流れか，制御の流れか，といった点が，同じ 1 つの図のなかで揺れているものをよく目にする．箱の方も同様で，プロセスのような処理主体と，データのような処理の対象が同じ箱だか丸だか表され，混在していることは日常茶飯事である．

グラフはなまじ直観的に理解しやすいがために，かえってその意味（semantics）をあいまいに捉えがちなのであろう．したがって，おなじグラフ構造であるという共通性と，その意味の違いを意識的に考えることが重要だと思われる．

2.2 グラフ表現によるモデル化の特徴

すでに述べたように，グラフ表現では通常，頂点に「もの」（概念，オブジェクト，プロセス，データなど）を結びつけ，辺にものとものとの関係を結びつける．その辺の役割によって，モデルを，構造を表す静的なモデルと振舞いを表す動的なモデルとに分類することもできる．

1. 静的なモデル

   頂点 A と B を結ぶ辺： A と B がある関係にあることを示す（A から B への有向辺の場合は,「A は B とある関係をもつ」ことを示す）．ER(実体関連）モデル，クラス図，意味ネットワークなどが，代表例.

2. 動的なモデル

   頂点 A から B への辺： A から B へ移動する（制御フロー, 状態遷移など），または A から B へものが流れる (データフロー, ワークフローなど).

静的なモデルと動的なモデルを混同することは少ないが，動的なモデル同士，たとえばデータフローと制御フロー，状態遷移とフローチャートの間の混同はよく目にする．

表 1 に，代表的なモデル図式の頂点と辺の組み合わせを示す.

表 1: 代表的なモデルのグラフ構造

	頂点	辺
データフロー	プロセス	データの流れ
ER	実体	関連
状態遷移	状態	遷移
JSD	プロセス	データストリーム結合 データベクトル結合
フローチャート	実行単位	制御の流れ
ペトリネット	プレース，トランジション	発火とトークンの流れ

2.3 グラフとモデル上の物理量

グラフは頂点と辺の関係というトポロジーを表している．しかし，グラフを用いてモデル化を行う場合は，頂点や辺に何らかの物理量を結びつけて扱うことが通常である．物理量といったが，ここでは「情報」を相手にしているから，多くの場合は物理量とは呼びにくい．しかし，たとえば電気回路のような典型的な物理を対象としたグラフ・モデルは，多くの示唆を与える．

電気回路の場合，頂点に電位，辺に電流が結びつけられる．辺には電位の差としての電圧も対応する．この物理量とトポロジーとを関係づける法則として，キルヒホフの法則 (Kirchhoff's Law) がある．

1. キルヒホフの電流法則

   頂点の周りに流入する電流の総量 (流入を正，流出を負として計る）は，0 である．あるいは，電流は各辺の上を，停滞したりとぎれたりすることなく連続に流れる．

2. キルヒホフの電圧法則

   閉路 (loop) に沿っての電圧の総和は 0 である．

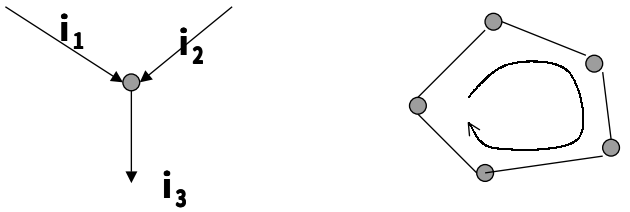


図 1: キルヒホフの法則

これに関連して，辺の接続に関し直列と並列の 2 種類があるが，それぞれの接続における物理的な性質は，次のように記述することができる（これはキルヒホフの法則から導出される）．

1. 直列：　電流は一定．電圧は加算
2. 並列：　電圧は一定．電流は加算

このような「流れるもの」とそれを駆動する圧力がそれぞれ辺と頂点に結びつけられるモデルは多いが，すべてのモデルがそれにあてはまるわけではない．しかし，キルヒホフの法則のように，

1. 各頂点の周りに出入りする辺全体についての性質を考える．
2. 辺が接続する経路に沿って成り立つ性質を考える．

ことは，基本的である．

2.4 グラフ理論の初歩概念の応用

グラフ理論というほど大げさなものは必要ないが，教科書に出てくるようなその初歩の概念は，ある程度役に立つ．

まず，グラフの種類に，いくつかのものが考えられる．それによって，モデルの性質にも違いを生じる．

- 有向グラフ/無向グラフ
- 木，(有向) 無閉路 (DAG)，一般グラフ

  木構造は階層構造と対応し，無閉路構造は共通的な下部構造を持つ階層構造に対応する．一般グラフは階層構造を持たない，いわゆるネットワーク構造と対応する．
- 頂点集合の分割 (e.g. ペトリネット)，辺集合の分割 (e.g. PAD)

グラフの一部をなし，そこに含まれる辺の両端点を頂点集合として含むものを部分グラフという．部分グラフを 1 つの頂点に縮約すれば，その部分グラフをカプセル化，抽象化したグラフとなる (図 2)．この操作は抽象の階層を取り扱う自然な枠組みを提供する．実際，データフロー図や Statechart では，この方法が用いられている．

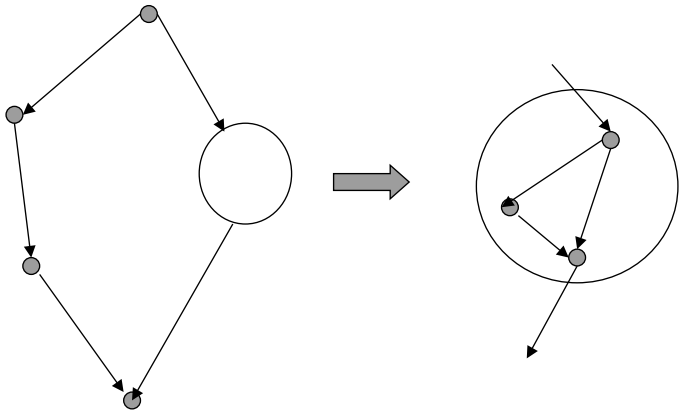


図 2: 部分グラフ

2.5 グラフの弱点と対策

原理的にはどんな複雑なモデルでもグラフとして図示できるとしても，実際に 1 枚の図に描ける頂点や辺の数は限られる．1 つの理由は物理的な制約で，紙やディスプレイの画面は限定された面積しかないから，いくら詰め込んでも高が知れた量しか表せない．さらに認知的な制約がある．よく引用される G. A. Miller の「魔法の数字 7 ± 2」[6] が主張するように，人間が自然に認知できるものの数は，案外少ない．また，頂点の数をある程度押さえても，辺の数が多くてクモの巣状になった図は，とても理解できない．

これはグラフによるモデル表現の弱点であるが，これに対して以下のようにいくつかの対策はある．

- 配置の工夫： 辺の交差をなるべく小さくするような配置や，その他の基準でグラフを見やすくするアルゴリズムがいろいろ提案されている．
- 画面操作の工夫： ズーミングやスクローリングが活用される．
- ハイパーグラフ： グラフの一般化として辺を 2 点間ではなく多点間の関係として扱うハイパーグラフを使うと，場合によっては図が簡素化される．
- 階層化：すでに述べたように，上位グラフの頂点を展開すると下位グラフとなるという階層構造を導入することで，1 レベルの図は簡素化される．この階層構造をツールで支援することも簡単である．

  関連して，頂点の箱に入れ子構造を描けるようにすることも，よく見られる工夫である．

  Statechart では，この両者の工夫が採用されている．

3 例としての状態遷移モデル

グラフ構造を持つモデルの典型例として，状態遷移モデルを見てみよう．状態遷移モデルの歴史は非常に古いが，並行分散システム，応答型システムがますます普及し，その振舞いの検証に対する要請が高まるにつれて，改めて重要性が見直されている [1].

3.1 状態遷移モデルの基本的な性質

3.1.1 グラフ構造と意味論

状態遷移モデルは，典型的な動的モデルである．そのグラフ表記である状態遷移図は，次のように特徴づけられる．

頂点　状態
辺　　遷移

グラフとしての位相構造 (topology) に対応するモデルの構成要素は，この「状態」と「遷移」のみである．しかし，状態遷移モデルに意味 (semantics) を与えるための重要な概念として「事象 (event)」がある．各辺，すなわち遷移には事象がラベルとして結びつけられており，その事象が発生したときに対応する遷移が起こるというように解釈される．

状態遷移モデルで記述される対象は，内部状態を持つ機械，すなわち状態機械の振舞いである．以下では，状態機械の振舞いをプロセスと呼ぶ．事象は状態機械に外部から与えられる入力と見なしてよい．状態の集合の中に，通常はただ 1 つの初期状態が存在することを仮定する．また，状態の

空でない部分集合として，終了状態が存在することも通常は仮定する．ただし，応答型 (reactive) のシステムを状態遷移モデルで記述する場合には，終了状態は存在しないか，特異な状態（deadlock) を指すことになる．

このような仮定のもとに，状態遷移モデルは次のような動的なプロセスを記述するものと解釈される．

1. プロセスは最初は初期状態にいる．
2. ある状態にいて，事象が発生した場合，その状態から出ている遷移の内で，その事象をラベルとして持つものがあれば，その遷移の先の状態に移る．
3. 終了状態に遷移したらそこでプロセスは終了する．

この説明から分かるように，状態遷移モデルでは「現在どこかの状態にいる」ことが仮定されている．現在いる状態は，必ず１つあり，かつ 1 つに限る[1]．ある状態にいて，ある事象が生じたとき，その事象をラベルとしてもつ遷移が常にちょうど 1 つある場合は決定的な状態遷移モデル，複数ある場合は非決定的な状態遷移モデルとなる．また，1 つもない場合は，その事象は無視され，何の遷移も起こらないと解釈される場合と，何らかのエラーと解釈される場合とがある．

遷移は瞬時に起こるものと想定され，状態はある経過時間を持つものと想定される．もちろん，瞬時とか一定の経過時間という概念は相対的であり，モデルで考慮されている時間軸において，無視できる時間間隔を瞬時と見なすわけである．

状態機械を外から見た場合，その機械は記憶を備えたもののように振舞う．つまり，機械の動作は現在の入力によってのみ決まるのではなく，過去の入力の経過に依存し，それを「記憶」していると見えるからである．この「記憶」は実は状態に他ならないが，外からこの状態は見えない．外部と機械が共有するものは事象である．すなわち，状態遷移モデルの２つの構成要素である状態と事象のうち，状態は外からは見えず，事象は外から見える（あるいは外から与える) という区別を意識することは，重要である．

3.1.2 具体例

具体例を示そう．ボーリングの点数を計算するプロセスのモデル化を考えてみる．点数計算の規則は，次のようである．

1. ストライクの得点は，10 に次の 2 回の投球で倒れたピンの本数を加えたものである．
2. スペアの得点は，10 に次の 1 回の投球で倒れたピンの本数を加えたものである．
3. オープン・フレームの得点は，そのフレームの 2 回の投球で倒れたピンの本数で，加算はない．

実はさらに，最後の 10 フレーム目に特殊な処理が必要で，そのための規則があるが，ここではそれは考えないことにする．

図 3 は，この点数計算をするプロセスを状態遷移図でモデル化した例である．この図では，状態集合は { 始，オープン，スペア，ストライク，ダブル，2 投前，ストライク後 2 投前 } である．なぜ，上に述べた規則には現れない「ダブル」という概念が現れているのだろうか．逆に，3 回連

[1]ただし，これは逐次プロセスの場合である．並行プロセスの場合は，後で述べるように「現在の状態」が状態の集合を指すことが一般的である．

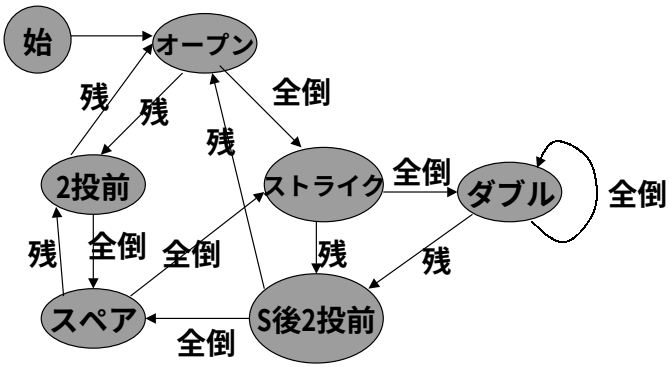


図 3: ボーリングの点数計算の状態遷移モデル

続ストライクのターキー（やフォースやそれ以上) という状態を入れていないのだろうか．あるいはフレームという概念に相当するものが直接現れなくてよいのだろうか．

また，事象集合は { 全倒，残 } である．舌足らずな言い方だが,「全倒」は立っているピンが全部倒れたという事象（1 投目ならストライクだし，2 投目ならスペアになる) を表し,「残」は 1 本以上ピンが残ったことを示す．なぜ，倒したピンの本数か残った本数によって事象を区別しないのだろうか．

これらはモデル構築の方針によって選択されたものである．モデルを作るには，目的がある．目的に必要なことがらが選択され，不要なものは捨象されるという「抽象化」がなされなければならない．ここでは，ボーリングの点数を計算するのに必要な場合分けを表すような状態を考えている．だから，事象は投球をしてピンが倒れた時点を取り，1 投目か 2 投目かを直接には区分せず，全部倒れたか否かという 2 つに 1 つの区別に抽象化しているのである．

この状態遷移図では終了状態を明示していない．上に述べたような抽象化をしているために，フレームという概念は表現されておらず，実際のボーリングのゲームが 10 個のフレームで終了するとしても，そのことはここで記述しているプロセスにとっての終了とは直接関係しないといってよい．

3.2 状態遷移モデルの拡張

状態遷移モデルは，さらに概念や記法を付加して拡張されることが多い．ここではとくに，出力事象，遷移条件，並行システム，状態の階層化という 4 項目を取り上げる．これらは D. Harel の Statecharts で採用されており，その Statecharts を採用している UML でも踏襲されている．

3.2.1 出力事象

これまで事象は状態機械にとっての入力であるとしてきた．機械といいながら，入力のみで出力がないのでは，役に立たないのではないだろうか．しかし，状態遷移モデルのもっとも簡素な形である有限状態オートマトンでは，出力がない．そこでは一連の入力事象の列にしたがってオートマトンが状態を遷移させていったとき，終了状態に達したらそれまでの入力列は受容されたとし，そうでなければ受容されないと判定することで，１つの言語を定めている，と解釈される．

しかし，やはり出力を扱えた方が，モデルの記述力は増す．そこで自然な拡張として，遷移に伴い状態機械から外界に向けて出力事象を発生するようなモデルが考えられる．出力事象を伴う状態遷移モデルは，有限状態オートマトンを拡張した順序機械に対応する．

例として，図 3 のボーリングの点数計算を取り上げる．このモデルは本来，点数を計算するための情報を出力しないと役に立たない．そこで，出力事象として，次の 3 つを考えることにする．

1. 加算 1（倒したピンの数を加える）
2. 加算 2（倒したピンの数の 2 倍を加える）
3. 加算 3（倒したピンの数の 3 倍を加える）

ここで，括弧書きしたものは，この出力事象を受け取る外部の計算主体が行うはずの計算をコメントとして書き加えたもので，図 3 の状態遷移モデルにとっては，3 つの事象が区分されるということのみが意味を持つ．なお，通常のフレーム単位の点数計算に合わせるなら，

1. 加算 1（倒したピンの数を現在のフレームに加える）
2. 加算 2（倒したピンの数を現在と 1 つ前のフレームに加える）
3. 加算 3（倒したピンの数を現在と 1 つ前と 2 つ前のフレームに加える）

という方がよいかもしれない．とくに 10 フレーム目の処理を考えると，この表現の方が扱いやすい．

この出力事象を加えて図示したのが，図 4 である．遷移を表す辺は,「入力事象/出力事象」によってラベルづけされている．

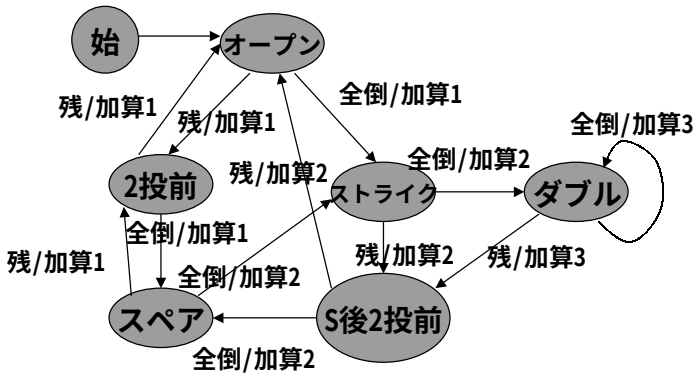


図 4: ボーリングの点数計算モデル (出力事象つき)

3.2.2 遷移条件

ある状態にあって，そこからの遷移を起こす事象が発生した場合に，常に遷移が起こるのでなく，さらにある条件が満たされたときにのみ遷移が起こるという記述をしたいことがある．とくに後で述べる並行システムの場合，遷移条件を書けることで，状態の数の節約になることが多い．

3.2.3 並行システム

並行システムは複数の状態遷移プロセスの共存という形で表現できる．複数の状態遷移プロセスを並べることで,「現在いる状態」は 1 箇所ではなく，プロセスの数だけあることになる．それぞれのプロセスがまったく独立に動作するのでは並行システムとしての面白さはないが，各プロセスは事象を共有することで相互作用を起こす．事象を共有するとは，1 つの入力事象の発生でそれをラベルにもつ複数の遷移が同時に起こることや，あるプロセスの出力事象が別のプロセスの入力事象になること，などを意味する．

3.2.4 状態の階層化

状態遷移モデルをシステム開発のためのモデル分析に使用する際のもっとも大きな問題点は，実用規模の問題に対し，ほとんど常に状態数が爆発的に大きくなってしまうことである．この状態数の爆発に対処するにはいくつかの方法が考えられるが，有効なものの 1 つは，いくつかの状態を包含した親状態 (superstate) を導入することである．包含されている状態を子状態と呼ぶとすると，プロセスが親状態に含まれる任意の子状態にある時，またそれらの子状態の間でのみ遷移が起こっている間は，1 つの親状態にあると見なす．親状態の外から中への遷移や，親状態の中からその外への遷移が，その階層レベルでの遷移になる．これは，部分グラフを 1 つの頂点として階層構造を作るという一般的な方法の例である．

3.3 Statecharts

前節で述べたような状態遷移モデルの拡張を導入し，実用規模のシステムに対して起こる「状態数の爆発」に対処した手法の代表的なものとして，D. Harel による Statechart がある．Statecharts の特徴を挙げると，次のようである．

1. 遷移には事象と，場合により条件や動作が結びつけられている．
2. 複数の状態をまとめたクラスタを導入できる．クラスタは内部を隠して抽象化された一つの状態とみなすことができる．逆に一つの状態を複数の状態からなる部分遷移モデルに詳細化して示すこともできる．
3. クラスタに対してデフォルトの初期状態を指定することができる．
4. 過去の遷移の「歴史 (history)」を使って，たとえばもっとも最近あるクラスタを離れた時にいた状態に入る，といった指定ができる．
5. And 独立な状態群が並行的に動作することを，記述できる．
6. 遷移の分岐 (fork) と合流 (join) を記述できる．

3.4 UML における状態遷移図

UML では，オブジェクトを状態機械と考え，その振舞いを状態遷移図で表す．記法としては，基本的に Harel の Statechart を採用している．たとえば，図 5 は Fowler の本 [2] からとった．

図の「動作中」と名前を付けているのが親状態 (superstate) である．その中に 3 つの子状態があるが，それを隠して 1 つの状態と見なしたものが「動作中」である．遷移の矢印に付けられた

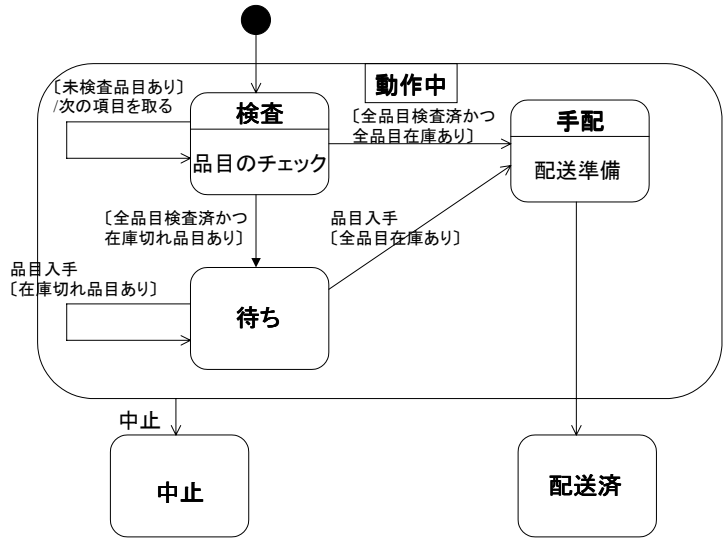


図 5: 親状態のある状態遷移図

ラベルのうち, 〔〕で囲われた表現は遷移が成立する条件を記述するものである．単なる文字列は, 遷移を引き起こす入力事象を示す．また，/の後に書かれた文字列は，遷移に際して出力として生じる事象を表す.

3.5 状態遷移モデルの系譜 (1)–オートマトン

状態遷移モデルの歴史は古い．ここでその系譜を振り返ってみることは，状態遷移モデルの本質を知るのに役立つだろうとの意図からである．その系譜を大きく 2 つに分けてみた．1 つは有限状態オートマトンに代表されるオートマトンの系譜である．もう 1 つは AI や OR でこれまた長く取り扱われてきた探索手法の系譜である．本節では，オートマトンの方を取り上げる．

3.5.1 有限状態オートマトン

計算機科学の教科書なら必ず有限状態オートマトンと正規表現の関係について，かなりのページが割かれているはずである．また，コンパイラや言語処理の本でも，字句解析や形態素解析の方法として，有限状態オートマトンは避けて通れない.

有限状態オートマトンの定義は改めて述べるもでもなく，3.1.1 で導入したモデルそのものである．有限状態という以上，状態集合は有限であることが仮定されている．正規表現の定義はどこででも見つけられるだろうが，ここでも煩を厭わず載せておくことにする.

アルファベット Σ 上の正規表現を考える．正規表現は，次のように構成的に定義される.

R1 ϵ(空列を示す)，$\emptyset$(空集合を示す)，および Σ の任意の要素は，それぞれ正規表現である.

R2 R_1 と R_2 が正規表現の時，$R_1 \cdot R_2$ および $R_1 \cup R_2$ は正規表現である．R が正規表現の時，R^* は正規表現である.

正規表現は，Σ を文字とする文字列からなる集合の部分集合（この部分集合を言語と呼ぶ）に写像される．その写像を与える規則は，次のようである．

R1' $\epsilon, \emptyset, a \in \Sigma$ は，それぞれ $\{\epsilon\}, \emptyset, \{a\}$ に写像される．

R2' 正規表現 R_1 と R_2 が写像された言語を L_1, L_2 とすると，$R_1 \cdot R_2$ は $L_1 \cdot L_2$ に，$R_1 \cup R_2$ は $L_1 \cup L_2$ に写像される．ただし，$L_1 \cdot L_2 = \{a \cdot b | a \in L_1 \ \& \ b \in L_2\}$．ここで文字列に対する演算 $\cdot$ は，連接である．

正規表現 R が写像された言語を L とすると，R^* は L^* に写像される．ただし，$L^0 = \{\epsilon\}$, $L^i = L^{i-1} \cdot L$ として，$L^* = \bigcup_{k=0}^{\infty} L^k$.

さて，有限状態オートマトンが受容する言語のクラスと，正規表現で記述できる言語のクラスとが同じであるというのが，Kleene による有名な定理である．証明は省略するが，直観的には明らかに見える．

ある言語を正規表現や有限オートマトンで表すことは，頭の体操としても面白い．以下の言語を正規表現ないし有限オートマトンで表してみよ．

1. 入力が 0 か 1 だとして，1 が偶数個ある文字列のみを表す正規表現，あるいはそれのみを受理するような有限オートマトンを作りなさい．

2. 入力が 0 か 1 だとして，それを順に左から右に並べて得られる 2 進数が 3 の倍数のもののみを表す正規表現，あるいはそれのみを受理するような有限オートマトンを作りなさい．

3.5.2 順序機械

順序機械は有限状態オートマトンに出力が加わったものである．主に，論理回路の設計や解析で用いられてきた．その形式的な定義は次のとおりである．

順序機械 M は，5 組 $(Q, X, Z, \delta, \omega)$ によって定義される．ここで

Q: 状態集合
X: 入力集合
Z: 出力集合
δ: 状態遷移関数，$X \times Q \to Q$
ω: 出力関数，$X \times Q \to Z$

状態および入力の集合が有限であれば，状態遷移関数と出力関数は表として表すことができる．なお，上の定義はいわゆる Mealy 型機械と呼ばれるもので，出力関数 ω を，入力に無関係に状態だけで決まるようにしたもの，すなわち ω: 出力関数，$Q \to Z$ という形のものは，Moore 型機械と呼ばれる．

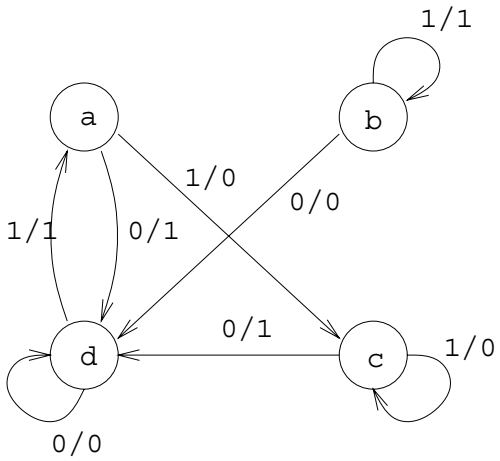
順序機械をグラフとして表現する場合は，頂点には対応する状態をラベルとして与え，辺には，その遷移を起こす入力とその時に出される出力の対を，ラベルとして与える．たとえば，下の表は $Q = \{a, b, c, d\}, X = Z = \{0, 1\}$ であるような状態遷移と出力の表であり，図はそれに対応する状態遷移図である．

遷移関数表

	0	1
a	d	c
b	d	b
c	d	c
d	d	1

出力関数表

	0	1
a	1	0
b	0	1
c	1	0
d	0	1



3.5.3 LTS とモデル検査

ラベルつき遷移システム (Labeled Transition System) は，並行性を組み入れた状態遷移モデルと同じと考えてよいが，最近,「モデル検査」と呼ばれる手法とともに脚光を浴びている [1]．システムの振舞いを LTS で記述し，それに要求される性質（たとえばデッドロックに陥らないとか，ある望ましい状態にいつかは必ず達するとか）を，多くの場合，時相論理 (temporal logic) 式で与え，それをモデル検査という手法で確かめるのである．この手法の特徴は，自動的に結果が出ることと，検証できない場合に反例が提示されることである．この手法は，1980 年代の前半に提唱され，とくにハードウェアの論理回路や通信プロトコルの正しさの検証に用いられてきたが，この数年，一般のソフトウェア（とくに並行分散システムの振舞いが問題となるもの) を対象とする研究や実践が盛んになってきている．

並行システムの形式的なモデル化と，その性質の数学的な取り扱いについては，C. A. R. Hoare の CSP(Communicating Sequential Processes) や Robert Milner の CCS(Calculus of Communicating Systems) が有名である．これらについては，それぞれの著者の教科書を読むとよい [4, 7].

3.6 状態遷移モデルの系譜 (2)–探索問題

AI や OR の分野で，古くから探索問題が研究されている．手法は解の空間を探索して目標を見つけるという形をとるが，通常はグラフ上の経路をたどって目標となる状態に達するという定式化がなされる．

1 つの典型は,「山羊と狼とキャベツ男」とか「宣教師と土人」といったクイズである．また，関連して，ゲーム，とくに 2 人の盤ゲームが類似の定式化で分析される．そこでは「盤面」が状態に対応し,「手」が遷移に対応する．

また，ロボットの行動計画 (planning) の問題も，状況とそこで可能な操作の選択を計画するという形で，状態遷移モデルに帰着する．さらにこれらを一般化した Simon & Newell による GPS(General Problem Solver) も同じ形式といってよい．

川合の作った問題を借用して，状態遷移の概念を使うときれいに解が求まるものを挙げる．なお，彼の「コンピューティング科学」[5] は名著である．

1. 3 個のランプ付きボタンが一列に並んだ装置があり，次のように振舞う．

あるボタンを押すと，そのボタンのランプ状態 (点灯・消灯) が反転するとともに，
そのボタンの隣のランプの状態も反転する．

真中のボタンには「隣」が 2 個あることに注意すること．最初はランプがすべて消えているものとする．真中のランプだけを点灯させるためには，ボタンを最低何回押せばよいか．

2. 容量 a リットルと b リットルの 2 つのバケツを使って，どちらかのバケツに c リットル残るようにする問題を考える．a と b は正の整数とする．実行できる操作は次のとおり．

   (a) 1 つのバケツを蛇口からの水でいっぱいにする．

   (b) 1 つのバケツに入っている水を全部捨てる．

   (c) 片方のバケツの水を，他方のバケツへ移せるだけ移す．全部移ることもあれば，移される側がいっぱいになった時点で終わる場合もある．

   初めは，両方のバケツとも空とする．$a = 4, b = 5, c = 2$ の場合に，最低何回の操作で解に達するか．

参考文献

[1] Clarke, E. M. Jr., Grumberg, O., and Peled, D. A.: *Model Checking*, MIT Press, 1999.

[2] Fowler, M. and Scott, K.: *UML Distilled: Applying the standard object modeling language*, Addison-Wesley, 1997.

[3] Harel, David: StateChart: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol. 8(1987), pp. 231–274.

[4] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall, 1985.

[5] 川合慧：「コンピューティング科学」，東京大学出版会，1995.

[6] Miller, G. A.: The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, *The Psycological Review*, Vol. 63(1956), pp. 81-97.

[7] Milner, R.: *Communication and Concurrency*, Prentice-Hall, 1989.