拡張有限状態機械モデルを用いた分散処理システムの全体記述から各ノードの動作記述の自動生成

岡野 浩三　今城 広志
東野 輝夫　谷口 健一

大阪大学 基礎工学部 情報工学科, 豊中市

あらまし 分散システム全体の要求仕様を形式的な仕様として記述し, その記述から要求仕様通りに動作する各ノードの動作仕様(どのようなタイミングでどのノードとどのようなレジスタ値や同期用メッセージを交換しながら自ノードの動作を実行したり, 自ノードのレジスタ値を更新していくべきかを記述したもの) を自動生成出来ることが望ましい. そこで, 本稿ではレジスタを持つ拡張有限状態機械 (EFSM) としてモデル化された分散システム全体の要求仕様から, 要求仕様通りに動作する各ノードの動作仕様 (EFSM) を自動生成するためのアルゴリズムを提案する. このモデルでは, 非決定性の動作が記述できる. また, 入力と状態だけでなくその時点のレジスタ値に依存して次の状態や次のレジスタ値を定めることができる. 各レジスタは分散システムのリソースを表すと考える. 提案する方法では, リソースの分散配置や同一リソースの複数分散配置等が行え, 設計者が指定したリソースの配置に従った各ノードの動作仕様が導出できる. なお, 作成した自動生成アルゴリズムでは, もとの一つの状態遷移を, 選択動作の通知, レジスタ値の転送, 更新完了の3フェーズで実現し, 0-1 線形計画問題の解法を用いて, その際のメッセージ交換の総数ができるだけ少なくなるように各ノードの動作仕様を導出している.

Synthesis of Protocol Entities' Specifications from Requirement Specification for Distributed System in EFSM Model

Kozo OKANO, Hiroshi IMAJO,
Teruo HIGASHINO and Kenichi TANIGUCHI

Faculty of Engineering Science, Osaka University,
Toyonaka-shi, Osaka, 560 Japan

Abstract In a distributed system, the protocol entities must exchange some data values and synchronization messages in order to ensure the temporal ordering of the actions described in a requirement specification for the distributed system. It is desirable that the protocol entities' specifications can be derived automatically from the requirement specification for the distributed system. In this paper, we propose an algorithm which synthesizes automatically the protocol entities' specifications from a requirement specification described as an EFSM model. The next state and values of the registers are determined depending on not only the current state and input, but also the current values of the registers. In this model, we can describe nondeterministic behaviors. Each resource is treated as a register. The registers are allocated to the protocol entities. According to the allocation of the registers, each protocol entity's specification is derived. The allocation is specified by the designer. An algorithm solving 0-1 integer linear programming problems is used to reduce the number of the messages between the protocol entities.

1. まえがき

近年, 分散システム全体の要求仕様を形式的な仕様記述言語で記述し, 要求仕様通りに動作する各ノードの動作仕様 (どのようなタイミングでどのノードとどのようなレジスタ値や同期用メッセージを交換しながら自ノードの動作を実行したり, 自ノードのレジスタ値を更新していくべきかの記述) を自動生成するための方法についての研究が進められている[1]. 従来の自動生成の方法は大きく2つに分けられる. 文献(2), (3), (4), (5) では, 要求仕様を形式仕様記述言語 LOTOS[6]で記述し, その記述から各ノードの動作仕様を自動生成している. また, 文献 (7), (8) では有限状態機械としてモデル化された要求仕様から各ノードの動作仕様を導出するための方法が提案されている. これらの研究はいずれもデータを含まない要求仕様から各ノードの動作仕様を導出する方法である. しかし, 実際の分散システムではレジスタ値などのデータが取り扱われるため, 同期メッセージのみならず, データの交換についても自動生成できるようにアルゴリズムを拡張する必要がある.

そこで, 本稿では有限個のレジスタを持つ拡張有限状態機械 (Extended Finite State Machine (EFSM)) で記述された分散システム全体の要求仕様から, 要求仕様通りに動作する各ノードの動作仕様 (EFSM) を自動生成するためのアルゴリズムを提案する.

提案するモデルでは, 非決定性の動作が記述できる. また, 各リソースをレジスタとして表し, 入力と有限制御部の状態だけでなくその時点のレジスタ値に依存して次の状態やレジスタ値を定めることができる. レジスタのデータ型や次のレジスタ値を定める関数は設計者が自由に記述できる. 一般に OSI プロトコルなど多くの分散システムの仕様をこのクラスで記述できる. 一般に分散システムでは, リソースの分散配置はもちろん, 協調作業の効率化やフォールトトレーランスのための同一リソースの複数分散配置等が行なわれる. このモデルではこれらについて, 設計者が指定したリソースの複数分散配置に従って各ノードの動作仕様の導出が行える. 作成した自動生成アルゴリズムでは, 要求仕様の各状態遷移をノード間の同期メッセージの送受信動作の系列に変換する. このとき, 0-1 整数線形計画問題の解法を用いてそれらの送受信動作の総数ができるだけ少なくなるように各ノードの動作仕様を導出している. このように提案する手法を用いれば, 設計者は分散システム全体の要求仕様のみを記述すればよく, ノード間でのデータ交換や同期制御などを効率よく行う各ノードの動作仕様が自動生成され, 分散システムの設計・実現が容易になる.

以下 **2.** で EFSM モデルを形式的に定義し, **3.** でその動作を定義する. **4.** で各ノードの動作仕様を自動生成するためのアルゴリズムやその正当性などについて述べる.

2. EFSM モデル

本稿では分散システム全体の要求仕様を次のような EFSM としてモデル化する. また要求仕様のレベルでは各ノード間でのデータや同期メッセージのやりとりは一切記述しない.

2.1 EFSM 仕様とその意味定義

有限個のレジスタを持つ EFSM (Extended Finite State Machine) を 5 字組 $M = (S, R, A, \delta, \text{init})$ で定義する.

S: 有限制御部の状態の有限集合 $\{s_0, s_1, \ldots, s_n\}$

R: 有限個のレジスタの集合 $\{r_1, r_2, \ldots, r_m\}$

A: 動作の集合

動作は次の 3 種類

$a?x$:ゲート a からデータを入力. 入力データは変数 x に代入される (複数データの場合 x はベクトル).

$a!E(r_{i_1}, r_{i_2}, \ldots, r_{i_n})$:データ $E(r_{i_1}, r_{i_2}, \ldots, r_{i_n})$ の値をゲート a に出力. $E(r_{i_1}, r_{i_2}, \ldots, r_{i_n})$ は $r_{i_1}, r_{i_2}, \ldots, r_{i_n} \in R$の値を引数とする関数.

a: 入出力を伴わない動作 (便宜上出力動作 "$a!\emptyset$" と考える). 各動作 "$a?x$", "$a!E(r_{i_1}, r_{i_2}, \ldots, r_{i_n})$", "$a$" の "$a$" はゲート名を表すこととする.

δ: 以下のような入出力動作に関する遷移規則の集合. 各 $C(\ldots)$ を遷移条件, 各 $[r_{v'} \leftarrow \cdots]$ をレジスタ更新式と呼ぶ.

$s_i - \langle C(x, r_{k_1}, \ldots, r_{k_n}), a?x, R\rangle \rightarrow s_j$ where
$R = [r_{v'} \leftarrow E_v(x, r_{i_1}, \ldots, r_{i_l}), \ldots, r'_w \leftarrow E_w(x, r_{j_1}, \ldots, r_{j_p})]$
$s_i - \langle C(r_{k_1}, \ldots, r_{k_{n'}}), a!E(r_{h_1}, \ldots, r_{h_{q'}}), R'\rangle \rightarrow s_j$ where
$R' = [r_{v'} \leftarrow E_{v'}(r_{i_1}, \ldots, r_{i_{l'}}), \ldots, r_{w'} \leftarrow E_{w'}(r_{j_1}, \ldots, r_{j_{p'}})]$

init: 初期状態と初期レジスタ値の指定 $\langle s_0, r_1\text{init}, \ldots, r_m\text{init}\rangle$ のように $m+1$ 字組で記述.

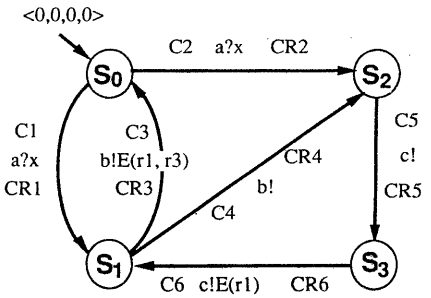
初期状態で各レジスタが, 指定された初期レジスタ値を持っているとする. この初期状態から始めて以下のように動作を行なう.

状態 s_iにおいて, 入力 x 及び現在のレジスタ値 $r_{k_1}, \ldots, r_{k_n}$がどの遷移条件 Cを満足するかによって次の実行可能な動作が決まる. 一つの動作 $a?x(a!E(\ldots))$ の実行後, 状態 s_jに遷移し, レジスタ更新式にしたがって $t(\leq m)$ 個のレジスタの値をそれぞれ $E_{v_1}(\ldots), \ldots, E_{v_t}(\ldots)$ に更新する (出力の場合も同様). 入力動作の場合, 次のレジスタ値は現在のレジスタ値と入力データの両方に依存して決まる. 出力動作の場合, 次のレジスタ値は現在のレジスタ値のみに依存して決まる. 各状態 s_iで複数個の実行可能な動作があってもよい. その場合, どの動作が選ばれるかは非決定性とする. なおこのモデルでは, 入力ゲートのデータを EFSM から観測することが可能で, その内容に依存して実際に入力動作を行なうかどうかを決定する. このようなモデルは LOTOS[6]などでも用いられている.

2.2 EFSM の例

EFSM の例をあげる (図 1). ここでは EFSM の仕様をグラフ表現している. 各ノードと枝はそれぞれ状態と状態遷移を表す. 状態遷移を表す枝には, 遷移条件, 動作, レジスタ更新式の 3 字組がラベルとして付けられている.

図 1の例では, 状態が $S_0, \ldots, S_3$の 4 つ, 状態遷移は, 6 つである. 使用されているレジスタは, $r_1, \ldots, r_4$の 4 つ. ま



$C1 \equiv r_1 = 0 \quad C2 \equiv r_1 \neq r_2 \quad C3 \equiv r_1 > r_3$
$C4 \equiv r_3 \geq 0 \quad C5 \equiv r_2 < r_4 \quad C6 \equiv \min(r_3, r_4) > 1$
$CR1 \equiv [r_3' \leftarrow x - r_1 + r_3, r_4' \leftarrow x - r_1]$
$CR2 \equiv [r_1' \leftarrow r_1 + r_2, r_2' \leftarrow x + r_3]$
$CR3 \equiv [r_1' \leftarrow r_4, r_2' \leftarrow r_3 + 1, r_3' \leftarrow r_2 - 1, r_4' \leftarrow r_1/2]$
$CR4 \equiv [r_3' \leftarrow -r_1]$
$CR5 \equiv [r_1' \leftarrow r_3, r_3' \leftarrow r_1 \times 2]$
$CR6 \equiv [r_4' \leftarrow r_3 + r_4]$

図 1: EFSM の仕様の記述例

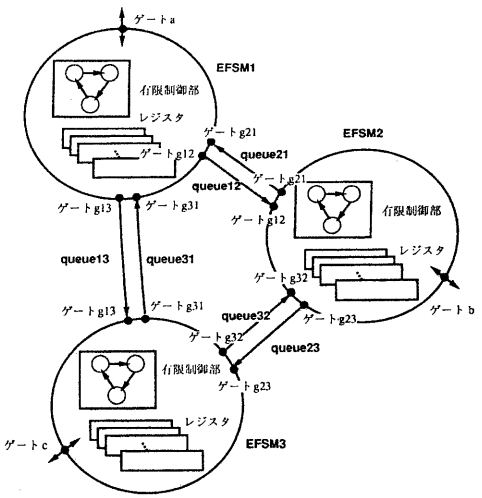


図 2: 分散 EFSM モデル

た, 使用されているゲートは, a, b, c の 3 つである. 状態 S_0 が初期状態であり, 各レジスタは 0 に初期設定されている. 状態 S_0 から状態 S_1 への遷移は, 遷移条件 $r_1 = 0$ が真のときのみ実行され, ゲート a から x の値を入力した後, レジスタ r_3 の値を $x - r_1 + r_3$ に, レジスタ r_4 の値を $x - r_1$ に更新し, 他のレジスタは変更しない.

3. 分散環境での EFSM の実現

与えられた EFSM を p 個のノードからなる分散システム上で協調して動作するシステムとして実現する (図 2).

各ノードの動作仕様も上述の EFSM としてモデル化する. 各ノード k の動作仕様を EFSM_k で表す. p 個のノードの動作仕様の組を $\langle \text{EFSM}_1, \ldots, \text{EFSM}_p \rangle$ で表し (p ノードの) 分散システムの動作仕様と呼ぶ.

m 個のレジスタ及び入出力ゲートがそれぞれどのノードに属するかをユーザが指定する (以下それらの割当を Alloc(EFSM) と記述する). 各入出力ゲートは必ず一つのノードに属すると仮定するが, 同一レジスタが複数ノードに属してもよいとする.

EFSM_i から EFSM_j への通信路は無限の容量を持つ FIFO キュー (queue_{ij}) で結ばれているとし, 両端のゲート名を g_{ij} とする. よって, EFSM_i が "g_{ij}!data" を実行すると queue_{ij} に data の値が入る (enqueue). また queue_{ij} に要素があるときに EFSM_j が "g_{ij}?x" を実行すると, queue_{ij} の先頭にある要素が変数 "x" に代入され, その要素が queue_{ij} から削除 (dequeue) される.

3.1 分散 EFSM モデルの等価性

分散システム全体の要求仕様 EFSM と, 分散システムの動作仕様 $\langle \text{EFSM}_1, \ldots, \text{EFSM}_p \rangle$ が等価であることを以下の様に定義する.

> (EFSM_i, EFSM_j 間の通信に用いられる) 送受信動作を, 観測不可能な動作としたとき, 観測可能な動作の系列について, EFSM と $\langle \text{EFSM}_1, \ldots, \text{EFSM}_p \rangle$ が互いに同じ動作系列を実行でき, かつ, 観測可能な任意の動作系列を行なった時点の, 実行可能動作が互いに等しいこと. これはプロセス代数における弱双模倣性等価[9],[10] に相当する.

3.2 分散 EFSM モデルの導出問題

[導出問題]

要求仕様 EFSM と分散システムの p 個のノード及び, 各ゲートやレジスタの割当 Alloc(EFSM) が与えられたとき, EFSM と等価であるような p ノードの分散システムの動作仕様 $\langle \text{EFSM}_1, \ldots, \text{EFSM}_p \rangle$ を導出する問題.

但し, 要求仕様として与えられる EFSM 及び, Alloc(EFSM) は次のような 2 つの制約条件を満足するものとする.

(1). 与えられた EFSM の各状態で 2 つの動作 "a..." と "b..." (a,b はゲート名) が記述されていたら, ゲート "a" と "b" の属するノードは同一でなければならない.

(2). 初期状態で各動作 "a..." が実行可能であるかどうかを判定する述語 $C(\ldots)$ は, ゲート "a" が属するノードに属するレジスタ $r_{v_1}, \ldots, r_{v_t}$ のみを用いた述語で記述されていなければならない. また, 初期状態からの出力動作で必要なレジスタもゲート "a" が属するノードに属していなければならない. □

制約条件 (1)を満足しないと, ある状態で実行可能な動作を実行するノードが複数個存在することになり, その場

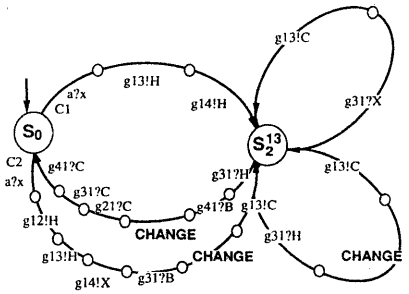


図 3: $EFSM_1$

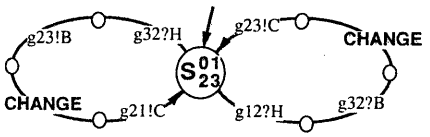


図 4: $EFSM_2$

合，それらの中の一つの動作を実行するには，ノード間で特別な同期をとる必要がある．この同期はノード間で一つのトークンを回して，そのトークンを獲得したノードが動作を実行できるようにすれば解決するが，以下での議論を簡単にするため，ここでは上記の制約条件 (1)を設ける．

制約条件 (2)は EFSM が最初に動作を始めるときにその動作の実行可能性判定，及びゲートへの出力を自ノードで行なえることを表しており，各ノードの EFSM の導出を簡単にするために与えた制約であり，本質的な制約ではない．制約条件 (2)を満足しない場合，特別な状態を導入しその状態を初期状態とみなし，ダミーの動作 (その動作の実行は何時でも可能で $C(\ldots)$ =true とする) でもとの初期状態に遷移するように EFSM を変形すれば，制約条件 (2)を満足する．

3.3 分散環境で動作する EFSM の例

図 1の EFSM の例で，ノードを 1,2,3,4 とし，各ゲートとレジスタの割当 Alloc(EFSM) を次のように決める．

	ノード 1	ノード 2	ノード 3	ノード 4
レジスタ	r_1, r_2	r_2	r_1, r_3	r_4
ゲート	a		b, c	

このとき，図 3~6で表す 4 字組 ($EFSM_1, \ldots, EFSM_4$) はもとの EFSM と等価である．

ここで，$EFSM_k$の各遷移に付けられた $g34!C, g13?A$ 等は，それぞれ，ゲート $g34$ にメッセージ Cを送信，あるいは，ゲート $g13$ から，メッセージ A を受信することを表している．これらの送受信の遷移の間は，個々のノードでレジスタの変更は行なわない．またそれらの遷移条件はつねに真である．また，例えば図 3における S_0からの最初の遷移は要求仕様と対応する遷移条件及び動作を表している．但しレジスタの変更は行なわない．レジスタの変更は **CHANGE**

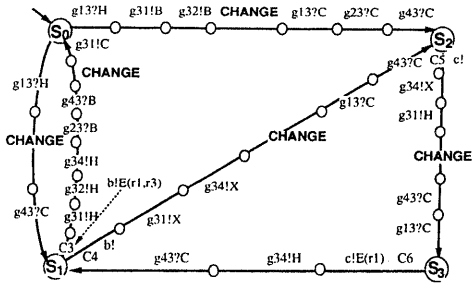


図 5: $EFSM_3$

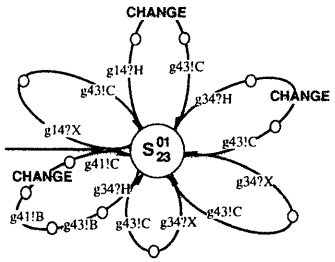


図 6: $EFSM_4$

と書かれた遷移でまとめて行なう．ここでの動作は観測されない内部動作とする．メッセージの内容及び，送受信の方法については後述する．

3.4 表記法

Snode　上の制約条件 (1)より，各状態 s_iで実行可能な動作は 1 つのノードで実行される．このノードを Snode(s_i) で表す．

Rnode　各レジスタ r_hの属するノードの集合を Rnode(r_h) で表す (r_hの属するノードが 1 つのときは Rnode(r_h) の要素は一個であるが，r_hの属するノードが複数の場合は Rnode(r_h) の要素数は複数になる．このため，Rnode(r_h) はノードの集合としている)．

Cset　状態 s_iに遷移したときノード Snode(s_i) が知っている (べき) レジスタ名の集合，すなわち，状態 s_i から始まる各遷移について，(イ) 動作の実行可能性を判定するために必要なレジスタ名，(ロ) 出力で用いられるレジスタ名をそれぞれ求め，各遷移の (イ), (ロ) に対する集合和，及び Snode(s_i) に含まれるレジスタ名の集合の和を Cset(s_i) で表す．

例えば，前節で述べた割当 Alloc(EFSM) に対して

Snode　Snode(S_0) ≡ ノード 1, Snode(S_1) ≡ ノード 3, Snode(S_2) ≡ ノード 3, Snode(S_3) ≡ ノード 3

Rnode　Rnode(r_1) ≡ { ノード 1, ノード 3}, Rnode(r_2) ≡ { ノード 1, ノード 2}, Rnode(r_3) ≡ { ノード 3}, Rnode(r_4) ≡ { ノード 4}

Cset　Cset(S_0) ≡ $\{r_1, r_2\}$, Cset(S_1) ≡ $\{r_1, r_3\}$, Cset(S_2) ≡ $\{r_1, r_2, r_3, r_4\}$, Cset(S_3) ≡ $\{r_1, r_3, r_4\}$

4. 各ノードの動作仕様の導出の概略

4.1 基本方針

まず, 与えられたEFSMから各ノードのEFSM$_k$を導出する方法の方針について述べる. 基本的にEFSM$_k$はもとのEFSMと同じような形の状態遷移図を持ち, もとの一つの状態遷移を通信のための幾つかの状態遷移系列に置き換える(挿入する)ことにより構成する. いま $S_i \rightarrow S_j$ をEFSMの一つの状態遷移とし, その動作を A とする. A を実行した後, レジスタ更新式に従ってレジスタ値を更新するためには, ノード間でレジスタ値の交換が必要である. ここでは, これらのレジスタ値を受けとったノードが新しいレジスタ値の計算を行なうことにする. そこで, この状態遷移に対して, p 個のノードは次のように動作するとする: (1) ノード Snode(S_i) が動作 A を実行し, (2) ノード Snode(S_i) から, レジスタ値を送信する必要のあるノードにメッセージを送り, (3) それらのメッセージを受け取ったノードからレジスタ値を必要とするノードにレジスタ値を転送し, (4) そのレジスタ値をもとに各ノードがレジスタ値を更新し, (5) 更新したノードからレジスタ値を更新し終ったことをSnode(S_j) へ知らせると共にSnode(S_j) の遷移条件の判定に必要なレジスタ値を保持しているノードからSnode(S_j) に更新後のレジスタ値を知らせる. もし, ノード kが上の(1) から (5) のどの動作にも関与しなければ, $S_i \rightarrow S_j$はϵ遷移に置き換える. この方針に従ってEFSM中の各 $S_i \rightarrow S_j$ を上の (1) から (5) に対応する動作系列に置き換えることによりEFSM$_k$が得られる.

ただし, (2) で送られるメッセージは送信依頼のためだけでなく, Cset(S_i) に属しているレジスタ値や入力 "x" の値を必要なノードに送信するためにも用いられる. さらに, 自ノードのレジスタ値のみでレジスタの更新が行えるノードに更新のきっかけを与えたり, (5) でSnode(S_j) にレジスタ値を知らせる必要があるノードに対する送信依頼も兼ねる.

例えば, 図1の状態遷移 $S_0 \rightarrow S_2$ に対して, 図3から図6の4つのEFSM$_k$は次のように動作する. まず, (1) ノード1が $a?x$ を実行した後, (2) ノード1からノード3へレジスタ r_3の値の送信依頼メッセージを送る (ノード 1, 2 がレジスタ r_2の内容を更新するためにはノード3からレジスタ r_3の値を送ってもらう必要がある) と共に, ノード2に入力 x の値を送る. また, ノード4にレジスタ r_4の値をノード3へ送ることを依頼する. (3) ノード3からレジスタ r_3の値をノード 1, 2 へ送る. (4) ノード 1, 2, 3 がレジスタの値を更新する. (5) ノード 1, 2 からノード3へレジスタ値の更新が終了したことを知らせると共に, ノード4からノード3へレジスタ r_4の値を転送する (ノード3が状態 S_2での遷移条件を判定するのに必要). この方法で $a?x$ の実行と各ノードのレジスタ値の更新が行える.

上記の方法では, (2), (3), (5) の3ステップでメッセージ交換が行われる. 勿論, 4ステップ以上のメッセージ交換による方法や, 適当なノードが更新後のレジスタ値を計算してその結果を必要なノードに配布する方法など, 上記以外の導出方法も考えられるが, 上の方法がメッセージ交換によるシステム全体での遅延時間を比較的短くできると考えられるので, ここでは上述の方針に従う.

上記の方法において, (2), (3), (5) の3ステップでのメッセージ交換の総数ができるだけ少なくなる方法を考える. 一般に, ノード kからノード k'へ送信依頼メッセージを送る場合と複数個のレジスタ値の組を送る場合ではメッセージ長に大きな違いがあるが, ここでは, その場合でも各メッセージを1個と考える. その仮定の下でメッセージ交換の総数を小さくする方法を考える (EFSM全体での最小化については後述). 例えば, あるノードがCset(S_i) に含まれるレジスタ r_hの値を必要とする場合, (2) でノード Snode(S_i) から送られるメッセージとしてその値を受け取ってもよいし, (3) で Rnode(r_h) から送られるメッセージとして受け取ってもよい. 全体のメッセージを少なくするにはどちらのステップで受信すべきかを決定する必要がある.

4.2 各 EFSM$_k$の動作

以下, 上述の (2), (3), (5) のステップで送受信されるメッセージをそれぞれμ型メッセージ, β型メッセージ, ξ型メッセージと呼び, μ型メッセージ, β型メッセージを受信するノードの集合をそれぞれμ, βで表し, ξ型メッセージを送信するノードの集合をξで表す.

集合μには, 集合βへβ型メッセージを送るノードの集合α, Snode(s_i) の知っているレジスタ値 (入力変数) を必要とするノードの集合η, 自ノードだけで更新ができるノードの集合λ, Snode(S_j) へ知っているレジスタ値を送信する必要があるが, (この遷移で) 更新は行なわないノードの集合χがある. また集合ξには, この遷移でレジスタ値の更新を行なうノードの集合γと, Snode(S_j) が必要とする (更新後の) レジスタ値を送るノードの集合ρがある.

このとき, 次のようなEFSMの各状態遷移に対して,

$$s_i - \langle C(x, r_{i_1}, \ldots, r_{i_n}), a?x, R\rangle \rightarrow s_j$$
$$\text{where} R = [r_v' \leftarrow E_v(\ldots),\ \ldots\ , r_w' \leftarrow E_w(\ldots)]$$

各EFSM$_1$, ..., EFSM$_p$は次のように動作すると考える.

手続き **Synthesis**

(Proc1). 状態 S_iで各動作が実行可能かどうかを判定する. 実行可能な動作があれば実行する.

(Proc2). Snode(S_i) から$\alpha+\eta+\lambda+\chi$に属するノードにμ型のメッセージを送る.

(Proc3). $\alpha+\eta+\lambda+\chi$に属するノードがSnode(S_i) からのμ型のメッセージを受け取る

(Proc4). αに属するノードからβに属するノードにレジスタ値 (β型のメッセージ) を送る.

(Proc5). βに属するノードがレジスタ値を受け取る.

(Proc6). γに属するノードがレジスタ値を更新する.

(Proc7). $\gamma-\rho$に属するノードからSnode(S_j)にレジスタ値の更新終了を知らせるξ型のメッセージを送る．またρに属するノードからSnode(S_j)に(更新後の)レジスタ値を送る．

(Proc8). Snode(S_j)がξからのメッセージを受け取る．

$EFSM_1,\ldots,EFSM_p$は最初，初期状態に対して上の(Proc1)～(Proc8)を実行する．次に(Proc8)で遷移した状態S_jに対して上の(Proc1)～(Proc8)を実行する．この操作を繰り返す．なお，β, η, ρについては，その集合の内容を定めるだけではなく具体的にどのレジスタ値をどのノードに送信するかを定める必要がある．

4.3 メッセージ数の削減法

4.3.1 各メッセージの決定に用いる述語

(I). Snode(S_i)をuとする．上の(Proc2)でノードuからノードuが知っているレジスタ値や入力xの値をノードvへ送らなければならないとき真となるような述語をηuvとする．また，ノードuからノードvへレジスタr_h(入力x)の値を送信する必要があるとき真となる述語をηuv_r_h (ηuv_x)とする．

(II). 同様に上の(Proc2)でノードuからノードvへレジスタ値の送信依頼メッセージを送る必要があるとき真となるような述語をαuvとする．

(III). 上の(Proc2)でλに属するノードvは自ノードのレジスタ値のみを用いてレジスタ値を更新できるノードである．このようなノードvにはノードuからレジスタ値の更新を指示するメッセージを送る必要がある．ノードuからノードvへこのメッセージを送る必要があるとき真となる述語をλuvとする．

(IV). 上の(Proc7)で$\rho-\gamma$に属するノードはレジスタ値の更新は行なわないが，Snode(S_j)にレジスタ値を知らせる必要のあるノード($\in\chi$)である．χに属するノードには，レジスタ値の送信を依頼するメッセージを送る必要がある．(Proc2)でこのメッセージを送る必要があるとき真となる述語をχuvとする．

(V). 上の(Proc2)で$\eta uv, \alpha uv, \lambda uv, \chi uv$の何れかが真であるとき真となる述語を$\mu uv$とする．

(VI). 上の(Proc4)でαに属するノードwからβに属するノードvにレジスタ値を送る必要があるとき真となる述語をβwvとする．また，ノードwからノードvへレジスタr_hの値を送信する必要があるとき真となる述語をβwv_r_hとする．

(VII). 上の(Proc8)でξに属するノードvからノードSnode(S_j)(以下ノードzとする)にレジスタ値の更新終了を知らせるメッセージ(またはレジスタ値)を送る必要があるとき真となる述語をそれぞれ，γvz, ρvzとする．このうち，レジスタr_hの値を送る必要があるとき真となる述語をρvz_r_hとする．また，$\gamma vz, \rho vz$の少なくとも一つが真のとき真となる述語をξvzとする．

4.3.2 各述語の決定に用いる制約条件

上の(I)～(VII)の記述から，ηuv, ηuv_r_h, ηuv_x, αuv, λuv, χuv, μuv, βwv, βwv_r_h, ξvz, γvz, ρvz, ρvz_r_hの間に次の不等式が成り立つ必要がある(ここでは各述語を0，1のいずれかの値を取る変数とみなす)．

uをSnode(S_i)とする．$r_h\in$Cset(S_i)なる各レジスタr_hとノード$v(1\le v\le p)$の組に対して，

(1) $$\eta uv \ge \eta uv\_r_h$$

uをSnode(S_i)とする．入力xとノード$v(1\le v\le p)$の組に対して，

(2) $$\eta uv \ge \eta uv\_x$$

ノードvが入力xを必要とするとき(ノードvのレジスタ値の更新に入力xの値が必要なとき)，

(3) $$\eta uv\_x = 1 \quad \text{上述の}(I)\text{に対応}$$

uをSnode(S_i)とする．各ノード$v(1\le v\le p)$と$w(1\le w\le p)$の組に対して，

(4) $$\alpha uv \ge \beta vw \quad \text{上述の}(II)\text{に対応}$$

uをSnode(S_i)とする．各ノード$v\in\gamma$に対して，

(5) $$\eta uv + \sum_{1\le y\le p\wedge y\ne u} \beta yv + \lambda uv \ge 1 \quad \text{上述の}(III)\text{に対応}$$

uをSnode(S_i)とする．各ノード$v(1\le v\le p)$に対して，

(6) $$\mu uv \ge \alpha uv, \mu uv \ge \eta uv,$$

(7) $$\mu uv \ge \lambda uv, \mu uv \ge \chi uv \text{上述の}(V)\text{に対応}$$

uをSnode(S_i)とする．各ノード$v(1\le v\le p)$とレジスタ$r_h(1\le h\le m)$の組に対して，もしノードvがr_hの値を必要とするなら，

(8) $$\sum_{w\in \mathrm{Rnode}(r_h)} \beta wv\_r_h + \eta uv\_r_h \ge 1$$

また，各ノード$v(1\le v\le p)$と$w(1\le w\le p)$の組に対して，

(9) $\beta wv \ge \beta wv\_r_1, \ldots, \beta wv \ge \beta wv\_r_m$ 上述の(VI)に対応

zをSnode(S_j)とする．各ノード$v\in\gamma$に対して，

(10) $$\gamma vz = 1$$

zをSnode(S_j)とする．各ノード$v\notin\gamma$に対して，

(11) $$\chi uv \ge \rho vz \quad \text{上述の}(IV)\text{に対応}$$

さらに，uをSnode(S_i)とする．各ノード$v(1\le v\le p)$に対して，

(12) $$\gamma uz \ge 1 - \sum_{(1\le v\le p)v\ne u} \mu uv$$

(これは，ノード間のメッセージのやりとりが全くなくて，しかも実行ノードが替わるときに，ノードuから直接ノードzへγ型のメッセージを送ることを表している．)

また，各レジスタ $r_h(1 \leq h \leq m)$ に対して，もしノード z が r_h の値を必要とする ($z \notin \mathrm{Rnode}(r_h)$) なら，

$$\sum_{w \in \mathrm{Rnode}(r_h)} \rho wz\_r_h \geq 1 \quad (13)$$

さらに，各ノード $w(1 \leq w \leq p)$ とレジスタ $r_h(1 \leq h \leq m)$ の組に対して，

$$\rho wz \geq \rho wz\_r_h \quad (14)$$

$\xi wz \geq \rho wz, \xi wz \geq \gamma wz$, 上述の (VII)に対応 (15)

4.3.3 各述語の決定の方法

上(1)~(15)の制約条件を満足するような変数の組の中で，

$$N = \left\{ \sum_{1 \leq y \leq p \wedge y \neq u} \mu uy \right\} + \left\{ \sum_{1 \leq w \leq p \wedge w \neq u} \sum_{1 \leq v \leq p} \beta wv \right\} + \left\{ \sum_{1 \leq y \leq p \wedge y \neq z} \xi yz \right\}$$

の値を最小にするように各変数の値を定めれば，メッセージの総数が小さくなる．制約条件は線形不等式であるので，Nを目的関数として，0-1 整数線形計画問題を解くアルゴリズムを用いて Nの値を最小にするような解を求めればよい．

例の EFSM に対して，実際に上述のアルゴリズムを適用したときの結果を紹介する．

S_0, S_1間の状態遷移

$\eta 13, \eta 14, \eta 13\_x, \eta 14\_x, \eta 14\_r_1, \mu 13, \mu 14, \xi 43, \gamma 43$ が 1 その他が 0 となる．

S_0, S_2間の状態遷移

$\eta 12, \eta 13, \eta 12\_x, \eta 13\_r_2, \alpha 13, \chi 14, \mu 12, \mu 13, \mu 14, \beta 31, \beta 32, \beta 31\_r_3, \beta 32\_r_3, \xi 13, \xi 23, \xi 43, \gamma 13, \gamma 23, \rho 13, \rho 43\ \rho 13\_r_2, \rho 43\_r_4$ が 1 その他が 0 となる．

S_3, S_1間の状態遷移

$\eta 34, \eta 34\_r_3, \mu 34, \xi 43, \gamma 43$ が 1 その他が 0 となる．

4.4 各 EFSM_kの構成法

手続き **Synthesis** にしたがって，各ノードの状態遷移系列が生成できる．

(Proc1)では，kが $\mathrm{Snode}(S_i)$ であるとき，要求仕様の，遷移条件，動作をそのまま記述する (レジスタ更新は行なわない．これは (Proc6)でまとめて行なう)．(Proc2)では，kが $\mathrm{Snode}(S_i)$ ならμ型のメッセージを複数出力する必要があるが，これらは適当な順序で直列に行なえばよい．遷移条件は常に真である．(Proc3)では，kが $\mathrm{Snode}(S_i)$ 以外のノードでμ型のメッセージを受信する必要のあるときは受信動作を行なう．遷移条件は次で述べるようにメッセージの内容に依存する．(Proc4)で，複数のβ型のメッセージの送信を行なう必要のあるときは (Proc2)と同様に適当な順序で直列に行なう．(Proc5)では，β型のメッセージを受信する必要のあるとき，それらのメッセージの受信動作を行なう．受信動作も適当な順序で直列に行なえばよい．(Proc6)でレジスタ更新を行なう．ここでまとめて更新を行なうために，(Proc3), (Proc5)で受信したレジスタ値を保持しておく一時レジスタを各ノードで一個用意しておく．この一時レジスタで複数個のβ型のメッセージを保持しておいて，この内容を用いて (Proc6)でレジスタ値を更新する．(Proc7)では，kがξに属していれば，ノード $\mathrm{Snode}(S_j)$ にξ型のメッセージを送信する．(Proc8)で kが $\mathrm{Snode}(S_j)$ なら，ξ型メッセージの受信動作を適当な順序で直列に行なう．

4.5 メッセージの内容と状態遷移図の簡約

状態遷移 $e(S_i \to S_j)$ について，ノード $u(=\mathrm{Snode}(S_i))$ からノード $v(1 \leq v \leq p)$ へ送信するμ型のメッセージの内容は次のとおり：$\langle \mathrm{label}(e), \{\langle \text{“}r_h\text{”}, \mathrm{value}(r_h)\rangle \mid \eta uv\_r_h = 1\}, \{\langle \text{“x”}, \mathrm{value}(x)\rangle \mid \eta uv\_x = 1\}\rangle$

但し，$\mathrm{label}(e)$ は辺 e を他辺と区別するためのラベル，$\mathrm{value}(r_h), \mathrm{value}(x)$ はそれぞれ，レジスタ r_h，入力 x の値．

同様に，ノード vからノード $w(1 \leq v, w \leq p)$ へ送信するβ型のメッセージの内容を $\langle \mathrm{label}(e), \{\langle \text{“}r_h\text{”}, \mathrm{value}(r_h)\rangle \mid \beta uv\_r_h = 1\}\rangle$ とし，ノード $v(1 \leq v \leq p)$ からノード $z(=\mathrm{Snode}(S_j))$ へ送信するξ型のメッセージの内容を $\langle \mathrm{label}(e), \{\langle \text{“}r_h\text{”}, \mathrm{value}(r_h)\rangle \mid \rho uv\_r_h = 1\}\rangle$ とする．すなわち，各メッセージには状態遷移 e のラベル名及び，送信すべきレジスタ名 (入力変数名) とその内容が含まれている．

実行ノードでない場合は，非決定遷移のどちらを実行すべきかをメッセージの状態遷移 e のラベル名を先読みして判断する．例えば，図 3の EFSM において，状態 S_{123}から，$g_{31}?M$, $g_{31}?M$, $g_{32}?H$, で始まる三つの状態遷移があるが，このうちどれを行なうべきかはゲート g_{31}で入力したデータに含まれるラベル名で判断する．

上述の手続きによって導出されるあるノードの状態遷移図は，いわゆるϵ遷移を含んでいたり，冗長な状態を含んでいることがある．例えば，ノード 2 において，手続き **Synthesis** のみを適用すると，S_0, S_1間，S_1, S_2間，S_2, S_3間，S_3, S_1間の遷移がϵ遷移となる．この遷移において，S_0, S_1間，S_1, S_2間，S_2, S_3間，S_3, S_1間のϵ を除去すると，前述の EFSM_2が得られる．同様に図 3, 6の EFSM_1, EFSM_4でもこのϵ遷移を除去し等価で簡潔な EFSM を構成している．

ϵ遷移や冗長な状態の簡約方法は以下の通り．

仕様記述の各状態遷移を手続き **Synthesis** で得られた状態遷移系列に置き換えた EFSM_iについて，状態遷移を辺，状態をノードと見なし，辺集合$\vec{E}$，ノード集合Vからなる有向グラフ $G_i(V, \vec{E})$ を考える．

要求仕様 EFSM での各状態に対応する各 EFSM_kでの状態は，手続き **Synthesis** によって生じる中間状態と区別できる．各 EFSM_kでの前者の状態の集合を Q_kで表す．

ノード集合Vのうち，Q_kに属する状態Sでかつ $\mathrm{Snode}(S)$ がノード kでないものの集合 NC_kを考える．NC_kの各要素の状態においては，ノード k以外のノードが実行ノードになっている．NC_kの各要素を始点，あるいは終点とするすべての辺について，その始点 (あるいは終点) を $S'(\notin V)$ に替えた辺の集合を$\vec{E}(NC_k)$ とする (始点，終点ともに NC_kの各要素の場合は $S' \to S'$なる辺と考える)．NC_kの各要素を始点，あるいは終点とするすべての辺の集合を$\vec{N}$とする．また辺集合$\vec{E}$のうち，ϵ遷移に対応する辺の集合を$\vec{e}$とする．

グラフ $G_k(V, E)$ からグラフ $G'_k((V - NC_k) \cup \{S'\}, (E -$

$\vec{N}-\vec{e})\cup\vec{E}(NC_k)\}$) を作る. グラフ G'_kは簡約された状態の EFSM_kを表す.

4.6 EFSM_kの構成法の正当性

まず, (1). 各状態遷移に対して, 手続き **Synthesis** のみで導出される分散システムの動作仕様と要求仕様 EFSM の等価性が成立つことを示す. 次に, (2). 簡約化アルゴリズムを適用した動作仕様について, 適用前の動作仕様と等価性がなりたつことを示す.

(1). 要求仕様 EFSM が初期状態にあるとき, 動作仕様の $\langle\mathrm{EFSM}_1,\ldots,\mathrm{EFSM}_p\rangle$ はそれぞれの初期状態にあり, すべてのキューは空である. また, この状態における実行ノードが一つあり, 遷移条件, 動作, レジスタ値ともに要求仕様 EFSM に等しいことは手続き **Synthesis** よりあきらか.

要求仕様 EFSM が状態 S_iにあるとき, $\langle\mathrm{EFSM}_1,\ldots,\mathrm{EFSM}_p\rangle$ がそれぞれ状態 S_iにあり, すべてのキューは空とする. ノード qが $\mathrm{Snode}(S_i)$ であるとする. EFSM_qの状態 S_iでは, 遷移条件, 動作ともに要求仕様 EFSM に等しい. また, q以外の EFSM_k では, 状態 S_iからの遷移は, 受信動作で始まるか, ϵ遷移であることは, 手続き **Synthesis** より簡単に分かる.

ノード qが状態 S_iからの遷移を一つ選んだとき, q以外のノードが, どの遷移を選べばよいかは, メッセージの構成法より, $\mathrm{label}(e)$ を調べればわかる. $\langle\mathrm{EFSM}_1,\ldots,\mathrm{EFSM}_p\rangle$ がこの遷移により, 正しくレジスタを更新することは, 手続き **Synthesis** より明らか.

遷移後, 要求仕様 EFSM の状態が S_jであれば, ノード q 及び, 何等かの受信動作が行なわれたノードの EFSM_kは状態 S_jにある. その他の $\mathrm{EFSM}_{k'}$では, 次に行なえる動作は結局, 受信動作しかないのでそのノードはどの状態にあるか分かっていると考えてよい.

またこの一連のメッセージの送受信は 1 対 1 対応しており, また $\mathrm{Snode}(S_j)$ は, 変更のあったすべてのノードからのメッセージを受け取った後でないと状態 S_jに遷移できないので, S_jに遷移した後はすべてのキューは空である.

(2). この (1) で得られた任意の EFSM_kにおいて, kが実行ノードである状態 Sと実行ノードでない状態 S'間にϵ遷移がないことが保証できるので, 実行ノードでない状態を前節のアルゴリズムで一つの状態にまとめても, もとの EFSM_kと同じ動作が行なえることを確認すれば十分である.

もとの EFSM_kにおける異なる 2 つの状態 S', S''がこのアルゴリズムにより, 一つの状態 Sになったとする. アルゴリズムより, ノード kは状態 S', S''の実行ノードではないので, S', S''からの遷移はすべて受信動作で始まる. したがって Sからの遷移もすべて受信動作で始まる.

$\mathrm{Snode}(S'')$ によって選ばれた遷移でノード kが受信動作を行なうのであれば, メッセージ内のラベル $\mathrm{label}(e)$ によって, 状態 Sからもともと状態 S''にあった対応する遷移を行ない, 異なる遷移をすることはない. 受信動作を行なわないときは, ノード kの動作はϵ遷移であり, ノード kは何もしなくてよい.

4.7 EFSM 全体でのメッセージ数の削減

複数の遷移を考えた場合, よりメッセージ数を減らすことが出来る. 例えば前述の例題の場合, **Synthesis** より, 状態 S_2に遷移した時点で EFSM_3はレジスタ r_4の値を知っている. また S_2, S_3の間の遷移で, レジスタ r_4の値は変化しないので, この遷移で EFSM_3はレジスタ r_4の値を再び問い合わせる必要はない. よって図 5の状態 S_2から S_3の遷移系列中の $g34!M, g43!C$を取り除くことができる (図 6の EFSM_4も同様). このように大局的に考えるとメッセージ数を減らすことができる. なお他にも幾つかのメッセージ削減法が考えられるが誌面の都合で割愛する.

5. 今後の検討事項

本稿では, EFSM モデルで記述された分散システムの要求記述から各ノードの動作仕様を自動生成するアルゴリズムを与えその正当性を示した.

我々の研究グループでは, EFSM 仕様を記述するため ASL/ASM という代数的言語を定めそのクラスに対するインタプリタ, コンパイラを開発している[11]. 現在 ASL/ASM で要求記述を行ない, その記述から各ノードの動作仕様 (ASL/ASM プログラム) を自動生成することを検討している. 実際の応用例に対し本アルゴリズムを適用し, その有効性を評価すること等が今後の課題である.

文献

(1). R. Probert and K. Saleh : "Synthesis of Communication Protocols: Survey and Assessment", IEEE Trans. on Comp., Vol.40, No.4, pp.468-476, 1991.

(2). R. Gotzhein and G. v. Bochmann : "Deriving Protocol Specifications from Service Specifications Including Parameters", ACM Trans. on Comp. Syst., Vol. 8, No. 4, pp.255-283, Nov. 1990.

(3). G. v. Bochmann and R. Gotzhein : "Deriving protocol specifications from service specifications", Proc. of ACM SIGCOMM '86, pp.148-156, 1986.

(4). R. Langerak : "Decomposition of functionality; a correctness-preserving LOTOS transformation", Proc. of Tenth IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification, North Holland, pp. 229-242, 1990.

(5). T. Higashino : "Service Specification and Its Protocol Specifications in LOTOS", IEICE Trans. Fundamentals, Vol. E75-A, No. 3, pp.330-338, March 1992.

(6). ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807, Jan. 1989.

(7). P.-Y.M. Chu and M.T. Liu : "Synthesizing Protocol Specifications from Service Specifications in FSM model", Proc. Computer Networking Symp. '88, pp.173-182, April 1988.

(8). P.-Y.M. Chu and M.T. Liu : "Protocl Synthesis in a State-Transition Model", Proc. COMPSAC'88, pp.505-512, 1988.

(9). D. Park: "Concurrency and automata on infinite sequences", Theoretic Computer Science, Lecture Notes in Computer Science, Vol. 104. pp.167-183, Springer-Verlay, 1981.

(10). R. Milner: "Comunication and Concurrency", Prentice-Hall, 1989.

(11). 大蔵 雅弘, 杉山 裕二, 谷口 健一:"代数的言語 ASL における抽象的順序機械型プログラムとその処理系", 信学論 (DI),**J73-D-I**, 12, pp.971-978(平 2-12).