マルチメディア通信と分散処理 91－18
（1999. 1. 22）

文脈自由プロセスに対するプロトコル合成の一手法

中田 明夫† 東野 輝夫‡ 谷口 健一‡

†広島市立大学 情報科学部 情報数理学科
〒731-3194 広島市安佐南区大塚東 3-4-1
‡大阪大学 大学院基礎工学研究科 情報数理系専攻
〒560-8531 大阪府豊中市待兼山町 1-3

本論文では文脈自由プロセスのクラスで記述された分散システムのサービス仕様からプロトコル仕様を合成する一手法を提案する．文脈自由プロセスとはISOによって標準化された分散システムの形式仕様記述言語LOTOSの部分クラスの一つで，プロセス定義式が文脈自由文法の形式をしているものを指す．まず，文脈自由プロセスで記述されたサービス仕様に記述されたすべてのイベントに対してイベントIDと呼ぶ一意の名前を割り振る．イベントIDは文脈自由プロセスの記述の構文木から機械的に導出される有限長の記号列である．次に，文脈自由プロセスの各部分プロセスが最初および最後に実行可能なイベントのIDの集合が正規言語であることを示し，各イベントの実行順序の関係（イベント構造）を正規表現を用いて有限的に表現できることを示す．最後に，得られたイベント構造の有限表現を用いて，与えられた文脈自由プロセスのクラスのサービス仕様からプロトコル仕様を合成する方法を与える．

A Protocol Synthesis Method for Context-Free Processes

Akio NAKATA† Teruo HIGASHINO‡ Kenichi TANIGUCHI‡

†: Dept. of Computer Science, Faculty of Information Sciences, Hiroshima City University,
Ozuka-higashi 3-4-1, Asaminami-ku, Hiroshima 731-3194, Japan
‡: Dept. Information and Computer Sciences, Faculty of Engineering Science,
Osaka University, Machikaneyama 1-3, Toyonaka, Osaka 560-8531, Japan

In this paper, we propose a protocol synthesis method for the class of context-free processes. First, we assign a unique name called an event ID to every executable event in a given service specification written in context-free processes. The event ID is a finite sequence of symbols derived from the context-free process. Then we show that the sets of the first and last executable events in each subprocess can be expressed by regular expressions on the symbols, and that the partial order of events (called event structures) can be finitely represented by a set of relations among the regular expressions. Finally, we present a method to derive a protocol specification which implements a given service specification on distributed nodes, by using the obtained finite representation of event structures.

1 まえがき

プロトコル合成問題[1]とは，システムのサービス仕様からそのサービスをネットワークで繋いだ複数の計算機上で実現するために必要なメッセージ交換動作を記述した各計算機毎の仕様（プロトコル仕様）を自動導出する問題である．プロトコル合成問題は，特に通信路にエラーやメッセージ消失がないことを仮定したとき，サービスの各動作間の半順序関係を分散環境において保証することに帰着される．すなわち，順序関係の存在しない動作群は各計算機で独立に並列実行し，順序関係が存在する動作間の順序は，先行動作を実行し終わった計算機からその直接後続動作を実行する計算機へメッセージを送信することによって保証すればよい．従来，さまざまな仕様記述モデルに対して，一定の前提の下でプロトコル合成問題を解く方法が提案されてきた[2, 3, 4, 5, 6, 7, 8]．特に，ISOにより標準化された仕様記述言語LOTOS[9]は，FSMなどのモデルよりも記述能力が高く実用的であることから，LOTOSを対象としたプロトコル合成は非常に有用である．

しかし，従来のLOTOSを対象としたプロトコル合成法には以下のような問題があった．文献[6]ではFSMに変換可能なLOTOS仕様のみが対象である．文献[7]ではプッシュダウンスタックが記述可能なクラスに対する導出法が与えられているが，サービス仕様の記述に強い制約があり，また，サービス仕様の構文要素毎に局所的にプロトコル仕様を導出するため，大域的に見たときに無駄なメッセージ交換が生じてしまう問題を抱える．各動作の半順序構造を大域的に解析し，各動作の直接先行動作および直接後続動作を同定することができれば，順序の保証に必

要な最小限のメッセージ交換のみを含むプロトコル仕様が合成可能となると思われる．

本論文では，LOTOSのサブクラスでFSMよりも広いクラスである文脈自由プロセスの動作式から半順序に基づくプロセスモデルの一つであるイベント構造 (Event Structures[10, 11, 12]) の有限表現を導く方法を提案し，それを用いた文脈自由プロセスのクラスに対するプロトコル合成法を提案する．イベント構造とはプロセスが実行可能なイベントの集合に因果関係 (causal relation) および競合関係 (conflict relation) が定義されたものである [10]．因果関係はイベントの実行順序の前後関係を表す半順序関係で，競合関係は2つのイベントのうちどちらか一方のみが排他的に実行されるという関係である．イベントとは動作の実行系列における出現 (occurrence) である．例えば実行系列 abac において動作 a の最初の出現と2番目の出現は異なるイベントとして区別される．まず，動作式で表現されたプロセスが実行可能なすべてのイベントにユニークな名前 (イベント ID) を付加する．イベント ID は与えられたプロセス動作式の構文木から特定の規則で導出される記号の有限系列で表現される．

本論文では，まず我々の定義におけるイベント ID の集合が正規言語，すなわち，有限オートマトンで認識可能であり，いくつかの興味のあるイベント集合に対してその有限表現である正規表現を機械的に求めることができることを示す．さらに，任意の文脈自由プロセスの構文木からイベント構造をイベント ID の正規表現に対する半順序関係として有限的に表現可能であることを示す．そして，そのイベント構造の有限表現を用いて任意の文脈自由プロセスに対して適用可能なプロトコル合成アルゴリズムを示す．

本論文の構成は以下の通りである．2章では記述言語のクラスである文脈自由プロセスを定義する．3章ではプロセス記述の構文木を用いることによりイベントに構造的な名前，イベント ID を付ける方法を提案し，文脈自由プロセスに対してはイベント ID の集合が正規言語であることを示す．4章ではイベント ID の正規表現の間に半順序関係などのイベント構造 (記号的イベント構造) を導入する．5章では記号的イベント構造を用いたプロトコル合成法を提案し，従来できなかったクラスの例題に対する適用例を示す．6章では結論および今後の課題を述べる．

2 文脈自由プロセス

本章では文脈自由プロセスの動作式の構文およびその操作的意味を定義する．

定義 1 文脈自由プロセスの**動作式**の構文は以下の BNF で定義される：

$$B ::= stop | exit | (B) | P | a;B | B[]B | B >> B$$

ここで，a は任意の動作名である．動作名の全体集合を Act，動作式の集合を Bex とする． □

定義 2 **文脈自由プロセス仕様** $Spec$ とは三字組み $< Procs, S, Defs >$ である。ただし、$Procs$ はプロセス名の有限集合、S は初期プロセス名、$Defs \stackrel{\text{def}}{=} \{(P,B) | P \in Procs, B \in Bex\}$ はプロセス名から動作式への有限写像である。$(P,B) \in Defs$ を**プロセス定義式**と呼び、以下 $P := B$ と表記する． □

直観的には動作式 $stop$ は何も動作しないプロセス，$exit$ は正常終了を表す特別な動作 δ を行って停止するプロセス，$a;B$ は動作 a を行った後 B の動きを行うプロセス，$B_1[]B_2$ は B_1 と B_2 のどちらかを (最初の動作で) 選択実行するプロセス，$B_1 >> B_2$ は B_1 が正常終了 (つまり δ を実行) した後に B_2 を実行するプロセスである．また，P はプロセス定義式 $P := B$ で定義されたプロセスを呼び出し，定義式右辺の動作 B を行うプロセスである．文脈自由プロセス仕様 $Spec$ の動作は初期プロセス S の動作と同じである．

形式的には，文脈自由プロセス仕様の各動作式の操作的意味は図1の推論規則で与えられる．文脈自由プロセスとは直観的にはプロセスの定義式が文脈自由文法の形をしているものをいう．したがって文脈自由プロセスの有限長の動作系列 (トレース) の集合は文脈自由言語となる．

3 イベント ID

本章では文脈自由プロセスの動作式に対して動作式のすべてのイベントに対するユニークな名前付け (イベント ID) を定義する．

定義 3 文脈自由プロセス仕様のプロセス定義式 $P := B$ に対して，右辺の動作式の構文木を $T(B)$ としたとき，$T(B)$ の根ノードから他の任意のノードへのパスを $(P :=, op_{1_{d_1}}, op_{2_{d_2}}, \ldots, op_{n-1_{d_{n-1}}}, op_n)$ と表し，op_n(を頂点とする部分木が表す部分式) の P における**出現位置**と呼ぶ．ただし，任意の $k \in \{1, \ldots, n-1\}$ に対して $op_k \in \{[], ;, >>\}$，$d_k \in \{L, R\}$ であり，$k = n$ に対しては $op_n \in \{[], ;, >>\} \cup Act \cup \{stop, exit\}$ である．op_{k_L} $[op_{k_R}]$ は演算子 op_k を頂点とする左 [右] 部分木をたどることを意味する． □

例えば，プロセス定義式 $P := (a;Q)[]((b;exit) >> (a;(d;exit)))$ の部分式 $(d;exit)$ の P における出現位置はパス $(P :=, []_R, >>_R, ;_R, ;)$ で表し，プロセス呼び出し Q の出現位置は $(P :=, []_L, ;_R, Q)$ で表す (図2参照)．また，動作名 a の出現位置は $(P :=, []_L, ; L, a)$ および $(P :=, []_R, >>_R, ; L, a)$ の2個所である．以降では各出現位置に一意の番号を振り，番号で略記することがある．

定義 4 プロセス呼び出しの出現位置の系列で

$$(S :=, \ldots, P_1)(P_1 :=, \ldots, P_2) \ldots (P_{k-1} :=, \ldots, P_k)$$

という形をしているものを**プロセス呼び出しスタック**

$$\frac{true}{exit \xrightarrow{\delta} stop} \qquad \frac{a \in Act}{a;B \xrightarrow{a} B}$$

$$\frac{B_1 \xrightarrow{\alpha} B'}{B_1 [] B_2 \xrightarrow{\alpha} B'} \qquad \frac{B_2 \xrightarrow{\alpha} B'}{B_1 [] B_2 \xrightarrow{\alpha} B'}$$

$$\frac{B_1 \xrightarrow{\alpha} B' \quad \alpha \neq \delta}{B_1 >> B_2 \xrightarrow{\alpha} B' >> B_2} \qquad \frac{B_1 \xrightarrow{\delta} B'}{B_1 >> B_2 \xrightarrow{i} B_2}$$

$$\frac{B \xrightarrow{\alpha} B' \quad P := B \in Defs \quad \alpha \in Act \cup \{i, \delta\}}{P \xrightarrow{\alpha} B'}$$

図 1: 文脈自由プロセスの操作的意味

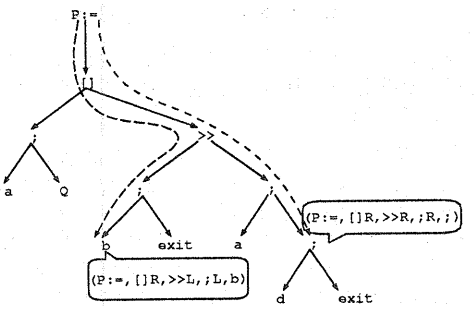

図 2: 構文木と出現位置

と呼ぶ. $(P_1, \ldots, P_k$ の中に重複があっても良い.) また, プロセス呼び出しスタック $(S :=, \ldots, P_1)(P_1 := , \ldots, P_2) \ldots (P_{k-1} :=, \ldots, P_k)$ の末尾にプロセス P_k における動作名 a(または $exit$) の出現位置を付加した系列を動作 a(または $exit$) のイベント ID と呼ぶ. □

例 1 以下はイベント ID の例である.

- $(S :=, []_L, >>_R, ;_L, a)$
- $(S :=, >>_R, P)(P :=, []_R, ;_R, exit)$
- $(S :=, \ldots, P)(P :=, \ldots, Q)(Q :=, \ldots, P)(P := , \ldots, a)$

□

イベント ID はある動作が実行された文脈 (構文上の出現位置) のみならず, 呼び出したすべてのプロセスの出現位置の履歴も含む. したがって, 文脈自由プロセス仕様 $Spec$ を 1 つ固定すれば, 同じイベント ID を持つ動作は高々1 回しか実行されない.

例 2 文脈自由プロセス仕様

$$\begin{aligned} Spec \;=\; & \langle \{S, P, Q\}, S, \\ & \{S := a; P, \\ & \;P := (b; Q) >> (a; P), \\ & \;Q := (c; P) [] (d; exit)\}\rangle, \end{aligned}$$

に対して, 以下のイベント ID の系列が実行可能である.

$(S :=, ;_L, a)$,
$(S :=, ;_R, P)(P :=, >>_L, ;_L, b)$,
$(S :=, ;_R, P)(P :=, >>_L, ;_R, Q)(Q :=, []_R, ;_L, d)$,
$(S :=, ;_R, P)(P :=, >>_L, ;_R, Q)(Q :=, []_R, ;_R, exit)$,
$(S :=, ;_R, P)(P :=, >>_R, ;_L, a)$,
$(S :=, ;_R, P)(P :=, >>_R, ;_R, P)(P :=, >>_L, ;_L, b)$,
...

上の系列において, 動作 b の最初の実行に対するイベント ID "$(S :=, ;_R, P)(P :=, >>_L, ;_L, b)$" と 2 番目の実行に対するイベント ID "$(S :=, ;_R, P)(P := , >>_R, ;_R, P)(P :=, >>_L, ;_L, b)$" がプロセス呼び出しスタックの違いによって区別されている. □

文脈自由プロセスのイベント ID に関して以下が成立する.

定理 1 文脈自由プロセス仕様 $Spec$ のすべてのイベント ID の集合は, 出現位置をアルファベットとする正規言語である.

[証明] 文脈自由プロセス仕様

$$\begin{array}{rcl} P_1 & := & \ldots P_i \ldots P_j \\ & & \quad k \quad l \\ & \ldots & \\ P_i & := & a; \ldots \\ & & \quad m \end{array}$$

(P_1 は初期プロセス名, k,l はそれぞれ P_1 におけるプロセス呼び出し P_i,P_j の出現位置, m は P_i における動作名 a の出現位置) から、プロセス名および動作名を状態、出現位置を遷移名, 初期プロセス名を初期状態とする決定性有限オートマトン (DFA)

$$\begin{array}{lll} P_1 & -k-> & P_i \\ P_1 & -l-> & P_j \\ & \ldots & \\ P_i & -m-> & a \end{array}$$

を構成できる。動作名を受理状態とすれば、受理言語はプロセス P_1 が発生可能なすべてのイベント ID の集合と一致する。 □

文脈自由プロセス仕様の任意の部分動作式 B の出現位置 p に対して，B が最初に [最後に] 実行可能な動作に対応するイベント ID の集合を開始イベント集合 $SE(p)$ [終了イベント集合 $EE(p)$] と呼ぶ．

定理 1 よりただちに $SE(p)$ および $EE(p)$ に対して以下の系が成立する．

系 1 $SE(p)$ および $EE(p)$ は正規言語である．

[証明] $SE(p)$ について証明する．定理 1 と同様に DFA を構成するが，構成の際に，$;_R$ や $>>_R$ を含むような出現位置を遷移名とする遷移は除外する．また，出現位置 p の部分式を B，B が右辺に現れるプロセス定義式を $P := B'$ とすると，新しい初期プロセス名 Q を導入し Q から B に現れるプロセス名や動作名への遷移を構成する．このとき遷移名にはもとのプロセス定義式 $P := B'$ における出現位置を用いる．こうして構成された DFA は $SE(p)$ を受理する．$EE(p)$ に関しては $;_L$ や $>>_L$ を含まない出現位置を考慮すれば良い． □

4 記号的イベント構造

イベント構造を用いると，プロトコル合成は，基本的に先行イベントが実行済であるようなイベント群を各ノードで独立に（イベント間に競合関係があればそれを調停して）並列実行し，実行済のイベントを直接後続イベントを実行するノードに通知するような仕様を導出すればよくなる．前節のイベント ID の概念を用いれば，そのようなイベント間の順序関係や競合関係をイベント ID の正規表現の関係として有限的に表すことが可能となり，上のようなプロトコル合成が可能となる．本節ではそのようなイベント構造の有限表現，記号的イベント構造を形式的に定義する．

定義 5 c をプロセス呼び出しスタック，r,r' をイベント ID の正規表現とし，$c.r$ を接頭語が c であり，c を除いた接尾語が r にマッチするようなイベント ID の集合としたとき[1]，**記号的イベント構造**とは以下のような 2 種類の関係の集合のことであると定義する:

> $c.r \longrightarrow c.r'$: $c.r$ に属するあるイベントの直後に $c.r'$ に属する任意のイベントが実行可能になる．(因果関係)
> $c.r \rightarrow\leftarrow c.r'$: $c.r$ に属する任意のイベントは $c.r'$ に属する任意のイベントと排他的に実行される．逆も同じ．(競合関係)

□

記号的イベント構造は文脈自由プロセス仕様 $Spec$ に現れる各演算子の出現位置毎に自動的に導出することができる．

例えば，$P := \ldots(a;B)\ldots$，すなわち，プロセス P の定義式右辺に現れる部分式 $(a;B)$ に対しては，

記号的イベント構造

$$c.(P :=, \ldots, ;_L, a) \longrightarrow c.Rex(SE((P :=, \ldots, ;_R, B)))$$

を導出する．ただし，$Rex(E)$ を正規集合 E の正規表現とする．

$P := \ldots(B[]C)\ldots$ のときは，

$$c.Rex(SE((P :=, \ldots, []_L, B))) \rightarrow\leftarrow c.Rex(SE((P :=, \ldots, []_R, C)))$$

$P := \ldots(B >> C)\ldots$ のときは，

$$c.Rex(EE((P :=, \ldots, >>_L, B))) \longrightarrow c.Rex(SE((P :=, \ldots, >>_R, C)))$$

となる．系 1 より，各 $SE()$,$EE()$ に対して機械的に DFA を構成し，対応する正規表現を機械的に求めることができる．

例 3 [分散スタック] 図 3 の例は，ノード A から動作 `push^A?x` により入力したデータ x をスタックに積み，ノード B で動作 `pop^B!x` によってスタックトップから順に取り出すバッファの文脈自由プロセス仕様である．データを取り出してスタックが空になれば停止する．最初に動作 `push^A?x` を実行するとき以外は常に現在のスタックトップを取り出す動作 `pop^B!x` および次のデータ y を積む動作 `push^A?y` とが排他的に実行可能であることが指定されている．

プロセス P の出現位置 k の演算子の左 [右] に現れる動作式の開始イベント ID 集合を $SE(P,k_L)$ [$SE(P,k_R)$] とする．同様に最終イベント ID 集合を $EE(P,k_L)$,$EE(P,k_R)$ とする．

各演算子の各出現が定めるイベント構造は以下の式でもれなく表すことができる．

[因果関係]

$$\begin{aligned} c.Rex(EE(S,8_L)) &\longrightarrow c.Rex(SE(S,8_R)) \\ c.Rex(EE(P,9_L)) &\longrightarrow c.Rex(SE(P,9_R)) \\ c.Rex(EE(P,11_L)) &\longrightarrow c.Rex(SE(P,11_R)) \\ c.Rex(EE(P,12_L)) &\longrightarrow c.Rex(SE(P,12_R)) \end{aligned}$$

[競合関係]

$$c.Rex(SE(P,10_L)) \rightarrow\leftarrow c.Rex(SE(P,10_R))$$

明らかに $Rex(EE(P,8_L)) = 1$ である．また，$Rex(SE(P,8_R)) = 2(3|5)$ と表せる[2]．

$Rex(EE(P,12_L)) = 6.Rex(EE(P))$ であり，$EE(P)$ に対応する DFA $M =< \{P, exit\}, \{P -4-> exit, P -7-> P\}, \{exit\} >$ を構成し，正規表現に直せば $Rex(EE(P)) = (7)^*4$ となるので，$Rex(EE(P,12_L)) = 6(7)^*4$ である[3]．

他も同様に求めると最終的に以下の記号的イベント構造が得られる．

[1] 正規表現 $(r.r')$ は r と r' の連接を表す．

[2] 正規表現 $(r|r')$ は r と r' の和集合を表す．

[3] 正規表現 $(r)^*$ は r の 0 回以上の繰り返しを表す

```
                8
S:=push^A?x;P(x)
   1        2

                9     10          11    12
P(x):=pop^B!x;exit [] push^A?y;P(y) >> P(x)
      3       4       5        6       7
```

各動作の出現位置: 1,3,4,5
各プロセス呼び出しの出現位置: 2,6,7
各演算子の出現位置: 8,9,10,11,12

図 3: 文脈自由プロセスによるサービス仕様の記述例

$$
\begin{array}{rcl}
c.1 & \longrightarrow & c.2(3|5) \\
c.3 & \longrightarrow & c.4 \\
c.5 & \longrightarrow & c.6(3|5) \\
c.6(7)^*4 & \longrightarrow & c.7(3|5) \\
c.3 & \rightarrow\leftarrow & c.5
\end{array}
$$

□

5 プロトコル合成

文脈自由プロセス仕様で記述された全体仕様 S とその記号的イベント構造および各動作のノードへの割り当てが与えられたとき，以下の方針で各ノードの動作仕様を導出する．

1. 各ノード i の動作仕様は基本的に全体仕様 S のノード i への射影 $Proj_i(S)$ である．すなわち，S のうちノード i で実行される動作以外を取り除き，以下の方針で必要なメッセージの送受信動作や補助的なデータ操作を付加したものが $Proj_i(S)$ である．
2. 全体仕様のプロセス定義式 $P := B$ に対しては，ノード i の仕様 $P_i(c) := Proj_i(B)$ を導出する．ただし，c はプロセス呼び出しスタックである．
3. 全体仕様のプロセス呼び出し P に対して，各ノード i のプロセス呼び出し $P_i(c.k)$ を導出する．ただし，k はプロセス呼び出し $P(x)$ の出現位置の ID である．プロセス呼び出しは基本的に各ノード毎に独立したタイミングで実行して良い．
4. 各動作の実行直前に (もしあれば) 先行イベント ID の受信を待つ．待つべき先行イベント ID の集合は一般に正規表現で表される．どの正規表現にマッチする先行イベント ID を待つかは，記号的イベント構造と呼び出された文脈 (一般に，プロセス呼び出しスタック c のトップから数個) から判断する．ただし，すべての先行イベント ID が自分と同じノードに属している場合および競合イベント ID が存在しない場合は除く．
5. 各動作の実行直後にその動作のイベント ID を送信する．ただし，$exit$ に対応するイベント ID は $exit$ の実行直前に送信する．ただし，すべての後続イベント ID が自分と同じノードに属する場合および競合イベント ID が存在しない場合は除く．
6. 選択実行 $B[]B'$ において B と B' の先頭イベント $e \in SE(B)$, $e' \in SE(B')$ が異なるノード i および j に属している場合は，補助的なプロセス $HaveToken_i(c.p, B)$ および $WaitToken_j(c.p, B')$ を用いて排他制御を行う．具体的にはノード i の仕様として

$$HaveToken_j(c.p, Proj_i(B))[]Proj_i(B'),$$

ノード j の仕様として

$$Proj_j(B)[]WaitToken_i(c.p, Proj_j(B'))$$

を導出する．ここで，c はプロセス呼び出しスタックで，p は選択演算子の出現位置である．また，B' [B] の先頭イベント集合にはノード i [j] に属しているものはないとする．$HaveToken_j(c.p, B)$ は記号列 $c.p$ をトークンとしてノード j に送って $WaitToken_j(c.p, B)$ を呼び出すか，動作式 B を実行するかどちらかをノード i でローカルに選択する．$WaitToken_i(c.p, B')$ はノード i からのトークン $c.p$ の受信を待ち，受信後は $HaveToken_i(c.p, B')$ を呼び出す．また，$Proj_i(B)$ の先頭イベントの実行後はノード j にそのイベント ID を送信し，ノード j は $Proj_j(B)$ の先頭でそれを受信し $Proj_j(B)$ の実行を継続する．逆の場合も同様である．

例 4 例 3 に対しては以下のようにプロトコル仕様を導出できる．まず，各イベントのノードへの割り当ては以下のようになる．

```
node A : c.1,c.5
node B : c.3,c.4
 where c=(2|6|7)*
```

前述の方針に基づけば図 4 のようなノード A,B の仕様が得られる (データの送受信に関しては手で加えた)．

```
[A]
S_A(c):=push^A?x;send_B(c.1,x);P_A(x,c.2)
P_A(x,c):=(if c=c'.7 then recv_B(c'.6(7)*4));
            (recv_B(c.3);exit
            [] HaveToken_B(c.10,
                push^A?y;send_B(c.5,y);P_A(y,c.6) >> P_A(x,c.7)))

[B]
S_B(c):=P_B(x,c.2)
P_B(x,c):=(if c=c'.2 then recv_A(c'.1,x)
          else if c=c'.6 then recv_A(c'.5,x));
           (WaitToken_A(c.10, pop^B!x;send_A(c.3);send_A(c.4);exit )
           [] P_B(y,c.6) >> P_B(x,c.7))

where  for i in {A,B}, p : occurrence position, P: behaviour expression
  HaveToken_i(c.p,P):=P[]send_i(c.p);WaitToken_i(c.p,P)
  WaitToken_i(c.p,P):=recv_i(c.p);HaveToken_i(c.p,P)
```

図 4: プロトコル仕様の導出例

□

6 あとがき

本稿では FSM より広いクラスである文脈自由プロセスのクラスに対するイベント構造を用いたプロトコル合成手法を提案した．今後の課題としては構文に並列合成を含むクラス，および，データや時間を含むクラスへの拡張が考えられる．

参考文献

[1] Probert, R. L. and Saleh, K.: Synthesis of Communication Protocols: Survey and Assessment, *IEEE Trans. Comput.*, Vol. 40, No. 4, pp. 468–475 (1991).

[2] Bochmann, G. v. and Gotzhein, R.: Deriving Protocol Specification from Service Specifications, in *Proc. of the ACM SIGCOMM '86 Symposium*, pp. 148–156, Vermont, USA (1986).

[3] Chu, P. M. and Liu, M. T.: Protocol Synthesis in a State Transition Model, in *Proc. IEEE COMPSAC '88*, pp. 505–512 (1988).

[4] Langerak, R.: Decomposition of Functionality; a Correctness-preserving LOTOS Transformation, in *Protocol Specification, Testing and Verification, X (PSTV-X)*, pp. 229–242, IFIP, Elsevier Science Publishers B.V.(North-Holland) (1990).

[5] Higashino, T., Okano, K., Imajo, H. and Taniguchi, K.: Deriving Protocol Specifications from Service Specifications in Extended FSM Models, in *Proc. of the 13th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS-13)*, pp. 141–148 (1993).

[6] Hultström, M.: Structural Decomposition, in *Protocol Specification, Testing and Verification, XIV*, pp. 201–216, IFIP, Chapman & Hall (1995).

[7] Kant, C., Higashino, T. and Bochmann, G. v.: Deriving Protocol Specifications from Service Specifications Written in LOTOS, *Distributed Computing*, Vol. 10, No. 1, pp. 29–47 (1996).

[8] Saleh, K.: Synthesis of Communication Protocols: An Annotated Bibliography, *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 5, pp. 40–59 (1996).

[9] ISO: *LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807 (1989).

[10] Winskel, G.: Event Structures, in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Vol. 354 of *Lecture Notes in Computer Science*, pp. 325–392, Springer-Verlag (1989).

[11] Langerak, R.: Bundle Event Structures: a non-interleaving semantics for LOTOS, in *Formal Description Techniques, V*, pp. 331–346, IFIP, Elsevier Science Publishers B.V.(North-Holland) (1993).

[12] Katoen, J.-P.: *Quantitative and Qualitative Extensions of Event Structures*, CTIT Ph.D-thesis series No.96-09, Centre for Telematics and Information Technology, Enshede, The Netherlands (1996).