マルチメディア通信と分散処理 80－14
グループウェア 21－14
（1997. 1. 30）

通信プロトコルのLOTOS仕様から並行EFSM群への変換の一手法

水野 健太郎†, 安本 慶一††, 森岡 澄夫†, 東野 輝夫†, 谷口 健一†

† 大阪大学基礎工学部情報科学科, 豊中市待兼山町 1-3, Tel: 06-850-6607
E-mail: {k-mizuno, morioka, higashino, taniguchi}@ics.es.osaka-u.ac.jp

†† 滋賀大学経済学部情報管理学科, 彦根市馬場 1-1-1, Tel: 0749-27-1060
E-mail: yasumoto@biwako.shiga-u.ac.jp

あらまし

本稿では，プロトコルのLOTOS仕様から複数並行EFSM群を自動生成する手法を提案する．本手法では，まず複数プロセス間に並列・同期などが指定可能なあるクラスで記述されたLOTOS仕様を，各EFSMに割当可能な部分動作式群に分解する．分解されたEFSM群を順序回路として並行に動作させる．この際，仕様どおり順にの動作が実行されるようにするため，EFSM群の各状態組でどのイベントを同期並列実行できるのかを計算する機構を組み合わせ回路として実現する．本手法は，マルチランデブなどが指定された抽象度の高いプロトコルの形式仕様からハードウェア回路を自動生成するために用いることが可能である．

キーワード：LOTOS，プロトコル，自動合成，EFSM，ハードウェア回路

A Method to Derive Concurrent EFSMs from Protocol Specifications in LOTOS

Kentaro Mizuno†, Keiichi Yasumoto††, Sumio Morioka†, Teruo Higashino† and Kenichi Taniguchi†

† Dept. of Information and Computer Sciences, Osaka University, Toyonaka, Osaka 560, Japan
Tel: +81-6-850-6607, E-mail: {k-mizuno, morioka, higashino, taniguchi}@ics.es.osaka-u.ac.jp

†† Dept. of Information Processing and Management, Shiga University, Banba 1-1-1, Hikone 522, Japan
Tel: +81-749-27-1060, E-mail: yasumoto@biwako.shiga-u.ac.jp

Abstract

In this paper, we propose a method to derive concurrent EFSMs from protocol specifications in a subclass of LOTOS where multiple processes can be combined by parallel and synchronization operators. In the method, first we decompose a behavior expression of a specification into sub-expressions such that each expression can be implemented by an EFSM. In order to keep the temporal ordering of events among those EFSMs, we have designed a rendezvous mechanism to calculate a synchronizing event and its combination of synchronizing EFSMs from each reachable global state of all EFSMs. The proposed method can be applied to the synthesis of hardware circuits from abstract protocol specifications with multi-rendezvous.

key words: LOTOS, protocol, synthesis, EFSM, hardware circuit

1 まえがき

FDDIやATMなどの高速通信ネットワークが発展するにつれ，その上で動作する通信プロトコルを，ハードウェア（LSIチップ）として実装することによって高速化する方法の研究が注目されている．通信プロトコルのハードウェア化においては，プロトコルの形式仕様からハードウェア回路を自動合成し，素早く性能評価を行うラピッドプロトタイピングの手法の確立が切望されている．

自動合成を行う際には，(1) 複数の並行プロセス間にマルチランデブ（複数プロセスの集合のうち，一定の条件を満たすサブセットの間でイベントを同期実行しデータ交換を行う機構）が指定されているような，抽象的かつ構造的な仕様が扱えること，(2) 高速に動作する回路として実現できること，(3) 回路の規模をある程度現実的なものに抑えられること，などが評価基準となりうる．

近年，形式仕様記述言語で書かれたプロトコルの形式仕様から，ハードウェアを自動導出するための研究が行われている [4, 5, 6]．文献 [5, 6] では，それぞれ，SDL, Estelleからの合成手法を提案しているが，マルチランデブを含むような構造的な仕様を対象としていない．文献 [4] では，LOTOS仕様からハードウェア記述言語VHDLへの変換手法が提案されているが，マルチランデブの指定を2つのプロセス間に限定している．

本稿では，抽象度の高いプロトコルの形式仕様からハードウェア回路を自動合成することを目的として，複数プロセス間に並列・同期などが指定可能なクラスのLOTOS仕様から複数並行EFSM群を自動生成するための一手法を提案する．

本手法では，まず，ハードウェアとして実現する際の回路の規模を考慮して，プロセス間の同期や並列，割込などの指定が可能でかつ複数EFSMからなるハードウェア回路として実現可能なLOTOSのあるサブクラスを設定する．次に，本クラスで記述されたLOTOS仕様を並行動作を含まない部分動作式群に分解し，同時に呼び出されるプロセス数に近い数のEFSMで実現する方法について述べる．

複数のEFSM間のマルチランデブを実現する一つの方法として，個々のEFSMの遷移条件に，他のEFSMとの同期のための待ち合わせおよび合意のための動作を埋め込む方法が考えられる．しかし，この方法では同期のたびに合意や競合回避のための通信動作が実行されることになり，同期が多く含まれる仕様では実行効率が悪くなる．そ

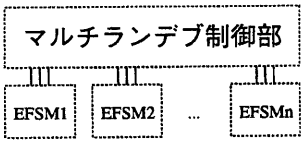

図 1: 回路の全体構造

こで，本手法では，イベントの同期実行を行うために成り立たなければならないガード式などを計算し，同期指定されている幾つかのEFSMがある状態組（各EFSMが互いに同期するイベントを実行できる状態）に達したときに，各EFSMに対してどのイベントが同期実行できるのかを知らせる機構（マルチランデブ制御部と呼ぶ）を全体で一つ用意し，それを用いてマルチランデブの実行制御を行う．マルチランデブ制御部を組合せ回路として構成することにより，どのEFSMの組合せでどのイベントが実行可能なのかを，EFSMの状態組から即座に計算できるため，EFSM間のランデブ処理が高速に行える．この制御部を用いて，複数個のマルチランデブ間の競合回避も行っている．

マルチランデブ制御部は，全EFSMの現到達状態，ゲートからの外部入力値および内部レジスタ値などから，各EFSMについて実行すべき遷移を1つずつ選んで出力する回路として実現する．現実的な規模の回路を得るため，複数のモジュールにより構成する．

マルチランデブ制御部を自動合成するためには，(1) EFSM群がどの状態組の時にどのEFSMの組合せでどのイベントが実行可能なのか，(2) 同期実行可能な各イベントについて，そのイベントを実行するためにはどのような条件（論理式）が成り立たなければならないのか，という情報が必要である．提案するLOTOSのサブクラスでは，同期するプロセスの数をゲートごとに固定し，得られる各EFSMについて初期状態から各遷移をトレースすることで，これらの情報を静的に求めている．

提案する手法をAbracadabra ProtocolのLOTOS仕様[2]に適用し，評価したところ，実用規模のプロトコル仕様を実行効率の良いハードウェア回路として実現可能であることが確かめられた．

2 LOTOS仕様から並行EFSM群への変換の基本方針

ハードウェア回路を自動合成するための一ステップとして，与えられたLOTOS仕様から並行に協調動作するEFSM群を得る．本稿におけるEFSM生成の基本方針を次に示す．

- LOTOS仕様の動作式を並列オペレータなどの構造に従って，イベントの連接，選択，これらの繰り返しなどからなる部分動作式に切り分け，EFSMに割り当てる．
- EFSM間のイベントの同期は専用のマルチランデブ制御部を用いて実現する．

LOTOS仕様から得られる回路の全体を図1に示す．

2.1 対象とするLOTOS仕様のクラス

LOTOSはプロセスの動的な生成を可能にしており，あるLOTOS仕様において，ランデブにより同期するプロセスの組合せは，一般に無限通り存在する可能性がある．一般にこのようなプロセスをハードウェア化することは容易ではない．このため，まず，同時に稼働するプロセスの数および同期するプロセスの組合せが有限となるようなクラスを設定する．

表 1: 対象とするLOTOSのクラス

$B1$ ::=	$B2 \mid B2 \lvert[G]\rvert B1$
$B2$ ::=	$B3 \mid P \mid B2 \lvert\lvert\rvert B2 \mid B2 >> B2 \mid B2 [> B3$
$B3$ ::=	$Action; B2 \mid Action; B2 [] B3$
$Action$ ::=	$gate\ Vdec \mid gate\ Vdec\ [guard]$
$Vdec$ ::=	$\varepsilon \mid ?var : sort\ Vdec \mid !exp\ Vdec$

（ここで，Gはオペランド間で同期させたいゲートの並び，Pはプロセス呼出しである．また，任意のサブプロセスPの動作式は，$B2$で表されるクラスで記述する）

本稿では，LOTOS仕様のメインプロセスの動作式として表1に示すクラスを対象とする．また，次のような制限を加えている．

(1) 各プロセスPによる自分自身のプロセス呼出しは，$\alpha >> P$あるいは$\beta; P$の形でなければならない．

(2) メインプロセスの動作式 （$B \lvert[G_1]\rvert B \lvert[G_2]\rvert \cdots \lvert[G_{n-1}]\rvert B$）において，$G_1 \supseteq G_2 \supseteq \cdots \supseteq G_{n-1}$でなければならない．

(3) 並列オペレータの両側のオペランドでは，同じ変数を参照してはならない．

(4) 割込を含むプロセス $P := B [> \sum a_i; B_i$において，各B_iからPが呼び出されてはならない．

(1) はシステムの動作を有限にするための制限であり，(2) は同期するプロセスの数を一定になるよう制限し，マルチランデブ制御部を一つの回路により実現するためのものである（ランデブ制御部を複数用いることにより，扱うクラスを拡張することは可能）．また，(3) は並行EFSM間で同じレジスタへのアクセスの競合が発生するのを防止するためである．また，LOTOSは，無限に階層化された割込の指定を可能にしているが，本稿はハードウェア回路を対象としているため，(4) において，有限の階層の割込に制限している．

表1は，同期オペレータが指定された動作式 $B\lvert[G_1]\rvert B \lvert[G_2]\rvert \cdots \lvert[G_{n-1}]\rvert B$において，各オペランド$B$に同期オペレータが含まれないようなクラスを表している．各動作式Bは幾つかの並行EFSMで実現可能であり，そのようなEFSMの集合を，以後同期グループと呼ぶ．上記の動作式では，各同期グループについてただ一つのEFSMが他との同期に参加するため，同期制御用の回路の簡単化が期待できる．

本稿で扱うLOTOSのサブクラスでは，一般のLOTOSと同様に各ゲートを入力・出力の両方に使用可能である．しかし，一般にハードウェア回路では，各入出力ポートは片方向にしか使用できないため，本稿では，ハードウェア回路の合成時に各ゲートを入力・出力の2ポートとして実現し，各イベントの入力および出力をそれらのポートに連結する．また，各ポートのデータ型は一意に定まるものとする．

2.2 変換対象とするEFSM

本稿では，LOTOS仕様の変換対象として，通常のFSMに複数の有限長のレジスタ，および各遷移に実行条件が指定可能なEFSM群を想定している．また，2.1節のクラスでは，複数パラメータを持つイベント（例えば，a?x:int!true

など）が記述可能なため，一つの遷移で複数レジスタへの同時代入が可能であるようなEFSMを想定する．

任意のイベントと連接，選択，およびそのような形式の有限のプロセス呼出しからなるLOTOSの動作式は，イベント実行後の動作式を状態，イベントを遷移，変数をレジスタに対応させることで，一つのEFSMにマッピング可能である．

2.2.1 EFSMにおける遷移の種別

2.1節のクラスのLOTOS仕様を並行EFSM群として実現する際には，a;(P ||| Q) のような動作式において，a;Pを一つのEFSM (EFSM1) に，Qを別のEFSM (EFSM2) に割り当てる．そのような割り当てを行った場合には，最初EFSM2を待機状態にしておいて，EFSM1においてイベントaを実行後，EFSM2を開始させる必要がある．

割込を含む動作式 $B[> \sum_{i=1}^{k} a_i; B_i'$ においては，部分動作式 B, $\sum_{i=1}^{k} a_i; B_i'$ をそれぞれEFSM1, EFSM2に割り当て，EFSM2においてイベント a_i が実行された時に，EFSM1を停止させる必要がある．

以上の議論から，各EFSMにはイベント以外の特別の遷移として，(1) EFSM間での起動信号の送受信動作 (SendInit, RecvInit)，(2) 他のEFSMへの停止信号の送信動作 (SendStop) を用いることにする．また，プロセスの再帰呼出しによるループを実現するため，(3) 各EFSMにおいて特定の状態に移るための動作 (Goto) を用意する．

2.2.2 各遷移の実行条件

2.1節のクラスでは，遷移（イベント）の実行条件（ガード）を指定することが可能である．ガードは，各EFSMで使用されるレジスタの内容およびゲートからの外部入力値とその上の演算および論理演算からなる論理式である．各EFSMにおいてある遷移が実行できるのは，以下の場合である．

（他のEFSMとの同期が指定されていない場合）現状態からの遷移のうち，ゲートからの外部入力，ガード（条件式）が成立するものが存在する．

（他のEFSMとの同期が指定されている場合）現状態からの遷移のうち，ゲートからの外部入力，ガード式が成り立ち，かつ，他との同期が可能なものが存在する．

2.1節のクラスでは，ある時点で上記の条件を満たす遷移は，各EFSMにおいて複数存在するかもしれない（例えば，EFSM1 |[a,b]| (EFSM2 ||| EFSM3), EFSM1:= a;exit [] b;exit [] c;exit, EFSM2:= a;exit, EFSM3:=b;exit のような場合では，EFSM1においてa,b,cのいずれかのイベントが実行可能になる）．このような場合には，EFSM間で矛盾を生じさせないため，EFSM間でどのイベントを実行するのかの合意が必要である．

本手法では，どの遷移が可能かの判定（EFSM間の合意を含む）を中央に設置された専用のマルチランデブ制御部により一括して行う．あるグローバル状態から複数の遷移が実行可能である場合に，ただ一つの遷移の実行が可能となるようにマルチランデブ制御部を構成することで，各EFSMに余分な遷移を付け加えずに，EFSM間の同期を含む実行制御を行う．

2.3 LOTOS動作式の各EFSMへの分解

割り当ての基本方針は，(1) LOTOSの動作式の並列・割込オペレータ（|||, |[G]|, ||, [>）で区切られた動作式を別々のEFSMに割り当てること，(2) 子プロセス呼び出し時に

表 2: EFSMへの割当てアルゴリズム

$Assign(B)$ ==
  if $(B = \sum_{i=1}^{k} a_i; B_i)$ *then*
    $\sum_{i=1}^{k} a_i; Assign(B_i)$
  if $(B = B' ||| B''$ または $B' |[G]| B'')$ *then*
    $SendInit(efsm\#(B''))$; $Assign(B')$
  if $(B = B' >> B'')$ *then*
    $Assign(B') >> Assign(B'')$
  if $(B = B' [> \sum_{i=1}^{k} a_i; B_i)$ *then*
    $Assign(B')$
  if $(B = P)$ *then*
    if (Pが末端再帰) *then*
      $Goto(B_P$の初期状態$)$; *stop*
    else
      $Assign(B_P)$

$EFSM(B, From)$ ==
  if $(From \neq nil)$ *then*
    $\{RecvInit(From); Assign(B)\} \cup EFSM(B, nil)$
  else if $(B = \sum_{i=1}^{k} a_i; B_i)$ *then*
    $\bigcup_{i=1}^{k} EFSM(B_i, nil)$
  else if $(B = B' ||| B''$ または $B' |[G]| B'')$ *then*
    $EFSM(B', nil) \cup EFSM(B'', efsm\#(B'))$
  else if $(B = B' >> B'')$ *then*
    $EFSM(B', nil) \cup EFSM(B'', nil)$
  else if $(B = B' [> \sum_{i=1}^{k} a_i; B_i)$ *then*
    $EFSM(B', nil) \cup \{Assign(\sum_{i=1}^{k} a_i; B_i)\}$
    $\cup \bigcup_{i=1}^{k} EFSM(B_i, nil)$
  else if $(B = P$かつ Pが末端再帰でない$)$ *then*
    $EFSM(B_P, nil)$
(ここで，$efsm\#(B)$ は Bに割当てられるEFSMの番号である)

は，その動作を同じEFSM内に展開すること，(3) 並行動作するEFSM群が全体として仕様の通りに動作するよう，前節で述べた特別な送受信動作を付け加えること，である．

動作式 B から逐次的に実行できる部分動作式を抽出するアルゴリズム $Assign(B)$ を表2に示す．まずメインプロセスの動作式 B_0 について，$Assign(B_0)$ により，逐次的に実行できる部分動作式を抽出し，一つのEFSMに割り当てる．アルゴリズム $Assign(B)$ は，途中で同期・並列，割込オペレータで複数動作式に分岐している場合に，構文上左側の動作式のみを抽出する．また，並列・同期オペレータにより分岐している構文上右側の動作式に割当てられるEFSMに起動信号を送信する動作を付け加える．

動作式 B から同期・並列，割込オペレータで分岐している構文上右側の動作式のEFSMへの割当てには，動作式 B から，各EFSMに割当て可能な部分動作式の集合を求めるアルゴリズム $EFSM(B, From)$ を用いる．パラメータ $From$ は分岐元の動作式に割当てられたEFSMの番号を記憶し，どのEFSMからの起動信号を受信するのかに用いられる．$EFSM(B, From)$ において，$From$ に分岐元のEFSM番号が記憶されている場合は，動作式 B はEFSMへまだ割当てられていないので，$Assign(B)$ により割り当て，前に起動信号の受信動作 $RecvInit(From)$ を付け加える．$From$ に値が記憶されていない時（nil），Bの一部動作式は $Assign()$ により既にEFSMに割当済なので，並列・同期・割込オペレータで分岐している部分を再帰的に探索する．割込が指定されている動作式では，割り込む側のEFSMのイベントを実行したとき，割り込まれる側のEFSM群に停止信号を送るようにする．各オペレータにより分岐した部分動作式についても再帰的にアルゴリズム $EFSM$ が適用され，全ての部分動作式がEFSMに割り当てられるまで繰り返す．メインプロセスの動作式を B_0

とすると，$\{Assign(B_0)\} \cup EFSM(B_0, nil)$ により，全ての EFSM の動作式の集合を求めることができる．

例えば $(a_1; B_1 ||| a_2; B2) >> (a_3; B_3 ||| a_4; B_4)$ のような動作式は，$EFSM_1 = a_1; B_1 >> a_3; B_3$,$EFSM_2 = a_2; B2$, $EFSM_3 = a_4; B4$ のように割り当てられる．

2.4 各 EFSM の状態割り付けおよび最適化

前節で得られた EFSM 群は LOTOS で記述されており，これらの記述からレジスタ転送レベルの EFSM 群を得るには，(1) 記述に含まれる内部変数のレジスタへの割り当て，(2) ガード式やイベント出力値の計算処理の明確化，などを行う必要がある．これらについては，我々が文献 [3] で提案した手法を利用し，各 EFSM についてある評価基準のもとで総実行ステップ数が最小となるようなものを生成する．

3 マルチランデブ制御部の構成

以下では，2.1節のクラスで記述された LOTOS 仕様のメイン動作式を $B = B_1 |[G_1]| \cdots |[G_{n-1}]| B_n$ で表すことにする（各動作式 B_k は ||| を含んでも良い）．また B中に登場する全てのイベントの集合を A で表し，各イベント$\alpha \in A$ について，$gate(\alpha)$，$guard(\alpha)$ でαで用いられているゲート，およびガード式が参照できるものとする．各 $g \in G_1$について，動作式間で同期指定されているイベントの組（同期イベント組）の集合を $sync\_set = \{(e_1, \cdots, e_k) | rend(e_1, \cdots, e_k) = true\}$ とする（ここで，$rend(e)$ は同期イベント組 e の各イベントが，e の他のイベントと同じゲートを持ち，かつ入出力パラメータの数，型が一致するとき真となる述語である．また，各 e_iは仕様中のイベントの出現である）．

マルチランデブ制御部は，(1) EFSM の現到達状態組，(2) ゲートからの外部入力，(3) 各 EFSM の所有するレジスタの内容，を入力として各 EFSM において次に実行するイベント（遷移）の ID を出力する組合せ回路として構成する（図 2）．

マルチランデブ制御部は，各同期イベント組を実行するために成り立つべき条件を計算するガード計算部および，同期イベント組が同時に複数存在する場合に，その一つを一意に決定する同期イベント組選択部，同期のための条件が成り立たないときに，同期指定されていない複数のイベントから実行するイベントを決定するローカルイベント選択部の 3 つの部分から成る．

一般には，同期グループの要素数や選択オペレータによる分岐の数が多いときは，同じグローバル状態から実行できる同期イベントの組み合わせが多くなり，同期条件判定にかかる計算量は多くなる本手法では，この点を考慮し，複数の同期条件を並列に計算したうえで，同期条件が満たされた同期イベント組の中から競合が起こらないようにイベント組を選択することにより，同期イベント組の総数に比例しない速度でマルチランデブの条件判定が行える．

3.1 ガード計算部の構成

ガード計算部は，EFSM で用いられているレジスタ群 $R = \{r_1, \cdots, r_m\}$ および外部入力 $I = \{i_1, \cdots, i_k\}$ と EFSM の状態組 $S = \{s_1, \cdots, s_n\}$ を入力とし，現状態組 S からの（遷移として存在する）各同期イベント組の実行条件の真偽を出力する組合せ回路 $Guard(R, I, S)$ として以下のように構成する．

(1) 各同期イベント組 $e = (e_1, \cdots, e_k) \in sync\_set$ について，各イベント e_iのガード式 $guard(e_i)$ を組合せ回路として実現し，それらの論理積をとることで，e の実行条件を判定する回路を構成する．

(2) これらの組合せ回路の出力のうち同期イベント組選択部で必要なものは，EFSM の現グローバル状態から可能な各同期イベント組の実行条件の真偽値だけであるため，EFSM の現状態組およびからそれらのみを選択する回路を設ける．

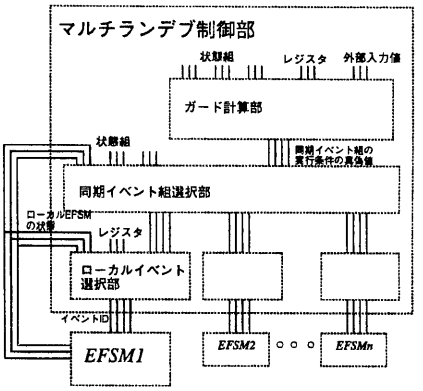

図 2: マルチランデブ制御部

マルチランデブの処理結果として，同期イベント組 $e = (\cdots, e_i, \cdots, e_j, \cdots)$ の入力イベント e_i中の入力変数には，出力イベント e_jの出力値が代入される [1] ため，組合せ回路 $guard(e_i)$ の入力は，同期相手の出力値（レジスタまたは演算回路）となるよう構成する（たとえば，イベント a?x[p(x)] と a!y[q(y)] は同期する時，x に y の値が代入されるため，実行条件として「p(y) かつ q(y)」が成立する必要がある）．ただし，イベント e_iと同期する可能性のある他の EFSM のイベントが複数ある場合（$(\cdots, e_i, \cdots)$ であるような同期イベント組が $sync\_set$ 中に複数ある場合）には，EFSM の状態組から，その状態で同期するイベントの出力値が e_iの入力変数に代入されるよう，組合せ回路 $guard(e_i)$ の入力はセレクタにより選択できるように構成する（図 3）．同期イベント組 e に出力イベントが含まれない場合には，各イベントのガード式において，外部ゲートからの入力値が評価されるよう回路を構成する．

また，2 つ以上の同期イベント組，例えば，$(\cdots, a?x[p(x)], \cdots, a!y, \cdots), (\cdots, a?x[p(x)], \cdots, a!z, \cdots)$ が，あるグローバル状態から同時に可能な場合には，両方の実行条件 p(y) および p(z) を同時に評価する必要がある．本手法では，このような場合には，同じガード式 p(x) を同時に可能な同期イベント組の数だけ用意する．これにより，同期イベント組の数に比例しない時間で同期条件を判定することが可能である．

通信プロトコルにおける条件判定には，外部入力の型チェックや内部レジスタと定数の比較など，比較的単純なものが多いため，$guard(e_i)$ の回路を複数設けることで回路規模が著しく大きくなることは少ないと考えられる．

3.2 同期イベント組選択部の構成

EFSM のある状態 s から出ている遷移（イベント）のうち，同期指定されているものの集合を $synch(s)$ で表す．$synch(s)$ に含まれるイベントはある識別子（遷移 ID）に

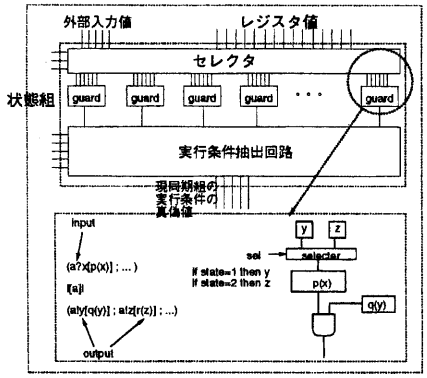

図 3: ガード計算部

より互いを識別可能とする．イベント e の遷移 ID を $ID(e)$ で表す．

同期イベント組選択部は，現状態組およびガード計算部の出力から実行する同期イベント組をプロセス間で競合が起こらないように選択する回路，および選択された同期イベント組から，各 EFSM において実行するイベントの ID（遷移 ID）を出力する回路から構成する．

複数のイベント組についてガード計算結果が真である場合，イベント組の間で競合が生じない限り，そのすべてのイベント組が並列に実行されるように出力される遷移 ID を決める．競合が生じる場合は，設計者があらかじめ決めておいた優先順位に従い，優先順位が高い同期イベント組を選択する（公平性を重視したい場合は，乱数回路やタイマー等を用いて実行可能なイベントが適当な確率で選択されるような回路にすることも可能である）．

選択された同期イベント組 $e=(e_1,e_2,\cdots,e_k)$ の各イベント e_i が属する EFSM には，$ID(e_i)$ が出力される．e が属さない EFSM には，$ID(nil)$ =*stall/local* を出力する．ここで，*stall/local* はその EFSM において，他の同期イベント組が可能になるまで待機（*stall*）するか，あるいは，同期指定されていないイベント（*local*）を実行することを表している．また，ガード計算部の出力が全て偽であった場合は，全ての EFSM に対して遷移 ID として *stall/local* を出力する．

3.3 ローカルイベント選択部の構成

同期イベント組選択部で同期イベントが選択された場合はその遷移 ID をそのまま出力し，*stall/local* が選択された場合には，現状態から実行可能なローカルイベント（同期指定されていないイベント）のうち 1 つを選ぶ．同期イベント組選択部と同様に，実行条件を満たすローカルイベントが複数ある場合は，設計者があらかじめ決めておいた優先順位に従い選択するものとする．

3.4 マルチランデブ制御部の自動合成

前節のマルチランデブ制御部を構成するには，(1) 全同期イベント組の集合 *sync_set* を求める必要がある．また，各イベント同期組 $e \in$ *sync_set* について，(2) e を実行可能な EFSM の状態組，(3) e の実行条件（e に属する全イベントのガードの論理積）も求める必要がある．

本手法では，これらの情報（以後ランデブ表と呼ぶ）を LOTOS 仕様をトレースすることで静的に求める．

表 3: ランデブ表

状態組				同期イベント組 (ID)				実行条件
c_1	c_2	$\cdots$	c_n	e_1	e_2	$\cdots$	e_n	*guard*
3	2		-	1	1		-	p(x)
				1	2		-	p(y) and q(y)
3	-		5	2	-		1	r(z)
-	4		7	-	1		3	s(w) and t(u)
:	:	:	:	:	:	:	:	:

LOTOS 仕様の動作式 Bから，2.3節の方法で抽出した EFSM の組を $EFSM=(EFSM_1,\cdots,EFSM_n)$ とする．各 $EFSM_i$ は 4 項組 (s_{i_0},S_i,T_i,V_i) で表すことができる．ここで，s_{i_0} は初期状態，S_i は状態集合，T_i は遷移（イベント）の集合，V_i は変数の集合である．ランデブ表の求め方の手順を以下に示す．

1. 現状態組を $C=(c_1,\cdots,c_k)$ とする．最初 $C=(s_{1_0},\cdots,s_{k_0})$ から始める（ただし，*RecvInit* を待っている状態の EFSM は，該当する EFSM の SendInit がトレースされるまで，状態を変えないものとする）．

2. EFSM 間に設定されているオペレータの情報から，現状態組 C において，同期イベント組 $(e_1,\cdots,e_l)$ が存在するか調べる．見つかった同期イベント組の集合を Eとする．

3. $E\neq\emptyset$ の時は，任意の $e\in E$に対して，e を実行するための条件（e に含まれる全イベントのガードの論理積）を求める．求めた条件式が成立する場合，現状態組 Cおよび e の各イベントの遷移 ID とその実行条件式をランデブ表に登録する（ただし，条件式にゲートからの外部入力変数が含まれる場合は，論理式の真偽を評価せず，常に成り立つものと考える）．

4. E中の条件式が成立した各 e について，e の各イベントを各 EFSM で実行した後の状態の組を新たな状態組 Cとし，2 からの処理を再帰的に繰り返す．ただし，幾つかの EFSM において，ローカルイベント（同期指定されていない遷移）が存在する場合は，それぞれの遷移を実行した後の状態を各 EFSM の新たな状態として，2 からの処理を再帰的に繰り返す．その際，各 EFSM に複数の遷移が存在する時は，EFSM 間で全ての組合せについて行う．

5. 全ての EFSM が終了状態に達した時あるいは既に訪れた状態組に達した時に探索を終える．

以上の手順により，到達可能な状態組における全ての同期イベント組およびその実行条件を求めることができる．求めたランデブ表の形式を表 3に示す．

3.5 マルチランデブ制御部の簡単化

マルチランデブ制御部は，上述のような組み合わせ回路で実現するが，各 EFSM の各状態からの同期イベントが多数あると，他との同期イベント組（同期するイベントの組合せ）の数が膨大になり，回路の規模が大きくなるため，そのような場合には以下の工夫を行う．

- 同時に実行されることがない幾つかの動作式群を 1 つの EFSM に割り当てる．
- 内部変数を引数に持つガード計算については，EFSM をレジスタ転送レベルに変換した時点で，ガード計算の処理を各 EFSM の動作に埋め込む．

- 通常は到達可能な状態空間が積マシンの状態空間に比べて小さいので，同期イベント選択部の前に状態エンコード部を設ける．

また，ガード条件の内容を用いた到達不能解析を行うことにより，到達可能なグローバル状態の数を減らすことも考えられる．

3.6 マルチランデブ制御部の実行効率

組合せ回路の実行効率を評価する際には，回路の実現に必要な論理ゲートの段数（クリティカルパスの長さ）が基準の一つとなる．以下では，ランデブ制御部において1回のマルチランデブの実行条件を判定し，各EFSMで実行する遷移を決定するのに必要な論理ゲートの段数を試算し，その実行効率を評価する．

マルチランデブ制御部を構成する論理ゲートの段数は，(1) EFSMのグローバル状態（状態組）の識別に必要なビット数 n，(2) 同期イベント組の総数 N，(3) ある状態組で同時に実行可能な同期イベント組の最大数 k，(4) 同期するEFSMの数の最大 m，(5) あるイベントが同期するイベントの組合せの数の最大 c，(6) 各EFSMのある状態から実行可能なローカルイベントの最大数 l，(7) 各イベントのガード式を実現する回路の論理ゲートの最大段数 gに依存する．

ガード計算部では，ある同期イベント組 $e=(e_1,\cdots,e_n)$ の実行条件の評価に，各イベント e_iのガード式 $guard(e_i)$ の計算やどのレジスタの値を用いるのかを選択する回路，および e の全イベントのガード式の論理積の計算のための回路が必要である．また，現状態組における同期イベント組の実行条件の真偽値のみを抽出する回路が必要であり，この回路には，各状態組を特定するための回路に最大 $log\ n$ 段，同期イベント組が存在する全ての状態組を識別するのに最大 $log\ N$段必要である．よって，一般のand, or回路を用いた場合，$g+log\ m+log\ c+log\ n+log\ N$が，ガード計算部の最大段数となる．

同期イベント組選択部においては，EFSMの各状態組の特定に $log\ n$ 段，および同期イベント組を持つ全ての状態組の識別に，最大 $log\ N$段必要である．また，ガード計算部の出力である現グローバル状態における同期イベント組の実行条件のうち一つを選択する回路に $log\ k$段+数段が必要である．ただし，状態組を特定する回路および，実行条件の評価結果は論理積で結合されるため，多いほうの段数 $log\ n+log\ N$段で抑えられる．

ローカルイベント選択部では，各ローカルイベントについてガード式の評価を行う回路，および全ローカルイベントから，実行するイベントの決定を行う回路に要する段数が必要で，最大 $g+log\ l$となる．

以上より，マルチランデブ制御部の段数は，最大で $2g+log\ l+log\ m+log\ c+2(log\ N+log\ n)$ で抑えられる．

ファイル転送プロトコルの典型であるAbracadabra ProtocolのLOTOS仕様 [2] の主要部分に対して，本手法を適用してみた．

Abracadabra ProtocolのLOTOS仕様の場合，EFSM数が5，各EFSMの状態数が数個～数十個程度（6bit以内で表せる）であるため，状態組を表すのに必要なビット数 n は約30bitとなる．また，同期イベント組の総数 N は100程度，同時に実行可能な同期イベント組の最大数 k およびローカルイベントの最大数 lはともに数個程である．また，各イベントのガード式の計算に必要な最大段数 g は2程度，あるイベントが他と同期する組合せの最大数は10程度，同期するEFSMの最大数は2である．

以上より，Abracadabra Protocolでは，4入力のand, or回路を用いた場合，ガード計算部に10段程度，同期イベント組選択部に7段程度，ローカルイベント選択部に4段程度必要となり，論理圧縮などを用いない場合マルチランデブ制御部の構成に最大20段程度の論理ゲートが必要となることがわかった．実際には，ハードウェア合成の研究分野で広く用いられている論理圧縮等の手法を用いることによって，実用上十分高速な十数段程度の論理ゲートで実現可能であると思われる．

圧縮後の段数やゲート数などの詳細については，現在調査中である．

4 あとがき

本稿では，あるクラスのLOTOS仕様から，マルチランデブにより通信を行うEFSM群を自動生成する方法および，EFSM群の状態組などから同期するEFSMの組合せおよび同期実行するイベントを計算するマルチランデブ制御部の構成およびその自動合成手法について提案した．本手法と我々が文献 [3] で提案した手法を組み合わせて用いることにより，マルチランデブを含む構造的なLOTOS仕様からハードウェア回路を合成することが可能になる．

前節の実験により，Abracadabra Protocolなどの実用規模のプロトコルに対して本手法を適用し，効率の良い回路を合成できることが確認された．現在，LOTOS仕様を解析しマルチランデブ制御部の構成に必要な情報を自動的に取得するシステムを作成中である．今後，レジスタ転送レベル (RTL) のEFSM仕様を自動生成するよう拡張し，得られたRTL仕様から既存のハードウェア合成系を用いて実際にハードウェア回路を合成し，評価実験を行う予定である．

参考文献

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

[2] ISO/IEC/TR 10167: Information Technology – Open Systems Interconnection – Guidelines for the application of Estelle, LOTOS and SDL (1991).

[3] Higashino, T., Yasumoto, K., Kitamichi, J. and Taniguchi, K. :Hardware Synthesis from a Restricted Class of LOTOS Expressions, Proc. of 14th IFIP Int. Symp. on Protocol Specification, Testing, and Verification (PSTV-XIV), pp. 355 – 362 (1994).

[4] Kloos, C. D., Moro, T. M., Filho, G. R. and Lopez, A. M.: VHDL generation from a timed extension of the formal description technique LOTOS within the FORMAT project, *Microprocessing and Microprogramming 38*, pp. 589 – 596 (1993).

[5] Leue, S. and Oechslin, P.: OpParIm: a method and tool for optimized parallel protocol implementation, *J. High Speed Networks 5*, pp. 125 – 143 (1996).

[6] Wytrebowicz, J.: Hardware Specification Generated from Estelle, *Proc. of 15th IFIP Int. Symp. on Protocol Specification, Testing and Verification (PSTV-XV)*, pp. 435 – 450 (1996).