博 士 論 文

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

指導教官　片山 卓也 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

立石 孝彰

平成 15 年 2 月 14 日

要 旨

開発対象システムや分散オブジェクトなどの振舞いをモデル化するためにステートチャートと呼ばれる状態遷移システムいることが多い．互いにイベントを用いて協調動作を行う複数のステートチャートにおいては，検証対象となるステートチャートの振舞いは，他のステートチャートの振舞いに依存する．このような依存関係のあるステートチャートの検証を行う場合，互いの相互作用を考慮して検証を行わなければならない．このための一般的な手法は，すべてのステートチャートを合成して，一つのステートチャートを構成することであるが，しばしば状態爆発という問題が起こる．ところが，他のステートチャートから送信されるイベントが，どのように検証対象のステートチャートに作用するかを考慮することによって，対象ステートチャートを合成後の振舞いに近似することができる．このような近似を用いた手法は，繰返し適用することができ，検証に必要なだけステートチャートの振舞いを詳細にすることができる．

本研究では，安全性の検証を行うことが目的であり，属性値に対する不変性を検証するための手法を考案した．そして，近似されたステートチャートで検証できる性質は，合成されたステートチャートでも検証できる．このような振舞い近似手法を，様々な通信方法に適用できるように汎用性を持たせて定義する．

この不変性の検証のために，ステートチャートを TSE(遷移列式, Transition Sequence Expression) と呼ぶ正規表現に類似した式を用いて表した．ステートチャートから TSE を求める方法は，有限オートマトンを正規表現に変換する方法と同じであり，自動的に遷移列式を求めることができる．単一ステートチャートに対する検証では，TSE の前後に表明を付け，表明が正しいことを証明する．このような遷移列式を A-TSE(表明付き TSE) と呼ぶ．この証明には演繹的体系を用いて行うため，無限の属性値を持ち得る属性の検証を行うことができた．用いた演繹的体系は，Hoare 論理に類似した体系を用いる．そして，提案する体系が，表明付き遷移列式の意味に対して健全かつ完全であることを証明した．

目 次

図 目 次

表 目 次

第 1章

はじめに

開発対象システムや分散オブジェクトなどの振舞いをモデル化するためにステートチャートを用いることが多い．ステートチャートは状態遷移図に基づくモデルである．複数のステートチャートは互いにイベントを用いて通信を行い協調して動作する．このため，複数のステートチャートを記述することによって，システム全体の振舞いを記述できる．

しかし，各々のステートチャートを独立に記述するため，互いに協調動作をさせた結果，予期していない動作が起こることがある．このような動作を避けるために，記述されたステートチャートが正しい動作を行うことを検証する必要がある．

ステートチャートの検証では，有限オートマトンに対する検証手法が有効であり，モデル検査 [EOD00] と呼ばれる手法がある．この手法では，すべての状態を探索することによって，与えられた性質が満たされるかどうかの検証を行う手法である．しかし，ステートチャートを用いたモデルでは，無限の値を持つ属性を使うことがしばしばあるため，モデル検査手法によって検証を行うことが困難な場合がある．そこで，本論文では無限の値を持つ属性を扱えるような検証手法を提案する．

提案する検証手法では，ステートチャートを TSE(遷移列式, Transition Sequence Expression) と呼ぶ正規表現に類似した式を用いて表す．ステートチャートから TSE を求める方法は，有限オートマトンを正規表現に変換する方法と同

じであり，自動的に遷移列式を求めることができる．

まず，単一ステートチャートに対する検証では，TSE の前後に表明を付け，表明が正しいことを証明する．このような TSE を表明付き TSE と呼ぶ．この証明には，プログラム検証で用いられる Hoare の形式的体系に類似した体系を用いる．提案する形式的体系は，健全かつ完全である．

互いに振舞いが依存する複数のステートチャートの検証を行う場合，通常，システム全体の振舞いを表す 1 つのステートチャートに合成して検証する．しかし，合成されたステートチャートは複雑になり，検証も複雑で長いものになることが多い．

本論文では，他のステートチャートの振舞いを考慮することによって，合成したステートチャートでは実際に起こることがない振舞いを排除することができる点に着目した．実際に起こることがない振舞いを取り除くと，検証対象となるステートチャートの振舞いを実際に起こる振舞いに近づけることができる．このことを**振舞い近似**と呼ぶ．複数のステートチャートにおいて，実際に起こり得ない振舞いを取り除くことは，お互いに繰返し行うことができる．このため，振舞いの近似手法は，繰返し適用することができ，検証に必要なだけステートチャートを詳細化することができるようになる．本研究では，このような振舞い近似手法を形式的に定義する．

一方，現実問題にステートチャートをモデルを用いる場合，イベントの通信方式には様々なものがある．また，ステートチャートの合成方法は，イベントの通信方法に依存する．本研究では，イベントの通信方法を形式的に取扱い，振舞い近似との関係を明らかにする．この結果，近似の方法に自由度を持たせることができる．そのため，検証者は，その自由度の範囲において，検証に都合が良いように近似方法を独自に定義することができる．

本論文の構成は次の通りである．3 章では，まず始めに検証対象となるステートチャートの構成要素を示す．検証は，遷移列式を用いて行うため，遷移列式の形式的な定義と，ステートチャートからの変換方法を述べる．4 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．5 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，7 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．8 章では，関連する研究を紹介し，本論文で提案する手法と比較する．最後に 9 章において，本論文のまとめを行う．

第 2章

背景と目的

2.1 背景

2.1.1 ステートチャート

ステートチャートは，Harel[Har87] により提案され，その後，オブジェクト指向分析・設計モデルや分散システム，リアクティブシステムのモデルにおいて，オブジェクトやシステムの振舞いを記述するために様々な分野で幅広く使われるようになった．これは，ステートチャートが属性・階層性・並行性の概念を伴い，様々な場面において振舞いを簡潔に記述するための記法を与えたためである．

複数のステートチャートはイベントを介して通信を行い，互いに協調して動作する．このため，個々のオブジェクトの振舞いをステートチャートで表し，オブジェクト間の通信方法を，ステートチャート間のイベント通信方法を用いて表すことができる．オブジェクト毎に振舞いを記述することができるため，それぞれのオブジェクトの振舞いだけに焦点を当てて，モデルを簡潔に記述できる．

ステートチャートの階層性は，振舞いの詳細度に応じて適切に状態と遷移をグループ化することを可能とした．さらに，並行性の概念は近代的なマルチプロセス・マルチスレッドなどを基調としたシステムのモデルには必要不可欠なものである．

2.1.2 オブジェクト指向分析・設計モデル

オブジェクト指向分析・設計モデルではオブジェクトの振舞いを表すためにステートチャートを用いることが多い．近年，標準化されソフトウェア開発に浸透しつつあるモデリング言語 UML[OMG00] においても，オブジェクトの振舞いやシステムの振舞いを表すためにステートチャートが定義されている．UML では，ステートチャートに対する表記法のみを定めているため，イベント通信方法などは一切定義されていない．このため，分析・設計モデルのモデリングの方針に応じて，設計者が適切なイベント通信方法を与える．

このために，例えば，シーケンス図と呼ばれるオブジェクト間のインタラクションと振舞いを時系列で表したモデルにおいて，オブジェクト同士が同期あるいは非同期にメッセージを送受信するための表記などが定義されている．

2.1.3 実開発システム適用時の問題

また，規模が大きいモデルでは複数のステートチャートを用いてシステム全体の振舞いを記述することが一般的であり，ステートチャートの振舞いは互いに依存することが多い．このようなステートチャートを検証する直接的な方法は，すべてのステートチャートを，一つの大きなステートチャートに合成することである．図 2.1 は 2 つの状態から構成されている 2 つのステートチャートを合成した場合の様子を示している．一般的に，n 個のステートチャート $S_0, \cdots, S_{n-1}$ が，それぞれ $N_0, \cdots, N_{n-1}$ 個の状態をもつとき，合成を行うと最悪で $N_0 \times \cdots \times N_{n-1}$ 個の状態を持つ．このため，合成に用いられたステートチャートの個数が増えると，合成されたステートチャートの状態数は爆発的に増加する．実開発に用いられるような規模が大きいモデルの検証では，合成を用いることは現実的ではない．

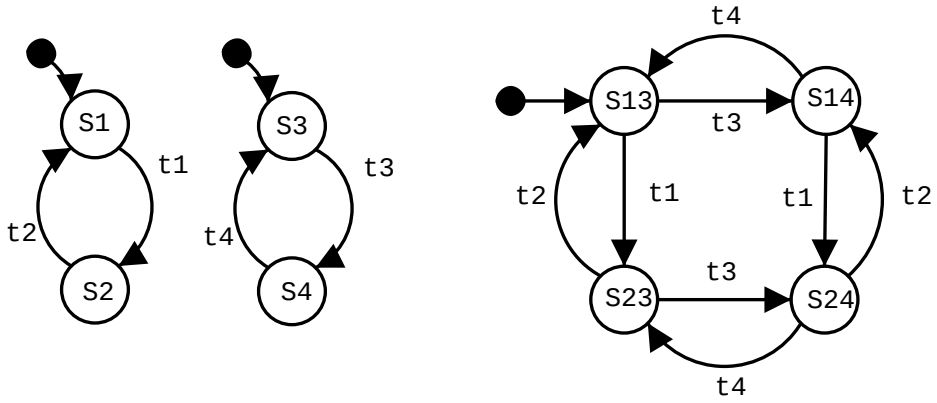

図 2.1: 合成による状態の増加

2.2 目的

本研究の目的は，(1) 無限の属性値を扱える検証手法と，(2) 合成を行わずに互いに振舞いが依存するステートチャートの検証手法を提案することである．(1) の目的のために，本研究では Hoare のプログラム検証の手法に着目した．Hoare の検証手法では，プログラムの正当性を構文主導の演繹的手法を用いて検証する．このため，本研究では，TSE と呼ばれるステートチャートに対する正規表現を用いることで，構文主導の検証を可能とし，演繹的な手法を用いることで無限の値を持ち得る属性に関する検証を可能とする．ステートチャートと正規表現は等価であるため，TSE 上で検証できる性質は，そのまま対応するステートチャートの性質である．(2) を達成するために，個々のステートチャートを合成後の振舞いに近似する方法を用いる．近似の方法は，合成と同じくステートチャート間のイベント通信方法に依存して決定しなければならない．このため，ステートチャート間のイベント通信と近似との関係を明らかにし，近似の操作に必要な制約を明らかにする．

第 3章

ステートチャートと遷移列式

3.1 ステートチャート

ステートチャートは，状態・遷移・属性から構成され，**初期状態**と呼ばれる状態から動作を開始する．遷移がでていない袋小路になっている状態を**終了状態**と呼び，この終了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良い．オブジェクト指向分析モデルや・設計モデル [OMG00] で用いられるステートチャートにおいては，オブジェクトの消滅を意味する場合に終了状態と呼ぶことが一般的であり，行き先のない状態は終了状態とは呼ばない．しかし，本論文ではオブジェクトの生成・消滅という概念は扱わず，行き先のない状態は最終状態として扱うことにする．属性は必ず初期値によって初期化されなければならず，この初期値は，**初期遷移**と呼ばれる遷移によって決定される．初期遷移は必ず 1 つだけ存在し，その遷移先は必ず初期状態でなければならない． 図 3.1 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，黒丸から伸びる遷移が初期遷移を表す．

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)

初期遷移にはアクション式のみを記述し，‘/’ を省いても良い．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

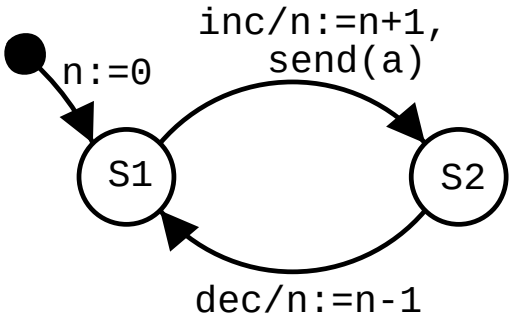

図 3.1: ステートチャート

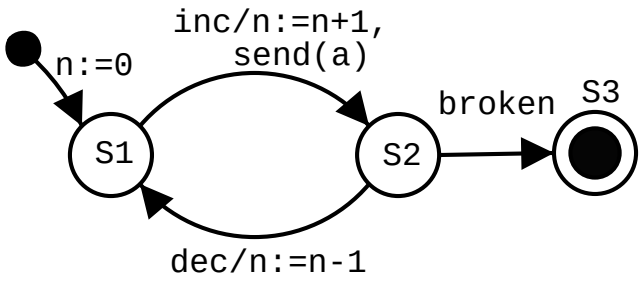

図 3.2: 終了状態のあるステートチャート

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．例えば，図 3.1 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．図 3.2 は，図 3.1 に終了状態 S_3 を付け加えたものである．このように終了状態を明示する場合には黒丸を用いることにする．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．

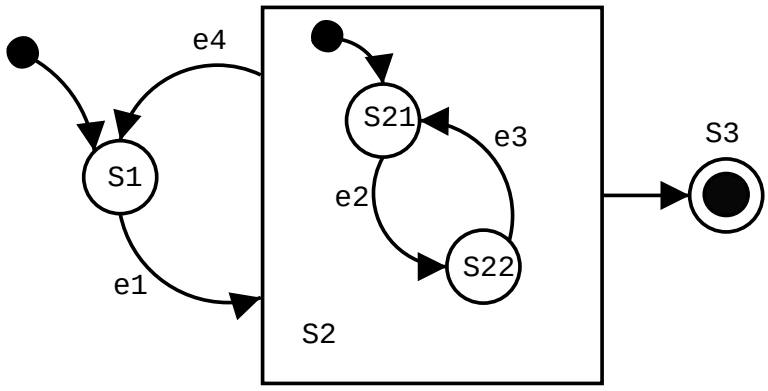

図 3.3: 階層化されたステートチャート

3.2 階層化されたステートチャート

ステートチャートの中にステートチャートを階層化して記述することができる．このようなステートチャートを**サブステートチャート**と呼ぶ．サブステートチャートは，親ステートチャートにおいては状態として扱われる．この状態のことを**複合状態**と呼ぶ．

図 3.3 は，階層化されたステートチャートの例である．このステートチャートでは S_2 が複合状態となっている．まず初期状態 S_1 から動作が始まる．イベント e_1 を受け取ると状態は S_2 に移り，この状態 S_2 は，サブステートチャートとしての動作が始まる．そのため，状態は，初期状態 S_{21} に変化する．次にイベント e_2 を受け取ると状態は S_{22} に変化する．サブステートチャートは，いつでも複合状態 S_2 として扱うことができるため，もし，状態 S_{21} においてイベント e_4 を受け取ると，現在の状態を S_2 として扱い，状態は S_1 に変化する．

3.3 遷移列式 (TSE)

ステートチャートの遷移だけに着目した式を *TSE(Transition Sequence Expression)* と呼ぶ．このため，TSE は，入力イベント，出力イベント，アクション，ガード条件，空列によって構成される．これらを**プリミティブ**と呼ぶことにする．プリミティブの集合 $Prim$ は，次の通りに定義する．

定義 1 (TSE のプリミティブ)

$$Prim \equiv ?E \cup !E \cup Act \cup Cond \cup \{\varepsilon\}$$

$$?E \equiv \{?e \mid e \in E\}$$
$$!E \equiv \{!e \mid e \in E\}$$
$$Act \equiv \{a \mid a \in A\}$$
$$Cond \equiv \{[c] \mid c \in C\}$$

ここで，E, A, C は，それぞれステートチャートの遷移ラベルに出現するイベント，アクション式，ガード条件の集合を表している．$?E, !E$ はイベント集合 E のそれぞれの要素に '?','!' を前置した集合を表している．

アクションとガード条件は，整数と真理値と属性，またそれらから構成される式を用いて記述する．これらの意味は，通常我々が用いる解釈によって与えられるものとする．まずアクションには，整数に関する加算と減算を用いた属性に対する代入のみ記述でき，これらの式を "[[" と"]]" で囲むことによってアクションを表す．文法は以下の通りである．

定義 2 (アクション)

$$\begin{aligned} action &:= [[\ attr\ :=\ iexpr\ ]] \\ iexpr &:= iexpr\ +\ iexpr \\ &\mid\ iexpr\ -\ iexpr \\ &\mid\ (\ iexpr\ ) \\ &\mid\ integer \\ &\mid\ attr \end{aligned}$$

ここで，$action$，$attr$，$integer$はそれぞれアクション，属性，整数を表す．$iexpr$ を整数と属性から構成される**式**と呼ぶ．アクションであることが明らかな場合には，"[[" と"]]" を省略できる．次にガード条件には，整数の比較と一階論理における真理演算 $\wedge, \vee, \neg$ と限量子 $\forall, \exists$ を記述でき，"[" と"]" で囲むことによって表す．文法は以下の通りである．

定義 3 (ガード 条件)

$$
\begin{array}{lcl}
condition & := & [\ cexpr\ ] \\
cexpr & := & cexpr\ \wedge\ cexpr \\
 & | & cexpr\ \vee\ cexpr \\
 & | & \neg\ cexpr \\
 & | & \forall\ var\ .\ cexpr \\
 & | & \exists\ var\ .\ cexpr \\
 & | & vexpr\ =\ vexpr \\
vexpr & := & vexpr\ +\ vexpr \\
 & | & vexpr\ -\ vexpr \\
 & | & (\ vexpr\ ) \\
 & | & integer \\
 & | & attr \\
 & | & var
\end{array}
$$

ここで，*condition*がガード 条件を表す．また，便宜上，以下の省略記法を定義する．

定義 4

$$
\begin{array}{lcl}
P \Rightarrow Q & \equiv & \neg P \vee Q \\
x \leq y & \equiv & \exists z\ .\ x + z = y \\
x < y & \equiv & (x \leq y) \wedge \neg(x = y) \\
x \geq y & \equiv & \neg(x < y) \\
x > y & \equiv & \neg(x \leq y)
\end{array}
$$

ステートチャートでは，遷移が連続して起こる (連接) 場合，複数の遷移のうち一つだけが発火する (選択) 場合があり，またこれらの組み合わせが繰返し起こる場合がある．このため，TSE は連接 ‘;’，選択 ‘+’，繰返し ‘*’ を表す 3 つの演算子を用いて，以下の文法に従って構成する．

定義 5 (TSE の文法)

$$
\begin{array}{rcl}
\mathrm{TSE} & = & \mathrm{TSE}\ ;\ \mathrm{TSE} \\
& | & \mathrm{TSE} + \mathrm{TSE} \\
& | & \mathrm{TSE}\ * \\
& | & \bot \\
& | & \mathit{primitive}
\end{array}
$$

ここで，*primitive* はプリミティブを表す．演算子の優先順位は '*', ';', '+' の順に高いものとし，必要に応じて括弧をつけるものとする．ここで提案した TSE では以下の性質が成り立つものとする．

公理 1 (TSE の公理)

$$
\begin{array}{l}
T;\varepsilon = \varepsilon;T = T \\
T;\bot = \bot;T = \bot \\
T + \bot = T \\
T_1 + T_2 = T_2 + T_1 \\
(T_1 + T_2) + T_3 = T_1 + (T_2 + T_3) \\
T_1;(T_2;T_3) = (T_1;T_2);T_3 \\
T;(T_1 + T_2) = T;T_1 + T;T_2 \\
(T_1 + T_2);T = T_1;T + T_2;T \\
T* = T;T* + \varepsilon
\end{array}
$$

ここで，T, T_1, T_2, T_3 は TSE である．

任意の TSE は，各遷移列の先頭にプリミティブが現れるように公理 1 を用いて変形することができる．このような TSE を ppTSE (primitive prefixed TSE) と呼ぶ．ppTSE の構成は，リストの構成に似ており，TSE に対する関数の定義に，非常に有効な場合わけと，定義すべき関数に対する性質の証明の指針となる．ppTSE の文法は以下の通りに定義する．

定義 6 (ppTSE) .

$$
\begin{aligned}
\text{ppTSE} \quad &= \quad \text{primitive ; TSE} \\
&\mid \quad \text{TSE} + \text{TSE} \\
&\mid \quad \varepsilon \\
&\mid \quad \bot
\end{aligned}
$$

ppTSEは，明らかにTSEである．また，公理1に従うと，以下の定理が成り立ち，TSEに対する関数定義と，ppTSEに対する関数定義が同じであることが分かる．

定理 1 (TSEと ppTSE) 任意のTSE T はppTSEに変形可能である．

証明 TSEの構造に従って帰納法を用いて証明する．

$T = T_1; T_2$ かつ T_1 がppTSEに変形可能であるとき

プリミティブ t_1 とTSE T_1' に対して $T_1 = t_1; T_1'$ と変形することができるか，または，$T_1 = T_1' + T_1''$ と変形できる．前者の場合，$T = t_1; T_1'; T_2$ となり，後者の場合，$T = (T_1' + T_2''); T_2 = T_1'; T_2 + T_1''; T_2$ となるため，T はppTSEに変形可能である．

$T = T_1 + T_2$ のとき

明らかに $T_1 + T_2$ はppTSEである．

$T = T_1*$ かつ T_1 がppTSEに変形可能であるとき

$T = T_1* = T_1; T_1 * + \varepsilon$ となるため，T はppTSEに変形可能である．

$T = \bot$ または $T = \varepsilon$ のとき

明らかに T はppTSEである．

T が ε 以外のプリミティブのとき

$T = T; \varepsilon$ とppTSEに変形できる．

以上より，任意のTSEはppTSEに変形可能である．

□

3.4 ステートチャートから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 [Mil99, Lee94] 等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同じである．

まずステートチャートの各々の遷移ラベルをTSEを用いて表す．次に，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を用いて表すものである．このような変数を**状態変数**と呼ぶことにする．すると，それぞれの状態間の関係を表す方程式を作ることができる．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，n 状態のステートチャートでは，状態 $S_1, S_2, \cdots, S_n$ に対する状態変数 $T_1, \cdots, T_n$ と，ステートチャートそのものを表す状態変数 T_0 を用いて次の連立方程式を作ることができる．

$$\begin{aligned}
T_0 &= A_{00};T_0 + A_{01};T_1 + \cdots + A_{0n};T_n \\
T_1 &= A_{10};T_0 + A_{11};T_1 + \cdots + A_{1n};T_n + B_1 \\
T_2 &= A_{20};T_0 + A_{21};T_1 + \cdots + A_{2n};T_n + B_2 \\
&\vdots \\
T_n &= A_{n0};T_0 + A_{n1};T_1 + \cdots + A_{nn};T_n + B_n
\end{aligned}$$

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，$\perp$ を用いる．B_i は，状態 S_i が終了状態のときに ε，そうでないときは $\perp$ とする．このような連立方程式を**状態方程式**と呼ぶ．このように構成された方程式は，一つずつ変数を消去するか，$T = A;T + B$ の解が $T = A*;B$ という事実を用いて解く．

定理 2 ステートチャート S_0 に対して，対応するTSEが必ず存在する．

証明 状態の個数に関する帰納法を用いる．

1個のとき

以下の方程式を得る．

$$\begin{aligned}
T_0 &= A_{00};T_0 + A_{01};T_1 \\
T_1 &= A_{10};T_0 + A_{11};T_1 + B_1
\end{aligned}$$

T_1 の式より，

$$T_1 = A_{11}*; (A_{10}; T_0 + B_1)$$

これを T_0 の式に代入すると

$$\begin{aligned} T_0 &= A_{00}; T_0 + A_{01}; A_{11}*; (A_{10}; T_0 + B_1) \\ &= A_{00}; T_0 + A_{01}; A_{11}*; A_{10}; T_0 + A_{01}; A_{11}*; B_1 \\ &= (A_{00} + A_{01}; A_{11}*; A_{10}); T_0 + A_{01}; A_{11}*; B_1 \end{aligned}$$

という式を得る．よって，

$$T_0 = (A_{00} + A_{01}; A_{11}*; A_{10})*; A_{01}; A_{11}*; B_1$$

という解を得ることができる．

n 個のとき

T_n を求めると，

$$T_n = A_{nn}*; (A_{n0}; T_0 + A_{n1}; T_1 + \cdots + A_{n(n-1)}; T_{n-1} + B_n)$$

となる．これを他の方程式に代入すると，任意の $T_i (i = 0, \cdots, n-1)$ に対して，

$$\begin{aligned} T_i &= A_{i0}; T_0 + A_{i1}; T_1 + \cdots + A_{i(n-1)}; T_{n-1} \\ &\quad + A_{in}; A_{nn}*; (A_{n0}; T_0 + A_{n1}; T_1 + \cdots + A_{n(n-1)}; T_{n-1} + B_n) \\ &= (A_{i0} + A_{in}; A_{nn}*; A_{n0}); T_0 \\ &\quad + (A_{i1} + A_{in}; A_{nn}*; A_{n1}); T_1 \\ &\quad + \cdots \\ &\quad + (A_{i(n-1)} + A_{in}; A_{nn}*; A_{n(n-1)}); T_{n-1} \\ &\quad + A_{in}; A_{nn}*; B_n \end{aligned}$$

となり，T_0 から T_{n-1} の連立方程式を得ることができる．帰納法の仮定より，この方程式の解は求まるので T_n の解は求まる．

以上より，任意のステートチャートに対する状態方程式は解を持つ．

□

例として，図 3.2 で示されたステートチャートに対する TSE を求める．まず以下の方程式を作る．

$$\begin{aligned} T_0 &= n := 0; T_1 \\ T_1 &= ?inc; n := n + 1; !a; S_2 \\ T_2 &= ?dec; n := n - 1; T_1 + T_3 \\ T_3 &= \varepsilon \end{aligned}$$

この方程式から次の通りに T_0 の解を得ることができる．まず，T_3 の式を T_2 の式に代入する．

$$T_2 = ?dec; n := n - 1; T_1 + \varepsilon$$

次に，この式を T_1 の式に代入する．

$$\begin{aligned} T_1 &= ?inc; n := n + 1; !a; (?dec; n := n - 1; T_1 + \varepsilon) \\ &= ?inc; n := n + 1; !a; ?dec; n := n - 1; T_1 + ?inc; n := n + 1; !a \\ T_1 &= (?inc; n := n + 1; !a; ?dec; n := n - 1)*; ?inc; n := n + 1; !a \end{aligned}$$

よって，T_0 は以下の通りである．

$$T_0 = n := 0; (?inc; n := n + 1; !a; ?dec; n := n - 1)*; ?inc; n := n + 1; !a$$

3.5 階層化されたステートチャートから TSE への変換

階層化されたステートチャートの場合，サブステートチャート内の状態からいつでも親ステートチャート内の状態へ遷移することができる．このため，サブステートチャート内のすべての状態はサブステートチャートを表す状態変数への遷移が必ず存在するものとして扱う．例えば，図 3.3 では以下の方程式を作る．

$$\begin{aligned} T_0 &= T_1 \\ T_1 &= e_1; T_2 \\ T_2 &= e_4; T_1 + T_3 + T_{21} \\ T_3 &= \varepsilon \\ T_{21} &= e_2; T_{22} + T_2 \\ T_{22} &= e_1; T_{21} + T_2 \end{aligned}$$

ここで，T_2 の式において T_21 を用いているのは，S_21 がサブステートチャートの初期状態であるためである．また，T_{21}, T_{22} のそれぞれの式には，T_2 がある．これは，サブステートチャートへの遷移があるものとして扱っているためである．この方程式を T_0 に対して解くと以下の式を得る．

$$T_0 = (e_1; ((e_2; e_1)*; (e_2 + \varepsilon))*; e_4)*; e_1; ((e_2; e_1)*; (e_2 + \varepsilon))*$$

第 4章

単一ステートチャートの検証

4.1 検証の概要

本章では，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．例えば，図 3.1 において，属性 n の値が常に非負であるという性質の検証などである．検証には主に，活性と安全性の検証があり，本研究で取り扱うこのような性質は安全性に含まれる．

無限の値をもち得る属性を取り扱うために演繹的な手法を用いる．そして，Hoare のプログラム検証で用いられる演繹体系に類似した体系を用いて，構文主導の検証を行う．このために，まず最初に，ステートチャートを TSE に変換する．この TSE は，各々の状態に到達するすべての遷移系列を表すものである．このため，すべての状態を終了状態とみなしてステートチャートを TSE に変換する．そして，TSE の前後に事前表明と事後表明を与えて検証を行う．

4.2 不変性検証のための TSE

ある性質が常に成り立つということを検証するためには，各々の状態に到達する可能な遷移系列を TSE を用いて表す．例えば，図 3.1 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．そのため，S_1, S_2 を終了状態とみなして，ステートチャートを TSE に変換する．まず，それぞれの状態に対する状態変数として T_1, T_2 を用いる．そして，T_0 はステートチャートを表す状態変数とする．次に，それぞれの状態変数の関係を用いて以下の連立方程式を作る．

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

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．

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

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

4.3 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，表明 P が成り立っているときに遷移 T が発火すれば，その後で表明 Q が成り立つということを表す．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．本論文では，アクションやガード条件に記述できる式

として，整数値と真理値を扱う式としたので，事前表明と事後表明には，ガード条件と同等の一階論理の式を記述できるものとする．

定義 7 (事前・事後表明)

$$
\begin{array}{rcl}
assertion & := & assertion \ \wedge \ assertion \\
& | & assertion \ \vee \ assertion \\
& | & \neg \, assertion \\
& | & \forall \, var \, . \, assertion \\
& | & vexpr \ = \ vexpr \\
vexpr & := & vexpr \ + \ vexpr \\
& | & vexpr \ - \ vexpr \\
& | & (\ vexpr \ ) \\
& | & integer \\
& | & attr \\
& | & var
\end{array}
$$

また，便宜上，以下の省略記法を定義する．

定義 8

$$
\begin{array}{lcl}
P \Rightarrow Q & \equiv & \neg P \vee Q \\
x \leq y & \equiv & \exists z \, . \, x + z = y \\
x < y & \equiv & (x \leq y) \wedge \neg (x = y) \\
x \geq y & \equiv & \neg (x < y) \\
x > y & \equiv & \neg (x \leq y)
\end{array}
$$

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 2 $\{P\}\varepsilon\{P\}$

公理 3 $\{P\}\bot\{Q\}$

公理 4 $\{P\}v := t\{P[t/v]\}$

公理 5 $\{P\}!e\{P\}$

公理 6 $\{P\}?e\{P\}$

公理 7 $\{P\}[C]\{P \wedge C\}$

εとイベント入力・出力の前後では，属性値は変化したいため，事前表明と事後表明は同じとなる．$\perp$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，次の 4 つを導入する．

推論規則 1

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

推論規則 2

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

推論規則 3

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

推論規則 4

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

4.4 例題への適用

論理 ATL を用いて，図 3.1 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 3.1 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，常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，$\{true\}T_0\{n \geq 0\}$

	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

表 4.1: $\{true\}T_0\{n \geq 0\}$ の導出過程

という A-TSE を論理 ATL を用いて導出することによって，常に $n \geq 0$ 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 4.1 に，前提となる A-TSE と使用した推論規則と共に，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．例えば，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．

4.5 健全性と相対完全性

論理体系 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，論理体系 ATL が相対完全であるとは，推論規則 conseq における $P' \Rightarrow P$ と $Q \Rightarrow Q'$ が論理体系 ATL 以外の体系，例えば，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．体系 ATL が健全かつ相対完全であることを証明するために，本節では，まず TSE に対する意味付けを行い，その次に A-TSE に対する意味づけを行う．

4.5.1 TSE の意味

TSE に対する意味づけを行うために，TSE の前後における属性環境の変化を表す述語 Tr を用意する．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 (TSE の意味) $Tr(T, \sigma, \sigma')$ は，TSE T で示される遷移の前後で属性環境が σ から σ' に変化することを表し，以下の通りに定義する．

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. $\forall \sigma \,.\, Tr([C], \sigma, \sigma) \Rightarrow t$
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. $\forall \sigma, \sigma' \,.\, Tr(T*, \sigma, \sigma') \Leftrightarrow Tr(T; T* + \varepsilon, \sigma, \sigma')$

特に，本研究が非決定的なステートチャートも検証の対象に含めているために 8 の定義を行っている．そして，TSE を実行した場合に同じ属性環境の変化を得ることを**振舞い等価性**と呼び，次の通りに定義する．

定義 10 (振舞い等価性)

$$T \simeq T' \equiv \forall \sigma, \sigma' \,.\, Tr(T, \sigma, \sigma') \Leftrightarrow Tr(T', \sigma, \sigma')$$

以上で，TSE が属性環境に及ぼす影響を特徴付けることができた．

4.5.2 A-TSE の操作的意味論

正しい A-TSE を表すために，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 に与える．これは，論理式には自由変数と属性環境が含まれ，これらの状態に応じて論理式の真偽が決定するためである．

定義 11 σ を属性環境，I を自由変数に対する解釈とする．属性環境 σ，解釈 I に対して論理式 A が成り立つことを次の通りに書く．

$$I, \sigma \models A$$

特に，任意の I に対して $I, \sigma \models P$ が成り立つとき，次の通りに書く．

$$\sigma \models A$$

また，さらに任意の σ に対して $\sigma \models P$ が成り立つとき，次の通りに書く．

$$\models A$$

$\models A$ は，$\models$ を省いて A と書くこともできる．

本研究において与える操作的意味論では，$\{P\}T\{Q\}$ と書いて表明 P が成り立っているときに遷移 T が発火すれば，その後で表明 Q が成り立つということを表す．このため，A-TSE の意味は次の通りに定義する．

定義 12

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

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

この定義により，本研究で行う検証では，**到達するようなすべての状態**で属性値に関するある性質が成り立つことを調べることができる．つまり，遷移 T が発火しない場合には $\models \{P\}T\{Q\}$ は真となる．このことは定義 9 の非決定性に密接に関係している．遷移 T_1 と T_2 を考える．定義 9 に従うと，

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

である．また，任意の論理式 A, B, C, D に対して

$$A \Rightarrow ((B \vee C) \Rightarrow D) \Leftrightarrow ((A \Rightarrow (B \Rightarrow D)) \wedge (A \Rightarrow (C \Rightarrow D)))$$

である．ここで，以下の通りにそれぞれの A, B, C, D を置き換える．

$$\begin{aligned} A &= \sigma \models P \\ B &= Tr(T_1, \sigma, \sigma') \\ C &= Tr(T_2, \sigma, \sigma') \\ D &= \sigma' \models Q \end{aligned}$$

すると次の式を得る.

$$\models \{P\}T_1 + T_2\{Q\} \Leftrightarrow \models \{P\}T_1\{Q\} \wedge \models \{P\}T_2\{Q\}$$

このため，$\models \{P\}T_1\{Q\}$ と $\models \{P\}T_2\{Q\}$ の両方が成り立つ必要があり，遷移が非決定的であっても検証を行うことができる．そして，発火しない遷移 T については $\models \{P\}T\{Q\}$ が成り立つため，到達するようなすべての状態について検証を行っていることになる.

4.5.3 健全性・相対完全性の証明

論理 ATL の健全性と相対完全性は次の通りに定義し，論理 ATL は健全かつ相対完全であることが証明できる.

定義 13 (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 14 (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．

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

定理 3 論理 ATL は健全である．

証明 証明には，公理と推論規則に対する帰納法を用いる．まず，論理 ATL の公理について，$\models \{P\}T\{Q\}$ であることを証明する．

公理 2 :

$Tr(\varepsilon, \sigma, \sigma)$
$\Rightarrow (\sigma \models P \Rightarrow (Tr(\varepsilon, \sigma, \sigma) \Rightarrow \sigma \models P))$
$\Rightarrow \models \{P\}\varepsilon\{P\}$

公理 3 :

$\neg Tr(\bot, \sigma, \sigma')$
$\Rightarrow (\sigma \models P \Rightarrow (Tr(\bot, \sigma, \sigma') \Rightarrow \sigma' \models P))$
$\Rightarrow \models \{P\}\bot\{Q\}$

公理 4 :

$Tr(v := t, \sigma, \sigma[t/v])$
$\Rightarrow \sigma \models P[t/v] \Rightarrow (Tr(v := t, \sigma, \sigma[t/v]) \Rightarrow \sigma[t/v] \models P)$
$\Rightarrow \models \{P[t/v]\}v := t\{P\}$

公理 5 :

$Tr(!e, \sigma, \sigma)$
$\Rightarrow (\sigma \models P \Rightarrow (Tr(!e, \sigma, \sigma') \Rightarrow \sigma \models P))$
$\Rightarrow \models \{P\}!e\{P\}$

公理 6 :

$Tr(?e, \sigma, \sigma)$
$\Rightarrow (\sigma \models P \Rightarrow (Tr(?e, \sigma, \sigma') \Rightarrow \sigma \models P))$
$\Rightarrow \models \{P\}?e\{P\}$

公理 7 :

$Tr([C], \sigma, \sigma) \Rightarrow \sigma \models C$
$\Rightarrow (\sigma \models P \Rightarrow ((Tr([C], \sigma, \sigma) \Rightarrow \sigma \models C) \Rightarrow \sigma \models P \wedge \sigma \models C))$
$\Rightarrow (\sigma \models P \Rightarrow ((Tr([C], \sigma, \sigma) \Rightarrow \sigma \models C) \Rightarrow \sigma \models (P \wedge C)))$
$\Rightarrow \models \{P\}[C]\{P \wedge C\}$

次に，各推論規則の上から下への証明が可能であることを示す. つまり，上の ATL を仮定として，下の ATL を導く.

推論規則 1 :

$\models \{P\}T\{Q\} \wedge (P' \Rightarrow P) \wedge (Q \Rightarrow Q')$
$\Rightarrow (\sigma \models P \Rightarrow (Tr(T, \sigma, \sigma') \Rightarrow \sigma' \models Q)) \wedge (P' \Rightarrow P) \wedge (Q \Rightarrow Q')$
$\Rightarrow \sigma \models P' \Rightarrow (Tr(T, \sigma, \sigma') \Rightarrow \sigma \models Q')$
$\Rightarrow \models \{P'\}T\{Q'\}$

推論規則 2 :

$\models \{P\}T_1\{Q\} \wedge \models \{Q\}T_2\{R\}$
$\Rightarrow \exists \sigma' \,.\, (\sigma \models P \Rightarrow (Tr(T_1, \sigma, \sigma') \Rightarrow \sigma' \models Q))$
$\qquad \wedge (\sigma' \models Q \Rightarrow (Tr(T_2, \sigma', \sigma'') \Rightarrow \sigma'' \models R))$
$\Rightarrow \exists \sigma' \,.\, \sigma \models P \Rightarrow (Tr(T_1, \sigma, \sigma') \Rightarrow (Tr(T_2, \sigma', \sigma'') \Rightarrow \sigma'' \models R))$
$\Rightarrow \exists \sigma' \,.\, \sigma \models P \Rightarrow ((Tr(T_1, \sigma, \sigma') \wedge Tr(T_2, \sigma', \sigma'')) \Rightarrow \sigma'' \models R)$
$\Rightarrow \sigma \models P \Rightarrow (Tr(T_1; T_2, \sigma, \sigma'') \Rightarrow \sigma'' \models R)$
$\Rightarrow \models \{P\}T_1; T_2\{R\}$

推論規則 3 :

$$
\begin{aligned}
&\models \{P\}T_1\{Q\} \wedge \models \{P\}T_2\{Q\} \\
&\Rightarrow (\sigma \models P \Rightarrow (Tr(T_1, \sigma, \sigma') \Rightarrow \sigma' \models Q)) \\
&\quad \wedge (\sigma \models P \Rightarrow (Tr(T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q)) \\
&\Rightarrow \sigma \models P \Rightarrow ((Tr(T_1, \sigma, \sigma') \Rightarrow \sigma' \models Q) \wedge (Tr(T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q)) \\
&\Rightarrow \sigma \models P \Rightarrow ((Tr(T_1, \sigma, \sigma') \vee Tr(T_2, \sigma, \sigma')) \Rightarrow \sigma' \models Q) \\
&\Rightarrow \sigma \models P \Rightarrow (Tr(T_1 + T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q) \\
&\Rightarrow \models \{P\}T_1 + T_2\{Q\}
\end{aligned}
$$

推論規則 4 :

$$
\begin{aligned}
&\neg \models \{P\}T*\{P\} \\
&\Rightarrow \neg(\sigma \models P \Rightarrow (Tr(T*, \sigma, \sigma') \Rightarrow \sigma' \models P)) \\
&\Rightarrow \neg(\sigma \models P \Rightarrow (Tr(T; T*+\varepsilon, \sigma, \sigma') \Rightarrow \sigma' \models P)) \\
&\Rightarrow \neg(\sigma \models P \Rightarrow ((Tr(T; T*, \sigma, \sigma') \vee Tr(\varepsilon, \sigma, \sigma')) \Rightarrow \sigma' \models P)) \\
&\Rightarrow \neg(Tr(T; T*, \sigma, \sigma') \vee Tr(\varepsilon, \sigma, \sigma') \Rightarrow \sigma' \models P) \Rightarrow \neg\sigma \models P \\
&\Rightarrow (\neg\sigma' \models P \Rightarrow \neg(Tr(T; T*, \sigma, \sigma') \vee Tr(\varepsilon, \sigma, \sigma'))) \Rightarrow \neg\sigma \models P \\
&\Rightarrow (\neg\sigma' \models P \Rightarrow (\neg Tr(T; T*, \sigma, \sigma') \wedge \neg Tr(\varepsilon, \sigma, \sigma'))) \Rightarrow \neg\sigma \models P \\
&\Rightarrow ((\neg\sigma' \models P \Rightarrow \neg Tr(T; T*, \sigma, \sigma')) \Rightarrow \neg\sigma \models P) \\
&\quad \vee ((\neg\sigma' \models P \Rightarrow \neg Tr(\varepsilon, \sigma, \sigma')) \Rightarrow \neg\sigma \models P) \\
&\Rightarrow \neg(\sigma \models P \Rightarrow (Tr(T; T*, \sigma, \sigma') \Rightarrow \sigma' \models P)) \\
&\quad \wedge \neg(\sigma \models P \Rightarrow (Tr(\varepsilon, \sigma, \sigma') \Rightarrow \sigma' \models P)) \\
&\Rightarrow \neg \models \{P\}T; T*\{P\} \wedge \neg \models \{P\}\varepsilon\{P\} \\
&\Rightarrow \neg \models \{P\}T; T*\{P\} \\
&\Rightarrow \neg \models \{P\}T\{P\} \wedge \neg \models \{P\}T*\{P\} \\
&\Rightarrow \neg \models \{P\}T\{P\} \\
&\therefore \models \{P\}T\{P\} \Rightarrow \models \{P\}T*\{P\}
\end{aligned}
$$

以上より，$\vdash \{P\}T\{Q\} \Rightarrow \models \{P\}T\{Q\}$ であるため，論理 ATL は健全である．

□

完全性を証明するためには，次に定義する最弱前条件が存在しなくてはならない．

定義 15 (最弱前条件) TSE T と事後表明 Q に対して，以下の通りに wp を定義する．

$$wp(T, Q) \equiv P$$

ここで，P は以下を満たすものとする．

$$\forall \sigma \in \Sigma(T, Q) \,.\, \sigma \models P$$
$$\Sigma(T, Q) = \{\sigma | \forall \sigma' \,.\, Tr(T, \sigma, \sigma') \Rightarrow \sigma' \models Q\}$$

このような $wp(T, Q)$ を TSE T と事後表明 Q に対する**最弱前条件**と呼ぶ．非形式的には，$\Sigma(T, Q)$ は，TSE T の後で事後表明 Q が成り立つような遷移前の属性環境の集合を表している．$wp(T, Q)$ は，TSE T の後で事後表明 Q が成り立つような最も弱い事前表明を表している．つまり，$\Sigma(T, Q)$ の任意の属性環境に対して成り立つような事前表明が最弱前条件である．次に，本論文で用いる表明においては，任意の表明と TSE に対して常に最弱前条件が存在することを証明する．

定理 4 任意の表明 Q と TSE T に対して $wp(T, Q)$ が存在する．

証明 TSE の構造に関する帰納法を用いて証明する．まず「任意の表明 Q とプリミティブ t に対して $wp(t, Q)$ が存在する (以下の通りに構成できる)」ことを証明する．

1. $wp(\varepsilon, Q) = Q$
   $\sigma' \models Q$ となる σ' を考える．
   Tr の定義より $Tr(\varepsilon, \sigma', \sigma')$ となる．このため，$\Sigma(\varepsilon, Q) = \{\sigma'\}$ である．
   $\sigma' \models Q$ であったので，$Q = wp(\varepsilon, Q)$ である．

2. $wp(\perp, Q) = true$
   $\sigma' \models Q$ となる σ' を考える．

$\forall\sigma\ .\ \neg Tr(\bot,\sigma,\sigma')$ となるため，$\forall\sigma \in \Sigma(T,Q)$ となる．任意の σ に対して $\sigma \models P$ が成り立つためには $P = true$ でなければならない．

3. $wp(v := t, Q) = \forall v'\ .\ ((v' = t) \Rightarrow Q[v'/v])$
   $\sigma' \models Q$ となる σ' を考える．
   $Tr(v := t, \sigma, \sigma[t/v])$ となるため，$\sigma' = \sigma[t/v]$ である．このため，

$$\begin{aligned}\Sigma(T,Q) &= \{\sigma|\forall\sigma'\ .\ \sigma' = \sigma[t/v] \Rightarrow \sigma' \models Q\}\\ &= \{\sigma|\forall v\ .\ (\sigma'(v) = \sigma[t/v](v)) \Rightarrow Q[\sigma'(v)/v]\}\\ &= \{\sigma|\forall v'\ .\ ((v' = t) \Rightarrow Q[v'/v])\}\end{aligned}$$

   となる．ここで，$v' = \sigma'(v)$ である．よって，$P = \forall v'\ .\ ((v' = t) \Rightarrow Q[v'/v])$ である．

4. $wp(!e, Q) = Q$
   ε の場合と同様.

5. $wp(?e, Q) = Q$
   ε の場合と同様.

6. $wp([C], Q) = Q$
   ε の場合と同様.

次に，連接，選択，繰返しについて帰納法を用いる．帰納法の仮定は，TSE T, T_1, T_2 に対して最弱前条件が存在するということである．

1. $wp(T_1; T_2, Q) = wp(T_1, wp(T_2, Q))$
   帰納法の仮定より，$R = wp(T_2, Q)$ となる Rが存在し，さらに $P = wp(T_1, R)$ となる Pが存在する．このとき，

$$\begin{aligned}\Sigma(T_1;T_2,Q) &= \{\sigma|\forall\sigma'\ .\ Tr(T_1;T_2,\sigma,\sigma') \Rightarrow \sigma' \models Q\}\\ &= \{\sigma|\forall\sigma'\sigma''\ .\ \sigma'' \models R \wedge Tr(T_1,\sigma,\sigma'') \wedge Tr(T_2,\sigma'',\sigma') \Rightarrow \sigma' \models Q\}\\ &= \{\sigma|\forall\sigma'\sigma''\ .\ (Tr(T_1,\sigma,\sigma'') \Rightarrow \sigma'' \models R)\\ &\quad \wedge(\sigma'' \models R \Rightarrow (Tr(T_2,\sigma'',\sigma') \Rightarrow \sigma' \models Q))\}\\ &= \{\sigma|\forall\sigma''\ .\ (Tr(T_1,\sigma,\sigma'') \Rightarrow \sigma'' \models R)\}\\ &= \Sigma(T_1,R)\end{aligned}$$

となるため，$wp(T_1;T_2,Q) = wp(T_1,wp(T_2,Q))$ である．

2. $wp(T_1 + T_2, Q) = wp(T_1, Q) \wedge wp(T_2, Q)$
   帰納法の仮定より，$P_1 = wp(T_1, Q), P_2 = wp(T_2, Q)$ となる P_1, P_2 が存在する．このとき，

$$\begin{aligned}\Sigma(T_1 + T_2, Q) &= \{\sigma | \forall \sigma' \,.\, Tr(T_1 + T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q\} \\ &= \{\sigma | \forall \sigma' \,.\, Tr(T_1, \sigma, \sigma') \vee Tr(T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q\} \\ &= \{\sigma | \forall \sigma' \,.\, (Tr(T_1, \sigma, \sigma') \Rightarrow \sigma' \models Q) \\ &\quad \wedge (Tr(T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q\})\end{aligned}$$

   となるため，$wp(T_1 + T_2, Q) = wp(T_1, Q) \wedge wp(T_2, Q)$ である．

3. $wp(T*, Q) = Q$

$$\begin{aligned}wp(T*, Q) &= wp(T; T* + \varepsilon, Q) \\ &= wp(T; T*, Q) \wedge wp(\varepsilon, Q) \\ &= wp(T, wp(T*, Q)) \wedge Q \\ &= Q\end{aligned}$$

   となり $wp(\varepsilon, Q) = Q$ のため，$wp(T_1;T_2, Q) = wp(T_1, Q) \wedge wp(T_2, Q)$ である．

□

任意の TSE に最弱前条件が存在すると wp に関して，次の定理が成り立つ．

定理 5

$$wp(T_1, wp(T_2, Q)) = wp(T_1;T_2, Q)$$

ここで，T_1, T_2 は TSE であり，Q は表明である．

証明 定理 4 の帰納段階の 2 の証明より明らか．

□

この定理は，証明に必要な中間表明が必ず存在することを述べている．中間表明が必ず存在することにより，論理 ATL は相対完全であることが証明できる．

定理 6 論理 ATL は相対完全である.

証明 TSE の構造に関する帰納法を用いて証明する.

プリミティブ :

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

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

$\models \{P\}v := t\{Q\} \Rightarrow \vdash \{P\}v := t\{Q\}$

$\models \{P\}!e\{Q\} \Rightarrow \vdash \{P\}!e\{Q\}$

$\models \{P\}?e\{Q\} \Rightarrow \vdash \{P\}?e\{Q\}$

$\models \{P\}[C]\{Q\} \Rightarrow \vdash \{P\}[C]\{Q\}$

帰結規則 :

$\models \{P\}T\{Q\} \wedge P' \Rightarrow P \wedge Q \Rightarrow Q'$

$\Rightarrow (\sigma \models P \Rightarrow (Tr(T, \sigma, \sigma') \Rightarrow \sigma' \models Q)) \wedge (P' \Rightarrow P \wedge Q \Rightarrow Q')$

$\Rightarrow \sigma \models P' \Rightarrow (Tr(T, \sigma, \sigma') \Rightarrow \sigma' \models Q')$

$\Rightarrow \models \{P'\}T\{Q'\}$

連接演算子 :

$\models \{P\}T_1; T_2\{Q\}$

$\Rightarrow \models P \Rightarrow wp(T_1; T_2, Q) \wedge \{wp(T_1; T_2, Q)\}T_1; T_2\{Q\}$

$\Rightarrow \models P \Rightarrow wp(T_1; T_2, Q) \wedge \{wp(T_1, wp(T_2, Q))\}T_1; T_2\{Q\}$

$\Rightarrow \models P \Rightarrow wp(T_1, wp(T_2, Q))$

$\wedge \{wp(T_1, wp(T_2, Q))\}T_1\{wp(T_2, Q)\}\wedge \models \{wp(T_2, Q)\}T_2\{Q\}$

$\Rightarrow \models P \Rightarrow wp(T_1, wp(T_2, Q))$

$\wedge \vdash \{wp(T_1, wp(T_2, Q))\}T_1\{wp(T_2, Q)\}\wedge \vdash \{wp(T_2, Q)\}T_2\{Q\}$

$\Rightarrow \vdash P \Rightarrow wp(T_1, wp(T_2, Q)) \wedge \{wp(T_1, wp(T_2, Q))\}T_1; T_2\{Q\}$

$\Rightarrow \vdash \{P\}T_1; T_2\{Q\}$

選択演算子:

$\models \{P\}T_1 + T_2\{Q\}$

$\Rightarrow (\sigma \models P \Rightarrow Tr(T_1 + T_2, \sigma, \sigma') \Rightarrow \sigma' \models Q)$

$\Rightarrow (\sigma \models P \Rightarrow (Tr(T_1, \sigma, \sigma') \vee Tr(T_2, \sigma, \sigma')) \Rightarrow \sigma' \models Q)$

$\Rightarrow (\sigma \models P \Rightarrow (Tr(T_1, \sigma, \sigma') \Rightarrow \sigma' \models Q))$

$\quad \wedge (\sigma \models P \Rightarrow (Tr(T_1, \sigma, \sigma') \Rightarrow \sigma' \models Q))$

$\Rightarrow \models \{P\}T_1\{Q\} \wedge \models \{P\}T_2\{Q\}$

$\Rightarrow \vdash \{P\}T_1\{Q\} \wedge \vdash \{P\}T_2\{Q\}$

$\Rightarrow \vdash \{P\}T_1 + T_2\{Q\}$

繰返し演算子:

$\models \{P\}T * \{Q\}$

$\Rightarrow (\sigma \models P \Rightarrow (Tr(T*, \sigma, \sigma') \Rightarrow \sigma' \models Q))$

$\Rightarrow (\sigma \models P \Rightarrow (Tr(T; T * +\varepsilon, \sigma, \sigma') \Rightarrow \sigma' \models Q))$

$\Rightarrow (\sigma \models P \Rightarrow (Tr(T; T*, \sigma, \sigma') \Rightarrow \sigma' \models Q))$

$\quad \vee (\sigma \models P \Rightarrow (Tr(\varepsilon, \sigma, \sigma') \Rightarrow \sigma' \models Q))$

$\Rightarrow (\sigma \models P \Rightarrow ((Tr(T; T*, \sigma, \sigma') \vee Tr(\varepsilon, \sigma, \sigma')) \Rightarrow \sigma' \models Q))$

$\Rightarrow \models \{P\}T; T * \{Q\} \wedge \models \{P\}\varepsilon\{Q\}$

$\Rightarrow \models \{P\}T; T * \{Q\} \wedge P = Q$

$\Rightarrow \models \{P\}T\{P\} \wedge \{P\}T * \{Q\} \wedge P = Q$

$\Rightarrow \models \{P\}T\{Q\} \wedge P = Q$

$\Rightarrow \models \{P\}T\{P\} \wedge P = Q$

$\Rightarrow \vdash \{P\}T\{P\} \wedge P = Q$

$\Rightarrow \vdash \{P\}T * \{P\} \wedge P = Q$

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

以上より，$\models \{P\}T\{Q\} \Rightarrow \vdash \{P\}T\{Q\}$ であるため，論理 ATL は相対完全である．

□

第 5章

協調動作を行うステートチャートの検証

5.1 検証の概要

図 5.1 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，本節では，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，4 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 5.2 は近似手法を図解したものである．S_1,S_2,S_3 はそれぞれステートチャートを表す．S_1 を S_3 の振舞いを用いて近似し，S_3 を S_2 の振舞いを用いて近似している．さらに，近似した S_3 を用いて S_2 を近似する．このように，あるステー

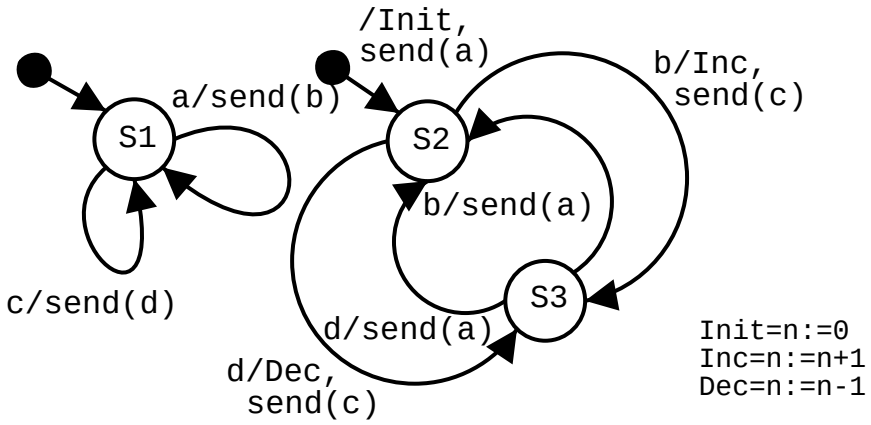

図 5.1: 依存するステートチャート

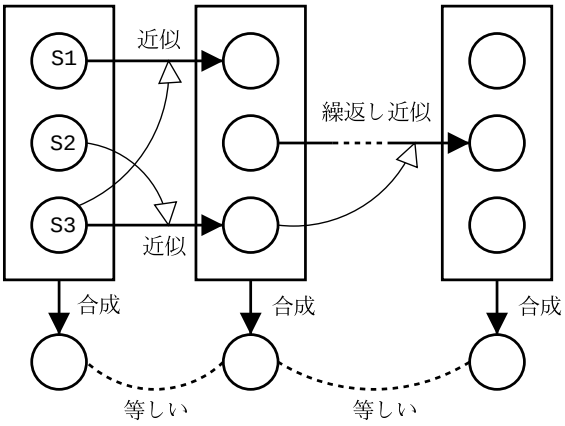

図 5.2: 合成と近似

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ことを，検証ができるようになるまで繰返し行う．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くなる．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 5.1 の例に適用する．

5.2 合成と除去関数

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，すべての遷移の組み合わせを求めてから，与えれたイベント通信方法では不可能な遷移の組み合わ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，遷移の組み合わせを求める**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み合わせを取り除き，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 16 (インターリーブ演算子) t_1, t_2 を $\varepsilon, \perp$ 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 に対して次の通りに定義する．

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

ここで，定義 6 の ppTSE に対する定義を行っている．このため，定理 6 とこの定義より，任意の 2 つの TSE のインターリーブを計算することができる．このようなインターリーブの定義は，[VM92] で述べられているインターリーブや [HMU00] で述べられているシャッフルと同じである．この定義の直観的な意味は，二つのステートチャートの直積をとることである．例えば，図 5.3 は，2 つのステートチャートを表す．この 2 つのステートチャートを表現する TSE に対してインターリーブを行うと，図 5.4 のステートチャートを表す TSE を得ることができる． まず，図 5.3 の 2 つのステートチャートを検証する際，それぞれ

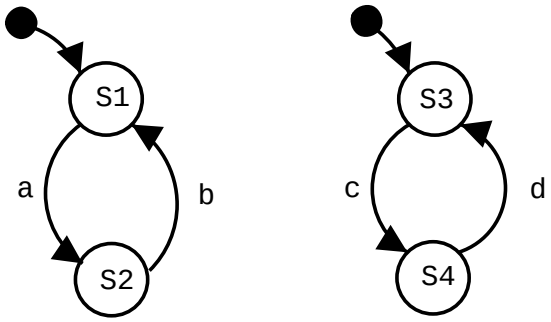

図 5.3: インターリーブ前のステートチャート

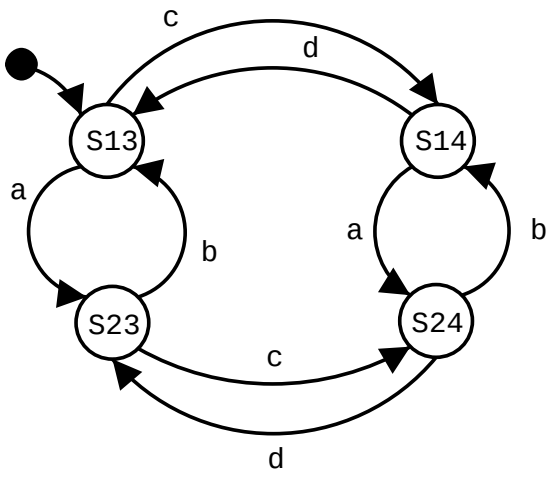

図 5.4: インターリーブ後のステートチャート

次の TSE を求める．L, R は，それぞれ左側，右側のステートチャートのための TSE である．

$$\begin{aligned} L &= (a;b)*;(a+\varepsilon) \\ R &= (c;d)*;(c+\varepsilon) \end{aligned}$$

これらに対してインターリーブを行うと次の通りである．

$$\begin{aligned} & L \times R \\ &= (a;b)*;(a+\varepsilon) \times (c;d)*;(c+\varepsilon) \\ &= ((a;b)*+(a;b)*;a) \times ((c;d)*+(c;d)*;c) \\ &= (a;b)*\times(c;d)*+(a;b)*\times(c;d)*;c+(a;b)*;a\times(c;d)*+(a;b)*;a\times(c;d)*;c \end{aligned}$$

ここで，$(a;b)*\times(c;d)*$ の式についてまず計算を行う．

$$\begin{aligned} & (a;b)*\times(c;d)* \\ &= (a;b;(a;b)*+\varepsilon) \times (c;d;(c;d)*+\varepsilon) \\ &= a;b;(a;b)*\times c;d;(c;d)*+a;b;(a;b)*\times\varepsilon+(\varepsilon\times c;d;(c;d)*)+(\varepsilon\times\varepsilon) \\ &= ((a;(b;(a;b)*\times c;d;(c;d)*))+(c;(a;b;(a;b)*\times c;d;(c;d)*))) \\ &\quad +a;b;(a;b)*+c;d;(c;d)*+\varepsilon \\ &= \cdots \\ &= (a;b;c;d+a;c;d;b+c;d;a;b+c;a;b;d+c;a;d;b);((a;b)*\times(c;d)*) \end{aligned}$$

$T = (a;b)*\times(c;d)*$ とすると，

$$T = (a;b;c;d+a;c;d;b+c;d;a;b+c;a;b;d+c;a;d;b);T$$

となり，この方程式を解くことによって次の通りに T を得る．

$$T = (a;b;c;d+a;c;d;b+c;d;a;b+c;a;b;d+c;a;d;b)*$$

$L \times R$ の残りの計算も同様にして行うことができ，以下の結果を得る．

$$\begin{aligned} & L \times R = \\ & (a;b;c;d+a;c;d;b+c;d;a;b+c;a;b;d+c;a;d;b)*; \\ & (a;b;c+a;c;d+c;d;a+c;a;b+c;a;d+a;b+a;c+c;d+c;a+a+c+\varepsilon) \end{aligned}$$

このように，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．このことは，有限オートマトンの直積を構成するが可能であることと，正規表現と有限オートマトンが等価であることから導くことができる [HMU00].

インターリーブ演算子は，以下の通りの交換則と結合則を満たす.

$$\begin{aligned} T_1 \times T_2 &= T_2 \times T_1 && \text{交換則} \\ (T_1 \times T_2) \times T_3 &= T_1 \times (T_2 \times T_3) && \text{結合則} \end{aligned}$$

$\prod_{T \in TS} T$ 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，以下の通りに定義する.

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

合成関数 C の定義は，次の通りである.

定義 17 (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同士の通信で用いられるイベントの集合とする．このとき，次の通りに定義される 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 18 (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. $\rho(\bot, E) = \bot$
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．

このように定義した除去関数 ρ に対して，次の定理が成り立つ．

定理 7

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

ここで，T_1, T_2 は TSE である．

証明

基底段階:

$\rho(\varepsilon \times T_2, E) = \rho(T_2, E)$

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

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

帰納段階:

$\rho(t_1; T_1 \times t_2; T_2, E) = \rho(t_1; T_1 \times \rho(t_2; T_2, E_{t_1,T_1}), E)$

仮定:

$\rho(T_1 \times t_2; T_2, E) = \rho(T_1 \times \rho(t_2, T_2, E - E_e(T_1)), E)$

$\rho(t_1; T_1 \times T_2, E) = \rho(t_1; T_1 \times \rho(T_2, E - E_e(t_1; T_1)), E)$

ここで，$T_1' \in t_1; T_1, T_2' \in t_2; T_2$ とし，簡単のため，$E - E_e(t)$ を E_t と書くことにする．

$\rho(t_1; T_1 \times t_2; T_2, E)$

$= \rho(t_1; (T_1 \times t_2; T_2) + t_2; (t_1; T_1 \times T_2), E)$

$= \rho(t_1; (T_1 \times t_2; T_2), E) + \rho(t_2; (t_1; T_1 \times T_2), E)$

$= \rho(t_1; \rho(T_1 \times t_2; T_2, E_{t_1}), E) + \rho(t_2; \rho(t_1; T_1 \times T_2, E_{t_2}), E)$

$= \rho(t_1; \rho(T_1 \times \rho(t_2; T_2, E_{t_1,T_1}), E_{t_1}), E) + \rho(t_2; \rho(t_1; T_1 \times \rho(T_2, E_{t_2,t_1;T_1}), E_{t_2}), E)$

$= \rho(t_1; \rho(T_1 \times \rho(t_2; \rho(T_2, E_{t_1,T_1,t_2}), E_{t_1,T_1}), E_{t_1}), E) + \rho(t_2; \rho(t_1; T_1 \times \rho(T_2, E_{t_2,t_1;T_1}), E_{t_2}), E)$

$= \rho(t_1; (T_1 \times t_2; \rho(T_2, E_{t_1,T_1,t_2})), E) + \rho(t_2; (t_1; T_1 \times \rho(T_2, E_{t_2,t_1;T_1})), E)$

$\rho(t_1; T_1 \times \rho(t_2; T_2, E_{t_1,T_1}), E)$

$= \rho(t_1; T_1 \times \rho(t_2; \rho(T_2, E_{t_1,T_1,t_2}), E_{t_1,T_1}), E)$

$= \rho(t_1; T_1 \times t_2; \rho(T_2, E_{t_1,T_1,t_2}), E)$

$= \rho(t_1; (T_1 \times t_2; \rho(T_2, E_{t_1,T_1,t_2})), E) + \rho(t_2; (t_1; T_1 \times \rho(T_2, E_{t_2,t_1;T_1})), E)$

$\therefore \rho(t_1; T_1 \times t_2; T_2, E) = \rho(t_1; T_1 \times \rho(t_2; T_2, E_{t_1,T_1}), E)$

□

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 [VM92] 等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．

5.3 イベント通信

TSE の合成は，モデルにおいて採用したイベント通信方法に依存することを前節で述べた．本節では，代表的な**同期イベント通信**と**非同期イベント通信**の 2 つのイベント通信方法を紹介する．

5.3.1 同期イベント通信

同期イベント通信は，イベントの到達に一切の遅延が許されないイベント通信方法である．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．このような通信方法は，CSP[Hoa85]，CRE[VM92] などをはじめ，オートマトンを用いた形式的検証では頻繁に用いられるものである．イベント通信方法は，合成で用いる除去関数を定義することによって定義する．

定義 19 (同期イベント通信)

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

ここで，T は TSE を表し，t は $\varepsilon, \perp$ 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．次に，このように定義した ρ_s は，定義 18 の除去関数としての制約を満たすことを証明する．

証明

1 の証明:

ρ_s の定義に従って，帰納法を用いて証明する．

$\underline{T = T_1 + T_2 \text{ のとき:}}$

$$\rho_s(T, \{\}) = \rho_s(T_1 + T_2, \{\}) = \rho_s(T_1, \{\}) + \rho_s(T_2, \{\}) = \rho_s(T_1, \{\}) + T_2 = T_1 + T_2 = T$$

$\underline{T = !e; ?e; T' \wedge e \in E \text{ のとき}}$

$E = \{\}$ よりあり得ない．

$\underline{T = t; T' \wedge t \notin ?E \wedge t \notin !E \text{ のとき}}$

$$\rho_s(T, E) = \rho_s(t; T', \{\}) = t; \rho_s(T', \{\}) = t; T' = T$$

$\underline{T = \varepsilon \text{ のとき}}$

$$\rho_s(T, \{\}) = \rho_s(\varepsilon, \{\}) = \varepsilon = T$$

$\underline{T = \bot \text{ のとき}}$

$$\rho_s(T, E) = \rho_s(\bot, \{\}) = \bot = T$$

よって，任意の TSE T に対して，

$$\rho_s(T, \{\}) = T$$

2 の証明:

ρ_s の定義に従って，帰納法を用いて証明する．

$\underline{T = T_1 + T_2 \text{ のとき}}$

$\rho_s(T_1) \subseteq T_1, \rho_s(T_2) \subseteq T_2$ を仮定すると，

$$\rho_s(T_1 + T_2, E) = \rho_s(T_1, E) + \rho_s(T_2, E) \subseteq \rho_s(T_1, E) + T_2 \subseteq T_1 + T_2 = T$$

$\underline{T = !e; ?e; T' \wedge e \in E \text{ のとき}}$

$$\rho_s(T, E) = !e; ?e; \rho_s(T', E) \subseteq !e; ?e; T' = T$$

$T = t; T' \wedge t \notin ?E \wedge t \notin !E$ のとき

$$\rho_s(T, E) = t; \rho_s(T', E) \subseteq t; T' = T$$

$T = \varepsilon$ のとき

$$\rho_s(\varepsilon, E) = \varepsilon \subseteq \varepsilon = T$$

$T = \perp$ のとき

$$\rho_s(\perp, E) = \perp \subseteq \perp = T$$

以上より，任意の TSE T に対して，

$$\rho_s(T, E) \subseteq T$$

3 の証明:

定義より，$\rho_s(\perp, E) = \perp$.

4 の証明:

定義より，$\rho_s(\varepsilon, E) = \varepsilon$.

5 の証明:

定義より，$\rho_s(T_1 + T_2, E) = \rho_s(T_1, E) + \rho_s(T_2, E)$.

6 の証明:

TSE に関する構造帰納法を用いて証明する.

$T = \varepsilon$ のとき

$\rho_s(\varepsilon, E - E_e(t)) = \varepsilon$ より，

$$\begin{aligned} &\rho_s(t; \varepsilon, E) \\ &= \rho_s(t; \rho_s(\varepsilon, E - E_e(t)), E) \end{aligned}$$

$T = \perp$ のとき

$$\begin{aligned} &\rho_s(t; \perp, E) = \rho_s(\perp, E) = \perp \\ &\rho_s(t; \rho_s(\perp, E - E_e(t)), E) = \rho_s(t; \perp, E) = \rho_s(\perp, E) = \perp \\ &\therefore \rho_s(t; \perp, E) = t; \rho_s(\perp, E - E_e(t)) \end{aligned}$$

$T = T_1 + T_2$ のとき

$$
\begin{aligned}
&\rho_s(t; (T_1 + T_2), E) \\
&= \rho_s(t; T_1 + t; T_2, E) \\
&= \rho_s(t; T_1, E) + \rho_s(t; T_2) \\
&= \rho_s(t; \rho_s(T_1, E - E_e(t)), E) + \rho_s(t; \rho_s(T_2, E - E_e(t)), E) \\
&= \rho_s(t; (\rho_s(T_1, E - E_e(t)) + \rho_s(T_2, E - E_e(t))), E) \\
&= \rho_s(t; \rho_s(T_1 + T_2, E - E_e(t)), E)
\end{aligned}
$$

$T = t'; T'$ かつ $t = !e \wedge t' = ?e \wedge e \in E$ のとき

$$
\begin{aligned}
&\rho_s(t; T, E) = \rho_s(!e; ?e; T', E) = !e; ?e; \rho_s(T', E) \\
&\rho_s(t; \rho_s(T, E - E_e(t)), E) \\
&= \rho_s(!e; \rho_s(?e; T', E - \{e\}), E) \\
&= \rho_s(!e; ?e; \rho_s(T', E - \{e\}), E) \\
&= !e; ?e; \rho_s(\rho_s(T', E - \{e\}), E) \\
&= !e; ?e; \rho_s(T', E) \\
&\therefore \rho_s(t; T, E) = \rho_s(t; \rho_s(T, E - E_e(t)), E)
\end{aligned}
$$

$T = t'; T'$ かつ $t \notin ?E \wedge t \notin !E$ のとき

$$
\begin{aligned}
&\rho_s(t; T, E) = \rho_s(t; T, E) = t; \rho_s(T, E) \\
&\rho_s(t; \rho_s(T, E - E_e(t)), E) \\
&= \rho_s(t; \rho_s(T, E), E) \\
&= t; \rho_s(\rho_s(T, E), E) \\
&= t; \rho_s(T, E) \\
&\therefore \rho_s(t; T, E) = \rho_s(t; \rho_s(T, E - E_e(t)), E)
\end{aligned}
$$

上記以外のとき

$$
\begin{aligned}
&\rho_s(t; T, E) = \bot \\
&\rho_s(t; \rho_s(T, E - E_e(t)), E) = \bot \\
&\therefore \rho_s(t; T, E) = \rho_s(t; \rho_s(T, E - E_e(t)), E)
\end{aligned}
$$

以上より，任意の TSE T とプリミティブ t に対して，以下の式が成り立つ．

$$\rho_s(t;T,E) = \rho_s(t;\rho_s(T,E-E_e(t)),E)$$

□

5.3.2 非同期イベント通信

同期イベント通信では，イベントを送信する側は，送信したイベントが受け取られないと次の動作に移ることができなかったのに対して，非同期イベント通信では，送信したイベントが受け取られなくても次の動作を行うことができる．ただし，送信されたイベントは，その順序で受け取られるものとする．このようなイベント通信は，次の除去関数を用いて表す．

定義 20 (非同期イベント通信)

$$\rho_a(T,E) \equiv \rho'_a(T,E,\varepsilon)$$

$$\rho'_a(T,E,R) \equiv$$

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

ここで，T は TSE を表し，t は $\varepsilon, \perp$ 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．

このように定義した除去関数 ρ_a もまた定義 18 の制約を満たす．

証明

1 の証明:

ρ'_a の定義に従って，帰納法を用いて証明する．

$\underline{T = T_1 + T_2 \text{ のとき}}$

$$\begin{aligned}
&\rho'_a(T, \{\}, R) \\
&= \rho'_a(T_1 + T_2, \{\}, R) \\
&= \rho'_a(T_1, \{\}, R) + \rho'_a(T_2, \{\}, R) \\
&= \rho'_a(T_1, \{\}, R) + T_2 \\
&= T_1 + T_2 \\
&= T
\end{aligned}$$

$\underline{T = !e; T' \wedge e \in E}$,

$\underline{T = ?e; T' \wedge R = ?e; R'}$,

$\underline{T = ?e; T' \wedge R = ?e'; R' \wedge e \neq e' \text{ のとき}}$

$E = \{\}$ よりあり得ない．

$\underline{T = t; T' \wedge t \notin ?E \wedge t \notin !E \text{ のとき}}$

$$\rho'_a(T, E, R) = \rho'_a(t; T', \{\}, R) = t; \rho'_a(T', \{\}, R) = t; T' = T$$

$\underline{T = \varepsilon \text{ のとき}}$

$$\rho'_a(T, \{\}, R) = \rho'_a(\varepsilon, \{\}, R) = \varepsilon = T$$

$\underline{T = \bot \text{ のとき}}$

$$\rho'_a(T, E, R) = \rho'_a(\bot, \{\}, R) = \bot = T$$

よって，任意の TSE T に対して，

$$\begin{aligned}
&\rho'_a(T, \{\}, R) = T \\
&\therefore \rho_a(T, \{\}) = T
\end{aligned}$$

2 の証明:

ρ'_a の定義に従って，帰納法を用いて証明する．

$\underline{T = T_1 + T_2 \text{ のとき}}$

$$\rho'_a(T, E, R) = \rho'_a(T_1 + T_2, E, R) = \rho'_a(T_1, E, R) + \rho'_a(T_2, E, R)$$
$$\subseteq \rho'_a(T_1, E, R) + T_2 \subseteq T_1 + T_2 = T$$

$\underline{T = !e; T' \wedge e \in E \text{ のとき}}$

$$\rho'_a(T, E, R) = \rho'_a(!e; T', E, R) = !e; \rho'_a(T', E, R; ?e) \subseteq !e; T' = T$$

$\underline{T = ?e; T' \wedge R = ?e; R' \text{ のとき}}$

$$\rho'_a(T, E, R) = \rho'_a(?e; T', E, ?e; R') = ?e; \rho'_a(T', E, R') \subseteq ?e; T' = T$$

$\underline{T = ?e; T' \wedge R = ?e'; R' \wedge e \neq e' \text{ のとき}}$

$$\begin{aligned}
&\rho'_a(T, E, R) \\
&= \rho'_a(?e; T', E, ?e'; R') \\
&= \rho'_a(?e; T', E, R') \\
&= \begin{cases} ?e; \rho'_a(T', E, R'') \subseteq ?e; T' = T \\ \bot \subseteq T \end{cases}
\end{aligned}$$

$\underline{T = t; T' \wedge t \notin ?E \wedge t \notin !E \text{ のとき}}$

$$\rho'_a(T, E, R) = t; \rho'_a(T', E, R) \subseteq t; T' = T$$

$\underline{T = \varepsilon \text{ のとき}}$

$$\rho'_a(\varepsilon, E, R) = \varepsilon \subseteq \varepsilon = T$$

$\underline{T = \bot \text{ のとき}}$

$$\rho'_a(\bot, E, R) = \bot \subseteq \bot = T$$

以上より，任意の TSE T に対して，

$$\rho'_a(T, E, R) \subseteq T$$
$$\therefore\ \rho_a(T, E) \subseteq T$$

3 の証明:

定義より，$\rho_a(\perp, E) = \perp$.

4 の証明:

定義より，$\rho_a(\varepsilon, E) = \varepsilon$.

5 の証明:

定義より，$\rho_a(T_1 + T_2, E) = \rho_a(T_1, E) + \rho_a(T_2, E)$.

6 の証明:

$t = !e \wedge e \in E$ のとき

$$
\begin{aligned}
&\rho_a(t; T, E) \\
&= \rho'_a(!e; T, E, \varepsilon) \\
&= !e; \rho'_a(T, E, ?e) \\
&\rho_a(t; \rho_a(T, E - E_e(t)), E) \\
&= \rho'_a(!e; \rho_a(T, E - \{e\}), E, \varepsilon) \\
&= !e; \rho'_a(\rho_a(T, E - \{e\}), E, ?e) \\
&= !e; \rho'_a(T, E, ?e)
\end{aligned}
$$

$t = ?e$ のとき

$$
\begin{aligned}
&\rho_a(t; T, E) = \rho'_a(?e; T, E, \varepsilon) = \perp \\
&\rho_a(t; \rho_a(T, E - E_e(t)), E) \\
&= \rho_a(?e; \rho_a(T, E - E_e(t)), E) \\
&= \perp \\
&\therefore \rho_a(t; T, E) = \rho_a(t; \rho_a(T, E - E_e(t)), E)
\end{aligned}
$$

t が上記以外のとき

$$
\begin{aligned}
&\rho_a(t; T, E) = \rho'_a(t; T, E, \varepsilon) = t; \rho'_a(T, E, \varepsilon) \\
&\rho_a(t; \rho_a(T, E - E_e(t)), E) = t; \rho'_a(\rho_a(T, E - E_e(t)), E, \varepsilon) = t; \rho'_a(T, E, \varepsilon) \\
&\therefore\ \rho_a(t; T, E) = \rho_a(t; \rho_a(T, E - E_e(t)), E)
\end{aligned}
$$

以上より，任意の TSE T とプリミティブ t に対して，

$$\rho_a(t; T, E) = \rho_a(t; \rho_a(T, E - E_e(t)), E)$$

□

5.3.3 除去関数の順序関係

除去関数の関係を表すために，次の関係 $\subseteq$ を定義する．

定義 21 (除去関数の順序関係)

$$\rho' \subseteq \rho \equiv \forall T, E \,.\, \rho'(T, E) \subseteq \rho(T, E)$$

ここで，T は TSE，E はイベント集合である．

イベント通信 C のための除去関数を ρ' とし，イベント通信 C' の除去関数を ρ' とする．このとき，$\rho' \subseteq \rho$ となれば，ρ' は ρ よりも**強い除去関数**であると呼び，逆に ρ は ρ' より**弱い除去関数**であると呼ぶ．ここで，便宜上，C' は C よりも**強いイベント通信**であると呼び，C は C' よりも**弱いイベント通信**であると呼ぶことにする．同期イベント通信と非同期イベント通信に関しては，以下の定理が成り立つ．

定理 8 同期イベント通信は，非同期イベント通信よりも強いイベント通信である．つまり，任意の TSE T とイベント集合 E に対して，以下の関係が成り立つ．

$$\rho_s(T, E) \subseteq \rho_a(T, E)$$

証明 T に関する構造帰納法を用いて証明する．

$T = \varepsilon$ のとき

$$\rho_s(T, E) = \rho_s(\varepsilon, E) = \varepsilon$$
$$\rho_a(T, E) = \rho_a(\varepsilon, E) = \varepsilon$$
$$\therefore\ \rho_s(T, E) \subseteq \rho_a(T, E)$$

$T = \perp$ のとき

$$\rho_s(T, E) = \rho_s(\perp, E) = \perp$$
$$\rho_a(T, E) = \rho_a(\perp, E) = \perp$$
$$\therefore\ \rho_s(T, E) \subseteq \rho_a(T, E)$$

$T = T_1 + T_2$ のとき

$$\rho_s(T, E) = \rho_s(T_1 + T_2, E) = \rho_s(T_1, E) + \rho_s(T_2, E)$$
$$\rho_a(T, E) = \rho_a(T_1 + T_2, E) = \rho_a(T_1, E) + \rho_a(T_2, E)$$
$$\therefore\ \rho_s(T, E) \subseteq \rho_a(T, E)$$

$T = t; T'$ のとき

以下のいづれかが成り立つ.

$$\rho_s(T, E) = \rho_s(t; T', E) = t; \rho_s(T', E)$$
$$\rho_s(T, E) = \rho_s(t; t'; T'', E) = t; t'; \rho_s(T'', E)$$
$$\rho_s(T, E) = \rho_s(t; T', E) = \bot$$

$t; \rho_s(T', E)$ となるのは $t \notin ?E \cup !E$ の場合である．このとき，

$$\rho_a(T, E) = \rho_a(t; T', E) = t; \rho_a(T', E)$$
$$\therefore \rho_s(T, E) \subseteq \rho_a(T, E)$$

$t; t'; \rho_s(T'', E)$ となるのは，$t = !e \wedge t' = ?e \wedge e \in E$ の場合である．このとき，

$$\rho_a(T, E) = \rho'_a(!e; ?e; T'', E, \varepsilon)$$
$$= !e; \rho'_a(?e; T'', E, ?e) = !e; ?e; \rho'_a(T'', E, ?e)$$
$$\therefore \rho_s(T, E) \subseteq \rho_a(T, E)$$

$\bot$ は任意の TSE T に対して $\bot \subseteq T$ なので，明らかに，$\rho_s(\bot, E) \subseteq \rho_a(T, E)$．以上より，

$$\forall T, E\ .\ \rho_s(T, E) \subseteq \rho_a(T, E)$$

□

5.4 振舞い近似手法

5.4.1 近似関数

振舞い近似手法では，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．例えば，あるステートチャートの振舞いを，別のステー

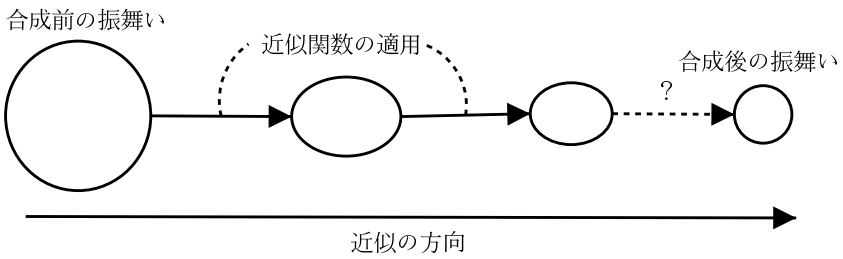

図 5.5: 近似関数

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．このような振舞い近似のための関数を**近似関数**と呼ぶ．近似関数による TSE の変化の様子は図 5.5 の通りである．このように元の振舞いよりも小さく，合成後の振舞いよりも大きい振舞いへ変形させる．そして，繰返し近似関数を適用することによって，合成後の振舞いへ近づけることができる．まず，合成後の振舞いを求めるために，合成後の TSE から，ある TSE の記号だけを集める関数を定義する．

定義 22 (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 $/$ を以下の通りに定義する．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' は TSE を表し，t は TSE の $\varepsilon, \perp$ 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．近似関数の定義は次の通りである．

定義 23 (近似関数) 以下の性質を満たす関数 γ を除去関数 ρ 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．

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

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，一つ目の性質の代わりに，

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

を用いてもよい．近似関数における関係 $\subseteq$ を次の通りに定義する．

定義 24

$$\gamma' \subseteq \gamma \equiv \forall T, T', E \,.\, \gamma'(T, T', E) \subseteq \gamma(T, T', E)$$

ここで，T, T' は TSE であり，E はイベント集合である．

このとき，γ' は γ より**強い近似関数**と呼び，逆に γ は γ' より**弱い近似関数**であると呼ぶ．

$\rho' \subseteq \rho$ となるとき，ρ に対する近似関数は ρ' に対する近似関数でもある．このことは，次の定理により表す．

定理 9 ρ', ρ を除去関数とする．$\rho' \subseteq \rho$ かつ γ が ρ に対する近似関数のとき，γ は ρ' に対する近似関数でもある．

証明 $\rho' \subseteq \rho$ より，任意の TSE T, T' とイベント集合 E に対して，

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

一方で，γ は ρ に対する近似関数であるため，

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

よって，$\rho'(T \times T', E) \subseteq \gamma(T, T', E)$ となる．
□

5.4.2 独立性

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，$C(\{T, T'\}, E)/T$ 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**と呼び次の通りに定義する．

定義 25 (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．

この独立性が成り立つとき，次の定理が成り立つ．

定理 10 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 17，18 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．

□

本論文では，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．

5.4.3 同期・非同期イベント通信に対する近似関数

定義 20 で示した非同期イベント通信に対する近似関数 γ_a を導入する．この近似関数は，定理 9 より，定義 19 で示した同期イベント通信に対する近似関数でもある．

定義 26 (近似関数 γ_a)

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

ここで，T_1, T_2 は TSE を表し，t_1, t_2 は $\varepsilon, \perp$ を除くプリミティブを表す．この近似関数 γ_a 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．

定理 11 γ_a は任意の TSE T_1,T_2 に対して定義 23 の近似関数に関する制約を満たす．つまり以下の 2 式が成り立つ．

$$\rho_a(T_1 \times T_2, E)/T \subseteq \gamma_a(T_1, T_2, E)$$
$$\gamma_a(T_1, T_2, E) \subseteq T$$

証明

$\rho_a(T_1 \times T_2, E)/T \subseteq \gamma_a(T_1, T_2, E)$ の証明:
定義に従って帰納法を用いて証明する．

$T_1 = T_1' + T_1''$ のとき，

$$\begin{aligned}
&\rho_a(T_1 \times T_2, E)/T_1 \\
&= \rho_a((T_1' + T_1'') \times T_2, E)/T_1 \\
&= \rho_a((T_1' \times T_2) + (T_1'' \times T_2), E)/T_1 \\
&= \rho_a(T_1' \times T_2, E)/T_1 + \rho_a(T_1'' \times T_2, E)/T_1 \\
&\subseteq \gamma_a(T_1', T_2, E) + \gamma_a(T_1'', T_2, E)
\end{aligned}$$

$T_2 = T_2' + T_2''$ のとき，

$$\begin{aligned}
&\rho_a(T_1 \times T_2, E)/T_1 \\
&= \rho_a(T_1 \times (T_2' + T_2''), E)/T_1 \\
&= \rho_a(T_1 \times T_2', E)/T_1 + \rho_a(T_1 \times T_2'', E)/T_1 \\
&\subseteq \gamma_a(T_1, T_2', E) + \gamma_a(T_1, T_2'', E)
\end{aligned}$$

$T_1 = ?e; T_1', T_2 = !e; T_2', e \in E$ のとき，

$$\begin{aligned}
&\rho_a(T_1 \times T_2, E)/T_1 \\
&= \rho_a(?e; T_1' \times !e; T_2', E)/T_1 \\
&= \rho_a(?e; (T_1' \times !e; T_2'), E)/T_1 + \rho_a(!e; (?e; T_1' \times T_2'), E)/T_1 \\
&= \rho_a(!e; (?e; T_1' \times T_2'), E)/T_1 \\
&= ?e; (\rho_a(T_1', T_2', E)/T_1') \\
&\subseteq ?e; \gamma_a(T_1', T_2', E)
\end{aligned}$$

$T_1 = t; T_1', t \notin ?E$ のとき，

$$\begin{aligned}
&\rho_a(T_1 \times T_2, E)/T_1 \\
&= \rho_a(t; T_1' \times T_2, E)/T_1 \\
&= t; (\rho_a(T_1' \times T_2, E)/T_1') \\
&\subseteq t; \gamma_a(T_1', T_2, E)
\end{aligned}$$

$T_1 = ?e; T_1', T_2 = t_2; T_2', t_2 \notin !E$ のとき，

$$\begin{aligned}
&\rho_a(T_1 \times T_2, E)/T_1 \\
&= \rho_a(?e; T_1' \times t_2; T_2', E)/T_1 \\
&= \rho_a(?e; (T_1' \times t_2; T_2'), E)/T_1 + \rho_a(t_2; (?e; T_1' \times T_2'), E)/T_1 \\
&= \rho_a(t_2; (?e; T_1' \times T_2'), E)/T_1 \\
&= \rho_a(T_1', T_2', E)/T_1' \\
&\subseteq \gamma_a(T_1', T_2', E)
\end{aligned}$$

$T_1 = \varepsilon$ のとき，

$$\begin{aligned}
&\rho_a(T_1 \times T_2, E)/T \\
&= \rho_a(\varepsilon \times T_2, E)/\varepsilon \\
&= \varepsilon
\end{aligned}$$

上記以外の場合，$\rho_a(T_1 \times T_2, E)/T_1 = \perp$ である．
以上より，$\rho_a(T_1 \times T_2, E)/T_1 \subseteq \gamma_a(T_1, T_2, E)$ が成り立つ．
$\gamma_a(T_1, T_2, E) \subseteq T_1$ も同様に帰納法を用いて証明できる．□

γ_a の近似に利用される T_2 については，出力イベントのみが結果に影響を与えている．このため，T_2 の出力イベント列 $E_o(T_2)$ を T_2 の代わりに用いても同じ結果を得ることができる．

定理 12 (γ_a の性質)

$$\gamma_a(T_1, T_2, E) = \gamma_a(T_1, E_o(T_2), E)$$

ここで，T_1, T_2 は任意の TSE である．

証明 定義に従って帰納法を用いて証明する．
$T_1 = T_1' + T_1''$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(T_1' + T_1'', E_o(T_2), E) \\
&= \gamma_a(T_1', E_o(T_2), E) + \gamma_a(T_1'', E_o(T_2), E) \\
&= \gamma_a(T_1', T_2, E) + \gamma_a(T_1'', T_2, E) \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

$T_2 = T_2' + T_2''$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(T_1, E_o(T_2' + T_2''), E) \\
&= \gamma_a(T_1, E_o(T_2') + E_o(T_2''), E) \\
&= \gamma_a(T_1, E_o(T_2'), E) + \gamma_a(T_1, E_o(T_2''), E) \\
&= \gamma_a(T_1, T_2', E) + \gamma_a(T_1, T_2'', E) \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

$T_1 = ?e; T_1' \wedge T_2 = !e; T_2' \wedge e \in E$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(?e; T_1', E_o(!e; T_2'), E) \\
&= \gamma_a(?e; T_1', !e; E_o(T_2'), E) \\
&= ?e; \gamma_a(T_1', E_o(T_2'), E) \\
&= ?e; \gamma_a(T_1', T_2', E) \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

$T_1 = t; T_1' \wedge t \notin ?E$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(t; T_1', E_o(T_2), E) \\
&= t; \gamma_a(T_1', E_o(T_2), E) \\
&= t; \gamma_a(T_1', T_2, E) \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

$T_1 = ?e; T_1' \wedge T_2 = t_2; T_2' \wedge t_2 \notin !E$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(T_1, E_o(t_2; T_2'), E) \\
&= \gamma_a(T_1, E_o(T_2'), E) \\
&= \gamma_a(T_1, T_2', E) \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

$T_1 = \varepsilon$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(\varepsilon, E_o(T_2), E) \\
&= \varepsilon \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

$T_2 = \varepsilon$ のとき

$$
\begin{aligned}
&\gamma_a(T_1, E_o(T_2), E) \\
&= \gamma_a(T_1, E_o(\varepsilon), E) \\
&= \gamma_a(T_1, \varepsilon, E) \\
&= \gamma_a(T_1, T_2, E)
\end{aligned}
$$

その他のとき

$$\gamma_a(T_1, E_o(T_2), E) = \bot = \gamma_a(T_1, T_2, E)$$

□

5.4.4 振舞い近似手法の妥当性

我々が提案する検証手法では，近似を繰返し適用することによって得られたTSE を 4 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 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，

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

と定義すると，

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

が成り立つ．このため，命題を証明するためには，(A) 式が成り立つことを言えれば十分である．このことは，以下の通りに証明できる．

$$\begin{aligned} & C(\{T_1, T_2\} \cup TS_{12}, E) \\ & = \rho(T_1 \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) \\ & = \rho((T_1' + T_1'') \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) \\ & = \rho((T_1' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T) + (T_1'' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T), E) \\ & = \rho(T_1' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) + \rho(T_1'' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) \\ & = \rho(T_1' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) + \rho(\rho(T_1'' \times T_2, E - E') \times \textstyle\prod_{T \in TS_{12}} T, E) \text{(定理 7)} \\ & = \rho(T_1' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) + \bot \\ & = \rho(T_1' \times T_2 \times \textstyle\prod_{T \in TS_{12}} T, E) \\ & = C(TS_1 \cup \{\rho(T_1 \times T_2, E)/T_1\}, E) \\ & \subseteq C(TS_1 \cup \{\gamma(T_1, T_2, E)\}, E) \end{aligned}$$

$= C(\{\gamma_{c12}, T_2\} \cup TS, E)$
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 13 (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し，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，

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

が成り立つ．ここで，P, Q は表明である．

証明 定理 10 より，

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

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，

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

が成り立つ．そのため，(A),(B) より，

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

が成り立つ．以上より，定理 13 は成り立つ．
□
この定理により，近似された TSE で証明できることは合成後のステートチャートにおいても証明できるという事実

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

と，繰返し近似を適用したTSEで証明できることは合成後のステートチャートにおいても証明できるという事実

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

を表している．

近似関数を用いてあるTSEを近似するとき，計算途中でより弱い近似関数に変更しても定理13は成り立つ．このことは次の定理より明らかとなる．

定理 14 γ, γ' を $\gamma' \subseteq \gamma$ となる近似関数とする．このとき以下の式が成り立つ．

$$\{P\}\gamma(T, T', E)\{Q\} \Rightarrow \{P\}\gamma'(T, T', E)\{Q\}$$

ここで，T, T' は任意のTSEであり，Eはイベント集合である．また，P, Q は表明である．

証明 $T_1 \subseteq T_2$ となる任意のTSEに対して，$T_1 + T_1' = T_2$ となる T_1' が存在する．このため，

$$\{P\}T_2\{Q\} \Leftrightarrow \{P\}T_1 + T_1'\{Q\}$$

が成り立つ．また，

$$\{P\}T_1 + T_1'\{Q\} \Rightarrow \{P\}T_1\{Q\}$$

となるため，

$$\{P\}T_2\{Q\} \Rightarrow \{P\}T_1\{Q\}$$

が成り立つ．ここで，$\gamma' \subseteq \gamma$ であるので，

$$\{P\}\gamma(T, T', E)\{Q\} \Rightarrow \{P\}\gamma'(T, T', E)\{Q\}$$

が成り立つ．

□

5.4.5 振舞い近似手法を用いた検証例

本章で提案した近似手法を，図5.1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，$n \geq 0$が常に成り立つことを，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，5.4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，4章の方法を用いて$n \geq 0$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，常にIncのみが実行されることが分かる．

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はそれぞれ次の通りである．

$$
\begin{aligned}
L &= (?a; !b{+}?c; !d)*\\
R &= Init; !a; (((?b; Inc; !c{+}?d; Dec; !c);\\
&\qquad\qquad (?b; !a{+}?d; !a))*);\\
&\qquad\qquad (?b; Inc; !c{+}?d; Dec; !c + \varepsilon)
\end{aligned}
$$

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' はそれぞれ次の通りになる．

$$\begin{aligned} L' &= \gamma_a(L, R, \{a, b, c, d\}) \\ &= \gamma_a(L, E_o(R), \{a, b, c, d\}) \\ &= (?a; !b; (?c; !d + \varepsilon))* \\ R' &= \gamma_a(R, L', \{a, b, c, d\}) \\ &= \gamma_a(R, E_o(L'), \{a, b, c, d\}) \\ &= Init; !a; (?b; Inc; !c; ?d; !a)* \end{aligned}$$

そして，R'では，Incのみが実行されることが分かる．ここで，計算を簡単に行うために定理12を用いたことに注意したい．$E_o(R) = !a; (!c + !c + !a + !a)*; (!c + !d + \varepsilon)$ となるが，$!c + !c$ などは一つにまとめて $E_o(R) = !a; (!a + !c)*; (!c + !d + \varepsilon)$ とできる．このように，近似関数によって，式の計算が簡単になる．このため，合成によって状態数が多くなるモデルにおいては，振舞い近似手法は計算量と状態数を減少させる有効な手段である．

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，4 章で示した方法によって証明できる．証明の詳細は表 5.1 に示す．

	導出できる A-TSE	理由
1	$\{true\}Init\{n \geq 0\}$	
2	$\{n \geq 0\}!a\{n \geq 0\}$	
3	$\{n \geq 0\}!b\{n \geq 0\}$	
4	$\{n \geq 0\}Inc\{n \geq 1\}$	
5	$\{n \geq 1\}!c\{n \geq 1\}$	
6	$\{n \geq 1\}?d\{n \geq 1\}$	
7	$\{n \geq 1\}!a\{n \geq 1\}$	
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

表 5.1: $\{true\}R'\{n \geq 0\}$ の証明

第 6章

現実問題への適用

6.1 DHCPサーバ

本論文における振舞い近似手法を用いて，ネットワーク上のIP管理で頻繁に用いられるDHCPサーバ [Dro97] の振舞いの検証を行う．簡単化のため，DHCPクライアントがIPを取得し解放するまでの仕様を以下の通りとし，このためのサーバとクライアントの動作のモデルをステートチャートを用いて図6.2に示す．

- クライアントはDHCPサーバを見つけるためにDHCPDISCOVERをブロードキャストする．
- サーバはリース可能なIPが1個以上あるときにDHCPDISCOVERを受け取ると，DHCPOFFERをIPアドレスと共にクライアントに送信する．
- クライアントはDHCPOFFERを受け取るとDHCPREQUESTを送信して，受け取ったIPアドレスを受け入れる．
- サーバはDHCPREQUESTを受け取るとDHCPPACKをクライアントに送信する．
- クライアントはDHCPPACKを受け取ってからIPアドレスを設定する．
- クライアントはDHCPRELEASEによってIPアドレスを解放することをサーバに伝える．

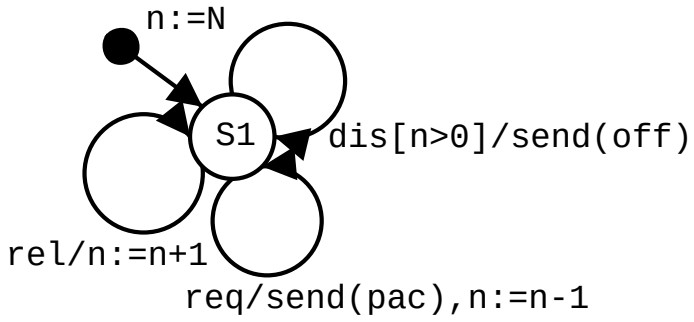

図 6.1: DHCP サーバ

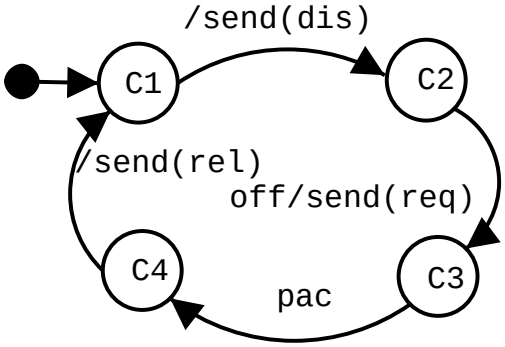

図 6.2: DHCP クライアント

- サーバは，DHCPRELEASE を受け取ると，リース可能な IP の数を 1 増やす.

このステートチャートで用いたイベントは，それぞれ以下の通りに DHCP プロトコルのメッセージに対応する.

DHCP メッセージ	イベント
DHCPDISCOVER	dis
DHCPOFFER	off
DHCPREQUEST	req
DHCPPACK	pac
DHCPRELEASE	rel

図 6.2 の DHCP サーバでは，リース可能な IP アドレスの個数を属性 n を用いて表している．N はサーバ起動時に保有しているリース可能な IP アドレスの個数である．このサーバの振舞いを表すステートチャートに対して常に $0 \leq n \leq N$

であることの検証を行う．サーバ側のステートチャートだけでは，$n := n - 1$ と $n := n + 1$ というアクションが，どのような順序で実行されるのかが決定できない．サーバの振舞いとクライアントの振舞いが相互に依存しているためである．そこで，サーバの振舞いとクライアントの振舞いを表すそれぞれのステートチャートに対して振舞い近似手法を適用して検証を行う．

6.2 検証の準備

通常，DHCP サーバとクライアントでは UDP を用いて通信を行う．この通信は非同期で行われるため，本論文では定義 20 の非同期イベント通信として図 6.2 のモデルを扱う．

まず最初に，DHCP サーバとクライアントの初期状態から各々の状態への遷移系列を求めるために，それぞれの TSE を $Server, Client$ として次の連立方程式を作る．

$$
\begin{aligned}
Server &= n := 1; S_1 \\
S_1 &= (?dis; [n > 0]; !off + ?req; !pac; n := n - 1 + ?rel; n := n + 1); S_1 + \varepsilon \\
Client &= C_1 \\
C_1 &= !dis; C_2 + \varepsilon \\
C_2 &= ?off; !req; C_3 + \varepsilon \\
C_3 &= ?pac; C_4 + \varepsilon \\
C_4 &= !rel; C_1 + \varepsilon
\end{aligned}
$$

これらの方程式から，以下の解を得る．

$$
\begin{aligned}
Server &= n := 1; (?dis; [n > 0]; !off + ?req; !pac; n := n - 1 + ?rel; n := n + 1)* \\
Client &= (!dis; ?off; !req; ?pac; !rel)*; (!dis; ?off; !req; ?pac + !dis; ?off; !req + !dis + \varepsilon)
\end{aligned}
$$

6.3 DHCP サーバの検証

サーバの振舞いがクライアントの振舞いに依存しているため，クライアントの振舞いを利用してサーバの振舞いを近似する．ここで，

$$
\begin{aligned}
?DIS &= ?dis; [n > 0]; !off \\
?REQ &= ?req; !pac; n := n - 1 \\
?REL &= ?rel; n := n + 1 \\
S &= ?DIS + ?REQ + ?REL \\
C &= !dis; !req; !rel \\
C' &= !dis; !req + !dis + \varepsilon \\
E &= \{dis, off, req, rel\}
\end{aligned}
$$

とする．近似関数は，定義 26 を用いる．

$$
\begin{aligned}
&Server' \\
&= \gamma_a(Server, E_o(Client), E) \\
&= \gamma_a(n := N; S*, C*; C', E) \\
&= \gamma_a(n := N; S*, (C; C* + \varepsilon); C', E) \\
&= \gamma_a(n := N; S*, C; C*; C' + C', E) \\
&= \gamma_a(n := N; S*, C; C*; C', E) + \gamma_a(n := N; S*, C', E) \\
&= n := N; \gamma_a(S*, C; C*; C', E) + n := N; \gamma_a(S*, C', E) \\
&= n := N; \gamma_a(S*, C; C*; C' + C', E) \\
&= n := N; \gamma_a(S*, C*; C', E)
\end{aligned}
$$

$$
\begin{aligned}
&\gamma_a(S*, C', E) \\
&= \gamma_a(S; S* + \varepsilon, C', E) \\
&= \gamma_a(S; S*, C', E) + \gamma_a(\varepsilon, C', E) \\
&= ?DIS; \gamma_a(S*, !req; !rel, E) + \varepsilon \\
&= ?DIS; (?REQ; (?REL; \gamma_a(S*, \varepsilon, E) + \varepsilon) + \varepsilon) + \varepsilon \\
&= ?DIS; (?REQ; (?REL; \bot + \varepsilon) + \varepsilon) + \varepsilon \\
&= ?DIS; ?REQ + ?DIS + \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
&\gamma_a(S*, C*; C', E)\\
&= \gamma_a(S*, C; C*; C', E) + \gamma_a(S*, C', E)\\
&= \gamma_a(S*, C; C*; C', E) + \gamma_a(S*, C', E)\\
&= \gamma_a(S; S* + \varepsilon, C; C*; C', E) + \gamma_a(S*, C', E)\\
&= \gamma_a(S; S*, C; C*; C', E) + \gamma_a(\varepsilon, C; C*; C', E) + \gamma_a(S*, C', E)\\
&= ?DIS; \gamma_a(S*, !req; !rel; C*; C', E) + \varepsilon + \gamma_a(S*, C', E)\\
&= \cdots\\
&= ?DIS; (?REQ; (?REL; \gamma_a(S*, C*; C', E) + \varepsilon) + \varepsilon) + \varepsilon + \gamma_a(S*, C', E)\\
&= ?DIS; ?REQ; ?REL; \gamma_a(S*, C*; C', E) + ?DIS; ?REQ + ?DIS + \varepsilon + \gamma_a(S*, C', E)\\
&= ?DIS; ?REQ; ?REL; \gamma_a(S*, C*; C', E) + ?DIS; ?REQ + ?DIS + \varepsilon
\end{aligned}
$$

$$
\therefore \gamma_a(S*, C * C', E) = (?DIS; ?REQ; ?REL)*; (?DIS; ?REQ + ?DIS + \varepsilon)
$$

$$
\therefore Server' = n := N; (?DIS; ?REQ; ?REL)*; (?DIS; ?REQ + ?DIS + \varepsilon)
$$

このようにして得られたTSEを4章の方法を用いて検証する．検証のために行った証明を図6.1に示す．この結果，DHCPサーバのリース可能なIPの個数nは，$0 \leq n \leq N$であることが証明できた．

6.4 複数のDHCPクライアント

DHCPクライアントが複数の場合，特に$N+1$個の場合を考える．ここで，Nはサーバがリース可能なIPの個数の初期値である．それぞれのクライアントを$C_0, C_1, \cdots, C_N$とする．このとき，これらのクライアントが独立性を満たすために，それぞれのクライアントが送受信するイベントに対しても添字を付けて区別する．

$$
\begin{aligned}
C_i = &(!dis_i; ?off_i; !req_i; ?pac_i; !rel_i)*;\\
&(!dis_i; ?off_i; !req_i; ?pac_i + !dis_i; ?off_i; !req_i + !dis_i + \varepsilon)
\end{aligned}
$$

	導出できる A-TSE	理由
1	$\{0 \leq n \leq N\}?dis\{0 \leq n \leq N\}$	
2	$\{0 \leq n \leq N\}[n > 0]\{0 < n \leq N\}$	
3	$\{0 < n \leq N\}!off\{0 < n \leq N\}$	
4	$\{0 \leq n \leq N\}?DIS\{0 < n \leq N\}$	1,2,3,concat
5	$\{0 < n \leq N\}?req\{0 < n \leq N\}$	
6	$\{0 < n \leq N\}!pac\{0 < n \leq N\}$	
7	$\{0 < n \leq N\}n := n - 1\{0 < n + 1 \leq N\}$	
8	$\{0 < n \leq N\}?REQ\{0 < n + 1 \leq N\}$	5,6,7,concat
9	$\{0 < n + 1 \leq N\}?rel\{0 < n + 1 \leq N\}$	
10	$\{0 < n + 1 \leq N\}n := n + 1\{0 < n \leq N\}$	
11	$\{0 < n + 1 \leq N\}?REL\{0 < n \leq N\}$	9,10,concat
12	$\{0 \leq n \leq N\}?DIS; ?REQ; ?REL\{0 < n \leq N\}$	4,8,11,concat
13	$\{0 \leq n \leq N\}?DIS; ?REQ; ?REL\{0 \leq n \leq N\}$	12,conseq
14	$\{0 \leq n \leq N\}(?DIS; ?REQ; ?REL) * \{0 \leq n \leq N\}$	13,iteration
15	$\{0 \leq n \leq N\}?DIS; ?REQ\{0 < n + 1 \leq N\}$ 4,8,concat	
16	$\{0 \leq n \leq N\}?DIS; ?REQ\{0 \leq n \leq N\}$ 15,conseq	
17	$\{0 \leq n \leq N\}?DIS\{0 \leq n \leq N\}$ 4,conseq	
18	$\{0 \leq n \leq N\}\varepsilon\{0 \leq n \leq N\}$	
19	$\{0 \leq n \leq N\}?DIS; ?REQ+?DIS + \varepsilon\{0 \leq n \leq N\}$ 16,17,18,choice	
20	$\{0 \leq n \leq N\}(?DIS; ?REQ; ?REL)*; (?DIS; ?REQ+?DIS + \varepsilon)\{0 \leq n \leq N\}$	14,19,concat
21	$\{true\}n := N\{0 \leq n \leq N\}$	
22	$\{0 \leq n \leq N\}n := N; (?DIS; ?REQ; ?REL)*; (?DIS; ?REQ+?DIS + \varepsilon)\{0 \leq n \leq N\}$	20,21,concat

表 6.1: DHCP サーバの検証

また，サーバ側も同様に以下の通りになる．

$$Server = n := 1; (\sum_{i=0}^{N}(?dis_i; [n > 0]; !off_i \\ +?req_i; !pac_i; n := n - 1 + ?rel_i; n := n + 1))*$$

次に，$Server'_i (= \gamma_a(Server, C_i, E_i))$ を考える．前節と同様に，式の簡単化のために以下のことを定義しておく．

$$\begin{array}{lcllcl} ?DIS_i & = & ?dis_i; [n > 0]; !off_i & S & = & \sum_{i=0}^{N} S_i \\ ?REQ_i & = & ?req_i; !pac_i; n := n - 1 & C_i & = & !dis_i; !req_i; !rel_i \\ ?REL_i & = & ?rel_i; n := n + 1 & C'_i & = & !dis_i; !req_i + !dis_i + \varepsilon \\ S_i & = & ?DIS_i + ?REQ_i + ?REL_i & E_i & = & \{dis_i, off_i, req_i, rel_i\} \\ S_i^- & = & \sum_{j=0}^{N} S_j - S_i & & & \\ ?DIS_i^+ & = & ?DIS_i + S_i^- & & & \\ ?REQ_i^+ & = & ?REQ_i + S_i^- & & & \\ ?REL_i^+ & = & ?REL_i + S_i^- & & & \end{array}$$

このとき，以下の通りの計算を行う．

$\gamma_a(Server, C'_i, E_i) \ = \ n := N; \gamma_a(S*, C_i*; C'_i, E_i)$

$\gamma_a(S*, C'_i, E_i)$

$= \gamma_a(S; S* + \varepsilon, C'_i, E_i)$

$= \gamma_a(S; S*, C'_i, E_i) + \gamma_a(\varepsilon, C'_i, E_i)$

$= ?DIS_i^+; \gamma_a(S*, !req; !rel, E_i) + \varepsilon$

$= ?DIS_i^+; (?REQ_i^+; (?REL_i^+; \gamma_a(S*, \varepsilon, E_i) + \varepsilon) + \varepsilon) + \varepsilon$

$= ?DIS_i^+; (?REQ_i^+; (?REL_i^+; \perp + \varepsilon) + \varepsilon) + \varepsilon$

$= ?DIS_i^+; ?REQ_i^+ + ?DIS_i^+ + \varepsilon$

$\gamma_a(S*, C_i*; C'_i, E_i)$

$= \gamma_a(S*, C_i; C_i*; C'_i, E_i) + \gamma_a(S*, C'_i, E_i)$

$= \gamma_a(S*, C_i; C_i*; C'_i, E_i) + \gamma_a(S*, C'_i, E_i)$

$= \gamma_a(S; S* + \varepsilon, C_i; C_i*; C'_i, E_i) + \gamma_a(S*, C'_i, E_i)$

$= \gamma_a(S; S*, C_i; C_i*; C'_i, E_i) + \gamma_a(\varepsilon, C_i; C_i*; C'_i, E_i) + \gamma_a(S*, C'_i, E_i)$

$=?DIS_i^+;\gamma_a(S*,!req;!rel;C_i*;C_i',E_i)+\varepsilon+\gamma_a(S*,C_i',E_i)$

$=\cdots$

$=?DIS_i^+;(?REQ_i^+;(?REL_i^+;\gamma_a(S*,C_i*;C_i',E_i)+\varepsilon)+\varepsilon)+\varepsilon+\gamma_a(S*,C_i',E_i)$

$=?DIS_i^+;?REQ_i^+;?REL_i^+;\gamma_a(S*,C_i*;C_i',E_i)+?DIS_i^+;?REQ_i^++?DIS_i^++\varepsilon+\gamma_a(S*,C_i',E_i)$

$=?DIS_i^+;?REQ_i^+;?REL_i^+;\gamma_a(S*,C_i*;C_i',E_i)+?DIS_i^+;?REQ_i^++?DIS_i^++\varepsilon$

$\therefore \gamma_a(S*,C_i*C_i',E_i)=(?DIS_i^+;?REQ_i^+;?REL_i^+)*;(?DIS_i^+;?REQ_i^++?DIS_i^++\varepsilon)$

よって，以上より，

$$Server_i'=n:=N;(?DIS_i^+;?REQ_i^+;?REL_i^+)*;(?DIS_i^+;?REQ_i^++?DIS_i^++\varepsilon)$$

となる．さらに，他のクライアントを用いて近似を行うと，

$$\underbrace{?dis_0;[n>0];!off_0;\cdots;?dis_N;[n>0];!off_N}_{N+1};\cdots$$
$$\cdots\underbrace{?req_0;!pac_0;n:=n-1;\cdots;?req_N;n:=n-1;!pac_N}_{N+1}$$

という部分式が現れる TSE を得ることができる．つまり，$N+1$ 個すべてのクライアントが同時に $[n>0]$ のガード条件を満たすことができる遷移列である．この場合，$n:=n-1$ のアクションが $N+1$ 回行われるため，$n>=0$ が満たされないことは明らかである．

複数のクライアントに対しても $0\leq n\leq N$ が成り立つためにはイベント通信モデルやモデルの改善が必要となる．例えば，サーバ側の振舞いを図 6.3 のステートチャートを考える．この図では，例えば

$$rel_0/n:=n+1$$
$$\cdots$$
$$rel_N/n:=n+1$$

という遷移は，$rel_0/n:=n+1, rel_1/n:=n+1,\cdots,rel_N/n:=n+1$ のそれぞれをラベルとしてもつ $N+1$ 個の遷移があるものとして考える．このステート

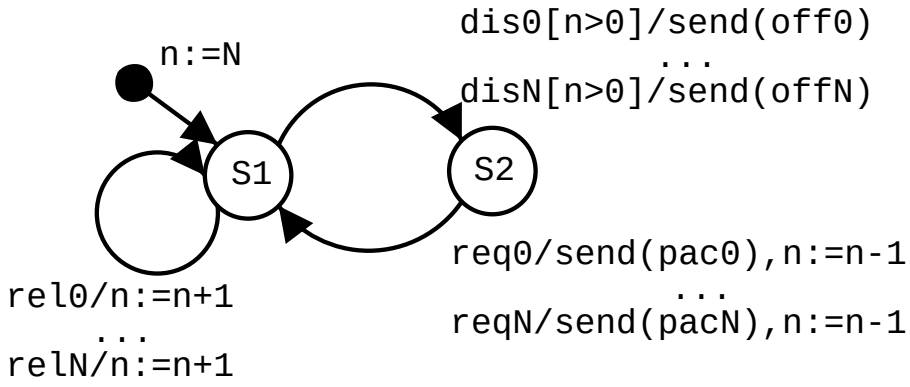

図 6.3: DHCP サーバ (改良後)

チャートでは，一つのクライアント i から dis_i イベント受け取ってから pac_i イベントを送信するまで他のクライアントの要求を受け付けないモデルとなっており，$[n > 0]$ のときには $!off_i; ?req_i; !pac_i; n := n - 1; \cdots$ という遷移が後に続くようになっているため，$n := n - 1$ がガード条件無しに連続して現れることがない．このため，$0 \leq n \leq N$ が保証できる．

第 7章

考察

7.1 論理 ATL の証明能力

4 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 7.1 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．常に $n \leq 0$ であることを証明する場合，$\{n \leq 0\}[n > 0]\{n \leq 0\}$ 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

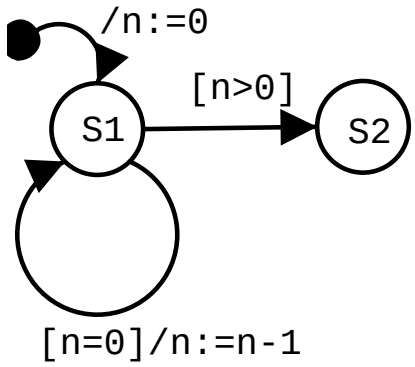

図 7.1: 到達できない状態が存在するステートチャート

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 により導びくことができる．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．

7.2 振舞い近似手法の拡張

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．例えば，図 7.2 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．

一方，複数の同じ振舞いを持つステートチャートを効率良く扱うことが重要である．6 章の DHCP サーバの例では，初めは，クライアントは一つだけを用意した．振舞い近似手法を用いるためには，5.4.2 節の独立性の制約が必要なためである．しかし，複数のクライアントと一つの DHCP サーバが存在するということが一般的であるため，クライアントを複数用意し，それぞれのクライアントで扱うイベントを，添字を付けることで区別した．同時に，サーバ側もそれらのイベントを扱えるように変更した．このように，同じ振舞いを行うそれぞ

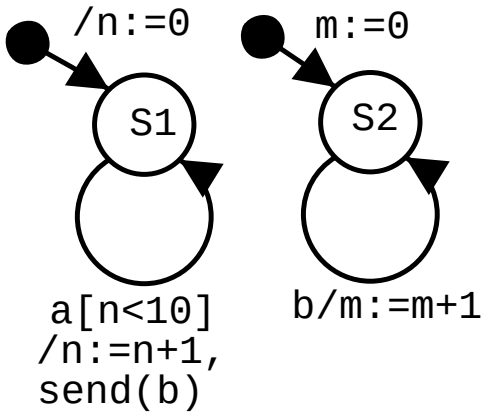

図 7.2: 近似手法では証明できない例

れのクライアントに対して，異なる属性とイベント名を用いなければならない．特に，オブジェクト指向方法論では，ステートチャートを用いてクラスから生成されるオブジェクトの振舞いを記述する．ある一つのクラスから生成された複数のオブジェクトはすべて同じ振舞いを持つことが多い．

7.3 イベント属性

本論文で提案した検証手法では，イベント属性を扱っていない．このため，DHCP サーバの検証においてはリース可能な IP アドレスを送信するなどの詳細までを含めた検証を行うことができない．このように，現実的な検証を行うためにはイベント属性を扱えるようにすることが非常に重要である．

イベントに値を付けて出力し，イベントが値を伴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り，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と考えている．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．

7.4 定理証明技術を用いた検証

CSP[Hoa85] 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なう方法 [SD01, SH96] 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，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，イベント通信方法を独立して定義できるようにしたため，幅広いソフトウェア開発へ適用可能である．

文献 [青木 01, SH96] 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 [青木 01] と連携した研究であり，本論文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．

第 8章

関連研究

8.1 モデル検査

有限オートマトンに対するモデル検査手法 [EOD00] に関する研究がある．この手法では，協調して動作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．このため，有限オートマトンに対するモデル検査手法は，本研究で行っている定理証明による検証方法よりも実用的に用いられている．また，BDD(Binary Decision Diagram) と呼ばれるデータ表現を用いて状態遷移図を表現することにより，非常に多くの状態と遷移を扱うことが可能である．しかし，オブジェクト指向分析・設計で用いられるステートチャートでは，属性値として整数や文字列などの無限の値を持つことのできる属性を扱うことがある．このような属性を扱う状態遷移図に対して，状態探索を基にしたモデル検査手法を適用することができない．一方で，本研究で提案している手法では，演繹による検証を行うため無限の値を持つ属性を直接取り扱うことができる．

8.2 ラベル付き遷移システム

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)[CK96, CGK97] 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において，CRA の安全性検証の説明することができる．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．しかしながら，検証すべき性質は，本論文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ことが明らかとなった．

また，振舞いを合成する際に，振舞いの仕様間の無矛盾性を検証するために中間仕様を用いる手法が提案されている [iso02]．この手法では，振舞いにプロセス代数 tCCA，仕様にプロセス論理 tLCA を用いる．仕様は，振舞いに対する性

質を記述するためのものであり，時間の概念を取り扱うことができる．しかし，プロセス論理 tLCA では，二つの仕様に対する無矛盾性は決定不能な問題であることが示されている．そこで，振舞いと仕様の間に，中間仕様というものを定義し，無矛盾性検証が決定可能であることを示している．この中間仕様にはプロセス代数 tICCA を用いている．中間仕様の意味はラベル付き遷移システムによって与えられ，互いの振舞いの相互作用によって着目した振舞い以外を隠蔽することが可能である．このため，状態数が減少し，検証コストを下げることができる．ここで，着目する振舞い以外を隠蔽することは，振舞い近似手法における近似と似た考えであるが，近似手法の概念を用いて説明するためには，TSE に時間の概念を入れる必要がある．

我々が提案した手法は安全性の検証手法であり，[CK96, CGK97, iso02] と同様に検証に必要のない振舞いを取り除く．しかし，いずれも本研究のような属性については取り扱っておらず，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，本研究では，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．そしてまた，この手法と協調して，プログラム検証で用いられる Hoare の検証手法と同様の演繹的かつ構文主導の検証手法が適用できることを示した．

第 9章

まとめ

本論文では，ステートチャートに関する不変性の検証を行う形式的な手法を提案した．提案した手法では，ステートチャートを TSE に変換し，表明を与えて構文主導の演繹的な証明を行うものである．このため，ステートチャートの属性値が無限の値をとり得る場合にも有効である．ここで用いた演繹体系は，健全かつ相対完全である．また，我々が導入した表明付き TSE の意味論は，非決定的なステートチャートも扱える．

一方で，システム開発では，システム全体の振舞いを複数のステートチャートの相互作用によって記述することが一般的に行われている．このよに，ステートチャートを分けることによって，それぞれのステートチャートは見通しの良いものとなる．しかし，検証に関しては，個々のステートチャートの振舞いだけでは非常に狭い範囲の性質しか検証を行えない．詳細な検証を行うためには，ステートチャートの相互作用を考慮して検証しなければならない．このための直接的な方法は，イベント通信方法を考慮して，互いのステートチャートの直積を構成することであるが，状態爆発という問題が生じる．そこで，互いに振舞いが依存する複数のステートチャートに対しては，互いに繰返し振舞い近似手法を行うことによって振舞いを徐々に制限する手法を提案した．この近似のために TSE に対する近似関数を用いる．本論文では，この近似関数とイベント通信方法の関係を形式的に表し，近似関数が満たすべき制約を与えた．このため，近似を行うための近似関数は，制約を満たす限り検証者が自由に定義できる．ま

た，近似関数を相互に繰返し適用することができる．ここで，近似関数の定義は，与えられたイベント通信方法との関係付けによって与えた．このため，様々なイベント通信方法のモデルに本研究の手法を適用することができる．

近似関数の例として，同期イベント通信と非同期イベント通信に対する近似関数 γ_a を示した．そして，この近似関数を用いて非常に簡単化した DHCP サーバ・クライアントのモデルに適用し検証を行った．簡単化しなければならなかったのは DHCP のプロトコルではメッセージに情報を付加してサーバとクライアント間で通信しなければならないためである．このように現実問題へ適用するためにはイベント属性を扱えるようにすることが重要な課題である．また，同じ振舞いを持つオブジェクトを効率よく扱う方法も今後の課題である．

謝辞

本研究を行うに当たり終始御指導賜わった片山卓也教授に深く感謝致します．また，日頃から有益な助言を頂いた青木利晃助手に感謝致します．ソフトウェア基礎講座の諸兄には，ゼミの活動など広い範囲に渡り情報交換や議論の相手をして頂けたことに感謝します．

参考文献

[CGK97] S. C. Cheung, D. Giannakopoulou, and J. Kramer. Verification of liveness properties using compositional reachability analysis. *ESEC/FSE 97*, 1997.

[CK96] S. C. Cheung and J. Kramer. Checking subsystem safety properties in compositional reachability analysis. *18th International Conference on Software Engineering*, 1996.

[Dro97] R. Droms. RFC2131: Dynamic Host Configuration Protocol. 1997.

[EOD00] E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, Vol. 8, No. 3, pp. 231–274, June 1987.

[HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory*. Addison-Wesley, 2000.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes, International Series in Computer Science*. Prentice-Hall, 1985.

[iso02] 並行合成エージェントのための真の並行性を考慮した時間付プロセス代数とプロセス論理. 日本ソフトウェア科学会 レクチャーノート FOSE2002, pp. 119–130, 2002.

[Lee94] Jan Van Leeuwen, editor. *Handbook of Theoretical Computer Sceience, Vol.B: Formal Models and Semantics*. MIT Press, 1994.

[Mil99] Robin Milner. *communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[OMG00] OMG. *Unified Modeling Language v1.3*. OMG, 2000.

[SD01] Graeme Smith and John Derrick. Specification, refinement and verification of concurrent systems – an integration of object-z and csp. *Formal Methods in System Design*, pp. 249–284, 2001.

[SH96] S. Graf and H. Saidi. Verifying invariants using theorem proving. In *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, Vol. 1102, pp. 196–207. Springer Verlag, / 1996.

[VM92] Vijay K. Garg and M. T. Ragunath. Concurrent regular expressions and their relationship to petri nets. *Theoretical Computer Science*, Vol. 96, pp. 285–304, 1992.

[青木 01] 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.

本研究に関する発表論文

[1] Takaaki Tateishi, Toshiaki Aoki, Katayama Takuya: “An Axiomatic System for Verifying Object-Oriented Analysis Model”, SCI/ISAS, pp.662–667, 2000

[2] Takaaki Tateishi, Toshiaki Aoki, Katayama Takuya: “Successive Behavior Approximation Method for Verifying Distributed Objects”，PDCAT Proceeding, pp.439–446, 2002

[3] Takaaki Tateishi, Toshiaki Aoki, Katayama Takuya: “Suppressing State Explosion for Verifying Statecharts”，FASE, 2003 投稿中

[4] 立石 孝彰，青木 利晃，片山 卓也: “HOL を用いたオブジェクト指向分析モデルの検証”, 日本ソフトウェア科学会 FOSE 2000, pp.117–124, 2000

[5] 立石 孝彰，青木 利晃，片山 卓也: “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”，情報処理学会 「オブジェクト指向技術特集」論文誌 投稿中.

[6] 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 コンピュータソフトウェア, Vol18 No.4, pp.18–47, 2001.