■7 群（コンピュータ-ソフトウェア）- 1 編（ソフトウェア基礎）

1 章 計算モデル

（執筆者：酒井正彦）［2009 年 5 月 受領］

■概要■

計算モデルとは，コンピュータによる計算機構を抽象的に捉えた数学的な枠組みであり，これにより計算に関する様々な議論が可能になる．

最初の計算モデルは，計算機が発明される以前の 1930 年代にチューリング（Turing）によって考案されたチューリング機械である．このようにコンピュータを抽象的な機械として捉え，計算をその機械上の操作と考えるモデルは，抽象機械計算モデルと呼ばれる．その後，より洗練された計算モデルとして，関数もデータとして扱えるラムダ計算に代表される関数型計算モデル，計算過程を書換えとして捉える項書換え系に代表される書換え型計算モデル，計算を論理式の証明過程として捉える論理型計算モデルが提案された．これらの計算モデルは，逐次的な処理を捉えることができるため，逐次的計算モデルと呼ばれる．

逐次的計算モデルでは，与えられた入力に対して計算が停止しないと出力が決まらないため，計算の停止性，すなわち計算が止まることは重要な概念である．これに対して，複数の計算が同時並列的に実行される並行計算では，停止せずに動き続けることが，より重要である．

並行計算のモデルとしては，1960 年代にペトリ（Petri）によって考案されたペトリネットが始まりと考えられる．ペトリネットは計算などの事象の依存関係のみに注目したモデルであったが，その後，洗練されたモデルとして，ホーア（Hoare）の CSP などのように，プロセス間の通信に注目したプロセス代数モデルが研究されている．

チューリング機械のような計算モデルの考案により，計算の限界を明らかにする計算可能性の理論などの理論的研究が発展した．また，計算モデル理論の研究が進むにつれ，プログラムの等価性・計算の一意性・動作検証，逐次的計算における停止性や並行計算においてデッドロックしない（行き詰まりに落ち入らない）ことなどのプログラムの性質を示すことも十分ではないが可能になってきた．

ほとんどの計算モデルでは，操作の定義を与える操作的意味論，等式から定義される代数として捉える代数的意味論，領域方程式の解として捉える表示的意味論，公理と推論規則によって意味を定める公理的意味論などのうち，一つ以上の方法により，プログラムの意味が定められるものが多い．これらの意味論に基づいたプログラムの形式仕様記述法が提案されている．

【本章の構成】

1-1 節では計算モデルにおける共通の言語構造であるばかりでなく，それらの性質を示す道具としても有効な木言語について述べる．1-2 節と 1-3 節は，それぞれ，並行計算のモデルである通信プロセス計算，並びに，関数型言語や代数的仕様記述法の基礎となる項書換え系について説明する．最後に，1-4 節では，代数に基づいてソフトウェアを仕様記述するための代数仕様を解説する．

■7 群 - 1 編 - 1 章

1-1 木言語

（執筆者：石原靖哲）[2008 年 7 月 受領]

木言語とは木の集合である．木とは，入射枝をもたない唯一の頂点（根と呼ぶ）をもち，かつ根以外のすべての頂点が入射枝をちょうど 1 本ずつもつ，有向無閉路グラフである．出射枝をもたない頂点を葉と呼ぶ．木の各枝の始点 u と終点 v に対し，u を v の親と呼び，v を u の子と呼ぶ．また，異なる 2 頂点 u，v に対し，u から v へ経路が存在するとき，u を v の先祖といい，v を u の子孫という．根から頂点 v への経路の長さ（すなわち，その経路に含まれる枝の本数）を v の深さといい，木 t に含まれるすべての頂点についての深さの最大値を t の高さという．

木の各頂点について，その子の集合上になんらかの全順序が指定されているとき，その木を順序木と呼ぶ．順序木の頂点集合からラベル集合への写像が指定されているとき，その順序木をラベルつき順序木と呼ぶ．本節では以降，ラベルつき順序木を単に木と呼ぶ．

1-1-1 有限木からなる木言語

有限個の頂点からなる木を有限木と呼ぶ．有限木は以下のように自然に項で表現できる．ラベル a をもつ頂点 v が n 個の子 $v_1,\ldots,v_n$ $(n \geq 0)$ をこの順にもち，かつ $v_1,\ldots,v_n$ を根とする木の項表現がそれぞれ $t_1,\ldots,t_n$ であるとすると，v を根とする木の項表現は $a(t_1,\ldots,t_n)$ である．$n = 0$ の場合，しばしば $a()$ の代わりに単に a と書く．以降，有限木と項を同一視して記述する．

(1) ランクあり木言語

Σ をランクつきアルファベットとする．すなわち，Σ は（ラベルの）有限集合であり，各要素 $a \in \Sigma$ にはランクと呼ばれる非負整数 $r(a)$ が指定されているとする．木の各頂点 v について，v の子の個数が v のラベルのランクと等しいとき，その木をランクあり木（ranked tree）と呼ぶ．Σ 上のすべてのランクあり木の集合を $\mathcal{T}(\Sigma)$ で表す．また，$X \cap \Sigma = \emptyset$ を満たす記号集合 X に対し，$\mathcal{T}(\Sigma)$ 中の木の任意の 0 個以上の葉のラベルを X 中の記号で置き換えて得られる木全体の集合を $\mathcal{T}(\Sigma, X)$ で表す．

ランクあり木は計算機科学における極めて重要なモデルの一つである．ランクが 0 のラベルはなんらかのデータを表しており，ランクが 1 以上のラベルはデータに対する演算を表しているとすると，ランクあり木はデータに対する一連の演算の手順もしくはその結果を表していると自然に解釈できるからである．

例 1 $\Sigma_0 = \{0, \mathsf{s}\}$ とし，$r(0) = 0$，$r(\mathsf{s}) = 1$ とする．0 が 0 という値を表し，s が引数の値に 1 を加えるという演算を表しているとすると，$\mathcal{T}(\Sigma_0) = \{0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots\}$ は非負整数全体の集合を表していると解釈できる．更に，$\Sigma_1 = \Sigma_0 \cup \{\mathsf{add}\}$ とし，$r(\mathsf{add}) = 2$ とする．add は非負整数上の加算を表しているとすると，$\mathcal{T}(\Sigma_1)$ は非負整数と加算演算子から構成できる任意の計算式（あるいはその計算結果）の集合であると解釈できる．

一般には，演算はどんなデータにでも適用できるわけではない．通常はデータ型を導入してデータの分類や演算の定義域の指定を行うことにより，可能な計算式の集合（すなわちラン

クあり木言語）に制限を加える．正規木言語（regular tree language）は，このような制限が加えられた木言語のクラスの一つであり，理論面と実用面の両方において重要なクラスである．以下では，正規木言語を指定するための文法やオートマトンを定義したのち，正規木言語の基本的な性質をいくつか紹介する．より詳しくは Comon らの著書[1]や，Gécseg と Steinby による解説[2]などを参照されたい．

(a) 正規木文法

まず正規木文法を定義する．

定義 2 ランクあり正規木文法（regular tree grammar）G は 4 項組 $(N, \Sigma, \hat{N}, P)$ で定義される．ここで，N は非終端記号の有限集合であり，$\hat{N} \subseteq N$ は開始記号の集合である．また，$A \in N$，$a \in \Sigma$ とし，W を長さ $r(a)$ の N 上の系列とすると，P は $A \to a(W)$ のかたちをした生成規則の有限集合である．

ランクあり正規木文法 $G = (N, \Sigma, \hat{N}, P)$ に対して，$\mathcal{T}(\Sigma, N)$ 上の導出関係を次のように定義する．$t \in \mathcal{T}(\Sigma, N)$ のある葉 v のラベルが $A \in N$ であるとし，生成規則 $A \to a(W)$ が P に存在するとする．t の葉 v を $a(W)$ に置き換えた木を $t' \in \mathcal{T}(\Sigma, N)$ とおく．このとき，G において t から t' が導出されるといい，$t \underset{G}{\Rightarrow} t'$ と書く．関係 $\underset{G}{\Rightarrow}$ の反射推移閉包を $\overset{*}{\underset{G}{\Rightarrow}}$ で表すと，G によって生成される木言語 $TL(G)$ は以下の式で定義される．

$$TL(G) = \{t \in \mathcal{T}(\Sigma) \mid S \overset{*}{\underset{G}{\Rightarrow}} t,\ S \in \hat{N}\}.$$

正規木文法によって生成される木言語を正規木言語と呼ぶ．

生成規則の右辺に $\mathcal{T}(\Sigma, N)$ 中の任意の項を書けるように拡張しても，生成能力は変わらない．

例 3 Σ_1 を例 1 と同様に定義し，正規木文法 $G_1 = (\{\mathsf{Nat}\}, \Sigma_1, \{\mathsf{Nat}\}, P_1)$ の生成規則集合 P_1 を次のように定義する．

$$P_1 = \{\mathsf{Nat} \to 0,\ \mathsf{Nat} \to \mathsf{s(Nat)},\ \mathsf{Nat} \to \mathsf{add(Nat, Nat)}\}.$$

このとき，$TL(G_1) = \mathcal{T}(\Sigma_1)$ である．

次に，G_1 に対して，偶数のみを引数にとることができ，その値を半分にする演算 halve を導入することを考える．$\Sigma_2 = \Sigma_1 \cup \{\mathsf{halve}\}$，$r(\mathsf{halve}) = 1$ とする．P_1 に $\mathsf{Nat} \to \mathsf{halve(Nat)}$ という生成規則を追加するだけでは，halve の引数として偶数しか現れないことを保証できない．そこで，次のような正規木文法 $G_2 = (N_2, \Sigma_2, N_2, P_2)$ を考える．

$$\begin{aligned} N_2 &= \{\mathsf{Nat}, \mathsf{Even}, \mathsf{Odd}\}, \\ P_2 &= P_1 \cup \{\mathsf{Even} \to 0,\ \mathsf{Even} \to \mathsf{s(Odd)},\ \mathsf{Odd} \to \mathsf{s(Even)}, \\ &\qquad \mathsf{Even} \to \mathsf{add(Even, Even)},\ \mathsf{Even} \to \mathsf{add(Odd, Odd)}, \\ &\qquad \mathsf{Odd} \to \mathsf{add(Odd, Even)},\ \mathsf{Odd} \to \mathsf{add(Even, Odd)}, \\ &\qquad \mathsf{Nat} \to \mathsf{halve(Even)}\}. \end{aligned}$$

このとき，

$$\begin{aligned}\mathsf{Nat} &\underset{G_2}{\Rightarrow} \mathsf{halve(Even)} \underset{G_2}{\Rightarrow} \mathsf{halve(add(Odd, Odd))} \\ &\overset{*}{\underset{G_2}{\Rightarrow}} \mathsf{halve(add(s(Even), s(Even)))} \overset{*}{\underset{G_2}{\Rightarrow}} \mathsf{halve(add(s(0), s(0)))}\end{aligned}$$

より $\mathsf{halve(add(s(0), s(0)))} \in TL(G_2)$ である．更に，各頂点に対応し得る非終端記号をボトムアップに決定していくことにより，$\mathsf{halve(add(s(0), 0))} \notin TL(G_2)$ であることも確認できる．ただし，この文法では halve を含む式を別の halve の引数に用いることができない．

局所木文法と呼ばれる正規木文法の部分クラスを与える．競合する非終端記号をもたない正規木文法を局所木文法（local tree grammar）という．ここで，非終端記号 A，B が競合するとは，$A \neq B$ であり，かつ A，B それぞれを左辺にもつ二つの生成規則が同じ終端記号を右辺にもつことをいう．例 3 の G_1 は局所木文法である．一方，G_2 では $\{\mathsf{Nat}, \mathsf{Even}, \mathsf{Odd}\}$ 中の任意の 2 非終端記号が競合するため，G_2 は局所木文法ではない．局所木文法によって生成される木言語を局所木言語（local tree language）という．任意の文脈自由文法について，その文法における構文木の集合が局所木言語であることは容易に確かめられる．

(b) 有限木オートマトン

文字列言語と同様に，オートマトンによって木言語を特徴づけることもできる．有限オートマトンを木言語に拡張した有限木オートマトン（finite tree automaton: FTA）には，入力木を根から葉の方向に走査するトップダウン型と，葉から根の方向に走査するボトムアップ型とがある．本質的には，以下で定義するトップダウン非決定性 FTA は，正規木文法の非終端記号を状態に，開始記号を初期状態に，生成規則を遷移規則に読み換えたものにほかならない．したがって，トップダウン非決定性 FTA が認識できる木言語のクラスは正規木言語のクラスに一致する．

定義 4 ランクありトップダウン非決定性有限木オートマトン M は 4 項組 $(Q, \Sigma, \hat{Q}, \delta)$ で定義される．ここで，Q は状態の有限集合であり，$\hat{Q} \subseteq Q$ は初期状態の集合である．また，$q \in Q$，$a \in \Sigma$ とし，W を長さ $r(a)$ の Q 上の系列とすると，δ は (q, a, W) のかたちをした遷移規則の有限集合である．

以下，M の遷移を定義する．Q 中の各状態をランクが 1 のラベルとみなすことにする．$\mathcal{T}(\Sigma \cup Q)$ 中の木を M の時点表示と呼ぶ．時点表示 s のある部分木が $q(a(t_1, \ldots, t_{r(a)}))$（$q \in Q$，$a \in \Sigma$, $t_1, \ldots, t_{r(a)} \in \mathcal{T}(\Sigma)$）のかたちをしており，かつ遷移規則 $(q, a, q_1 \cdots q_{r(a)})$ が δ に存在するとする．s のその部分木を $a(q_1(t_1), \ldots, q_{r(a)}(t_{r(a)}))$ に置き換えて得られる時点表示を s' とおくと，M は s から s' に遷移可能であると定義する．M に対して木 $t \in \mathcal{T}(\Sigma)$ が入力されたとき，ある $\hat{q} \in \hat{Q}$ が存在して M の初期時点表示 $\hat{q}(t)$ から時点表示 t に遷移可能ならば，M は t を受理するという．M が受理する木すべてからなる木言語を，M が認識する木言語と呼ぶ．

ボトムアップ非決定性 FTA は，構文的にはトップダウン型と同じであるが，意味的にはトップダウン型とはちょうど逆向きの遷移をするオートマトンとして定義される．したがって，ボトムアップ非決定性 FTA が認識できる木言語のクラスは正規木言語のクラスに等しい．

$M = (Q, \Sigma, \hat{Q}, \delta)$ とする．$\hat{Q}$ が空集合のとき，あるいは $\hat{Q}$ の要素数が 1 でかつ任意の遷移

規則の対 $(q,a,W),(q,a,W')\in\delta$ について $W=W'$ のとき，M はトップダウン決定性であるという．また，任意の遷移規則の対 $(q,a,W),(q',a,W)\in\delta$ について $q=q'$ ならば，M はボトムアップ決定性であるという．任意のボトムアップ非決定性 FTA に対し，それと等価なボトムアップ決定性 FTA を構成できる．一方，トップダウン決定性 FTA が認識できる木言語のクラスは，トップダウン非決定性 FTA のそれと比べて真に小さい．

任意のボトムアップ決定性 FTA に対し，それと等価（すなわち，認識する木言語が同じ）で状態数が最小のボトムアップ決定性 FTA を求める手続きが知られている．また，任意の正規木言語に対し，それを認識する状態数最小のボトムアップ決定性 FTA は，状態名のつけかえのもとで同型である．

(c) 単項二階述語論理

木言語を論理式によって指定することもできる．すなわち，木の性質を表す論理式 ϕ に対し，ϕ が指定する木言語 $TL(\phi)$ を，ϕ を満たす木の集合と定義する．ここで，単項二階述語論理（monadic second order logic）の一体系 $\mathcal{L}_2(\Sigma)$[2]を以下のように定義する．このとき，$\mathcal{L}_2(\Sigma)$ の閉論理式で指定できる木言語のクラス，すなわち

$$\{TL(\phi)\mid \phi \text{ は } \mathcal{L}_2(\Sigma) \text{ の閉論理式 }\}$$

は，正規木言語のクラスと一致することが知られている．

定義 5 n を Σ 中の記号のランクの最大値とする．$\mathcal{L}_2(\Sigma)$ の語彙は以下のとおりである．

- 頂点を値として取る変数 $u, v,\ldots$，及び頂点集合を値としてとる変数 $X, Y,\ldots$．
- 1 引数述語記号 P_a（$a\in\Sigma$）．$P_a(u)$ は u のラベルが a であることを表す．
- 2 引数述語記号 $<, \sigma_1,\ldots,\sigma_n, =, \in$．$u<v$ は頂点 u が頂点 v の先祖であることを表す．$u\sigma_i v$ は v が u の i 番目の子であることを表す．$u=v$ は u と v が同じ頂点であることを表す．$u\in X$ は u が X の要素であることを表す．
- 論理演算子 $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.
- 限量子 $\forall, \exists$. これらは頂点を値として取る変数にも頂点集合を値としてとる変数にも適用可能である．

$\mathcal{L}_2(\Sigma)$ の論理式は，これらの語彙に基づいて通常のとおり定義される．自由変数，束縛変数，閉論理式も通常のとおり定義される．

(d) 正規木言語の性質

正規木言語のクラスは和集合，積集合，補集合演算のもとで閉じている．これらは，文字列言語の場合と同様に，対応するオートマトンを構成できることから証明できる．また，線形な準同型写像[1, 2]や逆準同型写像[1, 2]に関しても閉じていることが知られている．

正規木言語に対してポンピング定理が成り立つ．その準備として文脈を次のように定義する．$\mathcal{T}(\Sigma)$ 中の木のちょうど一つの葉のラベルを記号 $\square$ に置き換えた木を Σ 上の文脈（context）

と呼ぶ．Σ 上の文脈全体の集合を $C(\Sigma)$ と書く．木 $t \in \mathcal{T}(\Sigma)$ と文脈 $p \in C(\Sigma)$ に対し，p 中のラベル $\square$ をもつ頂点を t に置き換えて得られる木を tp と書く．更に，$tp^0 = t$ と定義し，任意の $k \geq 1$ について $tp^k = tp^{k-1}p$ と定義する．

定理 6 (ポンピング定理) 任意の正規木言語 TL に対してある正整数 $n > 0$ が存在し，高さが n 以上の任意の $t \in TL$ について次の条件を満たす $s \in \mathcal{T}(\Sigma)$ と $p, q \in C(\Sigma)$ が存在する．

- $t = spq$.
- p の高さは 1 以上.
- 任意の $k \geq 0$ について $sp^kq \in TL$.

ポンピング定理は，与えられた木言語が正規木言語ではないことを示すのにしばしば有用である．例えば，木言語 $\{a(t,t) \mid a \in \Sigma, t \in \mathcal{T}(\Sigma)\}$ が正規木言語ではないことを，ポンピング定理により容易に示せる．また，与えられた有限木オートマトンの認識する言語が有限か無限かを判定する問題が決定可能であることを，ポンピング定理から導くことができる．

(2) ランクなし木言語

ランクなし木（unranked tree）とは，ラベルと，そのラベルがつけられた頂点の子の個数との間に制約がない木である．以下，Γ をランクの指定がないアルファベットとする．Γ 上のすべてのランクなし木の集合を $\mathcal{T}(\Gamma)$ で表す．また，$X \cap \Gamma = \emptyset$ を満たす記号集合 X に対し，$\mathcal{T}(\Gamma)$ 中の木の任意の 0 個以上の葉のラベルを X 中の記号で置き換えて得られる木全体の集合を $\mathcal{T}(\Gamma, X)$ で表す．

しばしば，ランクなし木は XML 文書のモデルとして用いられ，ランクなし木言語を指定するための文法は XML 文書のスキーマのモデルとして用いられる[3]．XML への応用においては，以下で定義する正規木文法とそのいくつかの部分クラスが重要である[4]．正規木文法は RELAX NG[5]というスキーマ記述言語に相当する．

定義 7 ランクなし正規木文法 G は 4 項組 $(N, \Gamma, \hat{N}, P)$ で定義される．ここで，N は非終端記号の有限集合であり，$\hat{N} \subseteq N$ は開始記号の集合である．また，$A \in N$，$a \in \Gamma$ とし，R を N 上の正規表現とすると，P は $A \to a(R)$ のかたちをした生成規則の有限集合である．

ランクなし正規木文法 $G = (N, \Gamma, \hat{N}, P)$ に対する導出関係 $\underset{G}{\Rightarrow}$ を次のように定義する．以下では正規表現 R が表す文字列言語を $L(R)$ と書く．$t \in \mathcal{T}(\Gamma, N)$ のある葉 v のラベルが $A \in N$ であるとし，生成規則 $A \to a(R)$ が P に存在するとする．t の葉 v を $a(W)$（ただし W は $L(R)$ 中の任意の系列）に置き換えた木を $t' \in \mathcal{T}(\Gamma, N)$ とおくと，$t \underset{G}{\Rightarrow} t'$ が成り立つ．G によって生成される木言語 $TL(G)$ はランクありの場合と同様に定義される．

ランクなし正規木文法においては，一般性を失うことなく，任意の生成規則の対 $A \to a(R)$, $A \to a(R') \in P$ について $R = R'$ であると仮定できる．なぜなら，$A \to a(R_1)$ と $A \to a(R_2)$ という二つの異なる生成規則は，$A \to a(R_1|R_2)$ という一つの生成規則にまとめることができるからである．

ランクありのときと同様，競合する非終端記号をもたない正規木文法を局所木文法という．

局所木文法は DTD というスキーマ記述言語にほぼ相当する．また，各生成規則 $A \to a(R)$ について，互いに競合する非終端記号が同時には R に含まれないような正規木文法を，単一型木文法（single-type tree grammar）という．単一型木文法は W3C XML Schema[6] というスキーマ記述言語にほぼ相当する．単一型木文法では，根を除く任意の頂点 v について，v に対応する非終端記号が，v のラベル及び v の親 v' のラベルと，v' に対応する非終端記号とから一意に定まる．従って，与えられた XML 文書がスキーマに従っているかを，その文書をトップダウンに一度走査することで検査できるという好ましい性質をもつ．局所木文法やトップダウン決定性の正規木文法（ランクあり有限木オートマトンのときと同様に定義される）もこの性質をもつが，局所木文法は，単一型木文法及びトップダウン決定性の正規木文法の両方の真の部分クラスである．また，単一型木文法とトップダウン決定性の正規木文法とは比較不能の関係にある．

ランクなし正規木言語のクラスも，ランクありの場合と同様に，和集合，積集合，補集合演算のもとで閉じている．また，ポンピング定理も成立する．

なお, 任意のランクなし木はランクがたかだか 2 の木で符号化できる．例えば, $a(b_1, b_2, \ldots, b_n)$ は，ランクが 2 の新たなラベル ω を用いて $a(b_1, \omega(b_2, \omega(\cdots, \omega(b_{n-1}, b_n)\cdots)))$ と表される．ただし，符号化により木の高さが変わることに注意する必要がある．

1-1-2 無限木からなる木言語

無限木とは，一般には無限の頂点からなる木を指すが，特に葉をもたないランクあり木を指すことが多い．本節の最初で述べた木の定義より，木は頂点を一つ以上もつので，葉をもたない木は必然的に無限の頂点をもつ．有限木オートマトンや単項二階述語論理を，ランクあり無限木からなる木言語向けに拡張したものがいくつか知られているが，詳細は Thomas による解説[7] などを参照されたい．なお，ランクありで順序のない無限木も，CTL（computation tree logic）のモデルとして重要である．

■参考文献

1) H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, “Tree Automata Techniques and Applications,” Available on: http://www.grappa.univ-lille3.fr/tata, 2007.
2) F. Gécseg and M. Steinby, “Tree Languages,” in Handbook of Formal Languages, vol.3, pp.1-68, Springer-Verlag, 1997.
3) T. Schwentick, “Automata for XML—A Survey,” J. Comput. Syst. Sci., vol. 73, pp. 289-315, 2007.
4) M. Murata, D. Lee, M. Mani, and K. Kawaguchi, “Taxonomy of XML Schema Languages using Formal Language Theory,” ACM Trans. Internet Technol., vol. 5, no. 4, pp. 660-704, 2005.
5) J. Clark and M. Murata: “RELAX NG Specification,” Available on: http://www.oasis-open.org/committees/relax-ng/spec-20011203.html, 2001.
6) World-Wide Web Consortium: “XML Schema Part 0: Primer Second Edition,” Available on: http://www.w3.org/TR/xmlschema-0/, 2004.
7) W. Thomas, “Languages, Automata, and Logic,” in Handbook of Formal Languages, vol. 3, pp. 389-455, Springer-Verlag, 1997.

■7 群 - 1 編 - 1 章

1-2 通信プロセス計算

（執筆者：結縁祥治）[2008 年 8 月 受領]

1-2-1 はじめに

本節では，並行計算の形式モデルとして**通信プロセス計算**（communicating processes）について説明する．近年，計算機ネットワークの技術の発達，デバイスの高度化，演算ユニットのマルチコア化などから，ソフトウェアにおける並行計算が強く意識されるようになっている．しかし，並行計算に基づくソフトウェアは逐次的に動作するソフトウェアに比べて，一般に複雑で，動作が不安定であることが多い．この主な原因は，複数のスレッドやプロセスが同時並行的に実行されるために，共有変数の更新，通信の応答などのタイミングが組合せ的に複雑になり，実行ごとの振る舞いが逐次プログラムのように一定しないためである．個別の入出力関係の観点で正しく動作する複数のプログラムを組み合わせて設計どおりに動作させるためには，すべてのプログラムの計算途中の状態が意図どおりの組み合わせでなければならない．ところが，入出力関係を基本とする関数に基づいたプログラミング手法では，入出力以外を隠蔽しながら関数合成を基にソフトウェアを構成することが基本であるため，相互に影響を及ぼす途中の状態を解析することが難しい場合が多い．排他制御などのほかのプログラムとの相互作用の制御メカニズムに対する解釈は，プログラム本体とは全く別のものとなる．

実行中の単一の制御によって状態が更新される逐次実行に比べ，並行実行ではほかのプログラムが同時に更新可能な環境に応じて，状態の更新が決定する．実行の途中で想定していない環境が出現すると，ソフトウェアが不具合を起す可能性があるため，実行部分の局所的状態だけでなく，それが実行され得る全体の環境まで含めてテストや検証を行わなければ信頼性を向上させることができない．

（最も単純な）逐次計算では，計算を開始する際に初期値として入力を与え，計算が停止したときの値を計算結果として出力する．このような場合，初期値を入力，最終値を出力とする**関数**が計算の意味となる．合成は出力を別の関数の入力に接続する関数合成である．これに対して，プログラムの実行過程では，一般にメモリや入出力装置に対する**副作用**が発生し，入出力が同じでも計算として区別されることが多い．更に環境に応じた状態の更新は副作用そのものにほかならない．並行プログラムでは，環境に応じて実行過程が変化することは本質的な特徴であり，別のプログラムの実行によって環境が変更される．副作用がプログラミングに及ぼす最も深刻な問題は，複数の実行単位の並行実行による合成に対する振る舞い意味を保存しないことである．計算の際に入出力以外に発生する副作用を入出力のみに着目して合成すると同じ入出力をもつ合成でも結果として違う副作用をもつ場合があり，副作用に応じた意味論の変更が必要になる．このため，構成要素の入出力だけを観測して並行プログラムを構成することは，意味の捉え方としては不十分である．

通信プロセス計算では，並行計算の基本的な合成操作は並行合成である．二つのプログラム P,Q が同時に実行されることによってより大きなプログラム $P|Q$ が合成される．更に P と Q の間の相互作用を，一対一の同期通信を基にして表現する．このような P 及び Q のことを**プロセス**（process）あるいは**動作体**（agent）と呼ぶ．$P|Q$ において，Q は P の実行環境であり，逆に P は Q の実行環境となる．P と Q が並行に動作しながら通信を行うことに

よって，並行プログラムの動的な環境の変化を表現する．

通信プロセス計算では，入出力の対応による関数的な等価性に代えて**双模倣意味論**（bisimulation equivalences）を展開する．双模倣意味論によれば，$P = Q$ ならばすべてのプロセス R に対して $P|R = Q|R$ であることを保証する．この等価性から，並行合成に対する代数を構成する．この代数が代表的な形式的な意味論とされることから，代表的通信プロセスモデルは，しばしば**プロセス代数**（process algebra）と呼ばれる．通信プロセスの代表的な体系である Hoare の CSP[21]や Milner の CCS[24]は，それぞれ，失敗集合意味論，観測合同関係といった，並行合成に対して保存する等価性に対する代数的意味をもつ．プロセスの代数的な意味論に基づいて，並行プログラムに対して構成的な意味づけを与える．

本節の構成は以下のとおりである．まず，通信プロセス計算の基本的な動作意味であるラベルつき遷移システムについて説明し，ラベルつき遷移システムに対する双模倣等価性について述べる．プロセス計算としてよく知られている Milner の CCS に対して，双模倣等価性が CCS の演算子に対して保存される合同関係であることから，プロセス代数が構成されることを述べる．更に，CCS を拡張したモバイル計算として π 計算の概要について述べ，最後にまとめを示す．

1-2-2 ラベルつき遷移システム

ラベルつき遷移システム（Labelled Transition System: LTS）は，通信プロセスモデルの基本的な動作を表現する．形式的には，以下のように定義される．

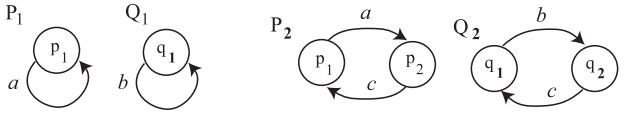
定義 1 (ラベルつき遷移システム) ラベル集合 Act 上のラベルつき遷移システムは，$\langle Q, \mathcal{T} \rangle$ で定義される．ここで，Q は状態の集合，$\mathcal{T} \subseteq Q \times Act \times Q$ は遷移関係である．

$(q_1, a, q_2) \in \mathcal{T}$ は，$q_1 \xrightarrow{a} q_2$ と書くことにする．これは，状態 q_1 からラベル a による通信を行って状態 q_2 に遷移することを表す．LTS は，古典的なオートマトンから初期状態と最終状態を除いたシステムである．この定式化は，通信プロセスモデルの動作の性質を自然に表現しているといえる．なぜなら，通信プロセスでは，環境との通信によってのみ振る舞いが同定できるので，一般には初期状態にあるのか最終状態にあるのかが外部から直接的には分からないためである．

複数のラベルつき遷移システムをラベルの対応づけによって合成することで，並行動作が表現できる．LTS で表されるプロセス P_1 と Q_1 が図 1･1(a) のように定義されているとする．P_1 と Q_1 を合成すると，a と b は並行して実行され，a と b が自由なタイミングで発生する．次に図 1･1(b) のように P_2 と Q_2 を定義する．P_2 と Q_2 とが，共通のラベル c で同期するとすると，P_2 と Q_2 の合成では a と b の発生は制限され，それぞれ同じ回数だけ発生するようになる．

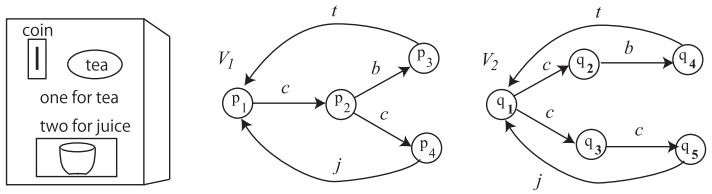
ラベルつき遷移系の並行合成では**インターリーブ**による並行実行を扱う．ここでは，a と b のラベルつき遷移の同時発生については，a と b のラベルつき遷移が，ab の順か ba の順で発生するかが決定できないことに帰着される*．インターリーブに基づく通信プロセスモデルでは，通信が発生する場合のみ同時に発生し，計算が同時に進行するとする．

* 同時発生を発生順序の非決定性と区別する意味づける体系も研究されている[12]．



(a) 非同期的　　(b) 通信ラベル c で同期

図 1･1　プロセスの並行合成



(a) 自動販売機　　(b) 振る舞いV_1　　(c) 振る舞いV_2

図 1･2　自動販売機の振る舞い

1-2-3 双模倣意味論

通信を基本とする並行計算のモデルでは，ラベルつき遷移系において非決定性は本質的な意味をもつ．図 1･2 のような簡単な自動販売機の例を考えてみよう．V_1，V_2 はいずれもコインを入れて，飲み物を販売する．コインを入れることをラベル c の遷移で，ボタンを押すことをラベル b の遷移で，また，お茶を出力することをラベル t の遷移で，更に，ジュースを出力することをラベル j の遷移で表す．V_1，V_2 共に，コイン 1 枚でボタンを押すとお茶，コイン 2 枚でジュースが出る．しかし，V_1 と V_2 で買い物をするお客にとっては，この二つの動作は同じではない．なぜなら，V_1 では，コインを 1 枚入れたあと，更にコインを入れてジュースを買うのか，ボタンを押してお茶を買うのかを選択できるのに対して，V_2 では，最初にコインを入れたときにお茶を買うのかジュースを買うのかが決まってしまう．このように単純な入出力関係ではこの二つの振る舞い区別することはできないため，どういう順序で入出力が行われているかをより詳細に記録しないと区別することができない．

(a) 双模倣関係と双模倣等価性

双模倣関係（bisimulation）は，プロセスにおける状態間の振る舞いが通信の順序に関して同じであることを示すための概念であり，以下のように定義される．

定義 2 (双模倣関係) ラベル集合 Act 上の $LTS\langle Q, \mathcal{T}\rangle$ における Q 上の関係 R が以下の条件を満たすとき，R は双模倣関係（bisimulation）である．

すべての $(p, q) \in Q$，$a \in Act$ に対して以下が成立する．

- $p \xrightarrow{a} p_1$ に対して，q_1 が存在して $q \xrightarrow{a} q_1$ かつ $(p_1, q_1) \in R$
- $q \xrightarrow{a} q_2$ に対して，p_2 が存在して $p \xrightarrow{a} p_2$ かつ $(p_2, q_2) \in R$

ここで，双模倣関係は関係の性質を定義していることに注意しよう．従って，双模倣性を満たす関係は複数存在する．このうち，プロセスの等価性として興味のある双模倣関係は最大の双模倣関係である．最大の双模倣関係は等価関係となることが示される*.

定義 3 (双模倣等価性) 双模倣等価関係（bisimulation equivalence）$\sim$ を以下のように定義する．

$\sim \triangleq \{(p, q); (p, q) \in R$ となる双模倣関係 R が存在する $\}$

命題 4 $\sim$ は等価関係である．

命題 5 $\sim$ は最大の双模倣関係である．

V_1 における状態 p_1 と V_2 における状態 q_1 が双模倣等価でないことは以下のように確かめられる．p_1 は c によって p_2 に遷移する．p_2 からは c と b が可能である．しかし，q_1 から c で遷移する q_2 と q_3 いずれの状態でも c と b の両方が可能となることはない．

双模倣等価性は，余帰納的（co-inductive）に定義される．通信を通じた振る舞いの違いが観測できないときに等価とする．この意味で，双模倣性を満たす関係のうち，最も大きい関係を意味論とすることは自然である．

定義から分かるように双模倣等価性は，次状態への遷移関係が定義されていれば定まる．遷移システムにおいて，もとの遷移システムのラベルつき遷移だけでなく，特定のラベルによる遷移を抽象化することによって得られる遷移に対しても同様に双模倣等価性を定義できる．CCS などの体系では，内部動作を表現する τ 遷移を抽象化して遷移関係を構成する．遷移関係を抽象化すると，より多くの状態に対して等価性をいうことができるようになる．このため，もとの LTS に対して定められる双模倣等価性を**強等価性**，τ などの外部から観測できないラベルを抽象化した関係から導かれる等価性を**弱等価性**と呼ぶことがある．

1-2-4 プロセス代数

通信プロセスモデルを形式的に定義するには，プロセスを表現する項を定め，項の間のラベルつき遷移関係を定義する．ここでは，Milner によって提案された形式体系である CCS（Calculus of Communicating Systems）によるプロセス代数を示す．

(1) CCS の構文

通信ラベル名の集合 Act が与えられているとする．Act の要素 a に対して，相補名 $\overline{a}$ が定義されているとし，$\overline{a} \notin Act$ とする．$Lbl = Act \cup \{\overline{a}; a \in Act\}$ に含まれない特別なラベル τ が存在するとし，$Act_\tau = Lbl \cup \{\tau\}$ で表す．以下，Lbl の要素を λ，Act_τ の要素を α で書き，$\overline{\overline{\lambda}} = \lambda$ とする．プロセス定数の集合を $\mathcal{K}$ とし，各 $A \in \mathcal{K}$ に対して定数式 $A \stackrel{\text{def}}{=} E$ が与えられているとする．

CCS 式 E の構文は以下のように定義される．

* 以下，命題，定理などの証明は文献 24) を参照いただきたい．

$$E ::= \alpha.E \;\; 0 \;\; E+E \;\; E|E \;\; E\backslash\mathcal{L} \;\; E[f] \;\; A$$

ここで，$\alpha \in Act_\tau$，$\mathcal{L} \subseteq Lbl$，$A \in \mathcal{K}$ とする．名前替え演算子 f は，$Act \to Act$ の関数である．名前替え $\{l_1 \mapsto l_2\}$ は，通信ラベル l_1 を l_2 に替え，そのほかの通信ラベルはそのままとする．各式の直感的な意味を説明する．0 は停止したプロセス，$\alpha.E$ は，α による通信でプロセス E となるプロセスを表す†．E_1+E_2 は，通信を行った後に E_1，E_2 のいずれか一方が選択される．$E_1|E_2$ は，E_1 と E_2 が並行に実行され，相補なラベルで遷移が可能な場合は，通信が発生して E_1 と E_2 が同時に状態遷移する．通信が発生しない場合はインターリーブによって並行に実行する．$E\backslash\mathcal{L}$ は，E の遷移のうち，$\mathcal{L}$ に属するラベルをもつ遷移を制限する．ここで，$\tau \notin \mathcal{L}$ であるので，E の内部で発生する通信は制限されないため，この演算によって通信の範囲を限定することができる．プロセス定数は，再帰的にプロセスを定義するために用いる．

(2) SOS による操作意味定義

以上のような操作意味は，構造操作意味定義（Structural Operational Semantics: SOS）で簡潔に定義することができる．SOS では，遷移規則間の推論規則によって遷移関係を定義する．図 1･3 に SOS 規則の一般的なかたちを示す．この規則は，前提の遷移である $t_i \xrightarrow{\alpha_i} t_i'$ が存在するとき結論の遷移 $u \xrightarrow{\alpha} u'$ が存在することを示す*．ここで，t_i は u の部分項であり，u の遷移を u の構造に沿って推論する．図 1･4 に Milner の CCS に対する SOS 規則を示す．各規則における X，Y，X'，Y' は変数であり，プロセス式が代入される．

$$\frac{\{t_i \xrightarrow{\alpha_i} t_i'\}_{i\in I}}{u \xrightarrow{\alpha} u'}$$

図 1･3　SOS 規則の一般形

この規則を使って推論されるラベルつき遷移のみが存在する．$f=\{c \mapsto b\}$ とするとき，ラベルつき遷移 $((a.P+b.Q)|(c.R+\overline{a}.S))\backslash\{b\} \xrightarrow{\tau} (P|R)[f]\backslash\{b\}$ は以下のような推論によって導かれる．

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a.P \xrightarrow{a} P}\,(\mathrm{Act})}{a.P+b.Q \xrightarrow{a} P}\,(\mathrm{Sum_l}) \qquad \dfrac{\dfrac{}{\overline{a}.S \xrightarrow{\overline{a}} S}\,(\mathrm{Act})}{c.R+\overline{a}.S \xrightarrow{\overline{a}} S}\,(\mathrm{Sum_r})}{(a.P+b.Q)|(c.R+\overline{a}.S) \xrightarrow{\tau} P|S}\,(\mathrm{Comm})}{((a.P+b.Q)|(c.R+\overline{a}.S))[f] \xrightarrow{\tau} (P|S)[f]}\,(\mathrm{Rel})}{((a.P+b.Q)|(c.R+\overline{a}.S))[f]\backslash\{b\} \xrightarrow{\tau} (P|S)[f]\backslash\{b\}}\,(\mathrm{Res})$$

CCS 式によって，従前にあげた自動販売機の振る舞いを記述してみよう．それぞれの振る舞いをプロセス定数 V_1，V_2 で表すことにすると，定義式によって以下のように記述できる．

† $\alpha=\tau$ の場合は通信を行わずに内部的な動作によって状態が遷移する．

* このほか前提に遷移が存在しないことを示す否定形を含む場合もある[5]．

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \quad (\mathsf{Act})$$

$$\frac{X \xrightarrow{\alpha} X'}{X+Y \xrightarrow{\alpha} X'} \quad (\mathsf{Sum_l}) \qquad \frac{Y \xrightarrow{\alpha} Y'}{X+Y \xrightarrow{\alpha} Y'} \quad (\mathsf{Sum_r})$$

$$\frac{X \xrightarrow{\alpha} X'}{X|Y \xrightarrow{\alpha} X'|Y} \quad (\mathsf{Comp_l}) \qquad \frac{Y \xrightarrow{\alpha} Y'}{X|Y \xrightarrow{\alpha} X|Y'} \quad (\mathsf{Comp_r}) \qquad \frac{X \xrightarrow{\lambda} X',\ \ Y \xrightarrow{\overline{\lambda}} Y'}{X|Y \xrightarrow{\tau} X'|Y'} \quad (\mathsf{Comm})$$

$$\frac{X \xrightarrow{\alpha} X'}{X\backslash A \xrightarrow{\alpha} X'\backslash A} \quad \text{if } \alpha \in A \cup \overline{A} \quad (\mathsf{Res}) \qquad \frac{X \xrightarrow{\alpha} X'}{X[f] \xrightarrow{f(\alpha)} X'[f]} \quad (\mathsf{Rel})$$

$$\frac{X \xrightarrow{\alpha} X'}{A \xrightarrow{\alpha} X'} \quad \text{if } A \stackrel{\text{def}}{=} X \quad (\mathsf{Con})$$

図 1·4　CCS に対する SOS 規則

$$V_1 \stackrel{\text{def}}{=} c.(c.j.V_1 + b.t.V_1) \qquad V_2 \stackrel{\text{def}}{=} c.c.j.V_2 + c.b.t.V_2$$

V_1 と V_2 の振る舞いは，顧客定義 $C \stackrel{\text{def}}{=} \overline{c}.\overline{c}.\overline{j}.\overline{happy}.0$ との並行合成の遷移によって区別される．ここで，コインの投入，ボタンの操作，飲物の受け取りは，自動販売機と顧客の間でのみ行われるので，$L = \{c, b, j, t\}$ によって制限する．$(V_1|C)\backslash L$ に対しては以下の遷移系列が存在する．

$$\begin{aligned}(V_1|C)\backslash L &\xrightarrow{\tau} ((c.j.V_1 + b.t.V_1)|\overline{c}.\overline{j}.\overline{happy}.0)\backslash L \xrightarrow{\tau} (j.V_1|\overline{j}.\overline{happy}.0)\backslash L \\ &\xrightarrow{\tau} (V_1|\overline{happy}.0)\backslash L \xrightarrow{\overline{happy}} (V_1|0)\backslash L\end{aligned}$$

$(V_2|C)\backslash L$ に対する遷移系列は以下の二つである．

$$\begin{aligned}(V_2|C)\backslash L &\xrightarrow{\tau} (c.j.V_2|\overline{c}.\overline{j}.\overline{happy}.0)\backslash L \xrightarrow{\tau} (j.V_2|\overline{j}.\overline{happy}.0)\backslash L \\ &\xrightarrow{\tau} (V_2|\overline{happy}.0)\backslash L \xrightarrow{\overline{happy}} (V_2|0)\backslash L \\ (V_2|C)\backslash L &\xrightarrow{\tau} (b.t.V_2|\overline{c}.\overline{j}.\overline{happy}.0)\backslash L\end{aligned}$$

1 番目の系列となった場合はラベル $\overline{happy}$ による遷移が可能であるが，2 番目の系列ではコインが投入できなくなってしまう．これは，自動販売機の最初のコインによる遷移によって決定し，顧客からは制御できない．

(3) 双模倣等価性による代数的意味論

CCS の項を与えるとその項からのラベルつき遷移が SOS による意味論で定義される．項 P_1 と P_2 を含む双模倣関係があれば，その振る舞いを区別することができないため，等価であるといえる．この関係は双模倣等価性 $P_1 \sim P_2$ にほかならない．

CCS の項の構成子に対してこの等価関係が保存されれば，CCS の項の構成は，並行プロセスモデルの基本要素として適当である．CCS では，双模倣等価性に対して以下の性質が成り立つ．

定理 6　*$P \sim Q$ である CCS の項 P_1,Q_2 に対して，以下が成り立つ．*

- 任意の $\alpha \in Act_\tau$ に対して $\alpha.P_1 \sim \alpha.P_2$
- 任意の Q に対して $P_1 + Q \sim P_2 + R$, $Q + P_1 \sim Q + P_2$
- 任意の Q に対して $P_1|Q \sim P_2|Q$, $Q|P_1 \sim Q|P_2$
- 任意の $L \subseteq Lbl$ に対して $P_1\backslash L \sim P_2\backslash L$
- 任意の f に対して $P_1[f] \sim P_2[f]$

上記の結果を繰り返し用いることによって，$P \sim Q$ であるような P,Q を任意の部分項と置き換えてもその等価性を保存することができる．特に，並行合成についてこの等価性が保存されることから，並行プログラムを CCS によってモデル化することで，並行プログラムにおけるモジュール性の基盤を与えることができる．

(4) 弱等価関係

上記で与えた双模倣等価性による意味論は，項の構成に対して等価性が保存されて，理論的には優れた性質をもっている．しかし，等価性としては，非常に細かい等価性であるため，等しい項は限定的である．実際のプログラムをモデル化するためには，双模倣性による観測をもう少し抽象化する必要がある．CCS における意味論で，通信は同じ名前をもつ相補なラベル間で一対一に発生し，通信によるラベルは τ で置き換えられる．このモデル化は，通信が発生すれば，それ以上は観測しないことを意味する．そこで，τ による遷移を抽象化した遷移を弱遷移関係として以下のように定義する．

定義 7 (弱遷移関係) $s \in Act^*$ に対して $\overset{s}{\Rightarrow}$ を以下のように定義する．

- $\overset{\varepsilon}{\Rightarrow} \triangleq (\overset{\tau}{\rightarrow})^*$
- $\overset{as}{\Rightarrow} \triangleq \overset{\varepsilon}{\Rightarrow} \circ \overset{a}{\rightarrow} \circ \overset{s}{\Rightarrow}$

弱遷移関係から導かれる弱双模倣関係と弱双模倣等価性は以下のように定義される．

定義 8 (弱双模倣関係) Act 上の $LTS\langle Q, \mathcal{T}\rangle$ における Q 上の関係 R が以下の条件を満たすとき，R は弱双模倣関係（weak bisimulation）である．

すべての $(p,q) \in Q$ に対して以下が成立する．

- $p \overset{a}{\rightarrow} p_1$ に対して，q_1 が存在して $q \overset{a}{\Rightarrow} q_1$ かつ $(p_1, q_1) \in R$
- $p \overset{\tau}{\rightarrow} p_2$ に対して，q_2 が存在して $q \overset{\varepsilon}{\Rightarrow} q_2$ かつ $(p_2, q_2) \in R$
- $q \overset{a}{\rightarrow} q_1$ に対して，p_1 が存在して $p \overset{a}{\Rightarrow} p_1$ かつ $(p_1, q_1) \in R$
- $q \overset{\tau}{\Rightarrow} q_2$ に対して，p_2 が存在して $p \overset{\varepsilon}{\rightarrow} p_2$ かつ $(p_2, q_2) \in R$

定義 9 (弱双模倣等価性) $\approx \triangleq \{(p,q); (p,q) \in R$ となる弱双模倣関係 R が存在する $\}$

$\sim$ と同様に $\approx$ は等価関係であり，最大の弱双模倣関係であることが示される．$\approx$ は，任意の τ 以外の遷移による振る舞いの違いを区別しない．このため，前節の双模倣等価性よりも

多くの CCS 項を等しいとみなす．例えば，$\tau.a.0 \not\sim a.0$ であるのに対して，$\tau.a.0 \approx a.0$ である*．τ 遷移がオートマトンにおける ε 遷移のように観測できないことから，弱双模倣等価性は，**観測等価性（Observation Equivalence）**とも呼ばれる．

命題 10 $p \approx q$，$s \in Act^*$ のとき，以下が成り立つ．

- $p \stackrel{s}{\Rightarrow} p_1$ ならば，q_1 が存在して，$q \stackrel{s}{\Rightarrow} q_1$ かつ $p_1 \approx q_1$
- $q \stackrel{s}{\Rightarrow} q_2$ ならば，p_2 が存在して，$p \stackrel{s}{\Rightarrow} p_2$ かつ $p_2 \approx q_2$

$\approx$ は，$+$ 以外の演算子に対して保存される．

$+$ 演算子で $\approx$ は保存されないことは以下の反例による．$P = \tau.a.0$，$Q = a.0$ とすると $P \approx Q$ である．しかし，$P + b.0 \not\approx Q + b.0$ である．なぜなら，$P + b.0 \stackrel{\varepsilon}{\Rightarrow} a.0$ であるが，$Q + b.0$ と $\stackrel{\varepsilon}{\Rightarrow}$ の関係にあるのは，$Q + b.0$ のみであり，明らかに $a.0 \not\approx Q + b.0$ である．

しかし，$+$ が前置演算子によってガードされていれば弱等価性は保存される．

定理 11 $P_1 \approx P_2$ ならば任意の $\alpha \in Act_\tau$ と Q に対して $\alpha.P_1 + Q \approx \alpha.P_2 + Q$ である．

$+$ 演算子について任意の文脈で保存される等価性として，文献 24) では，観測合同関係（Observation Congruence）が提案されている．これは，$\approx$ よりも細かい等価性のうちで，$+$ を保存する最大の等価性として定義される．しかし，$+$ 演算を弱等価性の意味で用いる場合，τ による選択と Lbl による選択を混在させて記述することは少ないことから，実際的には弱等価性は代数的な意味としては十分である．

一般に観測可能な通信と観測できない通信が混在する選択に対して弱等価性を定義する場合，同様の問題が発生する．CSP では，τ による前置演算を導入せず，外部から通信によって選択可能な選択演算と外部から制御できない内部的選択演算を導入している．CCS に対しても同様な体系が提案されている[17]．

1-2-5 等価性に基づく動作検証

個々のコンポーネントを並行合成によって組み合わせることで得られる振る舞いが仕様を満たすかどうかは，内部コンポーネント間の通信を制限演算子で局所化し，τ によるラベルつき遷移を抽象化することによって検証することができる．実現がコンポーネント Imp_i で構成され，全体の仕様が $Spec$ で表されるとすると，

$$(Imp_1 | \cdots | Imp_n) \backslash A \approx Spec$$

を証明することで，Imp_i による実現が $Spec$ を満たしていることを示すことができる．ここで A は外部仕様 $Spec$ には現れない内部的なポートの集合を表す．

以下に簡単なオルタネーティングビットプロトコル（Alternating Bit Protocol: ABP）の例を示す*

* $\sim$ を**強双模倣等価関係**と呼ぶ．
* この例は，文献 24) の 6 章に示されている例に基づく．

このプロトコルが，消失のある通信路においても信頼性のある通信を保証することを示す．送信側プロセスは，accept によって通信要求を受け取り，送信通信路 *Trans* に送出する．受信側プロセスは，trans から要求を受け取り，deliver から送信のあったことを出力して，ビット情報を反転して，送信側へ受信確認を返送する．受信側では，反転した確認信号を受け取ると次の受信要求を受け取る．通信路 *Trans, Ack* では信号が消失することがある．図 1･5 に全体の構成を示す．

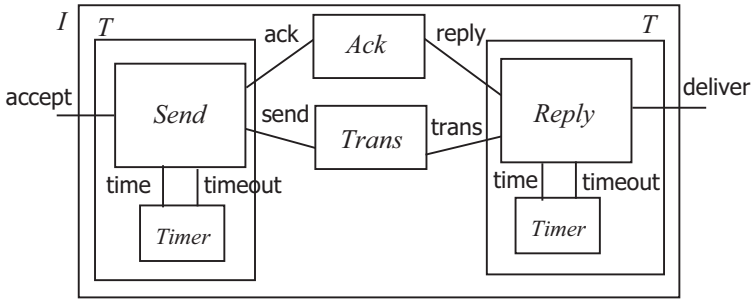


図 1･5　ABP の構成

以下にプロセス定義式を示す．ここで，b は 0, 1，$\hat{b}$ は $b-1$ とする．

$$
\begin{aligned}
Send_b &\stackrel{\text{def}}{=} \overline{\mathsf{send}_b}.\overline{\mathsf{time}}.Sending_b \\
Sending_b &\stackrel{\text{def}}{=} \mathsf{timeout}.Send_b + \mathsf{ack}_b.\mathsf{timeout}.Accept_{\hat{b}} + \mathsf{ack}_{\hat{b}}.Sending_b \\
Accept_b &\stackrel{\text{def}}{=} \mathsf{accept}.Send_b \\
Reply_b &\stackrel{\text{def}}{=} \overline{\mathsf{reply}_b}.\overline{\mathsf{time}}.Replying_b \\
Replying_b &\stackrel{\text{def}}{=} \mathsf{timeout}.Reply_b + \hat{\mathsf{b}}.\mathsf{timeout}.Deliver_{\hat{b}} + \mathsf{trans}_b.Replying_b \\
Deliver_b &\stackrel{\text{def}}{=} \overline{\mathsf{deliver}}.Reply_b \\
Timer &\stackrel{\text{def}}{=} \mathsf{time}.\overline{\mathsf{timeout}}.Timer
\end{aligned}
$$

$Send_b$ では，通信路に信号 0 または 1 を送出する．その後タイマーをセットして受信側からの確認信号を待つ．反転した信号が送られてくれば次の通信要求を受け取るが，タイムアウトした場合には再送し，反転しない確認信号が来た場合には，反転した確認信号を待つ．受信側でもビットを反転して同様の振る舞いをする．

通信チャネル *Trans* と *Ack* を以下のように定義する．

$$
\begin{aligned}
Trans &\stackrel{\text{def}}{=} \mathsf{send}_0.(\overline{\mathsf{trans}_0}.Trans + Trans) + \mathsf{send}_1.(\overline{\mathsf{trans}_1}.Trans + Trans) \\
Ack &\stackrel{\text{def}}{=} \mathsf{reply}_0.(\overline{\mathsf{ack}_0}.Trans + Trans) + \mathsf{reply}_1.(\overline{\mathsf{ack}_1}.Ack + Ack)
\end{aligned}
$$

それぞれの通信路は，入力信号を入力して出力にコピーする．しかし，出力をせずに入力を受け付けるかもしれない．これらの定義を合成して，全体プロセス *ABP* を以下のように構成する．

$$ABP \stackrel{\text{def}}{=} ((Accept_1|Timer)\backslash T \mid (\overline{\text{time}}.Replying_0)\backslash T \mid Trans \mid Ack)\backslash I$$

ここで，$I = \{\text{send}_b, \text{trans}_b, \text{reply}_b, \text{ack}_b; b = 0, 1\}$，$T = \{\text{time}, \text{timeout}\}$ である．

ABP が正しいプロトコルであることは $ABP \approx Buf$ であることによって検証することができる．

$$Buf \stackrel{\text{def}}{=} \text{accept}.\overline{\text{deliver}}.Buf$$

実際に $ABP \approx Buf$ であることを示すことは可能である．ただし，Buf のそれぞれに対応する ABP の状態が 100 以上となるため人手で行うことは現実的ではない．弱等価性検証のための手続を実現した等価性検証のためのツール（Concurrency Workbench NC[10]など）を用いて，この等価性を検証することができる．

1-2-6 モバイル計算（π 計算）

CCS における記述における制約は，計算の進行にともなって動的に通信状況が変化する並行システムを記述できないことにある．この点について拡張した体系が提案されている[7, 11, 29]．ここでは，CCS の拡張として Milner らによる提案されている π 計算について簡単に紹介する．より詳細な内容については文献 25, 30) を参照頂きたい．

π 計算は，CCS の通信の際に通信ラベル名を受け渡すように拡張した体系である．CCS では，前置演算が $Act_\tau.E$ であるのに対して，π 計算では，$\bar{x}y.E$,$x(z).E$,$\tau.E$ のようなかたちに拡張され，ポート自体を受け渡しできる．$\bar{x}y.E$ は，通信ラベル x から通信名 y を出力して，E になるプロセスを表し，$x(z).E$ は，通信ラベル x から通信名を受け取り，E に出現する（自由な）y とおきかえる．例えば，$P \stackrel{\text{def}}{=} ab.0$, $Q \stackrel{\text{def}}{=} \bar{a}x.\bar{x}c.0$ とすると，$P|Q \xrightarrow{\tau} 0|\bar{b}c.0$ という遷移が存在する．ここで，CCS との大きい違いは a による通信によって P から Q へ受け渡されたラベルが，Q の次の通信のラベルに使われることである．Q の 2 番目の通信相手は最初の通信によって決定する．

π 計算の前置演算子（prefix）は以下のように表される．

$$\pi \quad ::= \quad \bar{x}y \;\; x(z) \;\; \tau \;\; [x = y]\pi$$

π 計算のプロセスの構文 P は以下のように定められる．

$$P \quad ::= \quad M \;\; P|P' \;\; \nu z.P \;\; !P$$
$$M \quad ::= \quad 0 \;\; \pi.P \;\; M + M'$$

合成演算 |，選択演算 + は，CCS 同様の意味をもつ．$\nu z\, E$ は制限演算子である．CCS と同様に z に関する通信を E に限定する．ここで，E は，z を z で制約されない通信ポートから出力することができる．！は複製演算（replication）を表す．$!P$ は，プロセス P が必要に応じて無限に合成されることを表し，プロセスの無限の振る舞いを表現するために用いられる．

$x(z).P$ と $\nu z.P$ というかたちの項において項に現れる z は P において束縛されているという．P において束縛されていない名前を $\text{fn}(P)$ と表す．

(1) 基本意味：構造合同

π 計算ではまず構文的な等価性を定義し，その等価性に基づいて振る舞いの定義を行うこ

$$
\begin{aligned}
[x = x]\pi.P &\equiv \pi.P \\
M_1 + (M_2 + M_3) &\equiv (M_1 + M_2) + M_3 \\
M_1 + M_2 &\equiv M_2 + M_1 \\
M + 0 &\equiv M \\
P_1|(P_2\ vert P_3) &\equiv (P_1|P_2)|P_3 \\
P_1|P_2 &\equiv P_2|P_1 \\
P|0 &\equiv P \\
\nu v \nu w P &\equiv \nu w \nu v P \\
\nu z 0 &\equiv 0 \\
\nu z(P_1|P_2) &\equiv P_1|\nu z P_2, \text{ if } z \notin \mathsf{fn}(P_1) \\
!P &\equiv P|!P
\end{aligned}
$$

$$
\begin{aligned}
P &= P \\
P = Q &\quad \text{implies } Q = P \\
P = Q \text{ and } Q = R &\quad \text{implies } P = R \\
P = Q &\quad \text{implies } C[P] = C[Q]
\end{aligned}
$$

図 1･6　構造合同性

とが多い．構造合同関係 ≡ は以下の関係から等式論理規則を使って生成される合同関係である．

構造合同である項は構文的性質から等価であるとみなす．構造合同意味は π 計算における最も細かい意味論として位置づけられる．

(2) SOS による操作意味定義

π 計算でも CCS と同様に SOS によって操作意味を定義する．複製演算以外の操作意味 $\xrightarrow{\alpha}$ を図 1･7 に示す．ここで，α は，xy,$\bar{x}y$,$\bar{x}(y)$,τ のいずれかである．τ 以外のラベルの最初の名前は通信ポート，2 番目の名前は受け渡される名前を表す．$\bar{x}$ のかたちの通信ポートは出力，名前だけの場合は入力を表す．入力 $xz.P$ では，出力から x において受け取った名前を，P における自由な z と置換する．

π 計算で，特徴的なのは制限演算によって束縛された名前を出力することで名前のスコープを動的に変化させることができる点である．図 1･7 の操作意味論によって以下の遷移が導かれる．

$$
\dfrac{\dfrac{\dfrac{}{\bar{a}z.z(w).P \xrightarrow{\bar{a}z} z(w).P}\,(\mathrm{OUT})}{\nu z(\bar{a}z.z(w).P) \xrightarrow{\bar{a}(z)} z(w).P}\,(\mathrm{OPEN}) \qquad \dfrac{\dfrac{}{a(x).\bar{x}y.Q \xrightarrow{ax} \bar{z}y.Q\{z/x\}}\,(\mathrm{IN})}{\nu y(a(x).\bar{x}y.Q) \xrightarrow{az} \nu y(\bar{z}y.Q\{z/x\})}\,(\mathrm{RES})}{(\nu z)(\bar{a}z.z(w).P)|(\nu y)(a(x).\bar{x}y.Q) \xrightarrow{\tau} (\nu z)((z(w).P)|(\nu y)(\bar{z}y.Q\{z/x\}))}\,(\mathrm{CLOSE-L})
$$

ここで，遷移関係 $\xrightarrow{\tau}$ の右側の項における名前 z は，合成演算子 | の左側に制限されている

$$(\mathsf{OUT})\ \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad (\mathsf{IN})\ \frac{}{x(z).P \xrightarrow{xy} P\{y/z\}}$$

$$(\mathsf{TAU})\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad (\mathsf{MAT})\ \frac{\pi.P \xrightarrow{\alpha} P}{[x=x]\pi.P \xrightarrow{\alpha} P}$$

$$(\mathsf{SUM-L})\ \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad (\mathsf{SUM-R})\ \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

$$(\mathsf{PAR-L})\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ \mathsf{bn}(\alpha)\cap\mathsf{fn}(Q)=\emptyset \qquad (\mathsf{PAR-R})\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}\ \mathsf{bn}(\alpha)\cap\mathsf{fn}(P)=\emptyset$$

$$(\mathsf{COMM-L})\ \frac{P \xrightarrow{\overline{x}y} P',\ Q \xrightarrow{xy} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad (\mathsf{COMM-R})\ \frac{P \xrightarrow{xy} P',\ Q \xrightarrow{\overline{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$(\mathsf{CLOSE-L})\ \frac{P \xrightarrow{\overline{x}(z)} P',\ Q \xrightarrow{xz} Q'}{P|Q \xrightarrow{\tau} \nu z(P'|Q')} \qquad (\mathsf{CLOSE-R})\ \frac{P \xrightarrow{xz} (z)P',\ Q \xrightarrow{\overline{x}Q'}}{P|Q \xrightarrow{\tau} \nu z(P'|Q')}$$

$$(\mathsf{RES})\ \frac{P \xrightarrow{\alpha} P'}{\nu z P \xrightarrow{\alpha} \nu z P'} \qquad (\mathsf{OPEN})\ \frac{P \xrightarrow{\overline{x}z} P'}{\nu z P \xrightarrow{\overline{x}(z)} P'}\ z \neq x$$

図 1·7　π 計算の動作意味論 (1)

が，遷移後には項全体に対する制限となっている．これは，ポート a を通じて，左側から右側に z が知らされるようになり，制限のスコープは項全体となる．部分項に対して OPEN を適用することで部分項の制約情報を公開し，CLOSE によってスコープを拡げる．

更に以下の遷移によって，合成演算の右側にのみ有効であった y のスコープが項全体になる．このようなスコープの変化をスコープ拡大（scope extrusion）と呼ぶ．

$$\nu z((z(w).P)|\nu y(\overline{z}y.Q)) \xrightarrow{\tau} \nu z(\nu y(P\{y/w\}|Q\{z/x\}))$$

$P = \overline{x}a.0$, $Q = a(z).0$, $R = x(y).\overline{y}b.0$ とすると，$\nu a(P|Q)|R$ の振る舞いは以下のようになる．

$$\nu a(\overline{x}a.0|a(z).0)|x(y).\overline{y}b.0 \xrightarrow{\tau} \nu a(0|a(z).0|\overline{a}b.0) \xrightarrow{\tau} \nu a(0|0|0)$$

初期状態では，a による通信は，P と Q に制限されている．P は制限された a を R に通信し，a によって，P 及び Q と通信が可能になる．通信の組み合わせは図 1·8 のように変化する．

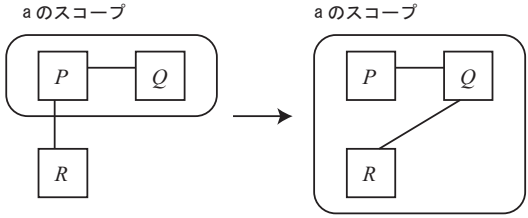


図 1·8　名前のスコープ拡大

複製演算の操作意味は図 1·9 のように定義できる．複製演算ではなく，CCS と同様にプロセス定数を導入し，再帰によっても同様な操作意味を構成することが可能である*．

* 複製演算と再帰による定義は一般には相違がある[1)]．

$$(\mathsf{REP}-\mathsf{ACT})\quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'|!P}$$

$$(\mathsf{REP}-\mathsf{COMM})\quad \frac{P \xrightarrow{\overline{x}y} P',\ P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P'|P'')|!P}$$

$$(\mathsf{REP}-\mathsf{CLOSE})\quad \frac{P \xrightarrow{\overline{x}(z)} P',\ P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} (\nu z(P'|P''))|!P}$$

図 1·9　π 計算の動作意味論 (2)

(a) 双模倣意味論について

π 計算に対して操作意味をラベルつき遷移として双模倣意味論が展開されている．代数的意味を構成する場合，名前の入力を表す前置演算子 $xy.$ について，意味を保存するのが難しくなっている．この理由は，y に束縛される名前がどのように後で通信に関係してくるかという自由度が大きいために，代数意味を構成しようとすると非常に意味論が細かくなる．

CCS と同様に強双模倣等価性 $\sim$ を定義すると，以下の等価性が成立する．

$$x(z).0|\overline{y}z.0 \sim x(z).\overline{y}z.0 + \overline{y}z.x(z).0$$

入力前置演算 $w(x).$ に対して $\sim$ が保存されるとすると，

$$w(x).(x(z).0|\overline{y}z.0) \sim w(x).(x(z).\overline{y}z.0 + \overline{y}z.x(z).0)$$

でなければならない．両辺を $\overline{w}y$ と合成すると

$$(w(x).(x(z).0|\overline{y}z.0)|\overline{w}y \not\sim\ w(x).(x(z).\overline{y}z.0 + \overline{y}z.x(z).0)|\overline{w}y$$

となる．これは，左辺のプロセス項を P とおくと，

$$P \xrightarrow{\tau} (y(z).0|\overline{y}z.0)|0 \xrightarrow{\tau} 0|0|0$$

という遷移が存在する．しかし，右辺にはこのような遷移が存在しない．これは，左辺が入力された名前の一致によって内部的な通信が可能になるのに対し，右辺にはもともと内部的な通信が行われることがないためである*.

入力に対しても代数的な意味を保存するためには，名前の等価性をすべて記録しながら意味を構成する必要がある．名前の等価性を記録することによって定義される意味論として開双模倣等価性（open bisimulation equivalene）[30, Section 4.6]がある．

このほか，π 計算に対しても様々な双模倣等価性意味論については，文献 30) に詳しく述べられている．

1-2-7 まとめ

本節では，形式的な並行計算モデルである通信プロセスモデルについて簡単に述べた．通

* 展開定理においてこのような問題を生じさせない項を定義するために，π 計算では条件付き前置演算 $[x = y]\pi.$ が導入されている．

信プロセスモデルのより完全な解説は文献 4) が詳しい．通信プロセスモデルの特徴は，振る舞いの等価性に基づいて，項を構成する演算子から代数的な意味論を構成することにある．ここでは，操作定義とそこから導かれる意味論の基本的な部分について説明した．この意味論に基づいて，代数としての性質を直接的に示す公理化，振る舞いの性質を論理式で表現して検証するモデル検査技法[18, 9]が研究され，更に時間や確率などの概念を導入してモデルを拡張する研究が行われている[31, 2, 22]．このほか，ACP[3]は，まず等式によって代数を定めてから，適合した操作的意味論を定義している．CCS，CSP などの通信プロセス計算は，記述に対して関数の外延性が自然に適用できないような並行プログラムに対して基盤となる解析技術を与える．更に，通信プロセス計算では双模倣関係を定義するラベルつき遷移の観測を定めることで振る舞いに対して様々な等価性が得られる[28, 13, 14, 15]．しかし，通信プロセス計算自体の与える一般的な意味論を，実際的規模の並行プログラムの意味記述にそのまま適用して，任意の望ましい性質を検証をすることは難しい．これは通信プロセス計算の問題点ではなく，並行計算自体がもつ意味の難しさを反映しているといえる．実験的な検証ツール[10, 23, 32, 26]が作成されているが実用的な規模のソフトウェアの検証を直接行うことは難しい．

更に，通信リンクの組み合わせが計算の進行に従って変化する体系として π 計算について簡単に説明した．π 計算では，CCS と同様に相補的な通信ラベル間で通信が発生するとし，この際に通信ラベルを値として受け渡し，その通信ラベルをその後の通信のために使うことができるように拡張している．この機能によって，実行途中で通信ラベルを変更することができるようになり，計算の進行と共に動的に通信の組み合わせが変化する並行計算を記述することができる．モバイル計算の体系によって，オブジェクト指向言語の意味記述[33]など，高階の機能をもつ並行ソフトウェアの意味記述が可能になる．更に，分散システムをモデル化するために計算の位置を区別する体系も提案されている[8, 19]．π 計算は非常に多くの種類の計算を記述することができるため，プロセス項の名前に対する型の付加による型理論が展開されている．特定の性質をもつ名前の型が振る舞いによって保存されることが示される．このことによって，プロセスの振る舞いの性質を構文的に特定することが可能となる．更に，有用な部分計算体系として非同期 π 計算の研究が行われている．非同期 π 計算では，出力前置子 $\overline{x}z$ 以降には，0 しか認めない．これでも計算としては十分な能力をもち，理論の展開の単純化に成功している．このほか，同様のアイデアに基づいた様々なモバイルプロセス計算体系が提案されている[11, 29, 7]．

これまで通信プロセス計算のモデルは，並行計算の理論な体系として 80 年代から広く研究が行われてきた．その研究の歴史の長さにもかかわらず，実際的な応用において直接的に適用された場合はそれほど多くない．しかし，ネットワーク上での通信を中心にした計算メカニズムが急速に広まっていく状況で，計算の基本的な表現として問題の解析手法を開発していくことで，より見通しのよい解析技術さらには設計技法が得られると期待できる．通信プロセスモデルと通信プロセス計算はこの目的において十分厳密な基盤を与える．

■参考文献

1) J. Aranda, C.D. Giusto, and C. Palamidessi, “On Recursion, Replication and Scope Mechanisms in Process Calculi,” Proc. of Formal Methods for Components and Objects 2007, Lecture Notes in Computer Science 4709, pp.185-206, Springer, 2007.

2) J.C.M. Baeten and C.A. Middelberg, “Process Algebra with timing: Real time and discrete time,” Chapter 10 in Handbook of Process Algebra, pp.627-684, Elsevier, 2001.
3) J.C.M. Baeten and W.P. Weijland, “Process Algebra,” Number 18 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.
4) J.A. Bergstra, A. Ponse, and S.A. Smolka, “Handbook of Process Algebra,” Elsevier, 2001.
5) B.Bloom, S. Istrail, and A. Mayer, “Bisimulation can’t be traced,” J. ACM, vol.42, no.1, pp.232-268, 1995.
6) S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe, “A theory of communicating sequential processes,” J. ACM, vol.31, no.3, pp.560-599, 1984.
7) L. Cardelli and A. Gordon, “Mobile Ambients,” Theoretical Computer Science, vol.240, no.1, pp.177-213, 2000.
8) I. Castellani, “Process algebra with localities,” Chapter 15 in Handbook of Process Algebra, pp.945-1046, Elsevier, 2001.
9) E.M. Clarke, O. Grumberg, and D.P. Peled, “Model Checking,” MIT, Press, 1999.
10) Concurrency workbench new century, http://www.cs.sunysb.edu/~cwb/.
11) C. Fournet and G.Gonthier, “The reflexive CHAM and the join-calculus,” in Proc. of POPL’96, pp.372-385, 1996.
12) R.V. Glabbeek and F. Vaandrager, “Petri Net Models for Algebraic Theories of Concurrency,” in Proc. of PEARL’87, Lecture Notes in Computer Science 259, vol.II, pp.224-242, Springer, 1987.
13) R.V. Glabbeek, “The linear time-branching time spectrum,” in Proc. of CONCUR90, Lecture Notes in Computer Science 458, pp.278-297, Springer, 1990.
14) R.V. Glabbeek, “The linear time-branching time spectrum II,” in Proc. of CONCUR93, Lecture Notes in Computer Science 715, pp.278-297, Springer, 1993.
15) R.V. Glabbeek, “The linear time-branching time spectrum I,” Chapter 1 in Handbook of Process Algebra, pp.3-100, Elsevier, 2001.
16) M. Hennessy and T. Regan, “A process algebra for timed systems,” Information and Computation, vol.117, no.2, pp.221-239, March, 1995.
17) M. Hennessy, “Algebraic Theory of Processes,” MIT Press, 1988.
18) M. Hennessy and R. Milner, “Algebraic laws for nondeterminism and concurrency,” J. ACM, vol.32, no.1, pp.137-161, 1985.
19) M. Hennessy, “A Distributed Pi-Calculus,” Cambridge University Press, 2007.
20) C.A.R. Hoare, “Communicating sequential processes,” Communication of ACM, vol.21, no.8, pp.666-677, August, 1978.
21) C.A.R. Hoare, “Communicating Sequential Processes,” Prentice-Hall International Series of Computer Science, Prentice Hall, 1985. Available online http://www.usingcsp.com/cspbook.pdf.
22) B. Jonsson, W. Yi, and K.G. Larsen, “Probablistic extensions of process algebras,” Chapter 11 in Handbook of Process Algebra, pp.685-710, Elsevier, 2001.
23) J. Magee and J. Kramer, “Concurrency:State Models & Java Programs,” Wiley, 1999.
24) R. Milner, “Communication and Concurrency,” Prentice-Hall International Series of Computer Science, Prentice Hall, 1989.
25) R. Milner, “Communicating and Mobile Systems: The Pi-Calculus,” Cambridge University Press, 1999.
26) Mobility Workbench, http://www.it.uu.se/research/group/mobility/mwb.
27) R. De Nicola and M.C.B. Hennessy, “Testing equivalences for processes,” Theoretical Computer Science, vol.34, nos.1-2, pp.83-133, November, 1984.
28) E.R. Olderog and C.A.R Hoare, “Specification-oriented semantics for communicating processes,” Acta Informatica, vol.23, pp.9-66, 1986.

29) J. Parrow and B. Victor, “The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes(extended abstract),” in Proc. of LICS98, IEEE, 1998.
30) D. Sangiorgi and D. Walker, “The π-calculus: A Theory of Mobile Processes,” Cambridge University Press, 2000.
31) S. Schneider, “Concurrent and Real-time Systems,” Wiley, 2000.
32) uppaal, `http://www.uppaal.com`.
33) D. Walker, “Wiley Objects in the π-calculus,” Information and Computation, vol.116, no.2, pp.253-271, 1995.

■7 群 - 1 編 - 1 章

1-3 項書換え系

（執筆者：草刈圭一朗）[2008 年 8 月 受領]

項書換え系とは計算モデルの一つであり，関数型言語に操作的意味を与えたり，代数仕様を記述するために用いられる．例えば，次の項書換え系 R_{ack} は項 $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \cdots$ を自然数 $0, 1, 2, \cdots$ とみなした場合，アッカーマン関数 *ack* の実現となっている．

$$R_{\mathrm{ack}} = \left\{ \begin{array}{rcl} \mathsf{ack}(0, y) & \to & \mathsf{s}(y) \\ \mathsf{ack}(\mathsf{s}(x), 0) & \to & \mathsf{ack}(x, \mathsf{s}(0)) \\ \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y)) \end{array} \right.$$

項 $\mathsf{ack}(\mathsf{s}(0), \mathsf{s}(0))$ を書換えて $\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$ を得る書換え列の一例を記す．なお，下線部は上述の規則を適用した個所である．

$$\underline{\mathsf{ack}(\mathsf{s}(0), \mathsf{s}(0))} \underset{R}{\to} \mathsf{ack}(0, \underline{\mathsf{ack}(\mathsf{s}(0), 0)}) \underset{R}{\to} \mathsf{ack}(0, \underline{\mathsf{ack}(0, \mathsf{s}(0))}) \underset{R}{\to} \underline{\mathsf{ack}(0, \mathsf{s}(\mathsf{s}(0)))} \underset{R}{\to} \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$$

この例は，$ack(1, 1)$ の値 3 を得る “計算” を表現している．項書換え系では，書換え関係 $\underset{R}{\to}$ を用いてこのように “計算” を表現する．

本節では，項書換え系と項書換え系において重要な停止性と合流性の代表的な証明法を紹介する．本節で取り扱わない完備化手続きや評価戦略などの話題やより最新の成果など，さらに深く学びたい方は節末に記してある参考文献を参照して頂ければ幸いである．

1-3-1 諸定義

関数記号の集合を Σ で記し，変数記号の集合を $\mathcal{V}$ で記す．ここで $\Sigma \cap \mathcal{V} = \emptyset$ であるとする．各関数記号 f は固有の引数個数を持ち，これを $arity(f)$ で記す．**項** (term) を以下のように帰納的に定義し，項の全体からなる集合を $\mathcal{T}(\Sigma, \mathcal{V})$ で記す．

- 変数 $x \in \mathcal{V}$ は項である．
- $f \in \Sigma$ かつ $arity(f) = n$ かつ $t_1, \ldots, t_n$ が項ならば $f(t_1, \ldots, t_n)$ も項である．

通例に従い項 $a()$ は単に a と記される．また，2 つの項 s と t が等しいことを $s \equiv t$ で表し，項 t に出現する変数全体からなる集合を $Var(t)$ で表す．

項 t における位置の集合 $\mathcal{O}(t)$ を正整数の列 (空列を ε で表現) を用いて，$\mathcal{O}(x) = \{\varepsilon\}$ かつ $\mathcal{O}(f(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \{iu \mid 1 \le i \le n,\ u \in \mathcal{O}(t_i)\}$ で定義する．$p \prec q$ を $pw = q$ $(w \neq \varepsilon)$ で定義する．$\mathcal{O}(t)$ の濃度 (元の個数) を項 t のサイズと呼び $|t|$ で記す．項 t の位置 p における **部分項** (subterm) を $t|_p$ で，位置 p に出現する記号を $(t)_p$ で記す．特に $p \neq \varepsilon$ のとき，$t|_p$ を **真部分項** (proper subterm) と呼ぶ．また，根位置に出現する記号 $(t)_\varepsilon$ を $root(t)$ で記すこともある．t の部分項全体からなる集合を $Sub(t)$ で記し，$s \trianglerighteq_{sub} t$ を $t \in Sub(s)$ で，$s \triangleright_{sub} t$ を $s \trianglerighteq_{sub} t \wedge s \neq t$ で定義する．

文脈 (context) とはホールと呼ばれる特別な定数記号 $\square$ を含む項である．ホール自身も文脈であり，このような文脈を空の文脈と呼ぶ．文脈 $C[, \ldots,]$ に出現する n 個の $\square$ を項 $t_1, \ldots, t_n$ で左から順に置き換えることによって得られる項を $C[t_1, \ldots, t_n]$ で記す．

代入 (substitution) とは変数から項への写像である．代入 θ に対して項上の代入 $\hat{\theta}$ を，

$\hat{\theta}(x) = \theta(x)$ かつ $\hat{\theta}(f(t_1,\ldots,t_n)) = f(\hat{\theta}(t_1),\ldots,\hat{\theta}(t_n))$ で定義する．以下では通例に従い θ と $\hat{\theta}$ を同一視する．また，$\theta(t)$ を $t\theta$ で記す．

書換え規則 (rewrite rule) とは項の対 (l,r) で変数条件，$l \notin V$ かつ $Var(l) \supseteq Var(r)$，を満たすものである．以下では $l \to r$ で記す．書換え規則の有限集合を**項書換え系** (term rewriting system) と呼ぶ．項書換え系 R に対し，$s \underset{R}{\to} t$ を $\exists l \to r \in R.\ \exists C[\,].\ \exists\theta.\ s \equiv C[l\theta] \wedge t \equiv C[r\theta]$ で定義する．関係 $\underset{R}{\to}$ を**書換え関係** (rewrite relation) と呼ぶ．また，項 $s \equiv C[l\theta]$ の部分項 $l\theta$ を**簡約基** (redex) と呼ぶ．簡約基を含まない項を**正規形** (normal form) と呼ぶ．$\underset{R}{\to}$ の推移閉包と反射推移閉包をそれぞれ $\underset{R}{\overset{+}{\to}}$ と $\underset{R}{\overset{*}{\to}}$ で記す．

反射性 $(\forall t.\ t \gtrsim t)$ と推移性 $(\forall s,t,u.[s \gtrsim t \gtrsim u \Rightarrow s \gtrsim u])$ を持つ関係 $\gtrsim$ を**擬順序**と呼ぶ．反射性と推移性と反対称性 $(\forall s,t.[s \geq t \wedge t \geq s \Rightarrow s = t])$ を持つ関係 $\geq$ を**半順序**と呼ぶ．推移性と非反射性 $(\forall t.\ t \not> t)$ を持つ関係 $>$ を**狭義の半順序**と呼ぶ．関係 $>$ が整礎であるとは，$t_0 > t_1 > t_2 > \cdots$ のような無限列が存在しない事である．項上の関係 $>$ が文脈に閉じるとは $\forall C[\,].\ C[s] > C[t]$ が，代入に閉じるとは $\forall\theta.\ s\theta > t\theta$ が，$s > t$ となる任意の項 s,t に対して成立することである．

1-3-2 停止性

項書換え系 R が**停止性** (termination) または**強正規性** (strongly normalizing) を持つとは，$t_0 \underset{R}{\to} t_1 \underset{R}{\to} t_2 \underset{R}{\to} \cdots$ のような無限書換え列が存在しないことである．項書換え系の代表的な停止性証明法として順序付けに基づく方法と再帰構造解析に基づく方法を紹介する．

(1) 簡約化順序と辞書式経路順序

停止性を証明する最も基本的な手法は簡約化順序と呼ばれる整礎順序により規則を順序付けるという手法である．ここでは，簡約化順序の概念を紹介し，代表的な簡約化順序として辞書式経路順序を紹介する．

定義 1 簡約化順序 *(reduction order)* とは文脈と代入に閉じている整礎な狭義の半順序 $>$ の事である．

定理 2 項書換え系 R の全ての規則 $l \to r \in R$ に対し $l > r$ となる簡約化順序 $>$ が存在するならば R は停止性を持つ．

証明: R が停止性を持たない，すなわち無限書換え列 $t_0 \underset{R}{\to} t_1 \underset{R}{\to} t_2 \underset{R}{\to} \cdots$ が存在すると仮定すると，簡約化順序 $>$ は文脈と代入に閉じているので無限減少列 $t_0 > t_1 > t_2 > \cdots$ が存在することになり，$>$ の整礎性に矛盾する． □

定義 3 優先順位 *(precedence)* とは関数記号上の狭義の半順序のことである．**辞書式経路順序** *(lexicographic path order)* とは優先順位 $\triangleright$ に基づき定義される項上の関係 $>_{lpo}$ である．具体的には $s \equiv f(s_1,\ldots,s_m) >_{lpo} t$ の定義は以下で与えられる．

- ある i が存在して $s_i \geq_{lpo} t$，または
- $t \equiv g(t_1,\ldots,t_n)$，各 j で $s >_{lpo} t_j$，かつ以下が成立

$$f \triangleright g,\ \text{または}\ f = g\ \text{かつ}\ [s_1,\ldots,s_m] >_{lpo}^{lex} [t_1,\ldots,t_n]$$

ここで，$>_{lpo}^{lex}$ は $>_{lpo}$ の辞書式拡張であり，$[s_1,\ldots,s_m] >_{lpo}^{lex} [t_1,\ldots,t_n]$ は $\exists k.\ (\forall i < k.\ s_i \equiv$

$t_i \wedge s_k >_{lpo} t_k$) で定義される．

本節の冒頭で与えた項書換え系 $R_{\mathtt{ack}}$ の全ての規則は優先順位を $\mathtt{ack} \rhd \mathtt{s}$ で与えることにより $>_{lpo}$ で順序付けることができる．次の定理より $>_{lpo}$ は簡約化順序であるので，定理 2 より $R_{\mathtt{ack}}$ の停止性が示される．

定理 4 辞書式経路順序 $>_{lpo}$ は簡約化順序である．

証明: (I) 推移性，すなわち $s >_{lpo} t >_{lpo} u \Rightarrow s >_{lpo} u$ は，$|s| + |t| + |u|$ に関する帰納法で示せる．(II) $>_{lpo}$ が文脈に閉じていること，すなわち $s >_{lpo} t \Rightarrow C[s] >_{lpo} C[t]$ は，$C[\,]$ に関する帰納法で示せる．(III) $>_{lpo}$ が代入に閉じていること，すなわち $s >_{lpo} t \Rightarrow s\theta >_{lpo} t\theta$ は，$|s| + |t|$ に関する帰納法で示せる．

以下では集合 $SN(>_{lpo})$ を，t で始まる $>_{lpo}$ の無限減少列 $(t >_{lpo} \cdot >_{lpo} \cdot >_{lpo} \cdots)$ が存在しないような項 t 全体からなる集合として定義する．

(IV) $s \equiv f(s_1, \ldots, s_m) >_{lpo} t$ かつ $s_1, \ldots, s_m \in SN(>_{lpo})$ ならば $t \in SN(>_{lpo})$ を，三つ組み $\langle f, [s_1, \ldots, s_m], |t| \rangle$ に関する帰納法で示す．なお，比較は $\rhd$ と $>_{lpo}^{lex}$ と自然数上の通常の順序 $>$ の辞書式結合を用いる．ある i で $s_i \geq_{lpo} t$ となる場合は明らか．$t \equiv g(t_1, \ldots, t_n)$ とし，各 j で $s >_{lpo} t_j$ とする．$|t| > |t_j|$ なので帰納法の仮定より $t_j \in SN(>_{lpo})$．$t >_{lpo} u$ となる任意の u を考える．今，$f \rhd g$ または $f = g \wedge [s_1, \ldots, s_m] >_{lpo}^{lex} [t_1, \ldots, t_n]$ なので，いずれの場合も $(f, [s_1, \ldots, s_m], |t|)$ より $(g, [t_1, \ldots, t_n], |u|)$ は小さい．よって，帰納法の仮定より $u \in SN(>_{lpo})$．$t >_{lpo} u$ となる任意の u に対して $u \in SN(>_{lpo})$ となるので $t \in SN(>_{lpo})$ が成立．

(V) 整礎性，すなわち任意の s で $s \in SN(>_{lpo})$ となることを s に関する帰納法で示す．$s \in \mathcal{V}$ の場合は明らか．$s \equiv f(s_1, \ldots, s_m)$ とする．帰納法の仮定より $s_1, \ldots, s_m \in SN(>_{lpo})$．(IV) より，$s >_{lpo} t$ となる任意の t に対して $t \in SN(>_{lpo})$ なので，$s \in SN(>_{lpo})$ が成立．

(VI) 非反射性を持たない ($t >_{lpo} t$ となる t が存在) と仮定すると (V) に矛盾． □

(2) 依存対法

依存対法は項書換え系の再帰構造を解析し，全ての再帰の部分 (再帰成分と呼ぶ) が非循環的であることを示すことにより項書換え系の停止性を示す手法である．多くの場合，簡約化順序で直接停止性を証明するよりも容易に停止性を証明することができる．

定義 5 項書換え系 R を考える．関数記号 f が**被定義記号** *(defined symbol)* であるとは，ある規則 $l \to r \in R$ が存在して $root(l) = f$ となることである．被定義記号でない関数記号を**構成子** *(constructor)* と呼ぶ．被定義記号の全体を $\mathcal{D}_R$，構成子の全体を $\mathcal{C}_R$ で記す．各 $f \in \mathcal{D}_R$ に対し印付き記号を $f^\sharp$ を用意し，項 t の印付き項 $t^\sharp$ を，$t \equiv f(t_1, \ldots, t_n)$ かつ $f \in \mathcal{D}_R$ のときには $f^\sharp(t_1, \ldots, t_n)$ で，それ以外のときには t で定義する．

印付き項の対 $\langle u^\sharp, v^\sharp \rangle$ が R の**依存対** *(dependency pair)* であるとは，ある規則 $u \to C[v] \in R$ が存在して $root(v) \in \mathcal{D}_R$ となることである．以下では依存対 $\langle u^\sharp, v^\sharp \rangle$ を $u^\sharp \to v^\sharp$ で記す．また，R の依存対の全体を $DP(R)$ で記す．

例 6 以下で与えられる項書換え系 $R_{\mathtt{div}}$ を考える．$\mathtt{sub}, \mathtt{div}$ は自然数上の引き算と割り算 *(*厳

密には微妙に違う) に相当する.

$$\left\{\begin{array}{rcl} \mathtt{sub}(x,0) & \to & x \\ \mathtt{sub}(0,y) & \to & 0 \\ \mathtt{sub}(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{sub}(x,y) \end{array}\right\} \cup \left\{\begin{array}{rcl} \mathtt{div}(0,\mathtt{s}(y)) & \to & 0 \\ \mathtt{div}(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{div}(\mathtt{sub}(x,y),\mathtt{s}(y))) \end{array}\right\}$$

このとき $\mathcal{D}_{R_{\mathtt{div}}} = \{\mathtt{sub}, \mathtt{div}\}$ であり，$DP(R_{\mathtt{div}})$ は次のようになる.

$$DP(R_{\mathtt{div}}) = \left\{\begin{array}{rcl} \mathtt{sub}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{sub}^\sharp(x,y) \\ \mathtt{div}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{div}^\sharp(\mathtt{sub}(x,y),\mathtt{s}(y)) \\ \mathtt{div}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{sub}^\sharp(x,y) \end{array}\right.$$

定義 7 依存対の列 $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, u_2^\sharp \to v_2^\sharp, \cdots$ が項書換え系 R の**依存鎖** *(dependency chain)* であるとは，各 i で，$v_i^\sharp\theta_i \xrightarrow[R]{*} u_{i+1}^\sharp\theta_{i+1}$ かつ $u_i^\sharp\theta_i$ と $v_i^\sharp\theta_i$ が停止性を持つ代入 θ_i が存在することである.

定理 8 項書換え系 R の無限依存鎖が存在しないならば R は停止性を持つ.

証明: 対偶を示す．t を停止性を持たない項のうちサイズが最小の項とする．全ての t の真部分項は停止性を持つので，ある規則 $l_0 \to r_0' \in R$ と代入 θ_0 が存在して $t^\sharp \xrightarrow[R]{*} l_0^\sharp\theta_0$ かつ $l_0\theta_0$ の全ての真部分項は停止性を持ち $r'{}_0\theta_0$ は停止性を持たない．よって，集合 $\{r' \in Sub(r_0') \mid r'\theta_0$は停止性を持たない $\}$ は空でない．ここで，r_0 をこの集合中でサイズ最小の項とする．任意の $x \in Var(l_0)$ に対し $x\theta_0$ は $l_0\theta_0$ の真部分項なので停止性を持つ．よって，$r_0\theta_0$ の真部分項も全て停止性を持つ．$r_0\theta_0$ は停止性を持たないがその真部分項は全て停止性を持つので，$root(r_0) \in \mathcal{D}_R$ となる．よって，$l_0^\sharp \to r_0^\sharp$ は依存対.

上述の議論中の項 t を $r_0\theta_0$ として同様の議論を行なうことにより，ある依存対 $l_1^\sharp \to r_1^\sharp$ と代入 θ_1 が存在して，$r_0^\sharp\theta_0 \xrightarrow[R]{*} l_1^\sharp\theta_1$ かつ $r_1\theta_1$ は停止性を持たないが $l_1\theta_1$ と $r_1\theta_1$ の真部分項は全て停止性を持つ，ということを示すことができる．この操作を繰り返すことにより無限依存鎖 $l_0^\sharp \to r_0^\sharp, l_1^\sharp \to r_1^\sharp, \cdots$ が生成できる. □

定義 9 項書換え系 R の**依存グラフ** *(dependency graph)* とは有向グラフである．その頂点は R の依存対で定義され，$u_i^\sharp \to v_i^\sharp, u_j^\sharp \to v_j^\sharp$ が依存鎖になるとき $u_i^\sharp \to v_i^\sharp$ から $u_j^\sharp \to v_j^\sharp$ へ辺があると定義される．R の**再帰成分** *(recursion component)* とは R の依存グラフの強連結部分グラフ (任意の頂点から任意の頂点へ道がある部分グラフ) の頂点集合である．R の再帰成分の全体からなる集合を $RC(R)$ で記す．再帰成分 C が**非循環的** *(non-looping)* とは，C 中の依存対のみから構成され，C 中の各依存対をそれぞれ無限回含む無限依存鎖が存在しないことである.

系 10 項書換え系 R の全ての再帰成分が非循環的であるならば R は停止性を持つ.

例 11 例 6 で与えられた項書換え系 $R_{\mathtt{div}}$ に対し，その再帰成分は以下のようになる.

$$RC(R_{\mathtt{div}}) = \left\{\begin{array}{rcl} \{\mathtt{sub}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{sub}^\sharp(x,y)\} \\ \{\mathtt{div}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{div}^\sharp(\mathtt{sub}(x,y),\mathtt{s}(y))\} \end{array}\right\}$$

依存対法の解析結果である再帰成分が直感的な再帰構造と対応しているという事が，上述の例から見て取れる．以下では再帰成分の非循環性を示すために導入された部分項基準と簡約化対の概念を紹介する．

定義 12 項書換え系 R の再帰成分 C が**部分項基準** *(subterm criterion)* を満たすとは，$\mathcal{D}_R$ から空でない正整数列への関数 π が存在して以下が成立することである．

- ある $u^\sharp \to v^\sharp \in C$ で $u|_{\pi(root(u))} \rhd_{sub} v|_{\pi(root(v))}$ が成立．
- 全ての $u^\sharp \to v^\sharp \in C$ で，$u|_{\pi(root(u))} \trianglerighteq_{sub} v|_{\pi(root(v))}$ かつ $\varepsilon \prec q \prec \pi(root(v))$ となる各 q に対し $(v)_q \in C_R$．

定理 13 項書換え系 R の再帰成分 C が部分項基準を満たすならば C は非循環的である．

証明: C 中の依存対のみから構成され，C 中の各依存対をそれぞれ無限回含む無限依存鎖 $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, u_2^\sharp \to v_2^\sharp, \cdots$ が存在すると仮定し，$\theta_0, \theta_1, \ldots$ を $v_i^\sharp\theta_i \xrightarrow[R]{*} u_{i+1}^\sharp\theta_{i+1}$ かつ $u_i^\sharp\theta_i$ と $v_i^\sharp\theta_i$ が停止性を持つような代入とする．各 $\pi(root(u_i))$ を p_i で記す．各 i で，$v_i^\sharp\theta_i \xrightarrow[R]{*} u_{i+1}^\sharp\theta_{i+1}$ より $root(v_i) = root(u_{i+1})$．また，$\varepsilon \prec q \prec p_{i+1}$ となる各 q で $(v_i)_q \in C_R$ なので，$(v_i\theta_i)|_{p_{i+1}} \xrightarrow[R]{*} (u_{i+1}\theta_{i+1})|_{p_{i+1}}$．よって，各 i で $(u_i\theta_i)|_{p_i} \trianglerighteq_{sub} (v_i\theta_i)|_{p_{i+1}}$ となり，$(u_0\theta_0)|_{p_0} \trianglerighteq_{sub} (v_0\theta_0)|_{p_1} \xrightarrow[R]{} (u_1\theta_1)|_{p_1} \trianglerighteq_{sub} (v_1\theta_1)|_{p_2} \xrightarrow[R]{*} \cdots$ となる．ある $u^\sharp \to v^\sharp \in C$ で $u|_{\pi(root(u))} \rhd_{sub} v|_{\pi(root(v))}$ なので，この無限列は無限個の $\rhd_{sub}$ を含む．よって，$(u_0\theta_0)|_{p_0}$ から始まる $\xrightarrow[R]{} \cup \rhd_{sub}$ の無限減少列が存在．これは，関係 $\xrightarrow[R]{} \cup \rhd_{sub}$ が停止性を持つ項上で整礎であることに矛盾． □

例 14 項書換え系 $R_{\mathtt{div}}$ の引き算 (sub) に関する再帰成分 $\{\mathtt{sub}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathtt{sub}^\sharp(x, y)\}$ は，$\pi(\mathtt{sub}) = 1$ または $\pi(\mathtt{sub}) = 2$ とすることにより部分項基準を満たすため非循環的である．一方，割り算 (div) に関する再帰成分 $\{\mathtt{div}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathtt{div}^\sharp(\mathtt{sub}(x, y), \mathsf{s}(y))\}$ は部分項基準を満たさない．

定義 15 擬順序 $\gtrsim$ と狭義の半順序 $>$ の対 $(\gtrsim, >)$ は以下の条件を満たすとき**簡約化対** *(reduction pair)* であると言う．

- $\gtrsim$ は文脈と代入に閉じている
- $>$ は整礎で代入に閉じている
- $\gtrsim \cdot > \subseteq >$ または $> \cdot \gtrsim \subseteq >$ が成立．

特に，擬順序 $\gtrsim$ とそれから生成される狭義の半順序 $\gtrsim$ の対 $(\gtrsim, \gtrsim)$ が簡約化対であるとき，$\gtrsim$ を**弱簡約化順序** *(weak reduction order)* と呼ぶ．

定理 16 $R \cup C \subseteq \gtrsim$ かつ $C \cap > \neq \emptyset$ となる簡約化対 $(\gtrsim, >)$ が存在するならば，項書換え系 R の再帰成分 C は非循環的である．

証明: C 中の依存対のみから構成され，C 中の各依存対をそれぞれ無限回含む無限依存鎖 $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, u_2^\sharp \to v_2^\sharp, \cdots$ が存在すると仮定し，$\theta_0, \theta_1, \ldots$ を $v_i^\sharp\theta_i \xrightarrow[R]{*} u_{i+1}^\sharp\theta_{i+1}$ となる代入とする．$\gtrsim$ は文脈と代入に閉じていて $R \cup C \subseteq \gtrsim$ なので，各 i で $u_i^\sharp\theta_i \gtrsim v_i^\sharp\theta_i \gtrsim u_{i+1}^\sharp\theta_{i+1}$．$>$ は代入に閉じていて $C \cap > \neq \emptyset$ なので，無限個の i で $u_i^\sharp\theta_i > v_i^\sharp\theta_i$．よって，$\gtrsim \cdot > \subseteq >$ または

$> \cdot \gtrsim \subseteq >$ なので $>$ の無限減少列が存在することになり $>$ の整礎性に矛盾. □

(a) 引数切り落とし法

弱簡約化順序を設計する代表的な手法である引数切り落とし法を紹介する.

定義 17 引数切り落とし関数 *(argument filtering function)* とは, 各関数記号 f に対し $\pi(f)$ が正整数のリスト $[i_1, \ldots, i_m]$ か正整数 i をとる関数である. ただし, 正整数 $i, i_1, \ldots, i_m$ は $arity(f)$ 以下であるとする. 切り落とし関数 π は以下のように項上へ拡張される.

$$\begin{cases} \pi(x) = x \\ \pi(f(t_1, \ldots, t_n)) = \pi(t_i) & \text{if } \pi(f) = i \\ \pi(f(t_1, \ldots, t_n)) = f(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) & \text{if } \pi(f) = [i_1, \ldots, i_m] \end{cases}$$

$s \gtrsim_\pi t$ を $\pi(s) \geq \pi(t)$ で定義する.

定理 18 任意の簡約化順序 $>$ と引数切り落とし関数 π に対し, $\gtrsim_\pi$ は弱簡約化順序になる.

証明: (I) $\gtrsim_\pi$ が擬順序であることは明らか. (II) $\gtrsim_\pi$ が整礎性を持たないと仮定すると $>$ の整礎性に矛盾する. (III) $\pi(C)[\pi(t)] \equiv \pi(C[t])$ は $C[\,]$ に関する帰納法で示せる. この性質を用いることにより $\gtrsim_\pi$ が文脈に閉じていることが導ける. (IV) θ_π によって $\theta_\pi(x) = \pi(\theta(x))$ で定義される代入を表すとすると, $\pi(t)\theta_\pi \equiv \pi(t\theta)$ は t に関する帰納法で示せる. この性質を用いることにより $\gtrsim_\pi$ と $\gtrsim_\pi$ が代入に閉じていることが導ける. □

例 19 項書換え系 $R_{\mathtt{div}}$ の引き算に関する再帰成分の非循環性はすでに示した (例 14). 引数切り落とし関数を $\pi(\mathtt{sub}) = [1]$ かつ他の $f \in \Sigma$ に対しては $\pi(f) = [1, \ldots, arity(f)]$ で与え, 優先順位を $\mathtt{div} \rhd \mathtt{s} \rhd \mathtt{sub}$ で与えると, $R_{\mathtt{div}} \subseteq \gtrsim^{\pi}_{lpo}$ かつ

$$\pi(\mathtt{div}^\sharp(\mathtt{s}(x), \mathtt{s}(y))) = \mathtt{div}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) >_{lpo} \mathtt{div}^\sharp(\mathtt{sub}(x), \mathtt{s}(y)) = \pi(\mathtt{div}^\sharp(\mathtt{sub}(x, y), \mathtt{s}(y)))$$

となるので, 割り算 (div) に関する再帰成分も非循環性を持つ. よって, $R_{\mathtt{div}}$ が停止性を持つ事が示される.

(b) 実効規則

弱簡約化順序 $\gtrsim$(または簡約化対 $\langle >, \gtrsim \rangle$) を用いて項書換え系 R の再帰成分 C の非循環性を示す際には, $R \subseteq \gtrsim$ のように再帰成分とは関係なさそうな規則も含めて全ての規則を $\gtrsim$ で順序付けなければならない. この非効率性を除去するために導入されたのが実効規則の概念である.

定義 20 R を項書換え系とする. $f >_{\mathtt{def}} g$ を, ある $l \to r \in R$ が存在して $root(l) = f$ かつ g が r 中に出現として定義する. 項 t の**実効規則** *(usable rule)* の集合 $\mathcal{U}(t)$ を以下で定義する.

$$\mathcal{U}(t) = \{l \to r \in R \mid \text{ある } t \text{ 中に出現する } f \in \mathcal{D}_R \text{ に対し } f >^*_{\mathtt{def}} root(l)\}$$

再帰成分 C の**実効規則**の集合 $\mathcal{U}(C)$ を $\bigcup_{u^\sharp \to v^\sharp \in C} \mathcal{U}(v^\sharp)$ で定義する.

項書換え系 $R_{\mathtt{div}}$ の div に関する再帰成分 $\{\mathtt{div}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathtt{div}^\sharp(\mathsf{sub}(x,y),\mathsf{s}(y))\}$ を C とすると，$\mathcal{U}(C)$ は sub に関する三つの規則となり，div に関する二つの規則を含まない．これは，割り算 div の再帰が減少関数 sub に基づき定義されているという直感に一致する．

残念ながら，定理 16 の条件 $R \subseteq \gtrsim$ を条件 $\mathcal{U}(C) \subseteq \gtrsim$ に単純に置き換えただけでは C の非循環性を保証することが一般にはできない．ここで，c を Σ に含まれない $\mathit{arity}(\mathsf{c}) = 2$ である新しい関数記号とし，項書換え系 C_e を $\{\mathsf{c}(x,y) \to x,\ \mathsf{c}(x,y) \to y\}$ で定義する．このとき，項書換え系 $R = \{\mathtt{f}(0,1,x) \to \mathtt{f}(x,x,x)\}$ は停止性を持ち，その再帰成分 $\{\mathtt{f}^\sharp(0,1,x) \to \mathtt{f}^\sharp(x,x,x)\}$ は非循環的である．しかしながら，この再帰成分は $R \cup C_e$ においては非循環的でない：$\mathtt{f}^\sharp(\mathsf{c}(0,1),\mathsf{c}(0,1),\mathsf{c}(0,1)) \underset{C_e}{\to} \mathtt{f}^\sharp(0,\mathsf{c}(0,1),\mathsf{c}(0,1)) \underset{C_e}{\to} \mathtt{f}^\sharp(0,1,\mathsf{c}(0,1)) \underset{R}{\to} \mathtt{f}^\sharp(\mathsf{c}(0,1),\mathsf{c}(0,1),\mathsf{c}(0,1))$．このように，再帰成分と一見関係なさそうな規則が非循環性を破壊する場合がある．

実は，このような問題点に対して C_e が本質的であるという事，すなわち，定理 16 の条件 $R \subseteq \gtrsim$ が条件 $\mathcal{U}(C) \cup C_e \subseteq \gtrsim$ に置き換え可能なことが知られている．

定理 21 $\mathcal{U}(C) \cup C_e \cup C \subseteq \gtrsim$ かつ $C \cap > \neq \emptyset$ となる簡約化対 $(\gtrsim, >)$ が存在するならば，項書換え系 R の再帰成分 C は非循環的である．

証明: $\Delta = \{\mathit{root}(l) \mid l \to r \in R \setminus \mathcal{U}(C)\}$ とする．停止性を持つ項 t に対し $I(t)$ を以下で定義．

$$I(t) = \begin{cases} t & \mathit{if}\ t \in \mathcal{V} \\ f(I(t_1),\ldots,I(t_n)) & \mathit{if}\ t \equiv f(t_1,\ldots,t_n)\ \text{かつ}\ f \notin \Delta \\ \mathsf{c}(f(I(t_1),\ldots,I(t_n)), \mathit{Red}(\{I(t') \mid t \underset{R}{\to} t'\})) & \mathit{if}\ t \equiv f(t_1,\ldots,t_n)\ \text{かつ}\ f \in \Delta \end{cases}$$

ここで，$\mathit{Red}(T)$ は，$\mathit{Red}(\emptyset) = \bot$ かつ，$T \neq \emptyset$ のときは $\mathit{Red}(T) = \mathsf{c}(\mathit{least}(T), \mathit{Red}(T \setminus \{\mathit{least}(T)\}))$ で定義し，$\bot$ を Σ に含まれない $\mathit{arity}(\bot) = 0$ である新しい関数記号とする．なお整列可能定理 (任意の集合上に整礎な全順序が存在するという定理) より，任意の空でない項の集合 T の最小元 $\mathit{least}(T)$ を与えている．ここで，関係 $\underset{R}{\to} \cup \rhd_{sub}$ は停止性を持つ項上で整礎であり，R の有限性より集合 $\{I(t') \mid t \underset{R}{\to} t'\}$ も有限であるので，解釈 I は矛盾なく定まっている事に注意．

停止性をもつ代入 θ に対し，θ^I を $\theta^I(x) = I(\theta(x))$ で定義すると，以下の性質が成立する．

(1) $t\theta$ が停止性を持つような項 t と代入 θ に対し，$I(t\theta) \underset{C_e}{\overset{*}{\to}} t\theta^I$ が成立．

(2) $r\theta$ が停止性を持つような $l \to r \in C \cup \mathcal{U}(C)$ と代入 θ に対し，$I(r\theta) \equiv r\theta^I$ が成立．

(3) s が停止性を持ち $s \underset{R}{\to} t$ ならば $I(s) \underset{\mathcal{U}(C) \cup C_e}{\overset{+}{\to}} I(t)$.

(1) t に関する帰納法で示せる．(2) 任意の $t \in \mathit{Sub}(r)$ で $I(t\theta) \equiv t\theta^I$ となることが t に関する帰納法で示せる．(3) $s \underset{R}{\to} t$ より，ある規則 $l \to r \in R$ と文脈 $C[\,]$ と代入 θ が存在して $s \equiv C[l\theta]$ かつ $t \equiv C[r\theta]$．ここで，$C[\,]$ に関する帰納法で (3) は示せる．

C 中の依存対のみから構成され，C 中の各依存対をそれぞれ無限回含む無限依存鎖 $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, u_2^\sharp \to v_2^\sharp, \cdots$ が存在すると仮定し，$\theta_0, \theta_1, \ldots$ を $v_i^\sharp\theta_i \underset{R}{\overset{*}{\to}} u_{i+1}^\sharp\theta_{i+1}$ かつ $u_i^\sharp\theta_i$ と $v_i^\sharp\theta_i$ が停止性を持つような代入とする．任意の i に対し，上述の (1),(2),(3) より $v_i^\sharp\theta_i^I \equiv I(v_i^\sharp\theta_i) \underset{\mathcal{U}(C) \cup C_e}{\overset{*}{\to}} I(u_{i+1}^\sharp\theta_{i+1}) \underset{C_e}{\overset{*}{\to}} u_{i+1}^\sharp\theta_{i+1}^I$．よって，$\mathcal{U}(C) \cup C_e \cup C \subseteq \gtrsim$ より $v_i^\sharp\theta_i^I \gtrsim u_{i+1}^\sharp\theta_{i+1}^I \gtrsim v_{i+1}^\sharp\theta_{i+1}^I$．また，$C \cap > \neq \emptyset$ なので無限個の j で $u_j^\sharp\theta_j^I > v_j^\sharp\theta_j^I$ が成立．これは $>$ の整礎性に矛盾． □

1-3-3 合流性

項書換え系 R が**合流性** (confluence property) を持つとは，$t_1 \overset{*}{\underset{R}{\leftarrow}} t \overset{*}{\underset{R}{\rightarrow}} t_2 \Rightarrow t_1 \downarrow t_2$，が成立することである．ここで，$t_1 \downarrow t_2$ は $\exists u.\, t_1 \overset{*}{\underset{R}{\rightarrow}} u \overset{*}{\underset{R}{\leftarrow}} t_2$ で定義される．また，項書換え系 R が**局所合流性** (locally confluence property) を持つとは $t_1 \underset{R}{\leftarrow} t \underset{R}{\rightarrow} t_2 \Rightarrow t_1 \downarrow t_2$ が，**ダイヤモンド性** (diamond property) を持つとは $t_1 \underset{R}{\leftarrow} t \underset{R}{\rightarrow} t_2 \Rightarrow \exists u.\, t_1 \underset{R}{\rightarrow} u \underset{R}{\leftarrow} t_2$ が成立することである．これらの性質の関係は以下のようになっている．

定理 22 項書換え系 R に対して以下が成立．

(1) R がダイヤモンド性を持つならば合流性を持つ
(2) R が合流性を持つならば局所合流性を持つ

証明: (1) $t_1 \overset{m}{\underset{R}{\leftarrow}} t \overset{n}{\underset{R}{\rightarrow}} t_2$ ならば $\exists u.\, t_1 \overset{n}{\underset{R}{\rightarrow}} u \overset{m}{\underset{R}{\leftarrow}} t_2$ を $m+n$ に関する帰納法で証明すれば良い．(2) 定義より明らか． □

(1),(2) 共に逆は成立しない．例えば，$R = \{\mathsf{a} \to \mathsf{b}\}$ は合流性を持つがダイヤモンド性を持たず，$R = \{\mathsf{c} \to \mathsf{g(c)}, \mathsf{g}(x) \to \mathsf{f}(x, \mathsf{g}(x)), \mathsf{f}(x, x) \to \mathsf{e}\}$ は局所合流性を持つが合流性を持たない．

本節では，局所合流性と密接な関係がある危険対の概念を紹介し，停止性を持つ場合は危険対を用いて効果的に合流性が判定できることと，直交性と呼ばれる性質を持つ項書換え系が合流性を持つことを紹介する．

(1) 危険対

合流性を証明する際に重要となる危険対の概念と，危険対と局所合流性の関係を紹介する．最初に危険対の定義に必要な単一化の概念を紹介する．

定義 23 項 s と t が**単一化可能** *(unifiable)* とは，ある代入 θ が存在して $s\theta \equiv t\theta$ となることである．このとき，θ を項 s と t の**単一化子** *(unifier)* と呼ぶ．項 s と t の単一化子 θ が**最汎単一化子** *(most general unifier)* であるとは，任意の単一化子 θ' に対してある代入 θ'' が存在して $\theta' = \theta'' \cdot \theta$ となることである．

なお，任意の単一化可能な項 s と t に対し，最汎単一化子が (変数の名前変えを除いて) 一意に存在する事が知られている．

定義 24 $l_1 \to r_1$ と $l_2 \to r_2$ を書換え規則とする．ここで，これらの書換え規則は変数を共有しないと仮定して良い．ある文脈 $C[\,]$ が存在して $l_2 \equiv C[l_2']$ かつ l_2' は変数でなく l_1 と l_2' が単一化可能であるとき，その最汎単一化子を θ とすると，項の対 $\langle C[r_1]\theta, r_2\theta\rangle$ を $l_1 \to r_1$ の $l_2 \to r_2$ に対する**危険対** *(critical pair)* と呼ぶ．

項書換え系 R の**危険対**とは R の二つの書換え規則 (同一の書換え規則をとってきても良い) の間の危険対である．取り出した 2 つの書換え規則が変数を共有していた場合は変数の名前変えを行っても良い．ただしこの定義では，任意の書換え規則 $l \to r$ に対し，明らかに $l \to r$ は $l \to r$ に根位置で重なり危険対 $\langle r, r\rangle$ を持つことになるが，この場合は特例として危険対とは呼ばないことにする．項書換え系 R の危険対全体の集合を $CP(R)$ で記す．

項書換え系 R_1, R_2 における危険対の集合 $CP(R_1), CP(R_2)$ は以下のようになる．

$$R_1 = \left\{ \begin{array}{lcl} \mathtt{f}(\mathtt{g}(x)) & \to & \mathtt{f}'(x) \\ \mathtt{g}(\mathtt{f}(x)) & \to & \mathtt{g}'(x) \end{array} \right. \qquad CP(R_1) = \left\{ \begin{array}{l} \langle \mathtt{f}(\mathtt{g}'(x)), \mathtt{f}'(\mathtt{f}(x)) \rangle \\ \langle \mathtt{g}(\mathtt{f}'(x)), \mathtt{g}'(\mathtt{g}(x)) \rangle \end{array} \right.$$

$$R_2 = \left\{ \begin{array}{lcl} \mathtt{f}(\mathtt{f}(x)) & \to & \mathtt{g}(x) \end{array} \right. \qquad CP(R_2) = \left\{ \begin{array}{l} \langle \mathtt{f}(\mathtt{g}(x)), \mathtt{g}(\mathtt{f}(x)) \rangle \end{array} \right.$$

定理 25 項書換え系 R が局所合流性を持つことと，全ての R の危険対 $\langle u, v \rangle$ が $u \downarrow v$ を満たすことは必要十分である．

証明: ($\Rightarrow$) は明らか．($\Leftarrow$) を示す．以下の 3 つの場合を示せば十分である．(1) $t_1 \equiv C[r_1\theta_1, l_2\theta_2] \underset{R}{\leftarrow} C[l_1\theta_1, l_2\theta_2] \underset{R}{\rightarrow} C[l_1\theta_1, r_2\theta_2] \equiv t_2$ の場合は，$t_1 \underset{R}{\rightarrow} C[r_1\theta_1, r_2\theta_2] \underset{R}{\leftarrow} t_2$ となり成立．(2) $t_1 \underset{R}{\leftarrow} l\theta \underset{R}{\rightarrow} r\theta$ かつ $\langle t_1, r\theta \rangle$ がある危険対 $\langle u, v \rangle$ の例 ($\exists\sigma.\ t_1 \equiv u\sigma \wedge r\theta \equiv v\sigma$) となっている場合は，仮定より $t_1 \downarrow r\theta$．(3) $t_1 \underset{R}{\leftarrow} l\theta \underset{R}{\rightarrow} r\theta$ かつ $\langle t_1, r\theta \rangle$ が危険対 $\langle u, v \rangle$ の例になっていない場合．このとき，ある変数 $x \in Var(l)$ が存在して，$t_1 \underset{R}{\leftarrow} l\theta$ の書換えは $l\theta$ のある部分項 $x\theta$ の内側で行なわれている．よって，x が l 中に複数出現する場合も考えて，ある代入 θ' が存在して $l\theta \underset{R}{\rightarrow} t_1 \underset{R}{\overset{*}{\rightarrow}} l\theta'$ かつ $\forall x \in Var(l).\ x\theta \underset{R}{\overset{*}{\rightarrow}} x\theta'$ となる．よって，$t_1 \underset{R}{\overset{*}{\rightarrow}} l\theta' \underset{R}{\rightarrow} r\theta' \underset{R}{\overset{*}{\leftarrow}} r\theta$． □

(2) 合流性

危険対を用いると局所合流性の判定が行なえるが，合流性の判定を行なうためには他の適切な条件も必要となる．これは，局所合流性を持つが合流性を持たない項書換え系が存在するためである (定理 22 直下の例を参照)．ここでは，停止性を持つ場合には効果的に合流性が判定できることと，停止性を持たない場合でも直交項書換え系と呼ばれる項書換え系は合流性を持つことを紹介する．

定理 26 項書換え系 R が停止性を持つとする．このとき，R が合流性を持つことと，全ての R の危険対 $\langle u, v \rangle$ が $u \downarrow v$ となることは必要十分である．

証明: ($\Rightarrow$) 明らか．($\Leftarrow$) $t_1 \underset{R}{\overset{*}{\leftarrow}} t \underset{R}{\overset{*}{\rightarrow}} t_2 \Rightarrow t_1 \downarrow t_2$ を t に関する帰納法で示す．ここで，比較は $\underset{R}{\rightarrow}$ で行なう．この帰納法の正当性は R の停止性が保証している．$t \equiv t_1$ または $t \equiv t_2$ のときは明らか．$t \underset{R}{\rightarrow} t_i' \underset{R}{\overset{*}{\rightarrow}} t_i$ $(i = 1, 2)$ とする．局所合流性より，ある u が存在して $t_1' \underset{R}{\overset{*}{\rightarrow}} u \underset{R}{\overset{*}{\leftarrow}} t_2'$．$t_1 \underset{R}{\overset{*}{\leftarrow}} t_1' \underset{R}{\overset{*}{\rightarrow}} u$ に対する帰納法の仮定より，ある v が存在して $t_1 \underset{R}{\overset{*}{\rightarrow}} v \underset{R}{\overset{*}{\leftarrow}} u$．$v \underset{R}{\overset{*}{\leftarrow}} t_2' \underset{R}{\overset{*}{\rightarrow}} t_2$ に対する帰納法の仮定より，ある w が存在して $v \underset{R}{\overset{*}{\rightarrow}} w \underset{R}{\overset{*}{\leftarrow}} t_2$．よって，$t_1 \downarrow t_2$． □

定義 27 項 t が**線形** *(linear)* であるとは，項 t の中に 2 回以上出現する変数がないことである．項書換え系 R の全ての規則の左辺が線形のとき**左線形** *(left-linear)* であると言う．危険対を一つも持たない左線形項書換え系を**直交項書換え系** *(orthogonal term rewriting system)* と呼ぶ．

定理 28 直交項書換え系 R は合流性を持つ．

証明: 並列書換え $s \twoheadrightarrow t$ を，ある $C[,\ldots,]$ と $l_i \to r_i$ と θ_i $(i = 1, \ldots, n)$ が存在して $s \equiv C[l_1\theta_1, \ldots, l_n\theta_n]$ かつ $t \equiv C[r_1\theta_1, \ldots, r_n\theta_n]$，として定義する．$\twoheadrightarrow$ がダイヤモンド性を持つこと，すなわち，$t_1 \twoheadleftarrow t \twoheadrightarrow t_2$ ならば $\exists u.\ t_1 \twoheadrightarrow u \twoheadleftarrow t_2$ を示す．これが示せると定理 22 より $\twoheadrightarrow$

が合流性を持つので，明らかに $\underset{R}{\rightarrow}$ も合流性を持つ．

最初に $t \twoheadrightarrow t_2$ が根位置での書換えだった場合，すなわち $t \equiv l\theta \rightarrow r\theta \equiv t_2$ の場合を考える．$t_1 \twoheadleftarrow t$ も根位置での書換えだった場合は直交性より同じ規則が利用されているので $t_1 \equiv t_2$ となる．並列書換え関係は反射性を持つので $t_1 \twoheadrightarrow t_1 \equiv t_2 \twoheadleftarrow t_2$ となり題意成立．そうでない場合は，直交性より $t_1 \equiv l\theta' \twoheadleftarrow l\theta$ とおける．よって，$t_1 \equiv l\theta' \twoheadrightarrow r\theta' \twoheadleftarrow r\theta \equiv t_2$ となり題意成立．

次に $t \twoheadrightarrow t_2$ が根位置での書換えでなかった場合を考える．$t_1 \twoheadleftarrow t$ が根位置での書換えだった場合は上記と同様に証明できる．$t_1 \twoheadleftarrow t$ も根位置での書換えでないとする．このとき，一般に $t \equiv C[s_1, \ldots, s_n]$ かつ $t_1 \equiv C[u_1, \ldots, u_n] \twoheadleftarrow C[s_1, \ldots, s_n] \twoheadrightarrow C[v_1, \ldots, v_n] \equiv t_2$ と書け，さらに各 i で $u_i \twoheadleftarrow s_i \twoheadrightarrow v_i$ かつ $u_i \twoheadleftarrow s_i$ と $s_i \twoheadrightarrow v_i$ のどちらか一方は根位置での書換えであると見做せる．よって，上述の議論と同様にして $\exists s_i'.\ u_i \twoheadrightarrow s_i' \twoheadleftarrow v_i$ が示せる．よって，$t_1 \equiv C[u_1, \ldots, u_n] \twoheadrightarrow C[s_1', \ldots, s_n'] \twoheadleftarrow C[v_1, \ldots, v_n] \equiv t_2$ となり題意が成立． □

■参考文献

1) F. Baader and T. Nipkow, “Term Rewriting and All That,” Cambridge University Press, 1998.
2) E. Ohlebusch, “Advanced Topics in Term Rewriting,” Springer-Verlag, 2002.
3) Terese, “Term Rewriting Systems,” Cambridge Tracts in Theoretical Comput. Sci., vol.55, Cambridge University Press, 2003.

■7 群 - 1 編 - 1 章

1-4 代数仕様

（執筆者：緒方和博・中村正樹・二木厚吉）[2008 年 6 月 受領]

代数仕様は，仕様を数学的に記述した形式仕様の一種であり，対象システムを分類された「もの」（object）の上にいくつか演算（operation）が定義されたシステムとしてモデル化・記述した仕様である．分類されたものの種類の名前を**ソート**（sort）という．例えば，命令型プログラミング言語の代数仕様[7]では，関連するソートして，記憶 Store，プログラム Pgm，変数名 Var，整数 Int などがある．関連する演算として，変数への値の代入 `assignment : Var Int -> Pgm` や，変数に代入されている値の取り出し `val : Store Var -> Int`，プログラム実行による記憶の更新 `apply : Store Prg -> Store` などが考えられる．図 1·10 は，ソートと演算の関係を描いた図で，ADJ 図と呼ばれる．ソートや演算が満たすべき性質は**公理**（axiom）で記述される．代数とは，台集合の上にいくつかの関数が定義されたシステムである．ソートに対応する台集合，演算に対応する関数をもち，公理を満たす代数が，代数仕様のモデル（実装）となる．仕様が満たすべき性質を演算の組合せで表現し，代数仕様がその性質を満たすことを形式的に検証することで，実装前の要求，設計段階における早期の欠陥修正が可能となる．

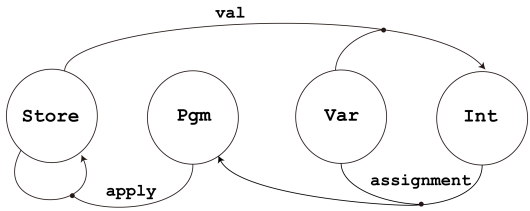


図 1·10　ADJ 図

1-4-1 代数仕様言語の歴史

代数仕様は，70 年代前半に行われた ADJ グループ*による圏論の計算機科学への応用及び抽象データ型の始代数意味論の研究に端を発している．70 年代後半に，これらを下敷に代数仕様言語 Clear が設計された[1, 2]．Clear は，厳密な意味論を有する最初の仕様言語であり，モジュールの概念を最初に取入れた計算機言語である．Clear のモジュールシステムは，仕様言語のみならず，Ada，ML 及び C++といったプログラミング言語にも影響を与えた．80 年代にかけて，Clear で提案されたモジュールシステムの最初の実装として，HISP[6]，OBJ (OBJ2, OBJ3)[5, 8]などの処理系が実装された．OBJ は，多くの後続の代数仕様言語（ACT ONE，Maude，CafeOBJ，CASL など）に影響を与えた．始代数意味論に基づく代数仕様は静的なデータしか扱えないが，90 年代には，動的なシステムを扱うための手法として書換え論理や隠蔽代数が提案された．

本節では，代数仕様言語 CafeOBJ を用いて代数仕様について解説する．CafeOBJ は，Institution 理論に基づき，順序ソート代数（「もの」と演算のモデル），隠蔽代数（振る舞いの

* J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright

モデル），擬順序代数（状態遷移のモデル）及びそれらの組合せを意味する仕様を記述できるという特徴をもつ[4, 3, 9]．まず順序ソート代数によるデータ仕様の構文と意味，検証手法を示し，最後に隠蔽代数，擬順序代数に基づくシステム仕様を解説する．

1-4-2 仕　様

CafeOBJ 仕様はモジュール単位で記述される．モジュールは，ほかのモジュールの輸入，シグネチャ，公理から構成される．例えば，ENAT という名前のモジュールは，mod! ENAT{ I_{ENAT} Σ_{ENAT} E_{ENAT} } のように書かれる．大括弧内の I_{ENAT}, Σ_{ENAT}, E_{ENAT} がそれぞれ輸入，シグネチャ，公理である．この例では輸入 I_{ENAT} は空とする． シグネチャは，ソートと演算の

```
1  [Zero NzNat < Nat]          4  op p_ : Nat -> Nat
2  op 0 : -> Zero              5  op _=_ : Nat Nat -> Bool
3  op s_ : Nat -> NzNat
```

図 1･11　シグネチャ Σ_{ENAT}

宣言で構成される．モジュール ENAT のシグネチャ Σ_{ENAT} を図 1･11 に示す．1 行目の角括弧内に，ソート Zero, NzNat 及び Nat が順序とともに宣言されている．2-5 行目には宣言したソート上に演算がキーワード op で宣言されている．定数は 0 のように引数なしの演算として宣言される．すべての CafeOBJ モジュールはブール代数の組込みモジュール BOOL を暗黙に輸入するため，ソート Bool が使用可能である．下線部 (_) は引数の位置の指定である．ENAT はペアノ流の自然数の仕様であり，0 と s_が自然数を構築する演算で，p_と _=_が自然数上の関数を表す演算である． 公理は，仕様のモデルが満たすべき性質である．等式仕様で

```
1  vars X Y : Nat              4  eq ( 0 =  0) = true .
2  eq p 0 = 0 .                5  eq ( 0 = s X) = false .
3  eq p s X = X .              6  eq (s X =  0) = false .
                               7  eq (s X = s Y) = (X = Y) .
```

図 1･12　公理 E_{ENAT}

は，等式の集合が公理となる．モジュール ENAT の公理 E_{ENAT} を図 1･12 に示す．1 行目に，公理中で用いる変数が宣言されている．2-3 行目及びに 4-7 行目にそれぞれ演算 p，_=_に関する等式がキーワード eq で宣言されている．true，false は組込みモジュール BOOL の真偽値の定数である．

1-4-3 モデル

代数仕様のモデルとは，ソートに対応する台集合と演算に対応する関数から構成される代数で，公理をすべて満たし，更に各モジュールで宣言された意味論及び輸入の条件を満たすものである．

(1) 代数

シグネチャ Σ に対して，Σ-代数 M は次の (1) (2) で構成される代数である．(1) 各ソート s に対応する台集合 M_s をもつ．ただし，$s < s'$ のとき $M_s \subseteq M_{s'}$ を満たす．(2) 各演算の宣

言 op f : $s_1 \cdots s_n$ -> s に対応する関数 $M_f : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ をもつ．公理 E のすべての等式を満たす Σ-代数を，(Σ, E)-代数と呼ぶ．$(\Sigma_{\texttt{ENAT}}, E_{\texttt{ENAT}})$-代数 N として自然数の集合 $\mathcal{N}$ を台集合にもつ代数を与える．各ソートに対する台集合を $N_{\texttt{Zero}} \triangleq \{0\}$, $N_{\texttt{NzNat}} \triangleq \mathcal{N} - \{0\}$, $N_{\texttt{Nat}} \triangleq \mathcal{N}$ とし，各演算に対する関数を $N_{\texttt{0}} \triangleq 0$, $N_{\texttt{s}}(x) \triangleq x+1$, $N_{\texttt{p}}(x) \triangleq x-1$（ただし，$x = 0$ のときは 0），$N_{=}(x,y) \triangleq (x = y)$ とする*．すると，代数 N は公理 $E_{\texttt{ENAT}}$ のすべての等式を満たす．

(2) 意味・モデル

モジュールで規定されたシグネチャ，公理を必要十分に満たす (Σ, E)-代数を，(Σ, E)-始代数という．すなわち，(Σ, E)-始代数 M では，(1) すべての台集合の要素 $e \in M_s$ に対し，e に解釈される基底項（変数をもたない項）t が存在し，(2) 公理の等式から導出可能な等式のみが等しく解釈される．代数を構成する要素の名前の違いを無視すれば，(Σ, E)-始代数はユニークに定まる．モジュール *Mod* が `mod!`で始まるとき，*Mod* はきつい意味をもつといい，(Σ, E)-始代数すべての集合を意味する．`mod*`で始まるとき，*Mod* はゆるい意味をもつといい，(Σ, E)-代数すべての集合を意味する*．*Mod* の意味に含まれる Σ-代数を *Mod* のモデルと呼ぶ．仕様 `mod! ENAT{...}` はきつい意味をもつ．$\Sigma_{\texttt{ENAT}}$-代数 N は，$(\Sigma_{\texttt{ENAT}}, E_{\texttt{ENAT}})$-始代数である．したがって N は仕様 `ENAT` のモデルである．一つのソートのみからなるモジュール `mod* TRIV{[Elt]}` は，典型的なゆるい意味をもつモジュールである．任意の代数が `TRIV` のモデルとなる．

1-4-4 検　証

検証作業の核となるのは，項書換えシステムに基づく等式推論である．CafeOBJ 処理系は，対話型のシステムであり，モジュールの読込みや項の簡約コマンドなどを備えている．例えば，モジュール `ENAT` における項 `s s s 0 = s s 0` の簡約は，モジュール `ENAT` を選択した状態で，処理系に `red s s s 0 = s s 0 .` と入力することで行われる．結果，項書換えシステムに基づく簡約が行われ，`false` が出力される．仕様に定数や等式を付け加えて簡約を行うことで，場合分けや帰納法などの証明技術を用いた検証作業が可能となる．そのような操作をまとめたものを証明譜と呼ぶ．図 1･13 に証明譜の例を示す． モジュールを `open`

```
1  open ENAT                    4    eq (a = a) = true .
2    red 0 = 0 .                5    red s a = s a .
3    op a : -> Nat .            6  close
```

図 1･13　証明譜

することで，そのモジュールに演算や等式の追加が可能となる（1 行目）．まず `0 = 0` の簡約を指示し（2 行目），次に定数 `a` に対して等式 `(a = a) = true` を宣言（3-4 行目），最後に `s a = s a` の簡約を指示している（5 行目）．この証明譜は，任意の `0` と `s_`から構成される項 A に対して述語 $P(A)$ =(`A = A`) が成り立つことの帰納法による検証を表す．等式の宣言

* $N_{=}(x,y) \triangleq (x = y)$ の右辺 $(x = y)$ の等号は自然数上の等式を表す等号である．
* 正確には，これは輸入を含まないモジュールの宣言的意味である．輸入を含むモジュールの意味は輸入に関する条件も満たす必要がある．詳細は関連文献 4) を参照．

(4 行目) が帰納法の仮定である．証明譜を処理系に入力すると，両方の簡約が true を返すため，帰納法による検証が成功したことになる．任意の自然数 $n \in \mathcal{N}$ に対し，n に解釈される項 $\mathtt{s}^n(\mathtt{0})$ が存在するため，この証明譜（の実行結果）は $\forall n \in \mathcal{N}.\, N_{=}(n, n)$ の証明となる．

1-4-5 振舞仕様

隠蔽代数をモデルにもつ振舞仕様について述べる．振舞仕様は，システムの状態を陽に記述せず，観測・操作を通してシステムの振る舞いを記述する仕様である．車庫の入出庫システムの振舞仕様 BH-GARAGE を示す．振舞仕様 BH-GARAGE を mod* BH-GARAGE{ I_{BHG} Σ_{BHG} E_{BHG} } とする．振舞仕様は，記述した振る舞いを満たすすべての実装をモデルとするため，ゆるい意味で宣言される．ここで I_{BHG} はモジュール ENAT の保護モードでの輸入 pr(ENAT) とする．保護モードは，仕様をそのままのかたちで利用する際に用いる．BH-GARAGE では，ソート Nat の項を車の識別子と入庫可能数の両者に用いる．

振舞仕様には，隠蔽ソートと呼ばれる特別なソートと振舞演算と呼ばれる特別な演算がある．隠蔽ソートの項の観測，操作は，振舞演算ごしにのみ行われる，という意味で，隠蔽という名前がつけられている．隠蔽ソートを引数に唯一つもつ演算のみが振舞演算になりうる．返り値のソートが隠蔽ソートの振舞演算を操作と呼び，可視ソート（隠蔽ソート以外）の振舞演算を観測と呼ぶ． 図 1･14 に，BH-GARAGE のシグネチャ Σ_{BHG} を示す．1 行目に，隠蔽

1	*[State]*	4	bops in out : Car State -> State
2	bop count : State -> Nat	5	ops c-in c-out : Nat State -> Bool
3	bop parked : Car State -> Bool	6	op init : -> State

図 1･14　シグネチャ Σ_{BHG}

ソートが宣言されている．ほかのソートと区別するため，特殊な角括弧*[ ]*で囲む．隠蔽ソート State でシステムの状態空間を表現する．2-3 行目に，観測 count, parked の宣言が振舞演算のキーワード bop で宣言されている．count はある状態での入庫可能数，parked はある状態で車が入庫しているかどうかを観測する．4 行目は操作 in, out の宣言である*．それぞれ入庫，出庫（の試み）を表す．5 行目は，入出庫の条件を表す演算である．6 行目は，初期状態の宣言である．例えば，項 parked(s 0,init) は初期状態で車 s 0 が駐車中かどうかを表す．項 in(0,s) は，状態 s で車 0 が入庫を試みた後の状態を表す．

振舞仕様では，システムの状態を陽に記述しないため，初期状態や操作は観測ごしに定義される．また隠蔽ソート上の等価関係は振舞等式と呼ばれる特別な等式で記述する． 図 1･15 に，BH-GARAGE の公理 E_{BHG} を示す．1 行目は変数の宣言である．2 行目は初期状態の定義である．count, parked による観測値を与えることで状態を定義している．3-6 行目は，入庫を表す操作 in の定義である．入庫の条件を入庫可能数が 0 でなく，かつ，その車が駐車中でないとする（3 行目）．条件を満たさない状態での操作 in の適用は状態を変えない（4 行目）†．条件を満たすとき操作 in を適用すると，入庫可能数が 1 減らされ in した車が駐車

* bops，ops を使うと複数の同ソートの演算を一度に宣言することができる．

† キーワード ceq，bceq は，条件付き等式，条件付き振舞等式の宣言である．条件付き（振舞）等式は，通常の等式に続けてキーワード if を書き，その後ろに Bool ソートの項（条件）を書く．モデルでは，条

```
1  vars C C' : Nat    var S : State
2  eq count(init) = s 0 .  eq parked(C,init) = false .
3  eq         c-in(C,S)  = (not count(S) = 0) and (not parked(C,S)).
4  bceq         in(C,S)  = S if not c-in(C,S) .
5  ceq    count(in(C,S)) = p count(S) if c-in(C,S) .
6  ceq parked(C',in(C,S)) = (C' = C) or parked(C',S) if c-in(C,S) .
7  eq         c-out(C,S)  = parked(C, S) .
8  bceq         out(C,S)  = S if not c-out(C,S) .
9  ceq    count(out(C,S)) = s count(S) if c-out(C,S) .
10 ceq parked(C',out(C,S)) = not ((C' = C) or parked(C',S)) if c-out(C,S) .
```

図 1･15　公理 E_{BHG}

中となる（5-6 行目）．7-10 行目の等式は出庫を表す操作 out の定義である．

簡約コマンドにより，初期状態から車 0，車 s 0 が続けて入庫を試みた後，車 0 が駐車中かどうかを表す項 parked(0, in(s 0, in(0, init))) は，true に簡約される．同じ状態で車 s 0 が駐車中かどうかを表す項 parked(s 0, in(s 0, in(0, init))) は，1 台目の入庫で入庫可能数の上限に達したため，false に簡約される．例えば，駐車中の車はたかだか 1 台という述語は項 parked(C,S) and parked(C',S) implies C = C' と書ける．帰納法や場合分けの証明技術を用い，複数の証明譜を組み合わせることで，このような性質を形式的に検証することができる．

1-4-6 書換え仕様

擬順序代数をモデルにもつ書換え仕様は，システムの状態を項として明示的に記述し，項の間の書換え関係によってシステムの動作を記述する仕様である．入出庫システムの書換え仕様 mod! RW-GARAGE{ I_{RWG} Σ_{RWG} R_{RWG} }を示す．輸入 I_{RWG} は，pr(ENAT) である．Σ_{RWG} は，ソート宣言 [Loc, State]，演算宣言 ops p np : -> Loc，及び op [_,_][_] : Loc Loc Nat -> State，である．Loc は，車の位置，すなわち駐車中か（p）そうでないか（np）を表す．状態を表す項 [loc_1,loc_2][*count*] において，車 1 の位置を loc_1，車 2 の位置を loc_2，入庫可能数を *count* とする．RW-GARAGE では車の数が 2 台に制限されているため，BH-GARAGE の特殊な場合の仕様となっている．

(1) 書換え規則

書換え仕様では，公理に書換え関係という項上の二項関係を記述できる*．図 1･16 に，RW-GARAGE の公理 R_{RWG} を示す．1 行目は変数の宣言である．2 行目で，車 1 の入庫を表す

```
1  var L : Loc   var X : Nat          4  trans [p, L][X] => [np, L][s X] .
2  trans [np, L][s X] => [p, L][X] .  5  trans [L, p][X] => [L, np][s X] .
3  trans [L, np][s X] => [L, p][X] .
```

図 1･16　公理 R_{RWG}

件が成り立つとき常に等式が成り立つ．

* 正確には，等式仕様から導かれる等価関係を法とした項の同値類上の二項関係である．

書換え規則がキーワード trans で宣言されている．=>の左辺から右辺で，車 1 の位置が np から p に，入庫可能数が s X から X に変わっている．書換え規則は，左辺のパターンにあった項にのみ適用される．左辺の入庫可能数の項 s X は，入庫可能数が正のときのみ書換え規則を適用できることを表している．3 行目は，車 2 に対する同様の書換え規則である．4 行目は，車 1 の出庫を表す書換え規則である．車 1 の位置が p から np に，入庫可能数が X から s X に変わっている．5 行目は，車 2 に対する同様の書換え規則である．

(2) 探　索

書換え仕様では，項の到達可能性を検査できる．モジュール RW-GARAGE で，車庫に 2 台以上車が駐車された状態は項 [p,p][X] で表せる．初期状態からそのような状態に到達できないことの検証は，到達可能性の述語 ==> を使って，red [np, np][s 0] ==> [p, p][X] と書ける．この簡約を実行すると，左辺からの書換えをすべて探索し，右辺のインスタンスがあれば true を返す．上の例では true が返らないため，初期状態からは 2 台以上駐車した状態にたどりつけないことが保証される．一方，初期状態の入庫可能数を 2 にする（左辺を [np, np][s s 0] にする）と true が返る．true が返った場合，図 1･17 のように左辺から右辺までの経路（反例）を表示できる．

```
[state 0] ([ np , np ] [ (s (s 0)) ]):State
  trans ([ np , L ] [ (s X) ]) => ([ p , L ] [ X ])
[state 1] ([ p , np ] [ (s 0) ]):State
  trans ([ L , np ] [ (s X) ]) => ([ L , p ] [ X ])
[state 3] ([ p , p ] [ 0 ]):State
```

図 1･17　反例の表示

■参考文献

1) R.M. Burstall and J.A. Goguen, “Putting Theories Together to Make Specifications,” Proc. of the 5th International Joint Conference on Artificial Intelligence, pp.1045-1058, 1977.
2) R.M. Burstall and J.A. Goguen, “The Semantics of CLEAR,” Proc. of the Abstract Software Specifications, 1979 Copenhagen Winter School, Springer, LNCS, vol.86, pp.292-332, 1980.
3) R. Diaconescu and K. Futatsugi, “Logical foundations of CafeOBJ,” Theor. Comput. Sci., vol.285, no.2, pp.289-318, 2002.
4) R. Diaconescu and K. Futatsugi, “CafeOBJ report,” AMAST Series in Computing, vol.6, World Scientific, 1998.
5) K. Futatsugi, J.A. Goguen, J.-P. Jouannaud, and J. Meseguer, “Principles of OBJ2,” Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp.52-66, 1985.
6) K. Futatsugi and K. Okada, “Specification Writing as Construction of Hierarchically Structured Clusters of Operators,” IFIP Congress, pp.287-292, 1980.
7) J.A. Goguen and G. Malcolm, “Algebraic semantics of Imperative Programs,” MIT Press, 1996.
8) J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud, “Introducing OBJ,” Software Engineering with OBJ: algebraic specification in action, Kluwer, 2000.
9) 二木厚吉，緒方和博，中村正樹，CafeOBJ 入門 (1)-(6)，コンピュータソフトウェア，日本ソフトウェア科学会，(1) vol.25, no.2, pp.1-13, (2) pp.14-27, (3) no.3, pp.69-80, (4) no.4, pp.68-84, (5) (6) to appear, 2008-2009.