情報処理学会論文誌 Vol. 50 No. 11 2633–2642 (Nov. 2009)

プロセス代数に基づくシステムレベル仕様合成

岩 政 幹 人[†1] 日 比 野 靖[†1]

LSI の設計フローにおいて，UML 等の上流工程の仕様書記述と，実現する LSI の設計書との間には，ギャップが存在する．特にシナリオ記述に基づく仕様書は動作の部分的な側面しか記述していない．このような断片的な仕様書から対象システムの全容を構成する手段が不足している．本論文では，仕様書として，Message Sequence Chart（MSC）で，動作シナリオ記述し，その記述をプロセス代数に基づきモデル化することにより，プロセス代数の計算規則により，複数の動作シナリオを機械的にマージし，全体のシステム仕様を生成する手法を与える．並行プロセスの仕様記述を与える形式的手法であるプロセス代数を用い，本手法のために，プロトコル（個別の動作シナリオ）の並行結合を行う計算法を示した．

System-level Specification Synthesis by Process Algebra

Mikito Iwamasa[†1] and Yasushi Hibino[†1]

The system level specification engineering is expected to be a useful vehicle for achievement of the automatic SOC synthesis. There still exists a gap between a higher level specification and detail level one. For example, a UML sequence diagram only captures one aspect of system behavior. A synthesizing method of the whole system specification from the system specification fragments is to be sought. The authors show a framework for system level specification synthesis using process algebra as a formal basis to synthesize the whole system specification from message sequence charts as system behavior scenarios by extending operators in a CCS-like process algebra.

†1 北陸先端科学技術大学院大学情報科学研究科
School of Information Science, Japan Advanced Institute of Science and Technologies

1. はじめに

ESL（Electric System Level）設計における上流工程からの設計自動化の目標の 1 つとして，システムレベルの仕様書からシステムを自動的に生成することがあげられている．たとえば UML（Unified Modeling Language）では要求分析から仕様書の策定までを行うための仕様記述形式を提供している．しかし，シーケンス図等による振舞い仕様記述はシステムの動作の一側面しか規定しないのでシステム全体としての振舞いそのものを記述できない等のギャップがあった[7]．

これらの課題に対して，動作の仕様記述としてメッセージシーケンス図（MSC: Message Sequence Chart）を採用して，MSC で記述された仕様書のみから，最終システムを合成してシステム設計を行う手法（eMSC システムと呼ぶ）が開発された[6]．eMSC システムでは MSC に階層性を導入し，コマンド図と呼ぶ下位の MSC で詳細なプロトコルを記述し，シナリオ図と呼ぶ上位の MSC によってシステムとしての振舞いを記述する．シナリオ合成機能によりシナリオを互いに連携しながら並列に動作する状態遷移機械の集合として合成し，シナリオマージ機能で複数のシナリオから全体のシステム仕様を合成する．

eMSC システムでは，シナリオ合成機能，シナリオマージ機能の実現に個別対応している部分があり，一般化が十分でなかった．これは形式化を行っていないことにその原因がある．

本論文では，プロセス代数に基づいたプロセスの並行結合法を導入し，シナリオ合成やシナリオマージを形式的に取り扱う手法を与え，実行順序制約の充足性を満たすという意味で正しい合成が保証されることを示す．

2. eMSC システム[6]

eMSC システムは，プロトコル通信や演算処理を MSC 形式で表現したコマンド図を「部品」として，コマンド図を組み合わせてシステム動作の一局面を表現するシナリオ図，複数のシナリオ図をマージしたマージドシナリオから構成される．

2.1 コマンド図

コマンドは，状態遷移機械の仕様を規定する．コマンド図には，1 つの状態遷移機械の仕様を規定するプロセス内コマンドと，2 つの状態遷移機械とそれらの間の通信仕様を規定するプロセス間コマンドがある．

状態遷移機械間では，メッセージ通信によって同期をとる．

図 1 は，プロセス間コマンドの例である．コマンドの処理内容は，活性体（Activity：図

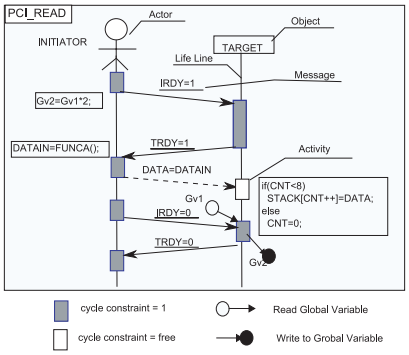

図 1　コマンド図
Fig. 1　Command chart.

では矩形で示す）に記述する．

活性体はコマンドオブジェクト（Object）に沿って配置され，活性体の間にはサイクル境界（cycle boundary）がある．コマンドオブジェクトが状態遷移機械に対応し，活性体が状態に対応する．活性体の実行はコマンドの最上部からスタートし下方に向かって最下部に達したら再び最上部に戻って動作を繰り返す．

2.2　シナリオ図

シナリオは，MSC 形式で表現した動作順序に従ってコマンドが動作する仕様を規程する．シナリオ図はコマンド図を下位部品とし，シナリオオブジェクト配下に展開した上位階層の MSC として表現される．プロセス内コマンドは，1 つのコマンド活性体（図では矩形で表現）として，プロセス間コマンドは，2 つのコマンド活性体を矢印で接続したものとして表される．コマンドは状態遷移機械の動作仕様を規定するが，シナリオ図は状態遷移機械どうしを所定の順序で動作するように連係させる仕様を規定している．

個々のコマンド活性体は並列に動作する状態遷移機械であり，シナリオオブジェクト（ScenarioObject）は複数プロセスを束ねる中間階層に相当する．**図 2** はシナリオ図の例である．

シナリオ図が規定するコマンド活性体間の実行順序制約には 2 種類ある．同一のシナリオオブジェクト上に配置されるコマンド活性体は，配置の上下方向に沿って実行順序制約があり（縦チェーンと呼ぶ），プロセス間コマンドの送信・受信側のコマンド活性体間には，同期して動作する実行制約があるものとする（横チェーンと呼ぶ）．**図 3** はコマンドとシナリオの階層関係を表している．コマンド図は，活性体 A，B，C がそれぞれ $a \leftrightarrows b$，$c \leftrightarrows d$，$e \leftrightarrows f$ の動作を繰り返す独立の状態遷移機械であり，シナリオ図は A，B 間に横チェーン，B，C 間に縦チェーンの制約を課していることを示している．

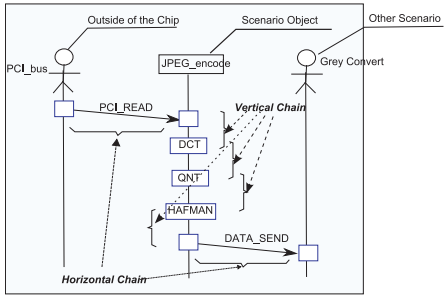

図 2　シナリオ図
Fig. 2　Scenario chart.

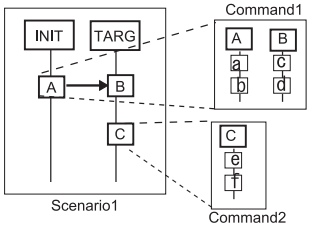

図 3　シナリオとコマンドの階層
Fig. 3　Chart hierarchies.

縦・横チェーンの制約は以下のとおりである，

横チェーン制約　A（B）の $n+1$ 巡目の実行開始は B（A）の n 巡目の実行終了を追い越さない．

縦チェーン制約　B の n 巡目の実行終了後に C の n 巡目の実行が開始され，B の $n+1$ 巡目の実行終了は C の n 巡目の終了を追い越さない．

2.3　シナリオ合成と RTL 出力

シナリオ合成とは，シナリオ図におけるコマンド活性体をコマンド図が規定する個別の状態遷移機械として生成し，シナリオ図が規定する接続に合わせて，個々の状態遷移機械間に通

信チャネルを，個々の状態遷移機械に連携プロトコル（チャネルへの読み書き）を挿入してシナリオ図が規定する仕様に沿った動作を行う通信状態遷移機械（CFSMs: Communicating FSMs）を生成する処理である．

eMSC の連携プロトコルでは，バッファ長が 1 の FIFO チャネルを連携ごとに用意し，このチャネルへの書き込み（write），読み込み（read）アクションを元々の FSM に挿入することにより，action 間の実行順序制御を行っている．FIFO バッファが full である場合に write を抑制し，バッファが空である場合に read を抑制する blocking 型，それぞれの場合に上書き，空読みする non-blocking 型のアクションを用意している．可能な連係パターンは $2 \times 2 = 4$ 種類の組合せがあり，たとえば blocking 型の read と non-blocking 型の write を組み合わせる連係を br-nbw 型の連係と呼ぶことにする．

コマンド活性体 X の n 巡目の繰返し実行の最初の action を $st_n(X)$，最後の action を $en_n(X)$ と表すと，eMSC のシナリオ合成において A，B，C 3 つの状態遷移機械の間に挿入される連携プロトコルは以下である．

連携プロトコル（横） $st_n(A)$ から $st_n(B)$ へ，および $en_n(B)$ から $en_n(A)$ への br-bw 型の連携プロトコル挿入

連携プロトコル（縦） $en_n(B)$ から $st_n(C)$ へ，および $en_n(C)$ から $en_{n+1}(B)$ への br-bw 型の連携プロトコル挿入

シナリオ合成の結果得られた CFSMs は LSI システム全体の仕様として，いったん LSI 設計用の専用記述言語にて出力されてから高位合成ツールを利用して RTL 記述の生成が行われる．

3. プロセス代数による eMSC の形式化

本論文では，eMSC を並行実行するラベル付き遷移システム（LTS）として形式化する．形式化においてはプロセス代数[1)–3),5)]（主に CCS と ACP の一部を採用）をベースとする．

3.1 ラベル付き遷移システム（LTS）

ラベル付き遷移システム（LTS）は以下のように構成される．

$$(Proc, Act, \{\xrightarrow{\alpha} | \alpha \in Act\})$$

ここで $Proc$ はプロセスの集合，Act はアクションの集合，$\{\xrightarrow{\alpha}\}$ はアクション $\alpha \in Act$ にともなう遷移関係の集合であるとする．

A はチャネル名（name）の集合で $\overline{A}$ を補名（co-name）の集合であるとする．$\overline{\overline{a}} = a$ である．

$$\frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \; K \stackrel{\text{def}}{=} P \qquad \frac{}{\alpha \cdot P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} Q' \mid Q}$$

$$\frac{P \xrightarrow{\alpha} P'}{\partial_L(P) \xrightarrow{\alpha} \partial_L(P')} \; \alpha, \overline{\alpha} \notin L \qquad \frac{P \xrightarrow{\alpha} P' \; Q \xrightarrow{\overline{\alpha}} Q'}{\partial_L(P \mid Q) \xrightarrow{\alpha|\overline{\alpha}} \partial_L(P' \mid Q')} \; \alpha, \overline{\alpha} \in L$$

$$\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

図 4　LTS の展開ルール
Fig. 4　Rules for LTS.

$$\overline{A} = \{\overline{a} | a \in A\}$$

ラベル L はチャネル名と補名の和集合である．

$$L = A \cup \overline{A}$$

アクションは，ラベルと観測不能な内部アクション τ で構成される．

$$Act = L \cup \{\tau\}$$

また 0 は特別なプロセスで，それ以上遷移がない終端プロセスであるとする．

プロセス式は以下の演算要素によって構成される．

$$P, Q := K \mid \alpha \cdot P \mid P \mid Q \mid P + Q \mid \partial_L(P) \mid P[f]$$

ここで K はプロセス名を表す定数である．演算要素の操作的な意味を**図 4** のように定義する．ここで，$\cdot$ は逐次実行結合，$\mid$ は並行実行結合，$+$ は選択実行結合演算であり，さらに，

(1)　プロセス定義式は $K \stackrel{def}{=} P$ としてプロセスを（再帰的に）定義する記法である，

(2)　∂_L 演算は，name と co-name アクションの同時実行（$\alpha \mid \overline{\alpha}$ ここで $\alpha \in L$）のみに動作を制限する演算であり，単独での α，$\overline{\alpha} \in L$ の実行を禁止する，

(3)　rename 演算（$P[f]$）は P における α 遷移を $f(\alpha)$ 遷移に書き換える．以降便宜的に $f = \alpha/\beta$ により α を β に置き換える関数を表すものとする．

CCS における通信をともなう同期遷移（ランデブー）は ∂ 演算と rename 演算を組み合わせたものとして表現できる．

$$\frac{P \xrightarrow{\alpha} P' \; Q \xrightarrow{\overline{\alpha}} Q'}{\partial_{\{\alpha\}}(P \mid Q)[(\alpha \mid \overline{\alpha})/\tau] \xrightarrow{\tau} \partial_{\{\alpha\}}(P' \mid Q')[(\alpha \mid \overline{\alpha})/\tau]}$$

3.2 コマンドの形式化

図 3 におけるコマンド活性体 A，B，C は，以下のプロセス定義式の集合として形式化される．

$$\{A \stackrel{def}{=} a \cdot b \cdot A,\ B \stackrel{def}{=} c \cdot d \cdot B,\ C \stackrel{def}{=} e \cdot f \cdot C\}$$

3.3 LTS の並行結合演算

LTS どうしの並行結合演算を以下のように定義する．

$LTS3$	$:= COM(LTS1, LTS2)$
$Proc_{LTS3}$	$:= Proc_{LTS1} \times Proc_{LTS2}$
Act_{LTS3}	$:= Act_{LTS1} \cup Act_{LTS2}$
S	$:= Act_{LTS1} \cap Act_{LTS2}$
$\stackrel{\alpha}{\rightarrow}$	$:= \partial_S(P \mid \overline{Q})[(\alpha \mid \overline{\alpha})/\alpha, \alpha \in S]$ の遷移関係に従う

LTS3 の初期プロセスは，“LTS1 の初期プロセス × LTS2 の初期プロセス” であるとする．また $P \in$ LTS1，$Q \in$ LTS2 であり，$\overline{Q}$ は LTS2 に含まれる $\beta \in Act_{LTS1} \cap Act_{LTS2}$ を co-name である $\overline{\beta}$ に置き換えたプロセスを指すものとする．

プロセス定義 $E = \{X \stackrel{def}{=} a \cdot b \cdot X, Y \stackrel{def}{=} c \cdot d \cdot Y,\}$ に対応する LTS を，LTS1 $= \langle Proc1, Act1, \stackrel{\alpha}{\rightarrow}\rangle$ および LTS2 $= \langle Proc2, Act2, \stackrel{\alpha}{\rightarrow}\rangle$ とするとき，LTS1 と LTS2 の並行結合演算を行い，結合後のプロセスを `M0=(X|Y)`，`M1=(b.X|Y)`，`M2=(X|d.Y)`，`M3=(b.X|d.Y)` とおくと，以下の初期プロセスが `M0` の LTS3 を得る．

```
LTS3:
Proc={M0,M1,M2,M3},
Act={a,b,c,d}
-a->={(M0,M1),(M2,M3)},-c->={(M0,M2),(M1,M3)}
-b->={(M1,M0),(M3,M2)},-d->={(M2,M0),(M3,M1)}
```

eMSC におけるコマンドは，定義式 $P \stackrel{def}{=} T \cdot P$（ここで T は $a \in Act, \cdot, +$ により構成されるプロセス式）の形式で再帰的に定義されるプロセス P として形式化でき，$P_i \stackrel{def}{=} T \cdot P_{i+1}$ として i 巡目の実行を区別することができる．

繰返し実行の n 巡目を区別する場合の LTS の並行結合演算は $\alpha \in Act_{LTS1}$，$\beta \in Act_{LTS2}$ をそれぞれ LTS1，LTS2 の初期プロセスからの最初のアクションであるとすると，$ocr(\alpha)$，$ocr(\beta)$ を動作の巡目を表す関数として，$\langle R, ocr(\alpha) - ocr(\beta)\rangle$（$R$ は n 巡目を区別しない並行結合 $P \mid Q$）で表せるプロセスを持つ LTS として計算できる．

LTS1 と LTS2 の並行結合結果は，アクション a，c の各々k，j 巡目を $ocr(a) = k$，$ocr(c) = j$ と表して，以下の無限の LTS である LTS4 を得る．ここで初期プロセスは $ocr(a) = 0$，$ocr(c) = 0$ より $\langle M0, 0\rangle$ である．

```
LTS4:
Proc={<M0,i>,<M1,i>,<M2,i>,<M3,i>}
Act={a,b,c,d}
-a->={(<M0,i>,<M1,i+1>),(<M2,i>,<M3,i+1>)}
-b->={(<M1,i>,<M0,i>),(<M3,i>,<M2,i>)}
-c->={(<M0,i>,<M2,i-1>),(<M1,i>,<M3,i-1>)}
-d->={(<M2,i>,<M0,i>),(<M3,i>,<M1,i>)}
```

3.4 LTS 上の順序制約

LTS 上の順序制約 $\psi^n_{\alpha_i \rightarrow \beta_j}$ は，アクション α，$\beta \in Act_{LTS}$ の間の実行順番に関して，“α の i 巡目は β の j 巡目より先行し，α の β に対する先行は n を超えない” という性質を表すものとする．

$P \in Proc_{LTS}$ を順序制約とは関係ない，元々のプロセス名であるとし，`exocr(x)` を x の先行度合いを表す整数であるとする．

$\psi^n_{a_i \rightarrow b_j}$ では `exocr(a)=ocr(a)-(ocr(b)-(j-i))` になる．先行度合いを考慮したプロセスは `<P,exocr(a)>` と表すこととすると，制約の意味を**図 5** の遷移ルールで定義することができる．

`exocr(x)` に着目して状態を抽象化するとルールは LTS として抽象化できる．

たとえば $\psi^1_{a_i \rightarrow c_i}$ は，`exocr(a)=ocr(a)-ocr(c)` を状態変数とし，プロセスを “`P` 状態変数” と記述すると以下の初期プロセスが `P0` の LTS5 として表現できる．NG プロセスは遷移ルールの δ に対応し，順序制約が充足しない場合を表している．

$$\frac{P \xrightarrow{\alpha} P'}{\langle P,k\rangle \xrightarrow{\alpha} \langle P',k+1\rangle}\ k<n \qquad \frac{Q \xrightarrow{\beta} Q'}{\langle Q,k\rangle \xrightarrow{\beta} \langle Q',k-1\rangle}\ k>0$$

$$\frac{P \xrightarrow{\alpha} P'}{\langle P,n\rangle \xrightarrow{\alpha} \delta} \qquad \frac{Q \xrightarrow{\beta} Q'}{\langle Q,0\rangle \xrightarrow{\beta} \delta}$$

図 5　順序制約の展開ルール
Fig. 5　Rules for order constraints.

```
LTS5:
Proc = {P0,P1,NG}
Act = {a,c}
-a->{(P0,P1),(P1,NG)},-c->{(P1,P0),(P0,NG)}
```

同様に i 巡目の d の実行が $i+1$ 巡目の a の実行に最大 1 巡先行する場合の制約 $\psi^1_{d_i \to a_{i+1}}$ は，`exocr(d)=ocr(d)-(ocr(a)-1)` としてこれを状態変数とすると，以下の LTS6 の初期プロセス $P1$ として表現できる．

```
LTS6:
Proc={P0,P1,NG}
Act ={a,d}
-a->{(P1,P0),(P0,NG)},-d->{(P1,NG),(P0,P1)}
```

3.5　順序制約の充足性

プロセス M と順序制約 ψ の間の充足性を定義する．M に対応する LTS を LTS_M，ψ に対応する LTS を LTS_ψ とし，$Act_{LTS_\psi} \subseteq Act_{LTS_M}$ とする．

3.5.1　充足性の定義と充足ゲーム

定義 3.1（充足性）　プロセス M が ψ を充足するとは，$\mathrm{COM}(LTS_M, LTS_\psi)$ の動作が初期プロセスから NG プロセス（遷移先が δ である場合を NG プロセスとする）に到達しないことである．

Player1 を，アクション $Act_{LTS_M} - Act_{LTS_\psi}$，Player2 を Act_{LTS_ψ} を実行できる player であるとすると，以下のような充足 Game として，充足性の判定計算を行うことができる[4]．

定義 3.2（充足 Game）　モデル M（Player1 とする）を記述する LTS である LTS_M と，順序制約 ψ に対応するプロセス P_ψ（Player2 とする）を記述する LTS である LTS_ψ が与えられたとき，LTS の並行結合結果である $\mathrm{COM}(LTS_M, LTS_\psi)$ に対して，Player2 が NG プロセスに遷移させようと，Player1 がそれを阻止しようとして互いに自分のアクションを実行する Game 問題においては，Player1 が Player2 に対して，常勝する場合に限り，M は ψ を充足する．

最初に，Game 構造（Game Structure）を導入する．Game 構造（Game Structure）は以下の 4 要素構造で構成される，

定義 3.3（Game 構造：Game Structure）

$$G = \langle S, M, \Gamma_1, \Gamma_2, \delta \rangle$$

S は状態の集合，M は Move の集合，Move 割当て（$\Gamma_i, i=1,2$）は状態 $s \in S$ に対して，Player i の可能な Move の集合 $\Gamma_i(s) \subseteq M$ を割り当てる関数，遷移 $\delta : S \times M \times M \mapsto S$ は，状態 s に対して Player1，Player2 の Move の後，次状態に達する関数により定義される．また G は Turn-based であるとする．すなわち互いに素な Player1 のみの手番の状態の部分集合 S_1 と Player2 の手番の部分集合 S_2 で S が構成される（$S = S_1 \cup S_2$）ものとする．

3.5.2　勝利集合と充足性計算の手順

Player1 が Player2 に対して常勝であることを示すためには，まず Player1 が Player2 に対して universal な勝利戦略を持つことを示し（空集合でない勝利集合が求まるということ），Player1 の手番となる状態すべてが勝利集合に含まれることを示す．

ここで universal な勝利戦略を持つとは，Player2 のいかなる手に対しても Player1 が goal の範囲に遷移先が含まれるような，選択手がつねに存在することを指す．

universal な勝利戦略を求める問題は safety-game と呼ばれ，Player1 の勝利集合 Win_1 を求めることに帰着される．ここで勝利集合 Win_1 とは Win_1 に含まれる状態 $s \in Win_1$ からスタートすれば，つねに Player1 が勝てる必勝戦略 π_1 が存在するような状態の集合である．

X を状態の集合であるとして，Player1 に対する $Pre_1(X)$：conditional predecessor を以下のように定義する．

定義 3.4（$Pre_1(X)$：Conditional Predecessor）
$Pre_1(X)$ は conditional predecessor と呼ばれる集合で，Player1 が 1 回の turn で game の状態が X に含まれるように制御できるような前状態 s の集合を表す．

$Pre_1(X) = \{s \in S | \exists a \in \Gamma_1(s). \forall b \in \Gamma_2(s). \delta(s,a,b) \subseteq X\}$

勝利集合 Win_1 とは，$Pre_1(X)$ を施した結果の集合も，もとの集合 X のうちにとどまる状態の集合，すなわち Win_1 は $X = Pre_1(X)$ となる方程式の解になる．

すなわち，勝利集合とは，S の部分モデル S' で，S' の conditional predecessor が必ず S' に含まれるような，部分モデル S' である．

$Pre_1(X)$ が単調な関数であるので，Tarski の不動点定理[8] より，最大不動点，最小不動点が存在し，適切な初期集合 X_0 を選択して，$Pre_1(X)$ を繰り返し適用して，収束先を求めることにより，不動点を手続き的に計算できることが，不動点理論より知られている．ここで R とは NG でない状態の集合 $S - NG$ である．

$Win_1 = \nu X.(R \cap Pre_1(X))$

最大不動点の場合は，$X_0 = S$ として $Pre_1(X)$ を繰り返し適用し，収束する状態の集合が最大不動点である．

$$
\begin{aligned}
X_0 &= S \\
X_1 &= R \cap Pre_1(X_0) \\
\cdots &= \cdots \\
X_n &= R \cap Pre_1(X_{n-1}) \\
\cdots &= \cdots \\
Win_1 &= lim_{N \to n}(X_N), X_n = X_{n-1}
\end{aligned}
$$

最大不動点が，Player1 の勝利集合 $X = Pre_1(X)$ のうち最大の解になる．

Player1 の勝利集合と Player1 の手番となる状態 $S1$ との交わり $Win_1 \cap S1$ が $S1$ と同じであるときに，Player1 は手番でつねに勝利戦略をたてることができるので，常勝である．Player1 が常勝であるときに，モデル M は順序制約 ψ を充足するという．

充足性の計算は $Pre_1(X)$ を適用するに従って状態の数が単調減少するので，最大 $|S|$ ステップで収束する．

$\mathrm{COM}(LTS_M, LTS_\psi)$ に対する充足 Game においては，並行結合演算結果 $\mathrm{COM}(LTS_M, LTS_\psi)$ において初期プロセスから NG プロセスへ到達可能でない場合に，M に対して ψ が充足可能であることになる．充足可能でない場合は，Player1 側の勝利戦略を計算することにより，プロパティ ψ を充足する最大の M の部分モデル M' を計算できる．部分モデル M' とは元々の M に対して ψ を充足する範囲で動作を制限したモデルになる．

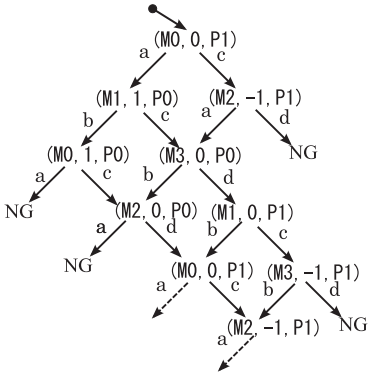

図 6　充足性の計算 1
Fig. 6　Calculating satisfiability.

3.6　∇ 演　算

$\psi^n_{\alpha_i \to \beta_j}$ に対応する LTS（$\mathrm{LTS}_{\psi^n_{\alpha_i \to \beta_j}}$ とする）において，NG への遷移を削除した LTS を $\mathrm{LTS'}_{\psi^n_{\alpha_i \to \beta_j}}$ とするとき，

任意の LTS に対して，$\mathrm{COM}(\mathrm{LTS}, \mathrm{LTS1'}_{\psi^n_{\alpha_i \to \beta_j}})$ によって定義される LTS を得る演算を ∇ 演算と定義する．以下順序制約 $\psi^n_{\alpha_i \to \beta_j}$ に対応する ∇ 演算を $\nabla_{\psi^n_{\alpha_i \to \beta_j}}$ と表す．

$$\nabla_{\psi^n_{\alpha_i \to \beta_j}} : Proc \times int \times Act \times int \times Act \times int \mapsto Proc$$

$\nabla_{\psi^n_{\alpha_i \to \beta_j}}(P)$ は順序制約 $\psi^n_{\alpha_i \to \beta_j}$ に従い，P を起点とする LTS の中で $\psi^n_{\alpha_i \to \beta_j}$ に矛盾する遷移を抑制する意味を持つ．

3.7　∇ 演算と充足性

充足性の定義および計算より，プロセス M に対する $\nabla_{\psi^n_{\alpha_i \to \beta_j}}$ 演算を施した結果の LTS は NG プロセスへの遷移を含まないので $\psi^n_{\alpha_i \to \beta_j}$ を充足することが自明である．さらにこの LTS は $\psi^n_{\alpha_i \to \beta_j}$ を充足する LTS_M の部分モデルの最大である．

たとえば，LTS4 に対して，順序制約 $\psi^1_{d_i \to a_{i+1}}$ の充足性を計算する．

$\mathrm{COM}(LTS4, LTS_{\psi^1_{d_i \to a_{i+1}}})$ を図示したものが**図 6** で，($\langle$M0, 0$\rangle$, P1) を `(M0,0,P1)` と表している．NG プロセスへの遷移があるので，LTS4 は $\psi^1_{d_i \to a_{i+1}}$ を充足しない．

一方 $\nabla_{\psi^1_{d_i \to a_{i+1}}}$ を LTS4 に適用すると図 6 における NG への遷移が抑制された LTS を

$$\frac{P \overset{\alpha\cdot bw(k)}{\longrightarrow} P'}{\langle P \mid Q, cnt(k)\rangle \overset{\alpha}{\longrightarrow} \langle P' \mid Q, cnt(k)+1\rangle} \quad cnt(k) < n$$

$$\frac{Q \overset{br(k)\cdot\beta}{\longrightarrow} Q'}{\langle P \mid Q, cnt(k)\rangle \overset{\beta}{\longrightarrow} \langle P \mid Q', cnt(k)-1\rangle} \quad cnt(k) > 0$$

$$\frac{P \mid Q \overset{\alpha\cdot bw(k)}{\to} P'}{\langle P \mid Q, cnt(k)=n\rangle \overset{\tau}{\to} \delta} \qquad \frac{P \mid Q \overset{br(k)\cdot\beta}{\to} Q'}{\langle P \mid Q, cnt(k)=0\rangle \overset{\tau}{\to} \delta}$$

図 7　連係プロトコルの展開ルール
Fig. 7　Rules for synchronization protocols.

得る．得られた LTS は，$\psi^1_{d_i \to a_{i+1}}$ を充足し，かつ $\psi^1_{d_i \to a_{i+1}}$ を充足する最大の LTS4 の部分モデルである．

3.8　連携プロトコルの形式化

eMSC ではコマンド起因の FSM 間に，シナリオが規定する順序制約に従うように，FIFO チャネルを介した連携プロトコルを，挿入して実現していた．

P から Q へ深さ $= n$ の FIFO チャネル k を介して通信を行う場合の動作意味を定義する．$bw(k)$ は blocking 型の書き込み，$br(k)$ は blocking 型の読み込み，であるとする．

br-bw 型の連係プロトコル挿入後の動作の意味を，**図 7** のルールで定義することができる．FIFO のバッファの初期値が $cnt(k) = i - j$ であるとすれば，図 7 のルールと $\nabla^n_{\psi_{\alpha_i \to \beta_j}}$ のルール（図 5）は同等である．

すなわち，上記制限の範囲内では，$\nabla^n_{\psi_{\alpha_i \to \beta_j}}$ 演算を適用して LTS を 1 つに合併しても，FIFO チャネルを介した br-bw 型の連携プロトコルを挿入しても，ルールが等しいので展開される遷移関係が等くなり結果として得られる LTS は互いに双模倣関係にあり，等価である．

4. シナリオ合成の正当性

4.1　シナリオの順序制約の形式化

シナリオは状態遷移機械間でデータの受け渡しが正しく行われることを強要する．ここでは，正しく受け渡されるとは，n 巡目の実行のデータが $n+1$ 巡目の実行のデータにより上書きされないことであるとする．そこで状態遷移機械どうしの連携動作に制約を設けることによりデータが正しく受け渡される範囲で，正しく n 巡目の動作が実行されることがシナリオの制約である．

横チェーンは，2 つの状態遷移機械の間で n 巡目の一連の通信手順（プロトコル）を連動させることにより，データを伝えることを保証する．縦チェーンは，2 つの状態遷移機械の片方（A とする）の n 番目の実行が終了するときにもう片方（B とする）の n 番目の実行を開始し，A の $n+1$ 番目の実行が B の n 巡目の実行を超えないことによりデータをリレー式に伝えることを保証する．

4.2　横チェーン・縦チェーンの形式化

横チェーン（ψ_h）は，並列動作する状態遷移機械どうしは n 巡目の動作を連動させながら，互いにデータをやりとりする．片方（A とする）が n 巡目のデータのやりとり中にもう片方（B とする）が $m = n+1$ 回目のデータのやりとりが開始されないことが順序制約になる．$st_n(X)$ を X の n 巡目の開始，$en_n(X)$ を n 巡目の終了アクションであるとすると，「先行する」関係により

横チェーン：　$en_n(X)$ は $st_{n+1}(Y)$ に先行∧
　　$en_n(Y)$ は $st_{n+1}(X)$ に先行する

という制約であると定義できる．

$X = a \cdot bX$，$Y = c \cdot dY$ であるとし，「先行する」を順序制約で読み替えると，横チェーンの順序制約は $\psi^1_{d_n \to a_{n+1}} \wedge \psi^1_{b_n \to c_{n+1}}$ になる．

一方，縦チェーン（ψ_v）は，並列動作する状態遷移機械どうしは，前側（A とする）の n 回目の終了時に，n 巡目のデータの後側（B とする）とのやりとりが行われるから，このとき B が $n-1$ 巡目の動作を終了していることが順序制約になり，横チェーンと同様に，以下の制約として記述できる．

縦チェーン：　$en_n(X)$ は $st_n(Y)$ に先行し∧
　　$en_n(Y)$ は $en_{n+1}(X)$ に先行する

横チェーンと同様に，縦チェーンの順序制約は $\psi^1_{b_n \to c_n} \wedge \psi^1_{d_n \to b_{n+1}}$ になる．

4.3　横チェーンの検証

eMSC では横チェーンを実現するために，FIFO チャネルを介する連携プロトコルを挿入していた，このプロトコルは，等価な ∇ 演算として $\nabla_{\psi^1_{a_i \to c_i}} \times \nabla_{\psi^1_{d_i \to b_i}}$ と表せ，以下の 2 つの LTS（それぞれ初期プロセスは P0，Q0）で表わすことができる．

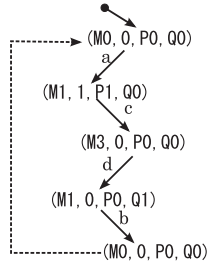

図 8　連携プロトコル挿入後の横チェーン
Fig. 8　Horizontal chain after inserting protocols.

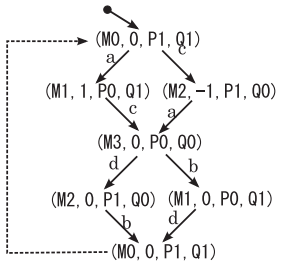

図 9　横チェーン制約を充足する最大の部分モデル
Fig. 9　Maximal model for horizontal chain constraints.

```
LTS7:           LTS8
Proc={P0,P1},   Proc={Q0,Q1}
Act ={a,c},     Act ={b,d}
-a->{(P0,P1)},   -d->{(Q0,Q1)},
-c->{(P1,P0)}    -b->{(Q1,Q0)}
```

LTS4 の初期プロセス $= \langle M0, 0\rangle$ に対して ∇ 演算を適用すると，**図 8** の LTS を得る．

図 8 の LTS では $a \cdot c \cdot d \cdot b$ の繰返しの実行順のみが許される．

一方，横チェーンの制約 $\psi_h := \psi^1_{d_n \to a_{n+1}} \wedge \psi^1_{b_n \to c_{n+1}}$ は，以下の 2 つの LTS（それぞれ初期プロセスは P1，Q1）で表すことができる．

```
LTS9:                 LTS10:
Proc={P0,P1,NG},      Proc={Q0,Q1,NG}
Act ={a,d},           Act ={c,b}
-a->{(P1,P0),(P0,NG)},-c->{(Q1,M0),(Q0,NG)},
-d->{(P0,P1),(P1,NG)} -b->{(Q0,M1),(Q1,NG)}
```

図 8 の LTS に対して LTS9 および LTS10 の充足性を計算すると，NG プロセスに遷移しないので，eMSC における連携プロトコルは横チェーンの制約を充足することが分かる．

前章より，順序制約の LTS を ∇ 演算の LTS に読み替えて，LTS4 のプロセス $= \langle P0, 0\rangle$ に対して並行同期結合することにより，横チェーン制約 ψ_h を充足する最大の解（**図 9**）が得られる．

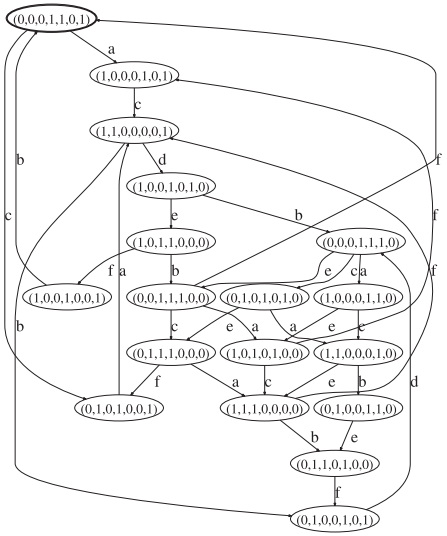

図 10　シナリオ 1 を満たす最大の LTS
Fig. 10　Maximal model for scenario 1.

図 8 と図 9 を比較すると，図 9 の方が許容される動作パターンが多いことが分かる．すなわち eMSC では横チェーンの制約に対して，過大に動作を制約していたことが分かった．

4.4　シナリオ合成の例

シナリオ図は，縦チェーンと横チェーンの組合せであるから，制約は $\psi^n_{x \to y}$ 型の順序制約の論理積として形式化される．

シナリオの制約を充足する，最大モデルの計算は同様に複数の ∇ 演算を組み合わせることにより得られる．

たとえば図 3 のコマンド A，B，C を下層に持つシナリオ 1 は以下のように形式化できる．

コマンド	$\{A \stackrel{def}{=} a \cdot b \cdot A, B \stackrel{def}{=} c \cdot d \cdot B,$ $C \stackrel{def}{=} e \cdot f \cdot C\}$
シナリオ	$\psi^1_{d_n \to a_{n+1}} \wedge \psi^1_{b_n \to c_{n+1}} \wedge \psi^1_{d_n \to e_n} \wedge \psi^1_{f_n \to d_{n+1}}$

3.6 節にあるように ψ 型の制約を ∇ 演算に読み替えてプロセス $(A \mid B \mid C)$ に順次適用すると，性質を満たす最大の部分モデルが得られる，**図 10** は結果の LTS を図示したものである．P_A，P_B，$P_C = \{0,1\}$ を A，B，C 起因の状態，$P1$，$P2$，$P3$，$P4 = \{0,1\}$ を $\psi^1_{d_n \to a_{n+1}}$，$\psi^1_{b_n \to c_{n+1}}$，$\psi^1_{d_n \to e_n}$，$\psi^1_{f_n \to d_{n+1}}$ の状態として $(P_A, P_B, P_C, P1, P2, P3, P4)$ として合成後の状態を表している．また初期状態は `(0,0,0,1,1,0,1)` である．

5. 関 連 研 究

Uchitel らの MTS[9] は，プロセスの部分仕様を MTS（Modal Transition System）を用いて記述し，複数の MTS 部分仕様を 1 つの MTS としてマージすることを行う．MTS は，required，possible，maybe という 3 種のトランジションを持つことにより必須の仕様（required）と可能性のある仕様（possible，maybe）を区別することを特徴としている．マージ対象の複数の MTS に成立するトランジションを比較して可能遷移（maybe）を必須遷移（required）に変更するあるいは不要ならば取り除く修正（refinement）を行い MTS の共通解を得ることにより MTS のマージを定義している．マージ処理は状態遷移機械に対して実行順序の部分的な仕様が複数与えられたときに，仕様をマージしてすべてを充足する単一の状態遷移機械の実体を計算することに相当する．

我々の定式化においては，コマンドの並行結合で得られた最初の状態遷移をシナリオの制約に従って制限していって最終的に得られた LTS がすべての制約を充足する最大の状態遷移系になっている．また並行結合自体も単一の状態遷移機械以外に連係プロトコルを介して違いに連係しながら並行動作する状態遷移機械としても実現できる．

6. まとめと展望

並行実行 LTS でモデル化される対象において，制約として順序制約を選択するとき，これを充足する最大の解を得る演算を明らかにした．

eMSC システムの場合には，シナリオが実行順序制約を規定する仕様であり，これを充足する解を得るために ∇ 演算を繰り返し適用すればよいことが分かった．またこの演算は FIFO を挿入することにより，実装に反映させることも可能である．

本研究によりシナリオ形式で記述された仕様に従ってプロセス間連携のための部分プロトコルを逐次挿入してシナリオを合成することによりシステム全体の仕様を構成的に構築する手法が確立された．

今後の展望としては，形式化をシナリオマージにおける合併操作に拡張することである．

参 考 文 献

1) Baeten, J.C.M.: A brief history of process algebra, *Theor. Comput. Sci.*, Vol.335, No.2-3, pp.131–146 (2005).
2) Bergstra, J.A. and Klop, J.W.: The Algebra of Recursively Defined Processes and the Algebra of Regular Processes (1983).
3) Bergstra, J.A. and Klop, J.W.: Process Algebra for Synchronous Communication, *Information and Control*, Vol.60, No.1-3, pp.109–137 (1984).
4) de Alfaro, L., Faella, M., da Silva, L.D., Legay, A., Roy, P. and Sorea, M.: Sociable Interfaces (2005).
5) Fokkink, W.: *Introduction to Process Algebra*, Springer (2000).
6) Iwamasa, M., Yamamoto, K. and Ishii, T.: System Level Specification Synthesis-An Extended Message Sequence Chart based Approach, *The 13rd Workshop on Synthesis and System Integration of Mixed Information Technologies* (*SASIMI*) (2006).
7) Liang, H., Dingel, J. and Diskin, Z.: A comparative survey of scenario-based to state-based model synthesis approaches, *Proc. 2006 International Workshop on Scenarios and State Machines: Models, algorithms and tools*, pp.5–12 (2006).
8) Tarski, A.: A lattice-theoretical fixpoint theorem and its applications, *Pacific Journal of Mathematics*, Vol.5, No.2, pp.285–309 (1955).
9) Uchitel, S. and Chechik, M.: Merging partial behavioural models, *SIGSOFT Softw. Eng. Notes*, Vol.29, No.6, pp.43–52 (2004).

(平成 21 年 2 月 2 日受付)
(平成 21 年 9 月 11 日採録)

岩政　幹人（正会員）

1989 年京都大学大学院理学研究科修士課程修了．同年（株）東芝入社．1994 年から 96 年にかけて米国スタンフォード大学客員研究員．2005 年より北陸先端科学技術大学院大学（JAIST）情報学研究科博士後期課程在学中．知能情報処理の研究・開発，システムレベル設計（回路設計）の開発，ソフトウエア高信頼化技術の研究・開発に従事．IEEE-CS 会員．

日比野　靖（正会員）

1972 年東京工業大学大学院電子物理工学専攻修士課程修了．同年 NTT 入社，1986 年同社基礎研究所第二研究室長．1993 年北陸先端科学技術大学院大学（JAIST）教授．2002 年国立情報学研究所教授．2004 年北陸先端科学技術大学院大学教授，2008 年より同大学副学長．博士（工学），LISP マシン（ELIS）の研究・開発・事業化，極限集積下でのプロセッサの設計法，高品位マルチメディア通信，暗号プロセッサの研究に従事．電子情報通信学会，IEEE，ACM 各会員．