NAIST-IS-DT0061203

博士論文

同期イベントを用いたプロセス分解法に関する研究

喜家村 奨

2003年 12月 8日

奈良先端科学技術大学院大学
情報科学研究科 情報処理学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

喜家村 奨

審査委員：　関 浩之 教授
渡邉 勝正教授
伊藤 実教授

同期イベントを用いたプロセス分解法に関する研究*

喜家村 奨

内容梗概

プロセス分解は分散システム設計のための有効なアプローチの一つである．例えば，通信プロトコルの設計では，通信システム全体の振舞いを満たすサービス定義をまず最初に記述する．次にそのサービス定義を複数の通信プロセスに分解し，プロトコル仕様を導出する．このようなシステムを設計する場合，要求仕様として与えられたプロセスを実装条件を満たすプロセス群に自動分解できれば，システム全体の設計コストおよびシステムの実装条件の変更にともなうコストを削減できる．上記のような観点から，本研究では，同期イベントを追加してプロセスを分解する問題について考察し，その新しい分解法を提案する．本研究で扱うプロセス分解問題とは次のような問題である：問題への入力は，１つのプロセス (L_{in} とする) の動作定義および L_{in} を分解した各成分プロセス $L_1,\ldots,L_n$ と環境間のインタフェイス情報である．あるプロセスのインタフェイス情報とは，そのプロセスのイベントの集合（アルファベット）を定めたものである．一方，プロセスの動作定義は遷移システム (LTS) で与えられる．プロセス分解問題とは，次の条件を満たすプロセス群 $L_1,\ldots,L_n$ を構成することである．条件：$L_1,\ldots,L_n$ の合成プロセス ($L_1\times\cdots\times L_n$ と表す) は L_{in} と弱双模倣等価であり，各 $L_1,\ldots,L_n$ はインタフェイス条件を満たす．ここで各 $L_i(1\leq i\leq n)$ は，インタフェイス条件で指定したイベント以外に，他の成分プロセスとの同期のための新しいイベントを利用してもよい．このプロセス分解問題に対して本論文では２つの分解法を提案する．

１つ目の提案法は，システム内の通信がシステムとその環境との通信より優先的に実行できるような実装に対して，同期イベントの挿入箇所を従来法より少なくできるというものである．また，本分解法の正当性，すなわち，L_{in} から得られた $L_1,\ldots,L_n$ に対して，L_{in} と $L_1\times\cdots\times L_n$ の弱双模倣等価性を証明する．更に，$L_1,\ldots,L_n$ において，各成分プロセスの状態数およびプロセス間の同期のためのメッセージ数を減らす手法についても考察する．

２つ目の提案法は，イベント間の発生の相関関係に着目した分解法である．この提案法の主な優位点は分解で得られた成分 LTS のサイズが従来の分解法よりしばしば小

*奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文, NAIST-IS-DT0061203, 2003 年 12 月 8 日.

さくなることである．特に単純カウンタ及び一般カウンタとよばれるクラスに属するプロセスを，提案手法に基づきサイズの小さいカウンタプロセスに分解するアルゴリズムを示す．

キーワード

並列システム，ラベル付き遷移システム，プロセス分解，カウンタプロセス，弱双模倣等価

Process Decomposition Methods for Labeled Transition System via Synchronization Events*

Susumu Kiyamura

Abstract

Process decomposition is one of the promising approaches to distributed system design. For example, in communication protocol design, we may first write a service definition which specifies the global behavior of the system, and then derive a protocol specification by decomposing the service definition into communicating processes. In such a system design, if a single process given as a requirement specification can be automatically decomposed into processes which satisfy implementation-dependent constraints, we can reduce the whole cost of the system design, especially, the cost for modifying the design when implementation-dependent constraints are changed. From this point of view, we consider a process decomposition problem which allows one to use additional synchronization events, and propose a new decomposition method.

The process decomposition is defined as follows. An input of the decomposition problem (or a requirements specification) is a pair of a behavior definition of a process and an interface condition for decomposed processes. A behavior definition of a process is given by a labeled transtition system (LTS), say L_{in}. An interface condition of a process specifies the set of events which the process should participated in. The problem is to construct LTSs $L_1, \ldots, L_n$ such that $L_1, \ldots, L_n$ meet the interface condition and also the composite process of $L_1, \ldots, L_n$ is observably bisimulation equivalent to L_{in}. Note that we can introduce new events other than those appearing in L_{in} and use them for synchronization between $L_1, \ldots, L_n$. These events are called synchronization events. In this thesis, we propose two methods for the process decomposition problem.

The first method can be applied to an arbitrary input LTS and is based on the following idea: Assume that $s \xrightarrow{a} s'$ (state transition is possible from state s to state s' by event a) in L_{in} and L_{in} is being decomposed into L_1 and L_2 where L_1 has event a and L_2 does not. Then, L_1 and L_2 synchronize with a new event immediately after L_1 executes event a to inform L_2 of L_1's executing a. The proposed method introduces a new event exactly when

*Doctor's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT0061203, December 8, 2003.

a synchronization is needed. We prove the correctness of the decomposition method, i.e., the observable bisimulation equivalence between an input LTS and the composite LTS constructed by the method. Furthermore, a method of decreasing the number of process states as well as decreasing the number of synchronization events is presented.

The second proposed method is in a sense an extension of the first method. The main advantage of the method is that the size of the decomposed LTSs is often much smaller than other methods. Especially, we focus on simple counters and generalized counters and present an algorithm which decomposes a simple (or generalized) counter into smaller counters by using the proposed method.

Keywords:

Parallel System, Labeled Transition System(LTS), Process Decomposition, Counter Process, Bisimulation

目 次

図 目 次

表 目 次

第1章　まえがき

1.1　研究背景

計算機システムが大規模，複雑化する今日, 形式的手法を用いたシステムの設計が不可欠になってきている．並列処理システムをプロセス代数などの形式的手法を用いて設計し，その正当性を検証する手法については既に多くの研究がなされている (例えば文献 [1], [2], [3])．また，大規模なシステムを構築する場合, 要求仕様の決定から実装までを多数のフェーズに分けて設計することが多い. すなわち, 抽象的なレベルの要求仕様をまず作成し, 段階的に詳細化して, 最終的に実装レベル仕様を得る. このような設計法では, 一般に各フェーズごとに設計者がシステムの構造を決定し, フェーズ間の整合性（下位のフェーズの設計仕様が上位のフェーズの設計仕様を満たしているかどうか）を（可能ならば）数学的手法を用いて検証する. 各フェーズ間の整合性の数学的検証法およびツールを用いた検証については，文献 [4], [3], [5] などで報告されている．しかし, このような従来の方法では, 各フェーズの仕様作成は人手で行われており, 仕様作成やフェーズ間の整合性の検証に多くの時間が必要となっているのが現状である.

実装時におけるプロセスの並列度など, 実装に関するいくつかの制約事項は, そのシステムに許されるハードウエアに対するコストやシステムの各コンポーネントの物理的な配置条件によって決定される. 要求仕様, 及び, 実装に関する制約事項が与えられたとき, これらの制約事項を満たし, かつ要求仕様と等価な実装レベルの仕様を自動的に導出できれば, システム全体の設計コストを削減できる．また, 仕様変更によって実装レベルの並列度などが変更された場合も，再度実装レベルの仕様を書き直し検証する必要がないため，設計コストを削減できる．例えば，図 1.1 に示すように１つまたは複数の部分仕様の合成として与えられた要求仕様を実装条件 (図では２つのプロセスに分解) に基づいて自動分解できれば，実装を人手により記述し，要求仕様との整合性を検証する必要がなく，更に，実装条件が変更（図では３つのプロセスで実装するように変更）されても再度，実装を記述，検証し直す必要がない．このような観点から，仕様の自動分解は並列システムを設計する上で非常に有効であると思われる.

本論文では, 上記のような観点から，単一プロセスからなる要求仕様から複数のプロセスからなる実装レベル仕様への分解法について考察する．並列プロセスの仕様は, プロセスと環境間のインタフェイス情報と, 各成分プロセスの動作定義からなると考え

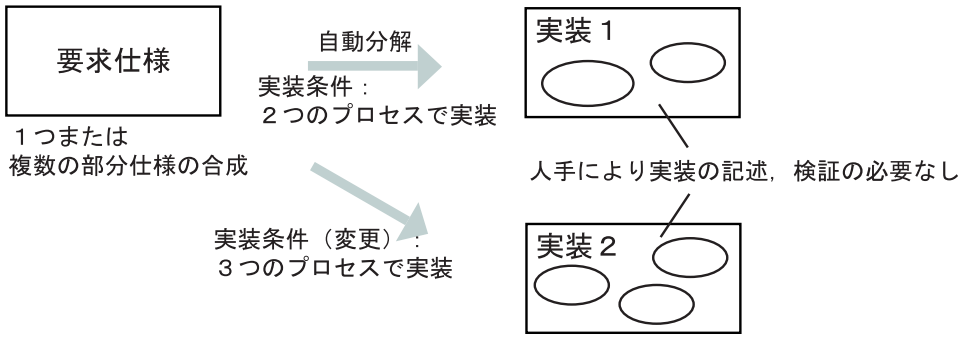


図 1.1 自動分解の効果

られる. インタフェイス情報とは, 成分プロセスの総数および各成分プロセスごとにその入出力イベントの集合（アルファベット）を定めたものである. 一方, プロセスの動作定義はラベル付き遷移システム (LTS) で与えるものとする. さて, 利用可能プロセス数や入出力ポートの配置などの実装上の制約や対コスト性能比の要求などから, 実装レベル仕様におけるインタフェイス情報はあらかじめ決定されている場合が多い. そこで, 本論文では, 要求仕様, および実装レベル仕様における環境とのインタフェイス情報が与えられたとき, 各実装プロセスの動作定義を与えるプロセスを求める問題をプロセス分解問題として定義し，プロセス分解問題を解くアルゴリズムを提案する.

プロセス分解問題（プロセス合成問題ともいう）とは，イベント（またはアクション）の集合 A に関する単一プロセス L_{in} および，その環境とのインタフェイス情報として各出力プロセスのアルファベット（$B_1,\cdots,B_n$）が与えられた時に，$L_1,\ldots,L_n$ の合成プロセス（$L_1\times\cdots\times L_n$ で表す）が L_{in} と等価な振舞をするようにプロセス $L_1,\ldots,L_n$ を構成することである.

並列システムを表現する方法としては，時相論理式を用いる方法，Z や VDM などの集合論に基づく形式的技法を用いる方法，CCS や CSP などのプロセス代数を用いる方法，LOTOS [5] や SDL のような仕様記述言語を用いる方法および，本論文のようにラベル付き遷移システムを用いる方法などがある．一つのプロセスとして表現される仕様を複数のプロセスに分解する問題の難しさは，分解後のプロセスが同時並列に動作した時，仕様プロセスと矛盾ない順序でイベントを実行するかという点である．LTS はプロセスの動作が状態遷移として陽に表現されているため，この点に着目する上で非常に適したモデルである．よって本論文ではプロセスを LTS として表現し，その分解法を考える．また，プロセス代数や LOTOS などの仕様記述言語では，LTS と同様，イベントの実行による状態遷移に基づきその操作的意味が定義されている．これらの言語は状態空間が無限集合であり LTS よりも一般的なモデルであるが，本論文で提案する LTS の分解アルゴリズムを適用できる可能性がある.

1.2 *LTS* 分解問題に関する関連研究

この節では，LTS 分解問題に関する研究について紹介する．文献 [6] において，Morin は，文献 [7] におけるペトリネットのための分解法をラベル付き遷移システム（LTS）へ拡張した．そして，合成プロセス $L_1 \times \cdots \times L_n$ の状態空間が L の状態空間と同形となるように LTS L を $L_1, \ldots, L_n$ に分解できるための必要十分条件を示している．Morin はまた，最適な分解を見つけるための計算量が NP-完全であること，および，I/O システムと呼ばれるサブクラスに対しては最適な分解が多項式時間で求められることを示した．文献 [8] において，Mukund は，状態空間の同形性 ([6] と同じ)，言語等価性，双模倣等価性の 3 種のプロセス等価性それぞれに関して分解問題を議論している．そして，状態空間の同形性と言語等価性について，分解可能であるための必要十分条件を示している．双模倣等価性については，決定性システムにおいては言語等価性と双模倣等価性が一致するという事実を利用し，決定性システムに分解する分解法が述べられている．これらの研究では，プロセス L のアルファベットと，$L_1, \ldots, L_n$ のアルファベットの和集合が一致することを前提としている．すなわち，$L_1, \ldots, L_n$ が互いの同期のために新しいイベントを用いることを許さない．

実用的な研究分野として，通信プロトコルの分野では, サービス定義からプロトコル仕様を自動合成する研究が多数行われている [9]. 要求仕様として LTS を用いた従来の研究のうち，要求仕様と実装レベル仕様の等価性を証明している代表的なものと本手法の特長を表 1.1 にまとめる．まず，文献 [10, 11] および本論文では，上記 [6–8] と異なり，新しいイベントを成分プロセス間の同期のために用いることを許している．文献 [10] では，1 つの LTS L を，合成システムが L と弱双模倣等価となるような，共通のイベントを持たない 2 つの LTS に分解する方法が提案されている．本論文 3 章で提案する分解法では，合成システムが L と弱双模倣等価となるような，任意の数の LTS 群に分解できるように，また，冗長な同期イベントを除去するように分解法が拡張されている．文献 [10] の方法は, 表 1.1 に示した前提条件が成り立つときのみ, 本論文第 3 章ステップ 3 (後述) に相当する最適化を行っているが，最適化を行った場合の厳密な等価性の証明は与えられていない. 文献 [11] の方法は本論文第 3 章のステップ 1（後述）に相当する．これらの方法は，次のような単純なアイデアに基づいている．入力 LTS L_{in} において遷移 $s \xrightarrow{a} s'$ (状態 s から s' へイベント a によって遷移可能) であるとし，L_1 はイベント a を自分のアルファベットに含んでいるが，L_2 は a を含んでいないと仮定する．このとき，L_1 と L_2 は新しいイベント (i_a とする) を使って次のように同期する．L_1 が a を実行した直後，L_1 による a の実行を L_2 に通知するため L_1 は i_a を L_2 に送信する．

文献 [12] の方法は 2 つのプロセスの同期を制御するプロセス群を配置する点で [10, 11] や本手法とアプローチが異なる．有限状態モデルより広いモデルを用いたものとして，例えば [13] では, レジスタをもつ拡張 FSM(EFSM) を用い, 一つの EFSM で表

表 1.1 LTS を要求仕様とする分解法の比較

	アルファベット	分割数	同期の手段	冗長な同期の削除	等価性の証明
Langerak [10]	互いに素	2	内部イベント	部分的にあり	弱双模倣等価
馬渕ら [11]	互いに素	≥ 2	内部イベント	なし	弱双模倣等価
郷ら [12]	互いに素	2	外部イベント	あり	強双模倣等価
本研究	制約なし	≥ 2	内部イベント	あり	弱双模倣等価

された要求仕様からプロセス間で交換されるメッセージ数が最小となる実装レベル仕様を, 0-1 整数線形計画法を用いて合成する方法が示されている．[14] では時間制約付き FSM で定義された要求仕様を扱っている．ただし, これら拡張モデルを用いた研究でも，本論文で行うような冗長な同期の削除は考慮されていない．

1.3 本研究の概要

LTS の分解において出力 LTS のサイズが小さくならない理由を例示するため，図 1.2(a) に示すような 1 つの状態からの遷移を複数の LTS に分解する部分を考える．図 1.2(a) ではイベント a,b,c をそれぞれ L_1,L_2,L_3 の 3 つのプロセスが実行するものとして分解する．このような分解において，出力 LTS の合成が入力 LTS と等価な振舞をするようにするためには図 1.2(b) にように同期イベント（例では i,i_a,i_b,i_c）を挿入し，各プロセスがイベント a,b,c を勝手に実行しないように制御する方法がある [10,11]．しかし，この方法では，同期イベントによる遷移数が多く，出力 LTS のサイズが小さくならない．

このような問題に関して，本論文 3 章では，連続イベント制約というある種の遷移に関する制約（例えば図 1.2(c) では，L_1 の状態 s においてイベント a が実行された後, イベント i_a 以外のイベントを他の LTS が実行できないという制約）を設けて，同期イベントの挿入箇所を減らし，小さい出力 LTS を得ることができる分解法を提案する．また，3 章では，提案したアルゴリズムが出力した LTS と入力 LTS 間の弱双模倣等価性を証明する．さらに，インターリーブ領域を簡単化して総状態数を小さくする方法や，システム内部で用いられる同期イベントを共通化する方法についても述べる．

本論文 4 章で提案するもう一つの分解法は，ある特定の LTS のクラスについて，サイズの小さい LTS を出力するアルゴリズムである．図 1.3(a) に示すような L_1,L_2,L_3

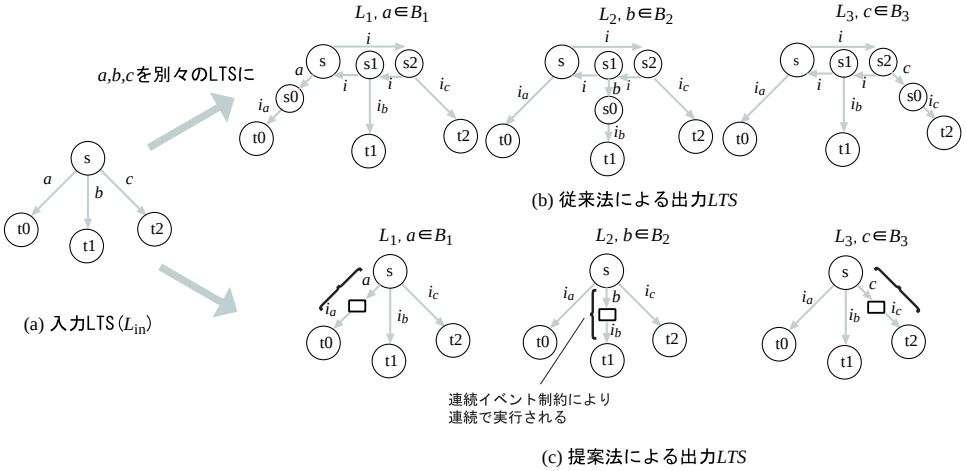


図 1.2 連続イベント制約を用いた分解

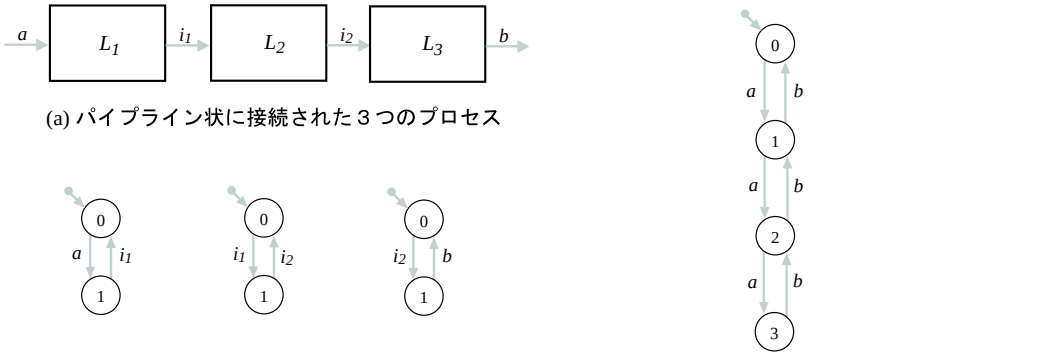


図 1.3 カウンタプロセスの分解

の３つのプロセスで構成されたシステムを考える．このシステムは L_1 から入力したイベント a を L_2 を通じて（内部イベント i_1, i_2 の実行によって）L_3 に b として出力するパイプラインを構成しているとする．このプロセスの構造は図 1.3(b) に示すように非常に単純である．しかし，図 1.3(c) に示すような，イベント a, b の実行について $L_1 \times L_2 \times L_3$ と等価な LTS(L_{in}) が与えられたとしても，従来法では，サイズの大きい LTS しか出力できない．システムの並列化において，パイプラインを自動生成できることは非常に有効である．よって，４章では，３章で示した分解法を一般化し，その分解法を用いると図 1.3 のような入力 LTS（カウンタプロセスと呼ぶ）をサイズの小さい LTS に分解できることを示す．

第2章　諸定義

この章では以下の章で共通に用いる記法や用語の定義を述べる.

定義 1 (イベント) *Evt* をイベントの有限集合とする．*Evt* は，観測者から見えない内部イベントと呼ばれる特別なイベント (τ と表す) を含む. □

定義 2 (連続イベント 制約付き LTS) 連続イベント制約付き遷移システム（LTS）は以下の 5 字組.

$$(A, S, \rightarrow, s_0, C)$$

where

$A \subseteq Evt$: イベントの集合 (アルファベット),
S : 状態の集合,
$\rightarrow \subseteq S \times A \times S$: 遷移関係,
$s_0 \in S$: 初期状態,
$C \subseteq S \times (A - \{\tau\}) \times S \times (A - \{\tau\})$:
　連続イベント制約.

$(s, \alpha, s') \in \rightarrow$ であることを $s \xrightarrow{\alpha} s'$ と書く．$s \xrightarrow{\alpha} s'$ であるような s' が存在することを $s \xrightarrow{\alpha} \cdot$ と書き，そのような s' が存在しないことを $s \not\xrightarrow{\alpha} \cdot$ と書く.
$\rightarrow$ は以下の性質を満たさなければならない.

$$(s, a, s', b) \in C \wedge s \xrightarrow{a} s' \supset s' \not\xrightarrow{\alpha} \cdot \text{ for } \forall \alpha \in A - \{b\}. \tag{2.1}$$

□

条件 (2.1) は，連続イベント制約 $(s, a, s', b) \in C$ があるとき, イベント a を実行した直後の状態においては, b 以外のイベント（τ を含む）による遷移が定義されるのを禁止している.

2 つの LTS $L^0 = (A, S, \rightarrow, s_0, \emptyset)$, $L^1 = (A, S, \rightarrow, s_0, C)$ について，L^0 と L^1 の動作は全く同じであるので, LTS を個別に扱う場合は連続イベント制約に本質的意義はない. しかし, 以下に述べる直積 LTS を考えた場合, 成分 LTS の連続イベント制約は直積 LTS の動作に影響する. 以下，$L = (A, S, \rightarrow, s_0, \emptyset)$ である LTS L を簡単に $L = (A, S, \rightarrow, s_0)$ と 4 字組みで書くことがある．また，$A[L]$, $S[L]$, $\rightarrow_L$，$I[L]$ をそれぞれ，L のアルファベット, L の状態集合, L の遷移関係, L の初期状態として書くことがある.

複数の成分プロセスからなる並列プロセスの動作は, 以下の直積 LTS で定義される.

定義 3 (直積 LTS) k 個の LTS, $L^j = (A^j, S^j, \to_j, s_0^j, C^j)$ $(j \in [1,k])$ の**並列合成**を直積 LTS とよび（$L^1 \times \cdots \times L^k$ で表す）, 以下のように定義する.

$L^1 \times \cdots \times L^k := (A, S, \to, s_0, C)$
where
$A := \bigcup_{j \in [1,k]} A^j$,
$S := S^1 \times \cdots \times S^k$,
$s \xrightarrow{\alpha} s' :\Leftrightarrow$
(1) もし $\alpha \neq \tau$ ならば,
任意の $j \in [1,k]$ について, $\alpha \in A^j$ ならば $s^{[j]} \xrightarrow{\alpha}_j s'^{[j]}$, $\alpha \notin A^j$ ならば $s^{[j]} = s'^{[j]}$.
もし $\alpha = \tau$ ならば,
ある $j \in [1,k]$ について $s^{[j]} \xrightarrow{\tau}_j s'^{[j]} \wedge s^{[j']} = s'^{[j']}$ for $\forall j' \neq j$.

加えて, いずれの場合も s, α が以下の条件を満たす.

(2) ある $j \in [1,k]$, $t \in S^j$, $a, b \in A^j$ が存在し, $t \xrightarrow{a}_j s^{[j]}$, $(t, a, s^{[j]}, b) \in C^j$ ならば $\alpha = b$（イベント a で状態 $s^{[j]}$ に到ったならば, $s^{[j]}$ からの遷移は必ずイベント b によって起こる.）
$s_0 := (s_0^1, \ldots, s_0^k)$,
$(s, a, s', b) \in C :\Leftrightarrow$
$(s^{[j]}, a, s'^{[j]}, b) \in C^j$ for $\exists j \in [1,k]$.

□

$\to$ の定義において, (2) は次のことを表している. ある LTS L^j が連続イベント制約 $(t, a, s^{[j]}, b)$ をもつとき, たとえ L^j 以外の LTS L^i において $s^{[i]} \xrightarrow{\alpha}_i \cdot (\alpha \neq b)$ だったとしても, 連続イベント制約が優先し, $L^1 \times \cdots \times L^k$ においては $s \not\xrightarrow{\alpha} \cdot$ となる. (1) は L^j が α をアルファベットに含んでいれば $s'^{[j]}$ に遷移し, 含んでいなければ状態は変化しないことを表す（ただし (2) も満たす）.

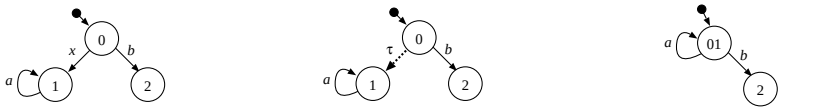
$|A|$ を集合 A の要素の個数, $[m,n]$ を m 以上 n 以下の整数の集合とする.

定義 4 $\widetilde{\Sigma} = (A_1, \ldots, A_k)$ を Evt の部分集合の組, α を 1 つのイベントとするとき, $loc_{\widetilde{\Sigma}}(\alpha) = \{i \in [1,k] \mid \alpha \in A_i\}$ と定義する. □

定義 5 (並列イベント) [6] Evt の部分集合の組 $\widetilde{\Sigma} = (A_1, \ldots, A_k)$, および 2 つのイベント $\alpha, \beta \in A_1 \cup \cdots \cup A_k$ について, 任意の $i \in [1,k]$ に $\{\alpha, \beta\} \not\subseteq A_i$ であるときかつそのときのみ, α, β は**並列**であるといい, $\alpha \parallel \beta$ と書く. □

次に LTS に関する基本操作（**イベントの隠蔽**, **縮約**）を定義する.

定義 6 (イベントの隠蔽) L を LTS とし $H \subseteq Evt - \{\tau\}$ とする. $L\backslash H = (A[L] \cup \{\tau\} - H, S[L], \to, I[L])$ と定義する. ここで $\to$ の定義は以下のとおり: $\alpha \notin H \cup \{\tau\}$ である任



(a) L (b) $L\backslash\{x\}$ (c) $cont(L,\{a,b\})$

図 2.1 イベントの隠蔽と縮約

意のイベント α について，$s \xrightarrow{\alpha}_L s'$ のときかつそのときのみ $s \xrightarrow{\alpha} s'$．$s \xrightarrow{\tau}_L s'$ またはある $\beta \in H$ について，$s \xrightarrow{\beta}_L s'$ のときかつそのときのみ $s \xrightarrow{\tau}_L s'$． □

集合 A 上の同値関係 $\sim$ と $a \in A$ について, a を含む同値類を $[a]_\sim$ と書く．

定義 7 (縮約 (contraction)) L を LTS, $A \subseteq Evt$ をイベントの部分集合とする．$\sim$を，以下の (c2) を満たす $S[L]$ 上の同値関係とする:

$$\text{もし，ある } \alpha \notin A \text{ について，} s_1 \xrightarrow{\alpha}_L s_2 \text{ ならば } s_1 \sim s_2. \qquad \text{(c2)}$$

LTS $L/_\sim$ を $L/_\sim = (A[L] \cap A, \{[s]_\sim \mid s \in S[L]\}, \rightarrow, [I[L]]_\sim)$ と定義する．ここで $\rightarrow$ は次のように定義される: $s_1 \xrightarrow{a}_L s_2$ であるような $s_1 \in h_1$ と $s_2 \in h_2$ が存在するときかつそのときにのみ，$h_1 \xrightarrow{a} h_2$.

また イベントの部分集合 A に対して (c2) を満たす最小の同値関係を $\sim_A$ とするとき，$L/_{\sim_A}$ を $cont(L,A)$ と書く． □

図 2.1 (a) の LTS L に対して，$L\backslash\{x\}$ と $cont(L,\{a,b\})$ を図 2.1 (b) と (c) に示す．定義 7 において, 関係 $\sim$ は，A に含まれないイベントによる遷移で接続されている状態 s_1, s_2 の組 (s_1, s_2) を全て含むような任意の同値関係である．言いかえれば，$s_1 \sim s_2$ ならば，s_1 と s_2 は A のイベントによって区別されない.

定義 8 ε を空のイベント列とする，各イベント $\alpha \in Evt$ について, $\hat{\alpha} \in Evt \cup \{\varepsilon\}$ は，もし $\alpha \neq \tau$ なら α 自身, もし $\alpha = \tau$ であるなら ε を表す. □

定義 9 (観測可能な遷移) 各 LTS L について, $\Rightarrow_L \subseteq S[L] \times (Evt \cup \{\varepsilon\}) \times S[L]$ を以下を満たす最小の関係とする: (a) $s \xRightarrow{\varepsilon}_L s$, (b) もし $s \xRightarrow{\varepsilon}_L s'$ でかつ $s' \xrightarrow{\tau}_L s''$ ならば $s \xRightarrow{\varepsilon}_L s''$, (c) もし $s \xRightarrow{\varepsilon}_L s'$ かつ $s' \xrightarrow{\alpha}_L t'$ かつ $t' \xRightarrow{\varepsilon}_L t$ ならば $s \xRightarrow{\alpha}_L t$. □

定義 10 (等価関係) LTS の組 (L_1, L_2) について

$R \subseteq S[L_1] \times S[L_2]$ が各 $(s_1, s_2) \in R$ と $\alpha \in Evt$ について以下の条件を満たすとき，R は (L_1, L_2) 上の**弱双模倣等価関係** (または単に**模倣関係**) であるという,

(a) If $s_1 \xrightarrow{\alpha}_{L_1} s_1'$, then there exists $s_2' \in S[L_2]$ such that $(s_1', s_2') \in R$ and $s_2 \xRightarrow{\hat{\alpha}}_{L_2} s_2'$.

(b) If $s_2 \xrightarrow{\alpha}_{L_2} s_2'$, then there exists $s_1' \in S[L_1]$ such that $(s_1', s_2') \in R$ and $s_1 \xRightarrow{\hat{\alpha}}_{L_1} s_1'$. □

定義 11 (弱双模倣等価) 2 つの LTS L_1 と L_2 について $(I[L_1], I[L_2]) \in R$ であるような (L_1, L_2) 上の弱双模倣等価関係 R が存在するとき，L_1 と L_2 は**弱双模倣等価**であるといい, $L_1 \approx L_2$ と書く. □

定義 12 (同形) 2 つの LTS L_1 と L_2 についての $s_1, s_1' \in S[L_1]$ と $\alpha \in Evt$ に対し，$s_1 \xrightarrow{\alpha}_{L_1} s_1'$ であるときかつそのときのみ $R(s_1) \xrightarrow{\alpha}_{L_2} R(s_1')$ であるような全単射 $R : S[L_1] \to S[L_2]$ が存在するとき，L_1 と L_2 は**同形** であるという. □

第3章　連続イベント制約を前提としたラベル付き遷移システムの自動分解法

3.1　はじめに

この章では, システム内の通信がシステムとその環境との通信より優先的に実行できるような実装に対して，同期イベントの挿入箇所を従来法より少なくできる分解法について考察する．並列プロセスの仕様は, プロセスと環境間の**インタフェイス情報**と, 各成分プロセスの**動作定義**からなると考えられる. インタフェイス情報とは, 成分プロセスの総数および各成分プロセスごとにその入出力イベントの集合（アルファベット）を定めたものである. 一方, プロセスの動作定義は遷移システム (LTS) で与えるものとする. 次に, この章の構成を示す．２節ではこの章で提案する分解法の概要を説明する．３節では提案する変換アルゴリズムを詳しく述べる．また, 変換前の LTS と, 変換によって得られた LTS との弱双模倣等価性 [5] を証明する（定理１）．４節では各成分プロセスの状態数や同期イベントの数を削減するための改良点について説明する. 特に, 状態数削減の手法として, インタリーブ領域（各プロセスが互いに素なイベントを非同期で実行するような領域）に着目した．3.5 節では，他の文献と同じ仮定をした場合について考察し，この場合も提案アルゴリズムが適用可能であることを述べる.

3.2　準備

3.2.1　要求仕様と実装レベル仕様との関係

本論文で扱う問題の概要を図 3.1 に示す. まず, 要求仕様図 (3.1(a)), および実装レベル仕様におけるインタフェイス情報 (図 3.1(b)) が与えられる. 前者は単一プロセス (L_{in} とよぶ) のインタフェイス情報 (アルファベット) および動作定義からなる．後者は, 実装レベル仕様のプロセスの個数 (n) と各プロセス $L^j_{\mathrm{out}}\ (1 \leq j \leq n)$ のアルファベット B^j からなる. この両者から実装レベル仕様の各プロセスの動作定義 (図 3.1(c)) を導出するのがここでの問題である.

この章ではプロセスの動作定義を連続イベント制約付き遷移システムで与える．これはイベントの発生順に制限を持たせた LTS である．以降，特に断らない限り，連続イベント制約付き遷移システムのことを単に LTS と呼ぶ. 例えば図 3.2(a) の要求仕様

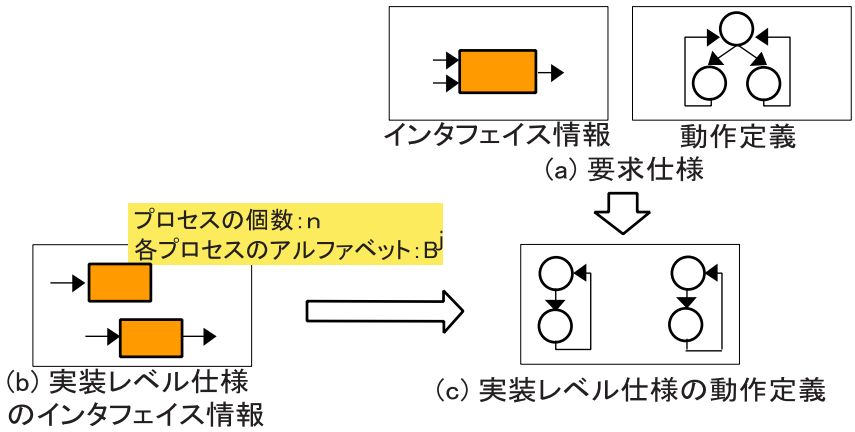


図 3.1 要求仕様と実装レベル仕様の関係

を 2 つのプロセス（$L1$ と $L2$）で実装し, イベント a は $L1$ のみ, イベント b は $L2$ のみが実行する場合を考える. このとき, 各プロセスは相手がイベントを実行したかどうかを知る必要がある. そこで各 LTS がイベントを実行した後, それを他方の LTS に通知する同期イベントを発生させる図 3.2(b). この同期イベントの発生の前に他のイベントが発生しないようにシステム全体に制約をつけなければならない（例えばイベント a が発生した後はそれを通知する同期イベント $i1$ が発生しない限り他のイベントは発生できないようにする）. このようなイベントの発生順に関する制約を連続イベント制約と呼ぶ. 連続イベント制約は $(0, a, 0', i1)$ のように状態, 外部イベント, 中間状態, 同期イベントの組で記述する. なお, [13, 14] 等で行っているように,「L_{in} で図 3.2(a) のような遷移があるときは, a, b は同じプロセスに割り当てられる (*1)」と仮定するならば, 連続イベント制約は不要となる. すなわち, 連続イベント制約は上記の仮定より強くない制約である. (*1) を仮定し, 代わりに連続イベント制約を用いない場合でも, 本手法はそのまま適用できる. これについては 3.5 節で述べる. L_{in} から実装レベル仕様の各 LTS への変換は次の 3 つのステップからなる（図 3.3）.

ステップ (1) L_{in} を n（実装レベル仕様のプロセスの数）個コピーし, $L^1_{\text{out1}}, \ldots, L^n_{\text{out1}}$ とする. ここで,

- LTS L^j_{out1} がイベント a に関与するとき ($a \in B^j$ のとき), イベント a に関与しないプロセスと同期をとるためのイベント i を a の直後に挿入する. a と i は連続イベント制約をみたすとする.
- LTS L^j_{out1} がイベント a に関与しないとき ($a \notin B^j$ のとき), a を上記の同期イベント i に置き換える.

ステップ (2) L^j_{out1} において不必要な同期イベントを ε に変更する (得られた各 LTS を L^j_{out2} とする).

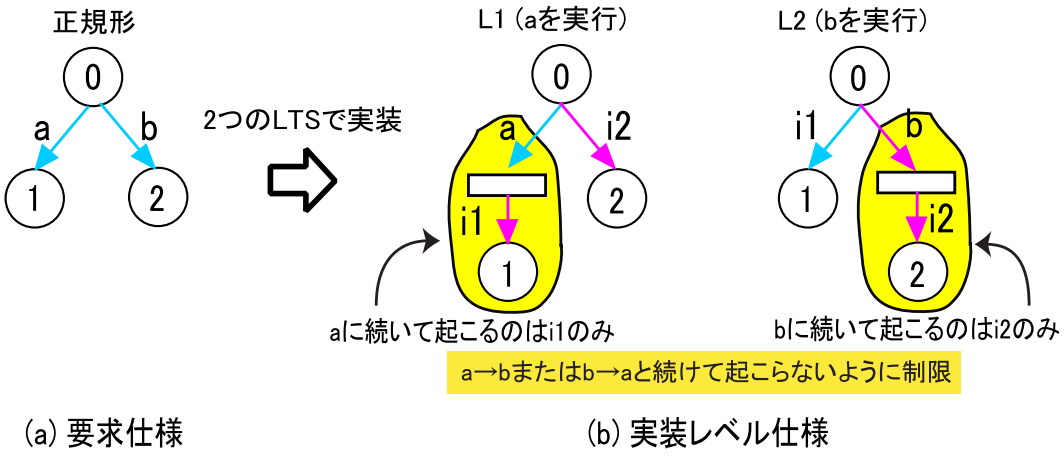


図 3.2 実装レベル仕様におけるプロセス間の同期

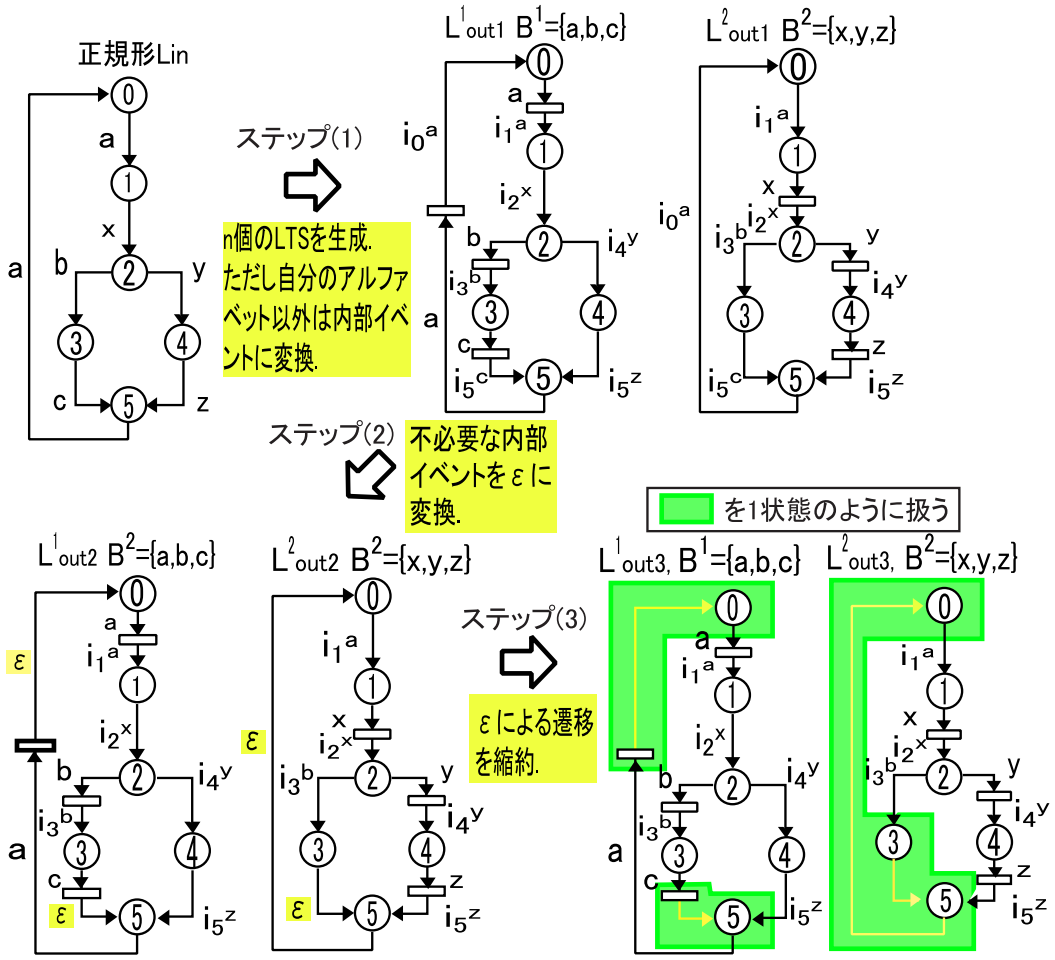


図 3.3 変換ステップの概略

ステップ (3) L^j_{out2} においてεによる遷移を縮約する (得られた各 LTS を L^j_{out3} とする).

ステップ 3 で得られた各 LTS L^j_{out3} を成分とする並列プロセスが, 実装レベル仕様となる.

3.3 LTS 分割問題を解くアルゴリズム

この節では, この章で扱う問題である LTS 分割問題を定義し, これを解くアルゴリズムを述べる.

定義 1 (LTS 分割問題) 以下の入出力で定義される問題を LTS 分割問題という.

入力: $L_{\text{in}} = (A, S, \to_{\text{in}}, s_0, \emptyset)$: LTS.
n : 自然数.
$B^j \subseteq A\ (j \in [1,n])$. ただし $B^1 \cup \cdots \cup B^n = A - \{\tau\}$.

出力: $L^j_{\text{out}} = (A^j, S^j, \to_j, s^j_0, C^j)$ for $\forall j \in [1,n]$: LTS の系列.
ただし $B^j \subseteq A^j, (A^j - B^j) \cap (A \cup \{\tau\}) = \emptyset$ かつ, $Evt_{\text{sync}} = \bigcup_{j=1}^{n}(A^j - B^j)$ とおくとき, $L_{\text{in}} \approx (L^1_{\text{out}} \times \cdots \times L^n_{\text{out}}) \backslash Evt_{\text{sync}}$.
Evt_{sync} は実装レベル仕様内で用いられる同期のためのイベント集合である.

□

以降, 定義 1 の入力 L_{in}, A, S, $\to_{\text{in}}$, s_0, B^j および n を固定する. ここで L_{in} は以下を満たすと仮定する.

仮定 2 任意の $s, s_1, s_2 \in S$, $a \in A - \{\tau\}$ について, $s \xrightarrow{a}_{\text{in}} s_1 \wedge s \xrightarrow{a}_{\text{in}} s_2 \wedge s_1 \neq s_2$ ならば, $a \in B^j$ となる $j \in [1,n]$ はちょうどひとつ. □

もし L_{in} が仮定 2 を満たさない場合, そのようなすべての $s, s_1, s_2 \in S$, $a \in A - \{\tau\}$ について, 遷移 $s \xrightarrow{a}_{\text{in}} s_2$ を削除した後, 新しい状態 s'_2 および遷移 $s \xrightarrow{\tau}_{\text{in}} s'_2$, $s'_2 \xrightarrow{\tau}_{\text{in}} s$, $s'_2 \xrightarrow{a}_{\text{in}} s_2$ を追加することで, 仮定 2 を満たす LTS を得ることができる. かつ, この LTS は元の L_{in} と弱双模倣等価である (遷移 $s \xrightarrow{\tau}_{\text{in}} s'_2$ および $s'_2 \xrightarrow{\tau}_{\text{in}} s$ を追加することによって, s, s'_2 どちらからもイベント a による遷移 (および τ による遷移) で s_1 および s_2 に遷移可能となる). よって仮定 2 を置いても一般性は失われない.

提案アルゴリズムが生成する実装レベル仕様および中間的な仕様においては, S, A, B^j に加えて以下の状態およびイベントの集合が用いられる.

定義 3 $L_{\text{in}} = (A, S, \to_{\text{in}}, s_0, \emptyset)$ に対し, 集合 S_{aux} および A_{aux} を以下のように定義する.

$$S_{\text{aux}} := S \times (A - \{\tau\}) \times S,$$
$$A_{\text{aux}} := \{(s, a, s') \in S \times A \times S \mid s \xrightarrow{a}_{\text{in}} s'\}.$$

S_{aux} の要素は状態として，A_{aux} の要素は同期イベントとして用いられる．見やすさのため，S_{aux} の要素 (s,a,s') を $t^a_{s,s'}$，A_{aux} の要素 (s,a,s') を $i^a_{s,s'}$ と書く．ただし，A_{aux} の要素の第 1 成分が重要でない場合は，省略して $i^a_{s'}$ と書く．

A_{aux} の部分集合 A_τ を以下のように定義する．

$$A_\tau := A_{\text{aux}} \cap (S \times \{\tau\} \times S).$$

□

A_{aux} の要素の第 1 成分による区別が必要となるのは，補題 4 の証明において，$s_1 \xrightarrow{\tau}_{\text{in}} s$ かつ $s_2 \xrightarrow{\tau}_{\text{in}} s$ であるような s, s_1, s_2 が存在するときのみである．このような s, s_1, s_2 が存在しなければ，A_{aux} の要素の第 1 成分を無視して構わない．以下では, 2.1 節の各ステップについて詳細を述べる．

3.3.1 ステップ (1)：L^j_{out1} を求める

ステップ (1) では, まず, 要求仕様を表わす LTS（L_{in}）を n 個複製する. 次に複製した各 LTS について, そのアルファベットに含まれないイベントを同期イベントに変換する. アルファベットに含まれるイベントについては, そのイベントの実行後, その実行を他の LTS に知らせる同期イベントを挿入する. 以下に L^j_{out1} の定義を示す.

定義 4 L_{in}, B^j $(j \in [1,n])$ に対して L^j_{out1} を以下の通り定義する．

$$
\begin{aligned}
&L^j_{\text{out1}} = (A^j_1, S \cup S_{\text{aux}}, \rightarrow_{1,j}, s_0, C^j_1) \\
&\text{where} \\
&\quad A^j_1 = B^j \cup A_{\text{aux}}, \\
&\quad \rightarrow_{1,j}, C^j_1 : \text{ 以下を満たす最小の関係 :} \\
&\qquad \text{for } \forall s, s' \in S, a \in A, \\
&\qquad a \in B^j \wedge s \xrightarrow{a}_{\text{in}} s' \supset \\
&\qquad\quad s \xrightarrow{a}_{1,j} t^a_{s,s'} \wedge t^a_{s,s'} \xrightarrow{i^a_{s,s'}}_{1,j} s' \wedge (s, a, t^a_{s,s'}, i^a_{s,s'}) \in C^j_1, \\
&\qquad a \notin B^j \wedge s \xrightarrow{a}_{\text{in}} s' \supset s \xrightarrow{i^a_{s,s'}}_{1,j} s'.
\end{aligned}
$$

□

3.3.2 ステップ (2)：L^j_{out2} を求める

ステップ (2) では，ステップ (1) で得られた L^j_{out1} の中で，実装レベル仕様においては必要でない同期イベントを ε に変換する．実装レベル仕様において必要な同期イベントとは，以下で述べる実行可能通知イベントおよび実行不能通知イベントである．

実行不能通知イベントとは, L_{in} のある状態 s から 2 つのイベント a, b による遷移があり, 一方のみ (仮に b とする) をアルファベットにもつ成分 LTS(図 3.4(a) の L^1_{out1}) が

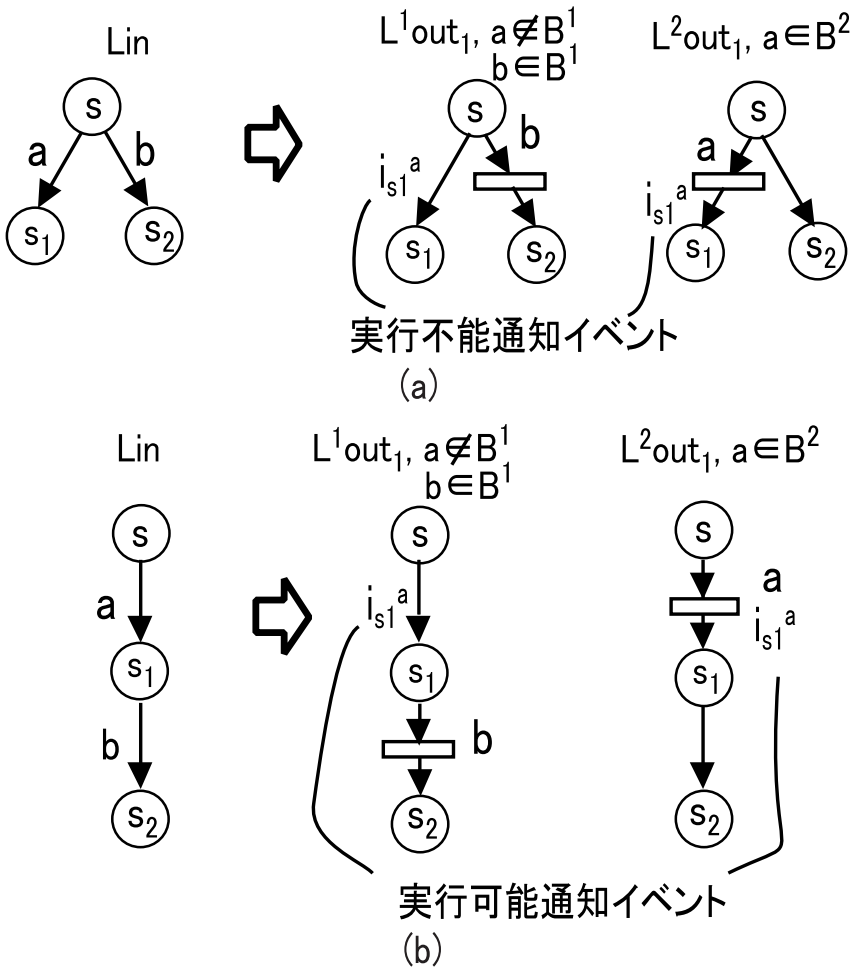


図 3.4 実行不能, 可能通知イベント

存在するときに, 他方 (ここでは a) を実行した LTS (L^2_{out1}) が実行する同期イベント (図 3.4(a) の $i^a_{s_1}$) のことである. すなわちこの同期イベントは, b の実行が禁止されたことを他の LTS に通知している.

実行可能通知イベントとは, L_{in} のある状態 s から s_1 への a による遷移があり，s_1 から s_2 への b による遷移があるとき，a をアルファベットに含まないが, b は含むような LTS(図 3.4(b) の L^1_{out1}) に対して, a を実行し状態 s_1 に到達した LTS(L^2_{out1}) が，b の実行が可能になったことを通知するイベント (図 3.4(b) の $i^a_{s_1}$) のことである.

L_{in} が内部イベント τ による遷移を含む場合は，以上の実行可能（不能）通知イベントの定義を，τ による遷移を 0 回以上行った後に行える遷移に関して拡張する必要がある．準備として τ 閉包を以下のように定義する.

定義 5 L_{in} の任意の状態 $s \in S$ に対し，以下の条件を満たす最小の集合を $CL_\tau(s)$ と書く.

$$
\begin{aligned}
&s \in CL_\tau(s), \\
&\quad s' \in CL_\tau(s) \wedge (s' \xrightarrow{\tau}_{\text{in}} s'') \supset s'' \in CL_\tau(s).
\end{aligned}
$$

□

実行不能通知イベントおよび実行可能通知イベントの形式的定義は以下のようになる.

定義 6 L_{in} および $B^1, \ldots, B^n$ において,

$$
\begin{aligned}
&s \xrightarrow{a} s_1 \wedge s' \in CL_\tau(s) \wedge s' \xrightarrow{b} s_2 \wedge \\
&\quad a \notin B^j \wedge b \in B^j
\end{aligned}
$$

が成り立つような $a, b \in A$, $s, s_1, s_2 \in S$ および $j \in [1, n]$ が存在するとき，i^a_{s,s_1} を L^j_{out1} における実行不能通知イベントと呼ぶ．同時に，$a \in B^k$ である任意の $k \in [1, n]$ について, i^a_{s,s_1} を L^k_{out1} における実行不能通知イベントと呼ぶ.

任意の $j \in [1, n]$ について，L^j_{out1} における実行不能通知イベントすべてからなる集合を $Disabling^j$ とする. □

定義 7 L_{in} および $B^1, \ldots, B^n$ において,

$$
\begin{aligned}
&s \xrightarrow{a} s_1 \wedge s'_1 \in CL_\tau(s_1) \wedge s'_1 \xrightarrow{b} s_2 \wedge \\
&\quad a \notin B^j \wedge b \in B^j
\end{aligned}
$$

が成り立つような $a, b \in A$, $s, s_1, s_2 \in S$ および $j \in [1, n]$ が存在するとき，i^a_{s,s_1} を L^j_{out1} における実行可能通知イベントと呼ぶ．同時に，$a \in B^k$ である任意の $k \in [1, n]$ について, i^a_{s,s_1} を L^k_{out1} における実行可能通知イベントと呼ぶ.

任意の $j \in [1, n]$ について，L^j_{out1} における実行可能通知イベントすべてからなる集合を $Enabling^j$ とする. □

以下に L^j_{out2} の定義を示す.

定義 8 L^j_{out1} ($j \in [1,n]$) に対し，L^j_{out2} を以下の通り定義する．

$$
\begin{aligned}
&L^j_{\text{out2}} = (A^j_2 \cup \{\varepsilon\}, S \cup S_{\text{aux}}, \rightarrow_{2,j}, s_0, C^j_2)\\
&\text{where}\\
&\quad A^j_2 = B^j \cup \mathit{Disabling}^j \cup \mathit{Enabling}^j,\\
&\quad \rightarrow_{2,j}\text{: 以下を満たす最小の関係 :}\\
&\qquad \text{for } \forall s, s' \in S \cup S_{\text{aux}}, a \in B^j,\ i \in A_{\text{aux}},\\
&\qquad s \xrightarrow{a}_{1,j} s' \supset s \xrightarrow{a}_{2,j} s',\\
&\qquad s \xrightarrow{i}_{1,j} s' \wedge i \in \mathit{Disabling}^j \cup \mathit{Enabling}^j \supset s \xrightarrow{i}_{2,j} s',\\
&\qquad s \xrightarrow{i}_{1,j} s' \wedge i \notin \mathit{Disabling}^j \cup \mathit{Enabling}^j \supset s \xrightarrow{\varepsilon}_{2,j} s'.\\
&\quad C^j_2\text{: 以下を満たす最小の関係 :}\\
&\qquad \text{for } \forall s \in S, t \in S_{\text{aux}}, a \in B^j,\ i \in A_{\text{aux}},\\
&\qquad (s,a,t,i) \in C^j_1 \wedge i \in \mathit{Disabling}^j \cup \mathit{Enabling}^j \supset (s,a,t,i) \in C^j_2.
\end{aligned}
$$

□

以上の定義より，以下が成り立つことが容易に言える．

補題 1 $\forall s, s', s'' \in S \cup S_{\text{aux}}$, $a, a' \in \mathit{Evt}$, $j \in [1,n]$ について以下が成り立つ．

(1) $a \in B^j$ ならば，$s \xrightarrow{a}_{1,j} s' \Leftrightarrow s \xrightarrow{a}_{2,j} s'$.

(2) $s \xrightarrow{\varepsilon}_{2,j} s'$ ならば，ある $i^b_{s'} \in A_{\text{aux}}$ が存在し，$s \xrightarrow{i^b_{s'}}_{1,j} s'$.

(3) $s' \in S_{\text{aux}}$ ならば，任意の $i \in A_{\text{aux}}$ について $s \not\xrightarrow{i}_{1,j} s'$.

(4) $s \in S_{\text{aux}}$, $s \xrightarrow{a}_{1,j} s'$ かつ $s \xrightarrow{a'}_{1,j} s''$ ならば，$a = a' \in A_{\text{aux}}$ かつ $s' = s''$.

□

3.3.3 ステップ (3)：L^j_{out3} を求める

ステップ (3) では，L_{out2} における ε 遷移を縮約する．図 3.3 では，ステップ (2) で得られた L^j_{out2} に対して ε 遷移を縮約した各成分 LTS L^j_{out3} を求めている．例えば L^2_{out2} で状態 3 から 0 回以上の ε 遷移で到達可能な状態 $3,5,0$ を L^2_{out3} では 1 つの状態のように扱う．この状態の集合を L^2_{out2} における (状態 3 の) ε 閉包と呼び，$CL^2_\varepsilon(3)$ で表す．L^2_{out3} は $CL^2_\varepsilon(3)$ の中のどの状態にいるときでも，L^1_{out3} が i^a_1 を実行すれば，それに同期して状態 1 に遷移する．

定義 9 L^j_{out2} の任意の状態 $s \in S \cup S_{\text{aux}}$ に対し，以下の条件を満たす最小の集合を $CL^j_\varepsilon(s)$ と書く．

$$
\begin{aligned}
&s \in CL^j_\varepsilon(s),\\
&s' \in CL^j_\varepsilon(s) \wedge (s' \xrightarrow{\varepsilon}_{2,j} s'') \supset s'' \in CL^j_\varepsilon(s).
\end{aligned}
$$

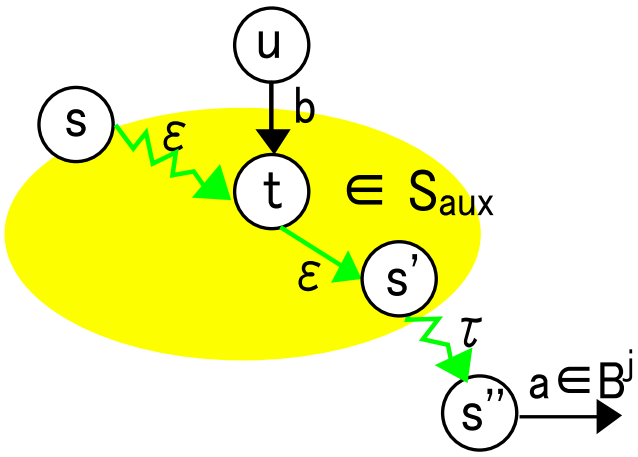


図 3.5 補題 2 の説明図 ($t \in S_{\mathrm{aux}}$ の場合)

□

補題 2 $s \in S \cup S_{\mathrm{aux}}$, $s', s'' \in S$, $j \in [1,n]$, $a \in B^j$, $s' \in CL_\varepsilon^j(s)$, $s'' \in CL_\tau(s')$, $s' \neq s$ かつ $s'' \xrightarrow{a}_{2,j} \cdot$ ならば, $s \in S_{\mathrm{aux}}$ かつ $CL_\varepsilon^j(s) = \{s, s'\}$. □

（証明）$s' \in CL_\varepsilon^j(s)$ かつ $s' \neq s$ であるから, $\exists t \in CL_\varepsilon^j(s)$ が存在し, $t \xrightarrow{\varepsilon}_{2,j} s'$. 補題 1(2) より $t \xrightarrow{i^b_{s'}}_{1,j} s'$ である $i^b_{s'} \in A_{\mathrm{aux}}$ が存在. そのような任意の $t, i^b_{s'}$ について考える.

(1) $t \in S$ の場合. $\rightarrow_{1,j}$ の定義から $b \notin B^j$. しかし $a \in B^j, s'' \in CL_\tau(s')$ かつ $s'' \xrightarrow{a}_{2,j} \cdot$ なので, $i^b_{s'}$ は実行可能通知イベントとなり ε に変換されない. よって, 仮定は矛盾.

(2) $t \in S_{\mathrm{aux}}$ の場合. 補題 1(2)(3) より $u \xrightarrow{\varepsilon}_{2,j} t$ となる u は存在しない (図 3.5). $t \in CL_\varepsilon^j(s)$ より, $t = s$. 従って $s \in S_{\mathrm{aux}}$. このとき補題 1(2)(4) より s からの遷移は $s \xrightarrow{\varepsilon}_{2,j} s'$ のみ. また, $s'' \in CL_\tau(s')$ かつ $s'' \xrightarrow{a}_{2,j} \cdot$ であるから, 実行不能通知イベントの条件から, 仮にある $i \in A_{\mathrm{aux}}$ について $s' \xrightarrow{i}_{1,j} \cdot$ であったとしても, i は ε に変換されない. つまり, $s' \not\xrightarrow{\varepsilon}_{2,j} \cdot$ である. 以上のことから, $s \in S_{\mathrm{aux}}$ かつ $CL_\varepsilon^j(s) = \{s, s'\}$ であるといえる. □

以下に LTS L_{out3} の定義を示す.

定義 10 L^j_{out2} $(j \in [1,n])$ に対し，L^j_{out3} を以下の通り定義する．

$$
\begin{aligned}
&L^j_{\text{out3}} = (A^j_2, S \cup S_{\text{aux}}, \rightarrow_{3,j}, s_0, C^j_3) \\
&\text{where} \\
&\quad \text{for } \forall s, s' \in S \cup S_{\text{aux}}, \alpha, a, i \in A^j_2, \\
&\quad s \xrightarrow{\alpha}_{3,j} s' :\Leftrightarrow \\
&\qquad t \xrightarrow{\alpha}_{2,j} s' \text{ for } \exists t \in CL^j_\varepsilon(s), \\
&\quad (s, a, s', i) \in C^j_3 :\Leftrightarrow \\
&\qquad (t, a, s', i) \in C^j_2 \text{ for } \exists t \in CL^j_\varepsilon(s).
\end{aligned}
$$

$L_{\text{out3}} = (L^1_{\text{out3}} \times \cdots \times L^n_{\text{out3}}) \backslash A_{\text{aux}}$ とする． □

以降，L_{in} と L_{out3} の双模倣等価性を示す．

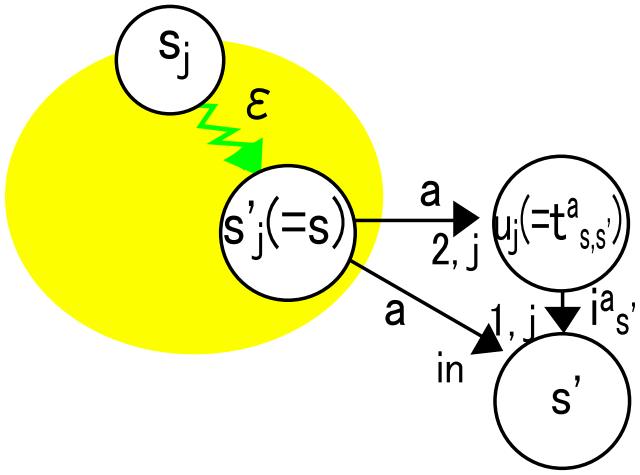


図 3.6 補題 3 の説明図

補題 3 $s \in S$, $j \in [1,n]$, $a \in B^j$，$s_j, u_j \in S \cup S_{\text{aux}}$，$s \in CL^j_\varepsilon(s_j)$ かつ $s_j \xrightarrow{a}_{3,j} u_j$ ならば，ある $s' \in S$ が存在し，$u_j = t^a_{s,s'}$, $u_j \xrightarrow{i^a_{s'}}_{1,j} s'$ かつ $s \xrightarrow{a}_{\text{in}} s'$ である． □

（証明）$s_j \xrightarrow{a}_{3,j} u_j$ より，ある $s'_j \in CL^j_\varepsilon(s_j)$ について $s'_j \xrightarrow{a}_{2,j} u_j$ である (図 3.6).

(1) $s'_j \neq s_j$ のとき．補題 2 より，$CL^j_\varepsilon(s_j) = \{s_j, s'_j\}$ かつ $s_j \in S_{\text{aux}}$．$s \in CL^j_\varepsilon(s_j)$ かつ $s \in S$ より $s = s'_j$．$s = s'_j \xrightarrow{a}_{2,j} u_j$, $a \in B^j$ であることと $\rightarrow_{1,j}$ の定義から，ある $s' \in S$ が存在し，$u_j = t^a_{s,s'} \xrightarrow{i^a_{s'}}_{1,j} s'$ かつ $s \xrightarrow{a}_{\text{in}} s'$.

(2) $s'_j = s_j$ のとき．$s_j \xrightarrow{a}_{2,j} u_j$ であるから，実行不能通知イベントの定義より $s_j \not\xrightarrow{\varepsilon}_{2,j} \cdot$ である．すなわち $CL^j_\varepsilon(s_j) = \{s_j\}$．よって $s = s_j = s'_j$．以降 (1) と同様． □

補題 4 $s \in S$, $j \in [1,n]$，$s_j, s' \in S \cup S_{\text{aux}}$，$s \in CL^j_\varepsilon(s_j)$ かつ，ある $i \in A_\tau$ について $s_j \xrightarrow{i}_{3,j} s'$ ならば，$s' \in S$ かつ $s \xrightarrow{\tau}_{\text{in}} s'$ である． □

（証明）$\rightarrow_{3,j}$ の定義より，$t \stackrel{i}{\rightarrow}_{2,j} s'$ である $t \in CL^j_\varepsilon(s_j)$ が存在する．$i^\tau_{s'} \in Disabling^j$ のとき，$i^\tau_{s'} \in Enabling^j$ のとき，それぞれの場合について，補題 3 と同様に証明できる．□

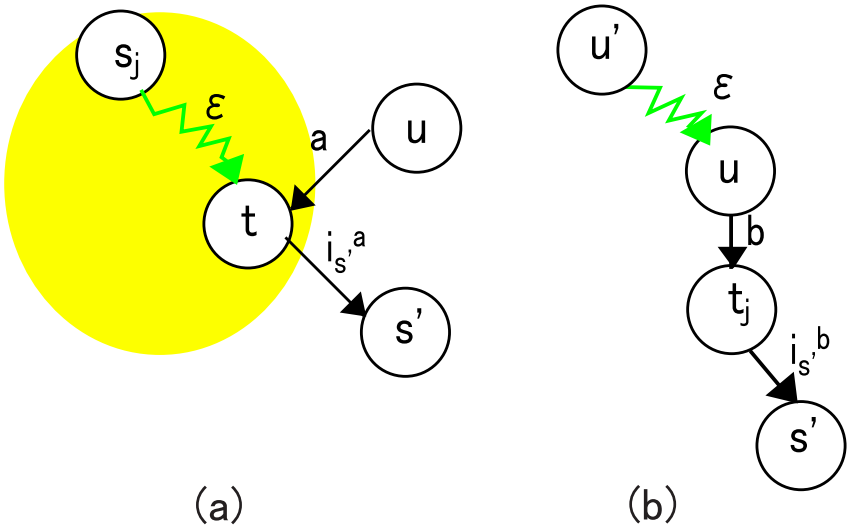


図 3.7 定理 1 の説明図

定理 1 $L_{\text{in}} \approx L_{\text{out3}}$.

（証明）$R \subseteq S \times (S \cup S_{\text{aux}})^n$ を以下のように定義する．

$$
\begin{aligned}
&R := R_1 \cup R_2 \\
&R_1 := \{(s,(s_1,\ldots,s_n)) \mid \\
&\quad s \in S,\ s \in CL^j_\varepsilon(s_j) \text{ for } \forall j \in [1,n]\} \\
&R_2 := \{(s,(t_1,\ldots,t_n)) \mid \\
&\quad s \in S,\ \exists i \in A_{\text{aux}} - A_\tau,\ \forall j \in [1,n], \\
&\quad \text{if } i \in A^j_2 \text{ then } (t_j \stackrel{i}{\rightarrow}_{3,j} s_j) \wedge \\
&\quad s \in CL^j_\varepsilon(s_j) \text{ for } \exists s_j \text{ else } s \in CL^j_\varepsilon(t_j)\}.
\end{aligned}
$$

R が $(L_{\text{in}}, L_{\text{out3}})$ に対する弱双模倣関係であることを示す．以降では，$L_{\text{out3}} = (A \cup \{\tau\}, (S \cup S_{\text{aux}})^n, \rightarrow_3, (s_0,\ldots,s_0), C)$ とする．また，τ による遷移 $(s_1,\ldots,s_n) \stackrel{\tau}{\rightarrow}_3 (s'_1,\ldots,s'_n)$ がイベント $i \in A_{\text{aux}}$ を隠蔽して得られたものであるとき，$(s_1,\ldots,s_n) \stackrel{\tau(i)}{\rightarrow}_3 (s'_1,\ldots,s'_n)$ と書く．

(a) R が定義 10 の条件 (a) を満たす（L_{out3} が L_{in} を模倣できる）ことを示す．

(1) $(s,(s_1,\ldots,s_n)) \in R_1$ の場合．

$s \stackrel{a}{\rightarrow}_{\text{in}} s'$ である任意の $a \in A, s' \in S$ について考える．

まず $a \neq \tau$ と仮定する．各 $j \in [1,n]$ について s'_j を以下のように定める．

- $a \in B^j$ のとき．定義より $s \xrightarrow{a}_{1,j} t^a_{s,s'}, t^a_{s,s'} \xrightarrow{i^a_{s'}}_{1,j} s'$ であり，$s \in CL^j_\varepsilon(s_j)$ なので $s_j \xrightarrow{a}_{3,j} t^a_{s,s'}$ である．
  - $i^a_{s'} \in A^j_2$ のとき．定義より $t^a_{s,s'} \xrightarrow{i^a_{s'}}_{3,j} s'$ である．このとき $s'_j = s'$ とする．明らかに $s' \in CL^j_\varepsilon(s'_j)$ である．
  - $i^a_{s'} \notin A^j_2$ のとき．$s'_j = t^a_{s,s'}$ とする．定義より $t^a_{s,s'} \xrightarrow{\varepsilon}_{2,j} s'$ なので $s' \in CL^j_\varepsilon(s'_j)$ である．
- $a \notin B^j$ のとき．定義より $s \xrightarrow{i^a_{s'}}_{1,j} s'$ である．
  - $i^a_{s'} \in A^j_2$ のとき．定義より $s_j \xrightarrow{i^a_{s'}}_{3,j} s'$ である．このとき $s'_j = s'$ と定める．明らかに $s' \in CL^j_\varepsilon(s'_j)$ である．
  - $i^a_{s'} \notin A^j_2$ のとき．$s'_j = s_j$ と定める．定義より $s \xrightarrow{\varepsilon}_{2,j} s'$ かつ $s \in CL^j_\varepsilon(s_j)$ なので，$s' \in CL^j_\varepsilon(s'_j)$ である．

以上，いずれの場合も $s' \in CL^j_\varepsilon(s'_j)$ なので $(s', (s'_1, \ldots, s'_n)) \in R_1$．また，$a \in B^j$ である j が少なくとも一つ存在する．以上より，各 $j \in [1,n]$ について

$$s''_j = \begin{cases} t^a_{s,s'} & \text{if } a \in B^j \\ s_j & \text{if } a \notin B^j \end{cases}$$

と定義すると，$(s_1, \ldots, s_n) \xrightarrow{a}_3 (s''_1, \ldots, s''_n)$ かつ $(s''_1, \ldots, s''_n) \xrightarrow{\tau(i^a_{s'})}_3 (s'_1, \ldots, s'_n)$（または $(s''_1, \ldots, s''_n) = (s'_1, \ldots, s'_n)$）である．これらの遷移が連続イベント制約の影響を受けずに定義されることを示す．ある j について，s_j を第 3 成分とする組 C^j_3 に含まれると仮定する．C^j_3 の要素の第 3 成分は S_{aux} の要素であるので，$s_j \in S_{\text{aux}}$．一方，$s \in S$ かつ $s \in CL^j_\varepsilon(s_j)$ であるので $s_j \xrightarrow{\varepsilon}_{2,j} \cdot$ である．これは，C^j_2, C^j_3 の定義に矛盾する．よって，任意の j について，s_j を第 3 成分とする組は C^j_3 に含まれない．次に，ある j について，$t^a_{s,s'}$ を第 3 成分とする組 c が C^3_j に含まれると仮定する．このとき c の第 4 成分は必ず $t^a_{s,s'}$ である．以上より，上記 2 つの遷移は，連続イベント制約の影響を受けずに必ず定義される．以上より，$(s_1, \ldots, s_n) \overset{a}{\Rightarrow}_{L_{\text{out}3}} (s'_1, \ldots, s'_n)$.

$a = \tau$ の場合は，上記において，すべての j について $a \notin B^j$ である場合に当たる．上記と同様に s'_j を決めると $(s', (s'_1, \ldots, s'_n)) \in R_1$ かつ $(s_1, \ldots, s_n) \overset{\varepsilon}{\Rightarrow}_{L_{\text{out}3}} (s'_1, \ldots, s'_n)$.

(2) $(s, (t_1, \ldots, t_n)) \in R_2$ の場合．
R_2 の定義より，ある $i \in A_{\text{aux}} - A_\tau$ が存在し，

- $i \in A^j_2$ のとき，ある $s_j \in S \cup S_{\text{aux}}$ について $t_j \xrightarrow{i}_{3,j} s_j$ かつ $s \in CL^j_\varepsilon(s_j)$.
- $i \notin A^j_2$ のとき，$s \in CL^j_\varepsilon(t_j)$．このとき $s_j = t_j$ とおく．

このとき，$(s,(s_1,\ldots,s_n)) \in R_1$ かつ $(t_1,\ldots,t_n) \xrightarrow{\tau(i)}_3 (s_1,\ldots,s_n)$（または $(t_1,\ldots,t_n) = (s_1,\ldots,s_n)$）である．ある j について t_j を第 3 成分とする組 c が C_3^j に含まれると仮定する．このとき，$t_j \in S_{\text{aux}}$ かつ $t_j \xrightarrow{\varepsilon}_{2,j} \cdot$ である．$i \notin A_2^j$ とすると，R_2 の定義より $s \in CL_\varepsilon^j(t_j)$ であるが，$s \in S$ であるので，$t_j \in S_{\text{aux}}$ かつ $t_j \xrightarrow{\varepsilon}_{2,j} \cdot$ であることに矛盾する．$i \in A_2^j$ とすると，$t_j \not\xrightarrow{\varepsilon} 2,j\cdot$ より t_j からの遷移は t_j からの遷移は $t_j \xrightarrow{i}_3 s_j$ のみであり，c の第 4 成分は必ず i である．以上より，遷移 $(t_1,\ldots,t_n) \xrightarrow{\tau(i)}_3 (s_1,\ldots,s_n)$ は，連続イベント制約の影響を受けずに定義される．

$s \xrightarrow{a}_{\text{in}} s'$ である任意の $a \in A, s' \in S$ について，(1) より，ある $(s'_1,\ldots,s'_n) \in (S \cup S_{\text{aux}})^n$ が存在し，$(s',(s'_1,\ldots,s'_n)) \in R_1$ かつ $(s_1,\ldots,s_n) \overset{\hat{a}}{\Rightarrow}_{L_{\text{out3}}} (s'_1,\ldots,s'_n)$ である．上記 (*) より $(t_1,\ldots,t_n) \overset{\hat{a}}{\Rightarrow}_{L_{\text{out3}}} (s'_1,\ldots,s'_n)$ が言える．

(b) R が定義 10 の条件 (b) を満たす（L_{in} が L_{out3} を模倣できる）ことを示す．

(1) $(s,(s_1,\ldots,s_n)) \in R_1$ の場合．R_1 の定義より，任意の $j \in [1,n]$ について $s \in CL_\varepsilon^j(s_j) \cdots (*1)$ である．$(s_1,\ldots,s_n) \xrightarrow{a}_3 (s'_1,\ldots,s'_n)$ である任意の $a \in A \cup \{\tau\}, (s'_1,\ldots,s'_n) \in (S \cup S_{\text{aux}})^n$ について考える．

まず $a \neq \tau$ と仮定する．$(s_1,\ldots,s_n) \xrightarrow{a}_3 (s'_1,\ldots,s'_n)$ より，$a \in B^j$ である任意の $j \in [1,n]$（少なくともひとつ存在）について $s_j \xrightarrow{a}_{3,j} s'_j$．補題 3 より，このような各 j に対して，$s'_j = t^a_{s,s'} \xrightarrow{i^a_{s'}}_{1,j} s'$ かつ $s \xrightarrow{a}_{\text{in}} s'$ である $s' \in S$ が存在する．仮定 2 より各 j に対する s' はすべて等しい．以下，$(s',(s'_1,\ldots,s'_n)) \in R_2$ を示す．任意の $j \in [1,n]$ について以下を考える．

- $a \in B^j$ のとき．$s'_j = t^a_{s,s'} \xrightarrow{i^a_{s'}}_{1,j} s'$ である．
  - $i^a_{s'} \in A_2^j$ のとき．$s'_j = t^a_{s,s'} \xrightarrow{i^a_{s'}}_{3,j} s'$ かつ $s' \in CL_\varepsilon^j(s')$ である．
  - $i^a_{s'} \notin A_2^j$ のとき．$s'_j = t^a_{s,s'} \xrightarrow{\varepsilon}_{2,j} s'$ なので $s' \in CL_\varepsilon^j(s'_j)$ である．
- $a \notin B^j$ のとき．$s_j = s'_j$ かつ $s \xrightarrow{i^a_{s'}}_{1,j} s'$ である．
  - $i^a_{s'} \in A_2^j$ のとき．$(*1)$ より $s'_j = s_j \xrightarrow{i^a_{s'}}_{3,j} s'$ かつ $s' \in CL_\varepsilon^j(s')$ である．
  - $i^a_{s'} \notin A_2^j$ のとき．$s \xrightarrow{\varepsilon}_{2,j} s'$ なので $s_j = s'_j$ と $(*1)$ より $s' \in CL_\varepsilon^j(s'_j)$ である．

いずれも s' と s'_j は R_2 の条件を満たす．

次に $a = \tau(i), i \in A_{\text{aux}}$ の場合について考える．$(s_1,\ldots,s_n) \xrightarrow{\tau(i)}_3 (s'_1,\ldots,s'_n)$ より，ある $k \in [1,n]$ が存在し，$i \in A_2^k$．また任意の $j \in [1,n]$ について，$i \in A_2^j$ ならば $s_j \xrightarrow{i}_{3,j} s'_j$，$i \notin A_2^j$ ならば $s_j = s'_j \cdots (*2)$ である．$i \in A_\tau$ かどうかによって場合分けする．

(i) $i \in A_\tau$ のとき．$i = i^\tau_{s'}$ とする．定義から，L^j_{out3} の任意の状態において $i^\tau_{s'}$ による遷移が存在すれば遷移先は s' に一意に決まる．よって，$i^\tau_{s'} \in A_2^j$ である j について

$s_j \xrightarrow{i^{\tau}_{s'}}_{3,j} s'_j = s'$ である．このような j が少なくともひとつ存在することから，補題 4 と $(*1)$ より，$s \xrightarrow{\tau}_{\mathrm{in}} s'$ である．任意の $j \in [1,n]$ について，

- $i^{\tau}_{s'} \in A_2^j$ のとき，$s'_j = s'$ より $s' \in CL^j_{\varepsilon}(s'_j)$.
- $i^{\tau}_{s'} \notin A_2^j$ のとき，$s \xrightarrow{\tau}_{\mathrm{in}} s'$ より $s \xrightarrow{\varepsilon}_{2,j} s'$. これと $(*1),(*2)$ より，$s' \in CL^j_{\varepsilon}(s'_j)$.

よって $(s',(s'_1,\ldots,s'_n)) \in R_1$ である．

(ii) $i \notin A_{\tau}$ のとき．$i = i^a_{s'}$ とする．定義 6, 7 より，$i^a_{s'} \in A_2^j$ である $j \in [1,n]$ のうち，$a \in B^j$ である j が少なくともひとつ存在する．以下，このような j について考える．$s_j \xrightarrow{i^a_{s'}}_{3,j} \cdot$ より，ある $t \in CL^j_{\varepsilon}(s_j)$ が存在し，$t \xrightarrow{i^a_{s'}}_{2,j} \cdot$．$a \in B^j$ より $t \in S_{\mathrm{aux}}$ (図 3.7(a))．補題 1(2),(4) より $t \not\xrightarrow{\varepsilon}_{2,j} \cdot$ である．$s_j = t$ と仮定すると，$t \not\xrightarrow{\varepsilon}_{2,j} \cdot$ かつ $t \neq s \in S$ なので $(*1)$ に矛盾．$s_j \neq t$ と仮定すると，補題 1(2),(3) より $u \xrightarrow{\varepsilon}_{2,j} t$ となる u は存在しないので，$t \notin CL^j_{\varepsilon}(s_j)$ となり矛盾．以上より $i \notin A_{\tau}$ はありえない．

(2) $(s,(t_1,\ldots,t_n)) \in R_2 - R_1$ の場合．
$(t_1,\ldots,t_n) \xrightarrow{a}_3 (s'_1,\ldots,s'_n)$ である任意の $a \in A \cup \{\tau\}$, $(s'_1,\ldots,s'_n) \in (S \cup S_{\mathrm{aux}})^n$ について考える．

$(s,(t_1,\ldots,t_n)) \in R_2 - R_1$ より，ある $i^b_{s'} \in A_{\mathrm{aux}} - A_{\tau}$ が存在して，ある j と s' について $t_j \xrightarrow{i^b_{s'}}_{3,j} s'$ かつ $s \in CL^j_{\varepsilon}(s') \cdots (*3)$ である．定義 6, 7 より，このような j のうち $b \in B^j$ となる j が少なくともひとつ存在する．以下ではこのような j について考える (図 3.7(b))．

$t_j \xrightarrow{i^b_{s'}}_{3,j} s'$ より，ある $v \in CL^j_{\varepsilon}(t_j)$ が存在し $v \xrightarrow{i^b_{s'}}_{2,j} s'$．$b \in B^j$ より $v \in S_{\mathrm{aux}}$．補題 1(2),(3) より $u \xrightarrow{\varepsilon}_{2,j} v$ となる u は存在しないので，$v = t_j$．このとき定義から，ある $u \in S$ が存在し，$u \xrightarrow{b}_{2,j} t_j$ かつ $(u,b,t_j,i^b_{s'}) \in C_2^j$．よって $u \xrightarrow{b}_{3,j} t_j$ かつ $(u,b,t_j,i^b_{s'}) \in C_3^j$ となるので，$(t_1,\ldots,t_n) \xrightarrow{a}_3 \cdot$ となるのは $a = \tau(i^b_{s'})$ のときのみである．

以降では任意の $j \in [1,n]$ について考える．$(t_1,\ldots,t_n) \xrightarrow{\tau(i^b_{s'})}_3 (s'_1,\ldots,s'_n)$ であることから，$i^b_{s'} \in A_2^j$ ならば $t_j \xrightarrow{i^b_{s'}}_{3,j} s'_j$，$i^b_{s'} \notin A_2^j$ ならば $s'_j = t_j \cdots (*4)$ である．定義から，L^j_{out3} の任意の状態において $i^b_{s'}$ による遷移が存在すれば遷移先は s' に一意に決まるので，$i^b_{s'} \in A_2^j$ である j について $s'_j = s'$．$(*3)$ より $s \in CL^j_{\varepsilon}(s'_j)$. また，$i^b_{s'} \notin A_2^j$ である j について R_2 の定義から $s \in CL^j_{\varepsilon}(t_j)$．よって $(*4)$ より $s \in CL^j_{\varepsilon}(s'_j)$．以上より，$(s,(s'_1,\ldots,s'_n)) \in R_1$.
$s \overset{\varepsilon}{\Rightarrow}_{L_{\mathrm{in}}} s$ すなわち $s \overset{\hat{\tau}}{\Rightarrow}_{L_{\mathrm{in}}} s$ より題意は示された． □

3.3.4 アルゴリズムの時間計算量

上記アルゴリズム（ステップ (1)–(3)）の時間計算量について述べる．以降，L_{in} において，遷移が入ることも出ることもない状態や，どの遷移にも現れないイベントはないと仮定する．この仮定により，L_{in} の記述長は L_{in} の遷移数に比例する．L_{in} の遷移数を $|\rightarrow_{\mathrm{in}}|$ と書く．

ステップ (1) は，L_{in} のコピーを n 個作成する．各コピーの記述長は L_{in} の記述長に比例するので，ステップ (1) の時間計算量は $O(n\cdot|\rightarrow_{\mathrm{in}}|)$ である．

ステップ (2) は以下の部分アルゴリズムから構成される．

(2-1) 各 $s\in S$ について $CL_\tau(s)$ を計算する．

(2-2) 集合 $S_{\mathrm{E}}^j(s)=\{s\in S\mid s\xrightarrow{b}_{\mathrm{in}}\cdot$であるような $b\in B^j$が存在 $\}$ を計算する．

(2-3) 各 $j\in[1,n]$ および各遷移 $s\xrightarrow{a}_{\mathrm{in}}s_1$（ただし $a\notin B^j$）について，$s'\in(CL_\tau(s)\cup CL_\tau(s_1))\cap S_{\mathrm{E}}^j(s)$ である s' が存在するか調べる．s' が存在しなければ，遷移 $s\xrightarrow{i^a_{s,s_1}}_{1,j}s_1$ を ε 遷移に置き換える．

アルゴリズム (2-1) は，L_{in} 中の τ 遷移を深さ優先探索で辿りながら，遷移先の τ 閉包の要素を遷移元の τ 閉包に加える（和集合を計算する）ことで実現できる[*1]．このとき，和集合の計算（$O(|S|)$ 時間）が，τ 遷移の数に比例する回数行われるので，時間計算量は $O(|S|\cdot|\rightarrow_{\mathrm{in}}|)$ となる．アルゴリズム (2-2) は，各 $j\in[1,n]$ それぞれにつき $O(|\rightarrow_{\mathrm{in}}|)$ 時間で実行できるので，全体の時間計算量はその n 倍となる．アルゴリズム (2-3) は，各 j および $s\xrightarrow{a}_{\mathrm{in}}s_1$ それぞれについて s' が存在するかどうか $O(|S|)$ 時間で調べられるので，全体の時間計算量は $O(n\cdot|\rightarrow_{\mathrm{in}}|\cdot|S|)$ となる．以上より，ステップ (2) 全体の時間計算量は $O(n\cdot|\rightarrow_{\mathrm{in}}|\cdot|S|)$ となる．

ステップ (3) は，各 $j\in[1,n]$ について L_{out2}^j 中の ε 遷移 $s_1\xrightarrow{\varepsilon}_{2,j}s_2$ を深さ優先探索で辿りながら，s_2 から出ている遷移を s_1 からも出るようにする（s_1 の隣接リストに加える）ことで実現できる[*2]．このとき，L_{out2}^j における各遷移が，最大 ε 遷移の数に比例する回数コピーされる．従って，各 j それぞれについて $O(|\rightarrow_{\mathrm{in}}|^2)$ 時間で実行でき，ステップ (3) 全体ではその n 倍となる．

以上より，ステップ (1)–(3) 全体の時間計算量は $O(n\cdot|\rightarrow_{\mathrm{in}}|^2)$ となる．

3.4 手法の改良

3.4.1 インタリーブ領域の簡単化

並列システムで, 複数の入力パラメータを必要とする処理を行うとき, そのパラメータの入力順を無視できることがよくある．例えば，最近の切符販売機では，行き先ボタンの選択とコインの投入はどちらを先におこなってもよいようになっている．システムが状態 s において a,b という入力イベント系列および x という入力イベントを待っているとする. この仕様をあらわす L_{in} は図 3.8(a) になる. この LTS の状態 s から状態

[*1]正確には，先に τ 遷移が作る強連結成分を 1 状態にまとめることによって τ 遷移による閉路を取り除いてから，深さ優先探索を行う．

[*2]τ 閉包の計算と同様に，先に ε 遷移が作る強連結成分を 1 状態にまとめてから深さ優先探索を行う．

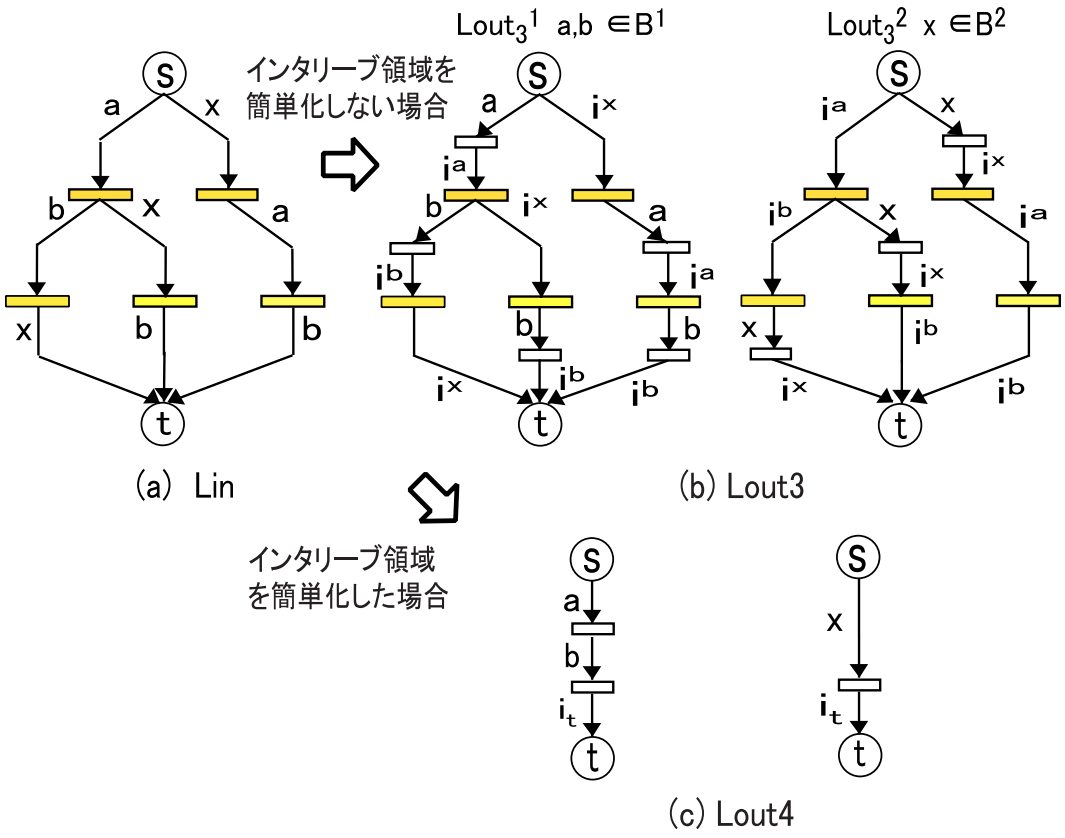


図 3.8 インタリーブ領域の簡単化

t までの領域をインタリーブ領域とよぶことにする. a,b と x が別のプロセスで実行される場合, もし, インタリーブ領域を簡単化せずに L_{out3} に変換すると図 3.8(b) のようになる. 図 3.8(c) のように簡単化できれば, システムの総状態数を削減できる. ここではこのインタリーブ領域を簡単化する方法について考察する.

任意の集合 Σ に対し，Σ の要素からなる有限の系列 $\langle a_1,\ldots,a_n\rangle$ を Σ 上の系列と呼ぶ. Σ 上の系列すべてからなる集合を Σ^* と書く．$\Sigma\subseteq Evt$ のとき，Σ 上の系列をイベント系列と呼ぶ．任意の系列 α において, 集合 A の要素である記号を残し他の記号を削除した系列を $\alpha\uparrow A$ と書く. 例えば　$\langle a,x,b,c,c,y,x,z\rangle\uparrow\{a,b,c\}=\langle a,b,c,c\rangle$. 空系列（長さ 0 の系列）を $\langle\rangle$ と書く．2 つの系列 α,α' の連接を $\alpha\hat{}\alpha'$ と書く．$\alpha=\langle a\rangle\hat{}\alpha'$ のとき, $head(\alpha):=a$, $tail(\alpha):=\alpha'$ と定義する．系列 α の長さを $|\alpha|$ と書く．$\forall k\in[1,|\alpha|]$ について，α の第 k 番の記号を $\alpha^{[k]}$ と書く．系列 $\alpha=\langle a_1,\ldots,a_m\rangle$ に対し，集合 $\{a_1,\ldots,a_m\}$ を $events(\alpha)$ と書く.

定義 11 遷移関係 $\rightarrow$ に対し，系列に対する遷移関係 $\rightarrow^*\subseteq S\times Evt^*\times S$ を，以下を満たす最小の関係と定義する.

$$
\begin{aligned}
&s\xrightarrow{\langle\rangle}{}^* s,\\
&s\xrightarrow{a}s'\wedge s'\xrightarrow{\alpha'}{}^* t\wedge a=head(\alpha)\wedge\alpha'=tail(\alpha)\\
&\quad\supset\ s\xrightarrow{\alpha}{}^* t.
\end{aligned}
$$

□

定義 12 任意のイベント系列 α およびイベント系列の系列 Γ に対して, 述語 $\alpha\,interleaves\,\Gamma$ を以下のように定義する [1].

$\langle\rangle\,interleaves\,\langle\alpha_1,\ldots,\alpha_m\rangle:\Leftrightarrow$
　$(\alpha_1=\langle\rangle\wedge\cdots\wedge\alpha_m=\langle\rangle)$,
$(\langle a\rangle\hat{}\alpha')\,interleaves\,\langle\alpha_1,\ldots,\alpha_m\rangle:\Leftrightarrow$
　for $\exists j\in[1,m]$, $\alpha_j\neq\langle\rangle\ \wedge\ head(\alpha_j)=a\wedge$
　$\alpha'\,interleaves\,\langle\alpha_1,\ldots,tail(\alpha_j),\ldots,\alpha_m\rangle$.
ただし，$a\in Evt$, $\alpha'\in Evt^*$. □

定義 13 LTS $L=(A,S,\rightarrow,s_0,C)$ に対する状態遷移グラフ $G(L)=(V,E)$ は,

$$
\begin{aligned}
&V:=S,\\
&E:=\{(s,t)\in S\times S\mid s\xrightarrow{a}t\ \text{ for }\exists a\in A\}.
\end{aligned}
$$

で定義される有向グラフである. □

定義 14 有向グラフ G におけるパス $\langle v_1,\ldots,v_m\rangle$ が単純であるとは，任意の相異なる $j,k\in[1,m]$ について，$v_j\neq v_k$ であることである. □

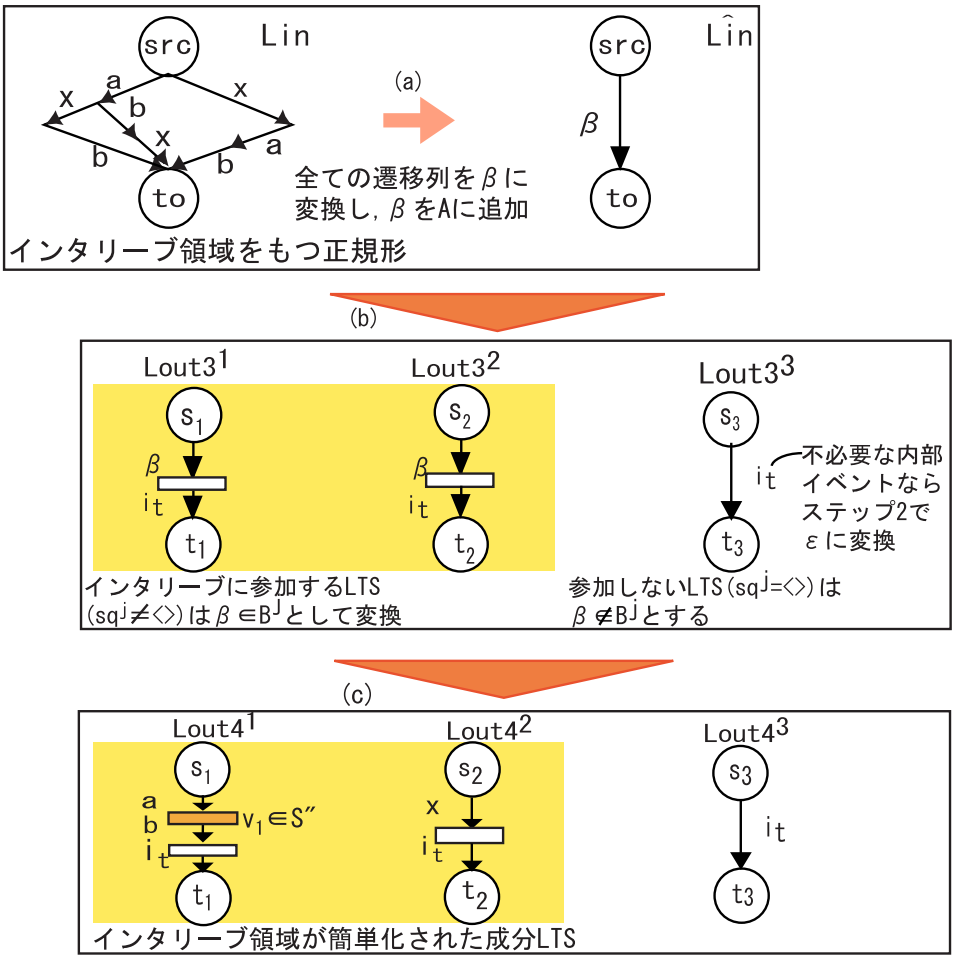


図 3.9 インタリーブ領域の簡単化の手順

定義 15 有向グラフ $G=(V,E)$ および相異なる 2 頂点 $s,t\in V$ について，G の部分グラフ $G|_s^t$ を以下のように定義する．

$$
\begin{aligned}
&G|_s^t := (V',E')\\
&\text{where}\\
&\quad V' := \{v\in V \mid t \text{ を含まないような } s \text{ から}\\
&\qquad v \text{ へのパスが存在，かつ } s \text{ を含まない}\\
&\qquad \text{ような } v \text{ から } t \text{ へのパスが存在 }\},\\
&\quad E' := \{(u,v)\in E \mid u,v\in V'\}.
\end{aligned}
$$

□

定義 16 定義 1 の $L_{\text{in}}=(A,S,\rightarrow_{\text{in}},s_0,\emptyset)$，$n$ および B^j について考える．相異なる 2 状態

$s,t \in S$ に対し，系列の集合 $T_{s,t}$ を以下のように定義する．

$$T_{s,t} := \{\alpha \in A^* \mid s \xrightarrow{\alpha}{}^*_{\text{in}} t\}.$$

$T_{s,t}$ が以下の条件を満たすとき，部分グラフ $G(L_{\text{in}})|^t_s$ をインタリーブ領域と呼ぶ．

- あるイベント系列 $sq_1,\ldots,sq_n \in A^*$ が存在し，

  $$T_{s,t} = \{\alpha \in A^* \mid \alpha \mathit{interleaves} \langle sq_1,\ldots,sq_n\rangle\}.$$

  ただし，任意の $j \in [1,n]$ について，

  $$\begin{aligned} &\mathit{events}(sq_j) \subseteq B^j \\ &\quad \wedge \mathit{events}(sq_j) \cap B^k = \emptyset \ \text{ for } \forall k \neq j. \end{aligned}$$

- $G(L_{\text{in}})|^t_s = (V,E)$ は決定性である．すなわち，任意の頂点 $u \in V$ およびイベント a について，$u \xrightarrow{a}_{\text{in}} u'$ かつ $u \xrightarrow{a}_{\text{in}} u''$ であるような相異なる 2 頂点 $u',u'' \in V$ は存在しない．

- 状態遷移グラフ $G(L_{\text{in}})$ が以下の条件を満たす．

  – 任意の $s' \in S$ について，t を含まないような s から s' へのパスが存在するならば，s' から t へのパスが存在．

  – 任意の $s' \in S$ について，s を含まないような s' から t へのパスが存在するならば，s から s' へのパスが存在． □

入力となる LTS を $L_{\text{in}} = (A, S_{\text{orig}}, \rightarrow_{\text{in}}, s_0, \emptyset)$ とする．ある 2 状態 $src,to \in S_{\text{orig}}$ について，部分グラフ $G(L_{\text{in}})|^{to}_{src} = (V,E)$ がインタリーブ領域であると仮定する．このとき，以下のアルゴリズムを用いることができる．

(1) 新しいイベント β を導入し，src から to への遷移系列の集合を，β によるひとつの遷移に置き換える (図 3.9(a))．こうして得られた LTS を $\hat{L_{\text{in}}}$ とする．すなわち

$$\begin{aligned} &\hat{L_{\text{in}}} := (A \cup \{\beta\}, S, \rightarrow_{\hat{\text{in}}}, s_0, \emptyset) \\ &\text{where} \\ &\quad S := (S_{\text{orig}} - V) \cup \{src, to\}, \\ &\quad u \xrightarrow{a}_{\hat{\text{in}}} u' :\Leftrightarrow (u \neq src \wedge u \xrightarrow{a} u') \vee \\ &\qquad (u = src \wedge a = \beta \wedge u' = to) \end{aligned}$$

また，以下のように $\hat{B^j}$ を決める．

$$\hat{B^j} := \begin{cases} B^j \cup \{\beta\} & \text{if } sq_j \neq \langle\rangle, \\ B^j & \text{otherwise.} \end{cases}$$

(2) 3.3 節のアルゴリズムを $\hat{L_{\text{in}}}, \hat{B^j}$ に適用し，L^j_{out3} を得る (図 3.9(b))．ただし，$\beta \in \hat{B^j}$ である任意の $j \in [1,n]$ については，L^j_{out2} の定義において，同期イベント i^{β}_{to} が $\mathit{Disabling}^j \cup \mathit{Enabling}^j$ に含まれるものとして扱う（ε に変換しない）．このようにしても 3.3.3 節の補題・定理には影響しない．

(3) L^j_{out3} における β による遷移を，系列 sq_j による遷移系列に変換する．こうして得られた LTS を L^j_{out4} とする (図 3.9(c))．すなわち，$L^j_{\text{out3}} = (A^j_3, S \cup S_{\text{aux}}, \rightarrow_{3,j}, s_0, C^j_3)$ に対し，$\beta \notin \hat{B^j}$ である j については $L^j_{\text{out4}} = L^j_{\text{out3}}$ とする．$\beta \in \hat{B^j}$ である j について，以下のように L^j_{out4} を定義する．

$$
\begin{array}{l}
L^j_{\text{out4}} := (A^j_4, S \cup S_{\text{aux}} \cup S^j_{\text{int}}, \rightarrow_{4,j}, s_0, C^j_4) \\
\text{where} \\
\quad A^j_4 := A^j_3 - \{\beta\}, \\
\quad S^j_{\text{int}} := \{V_1, V_2, \ldots, V_{|sq_j|-1}\} : \\
\quad |sq_j| - 1 \text{ 個の新しい状態からなる集合.} \\
\quad u \xrightarrow{\gamma}_{4,j} u' :\Leftrightarrow \\
\quad (\beta \neq \gamma \wedge u \xrightarrow{\gamma}_{3,j} u') \vee \\
\quad (u \xrightarrow{\beta}_{3,j} \cdot \wedge \gamma = sq^{[1]}_j \wedge u' = v_1) \vee \\
\quad (u = v_k \in S_{\text{int}} \wedge k < |sq_j| - 1 \wedge \gamma = sq^{[k+1]}_j \wedge u' = v_{k+1}) \vee \\
\quad (u = v_k \in S_{\text{int}} \wedge k = |sq_j| - 1 \wedge \gamma = sq^{[k+1]}_j \wedge src \xrightarrow{\beta}_{3,j} u') \\
\quad C^j_4 := C^j_3 - \{(u, \beta, t, i^{\beta}_{to}) \mid \\
\quad u \in S \cup S_{\text{aux}}, t \in S_{\text{aux}}, (u, \beta, t, i^{\beta}_{to}) \in C^j_3\}.
\end{array}
$$

$L_{\text{out4}} := (L^1_{\text{out4}} \times \cdots \times L^n_{\text{out4}}) \backslash A_{\text{aux}}$ とし，これを出力する． □

上の定義で S_{int} はイベント β による遷移を系列 sq_j へ変換するときに追加される新しい状態の集合である．例えば，$sq_1 = \langle a, b \rangle, sq_2 = \langle x \rangle$ とすると，L^1_{out4} では $|sq_1| = 2$ で $S_{\text{int}} = \{v_1\}$ となり，$src \xrightarrow{a}_{4,1} v_1 \xrightarrow{b}_{4,1} u \xrightarrow{i^{\beta}_{to}}_{4,1} to$ (ここで $src \xrightarrow{\beta}_{3,1} u$) となる．

このアルゴリズムによって得られる LTS L_{out4} は一般に，3.3 節のアルゴリズムで得られる LTS に較べて，状態数が少なく，同期イベントによる同期が必要な箇所も少なくなる場合が多い．すなわち，LTS を実現した際の動作効率がよいと考えられる．

定理 2 $L_{\text{in}} \approx L_{\text{out4}}$.

（証明の方針）$R \subseteq (S \cup V) \times (S \cup S_{\mathrm{aux}} \cup S_{\mathrm{int}})^n$ を以下のように定義する．

$$
\begin{aligned}
&R := R_1 \cup R_2 \cup R_3 \\
&\text{where} \\
&\quad R_1 := \{(s,(s_1,\ldots,s_n)) \mid \\
&\qquad s \in S, s \in CL^j_\varepsilon(s_j) \text{ for } \forall j \in [1,n]\} \\
&\quad R_2 := \{(s,(t_1,\ldots,t_n)) \mid \\
&\qquad s \in S, \exists i \in A_{\mathrm{aux}} - A_\tau,\ \forall j \in [1,n], \\
&\qquad \text{if } i \in A^j_2 \text{ then } (t_j \xrightarrow{i}_{3,j} s_j) \wedge s \in CL^j_\varepsilon(s_j) \\
&\qquad \text{for } \exists s_j \text{else } s \in CL^j_\varepsilon(t_j)\}. \\
&\quad R_3 := \{(u,(u_1,\ldots,u_n)) \mid \\
&\qquad u \in V,\ src \xrightarrow{\alpha}{}^*_{\mathrm{in}} u \text{ for } \exists \alpha \in A^*, \\
&\qquad \text{for } \forall j \in [1,n], \alpha'_j = \alpha \uparrow B^j \wedge \\
&\qquad \text{if } \alpha'_j = \langle\rangle \text{ then } src \in CL^j_\varepsilon(u_j) \text{ else } src \xrightarrow{\alpha'_j}{}^*_{4,j} u_j\}.
\end{aligned}
$$

R が $(L_{\mathrm{in}}, L_{\mathrm{out4}})$ に対する弱双模倣関係であることを示せばよい．以降では，L_{out4} の遷移関係を $\rightarrow_4$ とする．

(a) R が定義 10 の条件 (a) を満たすことを示す．

(1) $(s,(s_1,\ldots,s_n)) \in R_1$ について考える．$s \neq src$ のときは，$s \xrightarrow{a}_{\mathrm{in}} s'$ ならば $s' \notin V$ なので，定理 1 の証明の (a)(1) と同様．$s = src$ のときは $(s,(s_1,\ldots,s_n)) \in R_3$ なので，下の (3) で示す．

(2) $(s,(t_1,\ldots,t_n)) \in R_2$ については定理 1 の証明の (a)(2) と同様．

(3) $(u,(u_1,\ldots,u_n)) \in R_3$ について考える．$u = to$ ならば $(u,(u_1,\ldots,u_n)) \in R_2$ であるので（R_2 の定義の i を i^β_{to} とすればよい），(2) ですでに示した．
$u \neq to$ とし，$u \xrightarrow{a}_{\mathrm{in}} u'$ である任意の $a \in A, u' \in S_{\mathrm{orig}}$ について考える．このとき，$u \in V$ であり，$src \xrightarrow{\alpha\hat{}\langle a\rangle}{}^*_{\mathrm{in}} u'$ である．定義 16 より，$a \in B^j$ である j がちょうど 1 つ存在し，$src \xrightarrow{(\alpha\hat{}\langle a\rangle)\uparrow B^j}{}^*_{4,j} u'$ という遷移系列および状態 u'_j が存在する．このとき，$u_j \xrightarrow{a}_{4,1} u'_j$ である．この j 以外の $k \in [1,n]$ について $u'_k = u_k$ と定義すると，$(u_1,\ldots,u_n) \xrightarrow{a}_4 (u'_1,\ldots,u'_n)$ かつ $(u',(u'_1,\ldots,u'_n)) \in R_3$ が言える．

(b) R が定義 10 の条件 (b) を満たすことを示す．

(1) $(s,(s_1,\ldots,s_n)) \in R_1$ について，上の (a)(1) と同様に，定理 1 の証明の (b)(1) あるいは下の (3) に帰着できる．

(2) $(s,(t_1,\ldots,t_n)) \in R_2$ について，R_2 の定義の i が i^β_{to} であるときに $(t_1,\ldots,t_n) \xrightarrow{a}_4 \cdot$ となる $a \in A$ は存在しないので，いずれの i についても定理 1 の証明の (b)(2) と同様に示せる．

(3) $(u,(u_1,\ldots,u_n)) \in R_3$ について考える．上の (a)(3) と同様に $u \neq to$ の場合のみ考えればよい．

$u \neq to$ とし，$(u_1,\ldots,u_n) \xrightarrow{\gamma}_4 (u'_1,\ldots,u'_n)$ である任意の $\gamma \in Evt$, $(u'_1,\ldots,u'_n) \in (S \cup S_{\mathrm{aux}} \cup S_{\mathrm{int}})^n$ について考える．$u \neq to$ より $\beta \in \hat{B}^j$ かつ $u_j \not\xrightarrow{i^{\beta}_{to}}_{4,j} \cdot$ である j が存在する．従って，$\gamma \neq \tau(i^{\beta}_{to})$ である．よって $\gamma = a \in A$ とする．このとき $a \in B^k$ である $k \in [1,n]$ がちょうど 1 つ存在し，$u_k \xrightarrow{a}_{4,k} u'_k$ である状態 u'_k が存在する．k 以外の $j \in [1,n]$ について $u'_j = u_j$ と定義する．また，R_3 の定義中の α に対し，$src \xrightarrow{\hat{\alpha}\langle a \rangle}_{\mathrm{in}} u'$ という遷移系列および状態 $u' \in V$ が存在する．$GL_{\mathrm{in}}|^{to}_{src}$ が決定的であることから，この遷移系列は必ず $src \xrightarrow{\alpha}_{\mathrm{in}}{}^{*} u \xrightarrow{a}_{\mathrm{in}} u'$ の形をしている．従って，$u \xrightarrow{a}_{\mathrm{in}} u'$ かつ $(u',(u'_1,\ldots,u'_n)) \in R_3$ が言える．

□

3.4.2 同期イベントの共通化

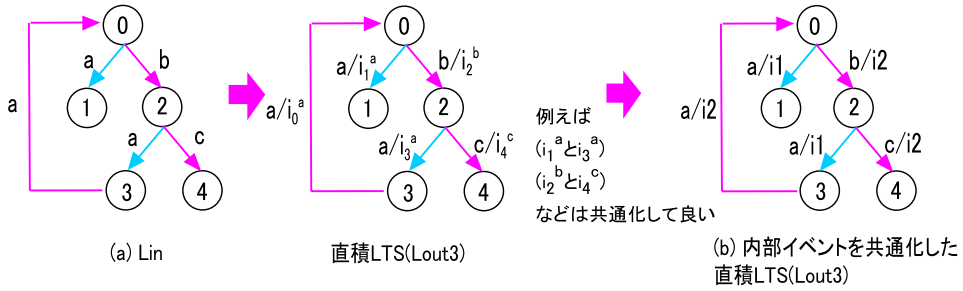


図 3.10 同期イベントの共通化

成分プロセス間で通信される同期イベントの種類を少なくできれば，通信する情報量が減少するので，通信がより効率的に行える．2 つの同期イベント i^a_s, i^b_t が次の条件を満たすとき，この 2 つの同期イベントを共通化することが可能である．

条件：

(1) L_{in} において，$s \xrightarrow{a}_{\mathrm{in}} u$ かつ $s \xrightarrow{b}_{\mathrm{in}} t$ である s が存在しない．

(2) 全ての $j \in [1,n]$ について，$s \xrightarrow{i^a_u}_{3,j} \cdot$ かつ $s \xrightarrow{i^b_u}_{3,j} \cdot$ である $s \in S \cup S_{\mathrm{aux}}$ が存在しない．

(3) 全ての $j \in [1,n]$ について，i^a_u と i^b_t はともに A^j_2(定義 8 参照) に含まれるか，またはどちらも A^j_2 に含まれない．

例えば図 3.10(a) のような L_{in} では, 状態 0 において $0 \xrightarrow{a}_{\text{in}} 1$ であるが, $0 \not\xrightarrow{a}_{\text{in}}$ である. 一方, 状態 2 においては $2 \xrightarrow{a}_{\text{in}} 3$ であるが, $2 \not\xrightarrow{a}_{\text{in}} 1$ である. 他の状態からはこの 2 つの同期イベントに関係する遷移がなく, 条件 (2)(3) も満たすので, (i_1^a と i_3^a) は共通化できる. 同様に (i_2^b と i_4^c) も共通化できる. この例で共通化可能な全ての同期イベントを共通化し, 同期イベントにつけられている不要なラベルを消去すると図 3.10(b) のようになる.

3.5 考察

以下では, 連続イベント制約のない通常の遷移システムでプロセスの動作定義を与えるとした場合を考察する. これは, 2.2 節の各定義において, 連続イベント制約が常に空であると仮定した場合に相当する. [10,11,13,14] 等で行われているように以下の仮定をすることで, この場合でも提案アルゴリズムを適用することができる.

L_{in}, B^j $(1 \leq j \leq n)$ は仮定 2 および, 以下の仮定を満たすとする.

仮定 17 任意の $s, s_1, s_2 \in S$, $a, b \in A - \{\tau\}$, $j \in [1, n]$ について, $s \xrightarrow{a}_{\text{in}} s_1 \wedge s \xrightarrow{b}_{\text{in}} s_2$ ならば, $a \in B^j \Leftrightarrow b \in B^j$. □

もし L_{in} が仮定 17 を満たさない場合, 仮定 2 の場合と同様の方法で, 仮定を満たしかつ元の L_{in} と弱双模倣等価であるような LTS を得ることができる [10,11].

上記の仮定を満たす L_{in}, B^j に対し, 3.3 節のアルゴリズムを適用する. ただし, 連続イベント制約の生成は行わない. このようにして得られた L_{out3} が L_{in} と弱双模倣等価であることを示すことができる（付録 A 参照）.

上記のように, 仮定 17 をみたさない L_{in} が与えられた場合, 仮定を満たすように変形するためには内部イベント τ による遷移を余分に追加する必要がある. これは実装レベル仕様において, 成分 LTS 間の A_τ の要素による同期を増やすこととなる. 連続イベント制約を仮定できる状況では, 連続イベント制約を用いたほうが, 一般に成分 LTS 間の同期数を少なくできる. 連続イベント制約を仮定できるようなシステムとしては図 3.11 のような例が考えられる. 仮定として LTS の各イベントによる遷移は時間 T_d 内に完了できるとする. このシステムは環境との通信にハンドシェイク機能を持ち, 例えば a_{in} から信号を入力した場合, その入力に対して受信完了通知または受信拒否通知を a_{out} から返す. 環境は受信完了通知を受けたときのみ, そのイベントが発生したと解釈する. また, enable はクロック信号で, この信号が 1 なら環境から信号を受信でき, 0 の間は受信できない. 0 の期間は $2 \cdot T_d$ 以上続く信号とする. このシステムにおいて $enable = 1$ である時間内に複数の信号を入力した場合, そのうち 1 つの入力のみシステム内に通知し, その入力に対して受信完了通知を出力し, その他の入力には受信拒否通知を出力する. その後, 少なくとも時間 $2T_d$ たってから, 再び $enable = 1$ となる. 例えば環境からの入力 a_{in}, b_{in} のうち a_{in} が選択されたとすると, 仮定より再び enable=1 となった時点では, システム内のプロセスはイベント a およびそ

れに伴う同期イベントによる遷移を完了している．従って，このシステムでは連続イベント制約は満たされる．図 3.11 のような入力の選択機構はプライオリティエンコーダ [15] 等によって実現することが可能である．

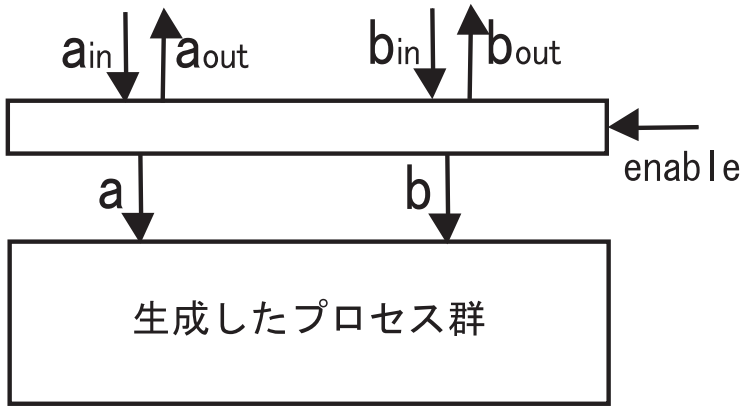


図 3.11 連続イベント制約が仮定できるシステム例

3.6 ３章のまとめ

この章では, 要求仕様および, 実装レベル仕様が持つべきプロセス数と各成分プロセスが用いる外部イベント集合（アルファベット）が与えられたとき, それらを満たす実装レベル仕様を導出する方法を示した. また, 導出した直積 LTS と入力 LTS 間の弱双模倣等価性を証明した. さらに, インタリーブ領域を簡単化して総状態数を小さくする方法や, システム内部で用いられる同期イベントを共通化する方法を示した.

第4章　分解法の一般化とカウンタプロセス分解への応用

4.1　はじめに

この章では，イベントの発生回数の相関関係に着目した分解法を提案する．提案法の主な利点は，LTS のサイズがしばしば従来の分解法 (文献 [10], 本論文第 3 章) より小さくなることである．本章では，提案法によって入力 LTS が分解できるための十分条件を示す．また，いくつかの分解例を示す．

特に本章では，カウンタプロセス $CNT_n(a,b)$（ここで a はカウントアップイベント，b はカウントダウンイベントそして n は容量を表す）に焦点をあてる．提案手法によって $CNT_n(a,b)$ を $CNT_m(a,i)$ と $CNT_{n-m}(i,b)$ に分解できる．ただし m は $1 \leq m < n$ である任意の整数で i は同期のための新しいイベントである．

最後に，$CNT_n(a,b)$ を拡張した LTS の部分クラス $GCNT_n(a,b)$ を導入する．$GCNT_n(a,b)$ の要素である LTS は $CNT_n(a,b)$ ように振舞うと同時に，その他の動作（すなわち a,b 以外のイベントによる遷移）も行うプロセスである．本論文では $GCNT_n(a,b)$ に対する分解法も示す．

4.2　分解法

以下の入出力によって LTS 分解問題を定義する．

定義 13 (LTS 分割問題)

入力: L_{in}　有限状態の LTS.

$\widetilde{\Sigma} = (B_1, \ldots, B_n)$　$B_1 \cup \cdots \cup B_n = A[L_{\text{in}}] - \{\tau\}$ を満たすイベントの部分集合の組.

出力: $L_1, \ldots, L_n$　有限状態の LTS の組．ただし，$i \in [1,n]$ について $A[L_i] - A_{\text{sync}} = B_i$．そして $(L_1 \times \cdots \times L_n) \backslash A_{\text{sync}} \approx L_{\text{in}}$．ここで $A_{\text{sync}} = Evt - (A[L_{\text{in}}] \cup \{\tau\})$．　□

すなわち，LTS 分解問題は，与えられた LTS L_{in} と和集合が $A[L_{\text{in}}] - \{\tau\}$ であるイベントの部分集合の組 $\widetilde{\Sigma} = (B_1, \ldots, B_n)$ から，以下のような LTS 群 $L_1, \ldots, L_n$ を構成することである：

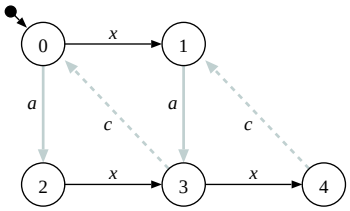
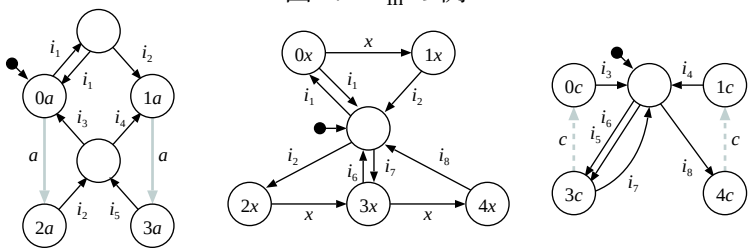
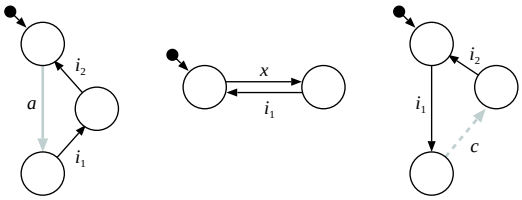


図 4.1 L_{in} の例



(a)



(b)

図 4.2 図 4.1 の L_{in} に対する出力の例

- L_i $(1 \leq i \leq n)$ は，L_j $(i \neq j)$ との同期のために，$A[L_{\text{in}}] \cup \{\tau\}$ の要素でない新しいイベントを使用できる．
- $L_1, \ldots, L_n$ を並列合成し，同期イベントを隠蔽した LTS $(L_1 \times \cdots \times L_n) \backslash A_{\text{sync}}$ は L_{in} と弱双模倣等価になる．

もし A_{sync} 内のイベントを同期のために利用しないで，かつ $\tau \notin A[L_{\text{in}}]$ であるなら，この分解問題は [8] における loosely cooperating system に対する弱双模倣等価性の下での分解問題と等価である．

例 1 図 4.1 の L_{in} と $\widetilde{\Sigma} = (\{a\}, \{x\}, \{c\})$ が与えられたとする．図 4.2 (a) と (b) は，この入力の出力例である．ここで $\{i_1, \ldots, i_8\} \subseteq A_{\text{sync}}$ であり，図 4.2 (a) では，$(A[L_1], A[L_2], A[L_3]) = (\{a, i_1, i_2, i_3, i_4, i_5\}, \{x, i_1, i_2, i_6, i_7, i_8\}, \{c, i_3, i_4, i_5, i_6, i_7, i_8\})$，そして図 4.2 (b) では $(\{a, i_1, i_2\}, \{x, i_1\}, \{c, i_1, i_2\})$ である． □

以下，定義 13 で述べた $L_{\text{in}}, n, \widetilde{\Sigma} = (B_1, \ldots, B_n)$ を固定する．また $A_{\text{sync}} = Evt - (A[L_{\text{in}}] \cup \{\tau\})$ とする．分解問題を解く手法として以下のような手法を提案する．

定義 14 (分解法)

Step 1: (p1) から (p4) の条件を満たす有限状態の L_{inx}, イベントの部分集合の組 $\widetilde{\Sigma}_{\mathrm{x}} = (A_1, \ldots, A_n)$，そして $S[L_{\mathrm{inx}}]$ 上の同値関係の組 $Eq = (\sim_{A_1}, \ldots, \sim_{A_n})$ をみつける．

(p1) $B_i \subseteq A_i \subseteq (B_i \cup A_{\mathrm{sync}})$ for each $i \in [1,n]$.

(p2) $A_1 \cup \cdots \cup A_n = A[L_{\mathrm{inx}}]$.

(p1) と (p2) から $\tau \notin A[L_{\mathrm{inx}}]$ が言える．

(p3) 条件 (p3-1) から (p3-4) を満たす全射 $R : S[L_{\mathrm{inx}}] \to S[L_{\mathrm{in}}]$ が存在する (すなわち L_{inx} の各状態は L_{in} の 1 つの状態と対応し，L_{in} の各状態が少なくとも 1 つの L_{inx} の状態と対応する)

(p3-1) $I[L_{\mathrm{in}}] = R(I[L_{\mathrm{inx}}])$.

(p3-2) If $s_1 \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s_2$ and $\alpha \notin A_{\mathrm{sync}}$, then $R(s_1) \xrightarrow{\alpha}_{L_{\mathrm{in}}} R(s_2)$.

(p3-3) If $s_1 \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s_2$ and $\alpha \in A_{\mathrm{sync}}$, then $R(s_1) = R(s_2)$ or $R(s_1) \xrightarrow{\tau}_{L_{\mathrm{in}}} R(s_2)$.

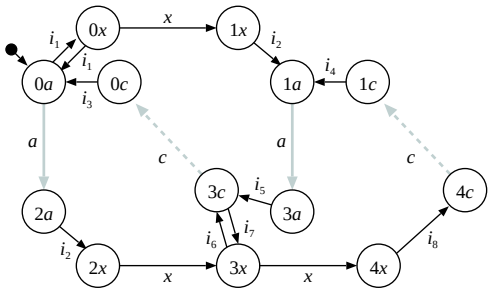
(p3-4) If $s_1 \xrightarrow{\alpha}_{L_{\mathrm{in}}} s_2$, then for each $s_1' \in S[L_{\mathrm{inx}}]$ such that $R(s_1') = s_1$,
$S[L_{\mathrm{inx}}]$ 内に以下の 3 つの性質を満たす状態 s_1''，s_2' が存在する．

- $R(s_2') = s_2$.
- 遷移関係 $\rightarrow_{L_{\mathrm{inx}}}$ をイベントの系列に対して自然に拡張する．A_{sync} 内のイベントのみからなる長さ 0 以上の系列 γ が存在し $s_1' \xrightarrow{\gamma}_{L_{\mathrm{inx}}} s_1''$ である．
- If $\alpha \neq \tau$, then $s_1'' \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s_2'$. Otherwise, $s_1'' = s_2'$.

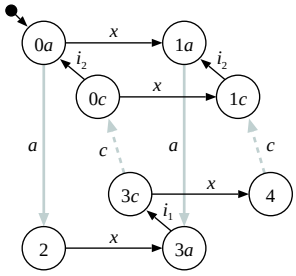
(p4) 各 $i \in [1,n]$ について, 関係 $\sim_{A_i}$ は定義 7 の (c2) を満たす．各 $s \in S[L_{\mathrm{inx}}]$ と $\alpha \in A[L_{\mathrm{inx}}]$ について，もし，全ての $i \in loc_{\widetilde{\Sigma}_{\mathrm{x}}}(\alpha)$ に対して $s \sim_{A_i} s_i$ かつ $s_i \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s_i'$ であるような s_i と s_i' が存在するなら，そのような s_i と s_i' の任意の選択について, $s \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s'$ かつ，全ての $i \in loc_{\widetilde{\Sigma}_{\mathrm{x}}}(\alpha)$ について $s' \sim_{A_i} s_i'$ である s' が存在する．

Step 2: 各 $i \in [1,n]$ について $L_i = L_{\mathrm{inx}}/_{\sim_{A_i}}$ を定義する (これは $A[L_i] = A_i$ を意味する). □

条件 (p1),(p2) は出力 LTS $L_i (1 \leq i \leq n)$ のアルファベット A_i が B_i を含むことを保証し，条件 (p3) は $L_{\mathrm{inx}} \backslash A_{\mathrm{sync}} \approx L_{\mathrm{in}}$ を保証する．また，条件 (p4) は $L_{\mathrm{inx}} \approx L_1 \times \cdots \times L_n$ を保証する（以下に示す補題 5 と 6 を参照）．また，条件 (p3) は，L_{in} の各状態を，同期イベントで互いに接続された 1 つ以上の状態に分割することで L_{inx} が得られることを言っている．条件 (p4) は [8] における定理 4.1 の条件 (*i*) と (*iii*) に対応する．これは，同形性の下で同期イベントを用いずに分解可能であるための条件を示している．条件 (p1) から (p4) を満たす組 $(L_{\mathrm{inx}}, \widetilde{\Sigma}_{\mathrm{x}}, Eq)$ を見つけたとき，Step 2 によって自動的に正しい出力 $L_1, \ldots, L_n$ が得られる．一般に, 同じ入力に対して，条件 (p1) から (p4) をみたす L_{inx} は無限に存在する．そしてそれらから得られる出力のサイズは一般に異なる．



(a)



(b)

図 4.3 図 4.1 の L_{in} に対する L_{inx} の例

例えば, 図 4.1 の L_{in} に対して図 4.3 (a) および (b) の L_{inx} は性質 (p1) から (p4) を満たす. ここで $\sim_{A_i}$ は (c2) を満たす最小の同値関係とする．そしてこれらから図 4.2 (a) および (b) が出力として得られる．本論文第 3 章において, 我々は Step 1 の実装であるアルゴリズムを提案している. そのアルゴリズムは任意の入力 $(L_{\text{in}}, \widetilde{\Sigma})$ に対して $(L_{\text{inx}}, \widetilde{\Sigma}_{\text{x}}, Eq)$ を導出するが，図 4.1 の L_{in} のような入力に対しては図 4.3 (a) を出力する．つまり，そのアルゴリズムは，$\widetilde{\Sigma}$ において $\alpha \| \beta$ であるような任意のイベント α, β について $s \xrightarrow{\alpha}_{L_{\text{inx}}} \cdot$ かつ $s \xrightarrow{\beta}_{L_{\text{inx}}} \cdot$ であるような s および $s_1 \xrightarrow{\alpha}_{L_{\text{inx}}} s_2 \xrightarrow{\beta}_{L_{\text{inx}}} \cdot$. であるような s_1および s_2 が存在しないような L_{inx} を出力する.

分解法の正当性

補題 5 もし条件 (p3) が成り立つなら, $L_{\text{inx}} \backslash A_{\text{sync}} \approx L_{\text{in}}$.
(証明の方針) 条件 (p3) における R が $(L_{\text{inx}} \backslash A_{\text{sync}}, L_{\text{in}})$ の模倣関係になる. □

補題 6 もし条件 (p1), (p2), (p4) が成り立つならば，定義 14 の Step 2 によって得られた $L_1, \ldots, L_n$ は，$L_{\text{inx}} \approx L_1 \times \cdots \times L_n$ を満たす.

(証明) $L_{\text{o}} = L_1 \times \cdots \times L_n$ と置く．以下の関係 $Q \subseteq S[L_{\text{in}}] \times S[L_{\text{o}}]$ が，L_{inx} と L_{o} の間の弱双模倣関係であることを示す.

$$Q = \{(s, (s_1, \ldots, s_n)) \mid s_i = [s]_{\sim A_i} \ (1 \leq i \leq n)\}.$$

QがL_{inx}からL_{o}への模倣関係であることは，縮約の定義から自明．よって，QがL_{o}からL_{inx}への模倣関係であることを示せばよい．

$s, s_1, \ldots, s_n$を，$(s, (s_1, \ldots, s_n)) \in Q$である任意の状態とする．$\alpha$を$\alpha \in A[L_{\mathrm{o}}]$ $(= A[L_{\mathrm{inx}}])$である任意のイベントとし，$(s_1, \ldots, s_n) \xrightarrow{\alpha}_{L_{\mathrm{o}}} (s'_1, \ldots, s'_n)$であるとする．

直積LTSの定義より，$\alpha \in A[L_i]$であるiが少なくとも1つ存在する$(loc_{\widetilde{\Sigma}}(\alpha)$が空でない$)$．また，全ての$i \in loc_{\widetilde{\Sigma}}(\alpha)$について$s_i \xrightarrow{\alpha}_{L_i} s'_i$である．このとき，このような各$i$についてある2状態$t_i, t'_i \in S[L_{\mathrm{inx}}]$が存在し，$t_i \in s_i = [s]_{\sim A_i}$, $t'_i \in s'_i$, かつ$t_i \xrightarrow{\alpha}_{L_{\mathrm{inx}}} t'_i$．よって，条件(p4)より，$s \xrightarrow{\alpha}_{L_{\mathrm{inx}}} u$かつ全ての$i \in loc_{\widetilde{\Sigma}}(\alpha)$について，$u \sim_{A_i} t'_i$(すなわち$u \in s'_i$)である$u \in S[L_{\mathrm{inx}}]$が存在する．

一方，$\alpha \notin A[L_j]$である各jについて，$s_j = s'_j$である．$s \xrightarrow{\alpha}_{L_{\mathrm{inx}}} u$であることから，$u \sim_{A_i} s$が言える．よって，$u \in s_j = s'_j$.

以上より，ある$u \in S[L_{\mathrm{inx}}]$が存在し，$s \xrightarrow{\alpha}_{L_{\mathrm{inx}}} u$，かつ任意の$i \in [1, n]$について$u \in s'_i$である．すなわち$(u, (s'_1, \ldots, s'_n)) \in Q$. □

定理3 定義14で得られた$L_1, \ldots, L_n$は分解問題の正しい出力である． □

定義14はStep 1の具体的なアリゴリズムを述べていない．我々は，より小さい出力を得られるL_{inx}を構成するようなStep 1のアルゴリズムをみつけたい．以下の節ではL_{in}のいくつかのサブクラスに対し，小さい出力を得られるような分解アルゴリズムについて議論する．

4.3 カウンタプロセス

図4.4(a)のようなプリントプロセスを考える．このプロセスのアルファベットを$A = \{print_req, print_out\}$(*print_req*:プリント要求を受けつけるイベント，*print_out*:印刷を実行するイベント)とする．このプロセスが3つまでのプリント要求を印刷を実行する前に受け付けることができるとすると，要求仕様を表すLTSは図4.4(b)のようになる．このプロセスは*print_req*イベントと*print_out*イベントの発生の回数の差を数えているプロセスとみることが出来るので，このようなプロセスをカウンタプロセスと呼ぶ．さて，このプロセスを$\widetilde{\Sigma} = (\{print_req\}, \{print_out\})$と2つのプロセスに分解する場合，図4.4のようなサイズの小さい出力が解として存在する．もし，このように自動分解できるアルゴリズムが存在すれば，パイプライン状に接続される並列プロセスをサイズの小さいLTSに自動分解できるという点で非常に有効である．そこで，本節ではカウンタプロセス分解について考察する．

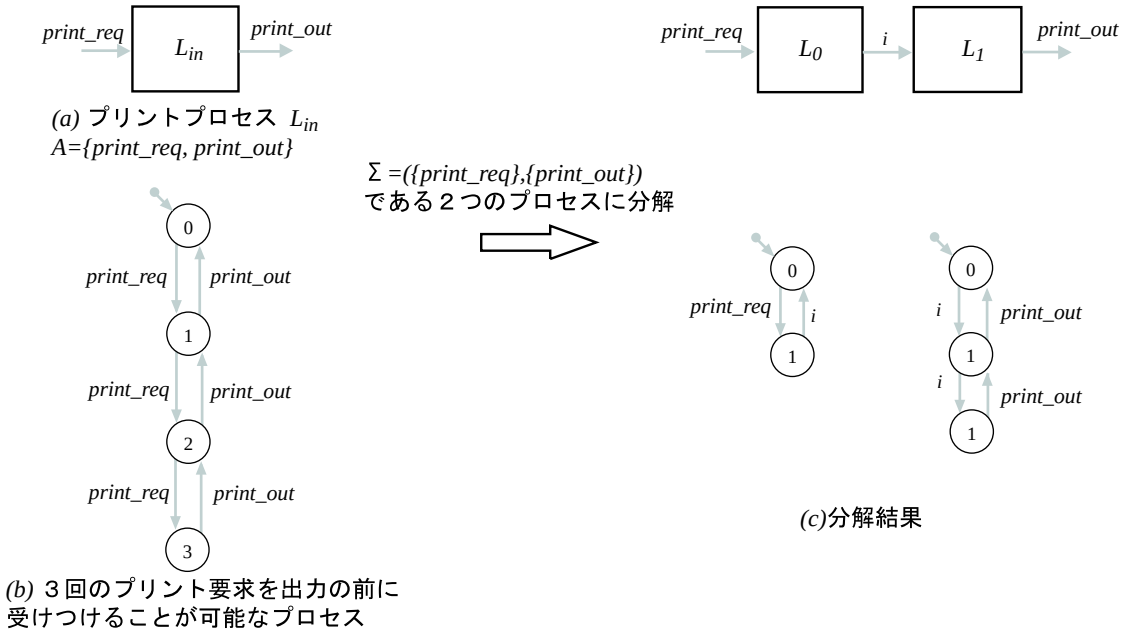


図 4.4 カウンタプロセスの例

4.3.1 単純カウンタ

以下，任意の部分集合 $A \subseteq Evt - \{\tau\}$ について, $\sim_A$ を定義 7 の (c2) を満たす最小の同値関係とする．

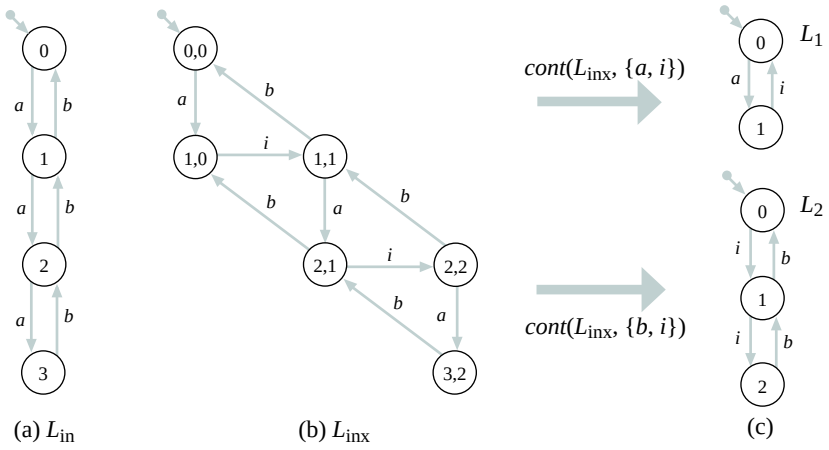
定義 15 (単純カウンタ) $a,b \in Evt - \{\tau\}$ を任意のイベント，m を負でない整数とする．

もし $A[L] = \{a,b\}$ でかつ L が以下の LTS と同形なら，L は，a をカウントアップイベント，b をカウントダウンイベントとする容量 m の**単純カウンタ** である．ここでいう容量 m とはイベント a,b の発生回数の差の最大値を表す値である．$CNT_m(a,b) = (\{a,b\},\{0,1,2,\ldots,m\},\rightarrow,0)$．ただし $\rightarrow = \{(j,a,j+1) \mid j \in [0,m-1]\} \cup \{(j,b,j-1) \mid j \in [1,m]\}$. □

図 4.5 (a) は $CNT_3(a,b)$ を図示したものである. 以下，同形の LTS は区別しないものとする．したがって, $CNT_m(a,b)$ は a，b，m で決定まる唯一の単純カウンタである．

単純カウンタ $L_{\text{in}} = CNT_m(a,b)$ と $\widetilde{\Sigma} = (\{a\},\{b\})$ に対する分解法を考える．L_{in} と 任意の整数 $l \in [1,m-1]$ について, L_{inx} を以下のように構成できる．$L_{\text{inx}} = (\{a,b,i\},\{(y,x) \mid x \in [0,m-l], y \in [x,x+l]\},\rightarrow,0)$．ただし，

$$
\begin{aligned}
\rightarrow = {} & \{((y,x),a,(y+1,x)) \mid x \in [0,m-l],\ y \in [x,x+l-1]\} \\
& \cup \{((y,x),b,(y-1,x-1)) \mid x \in [1,m-l],\ y \in [x,x+l]\} \\
& \cup \{((y,x),i,(y,x+1)) \mid \\
& \quad y \in [1,m-1],\ x \in [\max(0,y-l),\min(y-1,m-l-1)]\}
\end{aligned}
$$



(a) L_{in} (b) L_{inx} (c)

図 4.5 単純カウンタの分解

$i \in A_{\text{sync}}$ は同期イベントである．図 4.5 (b) は図 4.5 (a) の L_{in} の $l = 1$ における L_{inx} を示している．$A_1 = \{a, i\}$，$A_2 = \{b, i\}$ とし，$\widetilde{\Sigma}_{\text{x}} = (A_1, A_2)$，$Eq = (\sim_{A_1}, \sim_{A_2})$ とする．このとき, 組 $(L_{\text{inx}}, \widetilde{\Sigma}_{\text{x}}, Eq)$ は定義 14 の条件 (p1) から (p4) を満たす，そしてこの 3 字組から $L_1 = CNT_l(a, i)$ と $L_2 = CNT_{m-l}(i, b)$ が得られる．

カウンタ分解によって得られた各出力 LTS は現在のカウンタ値（カウントアップ, ダウンイベントの発生回数の差）を単独では判断できない．例えば図 4.5 においてイベント a が 2 回続けて起こった後，L_{in} は状態 2 にあり，カウンタ値が 2 であるとこはすぐにわかる．出力 LTS L_1 では，a が 2 回続けて起こった後の状態が，状態 0（$\langle a, i, a, i\rangle$ の順にイベントを実行した後の状態），状態 1（$\langle a, i, a\rangle$ の順にイベントを実行した後の状態）のどちらの場合もあるので，単独ではカウンタ値が今，何であるか判断できない．しかし，直積 LTS$(L_1 \times L_2)$ の状態としては $(1, 1)$ または，$(0, 2)$ であり，この直積 LTS の状態からカウンタ値は 2 であると判断できる．

4.3.2 一般化カウンタ

定義 16 (一般化カウンタ) $a, b \in Evt - \{\tau\}$ を任意のイベント，m を負でない整数とする．もし $cont(L, \{a, b\})$ が $CNT_m(a, b)$ と同形なら LTS L は，カウントアップイベント a, カウントダウンイベント b, 容量 m の**一般化カウンタ** である．カウントアップおよびダウンイベントが a, b かつ容量が m である一般化カウンタのクラスを $GCNT_m(a, b)$ と書く． □

定義 17 (一般化擬似カウンタ) $A, B \subseteq Evt - \{\tau\}$ を $A \cap B = \emptyset$ であるイベントの集合，m を負でない整数とする．$a, b \notin A \cup B \cup A[L] \cup \{\tau\}$ を，L や A, B に現れない新しいイベントとする．以下を満たす LTS L は，A をカウントアップイベントの集合, B をカウント

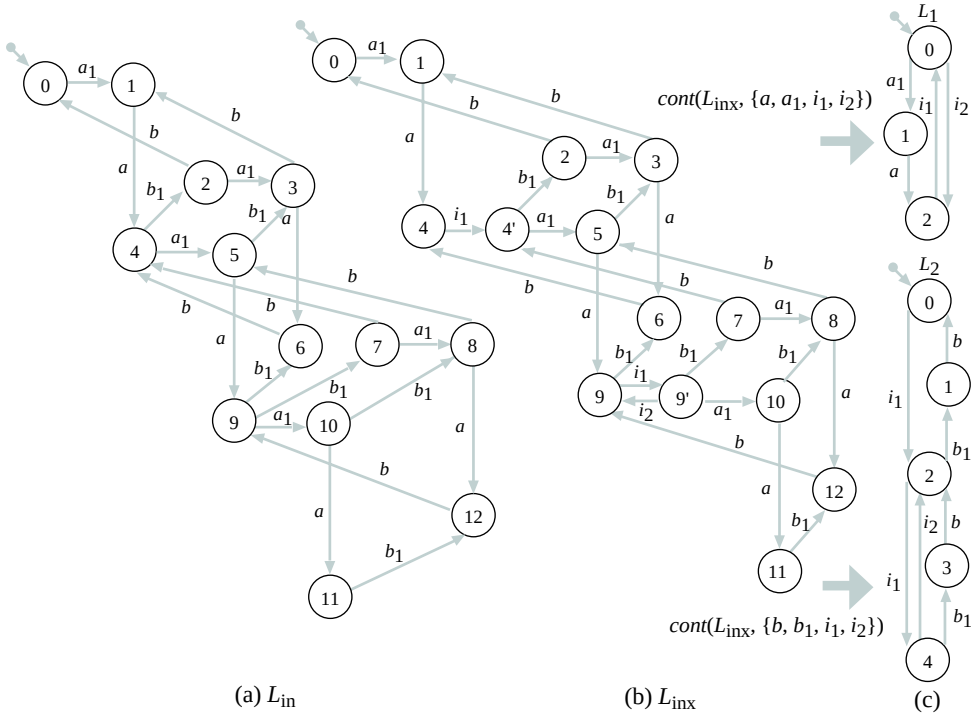


図 4.6 一般化カウンタの分解

ダウンイベントの集合, m を容量とする**一般化擬似カウンタ** である．L において，A 中のイベントを a に，B 中のイベントを b に置換することによって $GCNT_m(a,b)$ が得られるとき，カウントアップイベントおよびダウンイベントの集合が A，B，容量が m である一般化擬似カウンタのクラスを $GQCNT_m(A,B)$ と書く． □

例えば, 図 4.6 (a) はカウントアップイベント a, カウントダウンイベント b，容量 3 の一般化カウンタである．

次に，一般化カウンタ $L_{\text{in}} \in GCNT_m(a,b)$ と $\widetilde{\Sigma} = (B_1, B_2)$(ただし $\widetilde{\Sigma}$ に対して $a \| b$) に関する分解問題について考える. 一般性を失うことなく $a \in B_1$ と仮定する. 以下に，このような L_{in} と $\widetilde{\Sigma}$ のための，定義 14 Step 1 のアルゴリズムの概略を示す．入力 $(L_{\text{in}}, \widetilde{\Sigma})$ に対し，アルゴリズムは正しい結果 $(L_{\text{inx}}, \widetilde{\Sigma}_{\text{x}}, Eq)$ を出力するか，または分解に失敗したと出力する．

分解が成功したとき, 定義 14 の Step 2 により 2 つの LTS $L_1 \in GQCNT_1(\{a, i_2\}, \{i_1\})$, $L_2 \in GQCNT_{m-1}(\{i_1\}, \{b, i_2\})$ が得られる．例えば, 図 4.6 (a) の L_{in} と $\widetilde{\Sigma} = (\{a, a_1\}, \{b, b_1\})$ が与えられたとき，アルゴリズムは図 4.6 (b) の L_{inx}，および $\widetilde{\Sigma}_{\text{x}} = (A_1, A_2)$，$Eq = (\sim_{A_1}, \sim_{A_2})$ を出力する．ここで $A_1 = \{a, a_1, i_1, i_2\}$，$A_2 = \{b, b_1, i_1, i_2\}$ である. この組 $(L_{\text{inx}}, \widetilde{\Sigma}_{\text{x}}, Eq)$ から, 図 4.6 (c) の LTS の L_1 と L_2 が導出される．

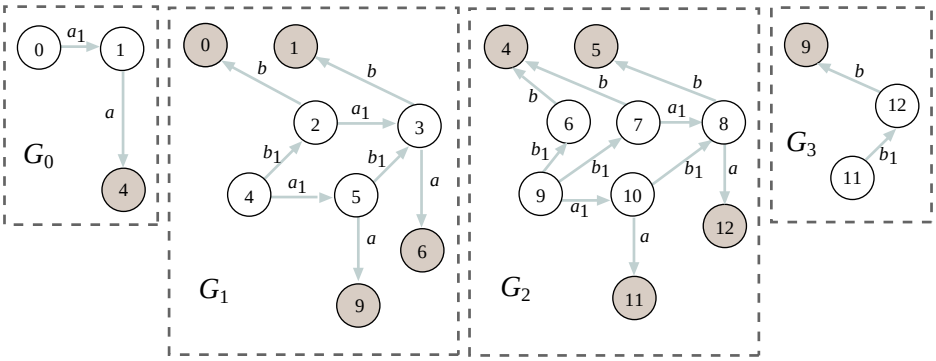


図 4.7 図 4.6 の L_{in} に対する G_i

一般化カウンタのための分解アルゴリズムの概略

(1) i_1 と i_2 を A_{sync} 内の相異なるイベントとする. $A_1 = B_1 \cup \{i_1, i_2\}$, $A_2 = B_2 \cup \{i_1, i_2\}$ とする.

(2) L_{in} をラベル付き有向グラフ $(S[L_{\text{in}}], \rightarrow_{L_{\text{in}}})$ とみなし, 各 $i \in [0,m]$ について L_{in} のサブグラフ $G_i = (S[G_i], \rightarrow_{G_i})$ を，以下の条件をみたす最小のグラフと定義する：$V_i = \{s \in S[L_{\text{in}}] \mid s\}$ は $CNT_m(a,b)$ の状態 i に対応するとすると，$V_i \subseteq S[G_i]$. もし $s \in V_i$ かつ $s \xrightarrow{\alpha}_{L_{\text{in}}} s'$ なら，$s' \in S[G_i]$ かつ $s \xrightarrow{\alpha}_{G_i} s'$. 例えば, 図 4.7 のグラフは図 4.6 (a) の L_{in} に対する G_i である．ここで，灰色に塗られた頂点は，V_i の要素でない頂点である.

(3) $G_{0,0} = G_0$ ， $G_{m,m-1} = G_m$ とする. 各 $i \in [1, m-1]$ について, 以下で説明するように G_i を $G_{i,i-1}$ と $G_{i,i}$ に分解する. そして, 図 4.8 のように L_{inx} の中に配置する．この配置は単純カウンタの分解からの類推である．G_i のこの分解は以下の考えに基づいている：アルファベットが B_1 であるグラフ $G_0^{B_1}, G_1^{B_1}$ と B_2 であるグラフ $G_0^{B_2}, G_1^{B_2}, \ldots, G_{m-1}^{B_2}$ が存在し，各 $i \in [0, m-1]$，$j \in [0,1]$ について，$G_{i+j,i}$ が $G_j^{B_1} \times G_i^{B_2}$ と (図 4.8 参照) 同形であるなら，L_{in} は小さい LTS L_1 と L_2 に分解できる.

このとき，L_1 は $G_0^{B_1}$ と $G_1^{B_1}$ で構成される容量 1 の一般化擬似カウンタであり，L_2 は $G_0^{B_2}, G_1^{B_2}, \ldots, G_{m-1}^{B_2}$ で構成されている容量 $m-1$ の一般化擬似カウンタである. この考えに基づき，各 $i = m-1, m-2, \ldots, 1$ について，G_i を以下のように分割する：$G_0^{B_1} = cont(G_{0,0}, B_1)$，$G_i^{B_2} = cont(G_{i+1,i}, B_2)$ とする. 次に G_i を, $G_0^{B_1} \times G_i^{B_2}$ と同形であるサブグラフが $G_{i,i}$ とその残りのサブグラフ $G_{i,i-1}$ に分割する (図 4.6 (b) 参照).

もし，$G_0^{B_1} \times G_i^{B_2}$ と同形であるサブグラフが G_i にないなら , アルゴリズムは分解に失敗したと出力して停止する.

(4) 各 $i \in [1, m-1]$, $s_1 \in S[G_{i,i-1}]$，$s_2 \in S[G_{i,i}]$ について, もし s_1 と s_2 が 対応する同

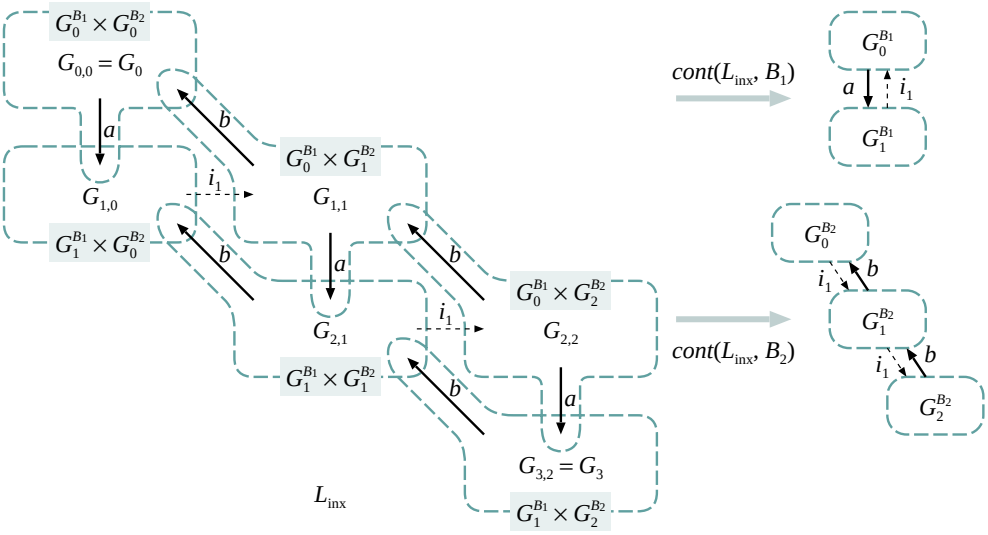


図 4.8 L_{inx} 上の $G_{i,j}$ の配置

じノードが G_i に存在するなら, 辺 $s_1 \xrightarrow{i_1}_{L_{\text{inx}}} s_2$ を追加する. また, もし s_1' について $s_1 \xrightarrow{\alpha}_{G_{i,i-1}} s_1'$ が存在し，$G_{i,i}$ に s_1' に対応するノードがなければ, $s_2 \xrightarrow{i_2}_{L_{\text{inx}}} s_1$ を追加する (図 4.6 (b) のノード 4 と 9 を参照).

(5) $\tilde{\Sigma} = (A_1, A_2)$，$Eq = (\sim_{A_1}, \sim_{A_2})$ とし，組 $(L_{\text{inx}}, \tilde{\Sigma}, Eq)$ が定義 14 の条件 (p4) を満たすかどうかチェックする．もし，満たしていなければアルゴリズムは分解に失敗したと出力して停止する． □

4.3.3 アルゴリズムの評価

ここでは，提案したアルゴリズムによる分解と従来法による分解について出力 LTS のサイズを比較する．表 1 は図 4.6(a) の L_{in} を従来法と本提案法で分解した場合の出力 LTS のサイズの比較である．この表より，本提案法は分解後の各 LTS の状態数，環境との通信イベント ($A[Lin]$ の要素) による遷移数および同期イベント (A_{sync} の要素) による遷移数とも従来法と比べて小さいことがわかる．

4.4 考察

4.4.1 カウンタプロセス分解の応用

4.3 節ではカウンタプロセスの分解法について説明した．ここでは，カウンタプロセスの分解法を利用して，図 4.9(c) に示すようなパイプライン状に接続されたプロセス群を出力する方法を示す．入力として，図 4.9(a) に示すカウンタプロセス $L_{\text{in}} =$

表 4.1 従来法と本提案法との出力サイズの比較

	従来法		提案法	
	L_1	L_2	L_1	L_2
状態数	30	32	3	5
$A[L_{\text{in}}]$ 中のイベントによる遷移数	10	12	2	3
同期イベント (A_{sync} の要素) による遷移数	37	37	4	3
A_{sync} の要素数	3		2	

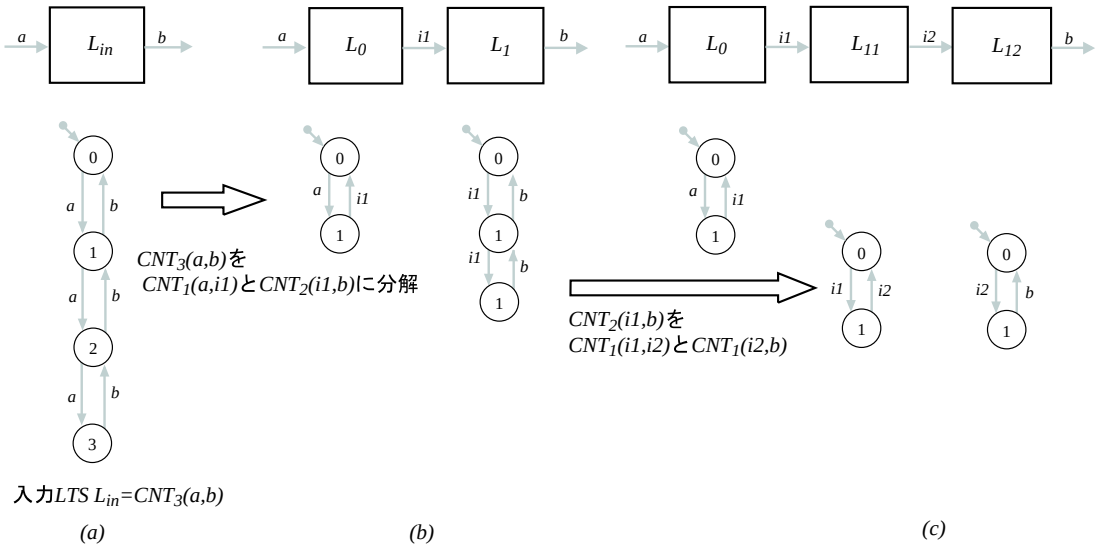


図 4.9 カウンタプロセス分解の応用

$CNT_3(a,b)$ と $\widetilde{\Sigma} = (\{a\},\{b\})$ を考える．このプロセスを単純化カウンタの分解法を用いて $CNT_1(a,i1)$ と $CNT_2(i1,b)$ に分解する（図 4.9(b)）．結果として $L_0 = CNT_1(a,i1)$ と $L_1 = CNT_2(i1,b)$ を得ることができる．さらに出力 LTS のうち，$L_1 = CNT_2(i1,b)$ を入力として $\widetilde{\Sigma} = (\{i1\},\{b\})$ で 2 つのプロセスに分解する．結果の LTS として $L_{11} = CNT_1(i1,i2)$ と $L_{12} = CNT_1(i2,b)$ が得られる．このように 4.3 を利用することによって，多段にパイプライン状に接続されたプロセスに分解できる．

4.4.2 複合カウンタプロセスの分解に対する考察

次により広い LTS のサブクラスの分解例として複数のカウンタプロセスが合成されたプロセスの分解について考察する．入力 LTS L_{in} として，図 4.10 のようなプロセスを考える．このプロセスを $\widetilde{\Sigma} = (\{a\},\{x\},\{c\})$ となる 3 つのプロセスに分解する．この L_{in} をイベント a,c について縮約したプロセス $cont(L_{\text{in}},\{a,c\})$ と x,c について縮約

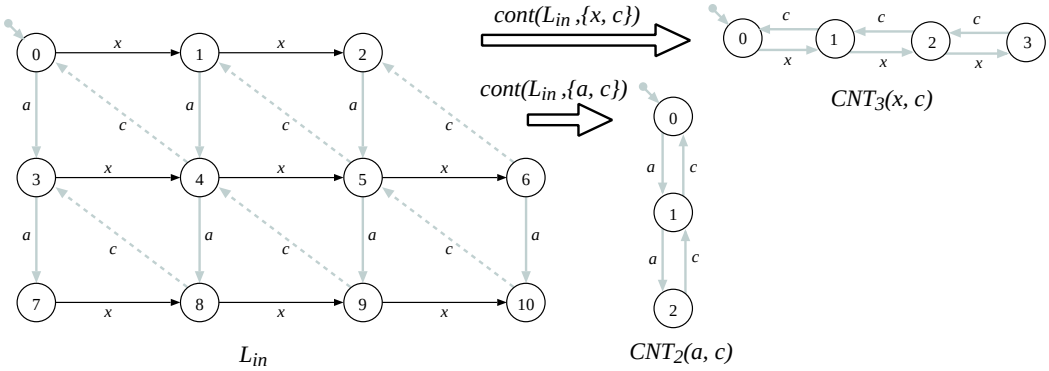


図 4.10 複合カウンタプロセスの分解 (入力 L_{in} とその縮約)

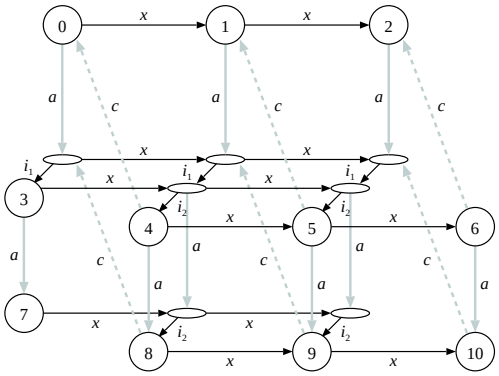


図 4.11 複合カウンタプロセスの分解 (L_{inx} の例)

したプロセス $cont(L_{\text{in}}, \{x,c\})$ は図 4.10 のように，カウンタプロセス $CNT_2(a,c)$ とカウンタプロセス $CNT_3(x,c)$ と等価な LTS となる．つまりこの入力 LTS L_{in} はこの 2 つのカウンタプロセスの合成であると推測することができる．この L_{in} に対して図 4.11 に示すような L_{inx} を得ることができれば，図 4.12 のようなサイズの小さい 3 つのカウンタプロセスを出力として得ることができる．

4.5 第 4 章のまとめ

この章では，第 3 章と同様，同期イベントを追加することを許す，弱双模倣等価性の下でのプロセスの分解について議論した．そして第 3 章の分解法を拡張し，分解後の LTS のサイズが従来法よりしばしば小さくなるような分解法を提案した．また，提案手法に基づいて, 単純カウンタと一般化カウンタという 2 つの LTS のサブクラスについて，分解アルゴリズムおよびその応用を示した．出力 *LTS* のサイズに関する提案

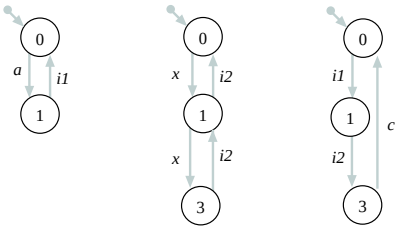


図 4.12 複合カウンタプロセスの分解 (出力例)

アルゴリズムのより詳細な評価が今後の課題である．さらに，考察で示したような，LTS のより広いサブクラスに対応できるようにアリゴリズムを拡張することも興味深い課題である．

第5章　あとがき

この論文では，同期イベントの追加を許す，弱双模倣等価性の下でのプロセス分解問題について考察し，分解法を提案した．

まず，環境とシステムとの通信とシステム内の通信とを区別できるようなシステムに対して，連続イベント制約を用いて分解後の LTS のサイズを小さくする手法を示した．また, 導出した LTS と入力 LTS 間の弱双模倣等価性を証明した. そして，インタリーブ領域を簡単化して総状態数を小さくする方法や, システム内部で用いられる同期イベントを共通化して同期イベントの種類を減らす方法を示した．

さらにイベント間の発生の関連性について着目し，分解後の LTS のサイズが従来法よりしばしば小さくなるような分解法を提案した．また，この手法に基づいて, 単純カウンタと一般化カウンタという 2 つの LTS の部分クラスについて，分解アルゴリズムを示した．

今後の課題として，出力 LTS のサイズに関するより詳細な評価や，第 4 章のアルゴリズムをより広い LTS のサブクラスに適用できるように拡張することなどが挙げられる．

謝 辞

この５年間，筆者の直感に頼りがちな研究をこの論文をまとめあげるまでに, 根気強くご指導くださり，また，仕事でなかなか時間の取れない筆者のために休日まで打ち合わせの時間を取って頂いた主指導教官である本学情報基礎学講座の関 浩之教授に深く感謝いたします．この５年間のご指導で，論証の大切さ，飾らず黙々と１つの研究をすすめる研究に対する姿勢など，多くの事を学ばせていただきました．重ねて深くお礼申し上げます.

また, ご多忙の中，副指導教官を引き受けていただき, 論文の細部にまで目を通していただき, 並列度に関する本質的見解，自動分解法の実用性および計算量に対する評価などについて，多くの心暖かなご助言, ご指導をくださった渡邉 勝正教授ならびに伊藤 実教授に深く感謝の意を表したいと思います.

修士論文の副指導教官を引き受けていただき，研究室内のディスカッションなどでも研究の方向性，アルゴリズムの評価の必要性などについて，多くの貴重なご意見をくださった椙 勇一 助教授に感謝いたします.

本研究の全般にわたって指導に多大な時間を費やしていただいた高田 喜朗助手に深く感謝いたします. この５年間のご指導で，論証における緻密さの重要性，および論文を執筆する上での配慮などコンピュータ科学者として必要な多くのスキルについて学ぶことができました．深くお礼申し上げます.

加えて，あまり研究室に来られない筆者をいつも暖かく迎えてくれた情報基礎学講座の皆様，この５年間, 筆者の仕事をフォローをしていただいた職場の仲間にもこの場を借りて深くお礼申し上げます.

最後に筆者にコンピュータ科学の意義を伝授していただいた帝塚山学院大学 人間文化学部の山本 正樹教授はじめ, 今まで筆者をご指導下さった先生方, 研究者の方々に深く感謝いたします.

参考文献

[1] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall (1985).

[2] Mizuno, T., Shiratori, N., Higashino, T. and Togashi, A.(eds.): *Formal Description Techniqures and Protocol Specification*, Chapman & Hall (1997).

[3] Wing, J. M., Woodcock, J. and Davies, J.(eds.): *FM'99 Formal Methods, Lecture Notes in Computer Science 1709*, Springer-Verlag (1999).

[4] Noguchi, S. and Ota, M.(eds.): *Correct Models of Parallel Computing*, IOS Press (1997).

[5] 高橋薫, 神長裕明: 仕様記述言語 LOTOS, カットシステム, pp. 147–154 (1995).

[6] Morin, R.: Decompositions of Asynchronous Systems, *CONCUR '98, LNCS 1466* (Sangiorigi, D. and de Simone, R.(eds.)), Springer, pp. 549–564 (1998).

[7] Ehrenfeucht, A. and Rozenberg, G.: Partial (Set) 2-structures; Part II: State Spaces of Concurrent Systems, *Acta Informatica*, Vol. 27, pp. 343–368 (1990).

[8] Mukund, M.: From Global Specifications to Distributed Implementations, *Synthesis and Control of Discrete Event Systems* (Caillaud, B. et al.(eds.)), Kluwer Academic Publishers, pp. 19–35 (2002).

[9] Saleh, K.: A Synthesis of Communications Protocols: An Annotated Bibliography, *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 5, pp. 40–59 (1996).

[10] Langerak, R.: Decomposition of Functionality: A Correctness Preserving LOTOS Transformation, *Protocol Specification, Testing and Verification (PSTV) X*, pp. 229–242 (1990).

[11] 馬渕博之, 高橋薫, 白鳥則郎: LOTOS に基づいたプロトコル仕様の導出, 電子情報通信学会技術研究報告, Vol. IN91-110 (1991).

[12] Go, K. and Shiratori, N.: A Decomposition of a Formal Specification: An Improved Constraint-Oriented Method, *IEEE Trans. Software Eng.*, Vol. 25, No. 2, pp. 258–273 (1999).

[13] Higashino, T., Okano, K., Imajo, H. and Taniguchi, K.: Deriving Protocol Specifications from Service Specifications in Extended FSM Models, *The 13th IEEE ICDCS*, pp. 141–148 (1993).

[14] Khoumsi, A., von Bochmann, G. and Dssouli, R.: On Specifying Services and Synthesizing Protocols for Real-time Applications, *Protocol Specification, Testing and Verification (PSTV XIV)*, pp. 185–200 (1994).

[15] Tokheim, R. L., 村崎憲雄, 藤村宏一, 青木正喜 (共訳): ディジタル回路, オーム社, pp. 98–100 (1995).

研究業績

1. 学術論文誌

1-1 喜家村奨, 高田喜朗, 関浩之: ラベル付き遷移システムで記述された要求仕様の並列プロセス群への一分解法, 情報処理学会論文誌, Vol.42, No.12, pp.2992–3003, Dec. 2002.

1-2 高田喜朗, 喜家村奨, 関浩之: 同期イベントを用いたプロセス分解法とカウンタプロセス分解への応用, 電子情報通信学会論文誌 D-I, 投稿中.

2. 国際会議論文集

2-1 Susumu Kiyamura, Yoshiaki Takata and Hiroyuki Seki : Process Decomposition via Synchronization Events and Its Application to Counter-Process Decomposition, *Fifth International Conference on Parallel Processing and Applied Mathematics (PPAM2003), Czestochowa, Poland, Sept. 2003*, Lecture Notes in Computer Science, To appear.

3. 研究会およびシンポジウム発表

3-1 喜家村奨, 高田喜朗, 関浩之: 動作定義と外部イベント集合からの並列プロセス自動生成法, 電子情報通信学会技術研究報告, SS2000-39, Jan. 2001.

3-2 喜家村奨, 高田喜朗, 関浩之: 同期イベントを用いたプロセス分解法とカウンタプロセス分解への応用, 電子情報通信学会 技術研究報告, SS2003-4, May 2003.

4. その他

4-1 Susumu Kiyamura, A.W. Roscoe : A case study of the formal specification of a parallel system using CSP, *Correct Models of Parallel Computing: The 7th Transputer/occam International Conference, Tokyo, Japan, 1997*, IOS Press, pp.68-86, 1997.

付 録

A　連続イベント制約を生成しない場合の定理 1 の証明

3.2, 3.3 節の補題・定理のほとんどは，連続イベント制約の生成を行わなくても影響ない．連続イベント制約を生成しないことが影響するのは，定理 1 の証明の (b)(2) の前半（$(t_1,\ldots,t_n) \xrightarrow{a}_3 \cdot$ ならば $a = \tau(i^b_{s'})$ であることの証明）のみである．従って，連続イベント制約を使わずにこのことを示せば十分である．このことを背理法で示すため，$(t_1,\ldots,t_n) \xrightarrow{a}_3 \cdot$ かつ $a \neq \tau(i^b_{s'})$ と仮定する．

(1) $a \in A - \{\tau\}$ と仮定する．このとき $a \in B^j$ である j が少なくとも一つ存在する．この j について，

  (i) $b \in B^j$ のとき，
  (ii) $b \notin B^j$ かつ $i^b_{s'} \in A_2^j$ のとき，
  (iii) $b \notin B^j$ かつ $i^b_{s'} \notin A_2^j$ のとき，

  いずれの場合も矛盾が示せる．

(2) ある t' について $a = \tau(i^\tau_{t'})$ であると仮定する．このとき $i^\tau_{t'} \in A_2^j$ である j が少なくとも一つ存在する．(1) と同様に，この j について，

  (i) $b \in B^j$ のとき，
  (ii) $b \notin B^j$ かつ $i^b_{s'} \in A_2^j$ のとき，
  (iii) $b \notin B^j$ かつ $i^b_{s'} \notin A_2^j$ のとき，

  いずれの場合も矛盾が示せる．

(3) ある t' および $c \neq \tau$ について $a = \tau(i^c_{t'})$ かつ $i^c_{t'} \neq i^b_{s'}$ と仮定する．このとき $c \in B^j$ である j が少なくとも一つ存在する．この j について，

  (i) $i^b_{s'} \in A_2^j$ のとき，
  (ii) $i^b_{s'} \notin A_2^j$ のとき，

いずれも R_2 の定義に矛盾する．以上より，$(t_1,\ldots,t_n) \xrightarrow{a}_3 \cdot$ ならば $a = \tau(i^b_{s'})$ であることが言える．　□