人 工 知 能 78－5
（1991．9．12）

様相論理式による基本プロセスの帰納推論アルゴリズム

木村 成伴† 富樫 敦‡ 野口 正一†

東北大学応用情報学研究センター†

東北大学電気通信研究所‡

様相論理式を具体的な事実(仕様)とし，意図するプロセスと観測等価な基本プロセス(CCS[14]のサブセット)を帰納的に推論するアルゴリズムを与える．このアルゴリズムはShapiroがモデル推論で用いたような漸増的な方法[8,9]をもとにしており，一般に無限個ある事実を1つずつ入力していくことで意図するプロセスに近づいていく．アルゴリズムは無限操作となるが，極限収束の概念を導入し，その収束性を論じる．また，推論に最低限必要な事実を与え，その正当性を証明する．

An Inductive Inference Algorithm of Basic Process by Modal Formulas

Shigetomo Kimura† Atsushi Togashi‡ Shoichi Noguchi†

Research Center for Applied Information Science,Tohoku University†

Research Institute of Electrical Communication,Tohoku University‡

2-1-1 Katahira Aoba-ku Sendai-shi Miyagi-ken,Japan

This paper presents an inductive inference algorithm that infers basic processes (a subset of CCS processes[14]) from concreted model formulas. The modal formulas are treated as a specification of the intended process. The resulting process is shown to be observationally equivalent with the target one, if any. This algorithm is based on the idea of Shapiro's Model Inference[8,9]. Since this algorithm does not terminate, a notion of convergence in the limit is introduced, and convergence of the output sequences of the algorithm is disscussed.

1. はじめに

コンピュータへの需要が高まるにつれ，ソフトウエアの開発の効率化が求められるようになってきている．

プログラムを作成することを考えてみると，最初にプログラムが満たすべき性質(仕様)が与えられ，それを満足するようにプログラムは作られる．この操作を自動化する，すなわち，作成したいプログラムが満たさなくてはならない仕様を与えることによってプログラムを自動的に生成することは，プログラム作成の効率化という観点から見ると非常に重要なテーマとなる．これを別の観点から見れば，仕様，プログラムが満たすべき事実からそれを帰納的に推論することになる．また，事実から推論するという操作を例題から学習させると考えることで計算論的学習の分野にも相通じる．

以下ではプログラムに相当するものとして数学的に形式化されたプロセスと呼ばれるものを，仕様として様相演算を用いた論理式を採用し，仕様に矛盾しないプロセスを生成するアルゴリズムを与え，その評価を行う．

2. プロセスの代数的記述とプロセス論理

2.1 プロセスの代数的形式化

もうこれ以上分解できない原子的な動作をアクション(action)と呼び，以下考察の対象とするアクションの集合Actを仮定する．アクションはシステムによって実行される原子的動作の単位であり，外部から観測可能であると同時に制御可能であるとする．はじめに，3種類の演算子を用いてプロセス(あるいはエージェント)を次のように記号的に表現する．ここでは，プロセスとプロセスの記号的表現を同一視し，項で表されたプロセス表現(あるいは，プロセス動作式)をプロセスと呼ぶことにする．

【定義2.1.1】基本プロセス(basic process)，あるいは基本プロセス動作式(basic process behaviour expression)は，次のように帰納的に定義される．

(0) 無動作プロセス(inaction) 0 は基本プロセスである．

(1) pが基本プロセスであるとき，アクション $a\in Act$ によるプレフィックス(prefix) a.p は基本プロセスである．

(2) p_1, p_2 を基本プロセスとするとき，これらによる和(summation) p_1+p_2 は基本プロセスである． □

ここではプロセスの表現形式としてCCS[14]のサブセットを採用している．しかし，この基本プロセスだけで理論的にはCCSの全てのプロセスを表現することが可能であることが知られている．以下では，基本プロセスのことを単にプロセスと呼ぶことにする．

プロセスは，アクションを行う非決定性プログラムと解釈することができる．例えば，

$$a.(b.0+a.b.0)$$

は最初にaを行い，その後bを行って終了するか，あるいは再度aを行い，さらにbを行って止まる非決定性プログラムである．

プロセスの意味は，アクションをラベルとするラベル付き遷移システムによって与えられる．

【定義2.1.2】ラベル付き遷移システム(labeled transition system)，あるいは単に遷移システムは，3項組$<S,Act,\rightarrow>$である．ここで，Sは状態の集合，Actはアクションの集合，$\rightarrow$は遷移関係であり $\rightarrow\subset S\times Act\times S$ として定義される． □

遷移関係について，$(s,a,s')\in\rightarrow$ であるとき単に $s\text{—}a\rightarrow s'$ と記すことにすると，遷移関係は $\rightarrow=\{\text{—}a\rightarrow|a\in Act\}$ と表すことができる．$s\text{—}a\rightarrow s'$ は状態sでアクションaを実行することができ，aを実行した結果状態s'に遷移することを表している．以上から，$s\text{—}a\rightarrow s'$のときs'をアクションaによるsのa-successorとか単にsuccessorなどと呼ぶ．

なお，$s\text{—}a\rightarrow$ として，sはアクションaが可能である状態に遷移が可能なことを示し，$s\text{—}/a\rightarrow$ でsからaの実行が不可能であることを示すものとする．

【定義2.1.3】プロセスの遷移関係は，次の遷移規則によって与えられる．

(1) $a.p\text{—}a\rightarrow p$

(2) $p\text{—}a\rightarrow p' \Rightarrow p+q\text{—}a\rightarrow p'$
$q\text{—}a\rightarrow q' \Rightarrow p+q\text{—}a\rightarrow q'$ □

例として，$p=a.(b.0+a.b.0)$の導出木(アクション木，遷移木)は次のようになる．

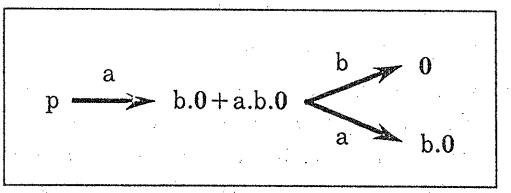

図.1 導出木

2.2 プロセスの等価性

【定義2.2.1】プロセス上の関係Rが強双模倣(strong bisimulation)(関係)であるとは，(p,q)∈Rならば，任意のa∈Actについて，

(1) $p \xrightarrow{a} p' \Rightarrow q \xrightarrow{a} q', (p',q') \in R$ for some q'

(2) $q \xrightarrow{a} q' \Rightarrow p \xrightarrow{a} p', (p',q') \in R$ for some p' □

【定義2.2.2】プロセスp,qが強等価(strongly equivalent)，あるいは強双模倣的(strong bisimilar)，p~q，であるとは，ある強双模倣Rに対して，(p,q)∈R．同値的に，~=∪{R|R:強双模倣}. □

あきらかに，~は関係として最も大きい強双模倣であり，同値関係である．

【命題2.2.3】プロセスに対して，次の等価式が成り立つ．

(1) p+q~q+p
(2) p+(q+r)~(p+q)+r
(3) p+p~p
(4) p+0~p □

【命題2.2.4】~は合同関係である．すなわち，$p_1 \sim p_2$のとき，

(1) $a.p_1 \sim a.p_2$
(2) $p_1+q \sim p_2+q$ □

【定義2.2.5】遷移システムが(→に関して)イメージ有限(image finite)であるとは，任意の状態pから任意のアクションaで1ステップで遷移可能な状態の集合は有限，すなわち$\{q \mid p \xrightarrow{a} q\}$は有限 □

2.3 プロセス論理

プロセスの等価性を定める方法の中に，性質あるいは論理を用いる方法がある．これは，2つのプロセスが等価であるとは，それぞれのプロセスが満足する性質(論理式)が全く同じとき．あるいは，2つのプロセスが異なるとは，2つのプロセスを区別する性質(論理式)が存在するとき，とする方法である．

【定義2.3.1】論理式は，次のように帰納的に定義される．

(1) Tは論理式である．
(2) A, A_iが論理式ならば，$\wedge_{i\in I}A_i, \neg A$は論理式である．ここで，Iはインデックス集合．
(3) Aが論理式ならば，$<a>A$は論理式である．ここで，a∈Act. □

(1), (2), (3)によって生成される論理式の集合を£と置く．

【定義2.3.2】論理式の充足性を次のように定める

(1) 任意のプロセスpに対して$p \models T$
(2) $p \models \wedge_{i\in I}A_i$ iff 任意のi∈Iに対して$p \models A_i$.
(3) $p \models \neg A$ iff $p \not\models A$. ここで，$p \not\models A$はプロセスpがAを満たさないことを示す．
(4) $p \models <a>A$ iff あるqが存在して$p \xrightarrow{a} q$かつ$q \models A$. □

【定義2.3.3】便宜的に，次の論理記号を導入する．

(1) $F \equiv \neg T$
(2) $\vee_{i\in I}A_i \equiv \neg\wedge_{i\in I}\neg A_i$
(3) $A_1 \wedge A_2 \equiv \wedge_{i\in\{1,2\}}A_i$
(4) $A_1 \vee A_2 \equiv \vee_{i\in\{1,2\}}A_i$
(5) $[a]A \equiv \neg <a> \neg A$. □

プロセスpがアクションaを実行することができないとき，pはa-デッドロックするという．定義より，Tは任意のプロセスによって満足されるから，Fはどんなプロセスによって満足されることはない．従って，$p \models [a]F$はpが→に関してa-デッドロックすることを意味する．

【例2.3.4】今定義した論理式を用いて記述したプロセスの性質の例を示す．

a) $p \models <a>T$: aの動作が可能．
b) $p \models [a]F$: pはa-デッドロックする．
c) $p \models <a>([b_1]F \vee [b_2]F)$: aを行った後，必ず$b_1$-デッドロックするかあるいは$b_2$-デッドロックすることが可能である．
d) $p \models [a_1](<a_2>[a_3]F)$: a_1を行った後のどの状態でも，a_2を行うことによってある状態にたどり着くことが可能で，そこではa_3-デッドロックする． □

論理式の集合Lに対して，

$$£(p) = \{A \in L \mid p \models A\}$$

と置く．また，$£_n$で様相演算子<a>のネストの回数が高々n回である論理式の集合を表す．

【命題2.3.5】遷移システムがイメージ有限とすると，プロセスpとqが強等価p~qであることと，£(p)=£(q)であることは同値[16]. □

【例2.3.6】この定理により，2つのプロセスが強双模倣等価であるかどうかを簡単に判定することができる．例えば，p=a(b+c), q=ab+acとし，

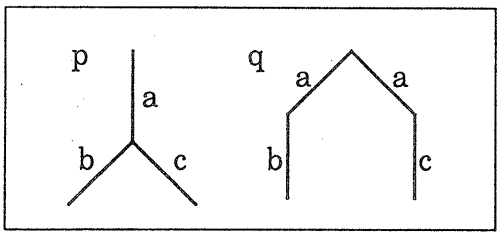

図.2 強双模倣等価の判定

$$A=<a>(<b>T\wedge<c>T)$$

と置くと，$p\models A, q\models A$より，$p\nsim q$.

$L\subseteq$£とし，Lによって定まる同値関係を

$$p\simeq_L q \Leftrightarrow L(p)=L(q)$$

で定める．ここで，$L(p)=\{A\in L \mid p\models A\}$.

もしL=ϕならば，$\simeq_L$ではどんなプロセスも同じになってしまう．ただ言えるのは，Lが大きくなればなるほど，対応する$\simeq_L$でより多くのプロセスが区別できるようになる．上記の定理は，Lとして£を取れば，$\simeq_L$は強等価関係$\sim$と同一になる．

3. 強等価性に関するプロセスの合成アルゴリズム

この章では，プロセスの具体的な仕様から意図したプロセスと強等価なプロセスを合成するアルゴリズムについて述べる．これを実現するアルゴリズムとしてShapiroがモデル推論で用いたような漸増的な方法[8,9]がある．これはプロセス全体の枚挙を考え，既に読み込んでいる事実に対して正しいプロセスを拾い上げ，間違っているプロセスを捨てるという方法である．しかしこのことは，プロセスの全体集合から事実に反しないプロセスを探しているに過ぎず，効率という点から考えてみるとあまりよい方法とは言えない．以下ではこれよりも効率の良いアルゴリズム考える．

3.1 合成アルゴリズム

基本的なアイデアは，入力された事実とそれまでに入力された事実に矛盾しないプロセスから全体に矛盾しないプロセスを効率よく作り出すことにある．

まず，準備を行う．

【命題3.1.1】$T,\wedge_{i\in I}A_i,\neg A,<a>A$で定義された任意の論理式は，$T,F,\wedge_{i\in I}A_i,\vee_{i\in I}A_i,<a>A,[a]A$のみを使った論理式に等価に変換される．

(略証)¬Aの場合のみ述べる．

A=T,A=Fの場合はそれぞれ¬A=F,¬A=Tと直せる．$A=\wedge_{i\in I}A_i$の場合は$\neg\wedge_{i\in I}A_i=\vee_{i\in I}\neg A_i$ A=¬Xの場合は¬(¬X)=X. A=<a>Xの場合は¬<a>X=[a](¬X). $A=\vee_{i\in I}A_i$の場合は$\neg\vee_{i\in I}A_i$ $=\wedge_{i\in I}\neg A_i$. A=[a]Xの場合は¬[a]X=<a>(¬X)と変形できる． □

命題3.1.1により，¬を陽に考えなくてもよくなった．以下では¬が陽には現れない論理式を考えていく．

以下ではプロセスpの導出木として各ノードに論理式を付けた特別な導出木を考える。ただし、論理式は付いているノードをルートとしたプロセスを満たしていなければならない。

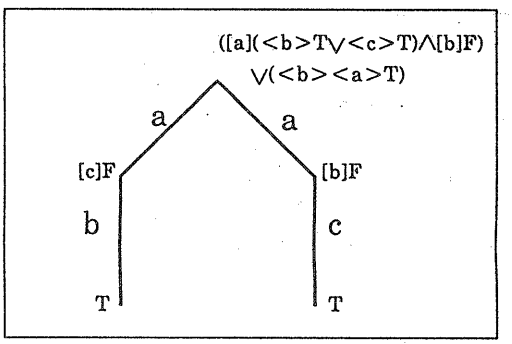

図.3 論理式をラベル付けした導出木

【定義3.1.2】次の論理式は矛盾した論理式という．

(1) Fは矛盾している論理式．
(2) 論理式Pが矛盾しているとき，<a>Pも矛盾した論理式．
(3) 論理式の集合$\{P_i \mid i\in I\}$が次のいずれかを満たすとき，$\wedge_{i\in I}P_i$は矛盾した論理式である．
(i) あるi∈Iに対してP_iが矛盾した論理式である
(ii) あるi,j∈Iに対して$P_i=<a>X, P_j=[a]Y$のとき，X∧Yが矛盾した論理式である．
(4) 論理式の集合$\{P_i \mid i\in I\}$において任意のi∈Iに対してP_iが矛盾した論理式であるとき$\vee_{i\in I}P_i$は矛盾した論理式である．

【定義3.1.3】論理式の系列$A_1,\cdots,A_k$において，$A_1\wedge\cdots\wedge A_k$が矛盾しているとき，系列は矛盾しているという．

以下で生成アルゴリズムについて述べる．

始めに入力された事実に矛盾しないプロセスの生成方法を考える．生成にはノードに論理式のラベルをつけた導出木を考え，生成の結果として各

ノードから論理式のラベルを取り除いた導出木だけを出力する．

まず，全体的なアルゴリズムを示す．

【アルゴリズム3.1】合成アルゴリズム

入力: 事実の枚挙 $A_1,A_2,\cdots$(満足すべき論理式の列であり，順番は任意)

出力: 推測のプロセスの列 $p_1,p_2,\cdots$(各 p_k はいままで読み込んだ k 個の事実に対して正しく振舞う)

手続き:{}内はコメント

```
begin
  k←0;
  pc←0; C←T;
  q←pc; Cq←C; Q←∅;
  repeat
    k←k+1;
    次の事実Akを読み込む;
    if pc⊨Ak then
        (pc,C)←(アルゴリズム3.2);
        {Ak,pc,CからA1,…,Akすべてに矛盾しないプロセスp(Ak,pc,C)とこれが満たすべき条件c(Ak,pc,C)を作成する．もしこのようなプロセスが存在しない場合は，便宜的にpc=0,C=Fを返すことにする．実際にはAk,pc,Cだけではプロセスを定めることはできず，この他にq,Cq,Qが必要となる}
        if C=F then
          入力事実に矛盾があり，エラー．
        endif
    endif
    pc を pk として出力する;
  forever
end
```

なお，アルゴリズム3.1の変数の中でCは条件ラベル付導出木のルートにつく論理式を，qは途中で得られたプロセス$p_1,\cdots,p_{k-1}$のどれかを，C_qはqの条件ラベル付導出木のルートにつく論理式を示している．また，Qは入力された事実の一時的な退避用として使っている．

さて，アルゴリズム3.1の中にあるp_c,Cの作成方法を与えていく．

【アルゴリズム3.2】

パラメータ: q,C_q,Q

入力: 事実A_k

$A_1,A_2,\cdots,A_{k-1}$に矛盾しないプロセスp_c

p_cのルートにつく条件C

出力: 推測のプロセス$p(A_k,p_c,C)$

p_cのルートに新たにつく条件$c(A_k,p_c,C)$

(1). $A_k=T$の場合

任意のプロセスpに対して$p\models T$となるので，元のp_cは$A_1,\cdots,A_k$を全て満たしている．すなわち

$$p(A_k,p_c,C)=p_c$$

また，条件の変更もない．

$$c(A_k,p_c,C)=C$$

(2). $A_k=F$の場合

Fは任意のプロセスに矛盾するので，$A_1,\cdots,A_k$を全て満たすプロセスは存在しない．従って，

$$p(A_k,p_c,C)=0$$
$$c(A_k,p_c,C)=F$$

(3). $A_k=<a>X$の場合

Xを満たすプロセスp'を作り，a.p'をp_cにつける．ただしここで作った枝は，他の枝同様，条件Cを満たしていなくてはならない．ただしc(X,0,T)=Fまたはc(C,a.p(X,0,T),T)=Fのときは，

Qに$\vee$の論理式がないときは，

$$p(A_k,p_c,C)=0$$
$$c(A_k,p_c,C)=F$$

Qに$\vee$の論理式があるときは，(7)でプロセスを作り直す．

Fにならないときは，

$$p(A_k,p_c,C)=p_c+p(C,a.p(X,0,T),T)$$

また，条件の変更はない．

$$c(A_k,p_c,C)=C$$

(4). $A_k=\wedge_{i\in I}X_i(I=\{1,\cdots,m\})$の場合

X_iを別々に適用する．

$$p_{c1}=p(X_1,p_c,C),C_0=c(X_1,p_c,C)$$
$$p_{cj+1}=p(X_j,p_{cj},C_j),C_{j+1}=c(X_j,p_{cj},C_j)$$
$$(1\leqq j\leqq m\text{-}1)$$
$$p(A_k,p_c,C)=p_{cm}$$
$$c(A_k,p_c,C)=C_m$$

(5). $A_k=[a]X$の場合

一般に，$p_c=a.p'_1+\cdots+a.p'_m+x_1.p'_{m+1}+\cdots+x_n.p'_{m+n}(x_1,\cdots,x_n\neq a)$と置くことができる．[a]Xはaのアクションができる枝への条件とみることができるので，$p'_1,\cdots,p'_m$のルートに付く論理式をそれぞれ$C'_1,\cdots,C'_m$とすれば，

$$p(A_k,p_c,C)=a.p(X,p'_1,C'_1)+\cdots+a.p(X,p'_m,C'_m)+x_1.p'_{m+1}+\cdots+x_n.p'_{m+n}$$

また，条件A_kは(3)で作られる枝に対しても適用される条件なので

$$c(A_k,p_c,C)=C\wedge A_k$$

ただし，CとA$_k$が矛盾しているか，c(X,p'$_1$,C'$_1$),…,c(X,p'$_m$,C'$_m$)のうちどれか1つでもFになったときはp(A$_k$,p$_c$,C)=0,c(A$_k$,p$_c$,C)=F．しかし，この場合もQに∨の論理式があるとき(7)へ．

(6)．A$_k$=∨$_{i\in I}$X$_i$の場合

∨がある場合が一番問題となる．なぜなら，A$_k$自体は正しくても，項ごとに分割したときに矛盾が生じるようなものが出てくるからである．例えば，次の例を考えてみる．

A$_k$=([a](<b>T∨<c>T)∧[b]F)
∨(<b><a>T)

ここで，[b]Fと<b><a>Tはあきらかに矛盾する．しかし，∨のためいずれかの項が成り立てばよいのでA$_k$自体は正しく，どちらが正しいのかは一般にこの時点では分からない．間違った方を適用した場合，A$_k$を適用する前の時点から作り直す必要が生じることがある．FとGがA$_k$,p$_c$,Cの3つで定まらない理由はこのためにある．そして，元の時点に戻すためにあるのがパラメータq,C$_q$およびQである．

(6-1)．Qに∨の論理式がないとき

まず，X$_i$(i∈I)の中からCと矛盾するものを取り除く．I=∅，すなわちX$_i$の要素が1つもなくなったらp(A$_k$,p$_c$,C)=0,c(A$_k$,p$_c$,C)=Fとする．

次に，p$_c$をqに，CをC$_q$に代入する．このqには∨を適用する直前のプロセスが入っていることになる．そして，X$_i$(i∈I)の中から任意のプロセスX$_j$を取り出し，印を付ける．

次にp(X$_j$,p$_c$,C)とc(X$_j$,p$_c$,C)を求める．c(X$_j$,p$_c$,C)=FならばX$_j$を取り除く．これでI=∅になったらp(A$_k$,p$_c$,C)=0,c(A$_k$,p$_c$,C)=F．I≠∅でなければ別なプロセスを取り出し，これを繰り返す．

c(X$_j$,p$_c$,C)≠Fならば，

p(A$_k$,p$_c$,C)=p(X$_j$,p$_c$,C)
c(A$_k$,p$_c$,C)=c(X$_j$,p$_c$,C)

最後に，X$_j$に印を付けた状態で，∨$_{i\in I-J}$X$_i$をQに加える．ここでJは途中で削除された論理式のインデックスの集合．

(6-2)．Qに∨の論理式があるとき

X$_i$(i∈I)の中から任意のプロセスX$_j$を取り出しX$_i$に印を付けてA$_k$をQに加える．次に，p(X$_j$,p$_c$,C)とc(X$_j$,p$_c$,C)を求める．c(X$_j$,p$_c$,C)=Fならばプロセスを作り直す（(7)へ）．そうでなければ，

p(A$_k$,p$_c$,C)=p(X$_j$,p$_c$,C)
c(A$_k$,p$_c$,C)=c(X$_j$,p$_c$,C)

(7)．プロセスの作り直し

∨の論理式がQに現れる前に戻ってプロセスを作り直す．

Qの要素で∨がトップレベルに現れている論理式をA$_{\ell_1}$,…,A$_{\ell_m}$，そうでないものをA$_{\ell_{m+1}}$,…,A$_{\ell_{m+n}}$とする．(1)~(6)から分かるように，この(7)にくるケースはm>0に場合に限られる．

(7-1)．n>0の場合

A$_{\ell_{m+1}}$,…,A$_{\ell_{m+n}}$までの事実をqに先に適用する．

q←p(∧$_{i\in\{m+1,\cdots,m+n\}}A_{\ell_i}$,q,C$_q$)
C$_q$←c(∧$_{i\in\{m+1,\cdots,m+n\}}A_{\ell_i}$,q,C$_q$)

この後，(7-2)へ．

(7-2)．n=0もしくは(7-1)からの継続

A$_{\ell_i}$=∨$_{j_i\in J_i}$Y$_{\ell_{ij_i}}$と置き，Dを次のように定める．このとき，Y$_{\ell_{ij_i}}$の中で印が付いている論理式を特にY$_{\ell_{ik}}$としておく．

D=∨{∧$_{i=1,\cdots,m}$Y$_{\ell_{ij_i}}$|
$j_i\in J_i$,(j_1,…j_m)≠(k,…k),
∧$_{i=1,\cdots,m}$Y$_{\ell_{ij_i}}$は矛盾しない}

Dの要素が1つもなかったらp(A$_k$,p$_c$,C)=0,c(A$_k$,p$_c$,C)=Fとする．そうでなければ，Y$_{\ell_{ik}}$についた印を消して,Q←∅とする．
これにより，(6-1)が適用できるので，

p(A$_k$,p$_c$,C)=p(D,q,C$_q$)
c(A$_k$,p$_c$,C)=c(D,q,C$_q$)

が求められる．□

【定義3.1.4】関数n:L→Nは論理式の大きさを示す関数であり，次で定義される．

(1) n(T)=n(F)=1
(2) Xが論理式のとき，
n(<a>X)=n([a]X)=1+n(X)
(3) X$_i$が論理式のとき(i∈I),
n(∧$_{i\in I}$X$_i$)=Σ$_{i\in I}$n(X$_i$)
(4) X$_i$が論理式のとき(i∈I),
n(∨$_{i\in I}$X$_i$)=max$_{i\in I}$n(X$_i$)

□

【定義3.1.5】関数μ:L→Lは論理式から[a]Xを取り出す関数であり，次で定義される．

(1) μ(T)=T
(2) μ(F)=F
(3) Xが論理式のとき，μ(<a>X)=T
(4) Xが論理式のとき，μ([a]X)=[a]X
(5) X$_i$が論理式のとき(i∈I)，μ(∧$_{i\in I}$X$_i$)=∧$_{i\in I}$μ(X$_i$)
(6) X$_i$が論理式のとき(i∈I)，μ(∨$_{i\in I}$X$_i$)=X$_j$ (j∈I)

□

【定理3.1.6】$A_1,\cdots,A_k$を有限な無矛盾な事実の系列とするとアルゴリズム3.1,3.2によって生成される$p(A_k,p_c,C),c(A_k,p_c,C)$は次を満足する．

(1) $p(A_k,p_c,C)\models\wedge_{i\in\{1,\cdots,k\}}A_i$
($k=0$のとき$\wedge_{i\in\{1,\cdots,k\}}A_i=T$とする)

(2) $c(A_k,p_c,C)=\wedge_{i\in\{1,\cdots,k\}}\mu(A_i)$
($k=0$のとき$\wedge_{i\in\{1,\cdots,k\}}\mu(A_i)=T$とする)

(3) $p(A_k,p_c,C)\models c(A_k,p_c,C)$

(証明)

定義3.1.4で定められる大きさに関する帰納法で証明される．紙面の都合により詳細は省く． □

【系3.1.7】Sを無矛盾な論理式の有限集合とすると，Sを満たすプロセスが必ず存在する．

(証明)

アルゴリズム3.1,3.2を適用することで，Sを満たすプロセスが得られる． □

【命題3.1.8】$A_1,\cdots,A_k$を有限な無矛盾な事実の系列とするとアルゴリズム3.1,3.2は有限時間で停止する． □

しかし，$A_1,\cdots,A_k$の全てがN個の要素を持った$\vee$の論理式の場合，最悪の場合，プロセスの作り直しが(トップレベルだけでも)N^k回かかることになる．一般にこのアルゴリズムでは多項式時間には収まらない．

合成アルゴリズムは永久に続く手続きなので，その正当性を極限における収束という概念を用いて示す．

【定義3.1.9】事実の枚挙を順番に読み込みプロセスを逐次出力するアルゴリズムが，ある時点以降pとは異なるプロセスを二度と出力しなければ，アルゴリズムが出す推測は，事実の枚挙上でpに極限において収束するという． □

【命題3.1.10】アルゴリズム3.1,3.2の仮定の元で，事実の枚挙を全て満たす基本プロセスp_0が存在するとする．このとき，アルゴリズム3.1,3.2が出す推測が極限において収束するならば，出力されるプロセスpはp_0と強等価である．

(証明)

p_0とは強等価にならないプロセスp'_0に極限において収束したと仮定する．強等価ではないので$\mathcal{L}(p_0)\neq\mathcal{L}(p'_0)$となる．これより$A\in\mathcal{L}(p_0),A\notin\mathcal{L}(p'_0)$となる論理式Aが存在する．

さて，アルゴリズム3.2にこのAが入力されるとAに矛盾しないプロセスが生成される．これ以降，論理式を入力しても生成されるプロセスはAに矛盾しない．従って，$p'_0\models A$．これは$A\notin\mathcal{L}(p'_0)$に反する． □

一般に入力される事実は無限なので次の2つの理由で収束しない。

①．p_0に収束しない

p_0と等価なプロセスになるために必要な論理式がいつまでたっても出てこない

②．p_0と等価なプロセスになるが、プロセスの形は収束しない

等価なプロセスがたくさんあることによる問題。例えば、$p_0=a.0$とすると、$<a>T,<a>T\wedge<a>T,<a>T\wedge<a>T\wedge<a>T,\cdots$という論理式の系列は$p_0$に矛盾しない。しかし、アルゴリズムによって生成されるプロセスは$a.0,a0+a0,a0+a0+a0,a0+a0+a0+a0,\cdots$となり、$p_0$と等価ではあるが収束したとは言えない。

【例3.1.11】実際にプロセス$p_0=a.(a.0+b.0)+a.b.0$を合成してみる．簡単のため，アクションの集合$Act=\{a,b,c\}$とする．事実として次の$A_1,\cdots,A_4$を与える．(図4-図中の$p_1\sim p_n$でノードにつける論理式は()をつけて表している．ただし，(T)となるものは省略してある)

$A_1=<a><b>T$
$A_2=<a>(<a>T\wedge<b>T)$
$A_3=<a>(<b>T\wedge[a]F)$
$A_4=<a>(<c>T\vee<b>T)$

p_2が目標のp_0と強等価であるが，A_1,A_2を満たすプロセスはたくさんある．従ってもっと事実を入力する必要がある．p_3でがノードのラベルとしてついた．p_4では$<c>T\vee<b>T$という事実のためp_0にはないcという枝ができてしまった．しかし，$\vee$のためA_4はp_0とは矛盾していない．後で作り直しが生じ，$<c>T$の代わりに$<b>T$が適用されることになる．

p_4はあきらかにp_0とは強等価にはなっていない．強等価になるには少なくとも次の2つの事実が必要となる．これを入力すると図4のp_nになる．

$[b]F\wedge[c]F$
$[a]((<a>T\wedge<b>T\wedge[c]F)$
$\vee(<a>T\wedge[b]F\wedge[c]F))$

これを入力すると図4のp_nになる．
($C_1=[b]F\wedge[c]F,C_2=[c]F,C_3=[a]F\wedge[c]F$とする)

p_0が分かっているとき，最低限必要な事実は何か．次の定義と定理がそれを示している．

【定義3.1.12】与えられたプロセスpで使われるアクションの集合Aが有限であり，pの深さが有限の

とき，論理式Pを帰納的に作ることができる．

(1) $p=0$のとき

$P=\wedge_{a\in A}[a]F$

(2) $p=\Sigma_{j=1,\cdots,m}\Sigma_{i=1,\cdots,n_j}a_j.p_{ji}$(ただし$\{a_1,\cdots,a_m\}=A, a_i\neq a_j(i\neq j$のとき$), n_j=0$のときは$\Sigma_{i=1,\cdots,n_j}a_j.p_{ji}=0$)

再帰によってp_{ji}からP_{ji}を作り出す．このとき，

$P=\wedge_{j=1,\cdots,m}((\wedge_{i=1,\cdots,n_j}<a_j>P_{ji})\wedge([a_j]\vee_{i=1,\cdots,n_j}P_{ji}))$

(ただし，$n_j=0$のときは

$\wedge_{i=1,\cdots,n_j}<a_j>P_{ji}=T, \vee_{i=1,\cdots,n_j}P_{ji}=F$)

【定理3.1.13】定義3.1.12によってプロセスpから作られた論理式Pは，次を満たす．

(1) $p\models P$

(2) $p\sim p' \Rightarrow p'\models P$

(3) pとp'のソートは同じであるとき，

$p'\models P \Rightarrow p\sim p'$

(ソートとはプロセスで生じる可能性のあるアクションの集合)

(証明)

(1). pの深さ(の最大値)に関する帰納法

深さ$=0$のとき．すなわち，$p=0$.

このとき，$P=\wedge_{a\in A}[a]F$である．任意の$a\in A$をとってくると，このとき$p\text{—}/a\rightarrow$より，$p\models[a]F$. これが任意のaで成り立つから$p\models P$.

深さ$=k$のとき．$p=\Sigma_{j=1,\cdots,m}\Sigma_{i=1,\cdots,n_j}a_j.p_{ji}$とおく．(ただし$\{a_1,\cdots,a_m\}=A, a_i\neq a_j(i\neq j$のとき$), n_j=0$のときは$\Sigma_{i=1,\cdots,n_j}a_j.p_{ji}=0$)

あきらかにp_{ji}の深さはkより小さい．よって，p_{ji}から定義3.1.13よりP_{ji}が作れて$p_{ji}\models P_{ji}$. このとき，

$P=\wedge_{j=1,\cdots,m}(\wedge_{i=1,\cdots,n_j}<a_j>P_{ji})\wedge([a_j]\vee_{i=1,\cdots,n_j}P_{ji})$

となる(ただし$n_j=0$のときは$\wedge_{i=1,\cdots,n_j}<a_j>P_{ji}=T, \vee_{i=1,\cdots,n_j}P_{ji}=F$)

任意の$a_j\in A$をもってくると，任意の$i\leqq n_j$に対して

$p\text{—}a_j\rightarrow p_{ji}\models P_{ji}$

よって，$p\models\wedge_{j=1,\cdots,m}\wedge_{i=1,\cdots,n_j}<a_j>P_{ji}$. また$p_{ji}\models\vee_{i=1,\cdots,n_j}P_{ji}$であるから，

$p\models\wedge_{j=1,\cdots,m}[a_j]\vee_{i=1,\cdots,n_j}P_{ji}$

これより，$p\models P$がいえた．

(2). $p\sim p' \Leftrightarrow £(p)=£(p')$である．(1)より，$p\models P$なので$P\in£(p)$. 従って$P\in£(p')$が成り立ち，$p'\models P$.

(3). 対偶を求める．すなわち，以下を証明する．

pとp'のソートは同じであるとき，

$p\not\sim p' \Rightarrow p'\not\models P$

(ここでpのソートをAとする)

pの深さに関する帰納法で証明する．

深さ$=0$のとき．すなわち$p=0$. よって$P=\wedge_{a\in A}[a]F$.

$p\not\sim p'$より，ある$a\in A$があって$p'\text{—}a\rightarrow$でなくてはならない．しかし，$P=\wedge_{a\in A}[a]F$であるので$p'\not\models P$.

深さ$=k$のとき．$p=\Sigma_{j=1,\cdots,m}\Sigma_{i=1,\cdots,n_j}a_j.p_{ji}$, $p'=\Sigma_{j=1,\cdots,m}\Sigma_{i=1,\cdots,n'_j}a_j.p'_{ji}$とおく．$p\not\sim p'$より次の4つの場合が考えられる．

(a). $p\text{—}a_j\rightarrow$, $p'\text{—}/a_j\rightarrow$ $(n_j>0, n'_j=0)$のとき

(b). $p\text{—}/a_j\rightarrow$, $p'\text{—}a_j\rightarrow$ $(n_j=0, n'_j>0)$のとき

(c). $p\text{—}a_j\rightarrow p_{ji}$, $p'\text{—}a_j\rightarrow p'_{jk}$, $p_{ji}\sim p'_{jk}$となるkが存在しない

(d). $p\text{—}a_j\rightarrow p_{ji}$, $p'\text{—}a_j\rightarrow p'_{jk}$, $p_{ji}\sim p'_{jk}$となるiが存在しない

$P=\wedge_{j=1,\cdots,m}((\wedge_{i=1,\cdots,n_j}<a_j>P_{ji})\wedge([a_j]\vee_{i=1,\cdots,n_j}P_{ji}))$とおく．

(a). $p\text{—}a_j\rightarrow p_{ij}\models P_{ij}$とする．$p'\text{—}/a_j\rightarrow$だから，$p'\not\models<a_j>P_{ji}$. ゆえに$p'\not\models P$.

(b). $p\text{—}/a_j\rightarrow$, $p'\text{—}a_j\rightarrow p'_j$とすると$n_j=0$より$[a_j]\vee_{i=1,\cdots,n_j}P_{ji}=[a_j]F$. よって，$p'\not\models[a_j]F$. ゆえに$p'\not\models P$.

(c). 任意のkに対して$p\text{—}a_j\rightarrow p_{ji}\models P_{ij}$, $p'\text{—}a_j\rightarrow p'_{jk}$, $p_{ji}\not\sim p'_{jk}$とおける．これより帰納法の仮定から，$p'_{jk}\not\models P_{ij}$. これが任意のkに対して成り立つので，$p'\not\models<a_j>P_{ji}$. ゆえに$p'\not\models P$.

(d). 任意のiに対して$p\text{—}a_j\rightarrow p_{ji}\models P_{ij}$, $p'\text{—}a_j\rightarrow p_{jk}$, $p_{ji}\not\sim p'_{jk}$とおける．帰納法の仮定から$p'_{jk}\not\models P_{ij}$. 従って，$p'_{jk}\not\models\vee_{i=1,\cdots,n_j}P_{ij}$, $p'\not\models[a_j]\vee_{i=1,\cdots,n_j}P_{ji}$. これより，$p'\not\models P$. □

4. おわりに

基本プロセスと強等価性という制約があったが，様相演算を用いた性質からそれらを満たすプロセスの生成アルゴリズムを与えることができた．今後の課題としては次のようなものがある．

○ 再帰以外の演算子を許した有限プロセスの合成

○ その他の等価性に関するプロセスの合成

○ 再帰的プロセスの合成アルゴリズム

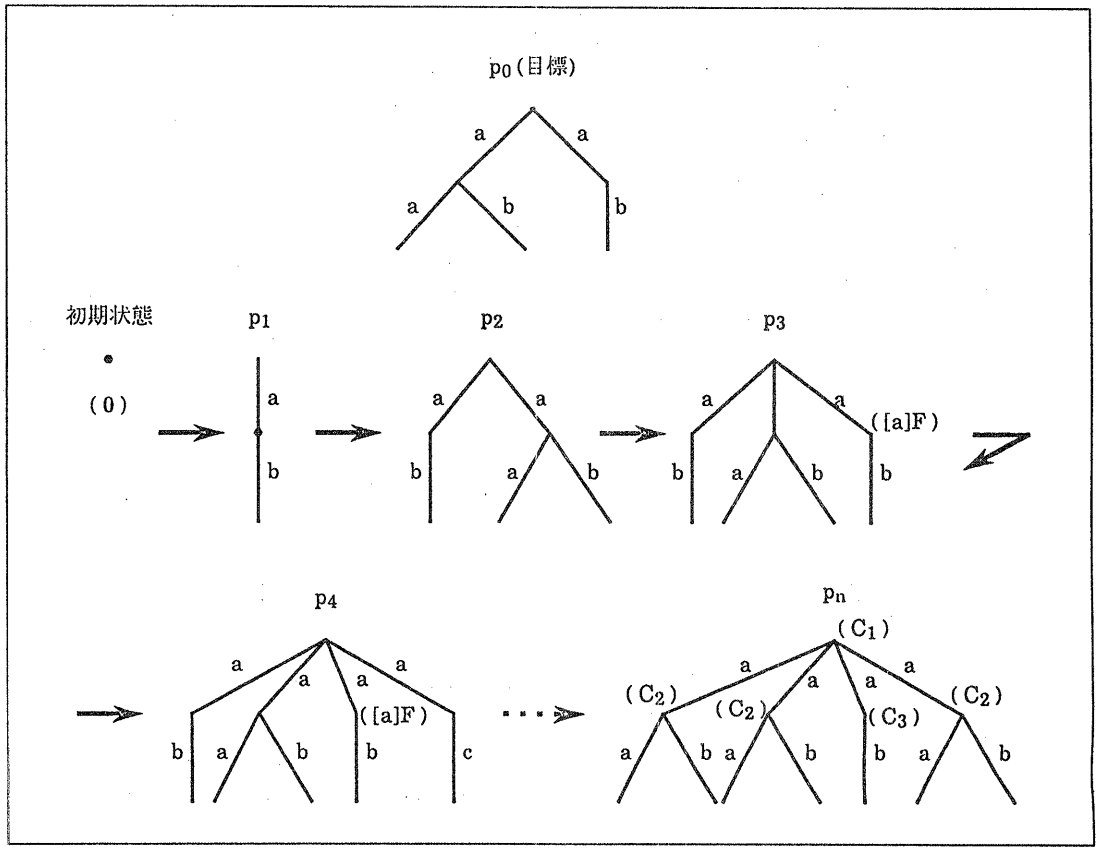

図.4 合成例

参考文献

[1] Biermamm, A.W.: The inference of regular Lisp programs from examples, IEEE Trans. on Systems, Man and Cybernetics SMC-8, pp.585-600, 1978.

[2] Darlington, J.: Program Transformation, Functional Programming and its Application, Cambridge Univ. Press, pp.193-215, 1982.

[3] 大特集:自動プログラミング,情報処理,Vol.28, No. 10, 1987.

[4] Knuth, D.E.: The Art of Computer Programming, Addison-Wesley, Reading, Masachusetts, 1968.

[5] Manna, Z., Waldinger, R.: A Deductive Approach to Program Synthesis, TOPLAS, Vol.2, No.1, pp.90-121, 1980.

[6] 大野監修,原田編: 自動プログラミングハンドブック,オーム社, 1989.

[7] Reynols, J.C.: Transformational systems and the algebraic structure of atomic formulas, Machine Intell., Vol.5, pp.135-153, 1970.

[8] Shapiro, E.Y.: Inductive Inference of Theories From Facts, Technical Report 192, Yale Univ., 1981.(有川節夫訳,知識の帰納的推論,共立出版,1986).

[9] Shapiro, E.Y.: Algorithmic Program Debugging, Ph.D. Thesis, The MIT Press, 1982.

[10] Summers, P.D.: A methodology for LISP program construction from examples, J.ACM, 24, pp.161-175, 1977.

[11] Togashi, A., Noguchi, S.: A Program Transformation from equational Programs into Logic Programs, J. Logic Programming, 4, pp.85-103, 1987.

[12] 富樫: 関数型プログラミングにおける計算モデル,情報処理, Vol.29, No.8, pp.817-828, 1988.

[13] 富樫,千葉,野口:代数を実現する項書換えシステムの帰納的推論,人工知能学会,Vol.6,No.4, 1991

[14]Robin Milner : A Calculus of Communicating Systems, Lecture Notes in Computer Science 92, Springer-Verlag, 1980

[15]Robin Milner : Calculi for Synchrony and Asynchrony, Journ. Theoret. Computer Science, Vol.25, pp.267-310,1983

[16]Matthew Hennessy, Robin Milner : Algebraic Laws for Nondeterminism and Concurrency, Journal of the ACM, Vol 32, No.1, pp137-161, 1985

[17]Robin Milner : Communication and Concurrency, Prentice-Hall, 1989.

[18]Robin Milner : Operational and Algebraic Semantics of Concurrent Processes, Handbook of Theoretical Computer Science, North-Holland, 1990.