マルチメディア通信と分散処理 69−14
（１９９５．３．３）

時間制約付 LOTOS 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の自動生成

中田明夫　東野輝夫　谷口健一

大阪大学基礎工学部情報工学科

分散システムの設計法の一つとして，分散システム全体の要求仕様 (全体仕様) から各ノードの動作仕様を自動生成する方法がある．しかし，システムの各動作に時間制約が指定された全体仕様に対してそのような合成を行なうことは，ノード間の通信遅延を考慮しなければならず一般に簡単ではない．本論文では，通信プロトコルの仕様記述言語 LOTOS を時間制約を指定できるように拡張した言語 LOTOS/T+ の部分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．LOTOS/T+は，並行動作や割り込みが記述できる，非隣接動作間の時間制約を不等式を用いた論理式で指定できる，などの特徴をもつ．分散システムの任意のノード間に通信路が存在し，各通信路の通信遅延の最大値が定数で抑えられるとする．また，全体仕様はデッドロックを含まない仕様であるとする．全体仕様の各動作の実行順や時間制約を満たすためにメッセージ交換をする方針 (模倣方針) を適当に定め，与えられた全体仕様 S と各ノード間の通信遅延の最大値から，ノード間の同期用メッセージの交換に必要な時間に関する時間制約を付加した仕様 S' を導出する．与えた模倣方針の下で S の分散実行が可能であるとき，すなわち，S と S' が生起時刻を無視した時に弱双模倣等価であるとき，S' から S を満足する各ノードの動作仕様を自動生成する．

Automatic Protocol Derivation from Timed Service Specifications written in LOTOS/T+

Akio Nakata　Teruo Higashino　Kenichi Taniguchi

Dept. of Information and Computer Science, Osaka University
Machikaneyama-cho 1-3,Toyonaka-shi, Osaka, 560 Japan

For designing distributed systems, protocol synthesis methods from service specifications have been studied so far. But as for real-time distributed systems, protocol synthesis is generally difficult because we must consider communication delay between any two distributed nodes. In this paper, we propose a method to synthesize protocol specifications automatically from service specifications written in time-extended LOTOS called LOTOS/T+. In LOTOS/T+, structured descriptions, such as Parallelism and Disabling are allowed to describe service specifications, and time-constraints among non-adjacent actions can be described using Presburger formulas. Here we assume that there are reliable communication channels 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 maximal communication delay of each channel. In S', time-constraints needed to exchange synchronization messages are added. If S and S' do the same behaviour, that is, S and S' are bisimulation equivalent when time is ignored, protocol entity specification for each node is derived from S' automatically.

1　まえがき

分散システムの設計法の一つとして，まず分散システム全体を一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 (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な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 [9]．近年，そのような合成法が EFSM, LOTOS 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 [5, 3, 2]．しかし，それらはいずれもシステムの定量的な時間性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 [4] 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+ (LOTOS/T [6] を一部変更した言語) 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, 並列動作や割り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.

提案する手法では, i) 通信路はメッセージ消失がなく, 遅延時間の上界が定数で与えられている, ii) 各ノードの動作開始時刻は等しく, 各ノードの時計の進む速さも等しい, とする. この仮定の下で全体仕様通りに動作を実行するための適当な模倣方針を一つ定め, 各ノードの動作仕様を導出する.

基本的には文献 [3] と同様の模倣方針に基づいている. 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. 例えば, もし「時刻3までにノード1で動作 a を行なった後, 時刻5までにノード2で動作 b を行なう」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 を行なうと, 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までに行なえない (この場合デッドロックに陥る) 可能性があり, 全体仕様の模倣がうまく行えない. このような場合, 例えば先行する動作 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 を行なった後, 時刻4から5の間にノード2で動作 b を行なう」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, 上記のような時間制約が指定された場合, 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で全体仕様 (それ自身はデッドロックを含まないと仮定する) 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, 上述の点を考慮して, 同期用メッセージの交換に必要な時間を時間制約として付加した仕様 S' を導出する. 導出する S' には自由度があるが, 必要最小限の時間制約のみを付加した S' になるよう工夫している. 与えた模倣方針で S の分散実行が可能であるとき, すなわち, S と S' が生起時刻を無視した時 (時間経過を表す tic 動作を観測不能な内部動作とみなした時) に弱双模倣等価であるとき, S' から S を満足する各ノードの動作仕様を自動生成する.

本論文は次のように構成される. まず, 2章では全体仕様及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の概略について述べる. 最後に4章で今後の課題について述べる.

表 1: LOTOS/T+の構文

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)
		| $E[>E$ (割り込み)
		| $E>>E$ (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)

2 時間制約付 LOTOS — LOTOS/T+

全体仕様及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 [6] 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+の構文と直観的な意味を示す.

定義 1 LOTOS/T+の動作式は表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) , $A \subset Act$, $k \in \mathbf{N}$, 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 [1], 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. □

例 1

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

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. $E = a[x = t]; b; c[t \ge x + 2]; stop$
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みなされる. この例ではさらに, 無限区間 ($t \ge x + 2$) 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. 一般に時刻は各プロセス毎に (自分自身を呼び出した時にはそれぞれのインスタンス毎に) 異なり, 常にそのプロセスが走りはじめた時刻が0となる[1]. 対応するLTSを図1に示す. □

LOTOS/Tとの違いは内部動作の挙動の解釈にある. 本稿で扱う導出法ではノード間でやりとりする内部メッセージの不確定な遅延を表現するため, LOTOS/T+では, 内部動作の生起時刻は時間制約の範囲で非決定的に定まるように意味定

[1] ただし, プロセスにパラメータとして変数 t を渡す場合は例外で, この場合は時刻はプロセスを呼び出した後にも継続する.

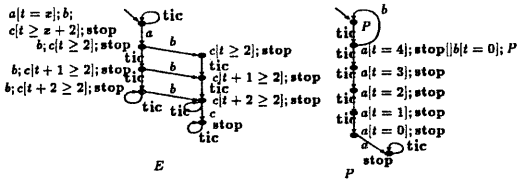

図 1: 動作式 E, Pの意味

義を行なった. しかし, プロセスの時間性を扱う多くの形式化では内部動作は最小遅延で, すなわち, 実行可能になった時点で即時に実行されると定義されている. 実行可能になった時点で即時に実行されるという意味も表現できる必要があると思われるので, LOTOS/T+では asap A in Eという構文によって, 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.

形式的な意味定義はLOTOS/T[6]とほぼ同様であるので追加変更部分のみを表 2に示す. 定義の残りは [6, 7] を参照されたい.

3 プロトコル仕様合成

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.

前提：
- 任意の 2 ノード間に信頼出来る（=エラーのない）双方向非同期通信路がある.
- 各ノードが交換できるメッセージの内容には制限を設けない.
- 各ノードの時刻は同期している. すなわち, 各ノードの時刻は任意の瞬間において等しい.

入力：
- 全体仕様 S. ただし, Sは以下の条件を満たすものに限る.

制限 1. Sはデッドロックを含まない.
並列動作は非同期並列のみ（ランデヴーを含まない）.

制限 2. Sの中の割り込み構文 $B_1[> B_2$の B_1は有限プロセスで, かつ, ある時刻 t_0以降,B_1は時間進行以外の如何なる動作も実行不能で B_2の先頭動作は t_0以降でのみ実行可能.

制限 3a. Sの中の 逐次合成構文 $B_1 >> B_2$ の B_1 は有限プロセス.

制限 3b. Sの中のすべてのプロセス呼出しはパラメータを持たない (プロセスを呼び出した後の動作は呼び出す前の動作に依存しない). プロセス呼び出しの直前の動作は唯一, すなわち, $a;P$または $a[P(t, \bar{x})]; P$という形のみに限定する.

- 各動作 a のノードへの割り当て place(a). ただし, place() は以下の条件を満たすものに限る (以下, a^k で place(a) $= k$を表す).

制限 4. Sにおいて選択可能な複数の動作 a_1,a_2があるとき,
place(a_1) = place(a_2).

- 各ノード i,j間の通信遅延の最大値 $d_{ij\max}$. ただし, $d_{ii\max} = 0$.

出力：以下の条件を満たすような, 各ノードの動作仕様 $Node_1$, $Node_2$, ... , $Node_n$, または, なし.

条件：
- $Node_1, Node_2, \ldots, Node_n$を通信路でつないで並列合成したプロセス ($I$とする) と全体仕様 Sは以下の関係 (双模倣性 [8] に基づく) を満たす (この関係の正確な定義は文献 [7] に示す).
  1. Iが行うことの出来るすべての動きを Sは生起時刻も含めて模倣できる.
  2. Sが行うことの出来るすべての動きを Iは生起時刻を無視して模倣できる（つまり, Iは Sの制御構造を完全に保有する）.

注：Iの各ノード間のメッセージ送受信は内部動作と見なす.

3.2 合成法

Sを分散環境で模倣する方法は [3] 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なった時刻も含める）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時, 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, このような模倣方法では次のような問題点が生じる. つまり, 終了通知が遅れた場合には, 通知を受け取った後では次の動作を時間制約を満たして実行することが不可能になる場合が一般に存在する. 本稿では, この問題点の解決法として次の方法をとる. 全体仕様 Sが与えられた時, 分散環境で模倣するときに同期メッセージの送受信が伴うような個所それぞれについて, 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最も自由度の大きい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 $S' = Restr(S)$ 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. アルゴリズムの詳細および正当性に関しては [7] に示す.

3.2.1 逐次実行 (Action Prefix)

逐次実行は, 同期メッセージで先行動作の終了を後続動作を実行するノードに通知することで模倣する.

例 3

全体仕様 $S = a^1[1 \le t \le 6]; b^2[5 \le t \le 8]$; **exit**
ノード間の遅延 $d_{12\max} = 2$

のとき,

ノード 1 の動作仕様 $Node_1 = a[1 \le t \le 6]; s_2(m)$; **exit**
ノード 2 の動作仕様 $Node_2 = r_1(m); b[5 \le t \le 8]$; **exit**

($s_i(m)$ はノード i にメッセージ m を送信する動作, $r_j(m)$ はノード jからメッセージ m を受信する動作を表す)

ノード 2 はノード 1 で動作 a が終ったという知らせを受信してから動作 b を実行する. このことによって a,b の順序が保証される. □

表 2: 動作の遷移関係を導出する推論規則 ([6] からの追加変更部分のみ)

Hide

$$\frac{B \xrightarrow{\beta} B'}{\text{hide } A \text{ in } B \xrightarrow{\beta} \text{hide } A \text{ in } B'} \text{ iff } \beta \in (Act - A) \cup \{\delta, i\} \quad (1)$$

$$\frac{B \xrightarrow{a} B'}{\text{hide } A \text{ in } B \xrightarrow{i} \text{hide } A \text{ in } B'} \text{ iff } a \in A \quad (2) \qquad \frac{B \xrightarrow{\text{tic}} B'}{\text{hide } A \text{ in } B \xrightarrow{\text{tic}} \text{hide } A \text{ in } B'} \quad (3)$$

Asap

$$\frac{B \xrightarrow{a} B'}{\text{asap } A \text{ in } B \xrightarrow{a} \text{asap } A \text{ in } B'} \quad (4) \qquad \frac{B \xrightarrow{\text{tic}} B' \quad B \not\xrightarrow{} \text{ for all } a \in A}{\text{asap } A \text{ in } B \xrightarrow{\text{tic}} \text{asap } A \text{ in } B'} \quad (5)$$

変数への値の代入と参照によって時間制約が指定されている場合は，今までに生起時刻などが代入されていて，その位置で値が利用可能なすべての変数の現在の値を同期メッセージでやりとりすることで実現する (LOTOS/T+では，構文 $a[P(t,\bar{x})]; B$における変数xは，Bの任意の位置で利用可能である)．

例 4

全体仕様 $S = a^1[x = t]; b^2[t \leq x+5 \wedge y = t]; c^3[t \leq x+7 \wedge t \leq y+5]$; exit

ノード間の遅延 $d_{12\max} = d_{13\max} = d_{23\max} = 2$

のとき，

ノード 1 の動作仕様 $Node_1 = a[x = t]; s_2(m,x); s_3(m,x)$; exit

ノード 2 の動作仕様 $Node_2 = r_1(m,x); b[t \leq x+5 \wedge y = t]; s_1(m',y)$; exit

ノード 3 の動作仕様 $Node_3 = r_1(m,x); r_2(m',y); c[t \leq x+5 \wedge t \leq y+5]$; exit

後続動作の生起可能時刻で先行動作のある生起可能時刻より小さいか等しいものが存在しないならばメッセージなしでも動作順序が保証される．例えば，$S = a^1[P(t,\bar{x})]; b^2[Q(t,\bar{y})]$; exit かつ,$\exists t,t',\bar{x},\bar{y}[P(t,\bar{x}) \wedge Q(t',\bar{y}) \wedge t' \leq t]$ が充足不能である時は，時間制約から常に a が b より先に実行されるため，動作 a,b をそれぞれノード 1,2 で並列に実行しても，全体仕様で指定された動作の順序が保たれる．したがって，このような場合には後続動作へのメッセージは省く．

例 5

全体仕様 $S = a^1[1 \leq t \leq 4]; b^2[5 \leq t \leq 7]$; exit

ノード間の遅延 $d_{12\max} = 2$

のとき，

ノード 1 の動作仕様 $Node_1 = a[1 \leq t \leq 4]$; exit

ノード 2 の動作仕様 $Node_2 = b[5 \leq t \leq 7]$; exit

□

次に通信遅延が全体仕様の模倣に影響がある場合を考える．上述の模倣方法を可能にするための時間制約の変更方法としては，次の方法をとる．すなわち，先行動作の実行後，同期メッセージが最も遅れた時でも，後続の動作の生起可能時刻が存在するように，先行動作の生起可能時刻をより制限したものを $Restr(S)$ とする．

例 6

全体仕様 $S = a^1[2 \leq t \leq 4]; b^2[5 \leq t \leq 7]; c^3[5 \leq t \leq 10]$; exit

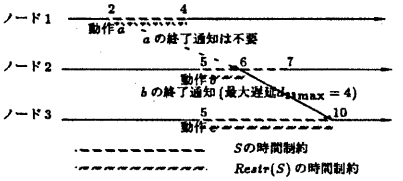

図 2: 時間制約の変更

ノード間の遅延 $d_{12\max} = 4$, $d_{23\max} = 4$

のとき，各動作の時間制約をそれぞれ次のように変更した全体仕様 $Restr(S)$ を出力する（図 2参照）．

a の時間制約：$2 \leq t \leq 4$　　(変更なし)

b の時間制約：$5 \leq t \leq 7 \wedge \exists t'(t' \geq t + d_{23\max} \wedge 5 \leq t' \leq 10)$ $(\equiv 5 \leq t \leq 6)$

c の時間制約：$5 \leq t \leq 10$　　(変更なし)　□

形式的な定義を以下に与える．

定義 2　$S = a[Q(t,x)]; B$のとき,$Restr(S)$ は以下のように帰納的に定義される：

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

ただし，$Restr(B)$ の各先頭動作 (時間制約式を含む) の集合を $\{b_k[Q_k(t,y_k)] \mid k \in K\}$ としたとき，

$$Q'(t) \stackrel{\text{def}}{=} \begin{cases} \bigwedge_{k \in K} \{\exists t' \exists y_k [t' \geq t + d_{\text{place}(a),\text{place}(b_k)\max} \\ \qquad \wedge Q_k(t',y_k)]\} \\ \quad \text{if ある } k \in K \text{に対して} \\ \qquad Q(t,x) \wedge Q_k(t',y_k) \wedge t' \leq t \text{ が充足可能}, \\ true \qquad \text{otherwise.} \end{cases}$$

□

定義 2では，一般に複数個ある次の動作のうちどれか 1 つに対して同期用メッセージが必要ならば，注目している動作の時間制約をよりきつくする．そのときの $Q'(t)$ の定義が論理積の形になっている理由は，全体仕様との双模倣性を保つことを意図して，すべての次の動作を実行可能であるようにするためである．

例 7　動作式 Bを

$$B = a^1[1 \leq t \leq 3 \wedge x = t]; b^2[t \leq x+3 \wedge y = t]; c^3[5+y \leq t]; P$$

としたとき，$Restr(B)$ は以下のように求められる．

$$Restr(B) = a[1 \leq t \leq 3 \wedge x = t \wedge Q^a(t)]; b[t \leq x+3 \wedge y = t \wedge Q^b(t)]; c[5+y \leq t \wedge Q^c(t)]; P)$$

ただし,

$$
\begin{aligned}
Q^c(t) &= true \\
Q^b(t) &= \exists t'[t' \geq t + d_{23}\max \wedge (5 + y \leq t' \wedge Q^c(t'))] \\
&= \exists t'[t' \geq t + d_{23}\max \wedge (5 + y \leq t')] \\
Q^a(t) &= \exists t' \exists y[t' \geq t + d_{12}\max \wedge \\
&\qquad (t' \leq x + 3 \wedge y = t \wedge Q^b(t'))] \\
&\quad \wedge \exists t' \exists y \exists t''[t' \geq t + d_{13}\max \wedge \\
&\qquad (t'' \leq x + 3 \wedge y = t'' \wedge 5 + y \leq t' \wedge Q^c(t'))] \\
&= \exists t' \exists y[t' \geq t + d_{12}\max \wedge (t' \leq x + 3 \wedge y = t \wedge \\
&\quad (\exists t''[t'' \geq t' + d_{23}\max \wedge (5 + y \leq t'')])] \\
&\quad \wedge \exists t' \exists y \exists t''[t' \geq t + d_{13}\max \wedge \\
&\qquad (t'' \leq x + 3 \wedge y = t'' \wedge 5 + y \leq t' \wedge Q^c(t'))]
\end{aligned}
$$

□

3.2.2 選択

例 8

全体仕様 $S = a^1[2 \leq t \leq 5 \wedge x = t]; b^2[\ldots]; c^3[\ldots]; \mathbf{exit}\ []d^1[3 \leq t \leq 7 \wedge y = t]; e^3[\ldots]; \mathbf{exit}$

ノード間の遅延 $d_{12}\max = \ldots$

のとき,

ノード 1 の動作仕様

$$
\begin{aligned}
Node_1 \;=\; & a[2 \leq t \leq 5 \wedge x = t \wedge \ldots]; s_2(m1, x); \mathbf{exit} \\
& []d[3 \leq t \leq 7 \wedge y = t \wedge \ldots]; \\
& (s_3(m2, y); \mathbf{exit} ||| s_2(m4); \mathbf{exit})
\end{aligned}
$$

ノード 2 の動作仕様

$$
\begin{aligned}
Node_2 \;=\; & r_1(m1, x); b[\ldots]; s_3(m3, \ldots); \mathbf{exit} \\
& []r_1(m4); s_3(m4); \mathbf{exit}
\end{aligned}
$$

ノード 3 の動作仕様

$$
\begin{aligned}
Node_3 \;=\; & r_2(m3, \ldots); c[\ldots]; \mathbf{exit} \\
& [](r_1(m2, y); e[\ldots]; \mathbf{exit} ||| r_2(m4); \mathbf{exit})
\end{aligned}
$$

□

- 複数のノードで選択可能な動作があるとアルゴリズムが複雑になる [5, 3] ので, [3] と同様に動作の選択が必ず 1 つのノードで起こる仕様に限定する. したがって, 任意の選択可能な 2 つの動作 a,d に対して place(a) = place(d) が成り立つような Sおよび place() に適用範囲が限られる (制限 4).
- 上の例ではノード 2 は 選択構文 $B_1[]B_2$ の片方 B_1のみにしか参加していない. この場合には B_2が終了した時にそのようなノードに対して B_2が選択されたことを通知するようにする [3] (上の例では $s_2(m4)$ および $r_1(m4)$). ただし, 文献 [3] とは異なり, $s_2(m4)$ のようなメッセージ送信は B_2の終了後ではなく, B_2の最初の動作を実行した直後に行なうようにし, また, このようなメッセージを受けとったノードは B_1の最終動作を実行するノードへ返事を返すようにする.

上の例のような模倣方法が可能であるためには, メッセージを送ってから返ってくるまでの時間が, 選択された動作式 B_2 をすべて実行終了するまでの時間より短い必要がある. さもなければ, メッセージを待つ時間が余分に加わって元の全体仕様の時間制約を満足できなくなる可能性を生じる.

すなわち,B_iが選択される時刻の最大値 (すなわち, B_iの先頭動作の生起可能時刻の最大値) と, B_iの先頭動作を実行するノード p から B_iに参加していないノード qを経由して B_iの最終動作を実行するノード rへとメッセージが達するまでの遅延の最大値が, B_iの最終動作の生起可能時刻の最小値より小さいことが, このような模倣を可能にする条件となる.

この条件を満たすようにするための $Restr(S)$ の形式的な定義は次のようになる.

定義 3 $S = B_1[]B_2$のとき, $Restr(S)$ は以下のように帰納的に定義される:

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

ただし, B_i $(i = 1, 2)$ に参加していて, $B_{(i \bmod 2)+1}$に参加していないノードの集合を$\Pi(S)$, $Restr(B_i)$ の先頭動作の集合を $\{b_k[Q_k(t, y_k)] \mid k \in K\}$ としたとき, $f(Restr(B_i))$ は各 $Q_k(t, y_k)$ を $Q_k(t, y_k) \wedge R'_k(t)$ で置き換えたものである. ここで, B_iの最終動作の時間制約式の集合を $\{R_l(t, z_l) | l \in L\}$ とし, p を B_iの先頭動作を実行するノード,$\Psi(B_i)$ を B_iの最終動作を実行するノードの集合とすると, $R'_k(t)$ は以下のように定義される論理式である:

$$
\begin{aligned}
R'_k(t) \;\stackrel{\text{def}}{=}\; & \bigwedge_{q \in \Pi(S), r \in \Psi(B_i), l \in L} \{\forall t' \forall z_l[R_l(t', z_l) \Rightarrow \\
& t' \geq t + d_{pq}\max + d_{qr}\max]\}
\end{aligned}
$$

□

3.2.3 逐次合成

逐次合成構文に対する導出は, 逐次実行の場合とほぼ同様であるので本稿では省略する. 合成法の詳細については文献 [7] を参照されたい.

3.2.4 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 4 $S = B_1|||B_2$のとき, $Restr(S) \stackrel{\text{def}}{=} Restr(B_1) |||$ $Restr(B_2)$ □

3.2.5 割り込み

本稿の対象とするクラスにおいては, 制限 2 より各ノードがある時刻 t_0までは B_1の動作を行い, t_0以降は B_2の動作を行なうことにより $B_1[> B_2$の模倣ができる. このため, 割り込み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 5 $S = B_1[> B_2$のとき, $Restr(S) \stackrel{\text{def}}{=} Restr(B_1)$ $[>$ $Restr(B_2)$ □

3.2.6 プロセス呼び出し

全体仕様においてはプロセスを呼び出す毎に時刻が 0 にリセットされる. これを分散環境でシミュレートする一つの方法として, 見かけ上, プロセスを各ノードで同時刻に起動したように動作させる. そのために (図 3参照),

1. 責任ノードと呼ばれるノードを 1 つ決めて, 責任ノードがプロセスの実行開始時刻 (プロセス起動直前の時刻) を決定する. 本稿では, プロセスを呼び出す直前の動作を実行するノード (制限 3b より一意に定まる) をその呼び出しにおける責任ノードに決める.

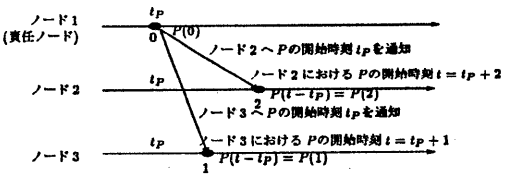

図3: プロセス呼び出しの分散環境におけるシミュレート方法

2. 責任ノードはプロセスの実行開始時刻を他の全ノードに通知する.
3. 他のノードは責任ノードからの通知を受け，プロセスの開始時の時刻（一般に責任ノードより遅れている）を見かけ上，責任ノードにおけるプロセスの開始時刻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[2].

上の3.を実現するため，全体仕様ではパラメータを持たないプロセス Pを1つのパラメータ e_Pをもつプロセス $P(e_P)$ に変更し，Pの定義式の右辺に出現するすべての変数 t を $t+e_P$ に変更する (この操作は Pの時間制約を見かけ上のプロセス開始時刻 $t+e_P$に対する制約に変換することに相当する). パラメータ e_Pはそのプロセスの実際の開始時刻と見かけ上の開始時刻との差を表す. 例えば $P(3)$ は Pの時間制約，例えば「$t \leq 5$」を「$t+3 \leq 5$」に変更したようなプロセスを表す. この変更操作は時間制約の変更を施した後の $Restr(S)$ に対して行う[3]. そして，全体仕様における Pの呼び出しに対して，責任ノードは現在時刻を他の全ノードに送信してから $P(0)$ を呼び出し，それ以外のノードでは責任ノードにおける Pの開始時刻 t_Pを受信してから，$P(t-t_P)$(もしこのプロセス呼出しがプロセス Q の定義式の右辺に表れているときは $P(t+e_Q-t_P)$) を呼び出すような各ノードの動作仕様を導出する.

例 9

全体仕様 $P := a^1[2 \leq t \leq 4 \wedge x = t]; b^2[t \leq x+5]; P[]c^1[5 \leq t]; \mathbf{exit}$

ノード間の遅延 $d_{12\max} = 4$, $d_{21\max} = 3$

のとき,

ノード1の動作仕様

$$\begin{aligned} Node_1 \;=\; & P(e_P) := a[2 \leq t+e_P \leq 4 \wedge x = t+e_P \wedge \\ & \exists t'(t' \geq t+e_P+d_{12\max} \wedge t' \leq x+5)]; \\ & s_2(m1,x); r_2(m3,t_P); P(t+e_P-t_P) \\ & >> s_2(m4); \mathbf{exit} \\ & []c[5 \leq t+e_P]; \mathbf{exit} \end{aligned}$$

ノード2の動作仕様

$$\begin{aligned} Node_2 \;=\; & P(e_P) := r_1(m1,x); b[t+e_P \leq x+5]; \\ & s_1(m2,x); s_1(m3,t+e_P); P(0) \\ & []r_1(m4); \mathbf{exit} \end{aligned}$$

□

[2]実際には各ノードそれぞれプロセスを呼び出した時点で時刻が0にリセットされる.

[3]したがって，$Restr()$ の適用前にはプロセスのパラメータは導入されていないので $Restr()$ は適用可能である.

このようなプロセス呼び出しの模倣方法を可能にするためには，各プロセスが呼び出された後，責任ノードからのメッセージが最も遅れた場合でも先頭動作が生起可能であるように各プロセスの先頭動作の時間制約を変更する. したがって，$Restr(S)$ の形式的な定義は次のようになる.

定義 6 $S = P$ where $P := B$の時,$Restr(S)$ は以下のように帰納的に定義される.

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

ただし，$h(Restr(B))$ は $Restr(B)$ 中の各先頭動作 a_kの時間制約式 $Q_k(t,x_k)$ を $Q_k(t,x_k) \wedge Q'_k$で置き換えた式である. ここで，Q'_kは以下に定義されるような論理式である（以下では Pを呼び出す直前の動作を実行するノードを p とする）:

$$Q'_k \stackrel{\text{def}}{=} \exists t' \exists x_k[t' \geq 0 + d_{p,\mathrm{place}(a_k)\max} \wedge Q(t',x_k)]$$

□

Q'_kは，責任ノードがプロセス Pを呼び出し，メッセージを送信したときに,Pの先頭動作を実行するノードがメッセージを受信した後に動作を実行できる場合には真，さもなければ偽になる.

4 あとがき

本稿では時間制約付 LOTOS 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合成するアルゴリズムを提案した.

ランデヴーへの対応 (ランデヴーを考慮した場合の問題点については [7] 参照) を含め，適用できるクラスの拡張を行なうことが今後の課題である.

参考文献

[1] 東野, 北道, 谷口. 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, 9(6):31–39, 1992.

[2] T. Higashino, K. Okano, H. Imajo, and K. Taniguchi. 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.

[3] C. Kant, T. Higashino, and G. V. Bochmann. Deriving protocol specifications from service specifications written in LOTOS. In *Proc. of 12th Annual Int'l Phoenix Conf. on Computers and Communications*, pp. 310–318. IEEE, Mar. 1993.

[4] A. Khoumsi, G. V. Bochmann, and R. Dssouli. On specifying real-time discrete event systems: An application for designing real-time protocols. In *Proc. of 14th IFIP WG6.1 Int'l Symp. on Protocol Specification, Testing and Verification*. IFIP, 1994.

[5] R. Langerak. Decomposition of functionality; a correctness-preserving LOTOS transformation. In *Protocol Specification, Testing and Verification, X*, pp. 229–242. IFIP, North-Holland, 1990.

[6] A. Nakata, T. Higashino, and K. Taniguchi. LOTOS enhancement to specify time constraints among non-adjacent actions using first order logic. In *Proc. of 6th IFIP Int'l Conf. on Formal Description Techniques*. IFIP, Oct. 1993.

[7] 中田, 東野, 谷口. 時間制約付 LOTOS 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の合成. I.C.S. Research Report 95-ICS-2, 大阪大学基礎工学部情報工学科, 1995.

[8] D. Park. Concurrency and automata on infinite sequences. In *Proc. of 5th GI Conference*, Vol. 104 of *Lecture Notes in Computer Science*, pp. 167–183. Springer-Verlag, 1981.

[9] R. L. Probert and K. Saleh. Synthesis of communication protocols: Survey and assessment. *IEEE Trans. Comput.*, 40(4):468–475, Apr. 1991.