時間制約の記述されたLOTOS仕様からのプロトコル合成

中田明夫[†] 東野輝夫[†] 谷口健一[†]

本論文では，分散システムの仕様記述言語LOTOSを時間制約を記述できるように拡張した言語LOTOS/T+のサブクラスで記述された全体仕様から，全体仕様と等価な働きをする各ノードの動作仕様を自動合成する1つの方法を提案する．LOTOS/T+では並列合成や割込みなどの構造的記述が可能であり，整数上の線形不等式の論理結合によって非隣接動作間の時間制約も記述可能である．提案する手法では，任意の2ノード間に通信路が存在し，各通信路の通信遅延はある定数で抑えられる，という仮定の下で，我々が定めた模倣方針に従って全体仕様 S の時間制約を満たす各ノードの動作仕様を導出できるかどうか調べ，模倣可能ならば導出する．具体的には，まず S と各通信路の通信遅延の最大値からノード間の同期メッセージの交換に要する時間制約を加えた仕様 S' を導出し，S と S' が時間性を無視したときに双模倣等価であるとき，S を正しく模倣する（複数ノードからなる）動作仕様群を S' から自動合成する．

Deriving Protocol Specification from Timed Service Specifications Written in LOTOS

AKIO NAKATA,[†] TERUO HIGASHINO[†] and KENICHI TANIGUCHI[†]

To design distributed systems, it is useful to synthesize protocol specifications from service specifications (protocol synthesis). However, the problem grows difficult if we consider time-constraints of systems. In this paper, we propose a method to synthesize protocol specifications automatically from service specifications written in a time-extended LOTOS called LOTOS/T+. In LOTOS/T+, structured descriptions, such as parallelism and interruption are allowed to describe service specifications, and time-constraints among non-adjacent actions can be described using Presburger formulas. Here we assume that there is a reliable communication channel between any two nodes and the maximal communication delay for each channel is bounded by a constant. Moreover we assume service specifications have no deadlocks. Under our simulation policy, a specification S' is derived from a given service specification S and a given maximal communication delay of each channel. In S', time-constraints necessary for exchanging synchronization messages are added. If S and S' can carry out the same behaviour, that is, if S and S' are bisimulation equivalent when time is ignored, then a correct protocol specification for simulating S is derived from S' automatically.

1. まえがき

分散システムの設計法の1つとして，まず分散システム全体を1つのシステムと考えたときの各動作の実行順序などを指定した全体仕様（サービス仕様）を記述し，その仕様から分散システムの各ノードの動作仕様の組（プロトコル仕様：各ノードの動作に加え，全体仕様で指定された順に動作を行うために必要なノード間での同期メッセージの交換などのやりとりを含む仕様）を機械的に合成する方法がある[1]．近年，そのような合成法がFSM，LOTOS[2]，ペトリネットなどのさまざまな並行計算モデルに関して提案されてきた[3)~8)]．しかし，それらはいずれもシステムの定量的な時間制約を考慮していなかった．一般に，定量的な時間制約が指定された全体仕様に対しても各ノードの動作仕様を導出できることが望ましい．近年，文献9)で時間制約付FSMで記述された全体仕様からプロトコル仕様を導出する手法が提案されている．しかし，FSMでは並行動作など複雑な実行順序が指定できないなどの問題がある．

本論文では，LOTOSに時間制約を記述できるように拡張した言語LOTOS/T+のサブクラスで記述された全体仕様から，各ノードの動作仕様を自動生成する手法を提案する．この言語では，並列動作や割込みなどの複雑な実行順序が指定できる．全体仕様の時間制

† 大阪大学基礎工学部情報工学科
Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University

約を等号や不等号を用いた論理式で比較的柔軟に記述できる，動作の実行時刻を記憶する変数を用いて後続の動作の実行時刻を制限できる，などの特徴を持つ．

本論文で提案する手法は，まず，任意の2ノード間に通信路が存在し，各通信路の通信遅延はある定数で抑えられる，という仮定をおく．その仮定の下で，時間デッドロック（その時刻以降指定されたいかなる動作も実行できないような状態）を含まないように時間約を指定した全体仕様と各動作のノードへの割当て，および，各2ノード間の通信遅延の最大値（定数）が与えられたときに，その全体仕様どおりに動作が実行されるように適当な模倣方針を1つ定め，その模倣方針に基づいて，通信遅延が時間制約を満たすために十分なものであるか否かを判定する．もし与えられた通信遅延において時間制約を満足することが可能ならば，全体仕様を分散環境で模倣する各ノードの動作仕様群を導出する．ここで，全体仕様を模倣する各ノードの動作仕様群とは，(A) 全体仕様で実行可能な動作はすべて実行できること（動作の実行時刻を無視して実行順序のみを考慮したときに全体仕様と双模倣等価であること），かつ，(B) 各動作の実行可能時刻の集合は対応する全体仕様の動作の実行可能時刻の集合に含まれること，の2点を満足する仕様であると定義する．

本手法は基本的には文献7), 10)と同様の模倣方針（時間制約がない場合の模倣方針）を用いる．すなわち，各動作 a の実行後，a の終了を通知する同期用メッセージ（必要なら a を実行した時刻などの情報も含める）を後続の動作 b を実行するノードに送り，そのメッセージを受信した後で b を実行するように各ノードの動作仕様を導出する．しかし，時間制約を考慮した場合には次のような問題が生じる．たとえば，もし「時刻3までにノード1で動作 a を行った後，時刻5までにノード2で動作 b を行う」という全体仕様が与えられ，ノード1からノード2までの通信遅延が最大で3単位時間かかる場合，時刻3に動作 a を行うと，同期用メッセージの交換に要する時間を考慮すると，動作 b を時刻5までに行えない（この場合時間デッドロックに陥る）可能性があり，前述の (A), (B) のいずれかを満足しないため，全体仕様の模倣を行うことができない．このような場合，たとえば先行する動作 a の時間制約を「時刻2まで」に変更すれば，通信遅延が3単位時間かかっても後続の動作 b が時刻5までに実行でき，全体仕様の模倣が可能になる．また，別の例として，「時刻1から3の間にノード1で動作 a を行った後，時刻4から5の間にノード2で動作 b を行う」という全体仕様が与えられたとする．もしノード1からノード2までの通信遅延が最大で3単位時間かかるとすると，a の実行後同期用メッセージを送信しても動作 b を時刻5までに実行できない．しかし，この場合は各ノードが自ノードにある時計を用いて各動作の実行時刻を決めれば，ノード間で同期用メッセージを交換しなくても，全体仕様で書かれた順に動作を実行することができる．このような模倣方針（以下，模倣方針Xとする）で全体仕様の分散実行が可能である（全体仕様を模倣する動作仕様群が存在する）とき，その動作仕様群を自動生成する．

考案した導出法では，まず，与えられた全体仕様 S と各ノード間の遅延の最大値から，上述のように模倣方針Xにおいて同期用メッセージの交換に必要な時間を考慮しても後続の動作がデッドロックに陥らないために各動作の時間制約をより厳しくした仕様 S' を導出する．S' の導出にはさまざまな方針が考えられるが，本論文では，先行動作の時間制約を（同期メッセージを省略しない模倣方針において）「同期メッセージが最も遅れた場合でも，その生起時刻が影響を及ぼすすべての後続動作の生起可能時刻が同期メッセージ受信後に存在するような最も条件の弱い（自由度の大きい）」時間制約になるように導出するという方針を採用する．この方針には導出が容易にできるという利点がある．導出された S' は一般に全体仕様 S より制限された仕様であるが，特に，S と S' が生起時刻を無視したとき（時間経過を表す *tick* 動作を観測不能な内部動作とみなしたとき）に外部から観測的に区別できないとき（双模倣等価），S' から S を満足する各ノードの動作仕様群を自動生成する．本手法では，動作仕様群が導出可能な場合，導出された動作仕様群が全体仕様のすべての動作を（一般に実行可能時刻の範囲は制限される）実現していることが保証される．

本論文は次のように構成される．まず，2章では全体仕様および各ノードの動作仕様の記述言語について述べる．3章では各ノードの仕様の合成問題の定義と合成アルゴリズムについて述べる．4章では，アルゴリズムに関する考察を述べる．最後に5章で結論と今後の課題について述べる．

2. 時間制約付 LOTOS — LOTOS/T+

全体仕様および導出される各ノードの動作仕様の記述言語としては，文献11)で提案した言語LOTOS/Tを一部拡張した言語 LOTOS/T+を用いる．以下にLOTOS/T+の構文と意味を示す．

定義1 LOTOS/T+の動作式およびその直観的な意味は**表1**のように定義される（演算子の優先順位

はLOTOSと同様)．ただし，表1において，$a \in Act \cup \{i\}$ (Act はすべての観測可能な動作の有限集合を，i は内部動作を表す)，$A \subseteq Act$，$k \in \mathbf{N}$，そして，$P(t,\bar{x})$ は，t (プロセスが実行を始めてからの経過時間を表す) と $\bar{x}$ ($\bar{x}$ は変数のベクトルを表す) を自由変数として持つプレスブルガー文[12]，すなわち，整数の大小比較と加減算のみを用いて記述された一階述語論理式を表す．e は式を，$\bar{e}$ は式のベクトルを表す．また，時刻は各プロセスごとに (自分自身を呼び出したときにはそれぞれのインスタンスごとに) 異なり，つねにそのプロセスが走り始めた時刻が0となる．□

LOTOS/T+では時間制約をプレスブルガー文で記述する．具体的には，$e_l \leq t$，$t \leq e_u$，$x = t$ という形の原子述語の論理結合で指定する．ここで，e_l は動作が生起可能な時刻の下界を表す整数上の線形式 (整数定数や変数を含んでもよい)，e_u は上界 (e_l，e_u の指定はそれぞれなくてもよい)，$x = t$ は動作の生起時刻を変数 x に代入することを表す．簡単のため $e_l \leq t \wedge t \leq e_u$ を $e_l \leq t \leq e_u$ と略記する．e_u が指定されている場合，この時間制約を指定された動作は遅くとも時刻 e_u までには必ず生起する，つまり，時刻 e_u における動作の緊急性 (urgency) が指定されていると解釈する．プレスブルガー算術の決定可能性[12]より，LOTOS/T+において各時刻における各動作の生起可能性，および，緊急性は機械的に判定できる．

表1 LOTOS/T+の構文
Table 1 Syntax of LOTOS/T+.

$E ::=$ **stop**	(停止)
| **exit**	(正常終了)
| $a; E$	(逐次実行，時間制約なし)
| $a[P(t,\bar{x})]; E$	(逐次実行，時間制約あり)
| $E[]E$	(選択)
| $E\|\|\|E$	(非同期並列)
| $E\|\|E$	(同期並列)
| $E\|[A]\|E$	(並列合成)
| $E[> E$	(割込み)
| $E >> E$	(逐次合成)
| **hide** A **in** E	(隠蔽)
| **asap** A **in** E	(即時実行の指定)
| $P[g_1,\ldots,g_k](\bar{e})$	(プロセス呼び出し)

例1

$$B = a[2 \leq t \leq 3 \wedge x_0 = t];$$
$$b[t = x_0 + 3];\ c[t = x_0 + 4];\ \mathbf{stop}$$

動作式 B は動作 a を時刻2から時刻3の間に必ず実行し x_0 に a の生起時刻を代入，そして，動作 b を a の実行時から3単位時間後に実行し，さらに動作 c を a の実行時から4単位時間後に実行するプロセスを表す．□

例2

(1) $E = a[x = t];\ b;\ c[t \geq x + 2];\ stop$

(2) $P = a[t = 5];\ stop[]b[t = 1];\ P$

最初の動作式は，時間制約のない動作が時間制約のある動作の間にはさまれている例である．時間制約のない動作 b はいつでも実行可能，すなわち，時間制約として $true$ が与えられていると見なされる．この例ではさらに，無限区間 ($t \geq x + 2$) が時間制約として与えられている．2番目の動作式は，再帰プロセスの記述例である．P が呼ばれるたびに時刻は0にリセットされる．対応するLTSを図**1**に示す．□

LOTOS/T+の形式的な意味は**表2**に示す遷移関係を導く推論規則で定義される．基本的にLOTOS/T+はLOTOS/T[11]と同じであるが，LOTOS/T+では **asap** A **in** E という構文を導入し式 E 中の動作 $a \in A$ は実行可能になった時点で即時実行されることを指定できるようにしている．表2では，述語 $P(0,\bar{x})$

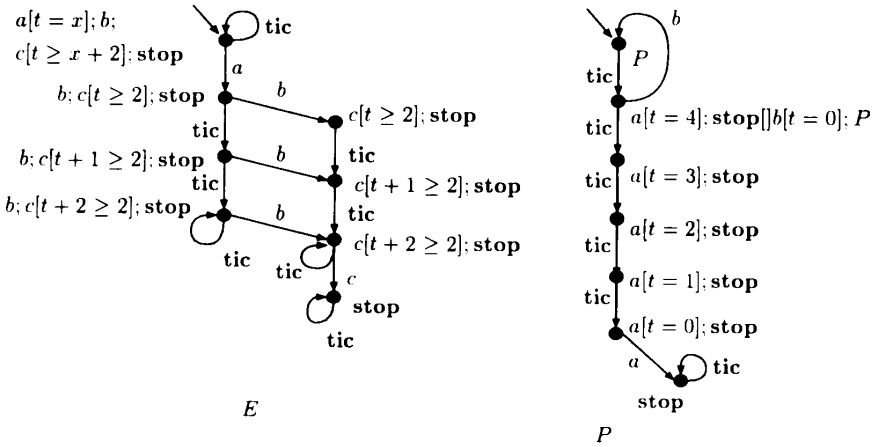

図1 動作式 E，P の意味
Fig. 1 Semantics of E and P.

表2 動作の遷移関係を導出する推論規則
Table 2 Inference rules of transition.

$$\frac{}{\mathbf{stop} \xrightarrow{\mathbf{tic}} \mathbf{stop}} \quad (S1)$$

$$\frac{}{\mathbf{exit} \xrightarrow{\delta} \mathbf{stop}} \quad (E1)$$

$$\frac{}{\mathbf{exit} \xrightarrow{\mathbf{tic}} \mathbf{exit}} \quad (E2)$$

$$\frac{P(0,\bar{e})}{a[P(t,\bar{x})];B \xrightarrow{a} [\bar{e}/\bar{x}]B} \quad (TAP1)$$

$$\frac{\exists t'\exists \bar{x}[t'>0 \wedge P(t',\bar{x})]}{a[P(t,\bar{x})];B \xrightarrow{\mathbf{tic}} a[P(t+1,\bar{x})];[t+1/t]B} \quad (TAP2)$$

$$\frac{}{a;B \xrightarrow{a} B} \quad (UAP1)$$

$$\frac{}{a;B \xrightarrow{\mathbf{tic}} a;[t+1/t]B} \quad (UAP2)$$

$$\frac{B_1 \xrightarrow{\beta} B_1'}{B_1[]B_2 \xrightarrow{\beta} B_1'} \text{ iff } \beta \in \mathrm{Act} \cup \{\delta, i\} \quad (CH1)$$

$$\frac{B_2 \xrightarrow{\beta} B_2'}{B_1[]B_2 \xrightarrow{\beta} B_2'} \text{ iff } \beta \in \mathrm{Act} \cup \{\delta, i\} \quad (CH2)$$

$$\frac{B_1 \xrightarrow{\mathbf{tic}} B_1' \quad B_2 \xrightarrow{\mathbf{tic}} B_2'}{B_1[]B_2 \xrightarrow{\mathbf{tic}} B_1'[]B_2'} \quad (CH3)$$

$$\frac{B_1 \xrightarrow{\mathbf{tic}} B_1' \quad B_2 \not\xrightarrow{\mathbf{tic}}}{B_1[]B_2 \xrightarrow{\mathbf{tic}} B_1'} \quad (CH4)$$

$$\frac{B_2 \xrightarrow{\mathbf{tic}} B_2' \quad B_1 \not\xrightarrow{\mathbf{tic}}}{B_1[]B_2 \xrightarrow{\mathbf{tic}} B_2'} \quad (CH5)$$

$$\frac{B_1 \xrightarrow{\beta} B_1' \quad B_2 \xrightarrow{\beta} B_2'}{B_1|[A]|B_2 \xrightarrow{\beta} B_1'|[A]|B_2'} \text{ iff } \beta \in A \cup \{\delta\} \quad (PA1)$$

$$\frac{B_1 \xrightarrow{\mathbf{tic}} B_1' \quad B_2 \xrightarrow{\mathbf{tic}} B_2'}{B_1|[A]|B_2 \xrightarrow{\mathbf{tic}} B_1'|[A]|B_2'} \quad (PA2)$$

$$\frac{B_1 \xrightarrow{a} B_1'}{B_1|[A]|B_2 \xrightarrow{a} B_1'|[A]|B_2} \text{ iff } a \notin A \vee a = i \quad (PA3)$$

$$\frac{B_2 \xrightarrow{a} B_2'}{B_1|[A]|B_2 \xrightarrow{a} B_1|[A]|B_2'} \text{ iff } a \notin A \vee a = i \quad (PA4)$$

$$\frac{B_1|[\emptyset]|B_2 \xrightarrow{\alpha} B'}{B_1|||B_2 \xrightarrow{\alpha} B'} \quad (PA5)$$

$$\frac{B_1|[\mathrm{Act}]|B_2 \xrightarrow{\alpha} B'}{B_1||B_2 \xrightarrow{\alpha} B'} \quad (PA6)$$

$$\frac{B_1 \xrightarrow{a} B_1'}{B_1[>B_2 \xrightarrow{a} B_1'[>B_2} \quad (DI1)$$

$$\frac{B_2 \xrightarrow{\beta} B_2'}{B_1[>B_2 \xrightarrow{\beta} B_2'} \text{ iff } \beta \in \mathrm{Act} \cup \{\delta, i\} \quad (DI2)$$

$$\frac{B_1 \xrightarrow{\delta} B_1'}{B_1[>B_2 \xrightarrow{\delta} B_1'} \quad (DI3)$$

$$\frac{B_1 \xrightarrow{\mathbf{tic}} B_1' \quad B_2 \xrightarrow{\mathbf{tic}} B_2'}{B_1[>B_2 \xrightarrow{\mathbf{tic}} B_1'[>B_2'} \quad (DI4)$$

$$\frac{B_1 \xrightarrow{a} B_1'}{B_1 >> B_2 \xrightarrow{a} B_1' >> B_2} \quad (EN1)$$

$$\frac{B_1 \xrightarrow{\delta} B_1'}{B_1 >> B_2 \xrightarrow{i} B_2} \quad (EN2)$$

$$\frac{B_1 \xrightarrow{\mathbf{tic}} B_1' \quad B_2 \xrightarrow{\mathbf{tic}} B_2' \quad B_1 \not\xrightarrow{\delta}}{B_1 >> B_2 \xrightarrow{\mathbf{tic}} B_1' >> B_2'} \quad (EN3)$$

$$\frac{B \xrightarrow{\beta} B'}{\mathbf{hide}\ A\ \mathbf{in}\ B \xrightarrow{\beta} \mathbf{hide}\ A\ \mathbf{in}\ B'} \text{ iff } \beta \in (\mathrm{Act} \setminus A) \cup \{\delta, i\} \quad (HI1)$$

$$\frac{B \xrightarrow{a} B'}{\mathbf{hide}\ A\ \mathbf{in}\ B \xrightarrow{i} \mathbf{hide}\ A\ \mathbf{in}\ B'} \text{ iff } a \in A \quad (HI2)$$

$$\frac{B \xrightarrow{\mathbf{tic}} B'}{\mathbf{hide}\ A\ \mathbf{in}\ B \xrightarrow{\mathbf{tic}} \mathbf{hide}\ A\ \mathbf{in}\ B'} \quad (HI3)$$

$$\frac{B \xrightarrow{a} B'}{\mathbf{asap}\ A\ \mathbf{in}\ B \xrightarrow{a} \mathbf{asap}\ A\ \mathbf{in}\ B'} \quad (ASAP1)$$

$$\frac{B \xrightarrow{\mathbf{tic}} B' \quad B \not\xrightarrow{\rho} \text{ for all } a \in A}{\mathbf{asap}\ A\ \mathbf{in}\ B \xrightarrow{\mathbf{tic}} \mathbf{asap}\ A\ \mathbf{in}\ B'} \quad (ASAP2)$$

$$\frac{[\bar{e}/\bar{x}]B\{g_1'/g_1,\ldots,g_k'/g_k\} \xrightarrow{\alpha} B'}{P[g_1',\ldots,g_k'](\bar{e}) \xrightarrow{\alpha} B'} \text{ iff } P[g_1,\ldots,g_k](\bar{x}) := B \text{ is a definition} \quad (PR1)$$

と $\exists t'\exists \bar{x}[t'>0 \wedge P(t',\bar{x})]$ の判定が機械的にできれば遷移の有無が機械的に判定できる．これらの述語はプレスブルガー文であるので，いずれも充足可能性を機械的に判定できる．したがって，表2の推論規則を適用すれば，各節点が動作式に対応するLTS（一般には状態数無限であるが）を機械的に構成できる．まず，例2のプロセス E に対する規則の適用例を以下に示す．

- $E = a[t=x];\ b;\ c[t \geq x+2];\ \mathbf{stop} \xrightarrow{a} b;\ c[t \geq 2];\ \mathbf{stop}$ [規則 (TAP1) より],
- $b;\ c[t \geq 2];\ \mathbf{stop} \xrightarrow{\mathbf{tic}} b;\ c[(t+1) \geq 2];\ \mathbf{stop}$ [規則 (TAP2) より],

例2のプロセス P に対しては次のようになる．

- $P \xrightarrow{\mathbf{tic}} a[t=4];\ \mathbf{stop}[]b[t=0];\ P$ [規則 (PR1), (CH3), (TAP2) より],
- $a[t=4];\ \mathbf{stop}[]b[t=0];\ P \xrightarrow{\mathbf{tic}} a[t=3];\ \mathbf{stop}$ [規則 (CH4), (TAP2) より]

3. プロトコル仕様の合成

3.1 合成問題の定義

本論文では，全体仕様からの各ノードの動作仕様を合成する問題を以下のように定義する．まず，準備としていくつかの記法を導入する．place(a) を動作 a が割り当てられたノードとする．以降，place(a) $= k$ であるとき，a を特に a^k と記述する．また，$SP(B)$, $EP(B)$, $AP(B)$ をそれぞれ，動作式 B の先頭動作を実行するノードの集合，B の最終動作を実行するノードの集合，B の動作に参加するすべてのノードの集合を表すものとする．たとえば $B =$

$v \in q$. Recall that every D_v contains the states of all resources in $\alpha(u)$ from the view of v. Intuitively, A_u is the set of resources currently available to u, as we will show in the next section.

If both of the above conditions hold, u arbitrarily selects a set of k resources from A_u, say R', sends a $\langle \texttt{LOCK}, u, C_u, R' \rangle$ message to every process $v \in q$, and accesses R'.

(2) **When a process u releases the set G_u of resources:**
It sends an $\langle \texttt{UNLOCK}, u, C_u, G_u \rangle$ message to every process $v \in q$.

(3) **When a process u receives a $\langle \texttt{RESPONSE}, v, D_v \rangle$ message from a process v:**
It stores D_v. If it has received an older version of D_v, it discards it and stores the latest one. Because messages are assumed to be delivered in order, u always holds the latest version among of the versions received so far.

(4) **When a process v receives a $\langle \texttt{QUERY}, u, t \rangle$ from a process u:**
If $W_v = \perp$, that is, if process v is not waiting for a $\langle \texttt{LOCK} \rangle$ message from another process, it sends a $\langle \texttt{RESPONSE}, v, D_v \rangle$ message to u, and sets $W_v := u$ and $T_v := t$. Recall that t is the logical time at the process u where the $\langle \texttt{QUERY} \rangle$ message was issued (see Case 1). Otherwise, $W_v = w$ for some process $w \in U$, that is, w waits for the two conditions in Case 1 to hold. If $T_v < t$, that is, if w has higher priority (since T_v is the timestamp attached to w's $\langle \texttt{QUERY} \rangle$), v stores $\langle \texttt{QUERY}, u, t \rangle$ to queue X_v. Otherwise, if $T_v > t$, u has the higher priority. Then, in order to preempt the right to lock resources, that v gave to w, v sends a $\langle \texttt{PREEMPT}, v \rangle$ to w, and waits for w to reply either with a $\langle \texttt{RETURN} \rangle$ or a $\langle \texttt{LOCK} \rangle$ message (see Cases 1 and 8), after storing the $\langle \texttt{QUERY} \rangle$ messages issued by u and w to X_v. When v again needs to send a $\langle \texttt{PREEMPT} \rangle$ to w while waiting for a reply from w, v ignores it.

(5) **When a process v receives a $\langle \texttt{RETURN}, w \rangle$ message from a process w:**
It takes the $\langle \texttt{QUERY}, x, t \rangle$ message from the top of queue X_v. This is the $\langle \texttt{QUERY} \rangle$ message that has the highest priority. Then, v sends a $\langle \texttt{RESPONSE}, v, D_v \rangle$ to x, and sets $W_v := x$ and $T_v := t$.

(6) **When a process v receives a $\langle \texttt{LOCK}, w, t, G_w \rangle$ message from a process w:**
It updates its data D_v by setting $D_v(r) := (w, t)$, for each $r \in G_w$. Then it continues (the algorithm fragment for) Case 5 if X_v is not empty.

(7) **When a process v receives an $\langle \texttt{UNLOCK}, w, t, G_w \rangle$ message from a process w:**
It updates its data D_v; it sets $D_v(r) := (\perp, t)$, for each $r \in G_w$. If $W_v \neq \perp$, it sends a $\langle \texttt{RESPONSE}, v, D_v \rangle$ message to W_v. Otherwise, it continues Case 5 if X_v is not empty.

(8) **When a process w receives a $\langle \texttt{PREEMPT}, v \rangle$ message from a process v:**
If it has sent back an $\langle \texttt{UNLOCK} \rangle$ message to v, it simply ignores the $\langle \texttt{PREEMPT} \rangle$ message. Otherwise, it sends back a $\langle \texttt{RETURN}, w \rangle$ message, and then discards the copy of D_v at w that was previously sent by a $\langle \texttt{RESPONSE}, v, D_v \rangle$ message from v. (Recall that w keeps D_v unchanged for each process v.) Then, w waits for another $\langle \texttt{RESPONSE} \rangle$ message from v. □

Although a $\langle \texttt{RESPONSE} \rangle$ message carries all of D_v in the above description of *AllocResource*, it is enough to carry the data on $\alpha(u)$ in D_v, since a process u will use information on $\alpha(u)$ in D_v.

5. Proof of Correctness

In this section, we show the correctness of *AllocResource*, provided that processes accessing resources release them within a finite time.

Since a process u selects the resources it accesses from the candidate set A_u, which is a subset of $\alpha(u)$, the following theorem holds.

Theorem 2 Algorithm *AllocResource* guarantees Allocation Validity. □

Before we proceed to the remaining properties, recall that a process u requesting k resources arbitrarily selects k resources from A_u determined from D_v's for $v \in q \in Q_u$. It then sends a $\langle \texttt{LOCK} \rangle$ message carrying the names of k resources to every v, after which it is free to access the k chosen resources. Process v, on the other hand, updates D_v in response

て，$I \stackrel{\alpha}{\Longrightarrow}_u I'$ かつ $I'\mathcal{R}S'$. □

正当性は以下のように定義される：

定義4 以下の条件が満たされるとき，導出された各ノードの動作仕様 $\{Node_i\}_{i=1,2,\ldots,n}$ は S に関して $\sqsubseteq_t$-**正当**であると呼ぶ：

$$\textbf{hide } G \textbf{ in } (\textbf{asap } G_s \textbf{ in } ((Node_1|||Node_2|||\ldots |||Node_n)|[G]|Medium)) \sqsubseteq_t S$$

□

3.2 合 成 法

S を分散環境で模倣する方法は文献7), 10)と基本的に同様の方法で行う．すなわち，全体仕様で規定されている動作の順序を分散環境でも保証するために，動作の終了（必要ならその動作を行った時刻も含める）を後続の動作を実行するノードに通信路を介して通知する．このような通知を**同期メッセージ**と呼ぶ．時間制約を考慮したとき，一般に通信路には遅延があり，しかも遅延時間は不確定であるため，終了通知が遅れた場合，通知を受け取った後では次の動作を時間制約を満たすように実行できない場合がある．この問題の解決策はいくつか考えられるが，本稿では次の方法をとる．全体仕様 S が与えられたとき，前節の制限4より，各動作の生起時刻が影響を与える後続動作の数は有限個で抑えられる．そこで最後尾の動作から順に各動作の時間制約を，同期メッセージの遅延の最悪値においても後続の動作の実行が時間制約を満足できるような時間制約に変更する．S に対してこの変更操作で得られる仕様を $Restr(S)$ で表す．そして，その変更された全体仕様 $S' = Restr(S)$ が S と時間性を無視したときに等価であれば，S' に基づいて各ノードの動作仕様を導出する．以下に全体仕様 S の各構成要素それぞれに対する分散環境における模倣方法および，$Restr(S)$ の定義を述べる．

3.2.1 逐次実行（Action Prefix）

一般に，逐次実行構文 $a^p[P(t,\bar{x})];B$ は，ノード p から $SP(B)$ に属すすべてのノードへ同期メッセージを送信することによって模倣できる．また，時間制約が変数への代入とその値の参照によって記述されているときには，変数への代入が起こったノードから，直接後続するノードへ代入された変数の値を伝搬させていく．

例3

$$S = a^1[x=t];b^2[t \le x+5 \wedge y=t];$$
$$c^3[t \le x+7 \wedge t \le y+5];\textbf{exit}$$
$$d_{12\max} = d_{13\max} = d_{23\max} = 2$$
$$Node_1 = a[x=t];s_{12}(m,x);\textbf{exit}$$
$$Node_2 = r_{12}(m,x);b[t \le x+5 \wedge y=t];$$
$$s_{23}(m',x,y);\textbf{exit}$$
$$Node_3 = r_{23}(m',x,y);$$
$$c[t \le x+7 \wedge t \le y+5];\textbf{exit}$$ □

時間性を考慮した場合，いくつかの冗長な同期メッセージを省くことができる．具体的には，与えられた時間制約から，もし先行動作の生起可能時刻が後続動作の生起可能時刻より必ず早いことが保証できるなら，（前提3より）時間制約のみによってこれらの動作の順序が保証されるため，これらの動作間の同期メッセージは必要ない．たとえば，$S = a^1[P(t,\bar{x})];\ b^2[Q(t,\bar{y})];$ **exit** とし，$\forall t,\ t',\ \bar{x},\ \bar{y}[[P(t,\bar{x}) \wedge Q(t',\bar{y})] \Rightarrow t < t']$ が真であると仮定する．その場合，時間制約より a はつねに b より先に実行される．よって，a と b を単純に異なるノードで並行に実行してもこれらの動作の順序は保たれる．一般に，動作式 $B = a^p[P(t,\bar{x})];B'$ において，B' のすべての先頭動作の時間制約の論理和をとった論理式を $TopTC(B',t,\bar{y})$ としたとき，論理式 $\forall t,\ t',\ \bar{x},\ \bar{y}[[P(t,\bar{x}) \wedge TopTC(B',t',\bar{y})] \Rightarrow t < t']$ が真であるならば，B の先頭動作 a^p は B' に対して**時間的に重なりがない**と呼ぶ．

例4 入力が以下のようなとき：

$$S = a^1[1 \le t \le 3];b^2[4 \le t \le 5];\textbf{exit}$$
$$d_{12\max} = 4,$$

$\forall t,\ t'[[(1 \le t \le 3) \wedge (4 \le t' \le 5)] \Rightarrow (t < t')]$ は真であるので，単純に以下のように各ノードの動作仕様を導出できる：

$$Node_1 = a[1 \le t \le 3];\textbf{exit}$$
$$Node_2 = b[4 \le t \le 5];\textbf{exit}$$ □

ここからは，通信遅延が模倣に影響を与える場合を考える．全体仕様 $S = a^p[P(t,\bar{x})];B$ に対して，ノード p から $SP(B)$ に属すノードに送信される同期メッセージが B に含まれる a^p のすべての後続動作のうち，a^p の生起時刻が影響を与えるものすべての生起可能時刻に間に合うように，a^p の時間制約をより制限した仕様 $Restr(S)$ を求める．制限4より，a^p の生起時刻はプロセス呼び出し後の動作には影響しない．したがって，まずプロセス呼び出し，**exit** および **stop** の直前の動作は制限を加えなくてもよい．そして，それらの動作から順に後ろから前に向かって直接後続動作に間に合うように逐次的に時間制約を制限していくことによって，上のような $Restr(S)$ を求めることが可能である．時間制約はプレスブルガー文で記述されているため，このような時間制約の制限は論理積をとることによって容易に行うことができる．

例5 入力が以下のようであるとき：

$$S = a^1[1 \le t \le 3]; b^2[4 \le t \le 7]; c^3[5 \le t \le 10]; d^2[6 \le t \le 12]; \mathbf{exit}$$

$$d_{12\max} = 4,\ d_{23\max} = 4,\ d_{32\max} = 3$$

S の時間制約をより制限した仕様 $Restr(S)$ を以下のように導出する．

d の時間制約： $6 \le t \le 12$ (変更なし)

c の時間制約： $5 \le t \le 10 \wedge \exists t'(t' \ge t + d_{32\max} \wedge 6 \le t' \le 12)$ $(\equiv 5 \le t \le 9)$

b の時間制約： $4 \le t \le 7 \wedge \exists t'(t' \ge t + d_{23\max} \wedge 5 \le t' \le 9)$ $(\equiv 4 \le t \le 5)$

a の時間制約： $1 \le t \le 3$ (変更なし（例4より))

よって，導出されるべき各ノードの動作仕様は以下のようになる☆：

$$Node_1 = a^1[1 \le t \le 3]; \mathbf{exit}$$

$$Node_2 = b^2[4 \le t \le 5]; s_{23}(m1); r_{32}(m2); d^2[6 \le t \le 12]; \mathbf{exit}$$

$$Node_3 = r_{23}(m1); c^3[5 \le t \le 9]; s_{32}(m2); \mathbf{exit}$$

□

$Restr(S)$ の形式的な定義を以下に与える．

定義 5 $S = a[Q(t,x)]; B$ のとき，$Restr(S)$ は以下のように定義される：

$$Restr(S) \stackrel{\text{def}}{=} Restr(S, \emptyset)$$

$$Restr(S,V) \stackrel{\text{def}}{=} \begin{cases} S & \text{if } B = \mathbf{exit}, B = \mathbf{stop} \\ & \text{or } B = P \\ & (P \text{ はプロセス呼び出し}), \\ a[Q(t,\bar{x}) \wedge Q'(t)]; & \\ \quad Restr(B, V \cup \bar{x}) & \\ & \text{otherwise.} \end{cases}$$

$$Q'(t) \stackrel{\text{def}}{=} \begin{cases} \bigwedge_{k \in K}\{\exists t' \exists y_k[(t' \ge t + d_{\text{place}(a),\text{place}(b_i)_{\max}} \wedge Q_k(t', y_k)] \\ \quad \text{if } \forall t, t', x, y[[Q(t,x) \wedge TopTC(B, t', y)] \Rightarrow t < t'] \text{ is false,} \\ \quad \text{or } B \text{ contains some variables of } V \cup \bar{x}, \\ true \qquad \text{otherwise.} \end{cases}$$

□

補足： 定義5によって，後続の動作が実行可能になるよう，再帰的に先行する動作の時間制約をきつくしている．すなわち，動作 a が後続動作に対して時間的に重なりがないならば，a の時間制約は，同期メッセージがすべての直接後続動作 b_k の実行可能時刻 $\{t' | \exists y_k Q_k(t', y_k)\}$ に間に合うように，各 k に対して $\exists t' \exists y_k[t' \ge t + d_{\text{place}(a),\text{place}(b_i)_{\max}} \wedge Q_k(t', y_k)]$ を満たす範囲の t に制限される．

本節をまとめると，提案する導出法は3段階からなる：

段階 1 同期メッセージが必要な箇所を特定する

段階 2 段階1の結果と $d_{ij\max}$ に基づいて，$Restr(S)$ を構成する．

段階 3 先に述べたように，文献7)，10)に似た方法を用いて $Restr(S)$ を各ノードに分解する．

3.2.2 選　　択

ここでは与えられた選択文 $B_1[]B_2$ に対して $Proj_k(B_1)$ [] $Proj_k(B_2)$ の形の各ノード k の動作仕様を導出することを考える．このような方針に従って選択を分散環境で模倣する場合，分散された選択 (distributed choice[7]) と空選択肢 (empty alternative[7]) の問題がある．選択文 $B_1[]B_2$ は，B_1 と B_2 の先頭動作が異なったノードで実行されうるとき，分散された選択と呼ばれる．また，もし B_i の中のある動作がノード p に割り当てられている一方，他方の $B_{i'}$ の中のいかなる動作もノード p に割り当てられていないならば，ノード p は空選択肢を持つという．分散された選択があれば，B_1 と B_2 の先頭動作が異なるノードで同時に実行されるという可能性が生じる．また，ノード p に対する空選択肢がある場合，$B_{i'}$ が選ばれたときにノード p にそのことを知らせる必要がある．分散された選択が生じないよう，我々は文献7)と同様に制限（制限6）をおく．制限6より，任意の選択文 $B_1[]B_2$ に対してあるノード p が存在し，$SP(B_1) = SP(B_2) = \{p\}$ が成り立つ．このことにより，ノード p が，B_i $(i = 1, 2)$ のうちどちらを実行するかをそのノード自身で一意に決定することができる．時間性を考えない場合の空選択肢の解決法は文

☆ この例の場合，次のように1つのイベントの時間制約を2つに分けるなど，より複雑な模倣法を用いれば異なる各ノードの動作仕様を導出できる：

$$Node_1 = a^1[1 \le t \le 3]; \mathbf{exit}$$

$$Node_2 = (b^2[4 \le t \le 5]; s_{23}(m1); \mathbf{exit}) [](b^2[6 \le t \le 7]; \mathbf{exit}) >> (r_{32}(m2); d^2[6 \le t \le 12]; \mathbf{exit}) []d^2[11 \le t \le 12]; \mathbf{exit})$$

$$Node_3 = (r_{23}(m1); c^3[5 \le t \le 9]; s_{32}(m2); \mathbf{exit}) [](c^3[t = 10]; \mathbf{exit})$$

しかし，簡単のため本稿ではこのような模倣方法は扱わない．

献7)で提案している．本稿ではそれを若干修正した方法を用いる．時間性を考えた場合，後続の動作ができるだけ時間制約を満たして実行できるように時間を稼ぐ必要があるため，本稿では，選択が起こったノードは開始時にただちに空選択肢を持つノードにどちらが選択されたかを通知することにする．また，空選択肢を持つノードへ送るメッセージが届く前に各 B_i が終了してしまわないように，選ばれた選択肢 B_i の最終動作を実行するノードは空選択肢を持つノードから受理確認のメッセージを受け取ることにする（ただし B_i の動作系列が長さ1の場合は B_i の先頭動作と最終動作が一致するので，ダミー動作を付加して対処する）．さらに，選択文 $B_1[]B_2$ を上の方法で模倣するため，原則として選択文では各 B_i に含まれる冗長な同期メッセージ（3.2.1項，例4参照）を除去しないことにする．さもなければ，各 B_i の先頭以外の動作が，どちらの選択肢が選ばれたかにかかわらず，勝手に別々のノードで実行される恐れがあるからである．

3.2.1項と同様に，上で述べたすべての同期メッセージが時間制約を満たして送信先に届くことを保証するためには，いくつかの動作の時間制約をより制限する必要がある．選択文 $B_1[]B_2$ に関しては，選択が行われたノードからのメッセージが選択された動作式 B_i の終了までに送信先に届かなければ，そのメッセージの受信のために余計な時間が消費される．したがって，我々はメッセージが時間内に届くように，各 B_i の先頭動作の時間制約をより制限する．

例6 以下の入力を考える：

$$S = a^1[2 \le t \le 5 \wedge x = t]; b^2[t \le x+3]; c^3[t \le x+6]; \mathbf{exit} \; [] \; d^1[3 \le t \le 9]; e^3[t \le 10]; \mathbf{exit}$$

$$d_{12\max} = 2,\ d_{23\max} = 4,\ d_{13\max} = 3$$

ノード1から2へ送信されるメッセージとノード2から3へ送信される受理通知が時刻10までに届くことを保証するためには，動作 d^1 の時間制約を制限する必要がある．

$$Restr(S) = a^1[2 \le t \le 5 \wedge x = t]; b^2[t \le x+2]; c^3[t \le x+6]; \mathbf{exit} \; [] \; d^1[\underline{3 \le t \le 4}]; e^3[t \le 10]; \mathbf{exit}$$

したがって，各ノードの動作仕様は以下のように導出される：

$$Node_1 = a^1[2 \le t \le 5 \wedge x = t]; s_{12}(m1,x); s_{13}(m3,x); \mathbf{exit} \; [] \; d^1[3 \le t \le 4]; (s_{13}(m2); \mathbf{exit} ||| s_{12}(m4); \mathbf{exit})$$

$$Node_2 = r_{12}(m1,x); b^2[t \le x+2]; s_{23}(m5); \mathbf{exit} \; [] \; r_{12}(m4); s_{23}(m4); \mathbf{exit}$$

$$Node_3 = r_{13}(m3,x); r_{23}(m4); c^3[t \le x+6]; \mathbf{exit} \; [] \; (r_{13}(m2); \mathbf{exit} ||| r_{23}(m4); \mathbf{exit}) >> e^3[t \le 10]; \mathbf{exit}$$

□

この導出法のために，時間的に先行する各動作に対して同期メッセージを除去しないこと以外は $Restr(S)$ と等価な $Restr'(S)$ を以下に定義する：

定義6 $Restr'(S)$ は以下のように帰納的に定義される：

もし，$S = a[Q(t,x)]; B$ ならば，

$$Restr'(S) \stackrel{\text{def}}{=} \begin{cases} S & \text{if } B = \mathbf{exit}, B = \mathbf{stop} \\ & \text{or } B = P \\ & (P \text{ はプロセス呼び出し}), \\ a[Q(t,x) \wedge Q''(t)]; Restr'(B) & \\ & \text{otherwise.} \end{cases}$$

ただし，$\{b_k[Q_k(t,y_k)] \mid k \in K\}$ を $Restr(B)$ の先頭動作の集合としたとき，

$$Q''(t) \stackrel{\text{def}}{=} \bigwedge_{k \in K} \{\exists t' \exists y_k [(t' \ge t + d_{\text{place}(a),\text{place}(b_i)\max} \wedge Q_k(t', y_k)]\}$$

さもなければ（$S \ne a[\ldots]; B$），

$$Restr'(S) \stackrel{\text{def}}{=} Restr(S).$$

□

以上の準備の下に，選択文に対する $Restr(S)$ を以下のように定義する：

定義7 $S = B_1[]B_2$ のとき，$Restr(S)$ は以下のように帰納的に定義される：

$$Restr(S) \stackrel{\text{def}}{=} Restr'(f(B_1))[]Restr'(f(B_2))$$

ただし，$\{b_k[Q_k(t,y_k)] \mid k \in K\}$ を B_i の先頭動作の集合とし，$f(B_i)$ を B_i の各先頭動作の時間制約 $Q_k(t,y_k)$ を $Q_k(t,y_k) \wedge R'_k(t)$ で置き換えた動作式とする．ここで，$R'_k(t)$ は以下のように定義されるプレスブルガー文である：

$$R'_k(t) \stackrel{\text{def}}{=} \bigwedge_{\substack{q \in AP(B_i) \setminus AP(B_{(i \bmod 2)+1}) \\ l \in L, r \in EP(B_i)}} \{\exists t' \exists z_l [(t' \ge t + d_{pq\max} + d_{qr\max} \wedge R_l(t', z_l)]\}$$

ただし，$\{R_l(t,z_l) \mid l \in L\}$ を B_i の各最終動作の時間制約とし，$SP(B_i) = \{p\}$ とする．

□

3.2.3 非同期並列

非同期並列では，2つのプロセスは互いに無関係に並行動作する．したがって，非同期並列の模倣に同期メッセージは不要である．$Restr(S)$ は以下のように自明に定義される．

定義8 $S = B_1|||B_2$ のとき，$Restr(S) \stackrel{\text{def}}{=}$

$Restr(B_1) \,|||\, Restr(B_2)$ □

3.2.4 逐次合成

逐次合成文 $B_1 >> B_2$ に対しては，制限3より，本質的には3.2.1項と同様の考え方を適用できる．相違点は動作間のみならずプロセス B_1，B_2 の間に時間制約を加える必要があるところである．このことは，B_1 の最終動作と B_2 の先頭動作との間に時間制約を加えることで実現する．

例7 以下の入力を考える：

$$S = (a^1[1 \le t \le 3]; \mathbf{exit} \,|||\, b^2[2 \le t \le 4]; \mathbf{exit}) >> (c^3[2 \le t \le 7]; \mathbf{exit} \,|||\, d^4[3 \le t \le 8]; \mathbf{exit})$$

$d_{13\max} = 5$, $d_{14\max} = 7$, $d_{23\max} = 4$, $d_{24\max} = 3$

動作 a^1 が c^3 や d^4 の前に実行されることを保証するためにノード1からノード3および4へ同期メッセージが送信される．したがって，そのメッセージがノード3へ時刻7までに，ノード4へ時刻8までに届くように a^1 の時間制約を制限する．動作 b^2 の時間制約に対しても同様に制限する．

$$Restr(S) = (a^1[\underline{1 \le t \le 1}]; \mathbf{exit} \,|||\, b^2[\underline{2 \le t \le 3}]; \mathbf{exit}) >> (c^3[2 \le t \le 7]; \mathbf{exit} \,|||\, d^4[3 \le t \le 8]; \mathbf{exit})$$

したがって，各ノードの動作仕様は以下のように導出される：

$$Node_1 = a^1[1 \le t \le 1]; \mathbf{exit} >> (s_{13}(m1); \mathbf{exit} \,|||\, s_{14}(m1); \mathbf{exit}) >> \mathbf{exit}$$

$$Node_2 = b^2[2 \le t \le 3]; \mathbf{exit} >> (s_{23}(m2); \mathbf{exit} \,|||\, s_{24}(m2); \mathbf{exit}) >> \mathbf{exit}$$

$$Node_3 = \mathbf{exit} >> (r_{13}(m1); \mathbf{exit} \,|||\, r_{23}(m2); \mathbf{exit}) >> c^3[2 \le t \le 7]; \mathbf{exit}$$

$$Node_4 = \mathbf{exit} >> (r_{14}(m1); \mathbf{exit} \,|||\, r_{24}(m2); \mathbf{exit}) >> d^3[3 \le t \le 8]; \mathbf{exit}$$

□

$Restr(S)$ の形式的定義は以下のようになる：

定義9 $S = B_1 >> B_2$ のとき，$Restr(S)$ は以下のように帰納的に定義される：

$$Restr(S) \stackrel{\text{def}}{=} Restr(g(B_1)) >> Restr(B_2)$$

ただし，$g(B_1)$ は B_1 の各最終動作 $a_k[P_k(t, x_k)]$ の時間制約 $P_k(t, x_k)$ を $P_k(t, x_k) \wedge P'_k(t)$ で置き換えた動作式である．ここで，$P'_k(t)$ は以下のように定義されるプレスブルガー文である：

$$P'_k(t) \stackrel{\text{def}}{=} \bigwedge_{l \in L} \{\exists t' \exists y_l [(t' \ge t + d_{\text{place}(a_k),\text{place}(b_l)\max}) \wedge Q_l(t', y_l)]\}$$

ただし，$\{b_l[Q_l(t, y_l)] \mid l \in L\}$ を $Restr(B_2)$ の先頭動作とその時間制約の集合とする． □

3.2.5 割込み

割込み文 $B_1[> B_2$ に対しては，我々は簡単化のため制限2をおく．これは，ある時刻 t_0 に対して B_1 のすべての動作は t_0 以降には実行できず，かつ，B_2 のすべての動作は t_0 以降でしか実行できないというものである．この制限により，B_1 の動作と B_2 の動作が同時に異なるノードで実行可能になることはない．このことと制限7より，$B_1[> B_2$ は B_1 の正常終了をすべてのノードに通知するメッセージ送受信動作を挿入し，すべてのノードは時刻 t_0 を過ぎた時点（時刻 $t_0 + 1$）で正常終了の通知を受け取っていないならば，内部動作 $i[t = t_0 + 1]$ によって割込みを発生させることによって模倣可能である．

この模倣方法を可能にするためには，B_1 の終了を通知するメッセージは時刻 t_0 までに届く必要がある．

例8 以下に示す入力は制限2および7を満たす（$t_0 = 11$）：

$$S = a^1[1 \le t \le 4]; b^2[3 \le t \le 8]; c^3[7 \le t \le 10]; \mathbf{exit} \; [> d^3[12 \le t]; e^2[14 \le t]; \mathbf{exit}$$

$d_{12\max} = 3$, $d_{23\max} = 4$, $d_{31\max} = 3$, $d_{32\max} = 4$,
その他の i, j に対して $d_{ij\max} = 2$.

ノード3からノード1および2へ送信される B_1 の終了通知が時刻 $t_0 = 11$ までに届くことを保証するために，動作 c^3 の時間制約を $7 \le t \le 7$ に制限する．そして，今までの節で述べた時間制約の制限が動作 b^2 および a^1 に適用される．

$$Restr(S) = a^1[\underline{1 \le t \le 1}]; b^2[\underline{3 \le t \le 4}]; c^3[\underline{7 \le t \le 7}]; \mathbf{exit} \; [> d^3[12 \le t]; e^2[14 \le t]; \mathbf{exit}$$

この結果から，各ノードの動作仕様は以下のように導出される：

$$Node_1 = a^1[1 \le t \le 1]; r_{31}(m1); \mathbf{exit} \; [> i[t = 12]; \mathbf{exit}$$

$$Node_2 = b^2[3 \le t \le 4]; r_{32}(m1); \mathbf{exit} \; [> i[t = 12]; r_{32}(m2); e^2[14 \le t]; \mathbf{exit}$$

$$Node_3 = c^3[7 \le t \le 7]; (s_{31}(m1) \,|||\, s_{32}(m1)) >> \mathbf{exit} \; [> i[t = 12]; d^3[12 \le t]; s_{32}(m2); \mathbf{exit}$$

□

$Restr(S)$ の定義は以下のとおりである：

定義10 $S = B_1[> B_2$ のとき，$Restr(S)$ は以下のように帰納的に定義される：

$$Restr(S) \stackrel{\text{def}}{=} Restr(g'(B_1))[> Restr(B_2),$$

ただし，$g'(B_1)$ は B_1 の各最終動作の時間制約 $P(t, \tilde{x})$ を $P(t, \tilde{x}) \wedge P'(t)$ で置き換えた動作式を表す．ここで，

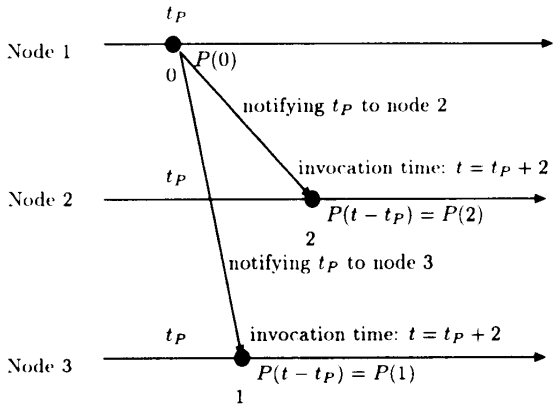

図2 分散環境におけるプロセス呼び出しの模倣
Fig. 2 Simulation of process invocation in distributed environment.

$$P'(t) \stackrel{\text{def}}{=} \bigwedge_{p \in EP(B_1), q \in ALL} \{t + d_{pq\max} \leq t_0\}. \quad \square$$

3.2.6 プロセス呼び出し

提案する仕様記述言語においては，プロセスが呼び出されるごとにそのプロセスの局所時計は0にリセットされる．このことによって時間制約が限りなく蓄積していくことを避け，本稿のような $Restr()$ の定義を可能にする．このようなプロセス呼び出しを分散環境で模倣するためには，すべてのノードが見かけ上プロセスを同時に起動するようにする．そのために我々は以下の模倣方法をとる（**図2** 参照）：

(1) **責任ノード**と呼ばれるノードを1つ決める．責任ノードはプロセスを呼び出す時刻（=プロセスを呼び出す直前の時刻）を決定する．本稿では，制限5よりプロセス呼び出しの文脈は $a;P$ か $a[P(t,x)];P$ の形に限られる．よって，place(a) をプロセス P の呼び出しに関する責任ノードと決める．

(2) 責任ノードはプロセスを呼び出す時刻を他のすべてのノードに対して通知し，ただちに自ノードにおいてプロセスを呼び出す．

(3) 責任ノード以外のノードは，プロセスを呼び出す時刻の通知を受けると，プロセスの開始時刻が仮想的に責任ノードにおけるそれと等しくなるように時間制約を変更したプロセスを呼び出す．

この模倣方法の実現のために，全体仕様において（制限4より）プロセスパラメータを持たない各プロセス P を，各ノードの仕様においてはパラメータ e_P を持つプロセス $P(e_P)$ に変更する．そして，P の定義式における変数 t のすべての出現を，$t+e_P$ に置き換える．パラメータ e_P はプロセス P の実際の呼び出し時刻と仮想的な呼び出し時刻（責任ノードにおける呼び出し時刻）との差を表す．たとえば，$P(3)$ はプロセス P の定義式の右辺の時間制約を，たとえば，$t \leq 5$ を $t+3 \leq 5$ に変更したプロセスを表す．そして，全体仕様の各プロセス呼び出し P に対して，以下のような各ノードの動作仕様を導出する：(1) 責任ノードは現在時刻 t_P を他のすべてのノードに送信し，ただちに $P(0)$ を呼び出し，(2) 他のすべてのノードは t_P を責任ノードから受信した後，ただちに $P(t-t_P)$ を呼び出す．$t-t_P$ は責任ノードからの通知に要した実際の通信遅延時間に対応する．

プロセス P は他のプロセス Q から呼び出される場合も存在する．この場合には，$P(t-t_P)$ の変数 t はプロセス Q の仮想的な呼び出し時刻に調整する必要がある．つまり，e_Q をプロセス Q に導入したパラメータとすると，Q の定義式の右辺に現れるすべての P の呼び出しは $P(t+e_Q-e_P)$ に変更する．

例9

$$P := a^1[2 \leq t \leq 4 \wedge x = t]; b^2[t \leq x+5]; P \;[]\; c^1[5 \leq t]; \mathbf{exit}$$
$$d_{12\max} = 4,\ d_{21\max} = 3$$
$$Node_1 = P(e_P) \text{ where}$$
$$P(e_P) := a^1[2 \leq t+e_P \leq 4 \wedge x = t+e_P \wedge \exists t'(t' \geq t+e_P+d_{12\max} \wedge t' \leq x+5)]; s_{12}(m1,x); r_{21}(m3,t_P); P(t+e_P-t_P) >> s_2(m4); \mathbf{exit} \;[]\; c^1[5 \leq t+e_P]; \mathbf{exit}$$
$$Node_2 = P(e_P) \text{ where}$$
$$P(e_P) := r_{12}(m1,x); b^2[t+e_P \leq x+5]; s_{21}(m2,x); s_{21}(m3,t+e_P); P(0) \;[]\; r_{12}(m4); \mathbf{exit} \quad \square$$

この模倣方法がどんな場合でも時間的に可能であることを保証するためには，各プロセスの先頭動作は責任ノードからの通知が最も遅れた場合でも実行可能であることが必要である．他の構文に対する導出法と整合性を保つために，我々はこのことのチェックを $Restr()$ の定義に含める．もしチェックの結果が偽ならば，$Restr()$ の結果としてプロセスの先頭動作の時間制約は $false$ に変更される．

定義11 $S = P$ where $P := B$ のとき，$Restr(S)$ は帰納的に以下のように定義される：

$$Restr(S) \stackrel{\text{def}}{=} P \text{ where } P := h(Restr(B))$$

ただし，$h(Restr(B))$ は $Restr(B)$ の各先頭動作 a_k の時間制約 $Q_k(t,x_k)$ を，それぞれ $Q_k(t,x_k) \wedge Q'_k$ に置き換えたものである．ここで，Q'_k は以下のように

$$
\begin{array}{rcl}
T_p(B) & = & T_p(B, \emptyset, p_0, true, empty, empty) \\
T_p(\mathbf{stop}, V, p', \rho, B', B'') & = & B'' >> \mathbf{stop} \\
T_p(\mathbf{exit}, V, p', \rho, B', B'') & = & B'' >> \mathbf{exit} \\
T_p(a^p; B, V, p', \rho, B', B'') & = & a^p; send_p(SP(B) \setminus \{p\}, N(a^p; B), V) >> B' >> T_p(B, V, p, \rho, empty, B'') \\
T_p(a^q; B, V, p', \rho, B', B'') & = & \begin{cases} receive_p(\{q\}, N(a^q; B), V) >> B' >> T_p(B, V, q, \rho, empty, B'') & \text{if } p \in SP(B) \\ B' >> T_p(B, V, q, \rho, empty, B'') & \text{otherwise.} \end{cases} \\
T_p(a^p[P(t,\bar{x})]; B, V, p', \rho, B', B'') & = & \begin{cases} a^p[P(t,\bar{x})]; send_p(SP(B) \setminus \{p\}, N(a^p[P(t,\bar{x})]; B), V \cup \bar{x}) \\ \qquad >> B' >> T_p(B, V \cup \bar{x}, p, \rho, empty, B'') \\ \qquad \text{if } \forall t, t', \bar{x}, \bar{y_k}[[P(t,\bar{x}) \wedge Q_k(t', \bar{y_k})] \Rightarrow (t < t')] \text{ is false, or } \rho = false \\ a^p[P(t,\bar{x})]; B' >> T_p(B, V \cup \bar{x}, p, \rho, empty, B'') \quad \text{otherwise.} \end{cases} \\
T_p(a^q[P(t,\bar{x})]; B, V, p', \rho, B', B'') & = & \begin{cases} receive_p(\{q\}, N(a^q[P(t,\bar{x})]; B), V \cup \bar{x}) >> B' >> T_p(B, V \cup \bar{x}, q, \rho, empty, B'') \\ \qquad \text{if } p \in SP(B) \text{ and} \\ \qquad \forall t, t', \bar{x}, \bar{y_k}[[P(t,\bar{x}) \wedge Q_k(t', \bar{y_k}] \Rightarrow (t < t')] \text{ is false or } \rho = false) \\ B' >> T_p(B, V \cup \bar{x}, q, \rho, empty, B'') \quad \text{otherwise.} \end{cases} \\
T_p(B_1 [] B_2, V, p', \rho, B', B'') & = & T_p(B_1, V, p', false, Alternative_p(B_1, B_2)) ||| B', Alternative2_p(B_1, B_2)) ||| B'') \\
 & & [] T_p(B_2, V, p', false, Alternative_p(B_2, B_1)) ||| B', Alternative2_p(B_2, B_1)) ||| B'') \\
T_p(B_1 ||| B_2, V, p', \rho, B', B'') & = & T_p(B_1, V, p', \rho, B', B'') ||| T_p(B_2, V, p', \rho, B', B'') \\
T_p(B_1 [> B_2, V, p', \rho, B', B'') & = & (T_p(B_1, V, p', false, B', B'') >> Rel_p(B_1)) \\
 & & [> T_p(B_2, V, p', B', false, Alternative_p(B_2, ALL)) ||| B', Alternative2_p(B_2, ALL)) ||| B'') \\
T_p(B_1 >> B_2, V, p', \rho, B', B'') & = & T_p(B_1, V, p', \rho, B', B'') >> Synch\_Left_p(B_1, B_2) \\
 & & >> Synch\_Right_p(B_1, B_2) >> T_p(B_2, V, \min EP(B_1), \rho, B', B'') \\
T_p(P[g_1, \ldots, g_k], V, p, \rho, B', B'') & = & send_p(SP(B_P) - \{p\}, N(P[g_1, \ldots, g_k], \{t\})) >> P[g_1, \ldots, g_k](0) \\
T_q(P[g_1, \ldots, g_k], V, p, \rho, B', B'') & = & receive_q(\{p\}, N(P[g_1, \ldots, g_k], \{t_P\})) >> P[g_1, \ldots, g_k](t_P - t) \\
 & & \text{iff a process definition } P[g_1, \ldots, g_k] := B_P \text{ exists.} \\
T_p(S \text{ where } P[g_1, \ldots, g_k] := B_P) & = & T_p(S) \text{ where } P[g_1, \ldots, g_k](e_P) := [t + e_P/t] T_p(B_P)
\end{array}
$$

図3 $T_p(B)$ の定義
Fig. 3 Definition of $T_p(B)$.

定義されるプレスブルガー文である：

$$Q'_k \stackrel{\text{def}}{=} \bigwedge_{p=1,\ldots,n} \{\exists t' \exists x_k [t' \geq 0 + d_{p,\text{place}(a_i)_{\max}} \wedge Q(t', x_k)]\}. \quad \square$$

3.3 合成アルゴリズム

合成アルゴリズムは大きく分けて2つの部分からなる．

(1) 与えられた全体仕様 S から $S' = Restr(S)$ を求める．

(2) S と S' が非時間的強双模倣等価，すなわち，$S \sim_u S'$ ならば，S' から各ノード i の動作仕様 $T_i(S')$ を生成する．さもなければ，導出は行わず，停止する．

ここで，非時間的強双模倣等価性 $\sim_u$ [11]は，動作の具体的な生起時刻を無視して動作の実行順序や実行可能性のみを考慮したときの等価性であり，形式的には以下のように定義される．

定義12 動作式上の関係 $\sim_u$ は以下の条件を満たす関係 $\mathcal{R}$ のうち最大の関係である：

- $\mathcal{R}$ は対称関係である．
- もし $B_1 \mathcal{R} B_2$ ならば，任意の $\alpha \in Act \cup \{i, \delta, \epsilon\}$ に対して，以下の条件が成り立つ：
  条件：もし，$B_1 \xrightarrow{\alpha}_u B'_1$ ならば，ある B'_2 が存在して $B_2 \xrightarrow{\alpha}_u B'_2$，かつ，$B'_1 \mathcal{R} B'_2$. $\square$

$Restr(S)$ の求め方は前節までにすべて与えた．S' から各 $T_i(S')$ を求めるアルゴリズムは以下のように定式化される．なお，以下では S の任意の部分式 B に対して $N(B)$ を B を同定する番号とし，q を $p \neq q$ である任意のノード，V を変数の集合とする．

定義13 全体仕様 B からノード p の動作仕様への写像 $T_p(B)$ を図3のように帰納的に定義する．なお，$T_p(B)$ の定義で用いられる補助的な関数は**図4**のように定義される． $\square$

例6の全体仕様 S に $Restr()$ を適用して得られた仕様 S' に対する定義13の関数 $T_p()$ の適用例を，$p = 1$ の場合について**図5**に示す．

補足：図3に現れる $T_p(B, V, p, \rho, B', B'')$ について，引数 B，V，p，ρ，B'，B'' はそれぞれ，全体仕様，既出変数の集合，責任ノード，冗長な同期メッセージを除去するか否かを表すフラグ，選択構文を模倣するために必要な，メッセージ交換動作を表す動作式，同じく選択構文を模倣するために必要な動作式で **stop** や **exit** の直前に挿入されるもの，を表す．

本アルゴリズムの正当性に関して，以下の定理を得る．

定理1 全体仕様 S に対して $S' = Restr(S)$ とする．もし $S \sim_u S'$ ならば，S' から導出された各ノードの動作仕様 $\{T_i(S')\}_{i=1,2,\ldots,n}$ は S に関して $\sqsubseteq_t$-正当である． $\square$

なお，一般に $T_p(B)$ によって導出される各ノードの動作仕様には冗長な部分が生じるが，文献4)と同様の手法によって，いくつかの冗長な部分を最適化することが可能である．

$$
\begin{aligned}
send_p(P,N,V) &= \text{if } P=\emptyset \text{ then } empty \\
&\quad \text{if } P=\{i,j,\ldots,k\} \text{ then } (s_{pi}(N,V);exit(V)|||\ldots|||s_{pk}(N,V);exit(V)) \\
receive_p(P,N,V) &= \text{if } P=\emptyset \text{ then } empty \\
&\quad \text{if } P=\{i,j,\ldots,k\} \text{ then } (r_{ip}(N,V);exit(V)|||\ldots|||r_{kp}(N,V);exit(V)) \\
Synch\_Left_p(B_1,B_2) &= \begin{cases} send_p((SP(B_2)\setminus\{p\}),N(B_1),\emptyset) & \text{if } p\in EP(B_1) \\ empty & \text{otherwise.} \end{cases} \\
Synch\_Right_p(B_1,B_2) &= \begin{cases} receive_p((EP(B_1)\setminus\{p\}),N(B_1),\emptyset) & \text{if } p\in SP(B_2) \\ empty & \text{otherwise.} \end{cases} \\
Rel_p(B) &= \begin{cases} send_p((\mathrm{Act}\setminus\{p\}),N(B),\emptyset)|||receive_p((EP(B)\setminus\{p\}),N(B),\emptyset) & \text{if } p\in EP(B) \\ receive_p(EP(B),N(B),\emptyset) & \text{otherwise.} \end{cases} \\
Alternative_p(B_1,B_2) &= \begin{cases} send_p(AP(B_2)\setminus AP(B_1),N(B_1)) & \text{if } p\in SP(B_1) \\ receive_p(SP(B_1),N(B_1)) >> send(EP(B_1),N(B_1)) & \text{if } p\in AP(B_2)\setminus AP(B_1) \\ empty \quad \text{otherwise.} \end{cases} \\
Alternative2_p(B_1,B_2) &= \begin{cases} receive_p(AP(B_2)\setminus AP(B_1),N(B_1)) & \text{if } p\in EP(B_1) \\ empty \quad \text{otherwise.} \end{cases}
\end{aligned}
$$

図4 図3で用いられる補助的な関数群
Fig. 4 Auxiliary functions used in Fig. 3.

$$
\begin{aligned}
T_1(S') &= T_1(a^1[2\le t\le 5\wedge x=t];b^2[t\le x+3];c^3[t\le x+6];\mathbf{exit} \\
&\quad []d^1[3\le t\le 4];e^3[t\le 10];\mathbf{exit},\emptyset,p_0,true,empty,empty) \\
&= T_1(B_1,\emptyset,p_0,false,Alternative_1(B_1,B_2), \\
&\quad Alternative2_1(B_1,B_2)) \\
&\quad []T_1(B_2,\emptyset,p_0,false,Alternative_1(B_2,B_1), \\
&\quad Alternative2_1(B_2,B_1)) \\
&\quad (\text{ただし, } B_1 \stackrel{\text{def}}{=} a^1[2\le t\le 5\wedge x=t];b^2[t\le x+3]; \\
&\quad c^3[t\le x+6];\mathbf{exit},\ B_2 \stackrel{\text{def}}{=} d^1[3\le t\le 4];e^3[t\le 10];\mathbf{exit}\ ) \\
&= T_1(B_1,\emptyset,p_0,false,empty,empty) \\
&\quad []T_1(B_2,\emptyset,p_0,false,s_{12}(N(B_1),\emptyset);\mathbf{exit},empty) \\
&= a^1[2\le t\le 5\wedge x=t];(s_{12}(N(B_1),\{x\});\mathbf{exit}||| \\
&\quad s_{13}(N(B_1),\{x\});\mathbf{exit}) >> T_1(b^2[t\le x+3];\ldots,\{x\},1, \\
&\quad false,empty,empty)) \\
&\quad []d^1[3\le t\le 4];s_{13}(N(B_2),\emptyset);\mathbf{exit} >> ((s_{12}(N(B_1),\emptyset); \\
&\quad \mathbf{exit}|||T_1(e^3[t\le 10];\mathbf{exit},\emptyset,1,false,empty,empty)) \\
&= \cdots
\end{aligned}
$$

図5 $T_p()$ の適用例 $(p=1)$
Fig. 5 Example of Application of $T_p()$ $(p=1)$.

4. 考　　察

本章では，前節までに提案した導出法の「時間制約の自由度」および「適用可能な全体仕様のクラス」に関する拡張について議論する．

4.1 時間制約の自由度について

前節までに定義した $Restr()$ は，同期メッセージを省略しない模倣方針において「その生起時刻が影響を及ぼすすべての後続動作を実行可能にするための最も自由度の大きい」時間制約になるように導出する．しかし，同期用メッセージの省略を考慮したとき，時間的に重なりがある時間制約に対しても，3.2.1 項の例4のような時間的に重なりがない制約に変更することによって模倣可能になる場合があり，通信路の遅延の最大値が大きい場合には，こちらの方が時間制約の自由度が大きくなることがある．ここでは，このような場合を考慮して時間制約の自由度をより大きくするための拡張について述べる．

まず，$TSync(t)$ を時刻 t が次の動作のどの生起可能時刻よりも小さいときに真となる述語と定義する．このとき，定義5の $Q'(t)$ に対して $\forall t[Q'(t)\Rightarrow TSync(t)]$ (同期メッセージが次の動作の生起可能時刻に間に合うような任意の生起時刻は次の動作の生起時刻より小さい) が成り立つときは同期メッセージを用いない方が自由度が大きくなる．したがって，定義5において，$\forall t[Q'(t)\Rightarrow TSync(t)]$ が真のときには $Q'(t)$ の代わりに $TSync(t)$ を用いて時間制約を厳しくして，同期メッセージを用いない模倣法に従えば，時間制約の自由度がより大きい模倣が可能になる．

4.2 適用可能なクラスの拡張について

本稿で導入した制限のほとんどはアルゴリズムの簡単化のためである．これらのほとんどは本質的な制限ではなく，本稿で提案する模倣法に強く依存したものである．また，提案する模倣法はすべての可能な方法のうちの1つにすぎない．したがって，より複雑な模倣法および導出アルゴリズムを採用することによって，多くの制限は除去できると思われる．しかし，ランデヴー（同期並列）に関する制限，および，割込みに関する制限を除去することは単純ではない．

まず，同期並列 $B_1|[G]|B_2$ に関しては B_1，B_2 中

の同期する動作およびその後続動作の時間制約がすべて $[e \leq t]$ の形（生起可能時刻に上限がない）であるような全体仕様に限定すれば，非同期並列の場合の導出法がそのまま適用可能である．

しかし，上記のクラスをこれ以上広げることは次の理由により困難である：

- 生起可能時刻が上限を含むと（たとえば $[e_1 \leq t \leq e_2]$ の形），同期する動作の同期可能な時刻の上限が B_1 や B_2 に局所的に記述された上限より一般に小さくなる．したがって，たとえ全体仕様としては同期可能な時刻が存在しても，同期する動作の直前の動作を遅く実行したとき，メッセージが実際に同期可能な時刻に間に合わない可能性がある．したがって，正当な動作仕様の導出を行うためには B_1 や B_2 を局所的に見て時間制約を厳しくするだけではなく，動作式を大域的に見る必要がある．
- 動作式を大域的に解析して同期する動作の組合せを判断し，それらの時間制約の論理積をとって動作の実際の生起可能時刻の解の範囲を求めることは，制約式も長くなり，また，プレスブルガー文の解の範囲を効率的に求めるアルゴリズムも知られていないため，実用的ではない．
- 動作式によっては同期する動作の組合せが動的に変わる場合もあり，解析がより複雑になる．

このクラスに該当しなくても，同期並列を展開して同期並列オペレータを含まない等価な有限長の動作式に変換できれば，もともとの全体仕様の構造は失われるものの，正当な各ノードの動作仕様を導出できる．

割込み構文 $B_1[> B_2$ に対する制限はかなり強いものであるが，理論的に，B_1 のいずれかの動作と B_2 の先頭動作が同時に実行可能ならば，一般に全体仕様との双模倣等価性は失われることが知られているので，B_1 の任意の動作と B_2 の先頭動作を同時に実行可能にならないようにするために，この種の制限は必要である．この制限の範囲内でも，たとえばタイムアウトメカニズムの多くは記述可能であり，一定の実用性はあると考える．

なお，B_1 の任意の動作と B_2 の先頭動作が同時に実行可能にならないような制限であれば，他の制限も可能である．たとえば，時間軸を一定長のタイムスロットに分割し，B_1 の実行ができず，B_2 のみが割込み可能であるようなスロットと，割込みができず B_1 のみが実行可能であるようなスロットが交互に繰り返すように書かれた全体仕様にも本手法は適用可能である．この場合，B_1 を実行中に B_2 が割込み可能な動作仕様を導出できる．

また，LOTOS の割込みの意味を若干修正して，割込み発生からその通知が到着するまでは B_1 の動作をしてもよいという意味に基づく場合は，割込みに関する時間制約は必要ない．

5. あとがき

本論文では，LOTOS の時間拡張である LOTOS/T+で記述された分散システムの全体仕様から各ノードの動作仕様を自動合成する方法を提案した．提案した方法によって時間制約が記述された LOTOS 仕様からのプロトコル合成が可能になる．文献 9) の提案とは対照的に，我々の方法では，通信路の時間制約を制限するのではなく，全体仕様の時間制約を制限する．これは，通信路の遅延は一般に物理的な回線に依存し，仕様の時間制約よりも変更が困難であると考えるからである．さらに，我々の正当性の基準は，導出された各ノードの仕様が全体仕様の部分的な実現ではなく，（非時間的に見た場合に）完全な実現であることを保証する．また，LOTOS 流の構造的な記述は状態遷移図に比べ，より理解しやすい．したがって，大規模なシステムの設計により有効であると考える．

本稿の結果は，LOTOS 以外に CCS などの他の体系にも我々と同様の時間制約の記述法を用いることによって，容易に適用可能であると考える．また，時間の領域が離散であるか密であるかは本稿の議論には影響しない．したがって，密時間の場合にも本稿の結果は適用可能である．

また，本稿では各ノードの時計が同期していることを仮定しているが，単位時間の粒度が各ノードの時計の誤差の範囲より小さいシステムへ適用する際，問題になる．しかし，分散ノードにおける時刻合わせの問題は以前から 13), 14) などで研究されており，最近では ISDN を用いて 10^{-11} 秒の精度で時刻合わせを行うシステムも発表されている[15]．したがって，少なくとも単位時間がミリ秒程度の粒度であるようなシステムにも本稿の結果を適用することは可能であると考える．

なお，時計の誤差が無視できないような分散システムの形式化が文献 16) でなされている．この文献の結果を用いて，時計の同期を仮定しない場合のプロトコル合成問題を考えるのも興味深い．

今後の課題は本導出法の適用可能なクラスを拡張すること，および，導出された仕様の効率の評価を行うことである．

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) ISO: Information Processing System, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, IS 8807 (1989).
3) Chu, P.M. and Liu, M.T.: Protocol Synthesis in a State Transition Model, *Proc. IEEE COMPSAC '88*, pp.505–512 (1988).
4) Gotzhein, R. and von Bochmann, G.: Deriving Protocol Specifications from Service Specifications including Parameters, *ACM Trans. on Computer Systems*, Vol.8, No.4, pp.255–283 (1990).
5) Higashino, T., Okano, K., Imajo, H. and Taniguchi, K.: Deriving Protocol Specifications from Service Specifications in Extended FSM Models, *Proc. the 13th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS-13)*, pp.141–148 (1993).
6) Hultström, M.: Structural Decomposition, *Protocol Specification, Testing and Verification, XIV*, Vuong, S.T. and Chanson, S.T. (Eds.), pp.201–216.
7) Kant, C., Higashino, T. and von Bochmann, G.: Deriving Protocol Specifications from Service Specifications Written in LOTOS, *Proc. 12th Annual Int'l Phoenix Conf. on Computers and Communications (IPCCC '93)*, IEEE, IEEE Computer Society, pp.310–318 (1993).
8) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K.: Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers, *Proc. 15th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS-15)* (1995).
9) Khoumsi, A., von Bochmann, G. and Dssouli, R.: On Specifying Services and Synthesizing Protocols for Real-time Applications, *Protocol Specification, Testing and Verification, XIV*, Vuong, S.T. and Chanson, S.T. (Eds.), pp.185–200.
10) Yasumoto, K., Higashino, T. and Taniguchi, K.: Software Process Description using LOTOS and Its Enaction, *Proc. 16th Int'l Conf. on Software Engineering (ICSE-16)*, pp.169–179 (1994).
11) Nakata, A., Higashino, T. and Taniguchi, K.: LOTOS Enhancement to Specify Time Constraints Among Nonadjacent Actions Using First Order Logic, *Formal Description Techniques, VI (FORTE '93)*, Tenney, R.L., Amer, P.D. and Uyar, M.Ü. (Eds.), IFIP, Elsevier Science Publishers (North-Holland), pp.451–466 (1994).
12) 東野輝夫, 北道淳司, 谷口健一：整数上の線形制約の処理と応用, コンピュータソフトウェア, Vol.9, No.6, pp.31–39 (1992).
13) Kopetz, H. and Ochsenreiter, W.: Clock Synchronization in Distributed Real-Time Systems, *IEEE Trans. Comput.*, Vol.C-36, No.8, pp.933–940 (1987).
14) Drummond, R. and Babaoglu, O.: Low-cost Clock Synchronization, *Distributed Computing*, Vol.6, pp.193–203 (1993).
15) 山下高生, 小野 諭：ISDN網を用いた分散高精度時刻/周波数同期, 情処研報 95-DPS-71-7, 情報処理学会 (1995).
16) 佐藤一郎, 所真理雄：分散計算のための局所時間性に基づく形式系, コンピュータソフトウェア, Vol.11, No.2, pp.32–44 (1994).
17) Vuong, S.T. and Chanson, S.T. (Eds.): *Protocol Specification, Testing and Verification, XIV (PSTV-XIV)*, IFIP, Chapman & Hall (1995).

(平成7年9月28日受付)
(平成8年3月12日採録)

中田 明夫

昭和44年生. 平成4年大阪大学基礎工学部情報工学科卒業. 平成6年同大学院基礎工学研究科物理系専攻博士前期課程修了. 現在同大学院博士後期課程在学中. 分散システム, プロセス代数, 時相論理などに興味を持つ.

東野 輝夫 (正会員)

昭和31年生. 昭和54年大阪大学基礎工学部情報工学科卒業. 昭和59年同大学院博士課程修了. 工学博士. 同年同大助手. 現在, 同大基礎工学部情報工学科助教授. 平成2年, 6年モントリオール大学客員研究員. 分散システム, 通信プロトコル等の研究に従事. 電子情報通信学会, IEEE_CS, ACM各会員.

谷口　健一（正会員）

昭和17年生．昭和40年大阪大学工学部電子工学科卒業．昭和45年同大学院博士課程修了．同年同大基礎工学部助手，現在，同情報工学科教授．工学博士．この間，計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法および支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法などに関する研究に従事．