■7 群（コンピュータ-ソフトウェア）- 1 編（ソフトウェア基礎）

2 章 ソフトウェア検証

（執筆者：関　浩之）［2009 年 5 月 受領］

■概要■

大規模なコンピュータシステムが複雑に作用しあう今日の高度情報社会では，一つのソフトウェアの不具合が社会に大きな損害をもたらす危険性をはらんでいる．したがって，ソフトウェアの信頼性を向上させるための組織的な方法論を構築することは重要な課題である．この課題は「あなたの作ったプログラムが正しいことをあなたはどうやって保証しますか」という素朴な問いに帰着する．ソフトウェアが意図したとおりに振る舞うとき，そのソフトウェアは（その意図に対して）正当性をもつと呼ぶことにする．純粋に論理的存在であるソフトウェアをモデル化するのにふさわしい体系を構築し，その体系のもとで，ソフトウェアの正当性を保証する手法を見いださねばならない．

ソフトウェアの正当性を保証する手法は，しばしばテスト法と検証法に分類される．テスト法では，与えられた仕様からテストスィートを自動生成し，それをテスト対象のソフトウェアに入力したときの振る舞いを観測することによって不具合の検出を行う．検証法では，検証対象のソフトウェアが与えられた仕様を満たしているかどうかを厳密に判定する．検証法はその判定技法によって，定理自動証明法とモデル検査法に分類される．定理自動証明法では，対象ソフトウェアの意味を公理系によって定義し，仕様で要求される性質がその公理系のもとで定理として成立することの証明を与えることによって正当性を保証する．定理証明は一般に人間と証明支援系の協力によって行われる．モデル検査法では，対象ソフトウェアの状態空間を網羅的に検査して正当性の可否を判定する．どの手法を用いるのが適切であるかは，正当性を保証すべき対象の規模や構造，許容できるコスト，検証に関与できる人間の能力に依存して決めるべきである．例えばソフトウェアはコピーによって実用上劣化が全く生じないのに対し，ハードウェアの生産ラインにおいては一定の欠陥品が生じる．このような欠陥品を実用的なコストと精度で発見するにはハードウェアのテスト法はなくてはならない技術である．また，ある不具合モデルのもとでは不具合を必ず検出できるテスト法も種々知られている．一方，交通，金融，医療に関するシステムなど，その障害が社会に大きな影響をもたらすシステムをクリティカルシステムと呼ぶ．クリティカルシステムの正当性を保証するには，正当性を完全に保証できるような検証法を適用すべきである．テスト技法については他群，他編で扱われているので，本章では，ソフトウェア検証法のいくつかの話題に絞って解説を行う．

ソフトウェア検証法の研究は，40 年を越える歴史をもつ．ここでは便宜的に 10 年単位でそれらをごく簡単に俯瞰する（ただし定理自動証明についての解説は筆者の能力を超えるので他群，他編を参照されたい）．1960 年代にプログラミング言語の意味定義の研究と相まって，プログラムの正当性検証（プログラム検証と略）の先駆的研究が始まった．代表的な手法に，フロイド・ホーアの公理論的手法がある．1970 年代に入ると，プログラミング言語の意味論の研究は，スコットらによる表示的意味論をはじめとし更に深化していった．一方，より工学的な見地から形式（数理的）手法（Formal Method）と総称される研究が活発に行われた．例えば，ソフトウェアの仕様自身を厳密に定義し，そこから段階的詳細化によって仕様を満たすソフトウェアを導出しようという形式的仕様記述法が盛んに研究された．なかでも，集

合論に基づく手法（VDM, Z, B）や抽象的データ型を代数系として定義する代数的仕様記述法（1-4 節）は，後者の操作的意味論と深い関係のある項書換え系（1-3 節）などとともにそれぞれ重要な研究潮流となり現在に至る．1980 年代には，現在主要なソフトウェア開発技法となっているオブジェクト指向が新しいパラダイムとして注目を集め，その形式的技法も盛んに研究されはじめた．一方，1980 年代初頭，クラーク，エマーソンらによる時相論理モデル検査の研究に端を発し，モデル検査の研究が大きく躍進した．1990 年代に入り，モデル検査の実用規模の問題への適用が試みられるなか，状態爆発の問題を解決するための種々の技法が考案され（3-4 節），それらを実装した SPIN, SMV といった優れたモデル検査器が広く利用されるようになった．一方，Isabelle/HOL, Coq, Agda といった，高階論理や型付きラムダ計算に基づく証明支援系の開発及びそれらを用いた検証法の研究も盛んに行われている．

【本章の構成】

本章初版では，限られた執筆期間でソフトウェアの検証法を網羅的に解説することを目指すのでなく，現在この分野で活躍する第一線の研究者に，各々の専門領域またはその基礎となる話題に絞って執筆をお願いした．具体的に，2-1 節ではプログラム検証の先駆的理論であるホーア論理，2-2 節では高階論理に基づく代表的な証明支援系の一つである Isabellel/HOL について解説する．2-3 節では有界モデル検査や制約充足などにも応用されている SAT ソルバについて述べる．2-4 節ではモデル検査法について概説する．モデル検査法の詳細は本編 3 章にて解説しているのでそちらも参照されたい．2-5 節ではオブジェクト指向開発手法におけるモデリング言語である UML に基づく数理的技法，検証法について述べる．

■7 群 - 1 編 - 2 章

2-1 ホーア論理

（執筆者：村上昌己）[2008 年 5 月 受領]

2-1-1 プログラムの正当性証明

本節で解説するホーア論理は，プログラム検証の理論において最も古くからよく知られた手法の一つである．ホーア論理について最初に報告された文献は，ホーア（C. A. R. Hoare）による 1969 年の文献 1) にさかのぼる．この論文でホーアは，それより先にフロイド（Floyd）らによって報告されていたプログラムの正当性の証明法[2)]について，自然演繹法の形式を模した公理的な手法を持ち込む手段を示した．本節では，簡単な手続き型のプログラミング言語に対してホーア論理の基本的な公理系を構成した例を示し，それを用いたプログラムの検証の例を示す．続いて公理系の健全性／完全性に関する結果について簡単に紹介する．また以後に示された結果との関連について簡単に触れる．本節で扱った話題に加えてさらに広く関連研究について紹介している文献としては文献 3) などがあげられる．またプログラムの正当性証明についての入門的な教科書としては，古典的な著書になるが文献 4, 5, 6) などがあげられる．

(1) 部分的正当性

本節では通常の第一階述語論理で用いられる用語のうち，述語記号，定数，関数記号，変数，自由変数，解釈，変数割当などの用語は通常どおりの意味に用いる．

以下では $\bar{x}$ で変数の並び $x_1, x_2, \ldots, x_n$ を表すことにする．また $P(\bar{x},\bar{y})$ で $\bar{x}$ を入力 $\bar{y}$ を出力とするプログラムを表す．ここではプログラムの入力値及び出力値は，ある対象領域 D の要素であるものとする．このようなプログラムを D 上のプログラムと呼ぶ．

また $\phi(\bar{x})$ を $\bar{x}$ に出現する変数以外の自由変数を含まない論理式，同様に $\psi(\bar{x},\bar{y})$ を $\bar{x},\bar{y}$ に出現する変数以外の自由変数を含まない論理式とする．ここでは各論理式に出現する述語記号，定数記号，関数記号の解釈はプログラムの対象領域 D 上のある解釈 I に固定されており，I のもとでの各記号の解釈はプログラム P 中に出現する同じ記号の意味と一致しているものとする．D の要素の並び $v_1, \ldots, v_n$ を $\bar{v}$ と表記する．$\bar{x}$ に $\bar{v}$ を割り当てるとは，$1 \le i \le n$ であるそれぞれの i について 変数 x_i に対して D の元 v_i を割り当てることをいう．論理式 $\phi(\bar{x})$ が値 v について真とは，その解釈のもとで $\bar{x}$ に $\bar{v}$ を割り当てる変数割当のもとでの $\phi(\bar{x})$ の真理値が 1 となることをいう．$\psi(\bar{x},\bar{y})$ が $\bar{v},\bar{u}$ について真であることも同様に定義される．

ここで部分的正当性とは以下のように定義される．

(a) 部分的正当性（partial correctness）

D 上のプログラム $P(\bar{x},\bar{y})$ が入力条件 $\phi(\bar{x})$ と出力条件 $\psi(\bar{x},\bar{y})$ について部分的正当であるとは，任意の値の並び $\bar{u}$ について以下のような条件が満足されることをいう．

> $\phi(\bar{x})$ が $\bar{u}$ について真でありかつ $\bar{u}$ を入力として $P(\bar{x},\bar{y})$ を実行したとき実行が正常に終了して出力の値が $\bar{v}$ となるならば，$\psi(\bar{x},\bar{y})$ が $\bar{u},\bar{v}$ について真となる．

例 1 (部分的正当性) 以下のようなプログラムを考える．D を自然数の集合，“=”, “+” などの数式記号は通常どおりの意味を表すとする．このとき $\phi(x)$ を $x \ge 1$, $\psi(x,y)$ を $y = x(x+1)/2$ とする．このとき以下のプログラムは，入力条件 $\phi(x)$ と出力条件 $\psi(x,y)$ について部分的正

当である．

```
begin
  z := 0; y := 0;
  while z ≠ x do
    begin
      z := z + 1; y := y + z
    end
end
```
(2･1)

部分的正当性はその定義から分かるように，プログラムが正常に停止したという仮定のもとで出力値の満足するべき条件が成立することを主張するものである．したがって部分的正当性が示されたとしても，プログラムが停止するかどうかは保証されない．したがって与えられた入力値についてプログラムの実行が停止することの証明は別途与える必要がある．

2-1-2 ホーアの公理系

以下ではこの部分的正当性を証明論的に公理系を用いて検証するための体系について解説する．証明論的な手法では，対象となる式の構文が明確に定まっていなければならない．ここで用いる式の構文を定めるには，対象となるプログラムの構文と入力条件及び出力条件を記述する述語の構文を定めなければならない．本節では，述語の構文は上で述べたような数式記号 “=”, “≤”, “+” などを通常どおりに用いる．また読みやすさのために，既に例にあったような通常の分数の記法や中置記法の等号, 不等号などを用いることを許すこととする．

(b) プログラムの構文

プログラムの構文は，以下の BNF で定められる文 P から構成されるものであるとする．

$$
\begin{aligned}
P \quad ::= \quad & x := t \quad \text{[代入文]} \mid \\
& \textbf{begin } P; P \textbf{ end} \quad \text{[連接文]} \mid \\
& \textbf{if } \phi \textbf{ then } P \textbf{ else } P \quad \text{[条件分岐文]} \mid \\
& \textbf{while } \phi \textbf{ do } P \quad \text{[繰り返し文]}
\end{aligned}
\tag{2･2}
$$

代入文の右辺 t は項，左辺 x は変数であるとする．本節ではここでも判りやすさのため通常の数式表現を用いることを許す．条件文及び繰り返し文に出現する条件 ϕ は上で定めた述語の構文によって与えられる論理式で束縛記号を含まないものとする．

以下では連接文 **begin** S_1; **begin** $S_2; \cdots ; S_n$ **end end** を **begin** $S_1; S_2; \cdots ; S_n$ **end** と，また **begin begin** $S_1; \cdots ; S_{n-1}$ **end**; S_n **end** を **begin** $S_1; \cdots ; S_{n-1}; S_n$ **end** と略記することを許す．

各文が直観的に意味するところは通常と同様である．すなわち代入文 $x := t$ は変数 x の値を t の値で置き換えることを意味する．連接文 **begin** $P_1; P_2$ **end** は， P_1 の実行を行った後ただちに P_2 を実行することを意味する．条件文 **if** ϕ **then** P_1 **else** P_2 はこの文の実行開始時の各プログラム変数の値について ϕ が真であるときは P_1 を実行し，ϕ が偽であるときは P_2 を実行する．繰り返し文 **while** ϕ **do** P_1 はこの文の実行開始時の各プログラム変数の値について ϕ が偽であればただちに実行を終了し，真であれば P_1 を実行した後にこの繰り返し文

自身を再度実行する．言い換えればよく知られているとおり，ϕ が真である間は P_1 の実行を繰り返し偽になったら終了することと同等である．

（c）ホーア記法

ホーア論理では $P(\bar{x},\bar{y})$ が入力条件 $\phi(\bar{x})$ と出力条件 $\psi(\bar{x},\bar{y})$ について部分的正当であることを以下のような式を用いて表す．以下ではこれをホーア記法と呼ぶことにする．

$$\{\phi(\bar{x})\}\ P(\bar{x},\bar{y})\ \{\psi(\bar{x},\bar{y})\} \tag{2･3}$$

例 2 以下は先の例 1 で示した (2･1) の部分的正当性をホーア記法で記述したものである．

$$\begin{array}{l}\{x \geq 1\}\\ \quad \textbf{begin}\\ \quad\quad z := 0;\ y := 0;\\ \quad\quad \textbf{while}\ z \neq x\ \ \textbf{do begin}\ z := z + 1; y := y + z\ \textbf{end}\\ \quad \textbf{end}\\ \{y = x(x+1)/2\}\end{array} \tag{2･4}$$

（d）公理系

以下にホーア論理の公理系を示す．公理系は，領域上で真である式とプログラムの各構文規則に対応する推論規則などから構成される．

各推論規則は，以下のようなたかだか二つの論理式またはホーア記法からなる前提とちょうど一つのホーア記法からなる結論の組である．以下で横線の上が前提を，下が結論をあらわす．

ここで $\phi(\bar{x})[t/x_i]$ は $\phi(\bar{x})$ 中の自由変数 x_i のすべての出現を t で同時に置き換えて得られる論理式を表す．ここで t 中に x_i が出現してもかまわないことに注意されたい．また $\forall\phi$ は論理式 ϕ の全称閉包，すなわち ϕ 中のすべての自由変数を全称束縛して得られる閉論理式を表す．

領域公理　　対象領域上で真である任意の閉論理式

代入規則

$$\frac{\forall(\phi(\bar{x}) \rightarrow \psi(\bar{x})[t/x_i])}{\{\phi(\bar{x})\}\ x_i := t\ \{\psi(\bar{x})\}}$$

連接規則

$$\frac{\{\phi_1\}P_1\{\phi'\} \qquad \{\phi'\}P_2\{\phi_2\}}{\{\phi_1\}\ \textbf{begin}\ P_1; P_2\ \textbf{end}\ \{\phi_2\}}$$

条件分岐規則

$$\frac{\{\phi_1 \wedge \psi\}P_1\{\phi_2\} \qquad \{\phi_1 \wedge \neg\psi\}P_2\{\phi_2\}}{\{\phi_1\}\ \textbf{if}\ \psi\ \textbf{then}\ P_1\textbf{else}\ P_2\ \{\phi_2\}}$$

繰り返し規則

$$\frac{\forall((\phi_1 \wedge \neg\psi) \rightarrow \phi_2) \quad \{\phi_1 \wedge \psi\}P_1\{\phi_1\}}{\{\phi_1\}\ \textbf{while}\ \psi\ \textbf{do}\ P_1\ \{\phi_2\}}$$

左帰結規則

$$\frac{\forall(\phi_1 \rightarrow \phi') \quad \{\phi'\}P_1\{\phi_2\}}{\{\phi_1\}P_1\{\phi_2\}}$$

右帰結規則

$$\frac{\{\phi_1\}P_1\{\phi'\} \quad \forall(\phi' \rightarrow \phi_2)}{\{\phi_1\}P_1\{\phi_2\}}$$

各推論規則の直感的な意味は, 以下のようなものである．代入規則の前提は, $\phi(\overline{x}) \rightarrow \psi(\overline{x})[t/x_i]$ という式がいかなる値についても真となることを仮定している．ここでこの式の $\rightarrow$ の右辺を t という項についての条件を表すものと考える．このとき代入文の実行の各変数の値について，左辺が真となれば t という項が表す値がこの右辺を真とすることを意味している．ここで代入文 $x_i := t$ の実行によって x_i の値が t となるので，実行後は t の値について真である条件すなわち $\rightarrow$ の右辺が，x_i について真となる．このことは代入文が入出力条件 ϕ と ψ について部分的正当であることの定義にあてはまる．

連結規則の前提は，各変数の値について ϕ_1 が真であるとき P_1 を実行して正常に停止したらその時点での各変数の値についてある条件 ϕ' が真，及び ϕ' が真となるその状態で引き続き P_2 を実行して正常に停止したら終了時の変数の各値について ϕ_2 が真となることを仮定している．このとき，ϕ_1 を真とする値について **begin** $P_1; P_2$ **end** を実行した場合，前節で述べた連接文の意味よりまず P_1 が実行され終了すればその後ただちに P_2 が実行される．前提より P_1 が終了した時点で ϕ' が真であるので，そのまま P_2 が実行され停止すればやはり前提より ϕ_2 が真となる．すなわち **begin** $P_1; P_2$ **end** の実行が終了すれば ϕ_2 が真となる．したがって **begin** $P_1; P_2$ **end** は入出力条件 ϕ_1, ϕ_2 について部分的正当である．

条件分岐規則の前提 $\{\phi_1 \wedge \psi\}P_1\{\phi_2\}$, $\{\phi_1 \wedge \neg\psi\}P_2\{\phi_2\}$ は，各変数の値が ϕ_1 を真とするとき，ψ が真の場合と偽の場合についてそれぞれ P_1 ないしは P_2 を実行して終了すれば，そのときの変数の値について ϕ_2 が真となることを仮定している．上で述べた条件分岐文の意味より，**if** ψ **then** P_1 **else** P_2 を実行するとは，ψ が真の場合は P_1 を，偽の場合を P_2 を実行することに等しい．したがって ϕ_1 が真となる値のもとで実行を開始すれば，前提よりいずれの場合も終了すればそのときの値のもとで ϕ_2 が真となる．すなわち **if** ψ **then** P_1 **else** P_2 は入出力条件 ϕ_1, ϕ_2 について部分的正当である．

繰り返し規則は，繰り返し文の本体 P_1 の実行される回数に関する数学的帰納法を形式化したものである．この前提のうち $\forall((\phi_1 \wedge \neg\psi) \rightarrow \phi_2)$ は，繰り返し文本体 P_1 の実行前でループを脱出する条件が成り立てば ϕ_2 が真となることを意味する．すなわち ϕ_1 が真となる値について繰り返し文の本体 P_1 を 0 回実行して繰り返し文の実行を終了したときは，ϕ_2 が真となることを示している．またもう一つの前提 $\{\phi_1 \wedge \psi\}P_1\{\phi_1\}$ は，ψ が真である場合は ϕ_1 が真となる値のもとで本体 P_1 を 1 回実行した後に，再度実行される繰り返し文が再び ϕ_1 が真となる値から実行が始まることを意味する．これよりこのあと再度実行される繰り返し文 **while** ψ **do** P_1 が ϕ_1 と ϕ_2 について部分的正当であると仮定したとき，最初に本体 P_1 を 1 回

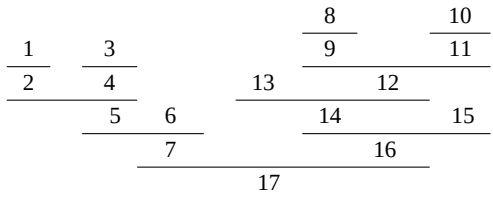
実行した後繰り返し文を再度実行して終了すれば ϕ_2 が真となる．したがって数学的帰納法より，繰り返し文の本体を任意の回数実行した場合について部分的正当性が示される．

左帰結規則は，P_1 が入力条件 ϕ' と出力条件 ϕ_2 について部分的正当でかつすべての値について ϕ_1 ならば ϕ' であるとき，P_1 は入力条件 ϕ_1 と出力条件 ϕ_2 についても部分的正当であることを述べている．すなわち入力条件 ϕ_1 を真とする値について P_1 を実行する場合を考える．このとき前提 $\forall(\phi_1 \to \phi')$ より，ϕ' を真とする値について P_1 を実行する場合とみなしてかまわない．一方もう一つの前提より P_1 が入力条件 ϕ' を真とする値について実行され終了するならば，そのときの変数の値について出力条件 ϕ_2 が真となる．すなわち P_1 は入力条件 ϕ_1 と出力条件 ϕ_2 について部分的正当である．右帰結規則についても同様に理解できる．

(e) 証明図

ホーア記法 $\{\phi\}P\{\psi\}$ の証明図とは，根が $\{\phi\}P\{\psi\}$，葉が領域公理で，各内部節点がその子を前提とするいずれかの推論規則の結論となるホーア記法である二分木である．

例 3 以下は先の例 2 の式 (2･4) のホーア記法の証明図である．各節点の番号は，下に示した式を表す．また各式を導くのに用いられた推論規則を各式の後に示す．



1. $\forall x(x \geq 1 \ \to \ (0 = 0 \wedge x \geq 1))$ ：領域公理
2. $\{x \geq 1\}z := 0\{z = 0 \wedge x \geq 1\}$: 1 より代入規則
3. $\forall z \forall y \forall x((z = 0 \wedge x \geq 1) \to (z = 0 \wedge x \geq 1 \wedge 0 = 0))$: 領域公理
4. $\{z = 0 \wedge x \geq 1\}y := 0\{z = 0 \wedge x \geq 1 \wedge y = 0\}$ ：3 より代入規則
5. $\{x \geq 1\}$ **begin** $z := 0; y := 0$ **end** $\{z = 0 \wedge x \geq 1 \wedge y = 0\}$: 2, 4 より連接規則
6. $\forall x \forall y \forall z(z = 0 \wedge x \geq 1 \wedge y = 0) \to ((y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2}) \wedge z \leq x)$ ：領域公理
7. $\{x \geq 1\}$ **begin** $z := 0; y := 0$ **end** $\{(y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2}) \wedge z \leq x\}$ ：5, 6 より右帰結規則
8. $\forall x \forall y \forall z((y + z + 1 + \frac{(z+1+1+x)(x-z-1)}{2} = \frac{x(x+1)}{2} \wedge z + 1 \leq x) \to (y + z + 1 + \frac{(z+1+1+x)(x-z-1)}{2} = \frac{x(x+1)}{2} \wedge z + 1 \leq x))$: 領域公理
9. $\{y+z+1+\frac{(z+1+1+x)(x-z-1)}{2} = \frac{x(x+1)}{2} \wedge z+1 \leq x\}z := z+1\{y+z+\frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \wedge z \leq x\}$: 8 より代入規則

10. $\forall x \forall y \forall z((y + z + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x) \rightarrow (y + z + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x))$: 領域公理

11. $\{y + z + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x\}\ y := y + z\ \{y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x\}$: 10 より代入規則

12. $\{y+z+1+\frac{(z+1+1+x)(x-z-1)}{2} = \frac{x(x+1)}{2} \land z+1 \leq x\}$ **begin** $z := z+1; y := y+z$ **end** $\{y+\frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x\}$: 9, 11 より連接規則

13. $\forall x \forall y \forall z((y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x \land x \neq z) \rightarrow (y + z + 1 + \frac{(z+1+1+x)(x-z-1)}{2} = \frac{x(x+1)}{2} \land z + 1 \leq x))$: 領域公理

14. $\{y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x \land x \neq z\}$ **begin** $z := z + 1; y := y + z$ **end** $\{y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x\}$ ：12, 13 より左帰結規則

15. $\forall x \forall y \forall z((y + \frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x \land \neg(x \neq z)) \rightarrow y = \frac{x(x+1)}{2})$: 領域公理

16. $\{y+\frac{(z+1+x)(x-z)}{2} = \frac{x(x+1)}{2} \land z \leq x\}$ **while** $z \neq x$ **do begin** $z := z+1; y := y+z$ **end**$\{y = \frac{x(x+1)}{2}\}$: 14, 15 より繰り返し規則

17. $\{x \geq 1\}$
    begin
    $z := 0;\ y := 0;$
    while $z \neq x$ **do begin** $z := z + 1; y := y + z$ **end**
    end
    $\{y = \frac{x(x+1)}{2}\}$
    : 7, 16 より連接規則

2-1-3 完全性と健全性

ホーア論理は証明論的な公理系であり，健全性及び完全性は重要な関心事である．これらのうち健全性については，比較的容易な議論で示すことができるが，完全性については注意を要する点がある．ここではその概略について述べる．

(1) ホーア記法の真偽

数理論理学における公理系の健全性と完全性は，その体系で扱う論理式の真偽の定義を与えた上で議論される．同様にホーア論理の健全性及び完全性は，論理式にあたるものであるホーア記法の真偽を定義することにより議論が始まる．ホーア記法の与えられた解釈のもとでの真偽を定めるためには，その解釈のもとでの繰り返しプログラムの意味論を形式的に定義する必要がある．ここでは，繰り返しプログラムは決定性のプログラムであることより，プログラムの意味は変数割当から変数割当への関数として定義する．プログラムの文 P の意味は，変数割当の集合を定義域及び値域とする以下のような関数 T_P である．

(a) プログラムの意味

- $T_{x:=t}(s) = s[x|t]$
- $T_{P_1;P_2}(s) = T_{P_2}(T_{P_1}(s))$
- $T_{\textbf{if } \psi \textbf{ then } P_1 \textbf{ else } P_2}(s) = \begin{cases} T_{P_1}(s) & \text{if } s \models_I \psi \\ T_{P_2}(s) & \text{otherwize} \end{cases}$
- $T_{\textbf{while } \psi \textbf{ do } P_1}(s) = \begin{cases} s & \text{if } s \not\models_I \psi \\ T_{\textbf{while } \psi \textbf{ do } P_1}(T_{P_1}(s)) & \text{otherwize} \end{cases}$

ただしここで $s[x|t]$ は以下のような変数割当である．

$$s[x|t](y) = \begin{cases} s_I[t] & \text{if } y = x \\ s(y) & \text{otherwize} \end{cases} \tag{2･5}$$

ここで $s_I[t]$ は、解釈 I と変数割当 s のもとで項 t が表す D の元である．また $s \models_I \psi$ は，解釈 I と変数割当 s のもとで ψ が真であることを意味する．また $s \not\models_I \psi$ は I と変数割当 s のもとで ψ が偽であることを意味する．このように定義された関数 T_P は，もし s から実行を開始したとき P が停止しなければ，$T_P(s)$ の値は存在せず未定義となり，したがって一般には T_P は部分関数となることに注意されたい．このように定義された意味のもとでホーア記法の真偽は以下のように形式的に定義される．ここで S はプログラムの変数割当すべての集合である．

(b) ホーア記法の形式的意味

$$I \models \{\phi_1\}P\{\phi_2\} \quad \text{iff} \quad \forall s \in S, s \models_I \phi_1 \rightarrow T_P(s) \models_I \phi_2 \tag{2･6}$$

(2) 健全性

ホーア論理の公理系の健全性は，以下のように定義される．

> A_I を解釈 I で真となる閉論理式すべての集合とする．任意の ϕ_1, ϕ_2，任意の P について，$\{\phi_1\}P\{\phi_2\}$ の A_I を領域公理の集合とする証明図が存在するならば，$I \models \{\phi_1\}P\{\phi_2\}$.

この結果は証明図の高さに関する数学的帰納法により容易に示される．

(3) 完全性

逆に完全性は，$I \models \{\phi_1\}P\{\phi_2\}$ ならば $\{\Phi_1\}P\{\phi_2\}$ の証明図が存在することを主張するものとなる．ここで問題となるのは，例えば連接規則の前提に出現する ϕ' である．ここで ϕ' は結論の側には出現しない式となっている．したがって $\{\phi_1\}P\{\phi_2\}$ がゴールとして与えられこの証明図を構成しようとする場合，P が連接 $P_1; P_2$ であった場合，適切な ϕ' を発見することが必要となる．特に例えば P が繰り返し文であったとき，ϕ_1, ϕ_2 に対して繰り返し規則より $(\phi \wedge \neg\psi) \rightarrow \phi_2$ かつ $\{\phi \wedge \psi\}P_1\{\phi\}$ を（及び左の帰結規則 より $\phi_1 \rightarrow \phi$ を）満足する ϕ が発見されなければならない．繰り返しプログラムの場合，このような条件を見つけることは自明ではない．例えば例 2, 例 3 のようにプログラム自身には足し算と定数のみ出現する場合で

も，入出力条件にはかけ算が出現している．このように証明図に出現する論理式を記述する言語は十分に強い表現力をもつことが求められる．この結果，領域公理に決定不能であるような論理式が出現することもありうる．したがってホーア論理では通常は，証明図中に出現する式で一階述語論理式で記述された式については，それ以上証明を求めることはなく，それらの式は真であれば領域公理と認める．すなわちホーア論理の公理系の完全性とは，ホーア記法を解釈の上での真な一階述語論理式に帰着することができることを主張するに留まる．以上のことより，完全性は以下のように定義することができる．

> A_I を解釈 I で真である閉論理式すべての集合とする．任意の ϕ_1, ϕ_2，任意の P について，$I \models \{\phi_1\}P\{\phi_2\}$ ならば $\{\Phi_1\}P\{\phi_2\}$ の A_I を領域公理とした証明図が存在する．

ここで領域公理の集合として解釈 I で真であるすべての閉論理式の集合を仮定していることに注意されたい．クック（Cook）らは，このような意味でホーア論理の完全性は領域公理の集合に依存した概念であるとして，相対的完全性（relative completeness）と呼んでいる[7]．クックは適切な領域公理の集合を選ぶことにより，ホーア論理の公理系を相対的完全なものとすることができることを示した．

2-1-4 ダイクストラの最弱前条件とホーア論理

ダイクストラ（Dijkstra）はホーア記法 $\{\phi\}P\{\psi\}$ の最初の ϕ を取り除いた $P\{\psi\}$ の部分に着目し，この部分を $wp(P,\psi)$ と表記した．これは P と ψ から定まるある述語で「P を実行して停止したら ψ が成立するために，入力値について成り立つべき最も弱い条件」を表すもので，最弱前条件（weakest precondition）と呼んだ[8]．すなわち wp は，プログラム P から定まる述語から述語への関数 $\lambda\phi.wp(P,\phi)$ と考えられる．この記法を用いるとホーア記法 $\{\phi\}P\{\psi\}$ は，$\phi \rightarrow wp(P,\psi)$ と表される．

フロン（Flon）と鈴木は，このような最弱前条件 $wp(P,\phi)$ は P と ϕ から定まる述語上の関数 Φ_P^ϕ の最大不動点として表されることに着目した[9]．すなわちプログラム P をある単純な構文（ただし本質的には本章 2-1, 式 (2･2) で導入した繰り返しプログラムと同等の表現力をもつ）で表したとき，領域公理を記述する言語が Φ_P^ϕ の最大不動点を表現できる十分な表現力を持つならば，以下のような推論規則をもつ公理系は前節の意味で相対的完全かつ健全であることを示した．

$$\frac{\psi \rightarrow \Phi_P^\phi(\psi)}{\psi \rightarrow wp(P,\phi)} \qquad (2\cdot7)$$

この結果より，具体的なプログラムの構文が与えられたとき，その部分的正当性を証明する健全かつ相対的完全な公理系は，本質的にはそのプログラムの構文をダイクストラの記法に変換する手法から構築可能であるといえる．

同様に部分的正当性以外のプログラムの性質についても，プログラムから定まる述語から述語への関数を用いて記述できる（このことは，次節の様相論理との関連の項目で述べる）．文献 9) では，部分的正当性以外の停止性（後述）などの性質について最大不動点または最小

不動点で表現できることを用いて，そのような性質の証明のための健全かつ相対的完全な公理系の構築方法を示した．

2-1-5 様相論理とホーア論理

ここでは今日の形式的プログラム検証技術の中心的な手法の基礎となっている様相論理とホーア論理との関連について述べる．様相論理をプログラムの正当性の証明に応用する研究の初期の成果は 文献 10, 11) にさかのぼる．基本的にはホーア論理で扱われる部分的正当性や停止性などのプログラムの性質は，様相論理によって記述されるプログラムの性質の一例と考えられる．

様相論理とは，通常の命題論理または一階述語論理に，様相演算子と呼ばれる新たな論理記号をいくつか追加することにより拡張した論理である．様相論理において用いられる基本的な様相演算子の例として代表的なものは　□, ◇ の 2 種類があり，その直感的な意味は以下のようなものである．

- □P: P がこれからいつでも常に真である．
- ◇P: P が真となるときが存在する．

プログラムの検証に用いられる場合は，多くの場合この 2 種類に加えて以下のような演算子も用いる．

- ○P: P が現在の次の時点で真である．

様相論理の論理式は，状態の集合の上の二項関係に基づいて真偽を定めることができる．このことを利用して，本章 2-1-3(1) (a) で変数割当の上の関数として定義したプログラムの意味を状態の集合の上の二項関係と見なすことで，与えられたプログラムの上での様相論理式の真偽が定まる．

上の □ 及び ◇ の演算子を用いると，P が本節で扱ったような決定性の逐次プログラムの場合，部分的正当性 $\{\phi\}\ P\ \{\psi\}$ は以下のように表すことができる．

$$\phi \rightarrow \Box(\neg P\downarrow \vee \psi) \tag{2･8}$$

ここで $P\downarrow$ は, P の実行が終了したときに真となる述語で，例えば P の実行前にある変数 v の値を 0 として，実行の最後に v の値を 1 とする操作を記述しておけば, $P\downarrow \equiv v = 1$ とできる．

このほかにもプログラムの様々な性質が様相論理の論理式を用いて表現できることが知られており，またそのような表現に用いられる様相演算子は多くがある種の不動点で特徴づけられるものであることも知られている[12)]．このことから先に触れた不動点を用いて記述した性質に対して健全かつ相対的完全な推論規則を構築する手法の有効性を理解することができる．

様相論理を用いたプログラム検証の手法では，このような論理式を真とするモデルが検証対象のプログラムから構成されることを示すことにより，プログラムが望まれる性質をもつことを示す手法がよく用いられる〔詳しくは本章 2-4 参照〕．その意味においてそういった手

法はモデル論的な手法であり，ホーア論理のように推論規則を適用して証明を行う証明論的手法とは異なる．

一方，様相論理の一種であるダイナミック論理[13)]では，プログラムの構文を様相演算子の中に含めることによりホーア記法に近い記述をする手法を用いている．このようなダイナミック論理では推論規則を与えて証明のための公理系を構成する手法がとられており，こちらはホーア論理の直接的な拡張となっている．

■参考文献

1) C.A. R. Hoare, “Axiomatic Basis of Computer Programming,” Commun. ACM., vol.12, no.10, pp.576-580, 583, 1969.
2) R.W. Floyd, “Assigning Meanings to Programs,” Proc. Symp. Appl. Math., vol.19, pp.19-32, 1967.
3) D. Kozen, and J. Tiuryn, “Logic of Programs,” Hadbook of Theoretical Computer Science, Vol. B, Formal Models and Semantics, The MIT Press/Elsevier, pp.789-840, 1990.
4) R.B. Anderson, Proving Programs Correct, John Wiley & Sons, 1979; 邦訳: 演習 プログラムの証明, 有澤訳, 近代科学社, 1980.
5) P. Cousot, “Method and Logic for Proving Program,” Hadbook of Theoretical Computer Science, Vol. B, Formal Models and Semantics, The MIT Press/Elsevier, pp.841-993, 1990.
6) Z. Manna, Mathematical Theory of Computation, McGrowhill, 1974.
7) S.A. Cook, “Soundness and Completeness of Axiom System for Program Verification,” SIAM J. on Comput., vol.7, no.1, pp.70-90, 1978.
8) E.W. Dijkstra, “Guraded Commands, Nondeterminacy and Formal Derivation of Programs,” Commun. ACM, vol.19, no.8, pp.453-457, 1975.
9) L. Flon and N. Suzuki, “The Total Correctness of Parallel Programs,” SIAM J. Comput., vol.10, no.2, pp.227-246, 1981.
10) Z. Manna, “Logic of Programs,” Proc. of IFIP Congress ’80, pp.41-51, 1980.
11) A. Pnueli, “The Temporal Logic of Programs,” Proc. of the 18th IEEE Symp. of FOCS., pp.46-57, 1977.
12) J. Bradfield and C. Stirling, “Modal Logics and mu-Calculi: An Introduction,” Handbook of Process Algebra, Elsevier Science, pp.293-330, 2001.
13) D. Harel, “First-Order Dynamic Logic,” LNCS, no.68, 1980.

■7 群 - 1 編 - 2 章

2-2 Isabelle/HOL

（執筆者：南出靖彦）[2008 年 8 月 受領]

Isabelle/HOL は，ポールソン（Paulson）とニプコウ（Nipkow）によって開発された証明支援系である[1)]．論理体系を形式化するためのメタ論理である Isabelle[2)]とその上で形式化された高階論理 HOL とから構成されている．Isabelle 自体は特定の論理には特化しておらず，Isabelle 上で論理体系を形式化することで，様々な論理体系の証明支援系が構築できる．標準のシステムでも， HOL, LCF, ZF などの多くの論理体系に対応おり，そのなかで最も広く使われているものが，高階論理を形式化した Isabelle/HOL である．Isabelle/HOL は，高階論理に基づく証明支援系 HOL[3)] に大きく影響を受けている．本節の説明は，Isabelle2008 に基づいている．

Isabelle では，論理体系を形式化するために用いる Isabelle 自体の論理をメタ論理と呼び，その上で形式化する論理体系をオブジェクト論理と呼ぶ．Isabelle/HOL では，高階論理 HOL がオブジェクト論理となっている．

2-2-1 Isabelle

Isabelle は，論理体系を形式化するためのメタ論理として直観主義高階論理を用いている．この高階論理は，次節で解説するチャーチ（Church）の高階論理を制限したものと考えることができる．Isabelle の論理は，型付きラムダ計算に基づいており， 型と式は次のように定義される．

$$
\begin{array}{lrcl}
\text{（単相型）} & \tau & ::= & \alpha \mid \mathsf{prop} \mid \tau \Rightarrow \tau \\
\text{（多相型）} & \sigma & ::= & \tau \mid \forall \alpha.\sigma \\
\text{（式）} & t & ::= & c \mid x \mid \lambda x.t \mid tt
\end{array}
$$

Isabelle の型体系は ML スタイルの多相型をもつが，定数のみが多相型をもち，明示的な let-式は構文に含んでいない．また，本稿では説明を省くが，Haskell スタイルの型クラスの機能ももっている．単相型は，型変数 α，基本型 prop，関数型 $\tau_1 \Rightarrow \tau_2$ から構成されている．基本型 prop は論理式の型である．オブジェクト論理を形式化する場合には必要となる基本型を必要に応じ追加する．

定数 c としては下の論理記号の定数を含んでおり，含意 $\Longrightarrow$， 等号 $\equiv$ には中置記法を用いる．

$$
\begin{array}{ll}
\Longrightarrow :: \mathsf{prop} \Rightarrow \mathsf{prop} \Rightarrow \mathsf{prop} & \bigwedge :: \forall \alpha.(\alpha \Rightarrow \mathsf{prop}) \Rightarrow \mathsf{prop} \\
\equiv :: \forall \alpha.\alpha \Rightarrow \alpha \Rightarrow \mathsf{prop} &
\end{array}
$$

全称命題は定数 $\bigwedge$ を用いて表現され，$\bigwedge(\lambda x.t)$ はすべての x に対して t が成り立つことを意味している．$\bigwedge(\lambda x.t)$ を $\bigwedge x.t$ と書く．また，Isabelle では，型 $\tau_1 \Rightarrow \tau_2 \Rightarrow \cdots \Rightarrow \tau_n \Rightarrow \tau$ を $[\tau_1, \tau_2, \ldots, \tau_n] \Rightarrow \tau$ と書き，論理式 $\phi_1 \Longrightarrow \phi_2 \Longrightarrow \cdots \Longrightarrow \phi_n \Longrightarrow \phi$ を $[\![\phi_1; \phi_2; \cdots \phi_n]\!] \Longrightarrow \phi$ と書く．

Isabelle は，論理体系を形式化するために用いられる．高階論理 HOL の形式化は複雑にな

るので，本稿では 1 階述語論理の自然演繹を Isabelle の論理上に形式化する．説明を簡略化するため，ここでは次のかたちの論理式のみを考える．

$$\begin{array}{lrcl} \text{（オブジェクト論理の項）} & s & ::= & c \mid f(s_1, \ldots, s_n) \\ \text{（オブジェクト論理の論理式）} & P & ::= & p(s_1, \ldots, s_n) \mid P \longrightarrow Q \mid \forall x.P \end{array}$$

自然演繹の推論規則としては，次の四つの推論規則を考える．

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \longrightarrow Q} \qquad \frac{\Gamma \vdash P \longrightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \forall x.P} \ (x \notin FV(\Gamma)) \qquad \frac{\Gamma \vdash \forall x.P(x)}{\Gamma \vdash P(t)}$$

この論理体系を，Isabelle の論理上に形式化する最初のステップは，項と論理式を埋め込むことである．そのために，オブジェクト論理の項と論理式を表す型，ind と bool をメタ論理に導入する．次に，オブジェクト論理の定数 c，関数記号 f，述語記号 p に対応する定数を次のようにメタ論理に導入する．

- 定数 c に対して ind 型の定数 c を導入し，n 引数の関数記号 f に対して $\overbrace{\mathsf{ind} \Rightarrow \ldots \Rightarrow \mathsf{ind}}^{n} \Rightarrow$ ind 型の定数 f を導入する．
- n 引数の述語記号 p に対して $\overbrace{\mathsf{ind} \Rightarrow \ldots \Rightarrow \mathsf{ind}}^{n} \Rightarrow$ bool 型の定数 p を導入する．

オブジェクト論理の論理演算子を埋め込むために，更に下の定数を導入する．

$$\mathsf{imp} :: \mathsf{bool} \Rightarrow \mathsf{bool} \Rightarrow \mathsf{bool} \qquad \mathsf{forall} :: (\mathsf{ind} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{bool}$$

これにより，例えば，オブジェクト論理の論理式 $\forall x.p(x, f(x))$ は，メタ論理の bool 型の式 $\mathsf{forall}(\lambda x.px(fx))$ に埋め込むことができる．更に，$\mathsf{bool} \Rightarrow \mathsf{prop}$ 型の定数 nd を導入し，$\mathsf{nd}(P)$ で P がオブジェクト論理で証明可能なことを表す．

このとき，オブジェクト論理の推論規則は，以下のようにメタ論理の論理式として表現できる．これらをメタ論理の公理として追加することで，自然演繹の証明をメタ論理の証明に埋め込むことができる．

$$\begin{array}{cc} (\mathsf{nd}(\phi) \implies \mathsf{nd}(\psi)) \implies \mathsf{nd}(\mathsf{imp}\,\phi\,\psi) & \mathsf{nd}(\mathsf{imp}\,\phi\,\psi) \implies \mathsf{nd}(\phi) \implies \mathsf{nd}(\psi) \\ (\bigwedge x.\mathsf{nd}(\phi)) \implies \mathsf{nd}(\mathsf{forall}(\lambda x.\phi)) & \mathsf{nd}(\mathsf{forall}(\lambda x.\phi)) \implies \mathsf{nd}(\phi[t/x]) \end{array}$$

ここまでに示した方法で実際に Isabelle で自然演繹を形式化したスクリプトを下に示す．スクリプトで導入している型 *o* が bool 対応しており，定数 *Trueprop* が，nd に対応している．下のスクリプトは，Isabelle に含まれている 1 階述語論理の形式化（FOL）を単純化したものである．*Trueprop* に関する記法の指定 *("(_)" 5)* より *Trueprop* が省略できるようになっているので，公理のなかで *Trueprop* は現れない．

theory *IFOL* **imports** *Pure* **begin**

typedecl *o*

```
judgment
  Trueprop      :: "o ⇒ prop" ("(_)" 5)

consts
  imp           :: "[o, o] ⇒ o"     (infixr "⟶" 25)
  All           :: "('a ⇒ o) ⇒ o"

axioms
  impI:         "(P ⟹ Q) ⟹ (P⟶Q)"
  mp:           "⟦ P⟶Q;  P ⟧ ⟹ Q"
  allI:         "(⋀x. P(x)) ⟹ (All (λx. P(x)))"
  spec:         "(All (λx. P(x))) ⟹ P(x)"
end
```

2-2-2 高階論理 HOL

Isabelle における高階論理 HOL は，チャーチによる型理論（simple theory of types）[4]に基づいている．メタ論理とは異なり，排中律が成り立つ古典論理の体系である．本節では，定理証明系 HOL[3]における形式化に基づき，高階論理 HOL について解説する．HOL は，Isabelle のメタ論理と同様に型付きラムダ計算に基づいており，型と式は以下のように定義される．

（単相型） τ ::= α | bool | ind | $\tau \Rightarrow \tau$
（多相型） σ ::= τ | $\forall\alpha.\sigma$
（式） t ::= c | x | $\lambda x.t$ | tt

型体系は，前節で示した Isabelle の型体系と同一の体系である．チャーチの元来の高階論理は多相型を含んでいなかったが，定理証明系 HOL おいて多相型が導入され，Isabelle/HOL にも引き継がれている．基本型としては，論理式の型である bool と個体を表すための型 ind をもつ．型 ind は公理で無限個の要素をもつことが保証され，自然数などを表現する基礎となる．本稿では，式のなかで bool 型をもつものをメタ変数 ϕ, ψ で表す．

定数 c としては，含意 $\longrightarrow$, 等号 =, 限定的な記述演算子（description operator）ι を含んでいる．$\longrightarrow$ と = には，中置記法を用いる．$\iota(\lambda x.t)$ は，t を満たす唯一の値を表しており，$\iota x.t$ と書く．$\lambda x.t$ を満たす値がない場合や複数ある場合は，どのような値を示すかは定義されていない．

$\longrightarrow$ :: bool $\Rightarrow$ bool $\Rightarrow$ bool　　= :: $\forall\alpha.\alpha \Rightarrow \alpha \Rightarrow$ bool
ι :: $\forall\alpha.(\alpha \Rightarrow$ bool$) \Rightarrow \alpha$

論理式の集合 Γ 対して，Γ から ϕ が証明できることを $\Gamma \vdash \phi$ で表す．HOL は以下の推論規則をもつ．

$$\frac{\phi \in \Gamma}{\Gamma \vdash \phi} \qquad \frac{}{\Gamma \vdash t = t} \qquad \frac{}{\Gamma \vdash (\lambda x.t_1)t_2 = t_1[t_2/x]}$$

$$\frac{\Gamma \vdash t = t' \quad \Gamma \vdash \phi[t/x]}{\Gamma \vdash \phi[t'/x]} \qquad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \longrightarrow \psi} \qquad \frac{\Gamma \vdash \phi \longrightarrow \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash \lambda x.t_1 = \lambda x.t_2} \; x \notin FV(\Gamma)$$

ほかの論理演算子は，上の二つの演算子を用いて定義されている．以下に代表的な論理演算子の定義を示す．

$$\begin{array}{lcl} \top & \equiv & \lambda x.x = \lambda x.x \qquad \bot \equiv \lambda\phi.\phi \qquad \forall\phi \equiv \phi = \lambda x.\top \\ \phi_1 \wedge \phi_2 & \equiv & \forall\psi.(\phi_1 \longrightarrow \phi_2 \longrightarrow \psi) \longrightarrow \psi \\ \phi_1 \vee \phi_2 & \equiv & \forall\psi.(\phi_1 \longrightarrow \psi) \longrightarrow (\phi_2 \longrightarrow \psi) \longrightarrow \psi \end{array}$$

HOL は以下の公理をもつ．1 番目の公理により，同値な命題が bool 上で等しい値となる．2 番目，3 番目の公理は，排中律とラムダ計算の η 規則に対応している．ラムダ計算と同様に，η 規則から関数の外延性が導ける．

$$\begin{array}{l} \forall\phi.\forall\psi.(\phi \longrightarrow \psi) \longrightarrow (\psi \longrightarrow \phi) \longrightarrow (\phi = \psi) \\ \forall\phi.(\phi = \top) \vee (\phi = \bot) \\ \forall f.(\lambda x.fx) = f \\ \forall t.(\iota x.x = t) = t \end{array}$$

記述演算子に関する公理の意味は分かりにくいが，$\forall P.(\exists! x.Px) \longrightarrow P(\iota x.Px)$ と同値になる．古いバージョンの Isabelle/HOL（Isabelle99-2 まで）では，ヒルベルト（Hilbert）による ϵ-演算子が論理体系の基礎となる演算として採用されていたが，現在は，ϵ-演算子は高階論理 HOL を拡張するセオリーのなかで公理として提供されている．ϵ-演算子は，下の公理で特徴づけられ，ϵ-演算子を導入することによって選択公理が証明できるようになる．

$$\forall P.(\exists x.Px) \longrightarrow P(\epsilon x.Px)$$

型 ind に対しては，自然数を表現する基礎となる以下の二つの定数とそれらに関する下の公理をもつ．

zero_rep :: ind　　suc_rep :: ind ⇒ ind

inj suc_rep　　$\forall t.\neg$(zero_rep = suc_rep t)

関数 inj は，関数が単射であることを表す述語で inj $f \equiv \forall x.\forall y.f(x) = f(y) \longrightarrow x = y$ と定義されている．上の二つの公理から，型 ind が無限個の要素をもつことを証明できる．

2-2-3 Isabelle/HOL

Isabelle/HOL は，Isabelle で前節の高階論理 HOL を形式化したものである．Isabelle/HOL では，高階論理を基礎として，関数型言語に見られる代数的データ型や原始帰納関数の定義が可能になっている．データ型や原始帰納関数の定義の機能は，高階論理 HOL の枠組みのなかで構築されており，論理の点でいえば新しい機能を導入しているわけではない．

Isabelle において証明を記述する方法としては，各ステップで証明に使うタクティック（tactic）のみを記述するスタイルと人が書く証明に近い証明記述言語 Isar[5]を用いるスタイルがある．本稿では，証明の記述には Isar を用いて Isabelle における証明法を解説する．

(1) 論　理

高階論理に関する基本的な証明のステップは，証明すべきゴールとメタ論理で表された推論規則に対する導出原理の適用である．メタ論理で表されたオブジェクト論理の推論規則は，下の形のメタ論理の定理となっている．

$$[\![P1;\ P2;\ \cdots\ Pn]\!] \Longrightarrow Q$$

具体例を示すと，含意に関する導入規則と除去規則は下のかたちをしている．

```
impI: "(?P ⟹ ?Q) ⟹ ?P ⟶ ?Q"
mp:  "⟦?P ⟶ ?Q; ?P⟧ ⟹ ?Q"
```

それぞれの公理には，*impI* と *mp* という名前が付いている．これらの公理に現れる，*?P* など，記号*?*が付いた変数はスキーマ変数（schematic variable）と呼ばれ，その変数が単一化のときに，代入可能な変数として扱われることを表している．これらの論理式は，Isabelle に公理として追加したメタ論理の論理式に現れる自由変数をスキーマ変数に置き換えたものになっている〔本章 2-2-2 参照〕．

Isabelle/HOL における基本的な証明のステップは，証明すべきゴール（論理式）に対して，ユーザが証明法を指定・適用し，新たなゴール（一般には複数個）を得ることである．メタ論理の論理式 $[\![P1;\ P2;\ \cdots\ Pn]\!] \Longrightarrow Q$ を用いた導出では，ゴール G と Q が単一化 θ をもつとき，新たなゴールとして $P1\theta$, $P2\theta$, . . ., $Pn\theta$ が生成される．この導出には，高階単一化が用いられる．

下の証明は，証明記述言語 Isar を用いて $P \longrightarrow (P \longrightarrow Q) \longrightarrow Q$ を証明したものである．

```
lemma "P ⟶ (P ⟶ Q) ⟶ Q"
proof (rule impI)
  assume P
  show "(P ⟶ Q) ⟶ Q"
  proof (rule impI)
    assume "P ⟶ Q"
    show "Q"
    proof (rule mp)
      show "P ⟶ Q" by assumption
      show "P" by assumption
    qed
  qed
qed
```

2 行目の *rule impI* は，定理 impI を用いて導出を行うことを指示している．その結果，新しいゴールとしては，$P \Longrightarrow (P \longrightarrow Q) \longrightarrow Q$ が得られ，3 行目以降では，P を仮定し $(P \longrightarrow Q) \longrightarrow Q$ を証明している．

このように高階論理の推論規則よる証明は，メタ論理の定理として表現されている推論規則とゴールに導出原理を適用し行う．Isar による証明の書き方は，ニプコウによるチュートリアル[6]に丁寧に解説されているのでそちらを参考にされたい．

上の証明では，Isabelle/HOL における証明の原理を説明するために，各ステップで基本的な証明法（ここでは導出原理）を適用したが，簡単な論理式の証明は自動定理証明法（例えば，下の例の *auto*）を用いて証明することもできる．

```
lemma "P ⟶ (P ⟶ Q) ⟶ Q"
  by auto
```

(2) データ型と関数定義

Isabelle/HOL では，代数的なデータ型の定義ができ，自然数，リストなどのデータ型が利用できる．本節では，リストとリスト上の関数を例として，データ型，原始帰納関数の定義，帰納法による証明について解説する．一般には，Isabelle/HOL では，原始帰納関数でなくても停止性が証明できる関数であれば定義可能である．そのような一般の関数定義については，クラウス（Krauss）による解説[7]を参照されたい．

下の例は，Isabelle/HOL 上で多相的なリスト型を定義している．

```
datatype 'a list =
    Nil                ("[]")
  | Cons 'a "'a list"  (infixr "#" 65)
```

定義自体は，ML などの関数型言語の代数的なデータ型の定義と同様であり，この定義により，*list* が型構成子として導入され，*Nil* と *Cons* が定数として導入される．更に，Isabelle/HOL では，下に示すリストの構造に関する帰納法の定理が同時に導入される．

```
⟦?P []; ⋀a list. ?P list ⟹ ?P (a # list)⟧ ⟹ ?P ?list
```

自然数やリスト上の再帰的な関数のうち，Isabelle/HOL では必ず停止するもののみを定義することができる．再帰関数を定義する基本的な方法は，キーワード **primrec** を用いて，原始帰納的な関数を定義する方法である．下の定義では，リストを連結する関数 *append* をリストに関する原始帰納関数として定義している．

```
primrec
  append :: "['a list, 'a list] ⇒ 'a list" (infixr "@" 65)
where
  "[]@ ys = ys"
| "(x#xs)@ys = x#(xs@ys)"
```

このような原始帰納関数の定義を行うと，関数を定義する等式が書換え規則として導入される．

下の証明は，リストの構造に関する帰納法を用いた証明例である．証明法として *induct xs* を指定することでリストに関する帰納法の定理を適用している．帰納法を適用することで，

二つのゴールが生成され，それぞれを書換え規則による単純化（*simp*）を行うことで証明している．帰納的な場合に用いている証明法 *simp!*は，その時点で仮定している命題も書換え規則として用いる．この場合では，帰納法の仮定を書換え規則として用いている．

```
lemma "xs @ [] = xs"
proof (induct xs)
  show "[] @ [] = []"
    by simp
next
  fix x xs
  assume "xs @ [] = xs"
  show "(x # xs) @ [] = x # xs"
    by (simp!)
qed
```

上の証明例では，非常に簡単な定理を人手で証明しているが，上の定理の場合には，帰納法を適用した後は，論理に関する証明の場合と同様に証明法 *auto* を用いれば自動証明できる．

```
lemma "xs @ [] = xs"
  by (induct xs, auto)
```

(3) 帰納的に定義される集合

Isabelle/HOL では，帰納的に定義される集合（関係）やデータ型の定義を最小不動点定理に基づいて行っている[8]．以下のように定義される lfp が，単調な関数に対して最小不動点演算子であることが Isabelle/HOL 上で証明されている．

$$\mathsf{lfp}\ h \equiv \cap\{X \mid h(X) \subseteq X\}$$

Isabelle/HOL では，帰納的に定義される集合（関係）を定義する構文が用意されており，内部で上の最小不動点演算子を適用し集合（関係）が定義される．*n* 番目のフィボナッチ数が *m* であることを表す関係 *Fib n m* を定義した例を示す．

```
inductive
  Fib :: "[nat, nat] ⇒ bool"
where
  "Fib 0 1"
| "Fib 1 1"
| "⟦Fib n x; Fib (n+1) y⟧ ⟹ Fib (n+2) (x+y)"
```

このような定義を行うと内部で，関係を生成する関数が単調であることが証明され，最小不動点演算子を用いて関係が定義される．導入規則や帰納法の定理も導入される．

記述演算子 ι を用いると関数を表す関係から関数を下のように定義できる．

```
definition
  fib :: "nat ⇒ nat"
where
```

```
"fib n ≡ THE x. Fib n x"
```

ここで，*THE x. P* は，$\iota x.P$ に対応する Isabelle/HOL の構文である．この定義から通常のフィボナッチ関数を定義する等式を導くためには，*Fib* が関数を表す関係になっていることの証明が必要で実際に行うとかなりの手間となる．

primrec 構文を用いた関数定義の基礎となる原始帰納法のためのコンビネータの定義でも，上の例のように関数を表す関係を構成し，記述演算子を用いて関数が定義されている．コンビネータの書換え規則を導くための証明が自動化されており，ユーザが行う必要はない．原始帰納関数の定義は，このように導入されるコンビネータに基づき行われている．

代数的なデータ型の定義も，帰納的に定義される集合を通じて行われている．高階論理 HOL の基本型 *ind* から自然数の型 *nat* は，以下のように構成されている．型*ind*の要素のうち，自然数に対応する要素であることを表す述語 *Nat* は，下のように帰納的に定義される集合として定義できる．

```
inductive
  Nat :: "ind ⇒ bool"
where
  "Nat Zero_rep"
| "Nat n ⟹ Nat (Suc_rep n)"
```

Isabelle/HOL は空でない集合から新しい型を定義する機能（証明支援系 HOL で導入された）をもっており，*Nat* を用いて下のように自然数の型 *nat* が定義されている．

```
typedef nat = "{n. Nat n}"
```

代数的データ型を定義する場合も，内部では上のような方法で新しいデータ型を導入している．

■参考文献

1) T. Nipkow, L.C. Paulson, and M. Wenzel, “Isabelle/HOL,” LNCS 2283, Springer, 2002.
2) L.C. Paulson, “Isabelle: A Generic Theorem Prover,” LNCS 828, Springer, 1994.
3) “Introduction to HOL: A theorem proving environment for higher order logic,” ed. by M.J.C. Gordon and T.F. Melham, Cambridge University Press, 1993.
4) A. Church, “A formulation of the simple theory of types,” J. Symbolic Logic, vol.5, no.2, pp.56-68, 1940.
5) M. Wenzel, “Isar – a generic interpretative approach to readable formal proof documents,” in Proc. of the 12th International Conference on Theorem Proving in Higher Order Logics, LNCS 1690, pp.167-184, 1999.
6) T. Nipkow, “A Tutorial Introduction to Structured Isar Proofs,” Distributed with Isabelle.
7) A. Krauss, “Defining Recursive Functions in Isabelle/HOL,” Distributed with Isabelle.
8) S. Berghofer and M. Wenzel, “Inductive datatypes in HOL – lessons learned in formal-logic engineering,” in Proc. of the 12th International Conference on Theorem Proving in Higher Order Logics, LNCS 1690, pp.19-36, 1999.

■7 群 - 1 編 - 2 章

2-3 SAT

（執筆者：廣川　直）[2008 年 8 月 受領]

連言標準形の命題論理式に対して，その充足性可能性を問う決定問題は SAT と呼ばれる．SAT ソルバは SAT を解くツールであり，著名な SAT ソルバは非常に高速に問題を解くことができる．SAT ソルバはしばしば高速な探索が必要とされる際に用いられ，有界モデル検査，電子設計自動化（Electronic Design Automation），制約解消系，定理証明系などの分野で採用され成功を収めている．

2-3-1 SAT とは

本稿では命題変数，論理和 $\vee$，論理積 $\wedge$，否定 $\neg$ からなる命題論理式を扱う．簡便のため，含意 $\rightarrow$，同値 $\leftrightarrow$，排他的論理和 $\oplus$ は次の略記

$$\phi \rightarrow \psi = \neg\phi \vee \psi \quad \phi \leftrightarrow \psi = (\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi) \quad \phi \oplus \psi = (\phi \vee \psi) \wedge (\neg\phi \vee \neg\psi)$$

とし，また真 $\top$ は $x \vee \neg x$（ただし x はほかで用いない変数），偽 $\bot$ は $\neg\top$ の略記とする．

定義 1 命題変数から真理値 $\{T, F\}$ への関数を真理値割り当てという．真理値割り当て α と論理式 ϕ に対し，充足可能性 $\vDash$ は次のように定義される．

$$\alpha \vDash x \iff \alpha(x) = T \qquad \alpha \vDash \phi_1 \wedge \phi_2 \iff \alpha \vDash \phi_1 \text{ かつ } \alpha \vDash \phi_2$$
$$\alpha \vDash \neg\phi \iff \alpha \nvDash \phi \qquad \alpha \vDash \phi_1 \vee \phi_2 \iff \alpha \vDash \phi_1 \text{ または } \alpha \vDash \phi_2$$

$\alpha \vDash \phi$ を満たす真理値割り当て α が存在するとき ϕ は充足可能であるといい，$\vDash \phi$ と書く．

定義 2 変数及び変数の否定をリテラル，任意個のリテラルの論理和を節，任意個の節の論理積を連言標準形（CNF）と呼ぶ．CNF の命題論理式の充足可能性を問う決定問題を SAT という．

定理 3 (Cook の定理) SAT は NP 完全問題である．

2-3-2 SAT ソルバ

SAT ソルバは与えられた CNF の充足可能性を判定し，更に充足可能である場合，充足する真理値割り当てを一つ出力するツールである．2002 年以降，SAT ソルバの性能を競う大会（SAT Competition, SAT Race）が毎年開催されており，飛躍な性能の向上が報告され続けている．現在利用できる SAT ソルバの多くは，この大会で用いられている入出力フォーマットを採用している（この入力書式は **DIMACS フォーマット** と呼ばれている）．以下，この入出力書式を簡単に説明する．

入力書式: 入力は `p cnf` m n という行で始める．m は使用する変数の個数，n は CNF の節数である．次の行からは節を次の記法で列挙する: 命題変数

は 1 から始まる整数で表し，変数の否定はマイナス記号を付与することで表現する．節は空白区切りの整数の列で終端に `0` を用いる．

出力書式: `s SATISFIABLE` という出力は入力の CNF が充足可能であることを表し，`s UNSATISFIABLE` は充足不可能を表す．前者の場合，更に充足させる真理値割り当てが `v` で始まる行に整数列として出力される．この列の正の整数は対応する変数の真理値割り当てが真であることを表し，負の整数は偽であることを表している．また列の終わりに `0` が置かれる．

例 4 $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_2) \wedge \neg x_3$ の充足可能性を SAT ソルバで解くには，図 2･1 左辺の入力を与えればよい．同図の右辺はそれに対する SAT ソルバの出力例である．出力は充足可能であり，実際 $\alpha(x_1) = T$，$\alpha(x_2) = \alpha(x_3) = F$ なる真理値割り当て α によって充足することを表している．

```
p cnf 3 4
1 -2 0
-1 2 -3 0
3 -2 0
-3 0
```

```
s SATISFIABLE
v 1 -2 -3 0
```

図 2･1　入力 (左) に対する SAT ソルバの出力例 (右)

多くの SAT ソルバは 1960 年代に登場した DPLL 法（Davis-Putnam-Loveland-Logemann 法）に基づく探索アルゴリズムを採用している．当時は数十変数の CNF が扱える程度であったが，1990 年代後半から衝突分析（conflict analysis）[1] や演繹を高速に行うデータ構造である two-watched literal[2] をはじめとする様々なブレークスルーがあり，Minisat などの最新の SAT ソルバは，数千から数万変数規模の CNF を短時間で解けるまでになっている．しかしどのような問題に対しても高速に解けるわけではなく，比較的小さな CNF でも極めて時間のかかるものもある．

入力 CNF と判定にかかる時間の関係は，一般に変数の数に実行時間が比例する傾向にあるが，節の数に関しては必ずしもそうではない．また解なしの場合の実行時間が解が存在する場合と比べて長くなるのは，探索アルゴリズム全般に見られる傾向であるが，現在の SAT ソルバは必ずしもそうではない．

2-3-3 論理式から CNF への変換

ここでは任意の命題論理式を CNF 形式に変換する手法を紹介する．論理式は次の否定の規則とド・モルガンの規則を左から右へ繰り返し用いることにより否定標準形になり，

$$\neg\neg\phi = \phi \qquad \neg(\phi \vee \psi) = \neg\phi \wedge \neg\psi \qquad \neg(\phi \wedge \psi) = \neg\phi \vee \neg\psi$$

更に $\wedge$ を外側へ引き出す分配律で同値変形を繰り返せば CNF になる．

$$(\phi \wedge \psi) \vee \xi = (\phi \vee \xi) \wedge (\psi \vee \xi) \qquad \phi \vee (\psi \wedge \xi) = (\phi \vee \xi) \wedge (\psi \vee \xi)$$

しかしこのような同値変形によって得られる CNF は，元の論理式に対して指数的に大きくなる場合がある．

例 5 論理式 $\phi_n = (x_1 \wedge y_1) \vee \cdots \vee (x_n \wedge y_n)$ を考える．例えば ϕ_3 を分配律で CNF に変換すると，

$$(x_1 \vee x_2 \vee x_3) \wedge (y_1 \vee x_2 \vee x_3) \wedge (x_1 \vee y_2 \vee x_3) \wedge (y_1 \vee y_2 \vee x_3) \wedge$$
$$(x_1 \vee x_2 \vee y_3) \wedge (y_1 \vee x_2 \vee y_3) \wedge (x_1 \vee y_2 \vee y_3) \wedge (y_1 \vee y_2 \vee y_3)$$

となる．このように ϕ_n と論理的に等価な CNF は ϕ_n の大きさに比べ指数的な大きさになる．

この問題を解決する手法である *Tseitin* **変換**を紹介する．この変換はたかだか定数倍の大きさで充足可能性が同じ CNF を生成する．まずは例 5 の ϕ_3 を用いてこの変換とそのアイディアを例示しよう．図 2･2 は ϕ_3 を論理回路として表したものである．

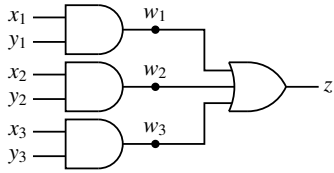


図 2･2　ϕ_3 の論理回路

図のように各ゲートの出力に命題変数を割り当てると，その命題変数と入出力の関係は $z \leftrightarrow (w_1 \vee w_2 \vee w_3)$ 及び各 $i = 1, 2, 3$ に対し，$w_i \leftrightarrow (x_i \wedge y_i)$ で表せる．それぞれを ψ と ξ_i と表せば，ϕ_3 が充足可能であることと $\psi \wedge \xi_1 \wedge \xi_2 \wedge \xi_3 \wedge z$ が充足可能であることは同値である．ψ と ξ_i は左辺が変数，右辺がリテラルの論理和か論理積の同値式であり，このかたちの論理式は直ちに CNF へ変換できる:

$$\psi = (\neg w_1 \vee z) \wedge (\neg w_2 \vee z) \wedge (\neg w_3 \vee z) \wedge (\neg z \vee w_1 \vee w_2 \vee w_3)$$
$$\xi_i = (\neg w_i \vee x_i) \wedge (\neg w_i \vee y_i) \wedge (\neg x_i \vee \neg y_i \vee w_i)$$

これらを $\psi \wedge \xi_1 \wedge \xi_2 \wedge \xi_3 \wedge z$ に代入すると ϕ_3 と充足可能性が同じ CNF を得る．この得られた CNF は元の論理式 ϕ_3 にはない命題変数を含むため，論理的に等価ではないが，それを充足する真理値割り当ては ϕ_3 も充足することに注意せよ．

次の定義はこの変換手法を定式化したものである．

定義 6 任意の論理式 ϕ から CNF への変換 ϕ° を $z \wedge \psi$ と定める．ここで $(z, \psi) = \phi^\bullet$ であり，$\phi^\bullet$ は次のように ϕ の構造に対して帰納的に定義される．

$$
\begin{aligned}
x^\bullet &= (x, \top) \\
(\neg\phi_1)^\bullet &= (\overline{y}, \psi_1) \\
(\phi_1 \vee \cdots \vee \phi_n)^\bullet &= (z, \psi \wedge (z \vee \overline{y_1}) \wedge \cdots \wedge (z \vee \overline{y_n}) \wedge (y_1 \vee \cdots \vee y_n \vee \overline{z})) \\
(\phi_1 \wedge \cdots \wedge \phi_n)^\bullet &= (z, \psi \wedge (\overline{z} \vee y_1) \wedge \cdots \wedge (\overline{z} \vee y_n) \wedge (\overline{y_1} \vee \cdots \vee \overline{y_n} \vee z))
\end{aligned}
$$

ただし x は命題変数，z は新しい命題変数，また各 i に対し $(y_i, \psi_i) = \phi_i^\bullet$, $\psi = \psi_1 \wedge \ldots \wedge \psi_n$ とする．また任意のリテラル y に対し $\overline{y}$ は，y が変数ならば $\neg y$，y が $\neg y'$ のかたちならば y' を表す．

定理 7 ϕ を任意の論理式とする．$\models \phi$ の必要十分条件は $\models \phi^\circ$ である．また任意の真理値割り当て α に対し，$\alpha \models \phi^\circ$ ならば $\alpha \models \phi$ が成立する．

2-3-4 算術の符号化

次の BNF の c で表される自然数に関する方程式と自然数 $k \geqslant 1$ に対し，SAT ソルバで 2^{k-1} 未満の自然数のなかから解を求める方法[3)]を紹介する．

$$
\begin{aligned}
e &::= x \mid n \mid e_1 + e_2 \mid e_1 \times e_2 && (x \text{ は変数，} n \text{ は自然数}) \\
c &::= (e_1 = e_2) \mid (e_1 > e_2)
\end{aligned}
$$

命題論理式の真偽を 0,1 と見たてると，整数はビット列，加算・乗算及び比較は論理演算として記述することができる．更に，各変数 x を k 個の命題変数 $(x_1, \ldots, x_k)$ を用いてビット列で表現すれば，与えられた方程式は純粋な論理式として記述でき，方程式を充足可能性問題として SAT ソルバで解くことができる．

定義 8 $k \geqslant 1$ を自然数とする．以下で定義される関数 B^+，$B^\times$，$B^=$，$B^>$ を用いて，算術式の k ビット符号化を次のように定義する．

$$
\ulcorner e \urcorner = \begin{cases}
(x_1, \ldots, x_k) & \text{if } e = x \\
(b_1, \ldots, b_{\lceil \log n \rceil}) & \text{if } e = n \\
B^+(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) & \text{if } e = e_1 + e_2 \\
B^\times(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) & \text{if } e = e_1 \times e_2
\end{cases}
\qquad
\ulcorner c \urcorner = \begin{cases}
B^=(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) & \text{if } c = (e_1 = e_2) \\
B^>(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) & \text{if } c = (e_1 > e_2)
\end{cases}
$$

ここで $b_1, \ldots, b_{\lceil \log n \rceil}$ は n の 2 進数表記で 0 と 1 をそれぞれ $\bot$ と $\top$ に置き換えた列である．以下では $(\phi_1, \ldots, \phi_n)$ と $(\psi_1, \ldots, \psi_m)$ をそれぞれ $\vec{\phi}$ と $\vec{\psi}$ と表す．

$$
B^+(\vec{\phi}, \vec{\psi}) = \begin{cases}
(\phi_1 \wedge \psi_1, \phi_1 \oplus \psi_1) & \text{if } n = m = 1 \\
((\phi_1 \wedge \psi_1) \vee (\psi_1 \wedge \xi_1) \vee (\xi_1 \wedge \phi_1), \phi_1 \oplus \psi_1 \oplus \xi_1, \xi_2, \ldots, \xi_n) & \text{if } n = m > 1 \\
B^+((\overbrace{\bot, \ldots, \bot}^{\max\{m,n\}-n}, \phi_1, \ldots, \phi_n), (\overbrace{\bot, \ldots, \bot}^{\max\{m,n\}-m}, \psi_1, \ldots, \psi_m)) & \text{if } n \neq m
\end{cases}
$$

$$B^{\times}(\vec{\phi},\vec{\psi}) = \begin{cases} (\phi_1 \wedge \psi_1, \ldots, \phi_n \wedge \psi_1) & \text{if } m = 1 \\ B^{+}((\phi_1 \wedge \psi_1, \ldots, \phi_n \wedge \psi_1, \overbrace{\bot, \ldots, \bot}^{m-1}), B^{\times}(\vec{\phi}, (\psi_2, \ldots, \psi_m))) & \text{if } m > 1 \end{cases}$$

$$B^{=}(\vec{\phi},\vec{\psi}) = \begin{cases} (\phi_1 \leftrightarrow \psi_1) \wedge \cdots \wedge (\psi_n \leftrightarrow \psi_n) & \text{if } n = m \\ B^{=}((\overbrace{\bot, \ldots, \bot}^{\max\{m,n\}-n}, \phi_1, \ldots, \phi_n), (\overbrace{\bot, \ldots, \bot}^{\max\{m,n\}-m}, \psi_1, \ldots, \psi_m)) & \text{if } n \neq m \end{cases}$$

$$B^{>}(\vec{\phi},\vec{\psi}) = \begin{cases} \phi_1 \wedge \neg\psi_1 & \text{if } n = m = 1 \\ (\phi_1 \wedge \neg\psi_1) \vee ((\phi_1 \leftrightarrow \psi_1) \wedge B^{>}((\phi_2, \ldots, \phi_n), (\psi_2, \ldots, \psi_n))) & \text{if } n = m > 1 \\ B^{>}((\overbrace{\bot, \ldots, \bot}^{\max\{m,n\}-n}, \phi_1, \ldots, \phi_n), (\overbrace{\bot, \ldots, \bot}^{\max\{m,n\}-m}, \psi_1, \ldots, \psi_m)) & \text{if } n \neq m \end{cases}$$

ただし B^{+} の定義において $(\xi_1, \ldots, \xi_n) = B^{+}((\phi_2, \ldots, \phi_n), (\psi_2, \ldots, \psi_m))$ とする．

例 9 方程式 $x + 1 = y \times y$ の整数解を 2 ビットで符号化する．両辺を符号化すると，

$$\ulcorner x + 1 \urcorner = (x_1 \wedge x_2, x_1 \oplus x_2, \neg x_2) \qquad \ulcorner y \times y \urcorner = (y_1 \wedge y_2, y_1 \wedge \neg y_2, \bot, \neg y_2)$$

となるので，$\ulcorner x + 1 = y \times y \urcorner$ は次の論理式と等価である．

$$(\bot \leftrightarrow (y_1 \wedge y_2)) \wedge ((x_1 \wedge x_2) \leftrightarrow (y_1 \wedge \neg y_2)) \wedge ((x_1 \oplus x_2) \leftrightarrow \bot) \wedge (\neg x_2 \leftrightarrow \neg y_2)$$

例えば，$\alpha(x_1) = \alpha(x_2) = \alpha(y_1) = T$ かつ $\alpha(y_2) = F$ である真理値割り当て α はこの式を充足する．ゆえに $x = (11)_2 = 3$，$y = (10)_2 = 2$ が方程式の解となる．

2-3-5 事例：項書換え系の停止性検証

SAT ソルバを用いた検証手法の事例として，項書換え系の停止性〔本章 1-3 参照〕の検証を取り上げる[3, 4]．現在の停止性検証ツールはほぼすべて SAT ソルバを用いて実装されている．ここでは辞書式経路順序を用いた停止性検証が，いかにして SAT へ帰着されるかを解説する．

以下では $\mathcal{F}$ を有限の関数記号の集合，$\mathcal{V}$ を変数の集合とし，これらから構成される項を考える．また項書換え系の規則数は有限であるものとする．

定義 10 $\triangleright$ を $\mathcal{F}$ 上の優先順位とする．項上の順序である辞書式経路順序 $>_{\mathrm{lpo}}$ は次のように定義される．$s >_{\mathrm{lpo}} t$ は，$s = f(s_1, \ldots, s_m)$ と表すことができ，以下の条件の少なくとも一つを満たす．

- ある $1 \leqslant i \leqslant m$ に対して $s_i = t$ または $s_i >_{\mathrm{lpo}} t$.
- $t = g(t_1, \ldots, t_n)$，すべての $1 \leqslant i \leqslant n$ に対して $s \succ t_i$ であり，更に $f \triangleright g$ が成立するか $f = g$ かつ $(s_1, \ldots, s_n) >_{\mathrm{lpo}}^{\mathrm{lex}} (t_1, \ldots, t_n)$ が成立する．

ここで $>_{\mathrm{lpo}}^{\mathrm{lex}}$ は $>_{\mathrm{lpo}}$ の辞書式順序を表す．

定理 11 $\mathcal{R}$ を項書換え系とする．ある優先順位 $>$ が存在し，すべての書換え規則 $l \to r \in \mathcal{R}$ が $l >_{\mathrm{lpo}} r$ を満たすのであれば，$\mathcal{R}$ は停止性をもつ．

定理 11 は，可能な優先順位が有限個しか存在しないので，網羅的な探索を行えば簡単に自動化できる．しかし可能な優先順位の数は集合 $\mathcal{F}$ の濃度に対し組合せ的な数存在するため，この方法は現実的ではない．そこで探索問題を SAT へ帰着させる．そのためにまず辞書式経路順序 $s >_{\mathrm{lpo}} t$ の定義を展開し，優先順位に関する条件だけからなる制約式へ変換する符号化を定義する．

定義 12 $\mathcal{F}$ 上の 2 項関係 $\triangleright$ と論理演算子からなる式を優先順位制約式と呼ぶ．優先順位制約式 ϕ が充足可能であるとは，制約を満たす $\mathcal{F}$ 上の順序，すなわち非反射かつ推移的な 2 項関係 $\triangleright$ が存在することである．

定義 13 優先順序式への符号化 $\ulcorner s >_{\mathrm{lpo}} t \urcorner$ と $\ulcorner s >^{=}_{\mathrm{lpo}} t \urcorner$ を以下のように帰納的に定義する．

$$\ulcorner s >_{\mathrm{lpo}} t \urcorner = \begin{cases} \bot & \text{if } s \in \mathcal{V} \\ \phi & \text{if } s = f(\vec{s}),\ t \in \mathcal{V} \\ \phi \vee (\psi \wedge \ulcorner \vec{s} >^{\mathrm{lex}}_{\mathrm{lpo}} \vec{t} \urcorner) & \text{if } s = f(\vec{s}),\ t = f(\vec{t}) \\ \phi \vee (\psi \wedge (f \triangleright g)) & \text{if } s = f(\vec{s}),\ t = g(\vec{t}),\ f \neq g \end{cases}$$

$$\ulcorner s >^{=}_{\mathrm{lpo}} t \urcorner = \begin{cases} \top & \text{if } s = t \\ \ulcorner s >_{\mathrm{lpo}} t \urcorner & \text{if } s \neq t \end{cases}$$

ここで $\vec{s}$ は $s_1, \ldots, s_n$，$\vec{t}$ は $t_1, \ldots, t_m$ を表し，ϕ，ψ，及び $\ulcorner \vec{s} >^{\mathrm{lex}}_{\mathrm{lpo}} \vec{t} \urcorner$（$n = m$ の場合）は以下のように定義される．

$$\phi = \bigwedge_{1 \leqslant i \leqslant m} \ulcorner s_i >^{=}_{\mathrm{lpo}} t \urcorner \qquad \psi = \bigwedge_{1 \leqslant i \leqslant m} \ulcorner s >_{\mathrm{lpo}} t_i \urcorner$$

$$\ulcorner \vec{s} >^{\mathrm{lex}}_{\mathrm{lpo}} \vec{t} \urcorner = \begin{cases} \bot & \text{if } n = 0 \\ \ulcorner s_2, \ldots, s_n >^{\mathrm{lex}}_{\mathrm{lpo}} t_2, \ldots, t_n \urcorner & \text{if } n > 0,\ s_1 = t_1 \\ \ulcorner s_1 >_{\mathrm{lpo}} t_1 \urcorner & \text{if } n > 0,\ s_1 \neq t_1 \end{cases}$$

項書換え系 $\mathcal{R}$ に対し，$\bigwedge_{l \to r \in \mathcal{R}} \ulcorner l >_{\mathrm{lpo}} r \urcorner$ を $\mathrm{LPO}(\mathcal{R})$ と表記する．

$\mathcal{R}$ を項書換え系としたとき，優先順位制約式 $\mathrm{LPO}(\mathcal{R})$ が充足可能であれば $\mathcal{R}$ は停止性をもつ．

例 14 アッカーマン関数を計算する項書き換え系 $\mathcal{R}$

$$\mathsf{a}(\mathsf{0}, y) \to \mathsf{0} \qquad \mathsf{a}(\mathsf{s}(x), \mathsf{0}) \to \mathsf{a}(x, \mathsf{s}(\mathsf{0})) \qquad \mathsf{a}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{a}(x, \mathsf{a}(\mathsf{s}(x), y))$$

に対する優先順序制約式 $\mathrm{LPO}(\mathcal{R})$ を計算する．各書換え規則を符号化すると

$$\ulcorner \mathsf{a}(\mathsf{0}, y) >_{\mathrm{lpo}} \mathsf{0} \urcorner = \ulcorner \mathsf{a}(\mathsf{s}(x), \mathsf{s}(y)) >_{\mathrm{lpo}} \mathsf{a}(x, \mathsf{a}(\mathsf{s}(x), y)) \urcorner = \top$$

$$\ulcorner \mathsf{a}(\mathsf{s}(x), \mathsf{0}) >_{\mathrm{lpo}} \mathsf{a}(x, \mathsf{s}(\mathsf{0})) \urcorner = \mathsf{a} \triangleright \mathsf{s}$$

となるので, $\mathsf{LPO}(\mathcal{R}) = (\mathsf{a} \triangleright \mathsf{s})$ である.

例 15 次の項書換え系 $\mathcal{R}$ は減算と除算を記述したものである.

$$\mathtt{sub}(x, \mathtt{0}) \to x \qquad \mathtt{div}(\mathtt{0}, \mathtt{s}(y)) \to \mathtt{0}$$
$$\mathtt{sub}(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{sub}(x, y) \qquad \mathtt{div}(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{s}(\mathtt{div}(\mathtt{sub}(x, y), \mathtt{s}(y)))$$

各書換え規則を符号化すると

$$\ulcorner \mathtt{sub}(x, \mathtt{0}) > x \urcorner = \ulcorner \mathtt{div}(\mathtt{0}, \mathtt{s}(y)) \urcorner = \ulcorner \mathtt{sub}(\mathtt{s}(x), \mathtt{s}(y)) \urcorner = \top$$
$$\ulcorner \mathtt{div}(\mathtt{s}(x), \mathtt{s}(y)) > \mathtt{s}(\mathtt{div}(\mathtt{sub}(x, y), \mathtt{s}(y))) \urcorner = (\mathtt{div} \triangleright \mathtt{s}) \wedge (\mathtt{sub} \triangleright \mathtt{s}) \wedge (\mathtt{s} \triangleright \mathtt{sub})$$

となるので, $\mathsf{LPO}(\mathcal{R}) = (\mathtt{div} \triangleright \mathtt{s}) \wedge (\mathtt{sub} \triangleright \mathtt{s}) \wedge (\mathtt{s} \triangleright \mathtt{sub})$ である.

優先順序制約式の充足可能性を命題論理式の充足可能性に帰着させる. 以下, 優先順序制約式を命題論理式として扱う場合は各 $(f \triangleright g)$ を命題変数とみなすこととする.

補題 16 非反射律と推移律を表す論理式 A と T を次のように定める.

$$\mathsf{A} = \bigwedge_{f \in \mathcal{F}} \neg(f \triangleright f) \qquad \mathsf{T} = \bigwedge_{f,g,h \in \mathcal{F}} ((f \triangleright g) \wedge (g \triangleright h) \to (f \triangleright h))$$

優先順序制約式 ϕ が充足可能であることと命題論理式 $\phi \wedge \mathsf{A} \wedge \mathsf{T}$ が充足可能であることは同値である.

定理 17 項書換え系 $\mathcal{R}$ は $\models \mathsf{LPO}(\mathcal{R}) \wedge \mathsf{A} \wedge \mathsf{T}$ のとき停止性をもつ.

例 18 (例 14 の続き) $\mathsf{A} = \neg(\mathsf{a} \triangleright \mathsf{a}) \wedge \neg(\mathsf{s} \triangleright \mathsf{s}) \wedge (\mathtt{0} \triangleright \mathtt{0})$, また $\mathsf{T} = (((\mathsf{a} \triangleright \mathsf{s}) \wedge (\mathsf{a} \triangleright \mathtt{0})) \to (\mathsf{a} \triangleright \mathtt{0})) \wedge (((\mathsf{s} \triangleright \mathsf{a}) \wedge (\mathsf{a} \triangleright \mathtt{0})) \to (\mathsf{a} \triangleright \mathsf{s})) \wedge \cdots$ である. 真理値割り当て α を $(\mathsf{a} \triangleright \mathsf{s})$ に対してのみ T, ほかは F と定めると, α は $\mathsf{LPO}(\mathcal{R}) \wedge \mathsf{A} \wedge \mathsf{T}$ を充足する. よって $\mathcal{R}$ は停止性をもつ.

例 19 (例 15 の続き) $\mathsf{A} = \neg(\mathtt{div} \triangleright \mathtt{div}) \wedge \neg(\mathtt{sub} \triangleright \mathtt{sub}) \wedge \neg(\mathtt{s} \triangleright \mathtt{s}) \wedge \neg(\mathtt{0} \triangleright \mathtt{0})$, また $\mathsf{T} = \cdots \wedge (((\mathtt{div} \triangleright \mathtt{sub}) \wedge (\mathtt{sub} \triangleright \mathtt{div})) \to (\mathtt{div} \triangleright \mathtt{div})) \wedge \cdots$ であるので, $\mathsf{LPO}(\mathcal{R}) \wedge \mathsf{A} \wedge \mathsf{T}$ は充足不可能であり, よって辞書式経路順序では $\mathcal{R}$ の停止性を示せないことが分かる.

■参考文献

1) J.P. Marques-Silva and K.A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," IEEE Trans. Comput., 1999.
2) M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," DAC, 2001.
3) C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl, "SAT Solving for Termination Analysis with Polynomial Interpretations," SAT 2007, LNCS 4501, pp.340-354, 2007.
4) M. Kurihara and H. Kondo, "Efficient BDD Encodings for Partial Order Constraints with Application to Expert Systems in Software Verification," IEA/AIE 2007, LNCS 3029, pp.827-837, 2004.

■7 群 - 1 編 - 2 章

2-4 モデル検査（総論）

（執筆者：関澤俊弦・高橋孝一）[2008 年 8 月 受領]

モデル検査（model checking）[1)−6)]は，検証対象を表すモデル $\mathcal{M}$ と検証したい性質 φ が与えられたとき，φ が $\mathcal{M}$ で成り立つか否かを，網羅的な全数探索（exhaustive search）によって検査する手法である．モデル検査は，自動検証法の一つとして期待されている．

当初，時相論理に関するモデル検査が開発され，論理回路に代表されるハードウェアの検証に適用されて成功を収めた．その後，モデル検査は大きく発展し，様々な検証に利用されるようになっている．例えば，通信プロトコルや認証アルゴリズム，リアクティブシステムや実時間システムの設計などである．更には C 言語や Java などで書かれたプログラムソースコードの検証にまで広がっている．

2-4-1 概　要

全数探索によって望ましい性質をもつかどうか検査する方法はすべてモデル検査と呼ばれる場合がある．それに対し，狭義のモデル検査は，検証対象を状態遷移系（transition system）で表し，検証したい性質を時相論理式で記述する手法である．ここでは狭義のモデル検査を中心に，モデル検査の概要を述べることにする．ただし，一般のモデル検査についても概要は同様である．モデル検査による検証手続きは，モデル化，モデル検査の実行，結果の解析に分けられる．モデル検査による検証手続きの概要を図 2･3 に示す．

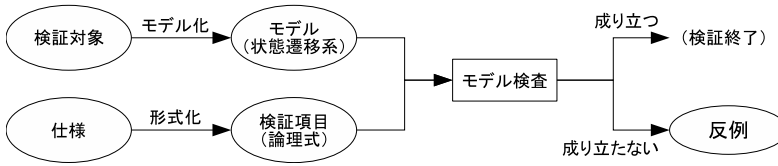


図 2･3　モデル検査による検証手続きの概要

（a）モデル化

モデル化では，検証対象及び仕様が，モデル検査の入力となる状態遷移系及び検証項目により記述される． 状態遷移系は，検証対象の振る舞いを有限の状態の集合と状態間の遷移関係で表したものである．状態は変数の値などの情報をもち，状態が移り変わるときの動作を表して状態間を関係づける．

例として，図 2･4 に示す交差点の信号機システムを取り上げる．信号機は，赤，青，黄の順番で点灯色が変わる．信号機の点灯色を状態，点灯色の変化を遷移として捉え，信号機システムを状態遷移系で表す．東西方向の信号機の点灯色を r, g, y と書くことにする．同様に，南北方向の信号機の点灯色を R, G, Y と書く．すると，システム全体の状態は，各信号の組で表される．点灯色の変化に従った状態間の遷移を考え，信号機システムを状態遷移系として記述したものが図 2･5 である．

状態遷移系は，人手で記述されることが多い．人手によるモデル化の誤りの防止や効率化のために，検証対象から自動的に生成する手法もある．モデル検査では，検証したい性質が

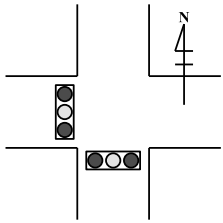


図 2·4　交差点の信号機システム

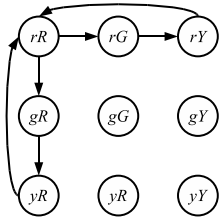


図 2·5　信号機システムの状態遷移系

状態遷移系の上で成り立つか否かを判定するため，モデルが表現している以上のことは検証できない．したがって，作成した状態遷移系が検証対象を正確に表現しているかの妥当性確認（validation）が重要である．例えば，実際の信号機システムでは，東西と南北の信号機が交互に青になる．しかし，図 2·5 の状態遷移系は，そのことを表現していない．

検証項目は，検証対象において検証したい性質を記述した論理式である．検証項目の記述には時相論理（temporal logic）が広く用いられている．時相論理は，命題の真理値の時間経過による変遷を記述することができる論理体系である．広く用いられている時相論理として，線形時相論理（LTL）と計算木論理（CTL）がある．時相論理は，リアクティブシステムの仕様を有効に記述できる．代表的な性質として，安全性（safety）と活性（liveness）がある．

- **安全性**「望ましくない事象は決して発生しない」という性質．信号機の例では，「二つの信号機が同時に青になることはない」などが安全性としてあげられる．「デッドロックは決して起こらない」というデッドロックフリー性（deadlock freedom）や「常に成り立ち続ける」という不変性（invariant）も安全性に含まれる．
- **活性**「望ましい事象はいつか必ず発生する」という性質．信号機の例では，「信号はいつか必ず青になる」などが活性としてあげられる．「要求があったら必ず応える」という応答性（response property）や「必ず終了する」という停止性（termination）も活性に含まれる．

(b) モデル検査の実行

モデル検査の実行とは，状態遷移系上で網羅的な全数探索によって検証項目が成り立つかどうかの判定を行うことである．この判定手続きはモデル検査器（model checker）として実装され，自動検証が可能になっている．

しかしながら，全数探索では，探索を行う状態空間の大きさが問題となる．状態遷移系の状態空間の大きさは，検証対象を表すのに必要なパラメータの数に対して多くの場合指数的に増加する．この問題は状態爆発問題（state explosion problem）と呼ばれ，モデル検査における最大の問題である．例えば，ソフトウェアモデル検査で検証対象のプログラム中に，32bit の整数型変数が 10 個用いられている場合，全状態数は $(2^{32})^{10}$ となる．そのため，実問題にモデル検査を適用するためには，モデル化の際に抽象化などの手法を用いて，状態数を削減する取り組みが必要である．

(c) 結果の解析

検証項目が状態遷移系で成り立つことが検証されたとき，その検証項目に関するモデル検査は終了する．一方，検証項目が成り立たないと判定されたときは，検証項目が成り立たない証拠である反例（counterexample）が得られる．この反例を解析し，反例が検証対象において発生しうる問題である場合は，検証対象が仕様を満たさないことが分かる．

(d) 強みと弱み

最後に，モデル検査の強みと弱みをまとめる．強みとしては，次の項目があげられる．

- **適用可能性**　モデル検査技法は，ハードウェア設計から，ソフトウェア設計，プロトコル，ソフトウェア実装に至るまで，状態遷移系によって対象を表現できる領域に適用することができる．
- **全数探索による検証**　テストやシミュレーションとは異なり，全数探索により，検証項目として記述した性質を取りうるすべての経路に対して検証できる．
- **自動検証**　定理証明などのほかの形式検証とは異なり，モデル検査は自動化されている．効率的なツールと計算機の進歩により，かなり大きな対象を検証することが可能になっている．
- **反例の生成**　検証項目が成り立たないときには反例が生成される．反例は，検証項目が成立しない経路を含むため，反例を解析することにより検証対象を修正するのに役立つ．

弱みとしては，次の項目があげられる．

- 有限状態遷移系で表現できる対象のみに適用可能である．状態空間が無限となるような検証対象を素直に検証することはできない．
- 状態爆発問題．モデル検査ツールが扱える対象は大きくなっているが，実問題は簡単にその限界を超してしまう．したがって，状態数の少ないモデルを作成することがモデル検査の鍵である．
- 検証項目はモデルに対して検証される．モデル化の方法によっては，検証対象の振る舞いとモデルの振る舞いが異なる．その結果，正しい検証結果を得られないことがある．このことは，前項目とトレードオフの関係にあり，状態数を削減するために単純化をしすぎることでモデルの正しさが損なわれることがある．

2-4-2 時相論理モデル検査

時相論理モデル検査は，クリプキ構造 $\mathcal{M}$ と検証項目である時相論理式 φ を入力とし，φ が $\mathcal{M}$ で成り立つか否か，すなわち $\mathcal{M} \models \varphi$ を判定する．代表的な時相論理が線形時相論理（Linear Temporal Logic: LTL）と計算木論理（Computation Tree Logic: CTL）である [7]．

(1) クリプキ構造

クリプキ構造（Kripke structure）は，状態遷移系であり，各状態で成り立つ原子命題が定められたものである．原子命題の集合を $AP = \{p_1, p_2, \ldots, p_m\}$ とする．クリプキ構造は，次

を満たす四つ組 (S, R, S_0, L) で定義される．ここで，S は状態の集合，$R \subseteq S \times S$ は状態集合間の関係で，状態間の遷移関係を表す．状態 $s, s' \in S$ に対して $(s, s') \in R$ のとき，状態 s から状態 s' に遷移するという．$S_0 \subseteq S$ は 空でない状態集合で，初期状態を表す．$L : S \to 2^{AP}$ はラベリング関数である．

集合 $\{s' \in S \mid (s, s') \in R\}$ の要素数が 1 のとき，状態 s からの遷移は決定的（deterministic）であるといい，要素数が 2 以上のときは非決定的（non-deterministic）であるという．要素数が 0 のときは，状態 s を終端（terminal）という．遷移関係 R に対して，すべての状態に遷移先が存在するとき，すなわち $\forall s \in S \exists s' \in S[(s, s') \in R]$ であるとき，クリプキ構造は全域的（total）であるという．

クリプキ構造の上の実行経路（path）とは，状態の無限列 $s_0 s_1 \cdots$ であり，隣り合う状態同士が遷移関係にあるものをいう．実行経路 π の i 番目の状態を $\pi(i)$ と書き，π の i 番目から始まる実行経路 $s_i s_{i+1} \cdots$ を π^i と書く．

（2）線形時相論理

時相論理の体系として，大きく分けて二つの時間の捉え方がある．捉え方の一つは，ある時点に対して次の時点はただ一つしかないと捉える．もう一つは，時間は木構造をなすように分岐しており，次の時点に複数の可能性があるものと捉える．

前者の立場の時相論理の体系の一つが線形時相論理（LTL）である．LTL の文法は次の BNF で定義される．

$$\varphi ::= \bot \mid \top \mid p_i \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathtt{X}\varphi \mid \mathtt{F}\varphi \mid \mathtt{G}\varphi \mid \varphi\ \mathtt{U}\ \varphi \mid \varphi\ \mathtt{R}\ \varphi$$

LTL 論理式は，全域的クリプキ構造 $\mathcal{M} = (S, R, S_0, L)$ の実行経路 π に対して以下のように解釈される．

$$\begin{aligned}
\pi &\not\models \bot \\
\pi &\models \top \\
\pi &\models p_i && \iff && p_i \in L(\pi(0)) \\
\pi &\models \neg\varphi && \iff && \pi \not\models \varphi \\
\pi &\models \varphi_1 \vee \varphi_2 && \iff && \pi \models \varphi_1 \text{ または } \pi \models \varphi_2 \\
\pi &\models \varphi_1 \wedge \varphi_2 && \iff && \pi \models \varphi_1 \text{ かつ } \pi \models \varphi_2 \\
\pi &\models \mathtt{X}\varphi && \iff && \pi^1 \models \varphi \\
\pi &\models \mathtt{F}\varphi && \iff && \exists i\, [\pi^i \models \varphi] \\
\pi &\models \mathtt{G}\varphi && \iff && \forall i\, [\pi^i \models \varphi] \\
\pi &\models \varphi_1\ \mathtt{U}\ \varphi_2 && \iff && \exists i\, [\pi^i \models \varphi_2 \text{ かつ } \forall j < i\, [\pi^j \models \varphi_1]] \\
\pi &\models \varphi_1\ \mathtt{R}\ \varphi_2 && \iff && \forall i\, [\pi^i \models \varphi_2 \text{ または } \exists j < i\, [\pi^j \models \varphi_1]]
\end{aligned}$$

直感的な意味を次に記す．線形時相演算子（linear temporal operator）$\mathtt{X}$（neXt），$\mathtt{F}$（Future），$\mathtt{G}$（Globally）は単項演算子であり，それぞれ，実行経路の「次に」「いつか，どこかで」「常に，いつでも」を意味する．線形時相演算子 $\mathtt{U}$（Until）は二項演算子であり，$\varphi_1\ \mathtt{U}\ \varphi_2$ は「いずれ

φ_2 が成り立ち，それまではずっと φ_1 が成り立ち続ける」を意味する．R（Release）は U の双対であり，$\varphi_1\,\mathtt{R}\,\varphi_2$ は「φ_1 が成り立つまでは，ずっと φ_2 が成り立ち続ける」を意味する．

クリプキ構造 $\mathcal{M}$ において，$\pi(0) \in S_0$ である任意の実行経路について $\pi \models \varphi$ が成り立つとき，$\mathcal{M}$ において φ が成り立つと定義し，$\mathcal{M} \models \varphi$ と書く．

LTL モデル検査アルゴリズムは，オートマトンを用いた判定アルゴリズムが広く使われている．与えられた LTL 式を満たす実行列すべてをちょうど受理する Büchi オートマトン（Büchi automaton）を作成し，クリプキ構造の任意の実行経路がオートマトンで受理できるか調べる．その計算量は，モデルの大きさに対して線形だが，LTL 式の長さに対しては指数時間かかるものしか知られていない．

(3) 計算木論理

時間は木構造をなすように分岐しており，ある時点に対して複数の続く時点があると捉える分岐時相論理（branching temporal logic）として，計算木論理（CTL）がある．CTL では，線形時相演算子 G, F, X, U, R に経路限量子（path quantifier）A, E が付随する．CTL の文法は次の BNF で定義される．

$$\varphi ::= \quad \bot \mid \top \mid p_i \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$
$$\mid \mathtt{AX}\varphi \mid \mathtt{EX}\varphi \mid \mathtt{AF}\varphi \mid \mathtt{EF}\varphi \mid \mathtt{AG}\varphi \mid \mathtt{EG}\varphi \mid \mathtt{A}[\varphi\,\mathtt{U}\,\varphi] \mid \mathtt{E}[\varphi\,\mathtt{U}\,\varphi] \mid \mathtt{A}[\varphi\,\mathtt{R}\,\varphi] \mid \mathtt{E}[\varphi\,\mathtt{R}\,\varphi]$$

CTL 論理式は，全域的クリプキ構造 $\mathcal{M} = (S, R, S_0, L)$ の状態 $s \in S$ で次のように解釈される．

$$\begin{aligned}
s &\not\models \bot \\
s &\models \top \\
s \models p_i &\iff p_i \in L(s) \\
s \models \neg\varphi &\iff s \not\models \varphi \\
s \models \varphi_1 \vee \varphi_2 &\iff s \models \varphi_1 \text{ または } s \models \varphi_2 \\
s \models \varphi_1 \wedge \varphi_2 &\iff s \models \varphi_1 \text{かつ } s \models \varphi_2 \\
s \models \mathtt{AX}\varphi &\iff \forall s'[(s,s') \in R \text{ ならば } s' \models \varphi] \\
s \models \mathtt{EX}\varphi &\iff \exists s'[(s,s') \in R \text{ かつ } s' \models \varphi] \\
s \models \mathtt{AF}\varphi &\iff \forall\pi[\pi(0) = s \text{ ならば } \exists i[\pi(i) \models \varphi]] \\
s \models \mathtt{EF}\varphi &\iff \exists\pi[\pi(0) = s \text{ かつ } \exists i[\pi(i) \models \varphi]] \\
s \models \mathtt{AG}\varphi &\iff \forall\pi[\pi(0) = s \text{ ならば } \forall i[\pi(i) \models \varphi]] \\
s \models \mathtt{EG}\varphi &\iff \exists\pi[\pi(0) = s \text{ かつ } \forall i[\pi(i) \models \varphi]] \\
s \models \mathtt{A}[\varphi_1\,\mathtt{U}\,\varphi_2] &\iff \forall\pi[\pi(0) = s \text{ ならば } \exists i[\pi(i) \models \varphi_2 \text{ かつ } \forall j < i[\pi(j) \models \varphi_1]]] \\
s \models \mathtt{E}[\varphi_1\,\mathtt{U}\,\varphi_2] &\iff \exists\pi[\pi(0) = s \text{ かつ } \exists i[\pi(i) \models \varphi_2 \text{ かつ } \forall j < i[\pi(j) \models \varphi_1]]] \\
s \models \mathtt{A}[\varphi_1\,\mathtt{R}\,\varphi_2] &\iff \forall\pi[\pi(0) = s \text{ ならば } \forall i[\pi(i) \models \varphi_2 \text{ または } \exists j < i[\pi(j) \models \varphi_1]]] \\
s \models \mathtt{E}[\varphi_1\,\mathtt{R}\,\varphi_2] &\iff \exists\pi[\pi(0) = s \text{ かつ } \forall i[\pi(i) \models \varphi_2 \text{ または } \exists j < i[\pi(j) \models \varphi_1]]]
\end{aligned}$$

線形時相演算子の直感的な意味は LTL の場合と同様であり，経路限量子 A（forAll）は「すべての経路で」を意味し，経路限量子 E（Exists）は「ある経路が存在する」を意味する．

クリプキ構造 $\mathcal{M}$ において，S_0 の任意の状態 s について $s \models \varphi$ が成り立つとき，$\mathcal{M}$ において φ が成り立つと定義し，$\mathcal{M} \models \varphi$ と書く．

CTL モデル検査の判定アルゴリズムとしては，ラベリングアルゴリズム（labeling algorithm）が広く知られている．その計算量は，モデルの大きさと CTL 式の長さの両方に対して線形である．

多くの性質は LTL でも CTL でも表現できる．「決して p は成り立たない」という安全性が成り立つかどうかは，LTL では $\mathcal{M} \models \mathtt{G}\neg p$ となり，CTL では $\mathcal{M} \models \mathtt{AG}\neg p$ となる．しかし，LTL では書けるが CTL では等価な性質を書けない場合（例えば $\mathtt{FG}p$）もあり，逆に CTL では書けるが LTL では書けない場合（例えば $\mathtt{AG}(\mathtt{EF}p)$）もある．

CTL では記述が困難なものとして，公平性（fairness）を仮定した上の安全性や活性がある．公平性は，複数のプロセスが動作しているとき，ある特定のプロセスのみが動作することなく，すべてのプロセスに動作の機会が与えられることをいう．先の信号機の例では，公平性の制約を前提としない場合，片方の信号のみが繰り返し青となり，もう一方の信号が青にならない状況が発生するなどの例があげられる．CTL モデル検査を拡張した，公平性付き CTL モデル検査（CTL model checking with fairness）も考案されている．

LTL と CTL の両方の拡張として，線形時相演算子と経路限量子の自由な組合せを許した論理を CTL*という．CTL*モデル検査の計算量は本質的に LTL モデル検査と同じであることが知られている．

2-4-3 モデル検査の発展

モデル検査は自動検証の強力な手法であるが，状態爆発問題という大きな問題を抱えている．モデル検査の研究は状態爆発問題への挑戦と言っても過言ではない．本節では，状態爆発問題への代表的な取り組みを概観する．また，時相論理モデル検査以外にも様々なモデル検査が考えられている．これについても簡単に紹介する．

記号モデル検査（symbolic model checking）は，状態の集合を二分決定グラフ（binary decision diagram: BDD）と呼ばれる構造を用いて記述し，モデル検査を BDD 上の操作によって行う．記号モデル検査によって非常に大きな状態空間も扱うことが可能になった．

半順序簡約（partial order reduction）は，遷移の順番を入れ替えても結果に影響を与えない場合に，一定の順番しか探索しないことによって探索空間を削減する手法である．並行システムでは，このようなことがしばしば起こるため，半順序簡約は非常に効果がある．

対称性簡約（symmetry reduction）は，対称性をもつ類似した状態を同値類として扱うことにより，状態空間を削減する．

抽象化（abstraction）は，与えられた状態遷移系から，より状態数の少ない状態遷移系を構築する手法である．抽象化の代表的な手法としてデータマッピング（data mapping）や述語抽象化（predicate abstraction）が知られている．

有界モデル検査（bounded model checking）は，探索空間を一定の深さに制限し，その範囲に反例があるかどうかを論理式の充足可能性判定問題（SAT problem）に帰着させ，充足可能性判定器（SAT solver）を用いて判定する方法である．充足可能性判定器は非常に高速

な実装が存在するため，有界モデル検査は大きなモデルを検査することが可能である．また，完全なモデル検査が困難な非常に大きなモデルに対しても，ある深さまでには反例が存在しないという部分的な結果を得られる．

モデル検査は，設計やプロトコルの検証に有効なことが知られているが，近年は，抽象化や有界モデル検査を利用し，プログラムのソースコードを直接モデル検査する研究も盛んに行われている．

時相論理モデル検査では扱うことが困難な対象に関しても，モデルや論理が拡張され，様々なモデル検査が提案されている．

実時間システムは一般的であるが，クリプキ構造によってモデル化することは困難である．そこで，実時間を扱えるようなモデル検査が多く提案された．例えば，時間オートマトン（timed automaton）は時計（clock）と呼ばれる実数値を取る変数を有限個もつオートマトンである．

確率的な振る舞いを検証したい要求に対しては, モデルとして離散時間マルコフ連鎖（Discrete Time Markov Chain: DTMC）をモデルとし，論理として PCTL（Probabilistic CTL）をとる確率モデル検査がある [8]．統計的な振る舞いに対しても，モデル検査が考えられている．

最後に代表的なモデル検査ツールを列挙しておく．

- SMV, NuSMV: 記号モデル検査を実装している．
- SPIN: 半順序簡約を実装している．
- LTSA: 有限状態機械によって仕様も記述する．
- UPPAAL: 時間オートマトンを検証する．
- PRISM: 確率モデルを検証する．
- Java PathFinder: Java のバイトコードを対象とする．
- SLAM: 述語抽象化によりデバイスドライバの C ソースコードを検証する．
- CBMC: 有界モデル検査により C ソースコードを検証する．

■参考文献

1) Z. Manna and A. Pnueli, “The Temporal Logic of Reactive and Concurrent Systems: Specification,” Springer-Verlag, 1991.
2) Z. Manna and A. Pnueli, “Temporal Verification of Reactive Systems: Safety,” Springer-Verlag, 1995.
3) E.M. Clarke, O. Grumberg, and D.A. Peled, “Model Checking,” The MIT Press, 2000.
4) 米田知洋, 梶原誠司, 土屋達弘, “ディペンダブル システム,” 共立出版, 2005.
5) C. Baier and J.P. Katoen, “Principles of Model Checking,” The MIT Press, 2008.
6) “25 Years of Model Checking,” ed. by O. Grumberg and H. Veith, LNCS 5000, Springer, 2008.
7) E.A. Emerson, “Temoporal and Modal Logic,” in Handbook of Theoretical Computer Science Vol. B: Formal Models and Semantics,” The MIT Press, pp.997-1072, 1990. (和訳: “様相論理と時間論理,” コンピュータ基礎理論ハンドブック II 形式的モデルと意味論, pp.963-1037, 丸善, 1992.)
8) H. Hansson and B. Jonsson, “A Logic for Reasoning about Time and Reliability,” Formal Aspects of Computing, vol.6, no.5, pp.512-535, 1994.

■7 群 - 1 編 - 2 章

2-5 UML/ステートチャート

（執筆者：青木利晃）[2008 年 8 月 受領]

2-5-1 UML 概要

オブジェクト指向開発手法は，開発フェーズの観点から，オブジェクト指向分析（Object-Oriented Analysis: OOA），オブジェクト指向設計（Object-Oriented Design: OOD），オブジェクト指向プログラミング（Object-Oriented Programming: OOP）に分類される．1990 年代に入り，OOA/OOD を扱う方法論が非常に多く提案され，それぞれにおいて記法も技法も異なっていた．そのため，オブジェクト指向方法論を整理し，基本的な概念をまとめようという動きが出てきた．この作業の中心を担ったのは，代表的なオブジェクト指向方法論の提案者である G. Booch，J. Rumbaugh，I. Jacobson であり，統一メソッド（Unified Method）を提案した．統一メソッドでは，記法と技法の両方を統合することを目的としていたが，技法の統合に困難さがあったため，これらをそれぞれ切り離して標準化を行った．そして，1997 年に記法を標準化した UML（Unified Modeling Language）の初期バージョンが OMG（Object Management Group）により採択された．UML の現在（2007 年）の最新バージョンは 2.1 である．

UML は，複数の図式（ダイアグラム）の書き方から構成されている．主要な図としては，クラス図，アクティビティ図，パッケージ図，シーケンス図，配置図，コラボレーション図，ユースケース図，ステートチャート図などがある．対象システムの開発の際，これらすべての図が使われるのではなく，それぞれの興味や意図を記述しやすい図が用いられるのが一般的である．また，これらの図式だけでは，動作や制約の詳細を記述することは困難である．そこで，図式を補足する言語として，OCL（Object Contraint Language）[5]や Action Semantics[7]と呼ばれるものがある．OCL は，UML の図に対して，一階述語論理に基づいた制約を宣言的に記述するための制約記述言語である．クラスがもつ操作や属性などを参照して，オブジェクト指向的に制約を記述することができる．Action Semantics は，UML の図に対して，手続き的に動作を記述するための動作記述言語を規定するものである．実装環境に依存しない形式で動作を記述し，モデルからの自動コード生成を行う MDA（Model Driven Architecture）[4]に応用することが意図されている．更に，UML の図を構成する要素の役割や用法上の違いを表現するために，ステレオタイプと呼ばれる拡張記法も準備されている．ステレオタイプは UML の仕様の外側で，目的ごとに整理されてステレオタイプの集合として定義されている．アプリケーション領域特有の概念を記述することに用いられる場合が多く，それぞれの領域ごとに標準化が行われている．

このように UML ではシステム開発で用いられる図式を定義しているが，記法面の標準化に注力されており，それにより記述されたものの意味については厳密には定義されていない．記述されたものに対してある程度の合意が得られるくらいの意味はつけられているが，形式検証などを行うには不十分である．そのため，UML により記述されたものの意味の詳細を独自に決めた形式化や，検証手法が提案されている．

2-5-2 クラス図

クラス図は，対象システムの静的な構造を記述するためのものであり，オブジェクトの集合

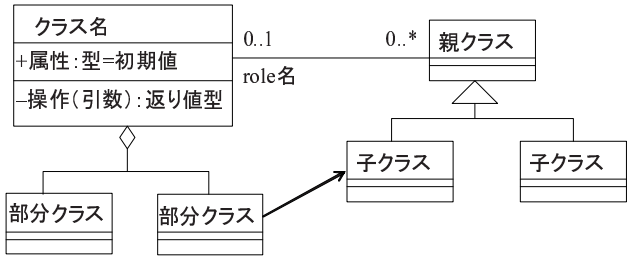


図 2･6　クラス図

を表現するクラスとそれらの間の関係で構成されている．図 2･6 にクラス図の例を示す．クラスには属性や操作を記述し，それらについて公開（public），非公開（private），保護（protected）などの可視性をつけることができる．

クラス間の関係としては，関連（association），継承（inheritance），集約（aggregation）などがある．関連は，クラス間になんらかの概念的なかかわり合いがあるときに用いる汎用的な関係である．クラスはオブジェクトの集合を表現しているため，一つのオブジェクトに対して何個のオブジェクトが対応するかを明確に記述したい場合がある．そこで，関連には多重度（multiplicity）をつけ，1 対 1，1 対 N，M 対 M などの関連に参加するオブジェクトの個数を記述することができる．

継承関係は，クラス間の「is-a」関係を表現するものである．例えば，B が A を継承する場合，「B is a A」，すなわち，B は A の 1 種であるという関係を意味している．この場合，A のことを親クラス（super class），B のことを子クラス（sub class）と呼び，A の方がより一般的なオブジェクトを，B の方がより特殊な限定されたオブジェクトを表現するクラスとなっている．

集約関係は，クラス間の「has-a」関係を表現するものである．例えば，A が B を集約する場合，「A has a B」，すなわち，A は B をもっているという関係を意味している．この場合，B のオブジェクトは A のオブジェクトの一部になっていることを表現している．関連と同様，集約関係にも多重度をつけることができる．

2-5-3 ステートチャート図

ステートチャート図は，対象システムの動的な側面である振る舞いを記述するためのものである．図 2･7 にステートチャート図の例を示す．基本的な構成要素は，状態と状態遷移である．状態遷移にはイベントやアクションを付加することができ，イベントが発生すると，その状態遷移が発火し，アクションを実行することを表現している．これらの状態遷移モデルを表現する単純な概念だけでは，複雑な振る舞いを記述するには不十分である．状態と状態遷移が多くなりすぎるからである．そこで，ステートチャート図では，複雑な状態遷移モデルを簡潔に記述するための工夫がされている．

D. Harel らは，状態を階層化したり並行化することにより，複雑な振る舞いを簡潔に記述する手法を提案した．UML のステートチャート図にも，このような状態の階層化と並行化

の概念が導入されている．また，イベントに引数や条件を付加することにより状態遷移をまとめて記述したり，状態内にアクションを埋め込むことにより，その状態に出入りする状態遷移に関するアクションをまとめて記述できるよう拡張されている．

D. Harel らの提案したステートチャートには明確な意味論が定義されており[1)]，それに基づいて実行するツールも提供されている[1)]．UML のステートチャート図は，D. Harel らのステートチャートとは，一部，異なる意味がつけられている．一方で，当時から，そのような状態遷移モデルの記述法に関しては，様々な解釈が可能であることが知られていた[3)]．UML のステートチャート図では，唯一に実行方式が決定できるほど厳密には意味を定義していないため，決まっていない部分に関しては様々な解釈が可能である．どのような意味づけが適切かは，対象システムの特性に依存することもあり，あえて，未定義にしているという意図もある．

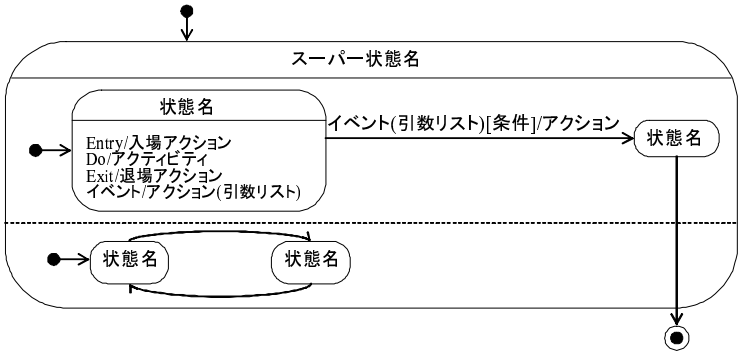


図 2･7　ステートチャート図

2-5-4 OCL と Action Semantics

OCL は，UML の図に関する特性を制約として記述するための言語であり，集合や算術計算を含む一階述語論理に基づいた制約を，オブジェクト指向的に記述することができる．UML の各図では，図の形式だけでは表現しきれない特性があり，この OCL により補足する場合がある．図 2･8 は，クラス図に対して，OCL による制約記述を追加した例である．クラス図の下に OCL による制約が記述されている．OCL では，不変表明や事前事後条件を記述することができる．図 2･8 では，左が不変表明，右が事前事後条件である．制約記述には，コンテキスト（context）と呼ばれる，制約を割り当てる要素を指定する．図 2･8 において，左の制約は，クラス A がコンテキストであり，クラス A に関する不変表明であることを示している．図 2･8 の右では，クラス C のメソッド add がコンテキストであり，クラス C のメソッド add に関する制約であることを示している．また，複数の要素にまたがった制約を記述するため，ナビゲーション（navigation）という概念がある．ナビゲーションでは，クラス図の関連に沿ってたどり，クラスなどの要素を参照することができる．図 2･8 の左の記述では，コンテキストはクラス A であるが，二つの関連をたどることにより，クラス C の属性 n を参照している．

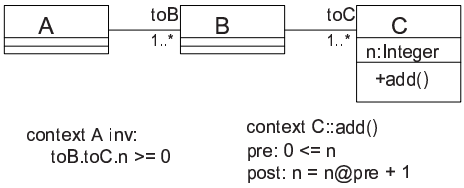


図 2･8 OCL

UML の図に関する動作を記述する言語はアクション言語と呼ばれる．Action Semantics では，アクション言語の記法は定義しておらず，そのメタモデルを規定しているのみである．また，代表的なアクション言語としては，S. Mellor と J. Balcer により提案された Executable UML[8)]で用いられているものがある．図 2･9 に，Executable UML のアクション言語の例を示す．これは，クラス A とクラス B から，それぞれ，オブジェクト aobj と bobj を実体化し，その属性に値を代入後，それらの間に関連 R から実体化されたリンクを張る処理を記述している．そのほかにも，オブジェクトやリンクの選択や削除，値に関する演算などを記述することができる．アクション言語を用いて動作の詳細が記述された UML の図は，実行したり，プログラムの自動生成を行うことができ，そのようなツールも提供されている．

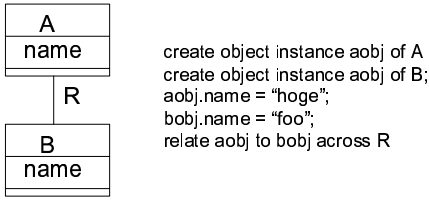


図 2･9 アクション言語

2-5-5 UML 図の検証法

UML では記法面の標準化に注力されており，それにより記述されたものの意味については厳密には定義されていない．そのため，UML の意味を独自に定義し，その意味に基づいた検証法が提案されている場合が多い．

大学の研究者を中心とした有志により，The precise UML group，通称，pUML group が 1997 年に作られた．このグループにより，UML の形式化や意味論に関するワークショップやセミナーが開催され，UML の検証に関する活動が活発になった．モデル検査ツールが整備され始めると，UML のステートチャート図を SPIN や SMV などのツールで検証する手法が提案された．一方，形式的仕様記述や定理証明システムのコミュニティにおいても，それらの手法を用いた UML 図の検証に関する研究が行われた．

UML に基づいて書かれた図の検証は，以下の 4 種類に分類することができる．

- **構文検査**

  UML 図に出現する要素の出現関係など，構文面の性質の検査を行う．異なる図の間の構文的な整合性の検査などが行われる．

- **モデル検査ツールによる検証**

  UML 図を，SPIN や SMV などのモデル検査ツールで取り扱う．ステートチャート図やコラボレーション図などにより記述された振る舞いの検証を行う．

- **形式仕様記述への変換による検証**

  UML 図を，Z や B などによる形式仕様記述に変換し，検証を行う．検証は，それぞれの形式仕様記述言語用に提供されている方法やツールで行う．

- **定理証明システムによる検証**

  UML 図を，PVS や HOL などの定理証明システムにより取り扱うことにより，検証する．

2-5-6 構文検査

UML では，複数の図を記述することが多いため，それらの間の構文面の整合性を保証するのが望ましい．このような整合性の検査は，それぞれの図の意味論にまでは踏み込まず，構文的な規則や制約に合致しているかどうかを調べるのが目的である．よって，それぞれの図に出現する記述要素間の関係や，複数の図の間にどのような関係があれば整合しているかなどの性質は，UML の記法に基づいて構文的に与える必要がある．

文献 9) では，文字列に基づいてそれぞれの図に出現する記述要素を対応づける．そして，図の間で成立すべき構文的な性質が与えられており，記述した複数の図がそれらに合致しているかどうか検査を行う．また，UML の複数の図で記述する場合，異なる文字列で同じ物を表現する場合がある．このような場合に対処するため，ステレオタイプにより要素間の対応づけを導入した方法も提案されている[10)]．

また，UML の前身である，OMT 法[12)]に基づいてクラス図，ステートチャート図，データフロー図を対応づけ，それらの間の整合性を保証する手法も提案されている[13)]．この手法では，それぞれの図を独立に集合に基づいて形式化を行い，それらの間を写像により対応づけている．そして，クラス図とステートチャート図により表現されているデータの流れが，データフロー図で記述されているかどうかを，構文的に検査する．

これらの手法は，図の間の整合性はあらかじめ決められたものであるが，対象システムやシステム開発ごとに，想定する構文的な整合性が異なる場合がある．UML では，記述される図のメタモデルも定義されている．すなわち，記述される図は，メタモデルに基づいて実体化されたものである．よって，このメタモデルに対して制約を与えることにより，UML の図の間の整合性を柔軟に定義することができる．文献 11) では，図の間の整合性を，OCL で記述したメタモデルに対する制約で定義し，それに基づいて記述された図の検査を行う手法が提案されている．これにより，構文的な整合性の定義を柔軟に変更することができる．

2-5-7 モデル検査ツールによる検証

UML では，対象システムの振る舞いを，ステートチャート図やコラボレーション図により記述する．そこで，記述した振る舞いが，意図したとおりになっているかどうか，UML による記述の段階で検証することが考えられる．そこで，このような振る舞いに関する記述を，モデル検査ツールにより検査する手法が提案されている．

文献 17) では，モデル検査ツールで取り扱うために，まず，UML のステートチャート図に操作的意味論を与えている．そして，その意味論に基づいて，モデル検査ツール SPIN[14] の入力言語である Promela に変換する手法と，vUML と呼ばれるツールを提案している．文献 19, 20, 21) では，ステートチャート図を，拡張階層オートマトン（Extended Hierarchical Automata）と呼ばれる枠組みを用いて意味づけを行い，Promela に変換する手法を提案している．文献 24) では，動作の詳細などを Promela により記述し，クラス図やステートチャート図に埋め込む．そして，埋め込まれた記述やそれぞれの図の意味を考慮して，Promela 記述を生成し，SPIN により検査を行う．

HUGO[22] と呼ばれるシステムでは，UML のステートチャート図の振る舞いをモデル検査ツール SPIN により検査する．この手法では，ステートチャート図で書かれた振る舞いが，コラボレーション図で記述した期待する動作と整合するかどうかを検査する．ステートチャート図で表現されている振る舞いを Promela に変換し，コラボレーション図で記述された期待する動作を never claim（Büchi オートマトン）と呼ばれる性質オートマトンに変換する．そして，それらを SPIN を用いて自動的に検査する．また，時間つきステートチャート図を UPPAAL[16] で取扱い可能な形式に変換し，時間を伴う性質の検査を行う，HUGO/RT[23] と呼ばれるツールも提案されている．

モデル検査ツール SMV[15] を用いてステートチャート図の検査を行う手法も提案されている[25]．これは，STP 手法と呼ばれ，STATEMATE[2] のステートチャート図を対象として，SMV で取り扱い可能な形式に変換するものである．また，文献 26) では，ワークフローの検証を目的として，UML のアクティビティ図を NuSMV で取扱い可能な形式に変換し，検査する手法について提案している．

これらの手法は，ステートチャート図やアクティビティ図に記述された状態遷移やイベントに関する性質を検査するものである．ステートチャート図にはアクション言語などを用いて詳細な動作を記述することができるが，そのような動作により生じるデータフローに関する検査を行う手法も提案されている[27]．この手法では，まず，ステートチャート図で書かれた振る舞いにより生じるデータの流れに関する意味論を定義している．そして，その意味論に基づいてステートチャート図をデータの流れを表現する Promela に変換する．また，データフローに関する性質を LTL 式で表現し，SPIN を用いて検査する．

2-5-8 形式仕様記述への変換

形式的仕様記述では，集合や関数など数学的に裏づけのある概念に基づいた言語を用いて厳密に仕様を記述する．Z，VDM, B などが代表的な手法である．このような言語を用いると，記述した内容に対して様々な性質を数学的に証明することができ，それを支援する検証ツールも提案されている．そこで，UML による記述を形式仕様記述言語による記述に変換すれば，その検証ツールを用いて UML の記述の検証を行うことができる．

文献 28) では，UML 図のエディタである Rose を用いて書かれたクラス図に，Z 記法で注釈を記述すると，それらに基づいて Z のドキュメントへ変換する．獲得されたドキュメントは，Z に関する支援ツールを適用することが可能であり，Z/EVES などの定理証明システムを用いて検証することができる．この手法の対象はクラス図のみであり，ステートチャート図などの図は扱うことができない．よって，メソッドや属性に関する機能的な性質が検証対象である．また，OCL やアクション言語に関しても取り扱ってはいない．

文献 29, 30, 31) では，UML で書かれた記述を B メソッドの記述に変換する手法が提案されている．これらの手法も，B メソッドの記述に変換することにより，B メソッドの検証手法や検証ツールを適用することが可能になる．文献 29, 30) では，クラス図のみが対象であり，不変表明や，メソッドの詳細を OCL の事前/事後条件で宣言的に記述する．アクション言語については取り扱っていない．文献 31) では，ステートチャート図をメソッド呼出しの制御を簡便に記述するために用いている．状態を表現する属性を導入し，その属性の値によって呼び出すメソッドを選択するのである．いわゆる，条件文のシンタックスシュガーである．また，OCL による事前/事後条件，不変表明の記述に加えて，独自に定義した簡単なアクション言語により動作を記述することもできる．

集合と一階述語論理に基づいた制約群が充足可能であるかどうか自動的に調べるツールとして，Alloy[32)]が提案されている．UML のクラス図では，多重度を記述したり，OCL で制約を記述するが，それらから実体化された内容は，集合と一階述語論理で特徴づけることができる．そこで，Alloy を用いると，クラス図と OCL による記述を自動的に解析することができる．

2-5-9 定理証明システムによる検証

UML，OCL，アクション言語により記述されたドキュメントには，算術演算，集合演算，論理演算などが含まれる．そこで，これらの演算を考慮した性質を，定理証明システムを用いて検証する手法が提案されている．

文献 33) では，UML の前身である OMT 法で提案されているオブジェクト図，ステートチャート図を用いて土星探査機 Cassini の故障対処ソフトウェアを記述し，それを定理証明システム PVS の記述と対応づけ，安全性にかかわる性質の検証を行った．PVS による検証結果として，いくらかの誤りが発見できたと報告されている．これは特定の事例に関して，定理証明システムを適用したというものである．

文献 34) では，UML のステートチャート図の意味論を PVS により記述している．この PVS による意味記述は，一般的なステートチャート図に関するものである．例えば，状態や状態遷移は，集合として取り扱われており，特定のステートチャート図とは独立である．そこで，この意味記述に合うように，特定のシステムのステートチャート図を PVS の記述に変換することにより，PVS でその検証を行うことができる．そのステートチャート図に出現する状態や状態遷移を，意味記述における状態集合や状態遷移集合の要素として記述するのである．文献 35) では，UML のシーケンス図の意味論を PVS により記述している．この手法も，文献 34) と同様に，一般的なシーケンス図の意味を記述している．そして，その意味記述に合うように，特定のシーケンス図を PVS の記述に変換することにより，検証を行うことができる．

文献 36) では，クラス図と OCL の意味論を Isabelle/HOL を用いて定義している．この手法では，文献 34, 35) とは異なり，特定のクラス図と OCL を Isabelle/HOL の記述に変換する方法を提案している．このような取り扱いのことは，浅い埋め込み（shallow embedding）と呼ばれる．一方，文献 34, 35) は，PVS の論理体系のなかで，一般的なステートチャート図やシーケンス図に対して意味論を定義しており，深い埋め込み（deep embedding）と呼ばれる．また，文献 36) の手法では，Isabelle/HOL の記述に変換する際，それらを取り扱うための様々なデータ型や定理などが必要となる．これらに関しては，Isabelle/HOL がもつ理論を用いて導出している．このことを，保守的埋め込み（conservative embedding）と呼ぶ．

文献 37) では，クラス図とステートチャート図により記述された動作を，定理証明システム HOL により検証する手法が提案されている．この手法では，クラス図におけるメソッドと属性に対して，HOL の記法で注釈を記述する．ステートチャート図には，状態遷移にメソッドと属性を用いた属性評価式を，状態には，表明を記述する．そして，それらの記述から，表明が成立するかどうか証明するための公理系を HOL の理論として生成する．文献 38) では，クラス図に，アクションセマンティクスの一部に相当するアクション言語を用いてメソッドの詳細を記述する．また，それらの動作を検証するために，想定するメソッドの呼出し列をステートチャート図とアクション言語を用いて記述し，期待する性質を OCL による表明として記述する．そして，それらの記述から，想定する呼出し列において期待したとおりの表明が成立するための検証条件を，HOL で取り扱うことができる形式で生成する手法が提案されている．この手法では，アクション言語で記述された部分には表示的意味論が与えられており，その意味関数が，そのまま，HOL の関数として実現される．そのため，アクション言語の部分には中間的な表明を割り当てる必要がなく，効率的に検証を行うことができる．

文献 39) では，クラス図とコラボレーション図を対象としている．クラス図のメソッドと属性に対して，HOL の記法で注釈を記述する．コラボレーション図は複数作成することができ，メソッドと属性を用いた動作と，不変表明を HOL の記法で記述する．ここでの検証の目的は，複数のコラボレーション図がどのような順番で実行されても，不変表明が成立することを証明することである．そこで，これらの記述から，不変表明が成立することを証明できる HOL の理論を生成する．また，この手法では，クラスやオブジェクトといったオブジェクト指向の一般的な概念を，HOL の理論として保守的に構築している．生成される公理系は，その理論を用いて実現されている．

■参考文献

1) D. Harel, et.al., “On the Formal Semantics of Statecharts,” Proc. of 2nd IEEE Sympo on Logic in Computer Science, pp.54-64, 1987.
2) D. Harel, et.al., “STATEMATE: A Working Environment for the Development of Complex Reactive Systems,” Trans. Softw. Eng., vol.16, no.4, pp.403-414, 1990.
3) M. Beeck, “A Comparison of Statecharts Variants,” LNCS 863, pp.128-148, Springer, 1994.
4) A. Kleppe, et.al., “MDA Exmapined,” Addison-Wesley, 2003.
5) J. Warmer and A. Kleppe, “The Object Constraint Language Second Edition,” Addison-Wesley, 2003.
6) OMG: The Unified Modeling Language, `http://www.uml.org/`.
7) OMG: UML Action Semantics, ptc/2002-01-09.
8) S. Mellor and M. Balcer, “Executable UML - A Foundation for Model-Driven Architecture, Addison-Wesley, 2002.

9) 大西淳, “UML におけるモデル整合性検証支援システム,” 信学論, vol.J84-D-1, no.6, pp.671-681, 2001.
10) 鷲見毅, 渡辺晴美, 大西淳, “ステレオタイプによる UML モデル間の整合性検証支援手法,” 情処学論, vol.43, no.6, pp.1554-1562, 2002.
11) D. Chiorean, et.al., “Ensuring UML models consistency using the OCL environment,” UML 2003 Workshop: OCL2.0 Industry Standard or Scientific Playground?, pp.99-110, 2003.
12) J. Ramgaugh, et.al., “Object-Oriented moodeling and design, Prentice-Hall,” 1991.
13) 青木利晃, 片山卓也, “オブジェクト指向方法論のための形式的モデル,” 日本ソフトウェア科学会学会誌 コンピュータソフトウェア, vol.16, no.1, pp.12-32, 1999.
14) G.J. Holzmann, “The SPIN Model Checker,” Addison-Wesley, 2004.
15) E.M. Clarke, et.al., “Model Checking,” MIT Press, 1999.
16) J. Bengtsson, et.al., “UPPAAL-a tool suite for automatic verification of real-time systems,” Hybrid Systems, pp.232-243, 1995.
17) I. Paltor and J. Lilius, “Formalising UML State Machines for Model Checking,” UML 1999, pp.430-445, 1999.
18) J. Lilius and I.P. Paltor, “vUML: a Tool for Verifying UML Models, Automated Software Engineering,” pp.255-258, 1999.
19) D. Latella, et.al., “Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker,” Formal Aspects of Computing, vol.11, no.6, pp.637-664, 1999.
20) E. Mikk, Y. Lakhnech, and M. Siegel, “Hierarchical Automata as Model for Statecharts,” Asian Comput. Sci. Conference Adv. Comput. Sci., pp.181-196, 1997.
21) E. Mikk, Y. Lakhnech, M. Siegel, and G. Holzmann, “Implementing Statecharts in Promela/SPIN,” WIFT’98, 1998.
22) T. Schafer, A. Knapp, and S. Merz, “Model Checking UML State Machines and Collaborations,” Electronic Notes in Theoretical Computer Science, vol.47, 2001.
23) A. Knapp, S. Merz, and C. Rauh, “Model Checking Timed UML State Machines and Collaborations,” Formal Techniques in Real-Time and Fault-Tolerant Systems, pp.395-414, 2002.
24) 岸知二, 野田夏子, “組込みソフトウェアのための UML 設計検証支援環境,” 情報処理学会 組込みソフトウェアシンポジウム, pp.50-57, 2006.
25) E.M. Clarke and W. Heinle, “Modular Translation of Statecharts to SMV,” Technical Report CMU-CS-00-XXX, Carnegie Mellon University School of Computer Science, 2000.
26) R. Eshuis and R. Wieringa, “Tool Support for Verifying UML Activity Diagrams,” IEEE Trans. Softw. Eng., vol.30, no.7, pp.437-447, 2004.
27) 青木利晃, 片山卓也, “オブジェクト指向分析モデルにおけるデータフローの形式化と解析手法,” 日本ソフトウェア科学会 学会誌 コンピュータソフトウェア, vol.21, no.4, pp.1-26, July, 2004.
28) S. Dupuy, et.al., “An Overview of RoZ: A Tool for Integrating UML and Z Specifications,” CAiSE 2000, pp.417-430, 2000.
29) R. Marcano and L. Levy, “Using B formal specifications for analysis and verification of UML/OCL models,” Consistency Problems in UML-based Software Development, pp.91-105, 2002.
30) K. Lano, D. Clark, and K. Andoutsopoulos, “UML to B: Formal Verification of Object-Oriented Models,” IFM, pp.187-256, 2004.
31) C. Snook and M. Butler, “Using a graphical design tool for formal specification,” Wokshop of the Psychology of Programming Interest Group, pp.311-321, 2001.
32) D. Jackson, “Alloy: a lightweight object modelling notation,” ACM Trans. Softw. Eng. Methodology, vol.11, no.2, pp.256-290, 2002.
33) 安保洋子, R. Lutz, “形式的検証手法を用いたソフトウェア安全性解析の実施と評価,” ソフトウェア工学の基礎ワークショップ, pp.165-170, 1995.

34) I. Traore, “An Outline of PVS Semantics for UML Statecharts,” J. Univ. Comput. Sci., vol.6, no.11, pp.1088-1108, 2000.
35) D.B. Aredo, “A Fremework for Semantics of UML Sequence Diagrams in PVS,” J. Univ. Comput. Sci., vol.8, no.7, pp.674-697, 2002.
36) A.D. Brucker and B. Wolff, “A Proposal for a Formal OCL Semantics in Isabelle/HOL,” TPHOLs 2002, pp.99-114, 2002.
37) 青木利晃, 立石孝彰, 片山卓也, “定理証明技術のオブジェクト指向分析への適用,” 日本ソフトウェア科学会学会誌 コンピュータソフトウェア, vol.18, no.4, pp.18-47, 2001.
38) 青木利晃, 片山卓也, “ステートチャートに基づいたオブジェクト指向設計モデルの検証,” ソフトウェア工学の基礎ワークショップ, pp.55-64, 2005.
39) 矢竹健朗, 青木利晃, 片山卓也, “コラボレーションに基づくオブジェクト指向モデルの検証,” 日本ソフトウェア科学会 学会誌 コンピュータソフトウェア, vol.22, no.1, pp.58-76, 2005.