振舞い近似手法を用いたステートチャートに対する不変性の検証

立石孝彰[†] 青木利晃[††,†††] 片山卓也[††]

本稿では，互いに依存する複数のステートチャートを合成することなく不変性の検証を行う手法を示す．単一のステートチャートの検証では，まず最初に与えられたステートチャートをステートチャートに対する正規表現に変換する．そして，本稿で提案する形式的体系を用いて，Hoare のプログラム検証手法と同じ方法を用いて不変性に関する検証を行う．振舞いが互いに依存する複数のステートチャートの検証においては，通常は合成と呼ばれる手法が用いられる．しかし，状態爆発などの問題を起こすため，本稿では合成後に得られるステートチャートの振舞いに徐々に近づける手法を提案する．この方法は，目的とする検証が行えるようになるまで何度でも適用できる．また，ステートチャートどうしの通信方法には様々ある．ステートチャートがどのように協調動作するのかに応じて，ステートチャート間の通信方法を決める必要がある．そこで，提案する手法では，通信方法に応じて柔軟に対応できるようにしている．

Behavior Approximation Method for Verifying Invariant on Statecharts

Takaaki Tateishi,[†] Toshiaki Aoki[††,†††] and Takuya Katayama[††]

In this paper, we propose a method for verifying invariant properties of statecharts without making the composition. We first convert a statechart to a regular expression to verify the given single statechart. We then verify it with the similar method to Hoare's program verification method. When verifying multiple statecharts which depend on the other, we make them to approximate composite behavior by repeating to apply a method we describe in this paper. The method can repeat to be applied until a target verification is completed. On the other hand, there are various communication mechanism between statecharts and we apply one of them considering how statecharts communicate with each other. Our method can be flexibly adapted to an applied communication mechanism.

1. はじめに

オブジェクト指向方法論において，オブジェクトの動作を状態遷移図を用いて記述することが広く行われている．1 つのシステムには複数のオブジェクトが存在し，互いに協調して動作する．そのため，あるオブジェクトの動作は，他のオブジェクトの動作に依存することがある．そのようなオブジェクトの動作を検証するために，複数のオブジェクトの動作を合成するのが普通である．しかし，状態遷移図を合成すると状態数が爆発的に増加し，合成した状態遷移図が複雑になる．また，複雑な状態遷移図に対して，オブジェクトの持つ性質を検証することは困難である．

一方，オブジェクト指向方法論の分析・設計モデルにおいて，UML[8] と呼ばれるモデリング言語が広く使われている．UML では，オブジェクトなどの振舞いを記述するために用いられるステートチャートが定義されている．UML で定義されているステートチャートは Harel のステートチャート[4] の変形であり，属性評価をともなう状態遷移図である．

そこで本稿では，互いに振舞いが依存する複数のステートチャートにおいて，合成を用いずに**近似**と呼ぶ手法を用いて属性の性質に関する検証を行う．この手法では，それぞれのステートチャートの振舞いを徐々に合成後の振舞いに近似し，近似後の単独のステートチャートに対して検証を行う．そのため，合成ほど詳細なステートチャートを用いることのない検証では，必要なだけ近似を適用したステートチャートを用いて検証を行うことができる．近似したステートチャートの検証には，我々が提案する形式的体系を用いる．この形式的体系は，プログラム検証で広く用いられる Hoare の形式的体系を参考にしたものである．

† 日本 IBM 東京基礎研究所
IBM Tokyo Research Laboratory
†† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology
††† 科学技術振興事業団さきがけ研究 21
PRESTO, Japan Science and Technology Corporation

また，近似方法が持つべき性質のみを与えることによって，設計・分析の方針に応じて，近似方法を柔軟に定義できるようにする．このため，検証者に都合がよいように近似方法を定義でき，合成を行う場合よりも簡単に検証を行うことが期待できる．近似方法は，ステートチャート間の協調動作の方法を決定する合成方法に依存するので，近似方法と合成方法の関係を明らかにする．

本稿の構成は次のとおりである．はじめにステートチャートを紹介する．そして，3 章において単一ステートチャートを検証する手法を示す．4 章では，複数の依存し合うステートチャートを検証する手法を示す．そして，5 章では，我々が提案する検証手法を適用できる範囲について議論する．6 章では，関連する研究を紹介し，本稿で提案する手法と比較する．最後に 7 章において，本稿のまとめを行う．

2. ステートチャート

ステートチャートは状態と遷移から構成され，初期状態と呼ばれる状態から動作が始まり，終了状態と呼ばれる状態で動作が終了する．初期状態は必ず 1 つだけ存在しなければならず，また，終了状態は複数存在しても存在しなくてもよい．**図 1** は終了状態が存在しない単純な 2 状態のステートチャートであり，黒丸は初期状態を表す．各遷移には以下の形式を用いて遷移ラベルを記述する．

(入力イベント) [(ガード条件)] / (アクション式)

それぞれの遷移は，入力イベントで示されたイベントをステートチャートが受け取り，ガード条件が成り立つときに発火する．そして，アクション式を評価し，イベントを出力する．アクション式には属性への代入と，出力イベントを記述する．代入は ':=' を用いて記述し，出力イベントは 'send' を用いて記述する．たとえば，図 1 では，$inc/n := n+1, \mathrm{send}(a)$ は遷移ラベルである．この遷移ラベルは，ステートチャートがイベント inc を受け取ると発火し，$n := n+1, \mathrm{send}(a)$ というアクション式を評価する．このアクション式は，$n := n+1$ によって属性 n の値を 1 増加し，$\mathrm{send}(a)$ によってイベント a を出力することを表している．また，出力されたイベントがどのように他のステートチャートに受け取られるのかという通信方法は，分析・設計方針により異なる．

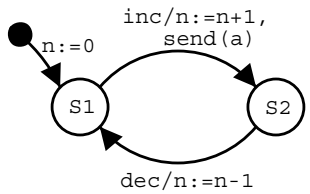

図 1 ステートチャート
Fig. 1 Statechart.

3. 単独のステートチャートの検証

本章では，前述したステートチャートのすべての状態において，属性に与えられた性質が成り立つことを検証するための手法を示す．たとえば，図 1 において，属性 n の値がつねに非負であるという性質の検証などである．このような検証を行うために，まず最初にステートチャートを**遷移系列式 *TSE***（*Transition Sequence Expression*）と呼ぶ正規表現に変換する．そして，Hoare のプログラム検証と同様に，TSE の前後に事前表明と事後表明を与えて，本稿で提案する形式的体系を用いて証明を行う．

3.1 TSE

TSE は，イベント入力，イベント出力，アクション，ガード条件，空列，エラーを表す 6 つのプリミティブから，以下の文法に従って構成する．

TSE	=	TSE ; TSE
	|	TSE + TSE
	|	TSE *
	|	primitive
primitive	=	? event-name
	|	! event-name
	|	[[action]]
	|	[condition]
	|	ε
	|	$\perp$
	|	(TSE)

';' は連接を表し，遷移 T_1 の後に遷移 T_2 が起こることを $T_1; T_2$ と表す．'+' は選択を表し，遷移 T_1 と T_2 のどちらかが起こることを $T_1 + T_2$ と表す．'*' は遷移の繰返しを表す．イベントの入出力は，'?' や '!' をイベントに前置することでそれぞれ表す．また，ガード条件 condition は '[' と ']' によって囲み，アクション action は '[[' と ']]' で囲むことによって表す．簡単のため，本稿ではアクションには属性への値の代入のみを扱い，属性値は整数値と真偽値のみを扱うことにする．ε と $\perp$ は，それぞれ空列とエラーを表す特殊なプリミティブである．ここで，エラーとは，絶対に起こりえない遷移を表すものである．たとえば，$[1 > 2]; !a; \varepsilon$ という遷移の列は決して起こらないため，本稿で提案する形式的体系において，$\perp$ と同じ意味を持つことに

なる．ここで提案した TSE は，交換則や分配則などの性質を満たすものとする．

$$T;\varepsilon = \varepsilon; T = T$$
$$T;\bot = \bot; T = \bot$$
$$T + \bot = T$$
$$T_1 + T_2 = T_2 + T_1$$
$$(T_1 + T_2) + T_3 = T_1 + (T_2 + T_3)$$
$$T_1;(T_2;T_3) = (T_1;T_2);T_3$$
$$T;(T_1 + T_2) = T;T_1 + T;T_2$$
$$(T_1 + T_2);T = T_1;T + T_2;T$$
$$T* = T;T* + \varepsilon$$

ここで，T，T_1，T_2，T_3 は TSE である．

3.2 TSE への変換

ステートチャートを TSE に変換する方法は，文献 6), 7) などで述べられている有限オートマトンを正規表現に変換する方法とほぼ同じである．異なる点は，すべての状態を終了状態と見なしてステートチャートを TSE に変換する点である．これは，TSE によって，各々の状態で動作が終了するまでの遷移列すべてを表せるようにするためである．これにより，Hoare の検証手法と同様の手法を用いて，属性の性質がどのような状態でも成り立つことを検証できる．たとえば，図 1 のステートチャートの S_1，S_2 のどちらの状態においても，ある性質が成り立つことを調べる場合を考える．このとき，S_1 で動作が終了したときに与えた性質が成り立つことと，S_2 で動作が終了したときに与えた性質が成り立つことを調べれば十分である．

まずステートチャートの各々の状態に対して変数を割り当てる．それぞれの変数は，その状態を初期状態としたときに可能な遷移の列を TSE を用いて表したものである．このような変数を**状態変数**と呼ぶ．次に，それぞれの状態間の関係を表す方程式を作り，その方程式を解くことによって初期状態から遷移可能な遷移ラベルの列を求めることができる．一般的に，初期状態 S_0 と他 n 状態のステートチャートでは，状態 $S_0, S_1, \cdots, S_n$ に対する状態変数 $T_0, T_1, \cdots, T_n$ を用いて次の連立方程式を作ることができる．

$$\begin{aligned}
T_0 &= A_{00};T_0 + A_{01};T_1 + \cdots + A_{0n};T_n \\
T_1 &= A_{10};T_0 + A_{11};T_1 + \cdots + A_{1n};T_n + \varepsilon \\
T_2 &= A_{20};T_0 + A_{21};T_1 + \cdots + A_{2n};T_n + \varepsilon \\
&\vdots \\
T_n &= A_{n0};T_0 + A_{n1};T_1 + \cdots + A_{nn};T_n + \varepsilon
\end{aligned}$$

ここで，A_{ij} は，状態 S_i から状態 S_j への遷移の遷移ラベルを TSE を用いて表現したものである．S_i から S_j への遷移が存在しない場合には，$\bot$ を用いる．初期状態を除くすべての状態を終了状態とするため，T_1 から T_n に対して最後に ε を用いている．このように構成された方程式は 1 つずつ変数を消去するか，$T = A;T + B$ の解が $T = A*;B$ という事実を用いることによって必ず解くことができる．

図 1 の場合，それぞれの状態に対して T_0，T_1，T_2 という状態変数を割り当てる．ただし，T_0 は初期状態に対して割り当てた状態変数である．次に，それぞれの状態変数の関係を用いて連立方程式を作ると，T_0 の解は，初期状態から遷移可能な遷移ラベルの列の集まりを表す．初期状態と状態 S_1，S_2 に対して状態変数 T_0，T_1，T_2 を割り当てると次の連立方程式を得ることができる．

$$\begin{aligned}
T_0 &= [[n := 0]];T_1 \\
T_1 &= ?inc;[[n := n + 1]];!a;T_2 + \varepsilon \\
T_2 &= ?dec;[[n := n - 1]];T_1 + \varepsilon
\end{aligned}$$

この方程式を T_0 に対して解くと，以下の目的の TSE を得ることができる．

$$[[n := 0]];((?inc;[[n := n + 1]];!a;?dec;$$
$$[[n := n - 1]])*;(?inc;[[n := n + 1]];!a + \varepsilon))$$

そして，この得られた TSE を用いて検証を行う．上記の TSE が表すことを明確にするために，式を展開すると以下の TSE を得ることができる．

$$[[n := 0]];(?inc;[[n := n + 1]];!a;?dec;$$
$$[[n := n - 1]])* \quad (1)$$
$$+[[n := 0]];(?inc;[[n := n + 1]];!a;?dec;$$
$$[[n := n - 1]])*;?inc;[[n := n + 1]];!a \quad (2)$$

式 (1) は S_1 で終了する遷移を表し，式 (2) は S_2 で終了する遷移を表していることが分かる．

3.3 検証のための形式的体系

与えられたステートチャートから検証で用いる TSE を得た後，仮定と検証したい性質を TSE の前後に表明としてそれぞれ記述する．このように，TSE の前後に表明を付けた TSE を *A-TSE*（*Asserted TSE*）と呼び，$\{P\}T\{Q\}$ という形式を用いて記述する．この式は，TSE T の前後で P, Q がそれぞれ成り立つことを表している．特に，TSE の前に付けた表明を**事前表明**，後ろに付けた表明を**事後表明**と呼ぶ．我々の手法では，得られた A-TSE を証明することによって，不変性の検証を行う．証明には我々が提案する**論理** *ATL* を用いる．本稿では，アクションやガード条件に記述できる式として，整数値と真偽値を扱う式としたので，事前表明と事後表明には，整数の比較などの真偽値を返す論理式を記述できるものとする．

論理 ATL の公理は TSE のプリミティブに対してそれぞれ導入する．

公理 1

$\{P\}\varepsilon\{P\}$
$\{P\}\bot\{Q\}$
$\{P[t/v]\}v := t\{P\}$
$\{P\}!e\{P\}$
$\{P\}?e\{P\}$
$\{P\}[C]\{P \wedge C\}$

ε とイベント入力・出力の前後では，属性値は変化しないため，事前表明と事後表明は同じとなる．$\bot$ はエラーを表し，エラーの場合にはどのような事後表明に対しても A-TSE が成り立つものとする．代入の公理において，$P[t/v]$ は事後表明 P に現れる属性 v を，代入する式 t で置き換えた表明を表している．ガード条件の後では，ガード条件で用いた論理式が成り立っていて，属性は変化しないため，事後表明は事前表明とガード条件の論理積となる．

推論規則には，まず事前表明を弱め，事後表明を強めるための帰結規則を導入する．

推論規則 1

$$\frac{P' \Rightarrow P \quad \{P\}T\{Q\} \quad Q \Rightarrow Q'}{\{P'\}T\{Q'\}}\ conseq$$

また，それぞれの演算子に対しても推論規則を導入する．2 つの TSE T_1 と T_2 において，T_1 の前後でそれぞれ P，Q が成り立ち，T_2 の前後で Q，R がそれぞれ成り立つとき，2 つを連接した $T_1;T_2$ の前後では，それぞれ P，R が成り立つ．このことは以下の推論規則 concat によって表現する．

推論規則 2

$$\frac{\{P\}T_1\{Q\} \quad \{Q\}T_2\{R\}}{\{P\}T_1;T_2\{R\}}\ concat$$

次の推論規則 choice は，2 つの TSE の前後でそれぞれ同じ性質が成り立つときには，選択演算子によって構成された TSE の前後でも同じ性質が保たれることを表している．

推論規則 3

$$\frac{\{P\}T_1\{Q\} \quad \{P\}T_2\{Q\}}{\{P\}T_1 + T_2\{Q\}}\ choice$$

また，推論規則 iteration は，ある TSE の前後で同じ性質が成り立つなら，そのような TSE を何度繰り返しても，性質は保存されることを表している．

推論規則 4

$$\frac{\{P\}T\{P\}}{\{P\}T^*\{P\}}\ iteration$$

我々がここで提案した論理 ATL は健全かつ完全である．このことは，付録 A.1 において述べる．

3.4 例題への適用

図 1 において，$n \geq 0$ という不変性の検証を行うことを考える．T_0 を先に求めた図 1 のステートチャートに対する TSE とする．このとき，T_0 は，個々の状態で終了するような遷移の列すべてを表すものである．このため，T_0 の後で $n \geq 0$ が成り立つことを調べることによって，つねにどの状態においても $n \geq 0$ であることを調べることができる．そのため，$\{true\}T_0\{n \geq 0\}$ という A-TSE を論理 ATL を用いて導出することによって，つねに $n \geq 0$ が成り立つことの検証を行う．事前表明が $true$ であるのは，仮定がないためである．

表 1 に，前提となる A-TSE と使用した推論規則とともに，これらの前提と推論規則から導出できる A-TSE を列挙する．たとえば，1 の $\{n \geq 0\}?inc\{n \geq 0\}$ は，公理より成り立ち，8 の $\{n \geq 0\}?inc;[[n := n+1]];!a\{n \geq 1\}$ は，1，2，3 の A-TSE と推論規則 concat を用いて導出できることを表している．この表から，$\{true\}T_0\{n \geq 0\}$ が導出できることが分かる．このようにして，A-TSE $\{P\}T\{Q\}$ が導出可能であることを $\vdash \{P\}T\{Q\}$ と書く．

4. 依存し合うステートチャートの検証

図 2 の例は，他方のステートチャートから出力されるイベントに依存して振舞いが決定される．このように，検証したいステートチャートの振舞いが，他のステートチャートの振舞いに依存する場合，それら依存するステートチャートを考慮しなければならない．従来の手法では，このようなステートチャートに関する検証では，合成を行うことが一般的に行われてきた．しかし，合成されたステートチャートの状態数は非常に多くなり，検証が困難になる．そこで，本章では，単独ステートチャートの振舞いを，合成後のステートチャートにおける振舞いに近似する手法を用いる．この手法は繰り返し用いることによって，各々のステートチャートの振舞いを徐々に制限するものである．そして，近似されたステートチャートに対して，3 章で示した検証手法を適用する．近似したステートチャートに対して検証を行った結果は，合成したステートチャートに対して検証を行った結果と同じになる．

図 3 は近似手法を図解したものである．SC_1，SC_2，SC_3 はそれぞれステートチャートを表す．SC_1 を SC_3 の振舞いを用いて近似し，SC_3 を SC_2 の振舞いを用いて近似している．さらに，近似した SC_3 を用いて SC_2 を近似する．このように，あるステートチャートを，別のステートチャートの振舞いを用いて近似する

表 1 $\{true\}T_0\{n \geq 0\}$ の導出過程
Table 1 Derivation process for $\{true\}T_0\{n \geq 0\}$.

	導出できる A-TSE	理由
1	$\{n \geq 0\}?inc\{n \geq 0\}$	
2	$\{n \geq 0\}[[n := n+1]]\{n \geq 1\}$	
3	$\{n \geq 1\}!a\{n \geq 1\}$	
4	$\{n \geq 1\}?dec\{n \geq 1\}$	
5	$\{n \geq 1\}[[n := n-1]]\{n \geq 0\}$	
6	$\{n \geq 0\}\varepsilon\{n \geq 0\}$	
7	$\{0 \geq 0\}[[n := 0]]\{n \geq 0\}$	
8	$\{n \geq 0\}?inc; [[n := n+1]]; !a\{n \geq 1\}$	1,2,3, concat
9	$\{n \geq 0\}?inc; [[n := n+1]]; !a\{n \geq 0\}$	8, conseq
10	$\{n \geq 0\}?inc; [[n := n+1]]; !a; ?dec; [[n := n-1]]\{n \geq 0\}$	4,5,9, concat
11	$\{n \geq 0\}(?inc; [[n := n+1]]; !a; ?dec; [[n := n-1]]) * \{n \geq 0\}$	10, iteration
12	$\{n \geq 0\}?inc; [[n := n+1]]; !a + \varepsilon\{n \geq 0\}$	6,9, choice
13	$\{n \geq 0\}(?inc; [[n := n+1]]; !a; ?dec; [[n := n-1]])*; (?inc; [[n := n+1]]; !a + \varepsilon)\{n \geq 0\}$	11,12, concat
14	$\{true\}[[n := 0]]\{n \geq 0\}$	7, conseq
15	$\{true\}[[n := 0]]; (?inc; [[n := n+1]]; !a; ?dec; [[n := n-1]])*; (?inc; [[n := n+1]]; !a + \varepsilon)\{n \geq 0\}$	13,14, concat

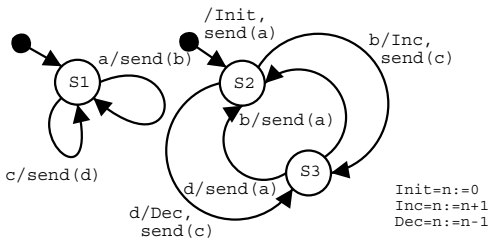

図 2 依存するステートチャート
Fig. 2 Dependent statechart.

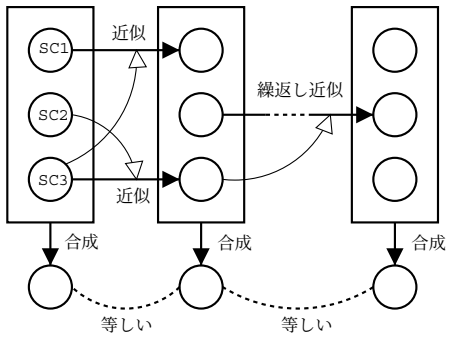

図 3 合成と近似
Fig. 3 Composition and approximation.

ことを，検証ができるようになるまで繰り返し行う．また，近似前の 3 つのステートチャートを合成したものと，近似後の 3 つのステートチャートを合成したものは，等しい TSE によって表すことができる．つまり，近似前と近似後では，全体としての振舞いに変化はないが，個々のステートチャートの振舞いは少しずつ詳細になる．このような近似手法を用いると，合成ほど詳細な振舞いを必要としない検証では，近似途中のステートチャートを用いて検証を行うことができる．

以降では，まず最初に，TSE に対する合成と近似の形式的な定義を示す．そして，合成と近似の関係を明らかにした後に，近似手法を図 2 の例に適用する．

4.1 TSE の合成

ステートチャートの合成を行うためには，それぞれのステートチャートがどのようにイベントを送受信するのかというイベント通信方法を与えなくてはならない．合成は，すべての遷移の組合せを求めてから，与えれたイベント通信方法では不可能な遷移の組合せを取り除くことによって求めることができる．そのため，TSE の合成は，遷移の組合せを求める**インターリーブ演算子** $\times$ と，与えられたイベント通信方法では不可能な遷移の組合せを取り除き，元のステートチャートの部分的な振舞いを求める**除去関数** ρ を用いて定義する．

定義 1（インターリーブ演算子） t_1，t_2 を ε，$\perp$ を除く TSE のプリミティブとし，$T_1 = t_1; T_1'$ と書いて，T_1 は T_1' という TSE の先頭に t_1 を連接したものであるとする．このとき，インターリーブ演算子 $\times$ は，任意の TSE T_1，T_2 に対して次のとおりに定義する．

$$T_1 \times T_2 \equiv \begin{cases} T_1 \times T_2' + T_1 \times T_2'' & (T_2 = T_2' + T_2'') \\ T_1' \times T_2 + T_1'' \times T_2 & (T_1 = T_1' + T_1'') \\ t_1; (T_1' \times t_2; T_2') & (T_1 = t_1; T_1' \\ \quad + t_2; (t_1; T_1' \times T_2') & \quad \wedge T_2 = t_2; T_2') \\ T_2 & (T_1 = \varepsilon) \\ T_1 & (T_2 = \varepsilon) \\ \perp & (T_1 = \perp \vee T_2 = \perp) \end{cases}$$

ここで，定義に '*' に関する場合分けがないのは，任意の TSE は以下に示す ppTSE（primitive prefixed

TSE）に変形できるためである．

定義 2（ppTSE）

$$\begin{array}{rcl} \text{ppTSE} & = & \text{primitive ; TSE} \\ & | & \text{TSE} + \text{TSE} \\ & | & \varepsilon \\ & | & \perp \end{array}$$

ppTSE は，TSE がつねに ‘+’ によって結合されているか，あるいはプリミティブが先頭に現れる形式の TSE のことである．任意の TSE が ppTSE に変形可能であることの証明は省略するが，TSE の構造に沿った帰納法を用いることによって証明した．ppTSE を用いると，関数や演算子の定義を簡潔に記述できるため，以降の関数や演算子の定義においても，ppTSE に対する定義を行っている．

任意の 2 つの TSE T_1，T_2 に対して，$T_1 \times T_2$ の結果は TSE となる．また，以下のとおりの交換則と結合則が成り立つ．

$$T_1 \times T_2 = T_2 \times T_1 \quad \text{交換則}$$
$$(T_1 \times T_2) \times T_3 = T_1 \times (T_2 \times T_3) \quad \text{結合則}$$

$\prod_{T \in TS} T$ は，TSE の集合 TS に含まれるすべての TSE をインターリーブ演算子で結合したものを表し，以下のとおりに定義する．

$$\prod_{T \in TS} T \equiv \begin{cases} T_0 \times T_1 \times \cdots \times T_n & (TS = \{T_0, T_1, \cdots, T_n\}) \\ \perp & (TS = \{\}) \end{cases}$$

合成関数 C の定義は，次のとおりである．

定義 3（合成関数） ρ を除去関数，TS を合成するステートチャートを表す TSE の集合，E をそれらのステートチャートどうしの通信で用いられるイベントの集合とする．このとき，次のとおりに定義される C を合成関数と呼ぶ．

$$C(TS, E) \equiv \rho(\prod_{T \in TS} T, E)$$

このように，通信に用いられるイベントを**内部イベント**と呼ぶ．採用される通信方法によって ρ の定義は異なるが，どのような通信方法も次のような制約を満たすものとする．

定義 4（除去関数） 除去関数とは，TSE とイベントの集合から TSE を返す以下の制約を満たす関数のことである．

(1) $\rho(T, \{\}) = T$
(2) $\rho(T, E) \subseteq T$
(3) $\rho(\perp, E) = \perp$
(4) $\rho(\varepsilon, E) = \varepsilon$
(5) $\rho(T_1 + T_2, E) = \rho(T_1, E) + \rho(T_2, E)$
(6) $\rho(t; T, E) = \rho(t; \rho(T, E - E_e(t)), E)$

ここで，T，T_1，T_2 はそれぞれ任意の TSE を表し，t は TSE のプリミティブを表す．また，$\subseteq$ は TSE に対する包含関係を表しており，

$$T' \subseteq T \equiv \exists T'' \,.\, T = T' + T''$$

と定義する．そして，T' は T の**部分 *TSE*** と呼ぶ．$E_e(T)$ は，TSE T に出現する入力イベントと出力イベントの集合を返す．

この定義において，ρ の第 2 引数であるイベント集合は，ステートチャート間で送受信される内部イベントの集合を表す．内部イベントが存在しないときには，ステートチャート間の通信は行われないので，通信方法によって除去されるべき遷移が存在しないことを (1) によって表している．(2) は，いくつかの遷移を除去関数によって除去しても，元のステートチャートの一部分になっていることを表している．この性質によって，合成が，与えられたすべての TSE をインターリーブしたものから部分 TSE を取り出すことを保証している．(3) は，エラーとなるものは除去できないことを表し，(4) は，空列はこれ以上何も遷移を除去できないことを表す．(5) は，選択演算子で構成されたそれぞれの TSE に，除去関数を先に適用しても結果が同じであることを表している．(6) は，TSE の一部分に除去関数を先に適用して，さらに全体に除去関数を適用しても同じ結果が得られることを表している．

いくつかのイベント通信方法はこの制約を満たしている．たとえば，イベントが出力されたらすぐに受信されなければならないという同期通信を次のとおりに定義する．

定義 5（同期通信） 同期通信とは，除去関数が定義 4 の (1) から (6) の制約のほかに，少なくとも次の 2 つの制約を満たすことである．

$$(T = !e; ?e; T' \wedge e \in E) \Rightarrow (\rho(T, E) = !e; ?e; \rho(T', E)) \quad (1)$$
$$(\exists t \,.\, T = !e; t; ?e; T' \wedge e \in E) \Rightarrow (\rho(T, E) = \perp) \quad (2)$$

ここで，T は任意の TSE であり，$!e; ?e; T'$ は TSE T' の前に $!e$ と $?e$ を連接したものである．(1) の式は送信されたイベントは遅延なく受信されることを表し，(2) の式は送信されたイベントが遅延して受信されることを禁止している．このようなイベント通信が同期通信である．他に，非同期通信として，イベントの受信に遅延のある通信モデルについても除去関数を用いることで定義できた．

定義 4 の性質を満たす除去関数では，1 つのステー

トチャートから送信されるイベントは，その順序を保ったまま受信され，また，送信されたイベントが消滅せず，必ず受信されるというイベント通信を定義することができる．このため，ネットワークにおける UDP のように送信した順序のとおりにパケットを受信することが保証できない通信を正確に取り扱うことはできない．

このような，インターリーブと除去関数を用いて合成された TSE は，3 章で示した方法を用いて検証を行う．

4.2 TSE の近似

近似では，あるステートチャートを別のステートチャートを用いて振舞いを制限する．たとえば，あるステートチャートの振舞いを，別のステートチャートの出力イベントを受け取れるように変形し，不要な遷移を取り除くという作業である．このような近似のための関数を**近似関数**と呼び，以下のとおりに定義する．

定義 6（近似関数） 以下の性質を満たす関数 γ を合成関数 C に対する近似関数と呼び，$\gamma(T, T', E)$ は，TSE T を TSE T' を用いて近似することを表す．

$$C(\{T, T'\}, E)/T \subseteq \gamma(T, T', E)$$
$$\gamma(T, T', E) \subseteq T$$

ここで，E はイベント集合である．また，$C(\{T, T'\}, E) = \rho(T \times T', E)$ であるので，1 つ目の性質の代わりに，

$$\rho(T \times T', E) \subseteq \gamma(T, T', E)$$

を用いてもよい．$/$ の定義を次に示す．

定義 7（限定演算子） TSE T を TSE L に現れるプリミティブのみを抽出するための演算子 $/$ を以下のとおりに定義する．この関数を限定演算子と呼ぶ．

$$T/L \equiv \begin{cases} (T_1/L)+(T_2/L) & (T = T_1 + T_2) \\ t'; (T'/L) & (T = t; T' \wedge t \in W(L)) \\ T'/L & (T = t; T' \wedge t \notin W(L)) \\ \varepsilon & (T = \varepsilon) \\ \perp & (T = \perp) \end{cases}$$

それぞれの T'，T_1，T_2 は TSE を表し，t は TSE の ε，$\perp$ を除くプリミティブを表す．また，$W(T)$ は T に含まれるプリミティブの集合を表す．たとえば，$(?a; !b; [[a := 1]]+!b)/!a; !b; [[a := 1]]$ では，$!b$ と $[[a := 1]]$ が共通して現れるプリミティブであるため，$!b; [[a := 1]]+!b$ という TSE を得ることができる．

近似関数の定義において，$C(\{T, T'\}, E)$ の T，T' 2 つの TSE に同じプリミティブが存在していると，$C(\{T, T'\}, E)/T$ によって得られた TSE は T の部分 TSE とはならない．近似によって，元の TSE の部分 TSE になるためには，2 つの TSE に同じプリミティブが存在してはならない．このことを TSE の独立性と呼び次のとおりに定義する．

定義 8（独立性） TSE の集合 TS の中の任意の T_i に対して $\prod_{T \in TS} T/T_i = T_i$ が成り立ち，同じ属性が異なる TSE に現れないとき，集合 TS は独立であると呼ぶ．

たとえば，$\{(!e_1; ?e_3)*, (?e_1; !e_3)*, !e_1\}$ という TSE の集合は独立ではない．$!e_1$ が 2 つの TSE に出現しているため $\prod_{T \in TS} T/T_i = T_i$ が成り立たないからである．このような独立性が成り立つとき，次の定理が成り立つ．

定理 1 TS を独立な TSE の集合とし，$T_i \in TS$ とする．このとき，任意の合成関数 C と表明 P，Q に対して，

$$\{P\}T_i\{Q\} \Rightarrow \{P\}C(TS, E)\{Q\}$$

が成り立つ．

証明 定義 3，4 より $C(TS, E)/T_i \subseteq T_i$ が成り立つ．このため，TSE の集合 TS に含まれる任意の TSE T_i と，表明 P, Q に対して，

$$\{P\}T_i\{Q\} \Rightarrow \{P\}C(TS, E)/T_i\{Q\}$$

となる．さらに，TS が独立であることから，

$$\{P\}C(TS, E)/T_i\{Q\} \Rightarrow \{P\}C(TS, E)\{Q\}$$

が成り立つ．このため，

$$\{P\}T_i\{Q\} \Rightarrow \{P\}C(TS, E)\{Q\}$$

が成り立つ．□

本稿では，合成や近似を行う TSE の集合は独立であることを仮定して以降の議論を行う．

4.3 近似の妥当性

我々が提案する検証手法では，近似を繰り返し適用することによって得られた TSE を 3 章の方法を用いて検証を行う．近似の妥当性とは，近似して証明できる性質は，合成を用いても証明できるということである．このことを証明するために，まず補題 1，2 を証明する．

補題 1 TS を TSE の集合，T_i' を TS 中の T_i を T_j を用いて近似した TSE であるとする．また，TS' を TS の中の T_i を T_i' で置き換えたものとするとき，

$$C(TS', E) \subseteq C(TS, E)$$

が成り立つ．

証明 T_i として，任意に T を選び，T を近似したものを T' とする．$T' \subseteq T$ が成り立つため，ある TSE T'' が存在して $T' + T'' = T$ となる．このとき，

$$TS_1 \equiv TS - \{T\}$$

とすると，次の式が成り立つ．

$$C(TS', E)$$
$$= C(\{T'\} \cup TS_1, E)$$

$\subseteq C(\{T'\} \cup TS_1, E) + C(\{T''\} \cup TS_1, E)$
$= \rho(\{T'\} \times \prod_{T \in TS_1} T, E) + \rho(\{T''\} \times \prod_{T \in TS_1} T, E)$
$= \rho(\{T' + T''\} \times \prod_{T \in TS_1} T, E)$
$= C(\{T' + T''\} \cup TS_1, E)$
$= C(\{T\} \cup TS_1, E)$
$= C(TS, E)$

よって，$C(TS', E) \subseteq C(TS, E)$ が成り立つ．□

補題 2 を証明するためには，次の定理が必要である．

定理 2 任意の TSE T_1，T_2 に対して，以下の等式が成り立つ．

$$\rho(T_1 \times T_2, E) = \rho(T_1 \times \rho(T_2, E - E_e(T_1)), E)$$
$$\rho(T_1 \times T_2, E) = \rho(\rho(T_1, E - E_e(T_2)) \times T_2, E)$$

証明は，TSE に対する構造帰納法を用いて行うことができるが省略する．この定理は，合成関数 C を用いて，

$$C(\{T_1, T_2\}, E) = C(\{T_1, C(T_2, E - E_e(T_1))\}, E)$$
$$C(\{T_1, T_2\}, E) = C(\{C(T_1, E - E_e(T_2), T_2\}, E)$$

と置き換えることができる．このため，TSE の集合に対して，一部分を先に合成しても，適切に内部イベントを与えれば，同じ TSE を求めることができることを示している．このような合成に関する性質は，文献 10) などで一般的によく用いられるものであるが，我々が提案する合成・近似手法の枠組においても成り立つ．

補題 2 TS を TSE の集合，T_i' を TS 中の T_i を T_j を用いて近似した TSE であるとする．また，TS' を TS の中の T_i を T_i' で置き換えたものとするとき，

$$C(TS, E) \subseteq C(TS', E)$$

証明 T_i，T_j として任意に T_1，T_2 を選ぶものとする．そして，$T_1' \equiv \gamma(T_1, T_2, E - E')$ とし，T'' は $T_1' + T_1'' = T_1$ を満たす TSE とする．このとき，$E' \equiv E_e(\prod_{T \in TS'} T)$ とすると，

$$C(T_1, T_2, E - E')/T_1 \subseteq \gamma(T_1, T_2, E - E')$$

が成り立つ．ここで，

$$TS_1 \equiv TS - \{T_1\}$$
$$TS_{12} \equiv TS - \{T_1, T_2\}$$
$$\gamma_{c12} \equiv C(T_1, T_2, E - E')$$
$$\gamma_{12} \equiv \gamma(T_1, T_2, E - E')$$

と定義すると，

$C(\{T_1, T_2\} \cup TS_{12}, E) \subseteq C(\{\gamma_{c12}, T_2\} \cup TS_{12}, E)$
$\Rightarrow C(\{T_1, T_2\} \cup TS_{12}, E) \subseteq C(\{\gamma_{12}, T_2\} \cup TS_{12}, E)$
$\Rightarrow C(TS, E) \subseteq C(TS', E)$ (A)

が成り立つ．このため，命題を証明するためには，式 (A) が成り立つことをいえれば十分である．このことは，以下のとおりに証明できる．

$C(\{T_1, T_2\} \cup TS_{12}, E)$
$= \rho(T_1 \times T_2 \times \prod_{T \in TS_{12}} T, E)$
$= \rho((T_1' + T_1'') \times T_2 \times \prod_{T \in TS_{12}} T, E)$
$= \rho((T_1' \times T_2 \times \prod_{T \in TS_{12}} T) + (T_1'' \times T_2 \times \prod_{T \in TS_{12}} T), E)$
$= \rho(T_1' \times T_2 \times \prod_{T \in TS_{12}} T, E) + \rho(T_1'' \times T_2 \times \prod_{T \in TS_{12}} T, E)$
$= \rho(T_1' \times T_2 \times \prod_{T \in TS_{12}} T, E) + \rho(\rho(T_1'' \times T_2, E - E') \times \prod_{T \in TS_{12}} T, E)$ (定理 2)
$= \rho(T_1' \times T_2 \times \prod_{T \in TS_{12}} T, E) + \bot$
$= \rho(T_1' \times T_2 \times \prod_{T \in TS_{12}} T, E)$
$= C(TS_1 \cup \{\rho(T_1 \times T_2, E)/T_1\}, E)$
$\subseteq C(TS_1 \cup \{\gamma(T_1, T_2, E)\}, E)$
$= C(\{\gamma_{c12}, T_2\} \cup TS, E)$

よって，補題 2 が成り立つ．□

近似の妥当性は，形式的には次の定理によって表す．

定理 3（近似の妥当性） TS を TSE の集合とし，C をこれらの集合に対する合成関数とする．また，T_i' は TS の中の TSE T_i を T_j を用いて近似したものとする．このとき，

$$\{P\}T_i'\{Q\} \Rightarrow \{P\}C(TS, E)\{Q\}$$

が成り立つ．ここで，P，Q は表明である．

証明 TS' を TS 中の T_i を T_i' で置き換えたものとする．このとき，定理 1 より，

$$\{P\}T_i'\{Q\} \Rightarrow \{P\}C(TS', E)\{Q\}$$

が成り立つ．一方，補題 1，2 より，

$$C(TS', E) = C(TS, E)$$

が成り立つため，

$$\{P\}C(TS', E)\{Q\} \Leftrightarrow \{P\}C(TS, E)\{Q\}$$

が成り立つ．以上より，定理 3 は成り立つ．□

4.4 近似を用いた検証

本章で提案した近似手法を，図 2 のステートチャートのモデルに適用する．2 つのステートチャートは，互いにイベントの送信と受信が同期して行われるものとする．このとき，$n \geq 0$ がつねに成り立つことを，次の手順によって検証を行う．まず最初に，除去関数を定義することによって，ステートチャート間のイベント通信方法を決定する．次に，この除去関数を用いた場合の近似関数を，4.2 節の制約を満たすように定義する．こうして得られた近似関数を，2 つのステートチャートに相互に繰り返し適用することで，近似されたステートチャートを得る．最後に，近似したステートチャートにおいて $n \geq 0$ が成り立つことを，3 章の方法を用いて検証する．

まず，イベント通信方法を以下の除去関数によって定義する．

$$\rho(T,E) \equiv \begin{cases} \rho(T_1,E)+\rho(T_2,E) & T = T_1 + T_2 \\ !e;?e;\rho(T',E) & T = !e;?e;T' \wedge e \in E \\ t;\rho(T',E) & T = t;T' \wedge t \notin ?E \\ & \wedge t \notin !E \\ \varepsilon & T = \varepsilon \\ \bot & otherwise \end{cases}$$

ここで，T は TSE を表し，t は ε，$\bot$ を除くプリミティブを表す．また，$?E$，$!E$ は，イベントの集合 E の中のすべてのイベントに，それぞれ '?'，'!' を前置した集合である．証明は省略するが，このように定義した除去関数 ρ は，定義 5 の制約を満たす．このため，ρ により定義されるイベント通信方法は同期通信である．

このようなステートチャートにおいて，$n \geq 0$ の不変性の検証を行う．右側のステートチャートにおいて，$n \geq 0$ の検証が行えないのは，左側のステートチャートにおいて，アクション Inc と Dec がどのような順序で実行されるのかが分からないためである．2 つのステートチャートに対して，定義したイベント通信方法に従って合成すると Dec が実行されず，つねに Inc のみが実行されることが分かる．

近似を行うためには，まず最初に近似関数を次のとおりに定義する．

$$\gamma(T_1,T_2,E) \equiv \begin{cases} \gamma(T_1',T_2,E)+\gamma(T_1'',T_2,E) & T_1 = T_1'+T_1'' \\ \gamma(T_1,T_2',E)+\gamma(T_1,T_2'',E) & T_2 = T_2'+T_2'' \\ ?e;\gamma(T_1',T_2',E) & T_1 =?e;T_1' \\ & \wedge T_2 =!e;T_2' \wedge e \in E \\ t;\gamma(T_1',T_2,E) & T_1 = t;T_1' \wedge t \notin ?E \\ \gamma(T_1,T_2',E) & T_1 =?e;T_1' \\ & \wedge T_2 = t_2;T_2' \wedge t_2 \notin !E \\ \varepsilon & T_1 = \varepsilon \\ \bot & otherwise \end{cases}$$

ここで，T_1，T_2 は TSE を表し，t_1，t_2 は ε，$\bot$ を除くプリミティブを表す．この近似関数 γ は，T_2 に現れる出力イベントを，出力された順序を保ったまま受け取れるように T_1 の振舞いを制限する．そして，2 つのステートチャートに対してどのように適用しても γ に関する制約 $\rho(T \times T', E)/T \subseteq \gamma(T, T', E)$ と，$\gamma(T, T', E) \subseteq T$ を満たす．このことは，定義に沿った帰納法を用いることによって証明した．

Inc と Dec は，それぞれイベント b と d を受け取ることで実行される．そのため，左側のステートチャートにおいてイベント b と d がどのような順序で出力されるのかが分かると，Inc と Dec がどのような順序で実行されるのかが分かる．そこで，まず左側のステートチャートを右側のステートチャートを用いて近似する．左側と右側のステートチャートに対する TSE を，それぞれ L，R とすると，$\gamma(L, R, \{a, b, c, d\})$ によって左側のステートチャートを近似することができる．ここで，L と R はそれぞれ次のとおりである．

$$L = (?a; !b + ?c; !d) *$$
$$R = Init; !a; (((?b; Inc; !c + ?d; Dec; !c); (?b; !a + ?d; !a)) *); (?b; Inc; !c + ?d; Dec; !c + \varepsilon)$$

L を R を用いて近似した TSE を L' とし，R を L' を用いて近似した TSE を R' とすると，R'，L' はそれぞれ次のとおりになる．

$$L' = \gamma(L, R, \{a, b, c, d\}) = (?a; !b; (?c; !d + \varepsilon)) *$$
$$R' = \gamma(R, L', \{a, b, c, d\}) = Init; !a; (?b; Inc; !c; ?d; !a) *$$

そして，R' では，Inc のみが実行されることが分かる．検証したいことは，$n \geq 0$ という不変性であるため，$\{true\}R'\{n \geq 0\}$ を証明する．この A-TSE は，3 章で示した方法によって証明できる．証明の詳細は **表 2** に示す．

表 2 $\{true\}R'\{n \geq 0\}$ の証明
Table 2 Proof for $\{true\}R'\{n \geq 0\}$.

	導出できる A-TSE	前提	推論規則
1	$\{true\}Init\{n \geq 0\}$	公理	
2	$\{n \geq 0\}!a\{n \geq 0\}$	公理	
3	$\{n \geq 0\}!b\{n \geq 0\}$	公理	
4	$\{n \geq 0\}Inc\{n \geq 1\}$	公理	
5	$\{n \geq 1\}!c\{n \geq 1\}$	公理	
6	$\{n \geq 1\}?d\{n \geq 1\}$	公理	
7	$\{n \geq 1\}!a\{n \geq 1\}$	公理	
8	$\{n \geq 1\}!a\{n \geq 0\}$	7	conseq
9	$\{n \geq 0\}$ $?b;;Inc;;!c;;?d;;!a$ $\{n \geq 0\}$	3,4,5,6,8	concat
10	$\{n \geq 0\}$ $(?b;;Inc;;!c;;?d;;!a)*$ $\{n \geq 0\}$	3,4,5,6,8	iteration
11	$\{true\}R'\{n \geq 0\}$	1,2,10	concat

5. 考　　察

5.1 論理 ATL の証明能力

3 章において示した検証手法では，すべての状態に到達可能であることを前提として TSE を構成する．このため，決して到達することのない状態では，属性に関する性質は無条件に成り立つものとしている．たとえば，**図 4** では，状態 S_2 には決して到達することはないステートチャートである．このステートチャートに対する TSE は，$[[n := 0]]; (([n = 0]; [[n := n - 1]]) * + [n > 0] + \varepsilon)$ である．つねに $n \leq 0$ であることを証明する

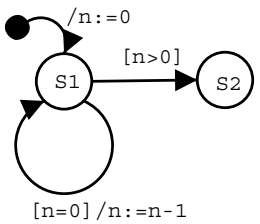

図 4 到達できない状態が存在するステートチャート
Fig. 4 Unreachable state in statechart.

場合，$\{n \leq 0\}[n > 0]\{n \leq 0\}$ が導出できなければならない．公理より，$\{n \leq 0\}[n > 0]\{n \leq 0 \wedge n > 0\}$ となるため，推論規則 conseq を用いて $\{n \leq 0\}[n > 0]\{false\}$ を導出できる．任意の論理式 P に対して，$false \Rightarrow P$ は恒真式であるため，事後表明にどのような論理式が書かれていても推論規則 conseq により導くことができる．そのため，$\{n \leq 0\}[n > 0]\{n \leq 0\}$ は導出可能である．このように，ある遷移が決して発火しない場合には，事後表明に関係なく A-TSE が成り立つ．

5.2 近似関数の柔軟性

定義 6 では，近似関数に対する制約だけを与えて，1 つの通信方法に対して複数の近似関数を柔軟に定義できるようにした．このように複数の近似関数を定義できると，対象とする検証に対して，できるだけ簡単な近似関数を選択することができる．また，どのような情報をステートチャートに与えれば検証が行えるようになるのかを考えることが，近似関数を選択する基準となる．

たとえば，除去関数 ρ に対して最も高い精度を得ることができる近似関数 γ は次のとおりの定義である．

$$\gamma(T_1, T_2, E) = \rho(T_1 \times T_2, E)/T_1$$

この定義において，$\rho(T_1 \times T_2, E)$ は T_1 と T_2 の合成そのものであるため，合成と同じだけ計算が複雑になる．ところが，4.4 節の例題では，イベントの受信順序だけを考えれば検証を達成できる．この事実を近似関数の選択の基準として，合成ほど複雑な計算が必要ではない近似関数を選択して用いている．

対象とする検証に対して，計算が簡単になるような適切な近似関数を発見することができると，振舞い近似手法によって検証コストを下げることができる．本稿では，近似関数によって合成にともなう検証コストを下げることができる場合があることを示し，近似関数と合成関数の関係を明らかにした．

5.3 近似関数の再利用性

本稿で示した近似関数は，発見的手法によって定義を行った．そして，与えられたイベント通信方式に対して，近似関数の制約を満たしていることを証明することによって近似関数であることを確認した．除去関数は制約を満たす限り自由に定義できるものであるが，制約を満たす除去関数を発見することは簡単には行えない．しかし，通信方式は，個々のステートチャートとは独立して定義できるため，一度近似関数を発見すれば，異なるステートチャートであっても，同じ通信方式に対しては発見した近似関数をそのまま再利用することができる．

さらに，ある通信方法に対する近似関数は，別の通信方法に対する近似関数ともなりうる．この近似関数と除去関数の関係は，通信方法を規定する除去関数に関係を与えることによって形式的にとらえることができる．

たとえば，本稿で示した近似関数は，定義 5 で示した同期通信に対する近似関数であるが，この近似関数は，送信されたイベントを遅延して受け取ることが可能な非同期通信に対する近似関数でもある．同期通信に対する除去関数と，非同期通信に対する除去関数をそれぞれ ρ_s，ρ_a とすれば，$\rho_s \subseteq \rho_a$ という関係が成り立ち，ρ_s は ρ_a よりも強い除去関数であるという．この関係の定義は以下のとおりの簡単な定義である．

$$\rho_1 \subseteq \rho_2 \equiv \forall T, E \,.\, \rho_1(T, E) \subseteq \rho_2(T, E)$$

ここで，ρ_1，ρ_2 が除去関数であり，T，E はそれぞれ TSE，イベント集合を表す．つまり，ある除去関数に対する近似関数は，その除去関数より強い除去関数の近似関数でもある．

それぞれの通信方法に対する除去関数と，それらの除去関数に対する近似関数の対応関係を蓄積していくことによって，検証者が近似関数を発見し，近似関数であることを証明する手間を軽減できる．

5.4 近似手法の拡張

我々が提案した近似手法では，それぞれのステートチャートの振舞いを制限していくだけである．片方のステートチャートのアクションやガード条件を，他方のステートチャートに採り入れる手段は提供していない．なぜならば，近似手法において，他方のステートチャートのアクションやガード条件が追加して含まれてしまうと，近似によって部分 TSE の関係が保たれるという重要な性質が失われてしまうからである．たとえば，**図 5** は，近似手法を用いて $m \leq 10$ を証明することができない例である．$m \leq 10$ を証明するためには，$m \leq n$ と $n \leq 10$ を導く必要がある．ここで，m と n を比較できるようにするためには，2 つのステートチャートのアクションやガード条件を証明において使えるようにする必要がある．そこで，証明

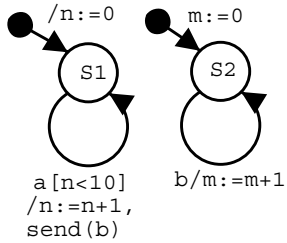

図 5 近似手法では証明できない例
Fig. 5 An example we can't verify with the approximation method.

過程に現れる不変表明を含めて近似を適用できるような枠組みを考えている．現在の近似手法は TSE に対する手法であるが，この手法を A-TSE に適用できるように拡張するのである．拡張した近似手法を用いると，表明には他のステートチャートの属性が含まれるが，表明を取り除いて得られる TSE は，近似する前の TSE の部分 TSE にすることができると考えているためである．

5.5 イベント属性

本稿で提案した検証手法では，イベント属性を扱っていない．イベントに値を付けて出力したり，イベントが値をともなって入力されるということは論理 ATL において導入されるべき概念である．しかし，片方のステートチャートから他方のステートチャートへイベント属性が渡ることはイベント通信方法の中で定義されることである．一方で，我々が提案した検証手法では，検証者が自由にイベント通信方法を決めることができるようにしている．このため，イベント属性を扱えるようするためには，論理 ATL もイベント通信方法に応じて柔軟に変更できるようにする必要があると考えている．

6. 関 連 研 究

6.1 モデル検査手法と状態システムの合成

有限オートマトンに対するモデル検査手法[2] に関する研究がある．この手法では，可能な状態を全探索することによって与えられた性質を自動的に検証する．しかし，複数の依存し合う有限オートマトンを検証するために合成を行うため，状態爆発という問題が発生する．さらに，属性をともなうオートマトンを検証するためには，属性の値をまとめることによって，状態として扱うという工夫が必要となる．しかし，どのように属性値を状態として扱うかということを発見することが難しい．

一方，ラベル付き遷移システムにおいて，合成時の検証コストを下げる手法が提案されている．まず，分散プログラムのアーキテクチャを検証する手法に *Compositional Reachability Analysis*（CRA）[1] がある．この手法では，分散プログラムのアーキテクチャを階層化されたサブシステムを用いて表し，それぞれのプロセスの仕様をラベル付き遷移システムによって表現する．それぞれのサブシステムでは，サブシステム内部で起こるアクションを外部から隠蔽することができる．ラベル付き遷移システム Q に対して，外部から観測可能なアクションの集合を L とするとき，外部から観測可能なプロセスの振舞いを $Q \uparrow L$ と表す．検証すべき性質は，有限オートマトンを用いて記述する．これを性質オートマトンと呼ぶ．検証対象のシステム Z と性質オートマトン T に対して，

$$tr(Z \uparrow \alpha T) \subseteq T$$

となることが安全性である．ここで，tr はトレースを表し，これは本研究における TSE と等しい概念を表している．また，αT は，T が受理できるアクションの集合を表している．つまり，性質オートマトンで受理できないトレースが存在しないことが安全性の検証である．そして，このような安全性の検証を可達性に帰着させるためにイメージオートマトン（image automaton）というものを性質オートマトンから導く．このイメージオートマトンでは性質オートマトンで受理できない状態を π という特別な状態で表している．このようにして，観測可能な振舞いを用いて検証を行うため，内部アクションを考慮する必要をなくし，検証コストを下げている．ここで，$Z \uparrow \alpha T$ というのは，本研究における近似の枠組みにおいて取り扱うことができ，$tr(Z \uparrow \alpha T) \subseteq T$ という安全性の性質は，部分 TSE の概念を用いて説明できる．また，システムに階層性を持たせ，内部アクションを隠蔽できる点は，我々がステートチャートの独立性として定義した性質と同じものである．このように，本研究で提案した近似手法の枠組みにおいて，CRA の安全性検証の説明ができる．さらに近似手法は，様々な通信方法に対応しているため，CRA の手法よりも一般的なものである．しかし，検証すべき性質は，本稿で提案した属性の不変性の検証とは異なるものである．このことから，振舞い近似手法は属性に対する不変性検証以外にもいくつかの検証にも有効であると考えている．

我々が提案した手法は安全性の検証手法であり，文献 1) と同様に検証に必要のない振舞いを取り除く．しかし，文献 1)，2) のいずれも属性を持つ遷移システムにおいて同様の手法を適用する試みは行われていない．さらに，本稿では，通信方法を一般化し，様々

な通信方法にもこのような手法が適用できることを示した．

6.2 定理証明技術を用いた検証

CSP [5] や独自の言語で仕様を記述し定理証明技術を用いて検証を行う方法[3),9)] がある．これらは，各々の遷移の性質から全体の振舞いの性質を証明する方法である．一方で，我々はステートチャートを TSE へ変換することによって，Hoare のプログラム検証と同様の検証が行えることを示した．そして，本稿では，振舞い近似手法を TSE に対して適用しているが，CSP などの言語へ適用することも可能であると考えている．また，文献 9)，11) では，ぞれぞれ HOL と PVS と呼ばれる定理証明器を用いて検証を行う方法を紹介している．特に本研究は，文献 11) と連携した研究であり，本稿で示した検証手法を計算機を用いて支援することが期待できる．

7. ま と め

本稿では，ステートチャートに関する不変性の検証を行うために，互いに補完的な 2 つの手法を提案した．1 つは，ステートチャートを TSE に変換し，表明を与えて論理を用いて証明する手法である．提案した論理は健全かつ完全なものである．他方は，複数のステートチャートにおいて，互いに繰り返し近似を行うことによって振舞いを制限する手法である．近似関数は，条件を満たす限り検証者が自由に定義できる．そのため，ステートチャートのイベント通信方法に依存することなく，振舞い近似手法を繰り返し適用することができる．また，計算が簡単になるような近似関数を発見することによって，検証コストを下げることができる．

最後に，本稿は，著者が北陸先端科学技術大学院大学在籍中に行った研究の成果をまとめたものであることを付記する．

参 考 文 献

1) Cheung, S.C. and Kramer, J.: Checking Subsystem Safety Properties in Compositional Reachability Analysis, *18th International Conference on Software Engineering* (1996).
2) Clarke, O.E.M. and Peled, D.: *Model Checking*, MIT Press (2000).
3) Smith, J.D.G.: Specification, Refinement and Verification of Concurrent Systems—An Integration of Object-Z and CSP, *Formal Methods in System Design*, pp.249–284 (2001).
4) Harel, D.: Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol.8, No.3, pp.231–274 (1987).
5) Hoare, C.: *Communicating Sequential Processes, International Series in Computer Science*, Prentice-Hall (1985).
6) Leeuwen, J.V. (Ed.): *Handbook of Theoretical Computer Science, Vol.B: Formal Models and Semantics*, MIT Press (1994).
7) Milner, R.: *Communicating and mobile systems: the π-calculus*, Cambridge University Press (1999).
8) OMG: *Unified Modeling Language v1.3*, OMG (2000).
9) Graf, S. and Saidi, H.: Verifying Invariants Using Theorem Proving, *Proc. 8th International Conference on Computer Aided Verification CAV*, Alur, R. and Henzinger, T.A. (Eds.), Vol.1102, pp.196–207, New Brunswick, NJ, USA, Springer Verlag (1996).
10) Garg, V.K. and Ragunath, M.T.: Concurrent Regular Expressions and Their Relationship to Petri Nets, *Theoretical Computer Science*, Vol.96, pp.285–304 (1992).
11) 青木利晃，立石孝彰，片山卓也：定理証明技術のオブジェクト指向分析への適用，日本ソフトウェア科学会学会誌コンピュータソフトウェア，Vol.18, pp.18–47 (2001).

付 録

A.1 健全性と相対完全性

論理 ATL が健全であるとは，論理 ATL を用いて導出した A-TSE は正しいことを表す．また，論理 ATL が相対完全であるとは，推論規則 conseq における $P' \Rightarrow P$ と $Q \Rightarrow Q'$ が論理 ATL 以外の体系，たとえば，算術計算の体系などで証明可能であれば，正しい A-TSE は論理 ATL を用いて導出できることである．正しい A-TSE を定義するために，表明に現れる自由変数に対する解釈と属性環境を用いて操作的意味論を A-TSE に与えた．属性環境とは，属性から属性値を求めるための関数である．

定義 9 σ，σ' を属性環境，I を自由変数に対する解釈とする．

$$I \models \{P\}T\{Q\} \equiv \forall \sigma, \sigma' \,.\, I, \sigma \models P \Rightarrow Tr(T, \sigma, \sigma') \Rightarrow I, \sigma' \models Q$$

$$\models \{P\}T\{Q\} \equiv \forall I \,.\, I \models \{P\}T\{Q\}$$

ここで，$\models \{P\}T\{Q\}$ のとき，$\{P\}T\{Q\}$ が正しいと呼ぶ．

$Tr(T, \sigma, \sigma')$ は，次のとおりに定義する．

定義 10 $Tr(T, \sigma, \sigma')$ は，TSE T の前後で属性環境が σ から σ' に変化することを表し，以下の性質が

成り立つ．

(1) $\forall\sigma\,.\,Tr(\varepsilon,\sigma,\sigma)$

(2) $\forall\sigma,\sigma'\,.\,\neg Tr(\bot,\sigma,\sigma')$

(3) $\forall\sigma\,.\,Tr(v:=t,\sigma,\sigma[t/v])$

(4) $\forall\sigma\,.\,Tr(!e,\sigma,\sigma)$

(5) $\forall\sigma\,.\,Tr(?e,\sigma,\sigma)$

(6) $\forall\sigma\,.\,Tr([t],\sigma,\sigma)\Rightarrow t$

(7) $\forall\sigma,\sigma'\,.\,Tr(T_1;T_2,\sigma,\sigma')$
$\Leftrightarrow(\exists\sigma''\,.\,Tr(T_1,\sigma,\sigma'')\wedge Tr(T_2,\sigma'',\sigma'))$

(8) $\forall\sigma,\sigma'\,.\,Tr(T_1+T_2,\sigma,\sigma')$
$\Leftrightarrow Tr(T_1,\sigma,\sigma')\vee Tr(T_2,\sigma,\sigma')$

(9) $\forall\sigma,\sigma'\,.\,Tr(T*,\sigma,\sigma')\Leftrightarrow Tr(T;T*+\varepsilon,\sigma,\sigma')$

そのため，健全性と相対完全性は次のとおりに定義する．

定義 11（健全性） 以下の式が成り立つとき，論理 ATL が健全である．

$\vdash\{P\}T\{Q\}\Rightarrow\models\{P\}T\{Q\}$

定義 12（相対完全性） 以下の式が成り立つとき，論理 ATL が相対完全である．

$\models\{P\}T\{Q\}\Rightarrow\vdash\{P\}T\{Q\}$

これらの定義に従い，提案した体系は健全かつ相対完全である．健全性と完全性の証明は省略する．

(平成 14 年 9 月 30 日受付)
(平成 15 年 4 月 3 日採録)

立石 孝彰（正会員）

1976 年生．東京理科大学理工学部情報科学学科（1994〜1998），北陸先端科学技術大学院大学博士前期課程（1998〜2000）を経て同大学博士後期課程（1998〜2003）を修了．工学博士を取得（2003）．形式的手法，オブジェクト指向方法論の研究に従事．2003 年 4 月，日本アイ・ビー・エム株式会社に入社．

青木 利晃（正会員）

1971 年生．1994 年東京理科大学理学部応用数学科卒業．1999 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学助手．2001 年科学技術振興事業団さきがけ研究 21 研究者兼任．形式的手法，オブジェクト指向分析/設計，組込みオブジェクト指向開発法に関する研究に従事．ソフトウェア科学会，IEEECS 各会員．

片山 卓也（正会員）

1939 年生．東京工業大学電気工学科（1962），同修士（1964），工学博士（1971），日本 IBM 株式会社（1964），東京工業大学工学部助教授（1974），教授（1985）を経て北陸先端科学技術大学院大学教授（1991）．現在附属図書館長．情報処理学会理事（1985〜1986）．日本ソフトウェア科学会理事長（1991〜1995）．文科省特定領域研究「ソフトウェア発展」研究代表者（1997〜2000）．学術振興会未来開拓研究プロジェクト「ソフトウェア開発方法論」プロジェクトリーダ（1996〜2001）．科学技術振興事業団さきがけ研究 21「機能と構成」領域総括（2000〜），通信放送機構北陸 IT 研究開発センター長（2002〜），形式的オブジェクト指向方法論，その組込みシステム開発への適用，ソフトウェア発展機構の研究に従事．