拡張有限状態機械モデルを用いた分散システムの要求仕様から各ノードの動作仕様の自動導出

岡野浩三† 今城広志†
東野輝夫† 谷口健一†

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（どのようなタイミングでどのノードとどのような同期用メッセージやレジスタ値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）を自動生成できることが望ましい．そこで，本論文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．どのノードに各リソースを配置するかは設計者が決める．同一リソースを複数ノードに分散配置してもよい．作成した自動生成アルゴリズムでは，与えられた要求仕様とゲートや，リソースの配置情報から各ノードの動作仕様を自動生成する．導出は要求仕様の各状態遷移を，選択動作の通知，レジスタ値の転送，レジスタ値の更新，更新終了の通知などの一連の動作の系列で置き換えることにより実現し，0-1 線形計画問題の解法を用いて，その際のノード間のメッセージ交換の総数をできるだけ少なくしている．

Synthesis of Protocol Entities' Specifications from Service Specification of Distributed System in Extended Finite State Machine Model

Kozo Okano,† Hiroshi Imajo,† Teruo Higashino† and Kenichi Taniguchi†

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 extended FSM (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. はじめに

近年，計算機ネットワークの発展に伴い，信頼性の高い分散システムを設計するための方法についての研究が盛んに行われている．抽象レベルでは，分散システムの各ノードで実行される動作（処理）の実行順序の記述をそのシステムの要求仕様とみなすことができる[1]．与えられた要求仕様に対し，分散システム上の各ノードはお互いに同期用メッセージやデータを交換しながら要求仕様で指定された順に動作を実行する．各ノードがどのような順に同期用メッセージやデータを交換しながら動作を実行していくべきかを記述したものを各ノードの動作仕様と呼ぶ．一般に設計者自身が各ノードの動作仕様を陽に記述するのは非常に繁雑であり，間違いも起こりやすい．分散システムの設計法として，設計者は要求仕様のみを記述し，その記述から各ノードの動作仕様を自動生成できることが望ま

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

しい．

従来の自動生成の研究は大きく2つに分けられる．文献 2)，3)，4)，5)，7) 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 8)，9) 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，本論文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，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．どのノードに各リソースを分散配置するかは設計者が決める．協調作業の効率化やフォールトトレーランスのため，同一リソースを複数ノードに分散配置してもよい．提案する自動生成アルゴリズムでは，与えられた要求仕様と設計者が指定したリソースの複数分散配置から，各ノードの動作仕様の導出を行う．各ノードの動作仕様は，要求仕様の各状態遷移を一定の方針にしたがってノード間のメッセージの送受信動作などを行う幾つかの状態遷移の系列に変換することによって導出する．その際，各状態遷移におけるノード間の送受信動作の総数が最小になるように各ノードの動作仕様を導出しており，効率よく動作仕様を実行できる．最小解を求めるために 0-1 整数線形計画問題の解法を用いている．

以下2章で EFSM モデルを形式的に定義し，3章でその動作を定義する．4章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などについて述べる．

2. EFSM モデル

分散システム全体の要求仕様（サービス仕様[1]とも呼ばれる）ではノード間での同期メッセージやデータのやりとりは一切記述しない．本論文では，そのような要求仕様を次のような EFSM としてモデル化する．

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) 仕様を5字組 $M=(S, R, \mathcal{A}, \delta, \text{init})$ で定義する．

S は有限制御部の状態の有限集合 $\{s_1, \cdots, s_n\}$，R は有限個のレジスタの集合 $\{R_1, R_2, \cdots, R_m\}$ である．$\mathcal{A}$ は次の2種類の動作の集合である．$a?x$ はゲート a からのデータ入力を表す．変数 x は入力データを表す（複数データの場合 x はベクトル）．$a!E(\cdots)$ で式 $E(\cdots)$ の値をゲート a に出力することを表す．E は R の要素である（複数の）レジスタを引数として持つ関数である．$a!\emptyset$ (空値の出力動作) を便宜上単に a と表す．δ は状態遷移関数の集合である．各状態遷移関数は $s-\langle a?x, C, CR\rangle \rightarrow s'$ で表す．ここで $a?x$ ($a!E(\cdots)$ も可) は動作であり，C はレジスタ $R_{k1}, \cdots, R_{kt}$ と入力変数 x を引数とする述語であり，CR は次の状態 s' でのレジスタ値を定める代入文 $R_i \leftarrow f(R_{h1}, \cdots, R_{hs}, x)$ の組である．C を遷移条件と呼び，CR をレジスタ更新式と呼ぶ．init は初期状態 s_1 と各レジスタの初期レジスタ値を指定する．

初期状態と初期レジスタ値から以下のように動作を行う．いま現在の状態を s_i，レジスタ値を $v_{R_1}, \cdots, v_{R_n}$ とする．状態遷移関数 $s_i-\langle a?x, C, CR\rangle \rightarrow s_j$ において，もし現在のレジスタ値 $v_{R_1}, \cdots, v_{R_n}$ および，入力変数値 v_x が遷移条件 C を満たせば $a?x$ が実行可能となる．s_i からのすべての状態遷移関数について実行可能な動作を求め，実行する動作を一つ非決定的に選択する．$a?x$ 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各レジスタ値を更新する（出力動作の場合，出力後に更新を行う）．例えば，$R_1 \leftarrow R_1 + R_2 - x \in CR$ の場合，レジスタ R_1 の新しい値は，$v_{R_1} + v_{R_2} - v_x$ になる．このモデルでは，入力データの内容に依存して実際に入力動作を行うかどうかを決定することができる．

図1 に EFSM 仕様（要求仕様）の例をあげる．ここでは EFSM 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，動作，遷移条件，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．レジスタは，$R_1, \cdots, R_6$ の6つである．また，使用されているゲートは，a, b, c の3つである．

この例は，店のレジの業務をモデル化したものである．状態 S_1 が初期状態であり，各レジスタは [0,0, data, 0, 0, 0] に初期設定されている（data は商品データベースの内容を表す）．商品番号読みとり装置（ゲート a）から商品番号を読み，レジスタ R_1 に保存する（$S_1 \rightarrow S_2$）．状態 S_2 で商品番号（R_1 の値）が0でなければ，商品データベース R_3 から該当商品番号の価格 $R_3(R_1)$ を調べレジスタ R_6 に代入する（$S_2 \rightarrow S_3$）．この状態遷移では出力はないので動作は b（$b!\emptyset$ のこと）としている．表示器（ゲート b）に商品番号，価格（レジスタ R_1, R_6 の値）を表示し，総合計 R_2 を更新し，初期状態に戻る（$S_3 \rightarrow S_1$）．一方状態 S_2 でレジスタ R_1 の値が0であれば商品入力の終了と見なし，総合計 R_2 を表示器に表示し（$S_3 \rightarrow S_4$），客から受けとった現金を順次レジ（ゲート c）に入力していく（$S_4 \rightarrow S_4$）．金額の総合計はレジスタ R_4 に入る．金額の総合計 R_4 が商品価格の総合計 R_2 以上になれば，釣り銭をレジに表示し，店の売上の総計 R_5 を計算し各レジスタをリセットして初期状態に戻る（$S_4 \rightarrow S_1$）．

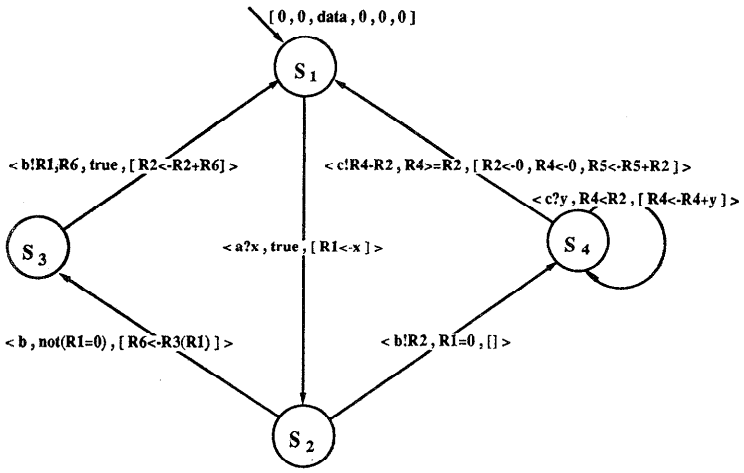


図 1 EFSM 仕様の例
Fig. 1 A service specification in EFSM model.

出力ゲート，各レジスタは必ずいずれかのノードに属するとする．同一ゲートが複数のノードに属することはないと仮定するが，同一レジスタが複数ノードに属してもよい（図2のレジスタ R_2）．これはリソースの複数分散配置に相当する．

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**）．

本論文では，各ノードの動作仕様も上述の EFSM 仕様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 $\langle \mathrm{EFSM}_1, \cdots, \mathrm{EFSM}_p \rangle$ で表し（p ノードの）分散システムの動作仕様と呼ぶ（プロトコル仕様とも呼ばれる[1]）．

m 個のレジスタおよ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し，それらの割当を Alloc (EFSM) と記述する．各入

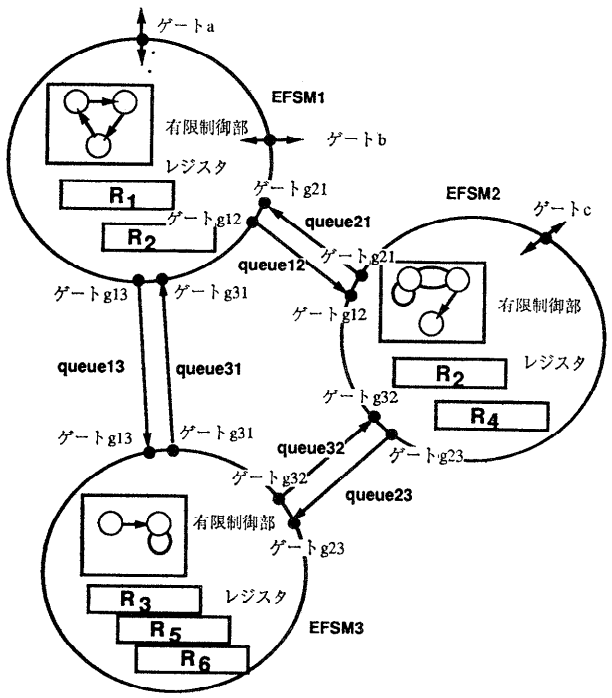


図 2 分散 EFSM モデル
Fig. 2 EFSM model in distributed system.

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．また queue$_{ij}$ に要素があるとき 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$_{ij}$ から削除される．

3.1 分散 EFSM モデルの等価性

分散システムの動作仕様 〈EFSM$_1$, …, EFSM$_p$〉 (以降，EFSM^{1-p} と略記) において，各 EFSM$_k$ ($1 \leq k \leq p$) が初期状態でかつ，通信路の各 FIFO キューがすべて空であるときを，EFSM^{1-p} の初期状態とする．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 EFSM^{1-p} が等価であることを以下のように定義する．

[等価性]

分散システムの動作仕様 EFSM^{1-p} において，各 EFSM$_i$, EFSM$_j$ 間の通信に用いられる送受信動作 $g_{ij}?x$, $g_{ij}!E(\cdots)$ を，観測不可能な動作とし，その他の動作を観測可能な動作とする．このとき EFSM と EFSM^{1-p} が弱双模倣性等価[10),11)]であれば，両者は等価であるという．2つの EFSM が弱双模倣性等価であるというのは，一方の EFSM で実行できる任意の観測可能な動作の系列が他方の EFSM でも実行でき，それが EFSM を入れ替えても成り立ち，かつ任意の実行可能な観測可能動作系列を行った時点での実行可能かつ観測可能な動作が互いに等しいことである．

3.2 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 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
レジスタ	R_1, R_2	R_2, R_4	R_3, R_5, R_6
ゲート	a, b	c	

このとき，**図3** で表す3字組 〈EFSM$_1$, EFSM$_2$, EFSM$_3$〉 は図1の EFSM と等価である．

例えば，図1の $S_1 \to S_2 \to S_3 \to S_1$ の状態遷移系列に相当する各 EFSM$_k$ の動作は次のとおりである．ただし，各 EFSM$_k$ には Alloc (EFSM) で割り当てられたレジスタに加え作業用のレジスタ R_0 があり，他ノードから送られたメッセージを一時的に格納しておくために利用する．

EFSM$_1$ は状態 S_1 から始め，ゲート a から x を読み込みレジスタ R_1 に代入し，S_2 に遷移する (put(R_0, m) は入力 m をレジスタ R_0 に格納する補助関数．get(R_0, R_h) はレジスタ R_0 からレジスタ R_h の最後に格納された値を取り出す補助関数．入力変数 x の値を取り出す関数 get(R_0, x) も同様に定義する)．ここで，動作 g_{11} は，レジスタ更新 ($R_1 \leftarrow$get(R_0, x)) のみを行うための非観測動作を表すために便宜上，このような記述を行っているものである．実際にゲート g_{11} があるわけではない (以下，同様)．この間 EFSM$_2$, EFSM$_3$ は状態 S_0 のまま．次に，EFSM$_1$ は状態 S_2 で $R_1 \neq 0$ の判定を行い，EFSM$_3$ に R_1 の値をメッセージ M_1 で送る．これを受けとった EFSM$_3$ は R_6 の値を計算し，その値をメッセージ M_2 として EFSM$_1$ に送り S_0 にもどる．EFSM$_1$ は M_2 を受けとった後，状態 S_3 に入り，ゲート b から，R_1 と今受けとった R_6 の値を出力し，EFSM$_2$ に R_6 の値をメッセージ M_4 として転送する．EFSM$_2$ は M_4 を受けとり，これを用いてレジスタ R_2 を更新し，更新終了メッセージ M_5 を EFSM$_1$ に送信し，S_0 にもどる．EFSM$_1$ は M_5 を受けとったのち S_0 にもどる．各メッセージの内容については 4.3 節で述べる．

3.3 分散 EFSM モデルの導出問題

[導出問題]

要求仕様 EFSM と分散システムのノードの集合 {ノード$_1$, …, ノード$_p$} および，ゲートやレジスタの割当 Alloc (EFSM) が与えられたとき，EFSM と等価な p ノードの分散システムの動作仕様 EFSM^{1-p} を導出する問題．

ただし，要求仕様として与えられる EFSM および，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 "$a\cdots$" と "$b\cdots$" (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. 初期状態を s_I とする．$s_I - \langle a\cdots, C(\cdots), CR\rangle \to s'$ なるすべての状態遷移に対し，遷移条件 $C(\cdots)$ は，ゲート "a" が属するノードに属するレジスタ $R_{v1}, \cdots, R_{vl}$ 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，

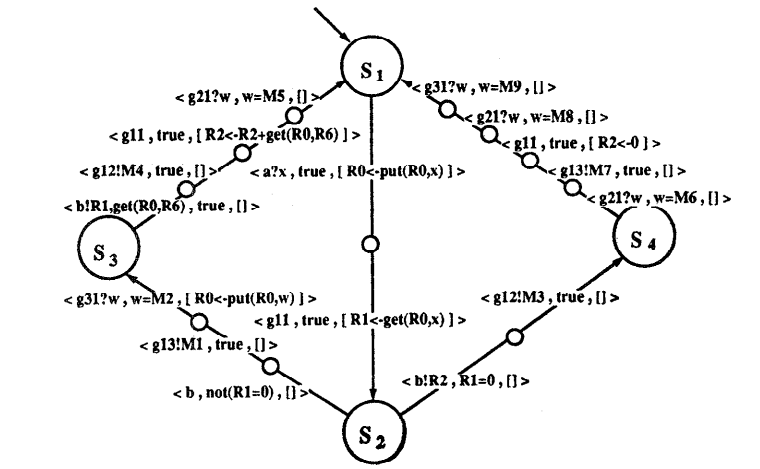
その場合，それらの中の1つの動作を実行するには，ノード間で合意をとる必要がある．この合意は例えば，ノード間で1つのトークンを回して，そのトークンを獲得したノードが動作を実行できるようにすれば解決するが[4]，以下での議論を簡単にするため，ここでは上記の制約条件1を設ける．

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，およびゲートへの出力を自ノードで行えることを表しており，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．

3.4 表記法

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) で表し，（状態 s_i の）責任ノードと呼ぶ．

また各レジスタ R_h の属するノードの集合を Rnode(R_h) で表す（一般に R_h の属するノードは複数考え



$$M_1 \equiv \mu^2, \{\langle R_1, v_1\rangle\} \quad M_2 \equiv \xi^2, \{\langle R_6, v_6\rangle\} \quad M_3 \equiv \xi^4$$
$$M_4 \equiv \mu^3, \{\langle R_6, v_6\rangle\} \quad M_5 \equiv \xi^3 \quad M_6 \equiv \mu^5$$
$$M_7 \equiv \beta^5, \{\langle R_2, v_2\rangle\} \quad M_8 \equiv \xi^5 \quad M_9 \equiv \xi^5$$

図 3 各 EFSM$_h$ の動作仕様
Fig. 3 Protocol entities' specifications.

られるので集合としている)．さらに，状態 s_i から始まる状態遷移の集合について，(イ)動作の実行可能性を判定する遷移条件の引数に指定されたレジスタ名，(ロ)出力動作の引数に用いられるレジスタ名，および(ハ)Snode(s_i) において (Alloc (EFSM) で指定されて)，保持すべきレジスタ名の集合和を 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_1)≡ノード 1, Snode(S_2)≡ノード 1,
Snode(S_3)≡ノード 1, Snode(S_4)≡ノード 2

Rnode Rnode(R_1)≡{ノード 1},
Rnode(R_2)≡{ノード 1, ノード 2},
Rnode(R_3)≡{ノード 3},
Rnode(R_4)≡{ノード 2},
Rnode(R_5)≡{ノード 3},
Rnode(R_6)≡{ノード 3}

Cset Cset(S_1)≡{R_1, R_2}, Cset(S_2)≡{R_1, R_2},
Cset(S_3)≡{R_1, R_2, R_6}, Cset(S_4)≡{R_2, R_4}

となる．

4. 各ノードの動作仕様の導出

4.1 基本方針

まず，導出する各ノードの EFSM$_k$ の動作の概略について述べる．

後で状態遷移図の縮約を行うが，基本的に各 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 EFSM の1つの状態遷移を通信のための幾つかの状態遷移の系列に置き換えることにより構成する．また，レジスタ更新の方法はいろいろ考えられるが，更新すべきレジスタを持つ各ノードが，計算に必要な引数のレジスタ値を他ノードから受け取り，自ノードで計算後更新を行うことにする．同一レジスタを複数のノードが個々に保持している場合でも，1つのノードでのみ計算して計算結果を他ノードに通信するということはせずに，各ノードで更新を独立に行うことにする．このとき，各ノードは EFSM の1つの状態 s_i に対して，次のことを行えば良い．(イ) s_i の責任ノード Snode(s_i) が実行可能な状態遷移 $s_i-\langle A, C, CR\rangle\rightarrow s_j$ を1つ選択し，動作 A を実行すること．(ロ)選択された状態遷移のレジスタ更新式 CR に従って各ノードのレジスタ値を更新すること．(ハ)状態 s_j の責任ノード Snode(s_j) が，(a)次に自分が(責任ノードとして)状態 s_j のことを実行すべきことを知ること，(b)(他ノードでの)すべてのレジスタ更新が終了し，状態 s_j に相当する状態に到達したことが分かること，および，(c)状態 s_j での次の動作を決定するのに必要なレジスタ値の情報 (Cset(s_j)) を知ること．

制約条件1より，(イ)の選択は Snode(s_i) 自身で行える．ただし，判定に必要なレジスタ値はすべて Snode(s_i) が知っていることを仮定しておく(制約条件2より初期状態ではこの仮定は成り立つ)．

上記(ロ)のためには，更新すべきノードは，(i)どの更新式に従って更新すべきか，という情報が必要である．またレジスタ値を送信すべきノードは，(ii)どこへどのレジスタ値を送るべきかという情報が必要である．各ノードがこれらの情報を(直接，あるいは間接的に)責任ノードからのメッセージとして受けとるために，以下で述べるメッセージすべてに状態遷移を識別するラベルを付加して送信することにする．これにより各ノードがメッセージを受信した際に，上記(i)，(ii)の情報を得ることが可能となる．

あとで，通信のコスト基準に合わせて，メッセージの削減を行うが，その評価基準としては，各状態遷移ごとに得られた状態遷移の系列で交換されるメッセージの総数のみを考える．従って，メッセージの内容(メッセージの長さ)は評価基準にはいれない．この仮定は実際のシステムを考えても妥当と思われる．

いま，Snode(s_i) が状態遷移 $s_i-\langle A, C, CR\rangle\rightarrow s_j$ を選択したとする．各 EFSM$_k$ は，(イ)，(ロ)を次の5つのフェーズで実現することにする．(1) Snode(s_i) が動作 A を実行する．(2) Snode(s_i) からレジスタ値を送るノードへきっかけを与えるメッセージ(以後 α 型メッセージと呼ぶ)を送る．(3) α 型メッセージを受けとったノードからレジスタ値を必要とするノードへレジスタ値の情報を持ったメッセージ(以後 β 型メッセージと呼ぶ)を送る．(4)各ノードがレジスタ更新式に従ってレジスタ値を更新する．(5)レジスタ更新を行った各ノードから更新が終了したことを知らせるメッセージ(以後 γ 型メッセージと呼ぶ)を Snode(s_j) に送る．

ただし，レジスタ更新を自ノードのレジスタのみを用いて行えるノードには β 型メッセージ等が届かないので，フェーズ(2)で更新のきっかけを与えるメッセージ(λ 型メッセージ)を Snode(s_i) から送ることにする．さらに，Snode(s_i) が知っている (Cset(s_i) に含まれる) レジスタの更新前の値や入力変数の値をそれを必要とするノードにフェーズ(2)において送るこ

とも考えられる．このメッセージをη型メッセージとする．

上述の(ハ)の(a)，(b)については上述のフェーズ(5)で，Snode(s_j)がレジスタ値の更新を行ったすべてのノードからγ型メッセージ（状態遷移の識別ラベル付）を受け取ることにより，自ノードが次の責任ノードであり，かつ，現状態遷移が終了しs_jに対応する状態に遷移したことを知ることができる．ただし，次責任ノードSnode(s_j)へまったくγ型メッセージが送られず，かつ，以下で説明するρ型メッセージも送られない場合は，現責任ノードが次責任ノードSnode(s_j)へ直接メッセージを送ることにする．このメッセージをθ型メッセージと呼ぶことにする．

(c)については，Snode(s_j)がCset(s_j)に含まれるレジスタのうち自ノードが保持していないレジスタの値を知る必要がある．そのレジスタ値（例えば，v_{R_h}）が今の遷移で更新されたものなら，更新後の値を上述のフェーズ(5)で，Rnode(R_h)に属するノードのいずれかから，メッセージ（以後ρ型メッセージと呼ぶ）として受け取ることにする．一方，更新されないレジスタ値の場合は，フェーズ(2)で送信依頼メッセージ（χ型メッセージ）をSnode(s_i)から送信し，それを受け取ったノードは必要なレジスタ値をSnode(s_j)へρ型メッセージで送信する．

これにより，次の責任ノードでも(イ)で述べた仮定が満たされる（よってすべての責任ノードで(イ)の仮定が成り立つ）．

まとめると，フェーズ(2)で送信するメッセージはα型，η型，λ型，χ型の4つの型があり，場合によっては2つ以上の型のメッセージを同時に送る必要がある場合もある．この場合，それらのメッセージはフェーズ(2)で1つのメッセージとして送信する．このようなフェーズ(2)で送信されるメッセージを，2つ以上の型の合成が生じたか生じなかったかにかかわらず，μ型メッセージと呼ぶことにする．同様にフェーズ(5)で送信するγ型，ρ型，θ型のメッセージを合わせてξ型メッセージと呼ぶことにする（**表1**）．

以上が各ノードの動作の概略である．以下では，各ノードが上述の各フェーズでどのノードとどのようなタイプ，内容のメッセージを交換すればよいのか，および，その際の交換メッセージの総数をできるだけ少なくするにはどうすればよいのかについて具体的に説明する．

表1 メッセージタイプ
Table 1 Message types.

型	from	to	内容	意味
α	Snode(s_i)	集合α	状態遷移ラベル	β型メッセージの送信依頼
μ η	Snode(s_i)	集合η	状態遷移ラベル レジスタ値 入力変数値	更新のためのレジスタ値，入力変数の転送
λ	Snode(s_i)	集合λ	状態遷移ラベル	更新指示
χ	Snode(s_i)	集合χ	状態遷移ラベル	ρ型メッセージの送信依頼
β	集合α	集合β	状態遷移ラベル レジスタ値	更新のためのレジスタ値の転送
γ	集合γ	Snode(s_j)	状態遷移ラベル	更新終了通知
ξ ρ	集合ρ	Snode(s_j)	状態遷移ラベル レジスタ値	Snode(s_j)が次状態の条件判定等に必要なレジスタ値の転送
θ	Snode(s_i)	Snode(s_j)	状態遷移ラベル	責任ノード交替

集合	集合に属するノードのもつ性質
α	更新に必要なレジスタの値を送るべきノード
β	更新に必要なレジスタの値を受け取るノード
γ	レジスタの値を更新すべきノード
η	Cset(s_i)に含まれるレジスタの値や入力変数値をSnode(s_i)から受け取るべきノード
λ	集合$\gamma-\eta-\beta$に含まれるノード
ρ	Cset(s_j)に含まれるレジスタの値をSnode(s_j)に送るべきノード
χ	集合$\rho-\gamma$に含まれるノード
μ	集合$\alpha\cup\eta\cup\lambda\cup\chi$に含まれるノード
ξ	集合$\rho\cup\gamma$に含まれるノード

4.1.1 各 EFSM$_k$ の動作

フェーズ(2)，(3)でそれぞれ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μ，βで表す．集合μは集合α，η，λ，χの和集合となる．ここで集合α，η，λ，χは，それぞれ，α型メッセージ，η型メッセージ，λ型メッセージ，χ型メッセージを受信するノードの集合を表す．

またフェーズ(5)で，γ型メッセージ，ρ型メッセージを送信するノードの集合をξで表す．集合ξは，γ型メッセージを送信するノードの集合γと，Snode(s_j)が必要とするレジスタ値を送るノードの集合ρの和集合となる☆．なお，ノード集合γは，この遷移でレジスタ値の更新を行うノードの集合となる（表1）．

このとき，EFSMの各状態遷移に対して，各EFSM$_k$

☆ メッセージの場合と異なり，θ型は除外している．

(ノード k) は次のように状態遷移の系列を実行すると考える.

[動作系列 **TS**]

(Proc 1). ノード k=Snode(s_i) ならば, k は状態 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.

(Proc 2). ノード k=Snode(s_i) ならば, k から $\mu(=\alpha+\eta+\lambda+\chi)$ に属するノードに μ 型のメッセージを送る.
ノード $k\in\mu$ ならば k が, Snode(s_i) からの μ 型のメッセージを受け取る.

(Proc 3). ノード $k\in\alpha$ ならば k 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.
ノード $k\in\beta$ ならば k が β 型メッセージを受け取る.

(Proc 4). ノード $k\in\gamma$ ならば k はレジスタ値を更新する.

(Proc 5). ノード $k\in\xi(=\gamma+\rho)$ ならば k から Snode(s_j) に ξ 型のメッセージを送る.
k=Snode(s_j) ならばノード k が ξ からのメッセージを受け取る.
θ 型メッセージを送る必要がある場合は, Snode(s_i) から Snode(s_j) へ直接送る.

(Proc 2), (Proc 3), (Proc 5) において, メッセージを受信するノードは, 受信の前に遷移条件を満たすかどうかを調べる. これらのメッセージ内容については 4.3 節, 4.4 節で詳細に述べる.

もし, ノード k が **TS** のどの動作にも関与しなければ, $s_i\to s_j$ は ϵ 遷移に置き換える. この方針に従って EFSM 中の各 $s_i\to s_j$ を **TS** 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.

4.1.2 メッセージ総数最少化に関する議論

上記の方法では, 3ステップ (フェーズ(2), (3), (5)) でメッセージ交換が行われる. もちろん, 4ステップ以上のメッセージ交換による方法など, 上記以外の導出方法も考えられる. これらの方法と比較して, 本方法はメッセージ総数が少ない, メッセージ交換によるシステム全体での遅延時間が比較的短い, 等の利点がある.

次にメッセージ数の削減について考える. 一般に, ノード k からノード k' へ同期メッセージのみ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, ここでは十分な転送速度が得られると仮定し, メッセージ長にかかわらず, 同一フェーズで各ノードに送られる μ, β, ξ 型メッセージをそれぞれ単位と考え, その仮定の下でメッセージの総数を少なくする方法を考える.

あるノードがあるレジスタの値を必要とする場合, η 型メッセージ, β 型メッセージのいずれかを用いることが考えられる. 例えばノード1が責任ノードで, ノード1, 3にレジスタ R_1 があり, ノード2が R_1 の値を必要とする場合, ノード1がノード3に α 型メッセージを送り, ノード3が β 型メッセージで R_1 の値をノード2へ送るよりは, ノード1が直接ノード2へ η 型メッセージで R_1 の値を送る方がメッセージ総数を少なくできる (前者は2個, 後者は1個). 一方, 今の例にレジスタ R_3 をノード3に加え, ノード2が R_1, R_3 の値を必要とする場合を考える. この場合ノード1からノード3へ α 型メッセージを送り, β 型メッセージを用いて R_1, R_3 の組をノード3からノード2に送るとメッセージ総数は2個で済むが, ノード1からノード2へ η 型メッセージでレジスタ R_1 の値を送るとレジスタ R_3 の値を送るためにさらに α 型と β 型メッセージを発行する必要があり合計3個のメッセージが必要である.

このような様々な場合を考慮に入れ, できるだけ効率的なメッセージの送信方法を考える必要がある.

4.2 メッセージの送信方法

以下では, 各ノードが, どのタイプのメッセージで, どのレジスタ値の組を, どのノードと送受信しなければならないかを, 具体的にどのように決定するかについて述べる.

4.2.1 メッセージの送信方法の決定に用いる命題変数

(I). Snode(s_i) を u とする. 上の (Proc 2) でノード u からノード v へ η 型メッセージを送らなければならないとき真となるような命題変数を ηuv とする. また, ノード u からノード v へレジスタ R_k (入力 x) の値を送信する必要があるとき真となる命題変数を ηuv_R_k (nuv_x) とする.

(II). 同様に上の (Proc 2) でノード u からノード v へ α 型メッセージを送る必要があるとき真となるような命題変数を αuv とする.

(III). 上の (Proc 2) でノード u からノード v へ λ 型メッセージを送る必要があるとき真となる命題変数を λuv とする.

(Ⅳ). 上の (Proc 5) でノード u からノード v へ χ 型メッセージを送る必要があるとき真となる命題変数を χuv とする.

(Ⅴ). 上の (Proc 2) で ηuv, αuv, λuv, χuv の何れかが真であるとき真となる命題変数を μuv とする.

(Ⅵ). 上の (Proc 3) 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命題変数を 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命題変数を βwv_R_h とする.

(Ⅶ). 上の (Pro 5) 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 γ 型メッセージ, ρ 型メッセージを送る必要があるとき真となる命題変数をそれぞれ, 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命題変数を ρvz_R_h とする. また, ξ が空集合のとき真となる命題変数を θuz とする. また, γvz, ρvz, θvz の少なくとも1つが真のとき真となる命題変数を ξvz とする.

4.2.2 各命題変数の決定に用いる制約条件

上の(Ⅰ)～(Ⅶ)の命題変数 ηuv, ηuv_R_h, ηuv_x, αuv, λnv, χuv, μuv, βwv, βwv_R_h, ξvz, γvz, ρvz, ρvz_R_h, θuz の間に次の不等式が成り立つ必要がある (ここでは各命題変数を 0, 1 のいずれかの値を取る整数型の変数とみなす).

以下, u を Snode(s_i) とする. また z を Snode(s_j) とする.

(1) $R_h \in$ Cset(s_i) なる各レジスタ R_h とノード $v(1\leq v\leq p)$ の組に対して,

$$\eta uv \geq \eta uv\_R_h$$

(2) 入力 x とノード $v(1\leq v\leq p)$ の組に対して,

$$\eta uv \geq \eta uv\_x$$

(3) 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),

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

(4) 各ノード $v(1\leq v\leq p)$ に対して,

$$\mu uv \geq \alpha uv,\ \mu uv \geq \eta uv,$$
$$\mu uv \geq \lambda uv,\ \mu uv \geq \chi uv \qquad \text{上述の(Ⅴ)に対応}$$

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

$$\alpha uv \geq \beta vw \qquad \text{上述の(Ⅱ)に対応}$$

これは β 型のメッセージを送信するノード v にはノード u から α 型のメッセージを送らなければならないことを表している.

(6) 各ノード $v(1\leq v\leq p)$ とレジスタ $R_h(1\leq h\leq m)$ 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,

$$\sum_{w\in \mathrm{Rnode}(R_h)} \beta wv\_R_h + \eta uv\_R_h \geq 1 \quad : R_h \in \mathrm{Cset}(s_i)$$
$$\sum_{w\in \mathrm{Rnode}(R_h)} \beta wv\_R_h \geq 1 \quad : R_h \notin \mathrm{Cset}(s_i)$$

これはレジスタ R_h の値を β 型, η 型のいずれのメッセージで受けとっても良いことを表している.

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

$$\beta wv \geq \beta wv\_R_1, \cdots, \beta wv \geq \beta wv\_R_m$$
上述の(Ⅵ)に対応

(8) 各ノード $v\in\gamma$ に対して,

$$\eta uv + \sum_{1\leq y\leq p \wedge y\neq u} \beta yv + \lambda uv \geq 1$$
上述の(Ⅲ)に対応

これはレジスタ値を更新するために γ に属するノード v は η 型, β 型, λ 型のいずれかのメッセージを受けとらなければいけないことを表している.

(9) 各ノード $v\in\gamma$ に対して,

$$\gamma vz = 1$$

(10) 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 ($z\notin$ Rnode(R_h)) なら,

$$\sum_{w\in \mathrm{Rnode}(R_h)} \rho wz\_R_h \geq 1$$

この式では, ρ 型メッセージを, 実際にレジスタを持っているところからのみ受けとることにしている. Snode(s_i) は Cset(s_i) として, 実際に自ノードが持っている以上のレジスタを知っている可能性があるがここではそれは取り扱わない.

(11) 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,

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

(12) 各ノード $v\notin\gamma$ に対して,

$$\chi uv \geq \rho vz \qquad \text{上述の(Ⅳ)に対応}$$

これはレジスタの更新は行わないが ρ 型のメッセージを送らなければならないノード v には, ノード u から χ 型のメッセージを送らなければならないことを表している.

(13) ξ が空集合のとき，

$$\theta uz=1$$

これは，ノード間のメッセージのやりとりが全くなく，しかて責任ノードが替わる場合，ノード u から直接ノード z へ θ 型のメッセージを送る必要があることを表している．

4.2.3 各変数の決定の方法

上述の(1)～(13)の制約条件を満足するような変数の組の中で，

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

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．ここで N は，この状態遷移において，p 個のノード間で交換されるメッセージの総数を表している．なお，各和演算では，自ノードへのメッセージの送信を行うような変数を排除している．

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．求めた解から各 EFSM$_k$ がどのタイミングでどのレジスタ値をどのノードと送受信しなければならないかがわかる．なお，0-1 整数線形計画問題は整数線形計画問題の制約式の各変数値を 0，1 に限定したもので，一般には NP 完全であるが[12]，実用的には様々な高速化の手法が考案されている．また多くの場合，レジスタの数は十数個程度，制約式の数も百個程度に抑えられると考えられるので，現実に計算可能であるといえる．

4.3 メッセージの内容

次にメッセージ間で交換するメッセージの内容について説明する．各状態遷移 $e(s_i\rightarrow s_j)$ について，ノード u (=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：

$$\langle\mu, \text{label}(e), \{\langle\text{“R}_k\text{”}, \text{value}(R_k)\rangle \mid \eta uv\_R_k=1\}, \{\langle\text{“x”}, \text{value}(x)\rangle \mid \eta uv\_x=1\}\rangle$$

ただし，label(e) は辺 e を他辺と区別するためのラベル，value(R_k)，value(x) はそれぞれ，レジスタ R_k，入力 x の値である．μ 型メッセージは，η，α，χ，λ 型メッセージの内容をあわせているが，α，χ，λ 型メッセージについては，内容は，タイプ，label(e) があればよい．η 型メッセージのためにレジスタ値等のデータが追加される．どのノード v にどのレジスタを送信すべきかは，変数 ηuv_R_k の値から判定できるので，これが1であるレジスタ値のみ送ればよい．

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

$$\langle\beta, \text{label}(e), \{\langle\text{“R}_k\text{”}, \text{value}(R_k)\rangle \mid \beta vw\_R_k=1\}\rangle$$

とし，ノード $v(1\leq v\leq p)$ からノード z (=Snode(s_j)) へ送信する ξ 型のメッセージの内容を

$$\langle\xi, \text{label}(e), \{\langle\text{“R}_k\text{”}, \text{value}(R_k)\rangle \mid \rho vz\_R_k=1\}\rangle$$

とする．すなわち，各メッセージにはタイプ，状態遷移 e のラベル名および，送信すべきレジスタ名（入力変数名）とその値が含まれている．

責任ノードでない場合は，責任ノードがどの状態遷移を実行したかがわからないので，メッセージに含まれる状態遷移 e のラベル名から状態遷移を判断する．なお図3では $\langle\eta, 2, \{\langle\text{“R}_1\text{”}, \text{value}(R_1)\rangle\}\rangle$ を η^2, $\{\langle R_1, v_1\rangle\}$ と略記している．また遷移条件 $w=\eta^2$ は入力メッセージのタイプが η 型でそのラベルが2であるとき真となる．

4.4 各 EFSM$_k$ の構成法

各ノードの状態遷移を 4.2 節で得られた述語の値にしたがって，動作系列 **TS** に置き換えれば各ノードの状態遷移系列を生成できる．もちろんこの構成法は，述語の値が正しく求まる限り停止する．ここでは，状態遷移の各ラベルの与え方を中心にやや詳しく述べる．各ノードには Alloc (EFSM) で割り当てられたレジスタに加え，作業用のレジスタ R_0 を割り当てる．

(Proc 1) では，ノード k が 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（レジスタ更新は行わない．これは (Proc 4) でまとめて行う）．ただし，Snode(s_i) であるノードが外部入力 x の受信動作をした場合，x の値をレジスタ R_0 に追加するレジスタ更新式 $R_0\leftarrow\text{put}(R_0, x)$ を与える．

(Proc 2) では，ノード k が 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，このための動作は適当な順序で直列に行えばよい．遷移条件は常に真である．各メッセージのタイプは μ でありラベルは選択された状態遷移が $e(s_i\rightarrow s_j)$ のとき，label(e) となる．k が Snode(s_i) 以外のノードで，μ 型のメッセージ m を受信する必要のあるノードなら，状態遷移 e に対応する状態遷移系列の遷移条件として $m=\mu^{\text{label}(e)}$ を与える．動作として，ノード u (=Snode(s_i)) からのメッセージ m の入力 $g_{uk}?m$，レジスタ更新式として，$R_0\leftarrow\text{put}(R_0, m)$ を与える．(Proc

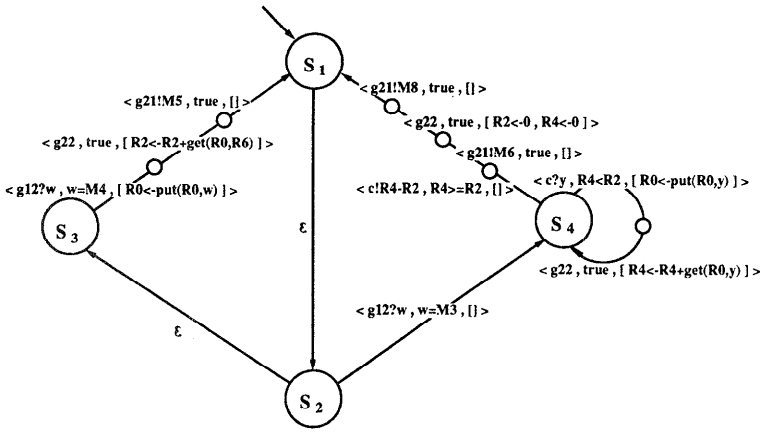


図 4 $EFSM_2$ の動作仕様（状態簡約前）
Fig. 4 $EFSM_2$ with ε moves.

3) で，複数の β 型のメッセージの送信を行う必要のあるノード $v(\in\alpha)$ は (Proc 2) と同様に適当な順序で直列に行う．また，β 型のメッセージを受信する必要のあるノード $k(\in\beta)$ は，それらのメッセージの受信動作を行う．k の動作として，ノード $v(\in\alpha)$ からのメッセージ m の入力 $g_{vk}?m$，レジスタ更新式として，$R_0 \leftarrow \mathrm{put}(R_0, m)$ を与える．受信動作も適当な順序で直列に行えばよい（受信メッセージの到着順は一意に定まらないが同一ゲートからは，同一タイプのメッセージは高々1個しか受信しないので受信動作の順序を適当に定めても問題は生じない．）．遷移条件として $m=\beta^{\mathrm{label}(e)}$ を与えておく．(Proc 4) でレジスタ R_0 から必要な入力レジスタの値を $\mathrm{get}(R_0, R_h)$，$\mathrm{get}(R_0, x)$ を用いて取り出し，レジスタ更新を行う．(Proc 5) では，k が ξ に属していれば，ノード $\mathrm{Snode}(s_j)$ に ξ のメッセージを送信する．k が $\mathrm{Snode}(s_j)$ なら，ξ 型メッセージの受信動作を先ほどと同様に行う．

4.5 状態簡約

上述の手続きによって導出される各 $EFSM_k$ の状態遷移図は，いわゆる ϵ 遷移を含んでいることがある．例えば，図1の EFSM から上述の手順によって導出されるノード2の $EFSM_2$ は**図 4** のようになる．

全体仕様にある状態は各 $EFSM_k$ において，責任ノードとなる状態とならない状態に分けることができる．各 $EFSM_k$ の作り方より，k が責任ノードである状態 s と責任ノードでない状態 s' 間に ϵ 遷移がないこと，および，責任ノードでない状態からの遷移は他ノードからのメッセージの受信動作で始まることが保証できる．そこで，各 $EFSM_k$ の状態の中で，責任ノードでない状態（図4では S_1, S_2, S_3）をすべて1つの状態（例えば S_0 とする）にまとめ，ϵ 遷移を取り除くことにより状態簡約を行う．図4の $EFSM_2$ から状態簡約を行うと図3の $EFSM_2$ が得られる．なお受信メッセージには遷移のラベルがついているので状態簡約で複数の状態を1つにまとめても受信メッセージの内容からもとの状態遷移図中のどの遷移が選ばれたかを区別することができる．よって簡約後の $EFSM_k$ は簡約前の $EFSM_k$ と同じ動作が行える．

4.6 $EFSM_k$ の構成法の正当性

上述の導出法で導出される分散システムの動作仕様 $\langle EFSM_1, \cdots, EFSM_p \rangle$ と要求仕様 EFSM の等価性が成り立つことの証明については文献 14) を参照のこと．なお本稿では各状態遷移ごとにメッセージの削減を考えているが複数の遷移を考慮するとさらにメッセージ数を削減できる場合がある．その例についても文献 14) を参照のこと．

5. おわりに

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[13]．現在 ASL/ASM で要求記述を行い，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．

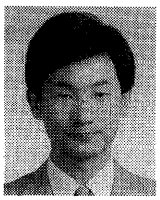
謝辞 有益なコメントをいただきました査読者の

方々に感謝いたします．

参考文献

1) Probert, R. and Saleh, K.: Synthesis of Communication Protocols: Survey and Assessment, *IEEE Trans. on Comp.*, Vol. 40, No. 4, pp. 468-476 (1991).
2) Khendek, F., Bochmann, G. V. and Kant, C.: New Results on Deriving Protocol Specifications from Service Specifications, *Proc. of ACM SIGCOMM '89*, pp. 136-145 (1989).
3) Bochmann, G. V. and Gotzhein, R.: Deriving Protocol Specifications from Service Specifications, *Proc. of ACM SIGCOMM '86*, pp. 148-156 (1986).
4) Langerak, R.: Decomposition of Functionality; A Correctness-Preserving LOTOS Transformation, *Proc. of Tenth IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification*, pp. 229-242, North Holland (1990).
5) Higashino, T.: Service Specification and Its Protocol Specifications in LOTOS, *IEICE Trans. Fundamentals*, Vol. E 75-A, No. 3, pp. 330-338 (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) 馬淵博之，高橋 薫，白鳥則郎：LOTOS に基づいたプロトコル仕様の導出，信学技報，IN-91-110 (1991).
8) Chu, P.-Y. M. and Liu, M. T.: Synthesizing Protocol Specifications from Service Specifications in FSM model, *Proc. Computer Networking Symp. '88*, pp. 173-182 (Apr. 1988).
9) Chu, P.-Y. M. and Liu, M. T.: Protocol Synthesis in a State-Transition Model, *Proc. COMPSAC '88*, pp. 505-512 (1988).
10) Park, D.: Concurrency and Automata on Infinite Sequences, *Theoretical Computer Science, Lecture Notes in Computer Science*, Vol. 104, pp. 167-183, Springer-Verlag (1981).
11) Milner, R.: *Comunication and Concurrency*, Prentice-Hall (1989).
12) Garey, M. R. and Johnson, D. S.: *Computers and Intractability*, FreeMan (1979).
13) 大蘆雅弘，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)，Vol. J 73-D-I, No. 12, pp. 971-978 (1990).
14) 岡野浩三，今城広志，東野輝夫，谷口健一：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ノードの動作記述の自動生成，情報処理学会研究会報告，92-PRG-8, 67, pp. 211-218 (1992).

(平成4年11月2日受付)
(平成5年4月8日採録)



岡野 浩三（正会員）

昭和 42 年生．平成2年大阪大学基礎工学部情報工学科卒業．平成5年同大学院博士後期課程中退．同年同大学基礎工学部情報工学科助手，現在に至る．代数的手法によるソフトウェア設計開発法，分散システム等の研究に従事．電子情報通信学会会員．

今城 広志

昭和 44 年生．平成4年大阪大学基礎工学部情報工学科中退．同年，同大学院基礎工学科研究科博士前期課程入学．現在，在学中．分散システムに関する研究等に興味を持つ．

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 年同大学博士課程修了．同年，同大学助手．平成2年モントリオール大学客員研究員．平成3年大阪大学基礎工学部情報工学科助教授，現在に至る．分散システム，通信プロトコル等の研究に従事．工学博士．電子情報通信学会，ACM, IEEE 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 年同大学院基礎工学研究科博士課程修了．工学博士．同年同大学基礎工学部助手．現在，同情報工学科教授．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．