高信頼性ソフトウェアシステムの実現のためのモデル修正技術に関する研究

熊澤努

目次

図目次

表目次

第 1 章

はじめに

近年，ソフトウェアシステムが多くの製品に組込まれるようになり，あるいはサービスとして提供されるようになった結果，ソフトウェアがわれわれの身近な存在となってきた．しかしながら，現在のソフトウェアシステムは，旧来指摘されている大規模化，複雑化だけでなく，開放化，分散化など多様化も進んでいる．それゆえ，顧客からの多様な要求に応えつつシステムの信頼性を確保することは容易ではない．実際，不具合を含むシステムによって，開発企業やその顧客のみならず社会的に深刻な影響を与える事例が数多く報告されている．したがって，信頼性の高いシステムをいかに社会に提供するかが重要な課題である．

信頼性の高いシステムを実現するために，システムの品質の向上を図る開発方法論がこれまで数多く提案されてきた．その 1 つである開発プロセスは，複数の工程からなる標準的なシステム開発手順を定めたものである．中でも多くの企業の標準工程の基盤となっている開発プロセスは，落水型開発プロセスと呼ばれる．これは，システムの開発運用過程を，要求分析，設計，実装，テスト，運用および保守の各工程に分解し，順に実施することで信頼性を確保すること目指す開発プロセスである．

もう 1 つのモデル化技術は，要求分析，設計工程において開発者が実装する前にシステムのあるべき姿を理解するために，システムを抽象化したモデルを作成する技術のことである．モデルは，システムの構造を表すクラス図などの静的なモデルと，システムの振る舞いを表す状態遷移機械などの動的なモデルに大きく分類される．現在主流となっているシステムは，外部からの入力に対して所定の動作をする応答型システムであり，このようなシステムの振る舞いの記述には動的なモデルが有効である．特に，複数の状態遷移プロセスが相互作用しながら計算を行う並行システムの場合は，しばしばシステム全体の振る舞いが複雑となるため，動的なモデルによるシステムの振る舞いの分析が必要である．

落水型プロセスでは，各工程間で手戻りが発生しないことが前提なので，要求分析，設計工程で作成したモデルが所望の性質や仕様を満たしていなければ，それらを元に開発したシステムもまた誤りを含む．このような誤りは，システムの運用時に重大な障害や不具合を引き起こすことがあるので，システムの信頼性を損なう要因となり，開発の手戻りを引き起こすこととなる．そのため，システム開発の要求分析，設計段階で，モデルの正しさを保証することがシステムの高い信頼性を確保するために必要である．

本論文では，モデルの正しさを検証するための技術の 1 つとして盛んに研究されているモデル検査技術に注目する．モデル検査技術は，モデルがシステムに要求される性質や仕様を満たすことを形式的に検証する技術である．モデル検査技術は，数学的にシステムの正しさを保証することを目標としているが，最大の特徴は検証手順を計算機によって完全に自動化できる点にある．しかし，既存のモデル検査技術は，モデルが性質を満たさない場合，モデルの誤りのある箇所やそれを正す方法を直接的には提供せず，開発者に対する支援が不十分である．それゆえ，開発者が正しいモデルを得るためには，手作業で誤った箇所を見出して，モデルの修正を行わなければならない．開発対象システムの振る舞いが複雑な場合には，この作業は開発者にとって大きな負担となり，信頼性の高いソフトウェアを開発する際の障害となる．

本論文においてわれわれは，高信頼性システムを実現する開発技術の確立に貢献するため，モデル検査の結果に基づくモデル修正技術を提案する．本論文では，特に要求分析，設計段階でモデル検査技術が用いられる状況を想定し，状態遷移機械により作るべきシステムの振る舞いが関連する事象に基づいて記述されていると仮定する．この前提の下で提案するモデル修正技術は，反復的なモデル修正法とモデルに含まれる誤りの特定手法である．モデル修正法は，対象領域についての知識を用いて，反例の実行が禁止され，それ以外の軌跡が実行可能なモデルを構成することで，開発者に修正モデルの候補を提示する．したがって，この手法は開発者との対話的な修正が可能であり，検証を行う様々な仕様や要求を定式化した性質に適用できる．

モデルの誤りを特定する手法は，モデル修正法を補完する方法であり，モデルにおいて性質違反となる原因箇所を発見する．提案する手法は，モデル修正法と同様に広範な種類の性質に対して適用できる．加えて，有向グラフ上の最短路探索アルゴリズムを用いることで，計算機により自動化が実現できる．

本章では，導入として，まず，現在に至るソフトウェア開発をめぐる背景および問題点について述べ，問題点の解決のために提案されてきた開発方法論が果たしてきた役割を論じる．1.1 節において，ソフトウェアシステム開発を成功させるために，ソフトウェア工学が従来行ってきた取り組みを簡単に振り返る．また，特に信頼性の観点から，現在の課題について概観する．1.2 節において，現在のソフトウェア開発プロジェクトで広く採用されている開発方法論である開発プロセスとモデル化技術について紹介する．続いて，1.3 節で，システムの信頼性確保のために行われるシステムやモデルの検証技術について説明する．検証技術の中でもモデル検査技術は，ソフトウェアツールとして実用化されるなど，現在活発に研究開発が行われている．加えて，1.3 節で，モデルの修正技術の確立というモデル検査技術の課題を提示して，その課題を解決することの重要性について議論する．そして，1.4 節において，本論文で提案するモデル修正技術の概要，提示した課題に対して果す提案手法の貢献，ならびに本論文の構成を述べる．

1.1　ソフトウェアシステム開発方法論と信頼性

1960 年代に叫ばれた「ソフトウェア危機」以来，ソフトウェアをどのようにして開発すべきかが，現在にいたるまでソフトウェア工学が取組んできた大きな課題である．ソフトウェア危機とは，以下に述べる当時のソフトウェア開発プロジェクトの実態を指す言葉として使われた．すなわち，要求されるソフトウェアの開発が納期どおりに行われず遅延を繰返し，それゆえ開発予算を超過してしまう．加えて，遅延した開発工程を少しでも早く終えるために，その品質が犠牲となっていたという状況である．この要因には，ソフトウェアシステムが原始コード行数で数百万行に達したというソフトウェアの複雑化，大規模化が挙げられる．このような問題は，単に開発プロジェクトへの人員の追加などの組織的な対処によって解決されるものではない．Brooks は，遅延の発生した開発プロジェクトへの人員の投入は, 更なる遅延をもたらすと論じたことで，問題の解決の難しさを明らかにした [14].

以来, ソフトウェア工学は, ソフトウェア危機を克服するための方法論を提唱してきた [112]. 1970 年代には，構造化プログラミングから続く構造化分析，設計方法論が提唱された [117]. その中の代表的なものには，DeMarco の構造化分析法がある [113]．これは，プロセスや外部実体，ファイルの間を結ぶデータの入出力関係を記述するデータの流れ図を用いて開発対象システムの分析を行う方法論で，現在でも広く活用されている．

1980 年代になると，プロジェクト管理や構成管理などの管理技術が注目された．また，オブジェクト指向プログラミング言語の研究開発が活発に行われ，そこで提唱された概念を用いた開発手法であるオブジェクト指向開発手法が 1980 年代の終わりから 1990 年代にかけて盛んに研究された．このような方法論には，例えば Jacobson による OOSE がある [118]．オブジェクト指向開発方法論は，その後も統合ソフトウェア開発プロセス [59] や統合モデル化言語（UML） [42, 111] として方法論の統一化が行われた．オブジェクト指向開発方法論では，開発対象のソフトウェアに関わる実体や個体，もしくは，ひとまとまりの抽象概念を，オブジェクトという概念として把握する．共通するオブジェクトの集合はクラスと呼ばれ，クラス間の関連をモデルとして記述する．関連はクラス間の概念的な関係であり，特別な関連として汎化（継承），集約がある．一方で，クラスやオブジェクトの振る舞いを状態遷移機械などにより記述することも可能である．オブジェクト指向開発方法論は，ユースケースと呼ばれる自然言語を用いたシステム記述を奨励し，また，図式を用いた直観的な表記法を提供しているので，現在も産業界において広く使われる方法論である．以上のようなソフトウェア開発方法論は，特に大規模システムの開発に対してはある程度の成功を収めてきた．

1990 年代以降，ソフトウェアシステムが多くの製品に組込まれるようになり，また，サービスとして広く社会に提供されるようになってきた．その結果，ソフトウェアがわれわれの身近な存在となっている．例えば，家庭用電化製品に代表される様々な電気機器や自動車などの機械製品には，組込みシステムという形でソフトウェアが内在するようになっている．一方で，インターネットの普及に伴い，インターネットを利用した商取引やサービスも普及している．加えて，オープンソースソフトウェアの開発も数多く行われるなど，多様なソフトウェア

システムが作成されている．このように，現在のソフトウェアシステムは，旧来指摘されている大規模化，複雑化だけでなく，開放化，分散化，小型化も進んでいるといえる．このように多様化するソフトウェアシステムにおいて，開発するシステムの信頼性の確保が大きな課題である．実際，不具合を含むシステムの事例が数多く報告されており，開発運用する企業のみならず，社会的に深刻な影響を与えることも少なくない [75, 73, 57, 95].

ソフトウェアシステムの信頼性という用語は，英語の reliability や dependability の訳であり，しばしば複数の意味で使われる．例えば，文献 [72] で Leveson は，reliability とは，「指定された外部環境の条件下において，指示された時間で，（システムを構成する）設備や部品の一部が十全に意図された機能を実行する確率」と定義している．一方，dependability は，同文献では reliability に複数の性質を加えた複合的な測定基準を指す言葉として定められている．これらの定義は，組込みシステムのように，システムの応答時間が重要なシステムには妥当であるが，例えば，Web サービスのような精確な応答時間が必ずしも要請されるわけではないようなシステムにおいては，必ずしも適当ではない．

Daniel Jackson が定めた信頼性（dependability）とは，次のようなものである [57]．システムが特定の障害を発生することのないまま特定の処理を実行するとき，そのシステムは信頼性が高いという．ここでは，単に欠陥や不具合，ならびにそれらが原因で生じる障害が存在しないことのみならず，障害が発生しないことを示す証拠が存在することが重要である．Jackson は，医療用システムなど故障の発生が人命に関わるようなシステムの事例から，システムを開発する過程でその信頼性を確保していくことの重要性を主張している．よって，開発時に障害が起こらないことを示すことが求められるように信頼性が定義されている．

以上の議論を基本として，本論文では，システムの信頼性の高さを以下のように考えることにしよう．

> システムに欠陥が存在することなく，その結果，障害が発生せずにシステムが要求された機能を実行するとき，そのシステムの信頼性が高いという．

ソフトウェアシステムの信頼性の確保は，これまで提唱されてきた開発方法論を用いて開発を行ったとしても達成するのが難しい．現在，最も流通しているソフトウェアシステムの 1 つは，組込みシステムである．組込みシステムは，複数のプロセスや実体が協調して動作する並行システムであることが多い．並行システムを構成する個々のプロセスが単純な計算のみを行う場合であっても，全てのプロセスが協調動作した結果示すシステムの振る舞いは極めて複雑なものとなる．このようなシステムの振る舞いは，並行性に起因する特有の問題を引き起こす恐れがある．例えば，所望の処理の実行が完了する前に各プロセスの実行が停止してしまうデッドロックの発生は，並行システムに発生する典型的な信頼性を損なう問題点の 1 つである．このシステムの信頼性を高めるためには，プロセス単体のみならず，システム全体の振る舞いを開発者が理解していることが重要である．

組込みシステムにおける低い信頼性が引き起こした不具合の事例として，1990 年代に発生したアメリカとヨーロッパの宇宙船舶事故が挙げられる．Leveson は文献 [73] で，5 件の宇宙船舶事故の原因を調査し，特にソフトウェアに起因する問題点について考察した．そして，宇

宙船舶に組込まれたソフトウェアの安全性分析が有効に行われず，したがって，開発者がソフトウェアが本来なすべきではない振る舞いに注意を払わなかったことが原因の 1 つであると主張している．

また，1990 年から 2000 年までの 10 年間におけるペースメーカと除細動器のリコール原因の調査では，累計約 52 万台であったリコール数の内，ソフトウェアが原因のリコールは約 4 割の約 22 万台に上ったという結果が報告されている [75].

2005 年に発生したみずほ証券株式会社による誤発注によって明らかになった東京証券取引所の有価証券取引システムの障害は，商業用システムの信頼性が損なわれた事例である [95]. この障害によるみずほ証券の損害額は 400 億円に達したという．玉井の報告によると，ソフトウェアの設計や仕様書，テストの不備がこの障害を引き起こした要因の一部に挙げられている．

以上で例示したように，顧客からの多様な要求に応えつつシステムの信頼性を確保することは，現代においても容易ではない．このことは，今もなお「ソフトウェアの危機」が去っていないこと意味している．そこで，本論文は，ソフトウェアシステムの信頼性の確保するための技術を開発することで，この課題の解決に貢献することを目標とする．

1.2　信頼性確保の手段としてのソフトウェア開発プロセスとモデル化技術

前節で述べた背景の下で信頼性の高いシステムを実現するために，システム開発時に開発者は複数の手段を組合せることが多い．そのような手段の 1 つは，標準的なシステム開発手順を定めた開発プロセスの採用であり，もう 1 つは，モデルを用いたシステムの分析，設計である．モデルとは，プログラムの実装上の制約などの微細な事柄を捨象して，人間がシステムを理解して分析できるように本質的に重要な側面を明らかにした，対象システムの抽象的な記述を指す用語である．

1.2.1　ソフトウェア開発プロセス

現在最もよく知られており，同時に多くの企業の標準工程の基盤となっている開発プロセスは，落水型開発プロセスと呼ばれる [89]．実際，独立行政法人 情報処理推進機構 ソフトウェアエンジニアリングセンターが行った調査によると，2005 から 2007 年度にソフトウェア開発企業で行われた 1959 の開発プロジェクトのうち，約 96% が落水型プロセスを採用している [114]．落水型プロセスは，図 1.1 に示すように，システムの開発運用過程を，要求分析，設計，プログラムの実装，テスト，運用および保守の各工程に分解し，各工程を順に実施することで信頼性を確保することを目指す開発プロセスである．以下，特に，要求分析，設計，プログラミング（実装），テストの各工程について簡単に説明する．

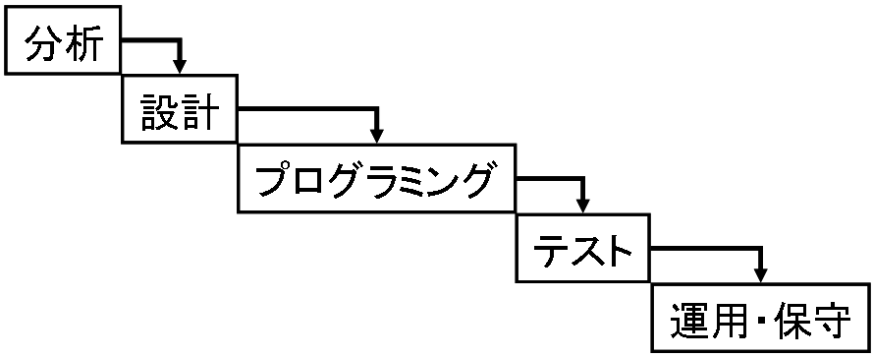


図 1.1. 落水型開発プロセス（文献 [111] より引用）

要求分析

要求分析は，これから作るべきシステムは何か，また，システムに望まれる事柄は何かを明確にし，仕様として記述する工程である．要求を開発者が容易に理解し，分析できるようにするために，要求はモデルを用いて記述されることが多い．加えて，ソフトウェアが扱う対象領域に課せられる制約や想定事項を明らかにしていく作業も行われる．文献 [111] は，要求分析の手法を以下のように 3 種類に分類して考察している．

- **要求抽出型**：ソフトウェアシステムに対して潜在的な要求を持つ利用者へのインタビュー等を通して，システムに対する要求を記述，分析する方法である．構築しようとしているシステムとユーザとの関係を，両者の相互作用として記述するシナリオ（またはユースケース）を利用することが多い．UML には，シナリオやユースケースに参加するシステムと利用者との関係を図式化するユースケース図が用意されており，要求抽出を支援している．
- **目標指向型**：システムが達成すべき目標を明確にして分析することで，システムの要求を求める方法である．代表的な方法は，Van Lamsweerde らにより研究されている目標指向要求工学 KAOS [100] である．KAOS は，開発すべきシステムなどの実体についての望ましい目標から，系統的に上位あるいは下位の粒度の目標を導出することで，システムに対する要求を明らかにする手法である．また，目標間の衝突分析手法や非機能的な目標の抽出手法を提供するなど，様々な視点から要求を分析することができる．
- **領域モデル型**：システムとそれと関係する外部の環境からなる対象領域を抽象化し，記述する方法である．Jackson の問題フレーム [58] がその例として挙げられよう．問題フレームは，システムに関連する領域の間の関係を，共有する現象を元に記述するための記法を提供する．この記法を基に作成したシステムの全体文脈図や問題図を通じて，ソフトウェアによって解決すべき問題を記述，分析する．それによって，開発者は問題解決の観点からソフトウェアに対する要求を明らかにすることができる．

設計

要求仕様が完成したら，それを実現するソフトウェアの構造や振る舞いを設計し，同様に仕様として記述する．この工程では，システムを図式化することで，システムをいかにして作るべきかを明らかにする作業に取組むことが多い．そのために，要求分析工程と同様にモデルがしばしば用いられる．作成されるモデルには，システムの静的な構造を表現するためのもの，また，システムの振る舞いを表現するためのものがある．例えば，システムの構造を表現するための手段に，UML のクラス図がある．クラス図は，システムを構成する部品や実体の集合間の関連を図示する．一方，システムの振る舞いは，オートマトンに代表される状態遷移機械を用いてモデル化される．また，シナリオにより実体間の相互作用を記述することもある．UML にはシナリオを記述する記法に系列図が用意されている．シナリオは特に重要なシステムの実行系列を記述したもので，直観的に分かりやすいので，状態遷移機械を補完する手法と考えられる．要求分析と設計工程は，開発すべきソフトウェアを構築する前の準備段階と位置づけることができ，上流工程といわれる．

プログラミング

設計仕様を基に実際にプログラムを実装することで，システムを構築する．プログラミングは，仕様に記された機能やシステムの構造，振る舞いをソフトウェアとして実現することである．要求分析や設計工程において誤りの混入や記述漏れがあると，それを基に作成したプログラムにも対応する誤りや漏れが発生する．したがって，要求分析，設計工程で作成した仕様が，開発すべきシステムを正しく記述していることが重要である．

テスト

テストは，完成したシステムが仕様に従った振る舞いを行うかを確認する工程である．具体的には，プログラムを既知の条件の下で実行し，結果が予め想定されたものと一致するか確認する作業を行う．入力と想定される結果の組をテストケースという．システムが用意された全てのテストケースに合格したならば，実際にシステムを運用することができる．不合格となるテストケースがあった場合は，作成した仕様あるいはプログラムに誤りが含まれていると考えられる．よって，以前の工程に戻って正しい仕様やプログラムを作成する必要がある．テストは，後述する検証作業の一部として位置づけることもできる．

落水型開発プロセスの前提は，次の 2 点であるといわれる．

- 工程ごとに文書化された成果物（仕様書，プログラム）を作成し，次の工程への入力とする．
- 工程間の手戻りが発生しない．

後者の手戻りの発生は，システムの開発期間の増加，ひいては開発費用の増加を引き起こす．ゆえに，落水型開発プロセスを成功させるための条件は，工程間の手戻りが発生しないことである．手戻りの発生を防ぐためには，成果物に曖昧な記述や，矛盾した記述，記述漏れ，そし

て記述の誤りがないこと，つまり，成果物が正しく作成されていることが必要である．特に，上流工程においては，仕様書が以降のプログラミングやテスト工程に入力する成果物となる．よって，仕様書が正しく対象システムを記述しているときに限り，開発者はそれらを元にプログラムを正しく構築することができる．正しい仕様書が入力されないときには，落水型開発プロセスに従って作られたシステムは誤りを含み，信頼性を脅かす恐れが大きい．その場合，正しい仕様を作成しなければならないので，開発工程の手戻りが発生する．しかしながら，手戻りによって開発工期が増大することになり，その結果ますますシステムの信頼性を高めることが困難になってしまう．

このような批判を受けて，落水型に変わる様々な開発プロセスが提案されている．例えば，極端プログラミング [9] は，逐次進化型と呼ばれる開発プロセスに分類される．極端プログラミングは，システム開発の一連の工程を短い期間で繰返し実行し，システム公開を断続的に行う．加えて，現在明らかになっている要求を優先的に実現することで，工程間の手戻りによって引き起こされる問題を防止する開発プロセスである．しかしながら，ソフトウェア開発プロジェクトの工程管理において落水型とは異なる考え方を採用しているので，考え方の組織的な転換が求められる．それゆえ，管理上都合のよい落水型が依然として最もよく使われている [111]．また，極端プログラミングを利用したとしても，仕様書などの成果物の正しさが保証されるわけではない．すなわち，誤った仕様によって開発が行われるならば，その結果実現されるシステムの信頼性は望ましい水準に達することはない．したがって，信頼性に関わる落水型開発プロセスの問題が，逐次進化型などの他の開発プロセスの採用のみによって解決するわけではない．

1.2.2　モデル化技術

先に述べた要求分析，設計工程において，システムを抽象化したモデルを用いることが多い．モデルを活用する目的は，開発者がモデルによってシステムの特に注目すべき点を記述して，実装する前にシステムのあるべき姿を理解することにある．これまで数多くのモデルの記法が提案されてきたが，それらは主にシステムの静的な側面，あるいは動的な側面を記述することを目的している．このような方法で記述されたモデルは，大きく静的なモデルと動的なモデルに分類される [111]．

静的なモデル

静的なモデルはシステムの構造を表すモデルであり，有向あるいは無向グラフによって表現されることが多い．静的なモデルに使われる記法の代表的な例には，実体関連図，UML におけるクラス図やオブジェクト図がある．これらの図のグラフの頂点は，システムを構成する部品や，システムの利用者などの外部実体である．そして，頂点同士を結ぶ辺は，頂点同士に何らかの関係があることを表す．例えば，クラス図の場合，頂点は属性とメソッドを持つクラスであり，辺はクラス間の汎化，集約などの関連を表す．

動的なモデル

動的なモデルは，システムの振る舞いを表現する際に用いられる．各種オートマトンに代表される状態遷移機械が，動的なモデルを記述するための記法の例に挙げられる．動的なモデルにおいて注目する振る舞いを視覚化するために，静的なモデルと同様にグラフ，特に有向グラフがしばしば用いられる．辺は，頂点からもう一つの頂点への何らかの流れや移動を表す．状態遷移機械ならば，頂点はシステムが実行時に持ちうる計算の状態や特定の時点（期間）を表す．一方，辺は 1 単位時間，もしくは時間が明確に定まっていない場合は，1 ステップのシステムの実行を表すと解釈されることが多い．電気回路図のように，流れるものが物理量である場合もあるが，ソフトウェア開発においては，より抽象的なデータの流れやシステムの実行，計算を表すモデルが用いられる．現在主流となっているシステムは，外部からの入力に対して所定の動作をする応答型システムであり，このようなシステムの振る舞いの記述には動的なモデルが有効である．特に，複数の状態遷移プロセスが相互作用しながら計算を行う並行システムの場合は，しばしばシステム全体の振る舞いが複雑となるため，予期しない動作が行われないように，動的なモデルによるシステムの振る舞いの記述が必要である．

動的なモデルの記法は多く研究されている．例えば，Hoare が提唱した CSP [53] や，Milner による CCS [78] および π 計算 [79] は，プロセス代数と呼ばれ，システムを構成する各プロセスの振る舞いを代数的に記述することができる．プロセス代数は，プロセス間の並行性と同期実行も考慮された計算モデルであることが多い．また，状態遷移機械を図式表現したものとして，Statecharts [50] が広く知られている．Statecharts は，複数の状態をまとめて記述する階層型記法や，並行動作を記述するための記法を導入している．階層化は大規模なシステムの振る舞いを簡潔に表現できるので，実用的な並行システムの記述に適した手法である．また，UML はオブジェクトの振る舞いを記述するために，Statecharts の記法を採用している [42, 111]．本論文では，動的なモデルの一種で，発生する事象によってシステムの振る舞いを表現する状態遷移機械であるラベル付き遷移系 [74] を対象とする．

落水型プロセスでは，各工程間で手戻りが発生しないことが前提なので，要求分析，設計工程で作成したモデルが所望の性質や仕様を満たしていなければ，それらを元に開発したシステムもまた誤りを含む．このような誤りは，システムの運用時に重大な障害や不具合を引き起こすことがあるので，システムの信頼性を損なう要因となり，開発の手戻りを引き起こすこととなる．また，テスト工程でシステムに誤りが発見されたときにも，既に述べたように前工程への手戻りが発生する．実際，システムのテスト時に発見された不具合が，要求，設計の問題に起因することも多い．そのため，システム開発の要求分析，設計段階で，作成したモデルの正しさを保証することがシステムの高い信頼性を確保するために必要である．ここで，モデルやプログラム，システムの正しさとは，各々が開発者や顧客などの利害関係者がシステムに求める要求や性質を満たすことをいう．

1.3　仕様とプログラムの検証

プログラムやモデルが要求を満たし（すなわち，正しく動作し），品質目標を達成していることを確認するために，対象のプログラムやモデルに対する検証が行われることが多い．プログラムの検証の代表的な手段は，図 1.1 にも示したテストの実施である．しかしながら，多くのシステムは自然数や実数などの型を持つデータを扱うため非常に多くの入力の組合せがあり，システムへのあらゆる入力に対するテストケースを用意するのは現実的ではない．また，プログラムがユーザからの入力に応じた振る舞いを行う場合や，プログラムに条件分岐や繰返し構造がある場合には，その実行の組合せもまた膨大な数となってしまう．この場合，プログラムの全ての実行の組合せをテストにより調べることは通常不可能である．よって，テストによる検証では，実行の標本を抽出して調べることでしか，プログラムの正しさを確認する作業が事実上実施できないという問題点がある．

この問題点を解決するための手段として，ソフトウェアや仕様の形式的検証技術がこれまで研究されてきた．正当性証明技法は，プログラムの動作が仕様が定める動作と一致することを数学的に証明する方法であり，1970 年代から研究が進められてきた．正当性証明技法の中で，最もよく知られているのは Hoare 論理を用いた証明である [52, 116]．Hoare 論理では，プログラムや仕様の一部を事前条件と事後条件を用いて記述する．また，繰返しがある場合は，繰返しに関する不変表明を記述する．続けて，プログラムの開始時の事前条件から，プログラム終了後の事後条件を，公理系によって導出する．その際，対象プログラムの実行経路についての条件や表明を中間的に用いることもある．この証明を人手で行うのは一般に困難であるので，機械的に証明を実行することを可能にする定理証明系の開発も行われている（例えば，PVS [85]）．しかしながら，定理証明系を利用したとしても，効率的な証明戦略を人間が立てることで証明を支援する必要があるなど，完全な自動化は難しい．

仕様の検証に関しては，複数の開発関係者がプログラムや仕様書を調べる作業が行われることも多い．このような検証作業には，査閲，見直し，徒歩検査がある [111]．これらは，それぞれインスペクション，レビュー，ウォークスルーとも呼ばれる．査閲は，通常 4 人程度の小規模のチームを組織して，対象となるプログラムや仕様の誤りを発見する作業である．一方，徒歩検査は，プログラムの原始コード上やモデル上で，実行列を模擬的に追跡しながら誤りを発見する．見直しは，査閲や徒歩検査と同一視されることが多い．このような方法は，テストと同様，仕様やプログラムの全ての詳細や実行を検査することが難しい．また，人手で実施するので，抽出する誤りに漏れが発生する恐れがある．

以上の技術が持つ問題点は，ソフトウェアやモデルの網羅的な検証を自動的に行うのが困難であるということである．近年，この問題点を解決するために，形式的な検証技術の 1 つであるモデル検査技術 [25] が注目されている．モデル検査は，システムに要求される性質をシステムの仕様やプログラムを記述したモデルが満たすことを，形式的に検証する手続きである．検査を行うモデルの記述には，ラベル付き遷移系やクリプキ構造などの有限状態遷移機械を用いることが多い．一方，性質の記述には時相論理などの論理式がよく用いられる．既に述べた

通り，状態遷移機械はシステムの振る舞いを表現するのに適しているので，並行システムについてのデッドロックの不在などの振る舞いに関する性質の検証に適している．モデル検査技術は，正当性証明技法と同様に，数学的にシステムの正しさを保証することを目標としているが，最大の特徴は検証手順を計算機によって完全に自動化できる点にある．実際，検査技術を自動化したモデル検査器が数多く開発されている [22, 8, 6, 55, 30, 74, 11]．Clarke は，モデル検査技術を利用する利点として以下の項目を挙げている [23].

- 証明の構築が不要であり，機械による完全な自動化が可能である．
- 定理証明系を使った検証に比べて高速に実行できる．
- モデルが性質を満たさない場合には，なぜ性質違反なのかを示す反例を提示する．対象モデルやプログラムの修正には，開発者は反例を手掛かりとすることができる．
- システムの部分的な記述に対しても適用可能な方法である．対象システムのプログラムの完全な記述が必要ないので，システムの設計工程においてもモデル検査技術を用いることができる．
- 時相論理は，並行システムにおいて検証される性質の多くを容易に表現可能である．このような性質は，手作業での検証では困難である．

モデル検査は Clarke 等によって 1980 年代に本格的に研究がはじめられた [26]．モデル検査技術は，既に述べた複数のプロセスからなる応答型並行システムが引き起こす問題を解決するために開発された．当初は，ハードウェアの論理回路や通信プロトコルの検証に応用されてきたが，近年はソフトウェアシステムやそれらを抽象化したモデルの検証にも使用用途が広げられてきている [104, 81, 20, 115, 110, 77]．システム開発における要求分析，設計工程においては，システムの仕様を記述したモデルの有効性確認や検証に用いられる [65]．また，作成したプログラムの検証については，例えば，Java の Enterprise Java Beans アーキテクチャの検証を，モデル検査器を用いて行った研究がある [81]．加えて，テスト工程でもモデル検査技術を用いたテストケース生成法が提案されるなど [3]，モデル検査技術はソフトウェア開発工程の各段階で広く利用されるようになってきている．

現在に至るまで，モデル検査技術に関する研究には，様々な検証手法の開発に加えて，検査の効率化の実現やモデルの表現力の向上させる試みがある．Clarke は，それらを以下のようにまとめている [23, 27].

- クリプキ構造やラベル付き遷移系などの状態遷移機械に対する古典的な検証技術の開発 [26, 101, 28, 25, 45].
- モデルの抽象化法などの状態空間爆発問題を回避する手段の開発 [15, 16, 24, 21].
- 有界モデル検査器の開発など SAT 求解器をモデル検査器に導入する研究 [13].
- 古典的な検証技術が扱う記法より表現力の高い時間オートマトンなどの記法で記述されたモデルや，プログラムに対する検証技術の研究 [25, 8, 6, 30, 11, 5].

モデル検査技術の利点で挙げた通り，既存のモデル検査器は，モデルが性質を満たさない場合には，性質違反を表すモデルの事象列（軌跡）である反例を出力する．この場合，モデルに

誤りがあると考えられる．特に，モデルを扱うことが多いのは，開発プロセスにおける上流工程，すなわち要求分析や設計工程であり，これらの工程においては，先に述べた通り正しいモデルを得る必要がある．そのためには，開発者は性質を満たすようにモデルを修正する作業を実行しなければならない．しかし，反例は，モデルの誤りのある箇所やそれを正す方法を直接的には提供しないので，開発者は反例を手作業で分析して誤りを特定し，修正作業を行わなければならない．開発対象システムの振る舞いが複雑な場合には，この作業は開発者にとって大きな負担となり，信頼性の高いソフトウェアを開発する際の障害となる．加えて，モデル検査技術に関連する研究分野においては，モデル検査の結果を元に正しいモデルを得る手法についての研究は，あまり行われていない．それゆえ，高信頼ソフトウェアシステムの実現のために，モデル修正法の確立が望まれる．

Clarke は，将来のモデル検査技術で行われるべき研究課題の 1 つに，「(長い) 反例の解釈」を挙げている [23, 27]．Clarke は具体的な課題の内容を説明してはいないが，特に反例が長い場合には，それが何を意味するか開発者が理解，解釈することが困難であるので，それを支援する手法の確立を意味すると思われる．反例の解釈とは，反例が示す誤りを開発者が理解することであり，モデルの誤りを発見することに深い関連がある．上で論じたように，モデルの誤りを特定する作業が重要となる工程は，モデルを用いてシステムを記述する要求分析，設計工程である．本論文では，要求分析，設計工程でモデル検査技術がモデルの検証，有効性確認に使われることを想定する．そして，Clarke が提示した課題を基に，モデル検査の結果得られる反例を用いたモデルの修正問題の解決を目指す．

1.4　論文の構成

本章でこれまで述べたように，われわれは，特に要求分析，設計段階でモデル検査が用いられる状況を想定し，モデル検査の結果に基づくモデル修正法の確立を研究の目標とする．加えて，計算機による自動化を実現することで，開発者によるモデル修正の支援を目指す．

2 章では，本論文で前提とするモデル検査技術について説明する．簡単に紹介したように，モデル検査技術は，システムの振る舞いを表現する動的なモデルが時相論理式で記述された所望の性質を満たすかどうかを形式的に検証する手法である [26]．本研究では，システムの振る舞いを事象に基づいて表現する状態遷移機械の 1 つであるラベル付き遷移系（Labeled Transition Systems: LTS） [74] で記述されたモデルを対象とする．このような事象に基づいたモデルは，要求分析，設計工程において作るべきシステムの動作を表現するために広く用いられる．LTS に対するモデル検査手法には，線形時相論理（Linear Temporal Logic: LTL），特に流動 LTL（Fluent LTL: FLTL）で記述された性質の検証技術が提案されている [45]．この技術は，LTSA と呼ばれるソフトウェアツールとして実用化されている [74]．FLTL の意味論は LTS の軌跡上で定められ，システムの振る舞いについての性質や仕様の記述に有用である．

文献 [45] で提案された検証手法は，クリプキ構造に対する LTL 式の古典的なモデル検査技術の一つである Büchi オートマトンを用いる方法 [101] を拡張した手法である．検証は，

FLTL 式の否定を受理する Büchi オートマトンとシステムのモデルとの積オートマトン上で受理する言語の存在を調べることで実施される．モデル検査器は，積オートマトンに受理する言語が存在しない場合にはモデルが性質を満たすと判定し，存在する場合には性質を満たさないと判定してモデル上の性質違反の軌跡である反例を求める．FLTL 式で表される性質は一般に，システムにとって望ましくない事象が決して起こらないことを表す安全性と，望ましいことがいつか起こることを表す活性に分類される．既存のモデル検査器は，活性に対して無限回反復をする閉路とそこへの有限長の接頭辞からなる無限長の軌跡を反例として出力するが，安全性に対しては有限長の反例を出力する．

3 章から 5 章では，われわれが提案するモデル修正法とモデルに含まれる誤りの特定手法を説明する．先行研究の多くは反例が有限長の安全性のみを対象としているが，われわれが提案する手法は，安全性に加えて活性にまで広範囲に適用ができる．加えて，対象モデルにおいて公平性を仮定するモデル検査に対しても適用可能である．

3 章で，提案するモデル修正法について説明する．この手法は，対象領域についての知識を用いて，反例が禁止されたモデルを構成することで，開発者に修正モデルの候補を提示する．領域知識は，修正を行う性質と同様に，FLTL 式で定式化された性質の集合である．ただし，修正対象モデルは，領域知識に含まれる全ての性質を満たす．この性質の集合と反例をモデルに変換して，両者を合併することで，修正モデルの候補を求める．このモデルは LTS の拡張であり，遷移関係にその遷移の発生しやすさの確度を追加したものである．合併したモデルにおいて，領域知識を満たす軌跡は生起しうる遷移により記述され，そこに現れる反例は生起が禁止された遷移により表現される．ゆえに，合併モデルは領域知識から反例を除いた軌跡の集合を表すと解釈され，所望の性質を満たす軌跡もこのモデルに含まれる．そこで，開発者は，モデルに付加された確度を手掛かりに，反例に代わってシステムで起こるべき遷移を選択して修正作業を実施する．次に，修正したモデルが依然として誤りが含むかどうかを，モデル検査器を用いて検証する．モデルが性質を満たさない場合，モデル検査器から新たに得られる反例を用いて上記の手続きを再度実行する．モデルが性質を満たすまで反復的にこの工程を繰返すことで，開発者は修正作業を実施する．

本手法の利点は，まず，既存のモデル検査技術を直接利用できる点にある．先行研究の多くは反例が有限長の安全性のみを対象としているが，本手法は，安全性に加えて活性にまで広範囲に適用ができる．これは，本手法が反例に対する修正手法であることに起因する利点である．加えて，対象モデルにおいて公平性を仮定するモデル検査に対しても適用可能である．また，本手法のモデルの合併演算は自動化が可能であり，計算機による修正作業の支援ができる．そして，本手法では，修正時に対象領域の知識と開発者による決定工程を導入することで，対象領域や開発者にとって無意味なモデルを生成する恐れが少ない．最後に，対象システムの正しい振る舞いの事象列が予め与えられていることが，従来提案されてきた手法の多くの前提だが，本手法は対象領域の知識を用いるので，この制約がない．なお，この章で論じるモデル修正法は，文献 [66] に基づくが，この文献では FLTL ではなく，LTS の事象を命題とする LTL に基づいて説明している．本論文では，それを FLTL に拡張して論じるが，本質的な議論に変わりはない．

われわれは，電子レンジシステムの事例研究を行い，本手法を用いて所望の性質を満たすモデルを得ることで，その有効性を確認した．方法論の説明に続いて，われわれが行った事例研究について報告する．

続いて，提案するモデル修正手法の改善を図るために，修正対象モデルを効果的に活用する方法を議論する．そのアイデアは，モデル修正作業は修正対象モデルを元にして行われるので，修正の結果得られるモデルは修正対象モデルと類似しているという想定に基づく．われわれは，Sokolsky 等が提案した指標 [92] を用いた振る舞いに基づくモデル間の類似度を定義する．開発者が遷移を選択することで修正モデルを作成する作業を支援するために，修正対象モデルと合併モデルとの類似度を算出する．また，合併モデルから得られた修正モデルと修正対象モデルとの類似度も算出する．前者は，合併モデルと修正対象モデルとの状態間の対応付けを開発者に提供するので，開発者は領域知識から修正モデルを求める際に修正対象モデルの振る舞いを参照することができる．一方，後者は修正結果の妥当性を測定する指標の 1 つとして活用できる．われわれは，以上の類似度を活用したモデル修正法を整理して事例研究を行った．

しかしながら，開発したモデル修正法において，反例に含まれる性質違反の原因は開発者が発見しなければならない．よって，開発者によるモデル修正作業を支援するためには，モデル上の修正すべき誤りを開発者に提示する手法を確立することが望ましい．

そこで，4 章で，反例の修正候補となる軌跡を求め，モデルの誤りを特定する方法である LLL-F（Lightweight error Localization for Labeled transition systems - First version）を提案する．LLL-F のアイデアは，対象の性質を満たす軌跡（正例）のうち，反例との編集距離の近いものを抽出し，その差分を求めることにある．この差分が，反例が正例から逸脱する箇所，つまり誤り箇所の候補である．ただし，既に述べた通り，多くのモデル検査器では，反例は，一般に無限長の軌跡で与えられるが，無限長文字列で編集距離を扱うことは困難である．そこで，LLL-F では，有限長文字列の編集距離を扱うために反例を接頭辞と閉路に分割し，それぞれについて任意回の編集操作を適用した軌跡の集合からなるモデルを構成する．これらのモデルと，正例の集合である性質の Büchi オートマトンとの積を求めることで，正例を探索する問題を有向グラフ上の最短路問題に帰着させる．なお，LLL-F は文献 [109] でわれわれが提案した手法に基づいている．

LLL-F は，従来の手法が活用する SAT 求解器のような技術を前提とせず，既存のモデル検査器の出力を直接用いる簡便な方法である．また，従来の手法の多くは，安全性の検証に関するプログラムの誤りのみを研究対象としているので活性に適用できないが，LLL-F は安全性，活性共に適用できる．従来の手法は，いずれも特定のプログラミング言語で記述されたプログラムの誤りの特定に注目している [7, 49, 19, 48, 47, 62, 93, 41]．したがって，プログラムが存在していることが前提なので，要求分析や設計時にモデルを検証した際に誤りの特定をする場合への適用が困難である．加えて，従来手法の多くは，対象プログラムに性質や仕様を満たす実行列が少なくとも 1 つ存在することが前提である．よって，そのような実行列が存在しない場合には，従来手法により誤りを特定することが困難であるが，LLL-F では，性質を満たす事象列の集合が Büchi オートマトンにより与えられるので，この制約がない．さらに，反例の

修正候補というモデルを修正する際の有力な手掛かりとなる情報を開発者に提示できる．

われわれは LLL-F を自動化したプロトタイプを Java 言語により実装した．まず，7 種類のシステム事例について LLL-F を実行し，各々の事例の実行時間と生成された結果を評価した．この結果，LLL-F は 6 つの事例について，モデルの誤りを正しく指摘することを確認した．続いて，正例の探索空間の規模を変化させたときの LLL-F の実行時間の変化を調べた．これらの結果より，LLL-F は大規模なシステムや反例に対しても実用的な実行性能を持つことを確認した．この事例研究の結果を 4 章の最後に結果を報告する．

しかしながら，LLL-F は軌跡間の類似性を明確に定義しておらず，有限長文字列間の編集距離を用いた発見的手法によって反例と類似した正例を求める．したがって，反例に含まれる誤りの修正が適切になされた正例を生成できない場合がある．そこで，5 章で無限長軌跡間の距離を用いて LLL-F の精緻化を行う．5 章では，まず，LLL-F の発見的な側面に起因する課題を具体例を用いて明らかにする．続いて，以上の課題を解決する正例の生成，誤り特定手法として LLL-S（Lightweight error Localization for Labeled transition systems - Second version）を提案する．LLL-S は LLL-F を拡張した方法であり，反例と正例間の類似度の基準として，有限長の文字列間の編集距離を拡張した指標を新たに導入する．この指標は，無限長の反例に対して適用可能であるように定義した距離関数であり，上述の問題点を解決することができる．そして，この指標が最小となる値を持つ正例を，LLL-F と同様に有向グラフ上の最短路問題を解くことによって求める．誤りを特定するための手続きには，LLL-F で開発した反例と正例の比較手法を用いる．

LLL-S の先行研究は LLL-F と同様であるが，反例と類似した正例を求める手法の多くは，反例，正例共に有限長であることを前提としているので，その類似度もまた有限長のプログラム実行列や事象列を前提としている [7, 49, 48, 47, 62, 93, 41, 10]．したがって，本研究で扱う無限長列間の類似性の計算には適用できない．

われわれは，LLL-S のプロトタイプツールを Java 言語を用いて実装した．LLL-S の実行性能の向上のための最適化処理を，プロトタイプツールは組込んでいるので，それらの処理について簡単に説明する．5 章では，このプロトタイプツールを用いた事例研究の結果を報告する．まず，4 章と同様の 7 種類のシステム事例に対して LLL-S を実行し，各々の事例の実行時間と生成された結果を評価した．この結果，LLL-S は 6 つの事例について，モデルの誤りを正しく指摘することを確認した．残り 1 つの事例に関しては，反例を手動で修正し，同一の軌跡を表すが，接頭辞の部分列を閉路に移した軌跡を作成することで，同様に誤りを特定できることを確認した．続いて，正例の探索空間の規模と LLL-S のプロトタイプの実行時間との関係を調べた．この結果から，LLL-F よりも正例探索に必要な計算時間が大きいが，LLL-S は大規模システムや反例に対して実用的な性能を持つことを確認した．LLL-S の計算時間が大きいのは，LLL-F を精緻化したことが要因である．

6 章において本論文で議論したことをまとめる．特に，3 章，4 章，5 章で提案した方法を統合することで，統一的なモデル修正技術を確立するという点から，各提案手法を議論する．提案したモデル修正法とモデルの誤り特定法は，モデル検査の対象となるモデルと性質の記法が共通していること，そして，同一のモデル検査手法が用いられることを想定している．また，

互いの欠点を補うことを目標としているので，統合が可能であると思われる．加えて，統一的な方法論の確立のために残された研究課題について議論する．最後に，モデル修正技術とモデル検査技術の統合を実用的に促進するための技術として，モデルの修正を前提とした効率的な検査技術の実現可能性を論じる．

第 2 章

オートマトンに基づくモデル検査技術

本章では，本論文で前提とするモデル検査技術について説明する．まず，2.1 節で，モデル検査技術と従来提案されている代表的な手法の概要を述べる．モデル検査技術は，状態遷移機械で記述されたモデルが，論理式によって与えられた性質を満たすかどうかを自動的に検証する技術のことである．従来研究されてきた手法は，オートマトンを用いる方法と記号的方法の 2 種類に大きく分類される．それぞれについて簡単に述べた後，近年研究が進められている SAT 求解器を用いた検査手法を紹介する．続いて，検証を実用化するために考案されている検査の最適化手法について述べる．モデル検査技術は各種のモデル検査器として実用化されている．本節の最後に代表的なモデル検査器を紹介する．

本論文が対象とするモデルの記法はラベル付き遷移系とし，線形時相論理式についてのオートマトンを用いるモデル検査法を前提とする．ラベル付き遷移系については，2.2 節で述べる．本論文で扱う線形時相論理は，古典的な線形時相論理に流動の概念を導入した流動線形時相論理と呼ばれるものである．流動線形時相論理は，ラベル付き遷移系の上で意味論が定義される．その詳細を 2.3 節で説明する．

ラベル付き遷移系を対象とする流動線形時相論理式のモデル検査法には，従来のオートマトンに基づくモデル検査法を拡張した方法が提案されている．これについて 2.4 節で説明する．

最後に，2.5 節において，現在のモデル検査技術の課題であり，われわれの研究目標であるモデルの修正法の必要性を論じる．

2.1 モデル検査とは

モデル検査とは，システムを記述したモデルが所望の性質を満たすことを自動的に検証する技術である [25, 5]．モデル検査の概要を図 2.1 に示す．入力は，モデルと性質であり，両者とも形式的に記述されたものであることが必要である．モデルには，システムの振る舞いを記述する記法が採用されることが多く，クリプキ構造に代表される状態遷移機械が広く用いられる．また，性質はシステムに要求される主張を形式的に記述したものであり，時相論理を用い

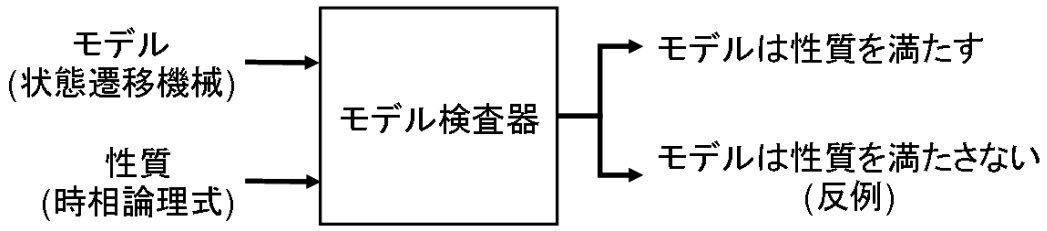


図 2.1. モデル検査の概要

た論理式で表されることが多い．一方，出力は，モデルが性質を満たす場合と満たさない場合で異なる．性質を満たす場合は，モデルが正しいことが確認されたので，単にモデルが性質を満たすことを通知するだけよい．性質を満たさない場合は，モデルが誤っていると考えられる．このとき，モデルが性質を満たさないことを通知するだけでは，モデルがなぜ誤っているのかが不明である．特に，複数のプロセスが並行に動作するシステムではモデルの規模が非常に大きくなり，人手により誤りの理由を発見するのは難しい．そこで，性質に違反するモデル上の実行例である反例を通知することで，モデルが誤っている証拠を開発者に示すことが多い．

従来から研究されているモデル検査の問題は，与えられたクリプキ構造に対する時相論理式の充足可能性問題として，以下のように述べられる．

> 「M をクリプキ構造（状態遷移グラフ）とする．また，f を時相論理式（仕様）とする．次の条件を満たす M の全ての状態 s を探索せよ：s は f を満たす（$M, s \models f$）.」 [23]

上の問題記述で，M のことをモデル，f のことを性質と本論文では呼称する．以降，本節では，上の問題に記述に従って，これまで広く研究されている古典的な技術を中心として，モデル検査技術について概説する．

2.1.1　モデルの記法

モデル検査で使われるモデルの記法として，最もよく研究されているものはクリプキ構造である．クリプキ構造は，$M =< S, s_0, \Delta, Label >$ の 4 つ組で定義される．ここで，S は状態の集合，$s_0 \in S$ は初期状態，$\Delta \subseteq S \times S$ は遷移関係，そして，$Label : S \rightarrow 2^{AP}$（$AP$ は命題の集合）は状態に対するラベル付け関数である．ラベル付け関数は，その状態において真である命題の集合を表す．クリプキ構造は，対象となる命題の真理値の時間的な変化を表現したモデルといえよう．なお，$M, s_0 \models f$ を簡単のため，$M \models f$ と表すことにする．これは，モデル M が性質 f を満たすということを意味する．

2.1.2 時相論理

時相論理は，様相論理の一種で，古典論理に時間に関する演算子を追加した論理体系である．このような論理は，システムの状態の時間的な推移に関する多様な主張を表現することができる．モデル検査で扱う時相論理には，線形時相論理（LTL） [76] や計算木論理（Computation Tree Logic: CTL, CTL*） [26] が用いられることが多い．LTL は，古典的な論理演算子に加えて，時間演算子として単項演算子 $\mathbf{X}$, $\mathbf{G}$, $\mathbf{F}$，および 2 項演算子 $\mathbf{U}$ を持つ論理体系である．表 2.1 に，LTL で用いられる各時間演算子とそれらの直観的な意味の説明を示す．この説明は必ずしも，各演算子の意味を正確に述べているわけではないが，後の議論に現れる LTL 式の理解に役立つ．

LTL 式の解釈は，クリプキ構造上の 1 つの計算（状態列）を固定することで与えられる．クリプキ構造 M において，状態 s が LTL 式 f を満たすのは，s から始まる全ての計算が f を満たす場合であると定義することによって，システムが実行することのできる全ての状態列（システムの振る舞い）が満たすべき望ましい主張を LTL を用いて記述できる．例えば，M が LTL 式 $\mathbf{G}p$ を満たすのは，s_0 から始まる M の全ての状態列に対して，各状態列を構成する全ての状態が p を満たすときであり，そのときに限る．

モデル検査でよく使われる LTL 式は，安全性と活性に分類される [67, 12, 74]．安全性は望ましくない事象が決して起こらないという性質であり，代表的なパターンは $\mathbf{G}\neg p$ と定式化される．ここで，p は望ましくない出来事や状態を記述した論理式である．一方，活性は望ましい事象がいつかは起こるという性質であり，その代表的な論理式は $\mathbf{F}p$ である．ここで，p は望ましい出来事や状態を記述した論理式である．

安全性と活性という分類とは異なる観点で論理式を分類した研究に，Dwyer 等により提案された仕様パターンがある [39]．仕様パターンは，論理式のパターンの中でもしばしば用いられるものを，論理式が表す意味に注目して分類したものである．Dwyer 等は，事象の発生を意味する論理式と，複数の事象の発生順序を表す論理式，という 2 つの観点から 8 種類のパターンを同定して，カタログにまとめた．このカタログは，注目する事象の発生や述語の成立に関する 4 種類のパターンと，事象や述語の順序に関する 4 種類のパターンからなる．そのうちいくつかを紹介する．

- 不在性 $\mathbf{G}\neg p$．これは，p が常に成り立たないということを表す論理式であり，事象の

表 2.1. LTL における時間演算子の直観的な意味

演算子	意味
$\mathbf{X}\phi$	次の時点で ϕ が成り立つ
$\phi_1\mathbf{U}\phi_2$	いつか ϕ_2 が成り立つまで，常に ϕ_1 が成り立つ
$\mathbf{F}\phi$	いつか ϕ が成り立つ
$\mathbf{G}\phi$	常に ϕ が成り立つ

発生や述語の成立に関するパターンの 1 つである．p がシステムにおいて望ましくない出来事や状態を表す論理式ならば，これは安全性を記述した代表的な論理式でもある．

- 存在性 $\mathbf{F}p$．これは，p がいつか成り立つということを表す論理式であり，事象の発生や述語の成立に関するパターンの 1 つである．また，上で述べたように，活性に分類される論理式である．
- 応答性 $\mathbf{G}(p \Rightarrow \mathbf{F}q)$．常に，$p$ が成り立つならばいつか q が成り立つ，ということを表現する LTL 式であり，順序に関するパターンの 1 つである．ただし，$\Rightarrow$ は含意を表す古典的な論理演算子である．このパターンは，例えば，外界からのシステムへの入力を表す式 p に対して，システムから適切な応答 q が p の成立以降に成り立つという主張を記述できる．そのため，応答型システムの望ましい振る舞いを表す性質の記述に用いられることが多い．これは，安全性と活性という分類基準においては，活性に分類される論理式である．
- 応答鎖 $\mathbf{G}(p \Rightarrow \mathbf{F}(s \wedge \mathbf{XF}t))$．これは，応答性を拡張したパターンで，$p$ の成立以降に s が成り立ち，また，s が成立した後に t が成り立つという性質であり，順序に関するパターンの 1 つである．よって，システムに与えられる入力に対して，2 つの応答がなされることを表す性質の記述に用いられる．

本論文では，LTL のモデル検査を前提とする．

一方，CTL 式の解釈は，注目する状態から始まるあらゆる計算経路を表す木（計算木）によって定義される．CTL は LTL とは異なり，全ての計算経路についての性質だけでなく，ある性質を満たす計算経路が存在するかどうかについての主張を記述することができる．CTL ではこのような計算経路に関する記述を論理式として明示的に表すために，計算経路についての量化子 $\mathbf{A}$, $\mathbf{E}$ を導入する．これらはそれぞれ，「全ての計算経路に対して」，「ある計算経路が存在して」ということを意味する．時間演算子は，LTL と同様に表 2.1 にあるものを扱う．ただし，CTL 式は，LTL のように時間演算子が自由に論理式中に現れることを許さず，常に量化子を伴わなければならない．また，量化子もまた自由に論理式に現れることができず，時間演算子を必ず伴う．例えば，上で挙げた不在性パターンを CTL 式によって記述すると，$\mathbf{AG}\neg p$ となる [39]．クリプキ構造 M において，状態 s が CTL 式 f を満たすのは，s が根となる計算木が f を満たす場合である．例えば，M が CTL 式 $\mathbf{EG}p$ を満たすのは，全ての状態が p を満たすような s_0 から始まる M の状態列が少なくとも 1 つ存在するときであり，そのときに限る．

CTL と LTL は表現能力が異なり，比較可能ではない．つまり，CTL 式では表現可能でない LTL 式，および逆に LTL 式では表現可能でない CTL 式が存在する．例えば，LTL 式 $\mathbf{FG}p$ は，p がいずれ成立し，以降も常に成立し続けることを主張する論理式である．これと等価な意味を持つ CTL 式は存在しない．逆に，CTL 式 $\mathbf{AGEF}p$ と等価な意味の LTL 式は存在しない．この式は，p がいずれ成り立つような計算経路が，全ての経路に対して常に存在するということを表す式である．

CTL*は，量化子 $\mathbf{A}$, $\mathbf{E}$ を時間演算子の前で任意に入れ子にできるように CTL を拡張した

体系である．CTL*は全ての CTL 式と LTL 式を表現することができ，また，CTL でも LTL でも表現できない式を表現できるので，両者よりも真に表現能力が大きい．例えば，CTL*式 $\mathbf{AFG}p \vee \mathbf{AGEF}q$ と等価な CTL 式や LTL 式は存在しない．なぜなら，部分式 $\mathbf{AFG}p$ は LTL 式 $\mathbf{FG}p$ と等価であり，よって上の議論より CTL で表現できず，また，$\mathbf{AGEF}q$ は LTL で表現できない CTL 式であるからである．

2.1.3 モデル検査手法

クリプキ構造に対するモデル検査技法は，古典的にはオートマトンを用いる方法と記号的方法の 2 種類に大きく分類することができる．これらの共通点は，クリプキ構造を有向グラフとみなして，モデルの検証をグラフ上の経路探索問題に帰着させることである．ただし，近年はモデルの検証を充足可能性問題に帰着させる手法も開発されている．

オートマトンを用いる方法

オートマトンを用いる方法は，LTL 式の検査に用いられる．この方法は，任意の LTL 式は等価な言語を受理する Büchi オートマトンに変換することができる，という事実に基づいた方法である [101]．Büchi オートマトンは，無限語を受理するオートマトンの一種である．オートマトンを用いるモデル検査のアイデアは，クリプキ構造 M が LTL 式 ϕ を満たすことを検証するために，$M \models \phi$ を反証する手続きを実施することにある．反証できないならば $M \models \phi$ と結論付ける．そのために，M 上で ϕ を満たさない計算経路，つまり，$\neg\phi$ を満たす計算経路を探索する．そのために，以下の手順に示すように性質の否定の Büchi オートマトンを利用する．

1. $\neg\phi$ と等価な Büchi オートマトンを構成する．
2. 構成した Büchi オートマトンと検査対象モデルとの積オートマトンを求める．
3. 積オートマトンを有向グラフとみなし，この積オートマトンが受理する計算経路をグラフ探索アルゴリズムを用いて探索する．

最後のステップで受理する経路が存在することは，モデルが性質の否定を満たす，すなわち，モデルが性質を満たさない経路を持つことを意味する．よって，この経路が発見された場合は，モデルは性質を満たさないと判定され，発見した経路を反例として返す．そうでない場合，モデルは性質を満たすと判定され，真を返す．本論文は, この Büchi オートマトンを用いる検査手法によってモデルの検証が行われることを前提とする．次節でこの方法の詳細を説明する．

記号的方法

記号的方法は，性質と等価な特別なオートマトンを構成せずにモデル上を探索する方法であり，主に CTL の検査に用いられる [26]．そのアイデアは，次のようにまとめられる．

- 性質を部分論理式に分割しながらモデルの状態空間を探索し，性質を満たす状態の集合を再帰的に求める．
- モデルの初期状態が性質を満たす状態集合の要素であるとき，モデルはその性質を満たすと結論付ける．そうでない場合は，モデルは性質を満たさないと判定する．

この方法は，完備束上の単調関数は不動点を持つという不動点定理 [96, 35] に基づいて，状態集合の包含関係によって構成される束上で最小不動点を探索する手続きを実行している．記号的方法は，LTL の検査にも応用されている [28]．この方法は，LTL 式から構成したタブローを検査の際に用いることで，LTL 式の検証を CTL 式の検証に帰着させる．

CTL*の検査には，LTL と CTL の検査手法を組み合せた手法が提案されている [25]．

有界モデル検査

近年の計算機の性能向上により，SAT 求解器を使ったモデル検査技術である有界モデル検査技術 [13] の研究が進められている．有界モデル検査は，与えられた特定の長さの反例を探索する検査手法である．そのために，与えられた長さの反例が存在するとき, かつそのときに限り充足可能である命題論理式をモデルの遷移関係から生成する．この論理式が充足可能であれば，その解を反例としてモデル検査器は出力し，そうでない場合は定められた長さの反例が存在しないことを利用者に報告する．この方法は，安全性の検証技術として開発が進められてきたが，近年は活性の検証にも応用されている [12]．

2.1.4　最適化手法

モデル検査技術が実用化されるにあたっての大きな課題は，ソフトウェアシステムの規模が大きくなるにつれてそのモデルの規模もまた大きくなることである．よって，現実のソフトウェアシステムの検証を行うには，大規模な計算資源が必要となってしまう．この問題は，状態空間の爆発問題といわれ，解決するための手法が数多く研究されてきた．

- **半順序簡約手法** [25, 55, 5]．半順序簡約手法は，性質の検証に無関係な状態空間を削減する手法である．異なる順序で実行された遷移が同じ状態に到達し，実行順序が性質の成立に無関係である場合に，検査器が探索する遷移の実行順序の数を簡約する．
- **抽象化技術**．モデルの抽象化技法は，モデル検査時に探索する状態空間を減らすために，状態数を圧縮した抽象化モデルを作成し，そのモデルに対して検査を実施する技術である．中でも，述語抽象技術は，主にプログラムを対象とするモデル検査に活用される抽象化技術である [8, 6, 11]．述語抽象は，検証したい性質に関係する述語の真理値の変化のみに注目した抽象モデルを検証対象のモデルから構成することによって，状態数の削減を実現する．

  抽象化モデルは元の検査対象モデルが表す振る舞いの一部を捨象しているので，抽象化モデルの検証結果が元のモデルの検証結果と一致するとは限らない．この場合，抽象化モデルを洗練し，元のモデルに近い具体化モデルを再構成し検証を行う手法が提案され

ている．この検証プロセスは，抽象-検査-洗練プロセスと呼ばれる．代表的な抽象化モデルの洗練技術に，反例を活用した抽象化洗練手法 (CEGAR) がある [24]． この方法は，抽象化モデルにモデル検査を実施した結果得られた反例を元に，抽象化モデルと元のモデルでモデル検査の結果が同じになるように，反復的にモデルの抽象化関数を更新する方法である．

- **3 値モデル検査** [15]．これは，抽象化したモデル，あるいは振る舞いが部分的に知られているモデルを不完全クリプキ構造と呼ばれる特殊な記法で表現し，3 値時相論理を用いて検証を実施する方法である．3 値時相論理は，真偽のほかに第 3 の値「不定」を持つ．この不定値は，モデルが性質を満たすか判定できないことを表す．不完全クリプキ構造は，クリプキ構造の各状態において命題の真理値を与えるラベル付け関数を，2 値から 3 値に拡張したものである．モデル検査の結果不定値が得られた場合は，抽象化やモデルの部分的な記述によって性質の検証に必要な元のモデルの情報が捨象されてしまい，モデルが性質を満たすか不明であると解釈される．この方法は，特に抽象-検査-洗練プロセスにおいて有効なアプローチである．検証結果が不定のときには，モデルの抽象化関数を更新して再度検証を行うことを示唆する．

  一般化モデル検査は，3 値論理を用いた抽象化の精度を改善するための検査手法である [16]．検証したい性質を満たす具体化モデルが存在するときに限り，抽象化モデルに対して不定を返すので，抽象-検査-洗練プロセスの効率化が実現できる．

2.1.5 代表的なモデル検査器

モデル検査技術は，ソフトウェアツールとして実用化されており，産業界での利用も含めて現在普及が進められつつある [104, 81, 20, 115, 110, 77]．例えば，Wing 等は，ネットワーク通信プロトコルの検証にモデル検査を適用した事例研究を行った [104]．また，中島等は，Java の Enterprise Java Beans アーキテクチャの妥当性を，モデル検査器を用いて検証した結果を報告している [81]．Miller 等は，産業界におけるソフトウェアを対象としたモデル検査適用事例を報告している [77]．

SPIN は Holzmann によって開発されたモデル検査器である [55]．SPIN は，クリプキ構造で表されたモデルと LTL 式で記された性質を対象として，Büchi オートマトンを用いた検査法を実現している．SPIN のモデル記述言語は Promela と呼ばれ，SPIN の処理系によってクリプキ構造に変換されて検査が実行される．

SPIN と同様に，LTL 式を対象としたモデル検査器に LTSA がある [74]．LTSA は，CSP に基づいて定義された FSP というモデルの記法を採用している．LTSA における検査対象のモデルはクリプキ構造ではなく，次に説明するラベル付き遷移系であり，特にシステムの振る舞いや事象に注目した性質の検証に適している．LTSA で扱うラベル付き遷移系を様相遷移系 [68, 56] に拡張した検証器である MTSA [38] も開発されている．

一方，CTL 式と LTL 式を記号的に検査する方法を採用したモデル検査器には，NuSMV が

知られている [22].

近年は，クリプキ構造など抽象的なモデルに対する従来の検証技術をプログラムの検証に応用する技術が活発に研究されている．このようなモデル検査器は，特にソフトウェアモデル検査器 [60] と呼ばれ，様々なツールが作成されている．

SLAM プロジェクトは，C 言語で書かれたプログラムの検証を実施するモデル検査器を作成することを目標としたプロジェクトである [8, 6]．SLAM は，これまで述べてきたモデル検査器とは異なり，プログラムの静的解析技術の 1 つである到達可能性解析を活用した手法を採用している．すなわち，性質違反となるプログラム状態への到達可能性を静的に調べることで検査を実施する．したがって，プログラムの安全性の解析に用いられる．加えて，モデル検査技術における抽象化技術である述語抽象と，反例に基づく抽象化モデルの洗練技術である CEGAR を組込むことにより，大規模なソフトウェアへの活用を目指している．SLAM プロジェクトの成果は，オペレーティングシステム Windows 用のドライバの検証器として実用化されている．

一方，BLAST も C 言語プログラムの安全性の検証を行うためのソフトウェアモデル検査器である [11]．BLAST は，SLAM で採用された述語抽象と CEGAR を改善した抽象化技術を実装している．

CBMC もまた ANSI-C プログラムに対する安全性の検証を行うことのできるモデル検査器である [30]．CBMC の特徴は，SAT 求解器を用いた有界モデル検査技術をツールとして実現している点にある．

2.2　ラベル付き遷移系

本論文で用いるモデルはラベル付き遷移系（LTS）で記述される．LTS はクリプキ構造と異なり，状態ではなく状態間の遷移にラベルが付けられる．このラベルは，システムに関連する事象を表すと解釈されるので，LTS は事象に基づいてシステムの振る舞いを記述するのに適している．

定義 1.（ラベル付き遷移系 [74]） ラベル付き遷移系（LTS）を 4 つ組 $L =< S, A, \Delta, s_0 >$ で定める．ただし，各項は以下の通りである．

- S は状態の有限集合である．
- A は事象の有限集合である．
- $\Delta \subseteq S \times A \times S$ は遷移関係である．
- $s_0 \in S$ は初期状態である．

■

L の遷移関係 $(s, a, t) \in \Delta$ を簡単のため，$s \xrightarrow{a} t$ で表すことがある．遷移関係 $(s, a, t) \in \Delta$ を s の出遷移あるいは t の入遷移といい，s, t をそれぞれ出状態，入状態と呼ぶ．$n \geq 1$ に対して，事象列 $\pi = [a_0, a_1, \ldots, a_{n-1}]$ $(\forall 0 \leq i < n.(s_i, a_i, s_{i+1}) \in \Delta)$ を L の軌跡と呼び，

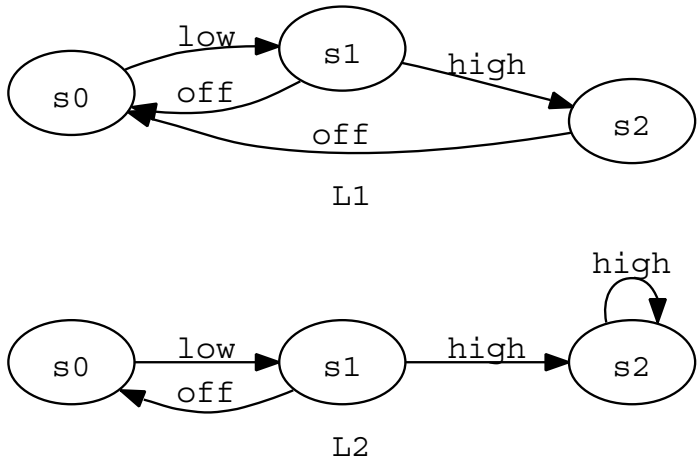


図 2.2. 3 段階スイッチモデル

$n = \infty$ のとき，π を無限長の軌跡，そうでない場合は有限長の軌跡という．特に断らない限り，本論文では無限長の軌跡を単に軌跡と呼ぶ．L の全ての軌跡の集合を $Tr(L)$ と記す．軌跡 π において，i 番目の事象から始まる接尾辞を $\pi[i] = [a_i, a_{i+1}, \ldots]$ で表す．$s \in S$ において $(s, a, s) \in \Delta$ を s の自己閉路と呼び，自己閉路以外の出遷移を 1 つも持たない状態を終端状態という．軌跡 $\pi = [a_0, a_1, \ldots]$ が閉路をなすとは，以下を満たす長さ $n \geq 1$ の π の部分列 $\pi' = [a_0, a_1, \ldots, a_{n-1}]$ が存在することである．

$$\forall i \geq 0. a_i = a_{i+n}$$

この閉路の定義は，π のみならず，π の接尾辞 $\pi[i]$ に対しても拡張して用いる．すなわち，$\pi[i] = [a_i, a_{i+1}, \ldots]$ が閉路をなすとは，以下を満たす長さ $n \geq 1$ の π の部分列 $\pi'[i] = [a_i, a_{i+1}, \ldots, a_{i+n-1}]$ が存在することである．

$$\forall j \geq i. a_j = a_{j+n}$$

例 1. LTS で記述したモデルの例として，図 2.2 に off, low, high の 3 段階のスイッチシステム L_1, L_2 を示す．L_1 と L_2 はともに状態 s_0 が初期状態で，このとき，スイッチは off となっている．その後，これらのスイッチを，事象 *low* によって low に切り替えることができる．同様に，事象 *off*, *high* は，それぞれスイッチを off, high に変更する事象を表す．

■

複数の LTS で記述されたモデルが与えられたとき，各 LTS を異なるプロセスと考えることにより，並行システムのモデル化が可能となる．並行システム全体の振る舞いは，各プロセスの並列合成 [74] を求めることで得られる．並列合成演算により得られるプロセスは，各プロセスで互いに共有していない事象は任意の相対順序で実行する．これは非同期実行と呼ばれ，一度に 1 つのプロセスだけが遷移を 1 ステップ実行することができる計算モデルである．他方，プロセス間の相互作用を表現するために，複数のプロセスで共有する事象は同時に実行す

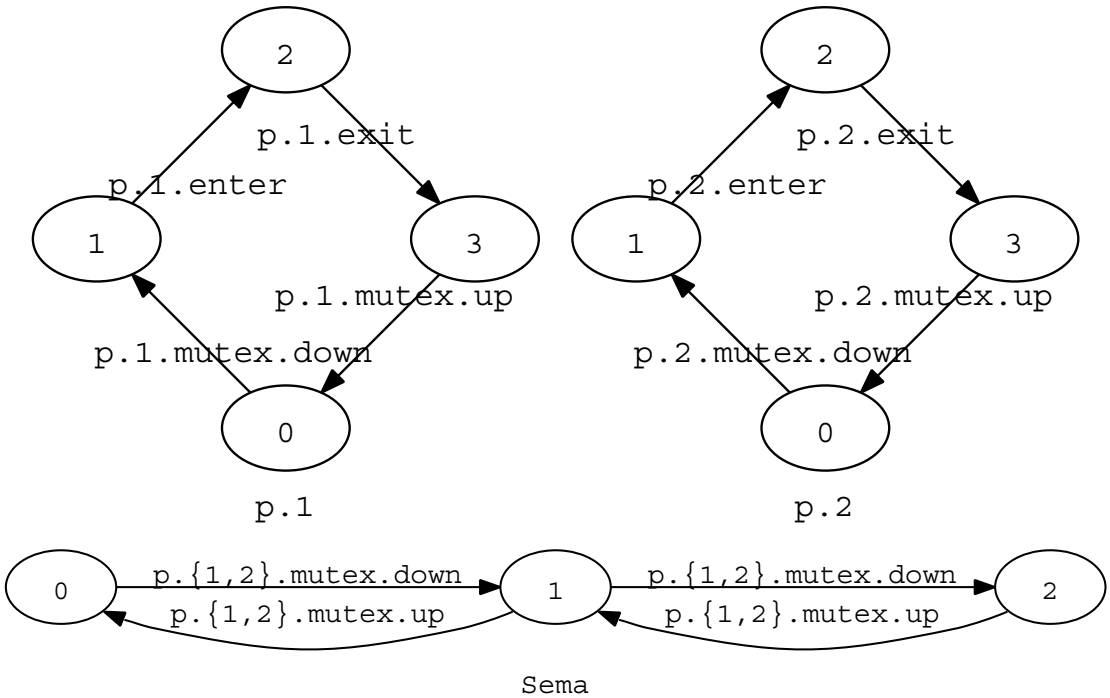


図 2.3. セマフォを用いた並行システム CSys

るようにする．これは同期実行と呼ばれ，共通する事象を持つプロセスが同期して協調動作することを表現する計算モデルである．

定義 2. （**LTS の並列合成 [74]**）$L_1 =< S^1, A^1, \Delta^1, q_0^1 >$, $L_2 =< S^2, A^2, \Delta^2, q_0^2 >$ を LTS とする．L_1 と L_2 の並列合成 $L_1 \parallel L_2$ もまた LTS であり，以下のように定義する．

$$L_1 \parallel L_2 =< S^1 \times S^2, A^1 \cup A^2, \Delta, (q_0^1, q_0^2) >$$

ここで，遷移関係 $\Delta \subseteq (S^1 \times S^2) \times (A^1 \cup A^2) \times (S^1 \times S^2)$ は以下の規則により定める．

$$\begin{aligned}\Delta =&\{((s^1, s^2), a, (s'^1, s'^2))|(s^1, a, s'^1) \in \Delta^1, (s^2, a, s'^2) \in \Delta^2\}\cup\\ &\{((s^1, s^2), a, (s'^1, s^2))|(s^1, a, s'^1) \in \Delta^1,\ a \notin A^2\}\cup\\ &\{((s^1, s^2), a, (s^1, s'^2))|(s^2, a, s'^2) \in \Delta^2,\ a \notin A^1\}.\end{aligned}$$

■

並列合成演算で得られる LTS は，その書き方に拠らずに一意に定まる．すなわち，次の交換律と結合律を満たす．

$$L_1 \parallel L_2 = L_2 \parallel L_1$$
$$L_1 \parallel (L_2 \parallel L_3) = (L_1 \parallel L_2) \parallel L_3$$

例 2. 図 2.3 にセマフォによる並行システム CSys のモデルを示す [74]．これは，計算プロセス 1（p.1）と 2（p.2）がセマフォプロセス（Sema）を使って排他的に計算資源を利用するモ

デルで，いずれも状態 0 が初期状態である．*p.1.enter* と *p.2.enter* により，p.1 と p.2 は危険領域に入って共有資源を利用して必要な処理を行い，その後，*p.1.exit* と *p.2.exit* により危険領域から出る．出状態，入状態が等しくプロセス名のみが異なる事象を持つ遷移に対して次の例に示すような省略表記を導入する．遷移 $(0, p.1.mutex.down, 1)$ と $(0, p.2.mutex.down, 1)$ を 1 つの遷移で $(0, p.\{1,2\}.mutex.down, 1)$ と表記する．

CSys のシステム全体の振る舞いは 3 つのプロセス p.1，p.2，Sema の並列合成である．例えば，各プロセスが初期状態にあるとき，p.1 が *p.1.mutex.down* により状態 0 から 1 に遷移する場合，同時にセマフォプロセスもまた *p.1.mutex.down* により状態 0 から 1 に遷移する．*p.1.mutex.down* は p.2 で共有されていないので，この場合 p.2 は遷移せず状態 0 に留まる．ゆえに，並列合成演算により得られる遷移関係は，$((0,0,0), p.1.mutex.down, (1,0,1))$ となる．

■

2.3 流動線形時相論理

線形時相論理（LTL） [76] は古典的な論理演算子（$\mathbf{t}$, $\mathbf{f}$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$）と時間演算子（$\mathbf{G}$, $\mathbf{U}$, $\mathbf{F}$, $\mathbf{X}$）で構成される．

LTS はシステムで観察される事象の順序，すなわちシステムの振る舞いを記述するモデルなので，事象に関する性質を LTL で記述できることが望ましい．このような性質を記述するために，事象によって真理値が定義される述語である流動の概念と，その流動の真偽について推論する LTL である流動 LTL（FLTL）が提案されている [45]．流動は FLTL で扱う命題で，直観的には軌跡に現れる事象の生起に応じて真偽が変化する．FLTL は LTS で記述されるモデル上の全ての軌跡が満たすべき性質を，そのモデルに現れる事象を用いて記述するのに有用な論理体系である．

定義 3. **（流動 [45]）** 流動 $\mathtt{fl}$ は 3 つ組 $\mathtt{fl} =< I_{\mathtt{fl}}, T_{\mathtt{fl}}, b_{\mathtt{fl}} >$ である．ここで，$I_{\mathtt{fl}}, T_{\mathtt{fl}} \subseteq A$，$I_{\mathtt{fl}} \cap T_{\mathtt{fl}} = \emptyset$，$b_{\mathtt{fl}} \in \{\mathbf{t}, \mathbf{f}\}$ とする．また，A は事象の集合である．$I_{\mathtt{fl}}$ は開始事象の集合，$T_{\mathtt{fl}}$ は終了事象の集合，$b_{\mathtt{fl}}$ は初期値を表す．軌跡 $\pi[i] = [a_i, a_{i+1}, \ldots]$ が流動 $\mathtt{fl}$ を満たす（$\pi[i] \models \mathtt{fl}$ と書く）ための必要十分条件は，以下の 2 条件のいずれかが成り立つことである．

- $b_{\mathtt{fl}} \wedge (\forall j \in \mathcal{N}.0 \leq j \leq i \Rightarrow a_j \notin T_{\mathtt{fl}})$,
- $\exists j \in \mathcal{N}.(j \leq i \wedge a_j \in I_{\mathtt{fl}}) \wedge (\forall k \in \mathcal{N}.j < k \leq i \Rightarrow a_k \notin T_{\mathtt{fl}})$.

■

$\mathtt{fl}$ の直観的な意味は次の通りである．与えられた軌跡上のある事象が開始事象の要素 $a_I \in I_{\mathtt{fl}}$ であるならば，a_I 以降に終了事象の要素 $a_T \in T_{\mathtt{fl}}$ がその軌跡上に現れるまで $\mathtt{fl}$ は真である．それ以外の場合は $\mathtt{fl}$ は偽となる．また，時間零の時点での $\mathtt{fl}$ の真理値を $b_{\mathtt{fl}}$ で表す．以降，流動の集合を $\mathtt{FL}$ で表す．

例 3. 図 2.3 のシステムにおいて以下のような流動を定義する．

$$\texttt{CRITICAL.p.1} =< \{p.1.enter\}, \{p.1.exit\}, \mathbf{f} >$$
$$\texttt{CRITICAL.p.2} =< \{p.2.enter\}, \{p.2.exit\}, \mathbf{f} >$$

これらは，それぞれプロセス p.1 と p.2 が危険領域に入っている間真であるような流動である．すなわち，CRITICAL.p.1 と CRITICAL.p.2 はそれぞれ p.1 と p.2 が危険領域に入っていることを表す．初期状態では，どちらのプロセスも危険領域には入っていないことを表現するために $b_{\texttt{CRITICAL.p.1}} = b_{\texttt{CRITICAL.p.2}} = \mathbf{f}$ としている．

■

与えられた軌跡上で事象 a が発生することもまた流動で表現することができる．そのためには，a が発生したときのみ真理値が真となり，それ以外の事象が発生すると偽となるように流動を定めればよい．その際，時点零においては事象が発生していないので，初期値は $\mathbf{f}$ とする．よって，A を LTS の事象の集合とすると，事象 a についての流動 $\texttt{fl}_a$ は以下のように定められる．

$$\texttt{fl}_a =< \{a\}, A - \{a\}, \mathbf{f} >$$

以下では，記法の簡素化のため，事象を表す流動 $\texttt{fl}_a$ を事象と同じ記号 a で記述することとする．

例 4. 図 2.2 において，各事象を流動として表すと以下のようになる．

$$\texttt{fl}_{high} =< \{high\}, \{low, off\}, \mathbf{f} >$$
$$\texttt{fl}_{low} =< \{low\}, \{high, off\}, \mathbf{f} >$$
$$\texttt{fl}_{off} =< \{off\}, \{high, low\}, \mathbf{f} >$$

以降の議論では，これら 3 つの流動 $\texttt{fl}_{high}, \texttt{fl}_{low}, \texttt{fl}_{off}$ をそれぞれ $high, low, off$ と略記する．図 2.3 の CSys に現れる各事象も流動として同様に表現することができる．

■

以上で定めた流動を命題として扱う LTL が FLTL である．FLTL の構文は以下のように定められる．

定義 4. **（FLTL の構文 [45]）** 流動の集合 FL 上の FLTL 式 ϕ を，以下の構文により帰納的に定める．

$$\phi = \mathbf{t} \mid \texttt{fl} \in \texttt{FL} \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \mathbf{X}\phi \mid \phi_1 \mathbf{U} \phi_2$$

■

π を LTS L の軌跡，ϕ を FLTL 式とすると，ϕ の意味論は π 上で真となるか偽となるかによって定められ [45]，それが真と評価される場合 π は ϕ を満たすという．π が ϕ を満たすことを $\pi \models \phi$ と表す．また，L が ϕ を満たすことを $L \models \phi$ と書き， L の全ての軌跡が ϕ を満

たすこと，すなわち，$\forall \pi \in Tr(L).\pi \models \phi$ を意味する．FLTL の意味論を以下のように帰納的に定める．

定義 5. （**FLTL 式の意味 [45]**）ϕ を流動の集合 FL 上の FLTL 式とし，$\pi \in A^\omega$ を事象の集合 A 上の軌跡とする．充足関係 $\models \subseteq A^\omega \times \mathtt{FLTL}$ は以下の通り定められる最小の二項関係である（FLTL は FLTL 式の集合とする）．

$\pi \models \mathbf{t}$

$\pi \models \mathtt{fl} \in \mathtt{FL}$ iff $\pi[0] \models \mathtt{fl}$,

$\pi \models \neg\phi$ iff $\pi \not\models \phi$,

$\pi \models \phi_1 \wedge \phi_2$ iff $(\pi \models \phi_1)$ かつ $(\pi \models \phi_2)$,

$\pi \models \mathbf{X}\phi$ iff $\pi[1] \models \phi$,

$\pi \models \phi_1 \mathbf{U} \phi_2$ iff $\exists k \geq 0.(\pi[k] \models \phi_2$ かつ $\forall 0 \leq j < k.\pi[j] \models \phi_1)$.

■

他の演算子は次のように定める．

- $\mathbf{f} = \neg\mathbf{t}$,
- $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$,
- $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$,
- $\mathbf{F}\phi = \mathbf{t}\mathbf{U}\phi$,
- $\mathbf{G}\phi = \neg\mathbf{F}\neg\phi$.

表 2.1 に示した LTL の各時間演算子の直観的な意味は，FLTL にも適用することができる．**F** と **G** は以降の議論でもよく用いるので，上と同様に意味を定義しておこう．

$\pi \models \mathbf{F}\phi$ iff $\exists k \geq 0.(\pi[k] \models \phi)$.

$\pi \models \mathbf{G}\phi$ iff $\forall k \geq 0.(\pi[k] \models \phi)$.

LTL と同様に，FLTL 式で記述された振る舞いの性質は安全性と活性に分類される [67, 12, 74]．既に述べたように，安全性は望ましくない事象が決して起こらないという性質で，例えば，2.1 節と同様に，FLTL 式 $\mathbf{G}\neg p$ によって表される．一方，活性は望ましい事象がいつかは起こるという性質で，例えば，FLTL 式 $\mathbf{F}p$ によって表される．

例 5. 図 2.2 の 3 段階スイッチモデルに関する性質を表す以下の論理式は安全性である．

- $\phi_1 = \mathbf{G}(\mathit{off} \Rightarrow \mathbf{X}\mathit{low})$.
- $\phi_2 = \mathbf{G}(\mathit{low} \Rightarrow \mathbf{X}\neg\mathit{low})$.
- $\phi_3 = \mathbf{G}(\mathit{high} \Rightarrow \mathbf{X}\mathit{low})$.

ϕ_1 は，常に，事象 *off* が生起するなら，その次の時点に *low* が発生する，ということを主張している．ϕ_2 と ϕ_3 がそれぞれ主張していることは，*low* は続けて発生することはないということ，および，*high* の次の時点では，常に *low* が起こることである．一方，活性の例には，$\phi_3' = \mathbf{F}\mathit{off}$ が挙げられる．ϕ_3' は，事象 *off* が現在または未来においていずれ起こるということ

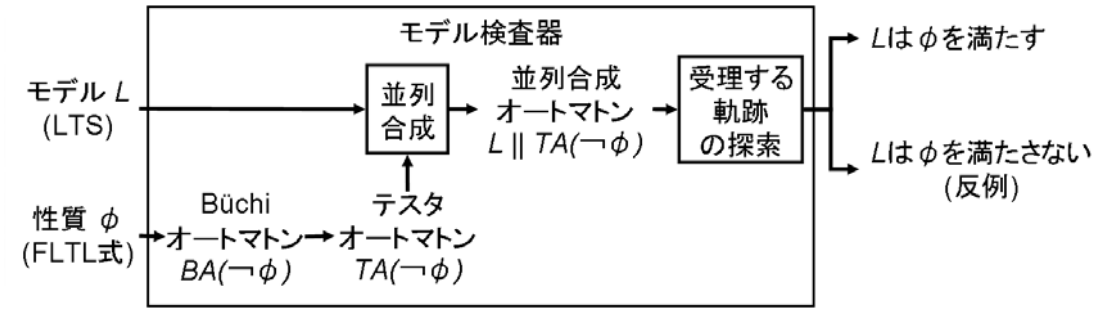


図 2.4. FLTL 式の LTS 上でのモデル検査の手順

を主張する性質である．

■

例 6. 図 2.3 の並行システムについて，性質 EXIT_1= $\mathbf{G}(p.1.enter \Rightarrow \mathbf{F}p.1.exit)$ が活性の例として挙げられる．これは，全ての時点で，p.1 が危険領域に入るならいずれ p.1 は危険領域から出るということを定式化した性質である．すなわち，プロセス p.1 が危険領域に入ったまま出てくることがないことにより，もう一つのプロセス p.2 が危険領域内で行うべき処理が実行できないという望ましくない状況が発生しないことを主張している．

■

2.4　流動線形時相論理式のモデル検査技術

Giannakopoulou と Magee は，FLTL の LTS 上でのモデル検査技術を開発した [45]．図 2.4 に示すとおり，LTS L 上での ϕ のモデル検査手法は，クリプキ構造上の Büchi オートマトンを用いる方法 [101] に基づいている．しかしながら，LTS 上のモデル検査は，クリプキ構造に対する検査法を直接用いることはできない．クリプキ構造の状態には，そこで成り立つ命題の集合がラベル付けされている．このとき，検証する性質は一般に，対象となるモデルでラベル付けされている命題が現れる論理式である．後で示す図 2.5 の例のように，Büchi オートマトンの各遷移は，モデルに現れる命題によってラベル付けされている．一方，FLTL で扱う命題は流動であるが，流動は LTS 上には明示的にラベル付けされていない．よって，FLTL 式の Büchi オートマトンを構成しても，両者のラベルの集合は一般に異なるため，モデルとの積オートマトンが受理する軌跡を探索することによって検証を実施することはできない．

そこで，Giannakopoulou と Magee は，性質に現れる流動の定義に基づいて，Büchi オートマトンを受理状態を持つ LTS に変換する方法を提案した．この結果得られる LTS の遷移に現れるラベルは，対象モデルに現れる事象で記述される．それゆえ，LTS 同士の並列合成演算によって積オートマトンを求めることができる．

以上の議論に基づき，LTS に対する FLTL 式のモデル検査は，以下の手順に従う（図 2.4）．

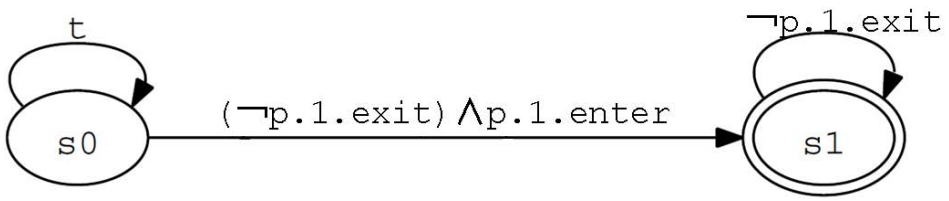


図 2.5. $BA(\neg$ EXIT_1)

1. $\neg\phi$ の Büchi オートマトン $BA(\neg\phi)$ を構成する.
2. $BA(\neg\phi)$ を等価な言語を受理する受理状態付き LTS （テスタオートマトン，TA）$TA(\neg\phi)$ に変換する.
3. L と $TA(\neg\phi)$ の並列合成オートマトンを求めて，Büchi の受理条件を満たす軌跡を探索する [25].
4. 受理する軌跡が存在する場合はその軌跡を反例として返し, そうでない場合は真を返す.

ここで，Büchi オートマトンを以下のように定義する.

定義 6. （**Büchi オートマトン [25]**）Büchi オートマトン BA は 5 個組 $BA =< S_b, A_b, \Delta_b, s_0, S_b^{acc} >$ である．ただし，各項は以下の通りである.

- S_b は状態の有限集合である.
- A_b はアルファベットである.
- $\Delta_b \subseteq S_b \times A_b \times S_b$ は遷移関係である.
- $s_0 \in S_b$ は初期状態である.
- $S_b^{acc} \subseteq S_b$ は受理状態の集合である.

A_b 上の無限列 $\pi = [a_0,\ a_1,\ \ldots]$ $(\forall i \geq 0.(s_i, a_i, s_{i+1}) \in \Delta_b)$ において，いかなる i に対しても $j \geq i$ なる $s_j \in S_b^{acc}$ が π の遷移列 $[(s_0, a_0, s_1), (s_1, a_1, s_2), \ldots]$ に現れるとき，$TA(\phi)$ は π を受理するという.

■

上で述べた BA の受理条件は次のように直観的に言い換えることができる.

BA が軌跡 π を受理するのは，π が BA の受理状態を無限回通過するときである.

LTL 式 ϕ と等価な言語を受理する Büchi オートマトンを $BA(\phi)$ で表す．性質 ϕ の LTL 式が与えられたとき，これを満たす Büchi オートマトン $BA(\phi)$ を自動的に構成するアルゴリズムが提案されている [43, 44]．これらのアルゴリズムが構成する Büchi オートマトン $BA(\phi)$ のアルファベットは，ϕ に現れる命題の冪集合となる．FLTL 式より構成した場合，アルファベットは流動の冪集合となる.

例 7. Büchi オートマトンの例として, 図 2.5 に性質 $\neg$EXIT_1 $= \neg(\mathbf{G}(p.1.enter \Rightarrow \mathbf{F}p.1.exit))$

の Büchi オートマトンを示す．図中の状態 s_0 が初期状態で，2 重丸で表された状態 s_1 が受理状態である．この Büchi オートマトンのアルファベットは，流動の冪集合 $2^{\mathtt{FL}}$ である（ただし，この場合 $\mathtt{FL} = \{p.1.enter, p.1.exit\}$）．また，図において，遷移ラベル $a \in 2^{\mathtt{FL}}$ は簡略化のため次のように表している．各流動 $\mathtt{fl} \in \mathtt{FL}$ に対して，$\mathtt{fl} \in a$ ならば $\mathtt{fl}$，$\mathtt{fl} \notin a$ ならば $\neg\mathtt{fl}$ を項として，全ての項の論理積を遷移ラベルとして記述する．したがって，図中の遷移 $(s_0, \neg p.1.exit \wedge p.1.enter, s_1)$ は，$(s_0, \{p.1.enter\}.s_1)$ を表し，$(s_1, \neg p.1.exit, s_1)$ は，$(s_1, \emptyset, s_1)$ と $(s_1, \{p.1.enter\}, s_1)$ の 2 つの遷移の省略表記を表す．$\mathbf{t}$ は $2^{\mathtt{FL}}$ の全ての要素からなる遷移の集合を意味している．よって，$(s_0, \emptyset, s_0)$, $(s_0, \{p.1.enter\}, s_0)$, $(s_0, \{p.1.exit\}, s_0)$, $(s_0, \{p.1.enter, p.1.exit\}, s_0)$ からなる遷移の集合の省略表記である．

■

検証手続きのステップ 1 で，性質 ϕ の否定の Büchi オートマトン $BA(\neg\phi)$ を構成する．既に述べたように，検証対象モデルと $BA(\neg\phi)$ はアルファベットが一致しないので，それらの積オートマトンを求めるために LTS の並列合成を用いることができない．そこで，検証手続きのステップ 2 で，Giannakopoulou 等によって開発された検証技術 [45] に従って，$BA(\neg\phi)$ のアルファベットが対象モデルの事象集合になるように，流動の定義を用いて $BA(\neg\phi)$ を検証対象モデルの事象によって記述される Büchi オートマトンに変換する．

テスタオートマトン TA は，アルファベットが対象モデルの事象の有限集合であるような Büchi オートマトンである（以降，テスタオートマトンを単に TA と呼称することがある）．よって，性質 ϕ の $BA(\phi)$ と等価な言語を受理するテスタオートマトン $TA(\phi)$ は，遷移ラベルが事象であるような Büchi オートマトンと考えることができ，同時に，受理状態を持つ LTS とみなすこともできる．$TA(\phi)$ の受理条件は，Büchi オートマトン $BA(\phi)$ の受理条件と同様である．

$TA(\phi)$ を $TA(\phi) =< S_t, A_t, \Delta_t, t_0, S_t^{acc} >$ と書くこととする．ただし，各項目は以下の通りである．

- S_t は状態の有限集合である．
- A_t は事象の有限集合である．
- $\Delta_t \subseteq S_t \times A_t \times S_t$ は遷移関係である．
- $t_0 \in S_t$ は初期状態である．
- $S_t^{acc} \subseteq S_t$ は受理状態の集合である．

$TA(\phi)$ 上の軌跡 $\pi = [a_0, a_1, a_2, \ldots]$ $(\forall i \geq 0.(t_i, a_i, t_{i+1}) \in \Delta_t)$ において，いかなる i に対しても $j \geq i$ なる $t_j \in S_t^{acc}$ が π の遷移列 $[(t_0, a_0, t_1), (t_1, a_1, t_2), \ldots]$ に現れるとき，$TA(\phi)$ は π を受理するという．以下では，$TA(\phi)$ が受理する軌跡の集合を，$Tr(TA(\phi))$ と表すことにする．

例 8. 文献 [45] に従って，図 2.5 の $BA(\neg\mathtt{EXIT\_1})$ と流動 $p.1.enter$ と $p.1.exit$ の定義を用いて構成したテスタオートマトン $TA(\neg\mathtt{EXIT\_1})$ を図 2.6 に示す．図 2.5 と同様に，s_0 が初期状

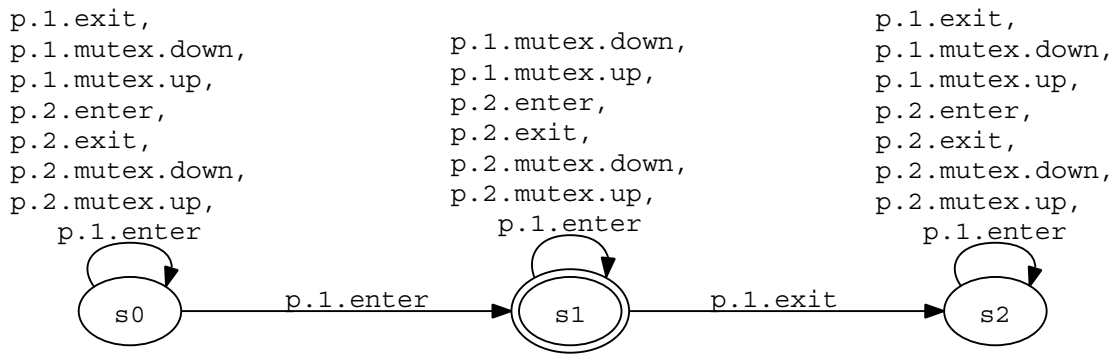


図 2.6. $TA(\neg\texttt{EXIT\_1})$

態，2 重丸の状態 s_1 が受理状態を表す．このオートマトンが $\neg$EXIT_1 を受理することは，以下の観察により分かる．初期状態 s_0 から事象 *p.1.enter* により受理状態 s_1 に達することで EXIT_1 の含意の前件が真となる．その後，後件である *p.1.exit* が真にならなければ，s_1 の自己閉路を遷移し続けるので，$\neg$EXIT_1 を満たす．一方，s_1 から *p.1.exit* により受理状態でない s_2 へ遷移してしまうと，s_1 に戻ることができないので，そのような軌跡は受理されない．EXIT_1 の含意の前件が真となることなく偽であり続ける場合は EXIT_1 はやはり真である．この場合，$TA(\neg\texttt{EXIT\_1})$ の s_0 にとどまるため，受理されない．

■

先に示した検証手続きのステップ 3 では，モデル $L =< S, A, \Delta, s_0 >$ が検証したい性質の否定 $\neg\phi$ を満たさないことを確認するために，L と $TA(\neg\phi)$ の並列合成オートマトンが受理する軌跡が存在しないか調べる．これは以下の議論に基づく．$Tr(\phi, A)$ を，L の事象の集合 A の要素のみが各事象に現れ，かつ，ϕ を満たす軌跡の集合，つまり，$Tr(\phi, A) = \{\pi \in A^\omega | \pi \models \phi\}$ とすると，$Tr(\phi, A) = Tr(TA(\phi))$ である．よって，次の関係が成り立つ．

$$
\begin{aligned}
L \models \phi \text{ iff } & \forall \pi \in Tr(L).\pi \models \phi \\
\text{iff } & Tr(L) \subseteq Tr(\phi, A) \\
\text{iff } & Tr(L) \subseteq Tr(TA(\phi)) \\
\text{iff } & Tr(L) \cap (A^\omega - Tr(TA(\phi)) = \emptyset \\
\text{iff } & Tr(L) \cap Tr(TA(\neg\phi)) = \emptyset
\end{aligned}
$$

L と $TA(\neg\phi)$ の並列合成オートマトンを受理する軌跡が存在しないならば，上式より L は ϕ を満たすと判定される．そうでないならば，L は ϕ を満たさないと判定され，受理する軌跡の 1 つを反例とする．Büchi の受理条件により，このオートマトンが受理する軌跡は，初期状態から到達可能な受理状態を無限回通過する軌跡を求めればよい．このような軌跡の探索は，2 重深さ優先探索法やグラフの強連結成分の探索手法を用いることで，有向グラフ上の経路探索問題に帰着させることができる [34, 84, 25, 33, 5]．中でも，簡潔なアルゴリズムである 2 重深さ優先探索法 [25] は以下のような手順で実施される．

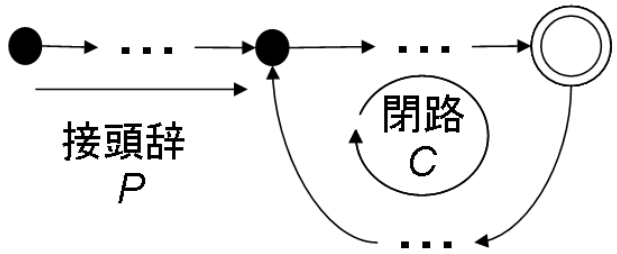


図 2.7. 2 重深さ優先探索法で探索する反例

1. 初期状態から深さ優先探索 [33] により，受理状態を探索する．
2. 受理状態に到達したならば，その状態を始点として改めて深さ優先探索を行い，ステップ 1 で到達した状態のいずれかに到達したら終了する．閉路を発見できない場合は，引き続きステップ 1 を実行する．

この探索によって発見される反例は，図 2.7 に示す「投げ縄型」の形状をなす．すなわち，2 重深さ優先探索法は，有向グラフ上の強連結成分と，そこへの有限長の事象列を求めるので，反例は無限長の軌跡 $\pi = PC^{\omega}$ の構造を持つ．ここで，$P = [a_0,\ a_1, \ldots, a_{m-1}]$, $C = [b_0,\ b_1, \ldots, b_{n-1}]$ は無限列や閉路を部分列として含まない有限長の事象列で，C は無限回繰返される閉路，P は C への接頭辞である．探索のステップ 1 では，初期状態から受理状態までの軌跡を求める．続いて，探索のステップ 2 において，その受理状態から始まる閉路を構成する事象列を求める．このようにして得られる反例は閉路が受理状態を通過するので，Büchi の受理条件を満たす．多くの既存のモデル検査器（例えば LTSA）はこの構造の反例を出力する．ただし，性質が安全性の場合には，有限長の反例 $\pi = P = [a_0,\ a_1, \ldots, a_{m-1}]$ を出力することが多い．これは，望ましくない事象が現れる π はその事象以降の事象列に関わらず常に性質違反なので，反例の探索を，受理状態への到達可能性解析問題に帰着させられるためである [45].

本論文では，活性の検証を行った場合には，モデル検査器が出力する反例 π は，投げ縄型の無限長軌跡であると想定する．上で述べたように，既存のモデル検査器は，投げ縄型反例を探索する反例探索技術を採用している．したがって，投げ縄型反例を仮定することは妥当であると考えられる．一方，同様に既存のモデル検査器が出力する反例が有限長の軌跡であることから，安全性の検証を行った場合には，反例が有限長軌跡であると仮定する．

例 9. 活性を検証する例として，図 2.3 の並行システムについて，活性 `EXIT_1` を検証する．そのためには，図 2.3 の各プロセス p.1, p.2, Sema と，図 2.6 の $TA(\neg\texttt{EXIT\_1})$ の並列合成を求め，それが受理する軌跡を探索すればよい．図 2.3 のシステムは `EXIT_1` を満たさない．これは，p.1 が危険領域にいる間に p.2 が無限回危険領域に入ることが可能なためで，モデル検

査器は次の反例を出力する．

$$\pi_{\texttt{EXIT\_1}} = [p.1.mutex.down, p.1.enter(p.2.mutex.down, p.2.enter, p.2.exit, p.2.mutex.up)^{\omega}]$$

■

例 10. 図 2.2 の L_1 において，2.3 節で取り上げた性質 ϕ_1, ϕ_2, ϕ_3 の検査を考えると，L_1 は ϕ_1, ϕ_2 を満たすが，ϕ_3 を満たさない．L_1 は，状態 s_1 あるいは s_2 から s_0 への遷移により，ϕ_1 の含意の前件を満たし，加えて，s_0 から s_1 の遷移が後件を満たすので，ϕ_1 を満たすことが分かる．なお，それ以外の遷移を実行する場合は，ϕ_1 の含意の前件が満たされないので，やはり ϕ_1 を満たす．ϕ_2 についても同様である．一方，L_1 が ϕ_3 を満たさないのは，状態 s_2 へ入る遷移が *high* なので ϕ_3 の含意の前件が満たされるが，s_2 からの出遷移が *off* であるので，後件を満たさないためである．実際，モデル検査の結果得られる反例は，以下の有限長の軌跡である．

$$\pi_{\phi_3} = [low, high, off]$$

■

活性に対しては，しばしば公平な選択と呼ばれる公平性を考えることが多い [46, 74]．公平性とは，モデル検査において，特定の計算経路のみに注目して検証を行うことによって，現実的でない計算経路を検査対象から除外するための検証の際に設ける制約である．LTS で扱われる公平性である公平な選択とは，ある遷移集合についての選択が無限回実行されるならば，その集合の全ての遷移もまた無限回実行されるという制約である．この制約の下でモデル検査を行うときには，2 重深さ優先探索法によって探索される受理状態を含む閉路が通過する状態の集合が，単なる強連結成分ではなく終了状態集合でなければならない [46]．ここで，終了状態集合とは，集合に含まれる全ての状態から，その集合の要素ではない状態への遷移が存在しないような強連結成分である．

例 11. 図 2.3 の並行システム CSys について，公平な選択を仮定して `EXIT_1` の検証を行うことを考える．この場合，モデル検査器は Csys は `EXIT_1` を満たすと判定されるため，反例が出力されることはない．なぜなら，公平な選択により，各プロセスの全ての選択を実行する軌跡，すなわち，CSys 上のプロセス p.1 が危険領域に出入りする軌跡のみが検証対象となるためである．CSys の検証で示した反例 $\pi_{\texttt{EXIT\_1}}$ は，p.1 が危険領域に留まり続ける軌跡を表しており，公平な選択を仮定した場合には，検証対象の軌跡とならない．以降の議論で，システムの全ての軌跡を検証の対象とするために，CSys の検証を行う場合は公平な選択を仮定しないこととする．

■

例 12. 図 2.2 の L_2 について性質 ϕ_1, ϕ_2, ϕ_3' の検査を考えると，L_2 は L_1 と同様に ϕ_1, ϕ_2 を満たす．しかし，公平な選択を仮定するかどうかに関わらず，L_2 は活性 ϕ_3' を満たさない．L_2 が ϕ_3' を満たすには，L_2 の全ての軌跡に 1 回以上 *off* が現れなくてはならないので，遷移

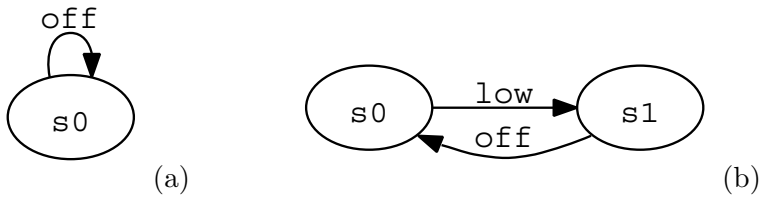


図 2.8. L_2 の不適切な修正結果

(s_1, off, s_0) を必ず通過しなければならない．この遷移を通らずに L_2 の状態 s_2 に入った場合，s_0 あるいは s_1 に戻る遷移が存在しないため，そのような軌跡は ϕ'_3 を満たさない．ゆえに，反例として以下の軌跡が得られる．

$$\pi_{\phi'_3} = [\mathit{low}, \mathit{high}(\mathit{high})^\omega]$$

■

2.5　モデル検査技術の課題

これまで行われてきたモデル検査技術に関する研究は，検証技術の開発やその効率化に注目したものが多い．それゆえ，既存のモデル検査器は，モデルが性質を満たさないと判定された場合には，反例を出力するのみであり，開発者が誤りを発見して修正するための支援が十分ではない．例えば，図 2.3 の並行システムや図 2.2 のスイッチにおいて性質を満たすモデルを得るためには，開発者は，前節に示した反例を用いて，モデルの誤りの原因を特定して修正作業を実施しなければならない．しかし，反例は誤りをどのように正すべきかを開発者に提示するわけではなく，開発者は，モデル上で反例を解釈する必要がある．加えて，反例の解釈の結果を元に行うモデルの修正作業は，開発者が手作業で行うことになる．このモデルの修正作業を支援する手法の研究は，従来のモデル検査技術の研究においてあまり注目されていない．また，修正作業を機械により自動化することが可能であるとしても，一般に，性質を満たすモデルは複数存在する．よって，どのモデルを修正の結果とすべきかの判断が適切に行われなくてはならない．

例 13. 図 2.2 のスイッチ L_2 と性質 $\phi'_3 = \mathbf{F}\mathit{off}$ を考える．前節の議論より，L_2 が性質を満たすためには，事象 off が反例中に必要である．この条件を満たすようにモデルを修正すると，例えば，次のような一連の修正操作の候補が考えられる．

1. 全ての軌跡が事象 off を実行するように，L_2 の全ての遷移を削除する．
2. s_0 に自己閉路 (s_0, off, s_0) を追加する．
3. 遷移を削除したモデルの初期状態から到達不可能な状態 s_1，s_2 を削除する．

以上の手続きで得られたモデルを図 2.8(a) に示す．この修正モデルは性質 ϕ'_3 を満たす．しかしながら，このモデルは L_2 が満たしていた性質 ϕ_1 を満たさない．性質 ϕ_1，ϕ_2 が対象領域の

知識や想定事項，仕様を表す場合，L_2 のみならず修正モデルもまたこれらの知識を満たさねばならない．よって，図 2.8(a) のモデルは対象領域の知識を反映した妥当な修正候補とはいえず，対象領域において無意味な修正候補である．このようなモデルが与えられる原因は，対象領域の知識や要求，システムの仕様が修正工程において活用されない点にある．

図 2.8(a) 以外にも修正操作は幾通りも考えられるので，妥当な修正モデルを開発者が選択しなくてはいけない．図 2.8(b) に示したモデルは，ϕ_3' のみならず，対象領域知識の知識あるいは仕様を定式化した ϕ_1，ϕ_2 も満たす．しかしながら，このモデルはスイッチを high に切り替えることができないので，3 段階スイッチの機能を実現できない．よって，このモデルは適切な修正モデルとはいえない．図 2.8(b) が不適切と判断されるのは，全ての対象領域知識が定式化されず，開発者にとって暗黙的な知識が存在するためである．要求分析，設計工程は，システムや問題領域で成り立つべき性質を明らかにしていく工程でもあるので，その過程で活用されるモデルにおいてそれら全てが必ずしも明らかになっている訳ではない．したがって，図 2.8 に示したモデルだけでなく，現在判明している対象領域の知識を含む複数の修正候補を開発者に提示することが望ましい．

■

上の例で論じた問題点の解決のためには，開発者に対して対象領域の知識を満たす修正候補のみを提供することが望ましい．加えて，開発者によるモデル修正作業を支援するには，複数の修正候補を開発者に提示し，開発者が妥当なモデルを候補の中から選択できることも必要である．本論文の目標は，これらの点の克服を目指したモデルの修正法によって，開発者によるモデル修正作業を支援する手法を提案することである．

開発者がモデルに対して行う修正作業は，まず，モデルが含む誤りを発見し，続いて，発見した誤りの修正を行う，という手順で実施されると考えられる．以上のような，開発者による修正作業の支援は後者に相当するので，前者に相当する作業の支援も必要である．すなわち，修正したモデルの候補を提示するだけでなく，モデル上の修正すべき箇所を機械的に特定することが望ましい．

例 14. 図 2.3 の並行システムと性質 `EXIT_1` を考える．前節で述べた通り，このモデルは `EXIT_1` を満たさず，モデル検査器により反例 $\pi_{\texttt{EXIT\_1}}$ が提示される．前節の反例の解釈により，セマフォプロセス Sema が排他制御を正しく実施していないことが誤りの原因であることが分かる．詳しく説明すると，p.2 が危険領域に進入することを許可する事象 *p.2.mutex.down* を Sema が実行するときに，p.1 が危険領域に入ることを許可する事象 *p.1.mutex.down* が実行されており，かつ，p.1 を危険領域から出す事象 *p.1.mutex.up* が実行されていない経路が Sema のモデルに存在することが，反例が示す Sema の誤りの原因である．モデルの修正のためには，まず，Sema を開発者が調査してこの誤りを発見しなければならない．ゆえに，反例が表すモデルの誤りを発見する技術を提供することが，モデルの修正技術を補うために必要である．

■

本論文で提案する技術は，次の 2 点に集約される．

- LTS モデルの修正手法.
- LTS モデルに含まれる誤りの特定手法.

次章以降でこれらの技術について説明する．

第 3 章

反例に基づくモデル修正法

本章では，提案するモデル修正法を述べる．提案手法は，修正の結果，開発者が性質を満たすモデルを構築することができるように支援する反復型の半自動的な方法である．本手法に対する入力は，修正対象のモデル，検証を行う性質，モデル検査の結果得られる反例，ならびに，モデルにおいて真であることが示されている性質の集合とする．検証を行う性質は，FLTL 式の活性または安全性のどちらで定式化された性質でもよい．一方，最後の性質の集合は，FLTL 式の安全性で定式化された性質の集合とする．この性質の集合は，開発対象システムに関する問題領域の知識やシステムに要求される仕様を表現していると考えられる．本章では，これらの厳密な区別を行わず，システムにとって望ましい振る舞いを記述した性質の集合を領域知識と呼ぶことにする．対象領域の知識を用いることで，修正したモデルが対象領域において前提とされる条件に従わず，修正対象モデルと全く無関係なモデルを構築するのを防ぐことができる．加えて，提案手法は開発者との対話的な手法なので，明確化されていない対象領域知識がある場合に開発者によるモデル選択が可能である．よって，開発者の意図した修正を行うための支援ができる．

提案手法は，モデル修正問題を，反例と領域知識を表す性質からそれぞれ構成したモデルのモデル合併問題に帰着させる．まず，反例と領域知識を多値遷移系（特に，4 値遷移系）と呼ばれるモデルに変換する．これは，LTS の遷移関係に，その遷移の発生しやすさの確度を追加したモデルである．反例から構成したモデルは，反例の実行を禁止し，それ以外の軌跡を表すモデルである．領域知識から構成したモデルは，領域知識に含まれる全ての安全性を満たす軌跡を表すモデルである．そして，それら 2 つのモデルを合併することで，修正モデルの候補を求める．この修正モデルの候補は，領域知識を満たすモデル上で反例が実行できないように，反例を削除したモデルである．開発者は，候補の中から最も望ましいと判断されるモデルを選択することで修正を実施する．これらの手続きを，モデルが性質を満たし，かつ，開発者により適切と判断されるモデルが抽出されるまで繰返すことで，修正モデルを得ることができる．

この手法の主な利点は次のとおりである．

- 提案手法は，既存のモデル検査器を直接用いて実施することができる．
- 提案手法は，安全性と活性の両方に適用が可能である．加えて，公平な選択を仮定した

モデル検査を行った場合にも適用できる．

- 提案手法は，モデル合併技術を利用しているので，修正モデルの候補を求めるための合併演算を計算機により自動化することが可能である．

提案手法は，領域知識と反例に基づいてシステムの振る舞いを修正する手法である．それゆえ，システムが複数のプロセスから成る並行システムの場合には，各プロセスの並列合成によって得られるシステム全体の振舞いに対してしか適用できない，という制限がある．この点は今後の研究で解決すべき課題である．

3.1 節で，モデル修正に用いるモデルの記法である多値遷移系とその合併演算の定義をし，3.2 節でモデル修正問題を定義する．この問題を解決するためにわれわれが提案するモデル修正法を 3.3 節で説明し，その有効性を 3.4 節で事例研究により評価する．3.5 節では，提案するモデル修正法の改善を図る．改善は，モデル修正作業は修正対象モデルを基にして行われるので，修正の結果得られるモデルは修正対象モデルと類似するという想定に基づいて行う．われわれは，モデルが表現する振る舞いに基づく類似度を，Sokolsky 等の指標 [92] を用いて定義して，以下のように活用する．第 1 に，修正対象モデルが持つ情報を十分に活用するため，対象モデルと合併操作で得られる修正モデルの候補との類似度を算出する．これによって，両者の関係を開発者に提示にできるので，修正作業の効率化が可能になると考えられる．そして，第 2 に，修正の結果得られるモデルと修正対象モデルとの類似度を計算する．これは，修正結果の妥当性を測定する指標を開発者に提供する．開発者にこれらの情報を手掛かりとして提示することで，修正モデルの選択を支援することができる．3.6 節では，提案手法や事例研究から得られた知見を整理し，今後に残された課題を論じる．続いて，3.7 節で，モデル修正についての先行研究について議論する．モデル検査の結果に基づいてモデルの修正を行う，というわれわれが想定する観点から進められている研究は決して多くはない．そこで，モデルの合成技術や，ソフトウェアテストに基づくプログラム修正技術に関する先行研究に言及する．最後に，3.8 節でまとめを行う．

3.1　モデル合併法

提案する手法のアイデアは，問題領域についての知識を表すモデルと，モデル検査によって得られる反例を除いた任意の軌跡を表すモデルから，領域知識から反例を除いたモデルを構築することにある．そのために，領域知識のモデルと反例を表すモデルを構築して，両者を合併することで該当するモデルを構成する．この点を達成するためには，構成すべきモデルにおいて領域知識を満たし，かつ実行可能な軌跡と，反例のように実行できてはならない軌跡を区別してモデル上で表現できることが望ましい．

そこで，本節では，提案するモデル修正法で扱うモデルの記法である多値遷移系を定義する．提案手法は，遷移にその実行可能性をラベル付けした 4 値遷移系を修正に用いることで，その遷移が修正したモデルにおいて実行可能であるかどうかをモデル上で表現する．4 値遷移系の各遷移を，それぞれ，実行が必須，実行可能，有限回のみ実行可能，実行禁止のいずれか

を表すと解釈することで，上記の要請を満たすと考えられる．特に，有限回のみ実行可能と解釈される遷移で表現される軌跡は，本論文で想定している投げ縄型の無限長反例を，実行されてはいけない軌跡としてモデル上で表現するのに適している．

記法の定義に引き続いて，多値遷移系，特に 4 値遷移系で記述されたモデルの合併演算を定義する．この 4 値遷移系についての合併演算は，文献 [98, 97] を拡張した演算で，問題領域において実行可能な振る舞いと実行が必須となる振る舞いを，合併を行った結果得られるモデル（合併モデル）上で明示的に表現するように定義される．そのために，合併対象となる 2 つのモデルにおいて実行可能な振る舞いは，合併モデルにおいても実行可能となるように合併演算を定義する．同様に，合併対象となる 2 つのモデルの内，少なくとも 1 つで実行が禁止される振る舞いは，合併モデルにおいても実行禁止となるように定める．その結果，反例は合併モデル上でも明示的に表現される．

3.1.1 多値遷移系

文献 [98, 97] で扱われているモデル合併法において，モデルは様相遷移系 (MTS) [68, 56] で記述される．MTS は LTS の拡張で，遷移関係はその実行の確度によって特徴付けられる．MTS は，必須遷移（must 遷移）と可能遷移（may または maybe 遷移）という 2 種類の遷移を持つ．直観的には，必須遷移は，システムが実現しなければならない振る舞いを表し，その遷移ラベルである事象が必ず実行される遷移である．MTS 上のある状態に達したときに，その状態からの出遷移が必須遷移であり，次に起こる事象がその遷移ラベルと一致するならば，必ずその遷移が実行される．一方，可能遷移は，システムが実現可能な振る舞いを表し，その遷移ラベルである事象が実行が可能であるが必須ではない遷移である．その状態からの複数の出遷移が可能遷移ならば，次にそれらの遷移ラベルのいずれかが事象として発生する．モデルを MTS で記述することによって，開発対象システムが実現すべき振る舞いと，実現することが許される振る舞いやシステムの振る舞いの候補を区別して表現することができる．

しかしながら，モデル修正の観点からは，MTS の表現力は次の 2 点について十分でない．

- MTS は，有限回の実行は許されるが，無限回の実行は許されないような遷移関係を記述することができない．このことは，モデル検査結果得られる反例に含まれる閉路を，モデル上で表現できないことを意味する．
- MTS は，実行することが禁止された振る舞いを明示的に表現する機構を有していない．この機構は，特に検証対象となる性質が安全性の場合に，有限長の反例をモデル上で表現するために必要である．

これらの点を克服するために，MTS の表現力をより高めた記法である多値遷移系（Multi-Valued Transition Systems: MVTS）を導入する．MTS の MVTS への拡張は，文献 [21] で論じられたクリプキ構造の χ クリプキ構造への拡張法と同様の方法で行われる．χ クリプキ構造は，状態に付加するラベルと遷移に対して，束を用いて値付けをするクリプキ構造の拡張記法である．

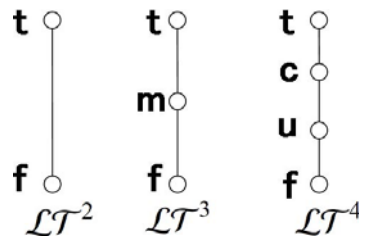


図 3.1. 束の例

われわれが導入する MVTS は，束を用いて遷移を分類する遷移系である．本論文では，束を対応する遷移の実行確度を表すと解釈する．まず，準備として束を定義する．

定義 7.（束 [35]）集合 LT において，LT の元の間に半順序 $\sqsubseteq$ が定義されているとする．半順序集合 $\mathcal{LT} = (LT, \sqsubseteq)$ において，LT の任意の 2 元が LT の中に下限と上限を持つとき，$\mathcal{LT}$ を束という．$a, b \in LT$ に対して，下限を $a \sqcap b$，上限を $a \sqcup b$ と書く．下限と上限をそれぞれ交わり，結びという．

■

束 $\mathcal{LT} = (LT, \sqsubseteq)$ において，演算 $\sqcap$ と $\sqcup$ は結合律，交換律，冪等律，吸収律を満たす．すなわち，以下の等式を満たす．

$a \sqcap (b \sqcap c) = (a \sqcap b) \sqcap c,$	$a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c$	結合律
$a \sqcap b = b \sqcap a,$	$a \sqcup b = b \sqcup a$	交換律
$a \sqcap a = a,$	$a \sqcup a = a$	冪等律
$(a \sqcap b) \sqcup a = a,$	$(a \sqcup b) \sqcap a = a$	吸収律

束 $\mathcal{LT}$ は結びと交わりを用いて，$\mathcal{LT} = (LT, \sqcap, \sqcup)$ とも定義される．実際，任意の $a, b \in LT$ に対して，$a \sqcap b = a$（または $a \sqcup b = b$）であるとき $a \sqsubseteq b$ と定めることで，集合 LT に半順序 $\sqsubseteq$ を定義することができる [35]．本論文で扱う束は有限束に限る．したがって，LT の任意の空でない有限部分集合が，LT の中に上限および下限を持つ．以降特に断らずに有限束を束と呼ぶことにする．

例 15. 束の例を図 3.1 にハッセ図でいくつか示す．図 3.1 の束はいずれも，任意の要素間において順序関係が存在するので，全順序集合である．このとき，各束には最大元と最小元が存在するので，本論文ではそれらをそれぞれ **t**, **f** と表す．

■

2 つの束 $\mathcal{LT}_1 = (LT_1, \sqcap_1, \sqcup_1)$, $\mathcal{LT}_2 = (LT_2, \sqcap_2, \sqcup_2)$ の直積もまた束となり，$\mathcal{LT}_1 \times \mathcal{LT}_2 = (LT_1 \times LT_2, \sqcap, \sqcup)$ と表される．ただし，任意の $a_1, a_1' \in LT_1, a_2, a_2' \in LT_2$ に対して，$(a_1, a_2) \sqcap (a_1', a_2') = (a_1 \sqcap_1 a_1', a_2 \sqcap_2 a_2')$, $(a_1, a_2) \sqcup (a_1', a_2') = (a_1 \sqcup_1 a_1', a_2 \sqcup_2 a_2')$ と定める．以降，簡単のため，$(a, b) \in LT_1 \times LT_2$ を ab と記す．

$\mathcal{LT}_1 = (LT_1, \sqsubseteq_1)$, $\mathcal{LT}_2 = (LT_2, \sqsubseteq_2)$ を束，$f : LT_1 \to LT_2$ を LT_1 から LT_2 への写像とする．このとき，任意の $a_1, a_2 \in LT_1$ に対して，$a_1 \sqsubseteq_1 a_2$ ならば $f(a_1) \sqsubseteq_2 f(a_2)$ が成り立つとき，写像 f は単調であるという．

以上に述べた束の概念を用いて MVTS を以下のように定義する．

定義 8. **（多値遷移系 (MVTS)）** MVTS は 5 個組 $MV =< S, A, \Delta, s_0, \mathcal{LT} >$ である．ここで，各項は以下の通りに定められる．

- S は状態の有限集合である．
- A は事象の有限集合である．
- $s_0 \in S$ は初期状態である．
- $\mathcal{LT} = (LT, \sqcap, \sqcup)$ は束である．
- $\Delta : S \times A \times S \to LT$ は，遷移関係を $\mathcal{LT}$ の要素に写す遷移関数である．

MV の全ての状態 $s \in S$ は全ての事象について全体性を満たさなくてはならない，すなわち，s は A の全ての要素について出遷移を持つ．

■

以下では，MVTS MV の事象集合 A を表すのに，$A(MV)$ を用いることがある．また，遷移 $\Delta(s, a, s') = l$ を $s \xrightarrow{a}_l s'$ と書く．

例 16. 束 $\mathcal{LT}$ が図 3.1 における $\mathcal{LT}^2 = (LT^2, \sqcap^2, \sqcup^2)$ であるとき, MVTS MV は LTS と等価であるとみなせる．なぜなら，Δ によって写される値は，真（**t**）または偽（**f**）であり，これらはそれぞれ LTS において対応する遷移が存在すること，および存在しないことと解釈されるからである．

同様に，束 $\mathcal{LT}$ が図 3.1 における $\mathcal{LT}^3$ であるとき, MVTS MV は MTS と等価であると解釈できる．$\mathcal{LT}^3$ は 3 値を表す束で，**t** と **f** は LTS の場合と同じであり，**m** は真でも偽でもない第 3 の値を表す．MTS の必須遷移と可能遷移は，$\mathcal{LT}^3$ を束として持つ MVTS において，それぞれ **t** と **m** を取る遷移に相当する．

■

本論文では，束 $\mathcal{LT}$ として図 3.1 において 4 値を表す $\mathcal{LT}^4 = (LT^4, \sqcap^4, \sqcup^4)$ をもつ MVTS を扱う．このような MVTS を 4 値遷移系（4-Valued Transition Systems: 4VTS）と呼ぶ．4VTS はそれぞれ実行確度を表す **t**, **c**, **u**, **f** の 4 種類の遷移からなり，それぞれ次のように解釈される．

- $\Delta(s, a, s') = \mathbf{t}$ を必須遷移と呼ぶ．この遷移は，状態 s に到達して a が起こったならば，この遷移が必ず実行されることを意味する．
- $\Delta(s, a, s') = \mathbf{c}$ を可能遷移と呼ぶ．直観的には，状態 s に到達したならば，この遷移が起こりうることを意味する．

- $\Delta(s, a, s') = \mathbf{u}$ を不確実遷移と呼ぶ．この遷移が起こりやすくはないが，必ずしも禁止されていないことを表す．特に，有限回の実行は許されるが，無限回の実行は許されない遷移を表現すると解釈する．
- $\Delta(s, a, s') = \mathbf{f}$ を禁止遷移と呼ぶ．状態 s に到達しても，この遷移は実行されてはいけないことを意味する．すなわち，状態 s で a が起こり，この遷移が実行される軌跡はモデル上で認識されないとする．したがって，a が起こるのはエラーを表すと解釈する．

LTS や MTS などの多くの遷移系では，禁止遷移は明示的には記述されることはないが，エラーを表す状態への遷移とみなすことで，ここでの解釈を用いて扱うことができる．一方，4VTS では，特定の事象の無限回の実行を禁止する不確実遷移や，事象の実行を許さない禁止遷移を明示的に扱う．これらは，モデル修正を行うために構築するモデルにおいて，モデル検査の結果得られる反例の実行禁止を表現するために活用する．また，必須遷移は，修正によって得られる修正モデルが実行すべき振る舞いを表すことに用いる．同様に，可能遷移は，修正モデルで開発者による選択が必要な振る舞いを表すことに用いる．以上のように，本論文では上述の全ての種類の遷移が 4VTS 上で明示的に記述されていると仮定する．

遷移についての以上の解釈を基にして, 4VTS の軌跡を定める．$FV =< S, A, \Delta, s_0, \mathcal{LT}^4 >$ を 4VTS とし，FV 上の事象列 $\pi = [a_0, a_1, \ldots](\forall i \in \mathcal{N}.a_i \in A)$ を軌跡と呼ぶ．FV は全体性を満たすので，FV は A の要素からなるあらゆる軌跡を表すことができる．次に，π を以下のように実行確度に基づいて分類する．

- **必須軌跡**：必須遷移のみを通過する軌跡である．形式的には，全ての $i \geq 0$ に対して $s_i \xrightarrow{a_i}_{\mathbf{t}} s_{i+1}$ であるような軌跡が FV 上に存在するとき，π を必須軌跡と呼ぶ.
- **可能軌跡**：可能遷移または不確実遷移を実行する軌跡である．また，それらに加えて，必須遷移を実行してもよい．ただし，不確実遷移を実行する場合は，有限回実行に限る．形式的には，π が必須軌跡ではなく，また，次の 2 条件を同時に満たすとき，π を可能軌跡と呼ぶ.
  - 全ての $i \geq 0$ に対して，$s_i \xrightarrow{a_i}_{\mathbf{t}} s_{i+1}$ または $s_i \xrightarrow{a_i}_{\mathbf{c}} s_{i+1}$ または $s_i \xrightarrow{a_i}_{\mathbf{u}} s_{i+1}$ である．
  - $j \geq i$ なる全ての j について $s_j \xrightarrow{a_j}_{\mathbf{t}} s_{j+1}$ または $s_j \xrightarrow{a_j}_{\mathbf{c}} s_{j+1}$ であるような $i \geq 0$ が存在する．
- **不確実軌跡**：必須遷移，可能遷移または不確実遷移を実行し，かつ，不確実遷移を無限回通過する軌跡である．形式的には，π が必須軌跡でも可能軌跡でもなく，次の 2 条件を同時に満たすとき，π を不確実軌跡と呼ぶ.
  - 全ての $i \geq 0$ に対して，$s_i \xrightarrow{a_i}_{\mathbf{t}} s_{i+1}$ または $s_i \xrightarrow{a_i}_{\mathbf{c}} s_{i+1}$ または $s_i \xrightarrow{a_i}_{\mathbf{u}} s_{i+1}$ である．
  - いかなる $i \geq 0$ に対しても，ある $j \geq i$ が存在し，$s_j \xrightarrow{a_j}_{\mathbf{u}} s_{j+1}$ である．
- **禁止軌跡**：上記のいずれにも分類されない，すなわち，禁止遷移を少なくとも 1 回通過する軌跡である．形式的には，π が上記のいずれの種類の軌跡でもなく，ある $i \geq 0$ に対して $s_i \xrightarrow{a_i}_{\mathbf{f}} s_{i+1}$ であるとき，π を禁止軌跡と呼ぶ.

次に，2 つの MVTS，特に 4VTS の合併演算を定義する．2 つの 4VTS 間の合併演算は，提案する修正プロセスにおいて，修正モデルの候補となる 4VTS を構築するために用いられる．

合併演算を定義するために，まず，2 つの MVTS の積を定義する．ただし，本論文では，2 つの MVTS が同一の事象集合を持つと仮定する．

定義 9.（MVTS の積）MVTS $MV_1 = (S_1, A, \Delta_1, s_{01}, \mathcal{LT}_1)$ と $MV_2 = (S_2, A, \Delta_2, s_{02}, \mathcal{LT}_2)$ が与えられているとする．ただし，$\mathcal{LT}_1 = (LT_1, \sqcap_1, \sqcup_1)$ と $\mathcal{LT}_2 = (LT_2, \sqcap_2, \sqcup_2)$ は束である．このとき，MV_1 と MV_2 の積 $MV_1 \otimes MV_2 = (S_1 \times S_2, A, \Delta, (s_{01}, s_{02}), \mathcal{LT}_1 \times \mathcal{LT}_2)$ は MVTS である．ただし，$\Delta : S_1 \times A \times S_2 \to LT_1 \times LT_2$ は，次のように定められる遷移関数である．

$$l_1 = \Delta_1(s_1, a, s_1'), l_2 = \Delta_2(s_2, a, s_2') \text{ ならば}, (l_1, l_2) = \Delta((s_1, s_2), a, (s_1', s_2'))$$

■

A, B, C を 3 つの集合とし，$f : A \to B$, $g : B \to C$ を写像とするとき，A から C への合成写像を $g \circ f : A \to C$ と記す．これと定義 9 を用いて，MVTS 間の結合を定義する．これは，モデルの合併演算の定義に用いる．

定義 10.（MVTS の結合）2 つの MVTS $MV_1 = (S_1, A, \Delta_1, s_{01}, \mathcal{LT}_1)$ および $MV_2 = (S_2, A, \Delta_2, s_{02}, \mathcal{LT}_2)$ に対して，MVTS $MV_1 \oslash_{cf} MV_2 = (S_1 \times S_2, A, cf \circ \Delta, (s_{01}, s_{02}), \mathcal{LT}')$ を MV_1 と MV_2 の結合と呼ぶ．ただし，$\mathcal{LT}_1 = (LT_1, \sqcap_1, \sqcup_1)$, $\mathcal{LT}_2 = (LT_2, \sqcap_2, \sqcup_2)$，および，$\mathcal{LT}' = (LT', \sqcap', \sqcup')$ は束である．また，Δ は $MV_1 \otimes MV_2$ の遷移関数の定義に従って定められる遷移関数であり，結合関数 $cf : LT_1 \times LT_2 \to LT'$ は，Δ の値を $\mathcal{LT}'$ の要素に関連付ける写像である．

■

本論文では，結合関数として単調な写像を用いる．MVTS の並列合成は，結合の概念を用いて定義することができる．LTS L_1 と L_2 の並列合成 [74] は $L_1 \parallel L_2 = L_1 \oslash_{cf_{\mathrm{LTS}}} L_2$ と定められる．ここで，$cf_{\mathrm{LTS}} : LT^2 \times LT^2 \to LT^2$ は，$cf_{\mathrm{LTS}}(\mathbf{tt}) = \mathbf{t}$ であり，それ以外は $\mathbf{f}$ であるような結合関数である．MTS の並列合成 [98]，および 4VTS の並列合成も同様の方法で定義することができる．特に，4VTS の並列合成の定義に必要な結合関数 $cf_{4\mathrm{VTS}} : LT^4 \times LT^4 \to LT^4$ を図 3.2 に示す．$cf_{4\mathrm{VTS}}$ は，図中の左端に示した束 $\mathcal{LT}^4 \times \mathcal{LT}^4$ の各要素（ハッセ図における頂点）を，図中央の束 $\mathcal{LT}^4$ の同じ模様の要素に写す写像である．言い換えるならば，$\forall a, b \in LT^4. cf_{4\mathrm{VTS}}(ab) = a \sqcap^4 b$ である．例えば，$(s_1 \xrightarrow{a}_{\mathbf{t}} s_1') \oslash_{cf_{4\mathrm{VTS}}} (s_2 \xrightarrow{a}_{\mathbf{u}} s_2')$ は，$(s_1, s_2) \xrightarrow{a}_{\mathbf{u}} (s_1', s_2')$ によって定義される．

3.1.2 4 値遷移系の合併演算

ここでは，Uchitel 等の提案する MTS に対する合併演算 [98, 97] を拡張することで，4VTS の合併演算子 ++ を定義する．

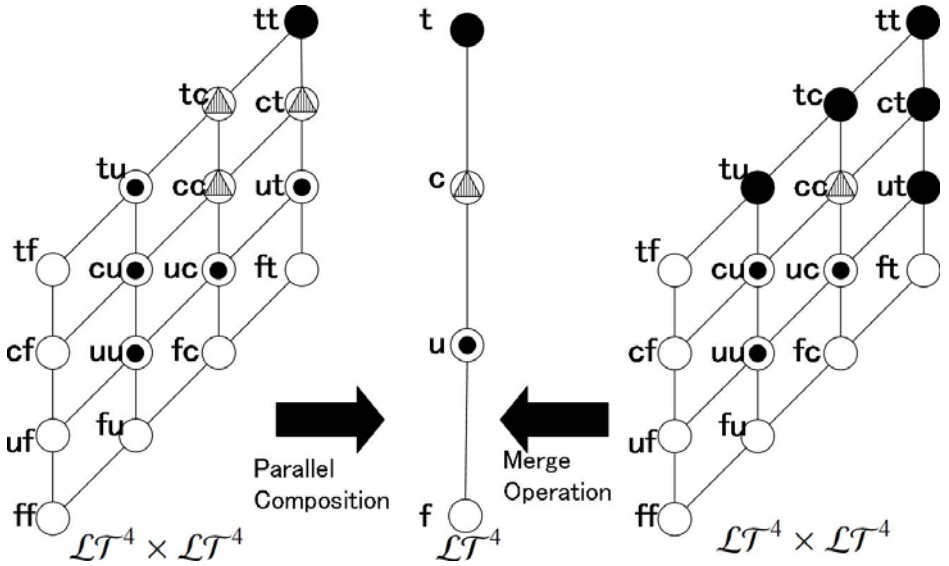


図 3.2. 4VTS の演算

Uchitel 等による合併演算 [98, 97] は，MTS を対象としたものであり，MTSA というツールで実用化されている [38]．合併演算の基本的な考え方は，2 つの MTS を合併することによって，それらの MTS の全ての実行が必須な振舞い（MTS モデルの必須遷移のみ通過する軌跡）と，全ての実行が禁止された振舞い（MTS モデルの必須遷移も可能遷移も通過しない軌跡）を保存する MTS を構成することである．

以上の合併演算を 4 値遷移系に拡張して，4VTS の合併演算子 ++ を定義する．ここで定義する合併演算は，合併を行うモデルの少なくとも 1 つにおいて必須の振る舞いが，合併後のモデルにおいても必須となるように定める．逆に，合併モデルにおいて実行可能な振る舞いは，合併対象モデルのどちらも実行可能であるようにする．ただし，合併対象モデル上で禁止遷移を実行する軌跡（禁止軌跡）は，合併後のモデルでも同様に禁止される．

定義 11.（**4VTS の合併**）$mg : LT^4 \times LT^4 \rightarrow LT^4$ を次のような結合関数とする：$mg(\mathbf{tt}) = mg(\mathbf{tc}) = mg(\mathbf{ct}) = mg(\mathbf{ut}) = mg(\mathbf{tu}) = \mathbf{t}$, $mg(\mathbf{cc}) = \mathbf{c}$, $mg(\mathbf{cu}) = mg(\mathbf{uc}) = mg(\mathbf{uu}) = \mathbf{u}$，それ以外の場合は $\mathbf{f}$ とする．4VTS FV_1 と FV_2 の合併 $FV_1{+}{+}FV_2$ を，$FV_1 \oslash_{mg} FV_2$ なる 4VTS によって定める．

■

合併演算子 ++ により，合併モデル FV は合併対象モデル FV_1 と FV_2 の必須遷移を保存する．すなわち，FV_1 または FV_2 においてある遷移の実行が必須ならば，FV において対応する遷移もまた実行が必須である．逆に，FV_1 または FV_2 は FV の不確実遷移を保存する．すなわち，合併後のモデル FV においてある遷移の実行が不確実ならば，合併対象モデル FV_1 と FV_2 において対応する遷移もまた実行が不確実である．以上を直観的に言い換えると，FV は FV_1 と FV_2 よりも実行の確実性が高い（遷移が実行されることが確定している）

ので，FV は FV_1 と FV_2 をより具体化したモデルである．例外は，$mg(\mathbf{tf}) = mg(\mathbf{ft}) = \mathbf{f}$ の場合であり，FV_1 の必須遷移と FV_2 の禁止遷移（または，FV_1 の禁止遷移と FV_2 の必須遷移）が合併されると，FV の対応する遷移もまた禁止遷移になることを意味する．

定義 11 の結合関数 mg を図 3.2 に図示する．mg は，図中の右端に示した束 $\mathcal{LT}^4 \times \mathcal{LT}^4$ の各要素を，図中央の束 $\mathcal{LT}^4$ の同じ模様の要素に写す写像である．合併演算子 $++$ は，可能遷移または不確実遷移を必須遷移と合併すると，必須遷移が得られるという点で $\oslash_{cf_{4\mathrm{VTS}}}$ と異なる．直観的には，必須遷移は可能遷移と不確実遷移よりも実行の確実性が高いため，合併の結果得られる遷移の実行の確実性もまた両者よりも高いということを意味する．不確実遷移は，可能遷移と不確実遷移を合併することにより得られる．これは，合併対象モデルの遷移が両者共に実行の不確実性を持つ可能遷移や不確実遷移ならば，合併モデルには両者のより実行確実性の低い遷移が対応して現れることを意味している．

3.2　モデル修正問題

本節では，モデル修正問題を定める．モデル修正問題は，LTS で記述された修正対象のモデル L から，L が満たさない性質を満たすモデル L' を求める問題である．本論文で扱うモデル修正問題において，次の条件を満たす FLTL 式で記述された性質の集合 Φ が存在すると仮定する．

Φ の全ての要素は，対象モデル L で満たされることが既に証明されている．

L が満たす性質の集合は，修正後のモデル L' も満たしていなければならない．ゆえに，L と L' は，両者が共に満たす性質からなる基盤モデルが存在することによって関連付けられる．すなわち，L と L' は Φ の全ての要素を満たさなければならないので，基盤モデルは Φ と等価な LTS である．したがって，Φ の各要素は開発対象領域において成り立つ知識を表すと考えることができる．

定義 12.（**モデル修正問題**）L を LTS 全体の集合とする．$L \in \mathsf{L}$ を LTS で記述されたモデル，Φ を，$\forall \phi_t \in \Phi . L \models \phi_t$ であるような FLTL で記述された論理式の集合，そして，ϕ を $L \not\models \phi$ なる FLTL の論理式とする．モデル修正問題は次の条件を満たすモデル $L' \in \mathsf{L}$ を発見する問題である．

$$L' \models \phi \text{ かつ } \forall \phi_t \in \Phi . L' \models \phi_t$$

■

定義 13.（**基盤モデル**）L_1 と L_2 を LTS とする．また，Φ を $\forall \phi \in \Phi . L_1, L_2 \models \phi$ なる FLTL の論理式の集合であるとする．このとき，$\forall \phi \in \Phi . L_c \models \phi$, $Tr(L_1) \subseteq Tr(L_c)$ と $Tr(L_2) \subseteq Tr(L_c)$ を満足する LTS L_c を，Φ についての L_1 と L_2 の基盤モデルという．

■

2 つのモデル L_1 と L_2 の基盤モデル L_c は，L_1 と L_2 が共に満たす性質を満たす．加えて，L_c は，L_1 の振る舞いと L_2 の振る舞いを実行できるモデルである．後者の条件は，以下の想定に基づく．L_1 を修正対象モデル，L_2 を修正によって得られる対象の性質を満たすモデルとすると，L_c は L_1 の振る舞いを含むのみならず，L_2 の振る舞いも含む．よって，L_c から修正対象の性質を満たさない軌跡（反例）を除くことで，L_2 を求めることができる．次節で述べるように，本論文では，Φ に含まれる全ての性質を満たす軌跡の集合をそのテスタオートマトンから求めて，基盤モデルを構成する．構成した基盤モデルはこの条件を満たす．したがって，基盤モデルを用いて対象となる性質を満たすモデルを求めることが可能であると考えられる．

次節において，このモデル修正問題を解く方法を論じる．

3.3　反復型モデル修正手続き

本節では，モデル修正問題を解決する反例に基づいた反復型モデル修正手法を提案する．提案手法への入力は，LTS で記述された修正対象モデル L，L において満たされる性質の集合 Φ，そして L で満たされない性質 ϕ である．ただし，Φ の各性質は安全性で記述されていなくてはならない．一方，ϕ は検証対象の性質であり，安全性でも活性でもよい．

最初に，L 上で ϕ のモデル検査を実施し，反例 $\pi \in Tr(L)$ を得る．提案する手法は，モデル修正問題をモデル合併問題に帰着させる [97]．合併すべきモデルは，以下のようにして得られる Φ から構築した基盤モデルと π から構築した反例モデルである．

- Φ から構築する基盤モデルは，Φ に含まれる全ての安全性を満たすモデルである．この基盤モデルは，各性質の論理積を取った FLTL 式の否定を受理するテスタオートマトンから構成できる LTS である．このモデルは，Φ の各性質を同時に満たす全ての軌跡を表現し，また，ϕ を満たす軌跡も含む．そのような ϕ を満たす軌跡を表現する LTS を基盤モデルから求めることが，本論文で扱う修正作業の目標である．そのために，基盤モデルを，各安全性を満たさない軌跡を禁止遷移によって明示的に禁止し，それ以外の軌跡を必須遷移や可能遷移で表されるように 4VTS に変換する．すなわち，各安全性を満たさない軌跡を禁止軌跡によって表し，満たす軌跡を必須または可能軌跡で表す．以降，構成された 4VTS を FV_Φ と記す．
- π から構築する反例モデルは，π の実行を禁止してそれ以外の軌跡を実行可能とする 4VTS である．このアイデアを実現するため，π 以外の他の軌跡は可能遷移で表現する．また，モデル上で，π において性質違反となる事象を，安全性の場合は禁止遷移で，活性の場合は不確実遷移で表す．活性の場合に，性質違反事象を禁止遷移ではなく不確実遷移で表すのは，性質違反事象の有限回の実行は必ずしも性質違反となるわけではなく，無限回実行のみが禁止されることを明示的に表現するためである．以上より，性質が安全性の場合は禁止軌跡で，また，活性の場合は不確実軌跡で反例を表し，それ以外の軌跡は可能軌跡で表現するモデルを構築する．得られる 4VTS を FV_π と記す．

求めた 2 つの 4VTS FV_Φ と FV_π を定義 11 に基づいて合併することで，FV_Φ において反

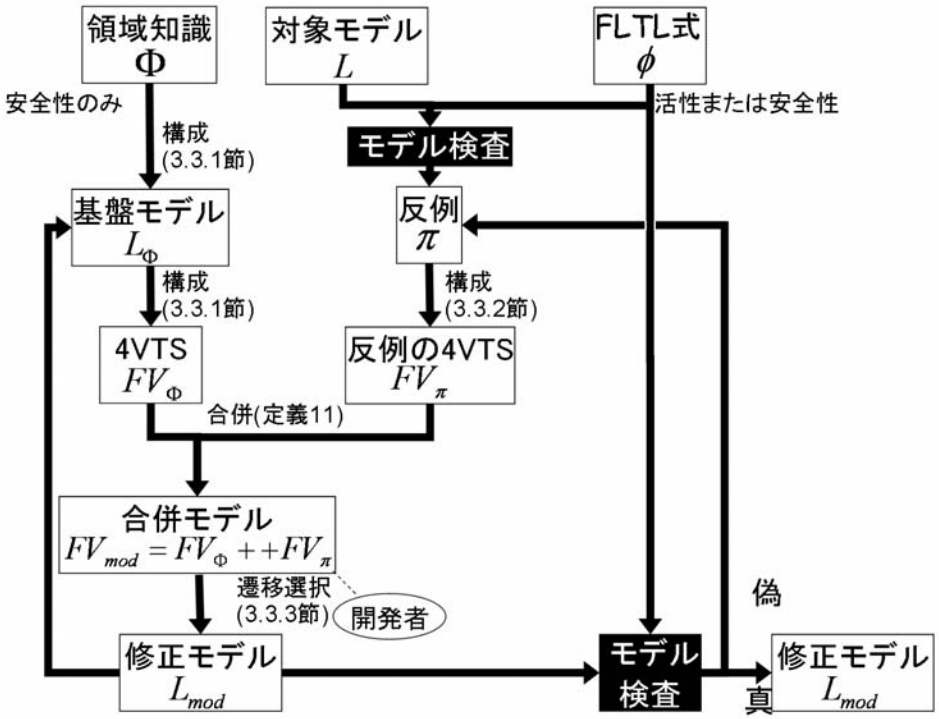


図 3.3. モデル修正プロセス

例 π を除いたモデル $FV_{mod} = FV_\Phi ++ FV_\pi$ を構成する．FV_{mod} は，性質 ϕ を満たす修正モデルの候補を開発者に提示する．開発者は，候補の中から望ましい LTS モデル L_{mod} を選択することで，1 回の修正作業を行う．選択したモデル L_{mod} は，ϕ を満たす軌跡を含むが，反例を除いた後も依然として違反する軌跡を持つ可能性があるので，L_{mod} に対して ϕ のモデル検査を行う．以上の手続きを，L_{mod} が ϕ を満たすか，もしくは，開発者が予め設定した基準を L_{mod} が満足するまで繰返す．ただし，ϕ が複雑な FLTL 式からなる性質である場合，L_{mod} が ϕ を満たすようにするのが容易でない恐れがあるため，後者の開発者が設定した基準が必要である．以上の議論をまとめると，提案するモデル修正プロセスは以下のようになる．

1. Φ から FV_Φ を構成する．
2. モデル検査を実行して $L \models \phi$ であるかどうかを検証し，反例 π を得る．
3. π から FV_π を構成する．
4. 定義 11 を用いて FV_Φ と FV_π を合併し，$FV_{mod} = FV_\Phi ++ FV_\pi$ を求める．
5. FV_{mod} が提示する修正モデルの候補から開発者が望ましいモデルを選択することで，L_{mod} を構成する．そして，L_{mod} を新たな基盤モデルとして FV_Φ を更新する．
6. L_{mod} から反例が存在しなくなる，あるいは，L_{mod} が事前に定義された基準を満たせば終了する．そうでない場合は，ステップ 2 へ戻る．

モデル修正プロセスの実行の流れを図 3.3 に示す．

提案手法は反例に基づく手法である．領域知識から構成された基盤モデルは，領域知識に含

まれる各性質を満たすモデルであるが，このモデルは修正対象モデル L で成り立つ全ての領域知識の定式化であるとは限らず，L の情報を十分に含んでいるとはいえない．したがって，修正を行う際に L の情報を使うことが望ましい．反例 $\pi \in Tr(L)$ は L 上の軌跡なので，L の振る舞いに関する情報も含むと考えられる．したがって，反例を活用することにより，L の情報を合併モデルに対する選択作業で扱うことができる．

ただし，提案するモデル修正法には，複数プロセスから構成される並行システムには適用できないという制約がある．提案手法は，領域知識と反例に基づいてシステムの振る舞いを表すモデルを修正する手法である．システムが複数のプロセスから成る並行システムの場合には，システム全体の振舞いは各プロセスの並列合成によって得られる．しかしながら，領域知識に使われるモデル上で成り立つ性質は，システム全体の振舞いに対する主張であることが多い．それゆえ，個々のプロセスの振舞いを独立して取り扱うことが困難なので，単一プロセスから成るシステムに対してのみ適用可能である．この点は今後の研究で解決すべき課題である．

以下で，各手続きの詳細を説明する．

3.3.1　基盤モデルからの 4VTS の構成

この節では，上に示した修正手続きのステップ 1 について詳しく説明する．ステップ 1 において，修正対象である LTS モデル L と修正の結果得られる LTS モデル L' の基盤モデルを求める．既に述べた通り，この基盤モデルは L 上で満たされている性質の集合 Φ から得られる．まず，基盤モデルを LTS で記述し，その後 4VTS に変換する．4VTS に変換された基盤モデルを FV_Φ と表記する．FV_Φ を構成する手法は，Uchitel 等 [97] が提案した性質から MTS を構成する方法に基づいている．特に，FLTL 式から基盤モデルを構築する方法は文献 [70] で提案された方法を用いる．本論文では，Φ は，FLTL 式の安全性で定式化された修正対象モデルで満たされる性質の集合と仮定する．これは，安全性は状態数が有限の LTS に変換することができるが，安全性以外の性質は一般にはそれが不可能なためである [70].

FV_Φ を構成するために, まず, Φ の全要素の論理積 ϕ_{all} を求める．次に, ϕ_{all} に違反し, かつ, L の事象集合 $A(L)$ の要素からなる無限長の軌跡を受理するテスタオートマトン $TA(\neg\phi_{all})$ を構成する [45]．安全性を表す論理式の論理積もまた安全性であるので, $TA(\neg\phi_{all})$ は，$A(L)$ の全ての要素を遷移ラベルとする自己閉路を有する唯一の受理状態を持つ．したがって，$TA(\neg\phi_{all})$ の受理状態に達すると，その受理状態から異なる状態に遷移することができない．それゆえ，受理状態に達する全ての軌跡は $TA(\neg\phi_{all})$ に受理される．すなわち，受理状態に入る遷移は性質違反の原因なので，このような遷移の実行は禁止されなくてはならない．$TA(\neg\phi_{all})$ からこれらの遷移を除いた後に，初期状態から到達不可能な全ての状態を削除すると，その結果得られるモデルは LTS L_Φ であり，かつ，ϕ_{all} を満たす全ての無限長軌跡を受理する．すなわち，文献 [70] の議論により $Tr(L_\Phi) = Tr(TA(\phi_{all}))$ である．L は ϕ_{all} を満たすので，2.4 章の議論から $Tr(L) \subseteq Tr(TA(\phi_{all})) = Tr(L_\Phi)$ である．よって，修正モデル L_{mod} が $Tr(L_{mod}) \subseteq Tr(L_\Phi)$ を満たすように L_{mod} を求めることで，L_Φ は基盤モデルの条件を満たす．

以上の考察より，基盤モデル L_Φ は文献 [70] に従って，以下の手順で構築される．

1. $\phi_{all} = \bigwedge_{\phi_t \in \Phi} \phi_t$ を求め，Giannakopoulou 等が提案する方法 [45] を用いて $A(L)$ 上のテスタオートマトン $TA(\neg\phi_{all})$ を構築する．
2. $TA(\neg\phi_{all})$ の唯一の受理状態と，その受理状態に入る全ての遷移を除くことで，L_Φ を構築する．

既に述べたように，上の手続きは，安全性にのみ適用可能である [70, 97]．なぜなら，活性を検証する際に構築するテスタオートマトンは，安全性に対して構築するテスタオートマトンとは異なり，そこから脱出不可能な唯一の受理状態を持つとは限らない（2.4 節の図 2.6, ¬EXIT_1 のテスタオートマトンを参照）．したがって，対象となる活性の否定から構築したテスタオートマトンから，上の手続きに従って，全ての受理状態と，各受理状態への遷移を除いたモデルを構築しても，その活性を満たす軌跡の集合を表すモデルとなるとは限らない．ただし，文献 [97] で論じられているように，要求分析で扱われる領域知識は，対象領域で常に成り立つ性質を定式化したものであることが多いので，安全性のみを扱うことは，妥当な仮定であると考えられる．

上の手続きで求められた L_Φ は LTS で記述されている．文献 [70] では，この LTS を構築することが目標である．本論文では 4VTS を扱うので，L_Φ の 4VTS FV_Φ を構成しなければならない．そのために，文献 [97] で提案されている L_Φ から MTS を構成する手法を活用する．Φ の各性質を満たす軌跡は，基盤モデル上の実行可能な軌跡であるので，4VTS の必須軌跡や可能軌跡で表す．この考え方は文献 [97] の MTS の構築法と同様であるが，MTS と異なり，4VTS 上では，実行が禁止された軌跡である禁止軌跡も記述する．そこで，各性質を満たさない軌跡が，基盤モデル上で実行が許されない軌跡であるので，4VTS の禁止軌跡で表す．以上をまとめると，FV_Φ の構成手順は以下の通りである．

3. L_Φ の各状態に対して，その状態が 2 つ以上の出遷移を持つならば，その状態の全ての出遷移を可能遷移に変換する．そうでない場合，その状態の唯一の遷移を必須遷移とする．
4. 吸収状態（sink state）s_s と，全ての $a \in A(L)$ ついて s_s の自己閉路 $s_s \xrightarrow{a}_{\mathbf{f}} s_s$ を，ステップ 3 で構成されたモデルに追加する．
5. ステップ 4 で得られたモデルの各状態 s について，各事象 $a \in A(L)$ を遷移ラベルとする出遷移が存在しないならば，禁止遷移 $s \xrightarrow{a}_{\mathbf{f}} s_s$ を追加する．この結果構築されたモデルを FV_Φ とする．

上の手続きのステップ 5 が必要なのは，定義 8 より，4VTS の各状態が，全ての事象について全体性を満たすという条件を満たすためである．

例 17. 2.2 節で挙げた図 2.2 の三段階スイッチのモデル L_1 と L_2 を考える．2.4 節で述べた通り，L_1, L_2 は共に安全性 ϕ_1, ϕ_2 を満たす．よって，性質の集合を $\Phi = \{\phi_1, \phi_2\}$ とすると，L_1 と L_2 について，上の手続きを用いて FV_Φ を構成することができる．ステップ 1 より，

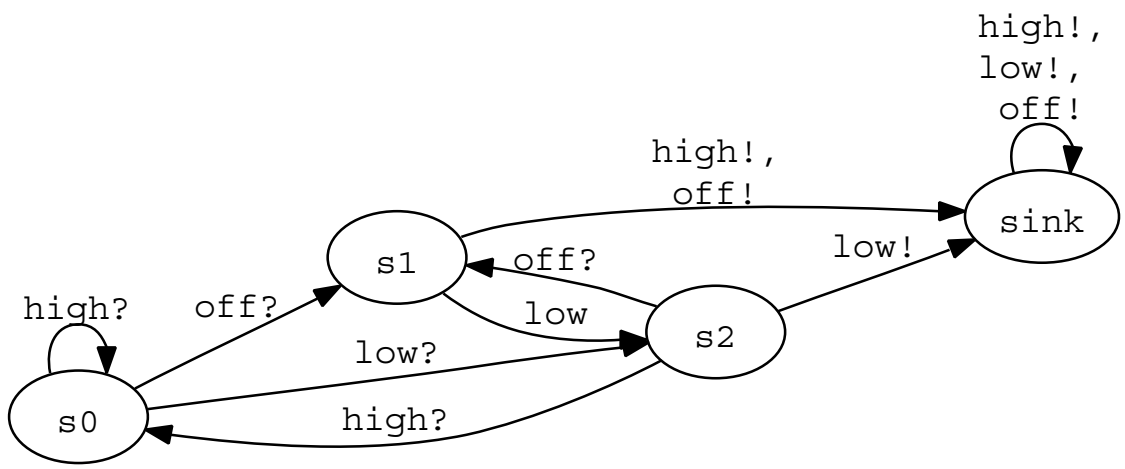


図 3.4. 基盤モデルの 4VTS 表現

$\phi_{all} = \phi_1 \wedge \phi_2$ を得る．この ϕ_{all} に対して，ステップ 2 からステップ 5 を適用することで，図 3.4 に示す 4VTS FV_Φ が構成される．

■

上の例で挙げた図 3.4 のように，今後，4VTS の可能遷移を，遷移ラベルの後ろに疑問符「?」を付加した矢印で図示することとする．同様に，遷移ラベルの後ろに感嘆符「!」を付加した遷移は禁止遷移を表す．必須遷移の場合は，後ろに記号を付加せず単に遷移ラベルのみを持つ矢印として示すこととする．なお，図 3.4 には表示されていないが，不確実遷移の場合は，遷移ラベルの後ろに 2 重疑問符「??」を付加することで図示する（図 3.6 を参照）．

例 18. 図 3.4 の遷移 (s_1, low, s_2) は必須遷移である．4VTS に対するこの記法は以降でも特に断らずに用いる．FV_Φ 上で，軌跡 $[(off)^\omega] = [off,\ off, \ldots]$ や $[(low)^\omega] = [low,\ low, \ldots]$ の実行は禁止されている．この 2 つの軌跡は，それぞれ ϕ_1 と ϕ_2 を満たさず，どちらも禁止遷移を実行するためである．FV_Φ の状態 $sink$ は，上の 4VTS 構成法のステップ 4 で追加した吸収状態である．ステップ 4 において，$sink$ の自己閉路もまた追加される．この状態は，テスタオートマトン $TA(\neg(\phi_1 \wedge \phi_2))$ の受理状態に相当する．ステップ 5 で追加する禁止遷移は，状態 s_1 と s_2 から $sink$ への遷移である．

■

3.3.2　反例からの 4VTS の構成

この節では，モデル L に対して FLTL 式 ϕ のモデル検査を実行した結果得られる反例 $\pi \in Tr(L)$ から，4VTS モデルを構築する方法を説明する．反例 π は性質を満たす修正モデルにおいて実行されてはならないので，ここで構成する 4VTS は π の実行を禁止するモデル

である．モデルを構築するために，以下に述べる π の特徴を利用する．

2.4 節で論じたとおり，Büchi オートマトンの受理条件により，一般に π は，有限長の接頭辞と，それに続く，有限長の事象列からなる閉路の無限回の繰返しからなる軌跡である [67, 12]．ゆえに，P, C を無限列や閉路を部分列として含まない有限長の事象列とし，C^ω を C の無限回の繰返し，P を C への接頭辞とすると，$\pi = PC^\omega$ と表すことができる．ϕ が安全性のときは，π は P のみからなる有限長の軌跡になるが，この場合，C は任意の有限長の軌跡と考えることができる．

以上の考察を用いて，π を構成する各事象を次のように 4VTS の遷移に変換する．構築する 4VTS を FV_π とする．まず，接頭辞 P の各事象は，その生起順序を保存しつつ可能遷移に変換する．これは，反例の接頭辞に性質違反となる誤りが常に含まれているわけではなく，それゆえ，接頭辞の実行を禁止することが，必ずしも反例を禁止したことになるわけではないからである．ただし，ϕ が安全性のときには，P の最後の事象は性質違反を引き起こすので，禁止遷移で表す．次に，C は，不確実遷移あるいは禁止遷移からなる強連結成分をなすようにモデルを構築する．この考えに基づいて，性質 ϕ が安全性と活性のそれぞれの場合について，FV_π を構成する方法を以下で詳細に論じる．

性質が安全性の場合の 4VTS の構築

まず，性質 ϕ が安全性の場合を考える．安全性が持つ特徴により，モデル L が ϕ を満たさないのは，ϕ に違反する望ましくない事象が L において実行される場合である．このような事象は，$TA(\neg\phi)$ の受理状態への入遷移に付加される遷移ラベルに相当する．2.4 節で述べたように，安全性の場合，性質違反の原因となる望ましくない事象は，有限長反例の最後の事象，すなわち，π における P の最後の事象に現れるので，反例の実行を禁止するにはこの事象の出現を許さないことが必要である．ただし，訂正すべき事象が，P の最後の事象に至る過程で起こる事象である場合も考えられる．その点を考慮して，反例に含まれる事象が他の事象に変更可能であるように，反例の実行を禁止するモデルを構築する．

以上より，P は $P = P_0[a_b]$ と二つの部分に分割可能である．ここで，P_0 は，P の有限長の接頭辞であり，a_b は，P_0 に続く P の最後の事象である．加えて，$P_0[a_b]$ にいかなる事象列が続く軌跡もまた性質違反となるので，C は，L が実行する可能性のあるあらゆる軌跡となる．ただし，P_0 の各事象は，上で論じたように，反例が指し示す性質違反の要因となる可能性がある．そこで，他の事象に変更することができるように，その P_0 における出現順序を保存しながら可能遷移でモデル化する．次に，a_b と C を構成する各事象は，禁止遷移によって記述される．また，それ以外の全ての軌跡については実行が許されるので，可能遷移によってモデル化すればよい．以上より，π を部分列として含むあらゆる軌跡は，禁止軌跡によって表される．そして，それ以外の軌跡は可能軌跡で表されることが分かる．

以上の考察から，π から 4VTS FV_π を構築する手続きは以下のようになる．

1. FV_π に初期状態 s_0 を追加する．
2. $P_0 = [a_0,\ a_1, \ldots, a_{m-2}]$ に対して，状態 $s_1, \ldots, s_{m-1}$ と $\forall j \in \{0, \ldots, m-2\}.s_j \xrightarrow{a_j}_{\mathbf{c}}$

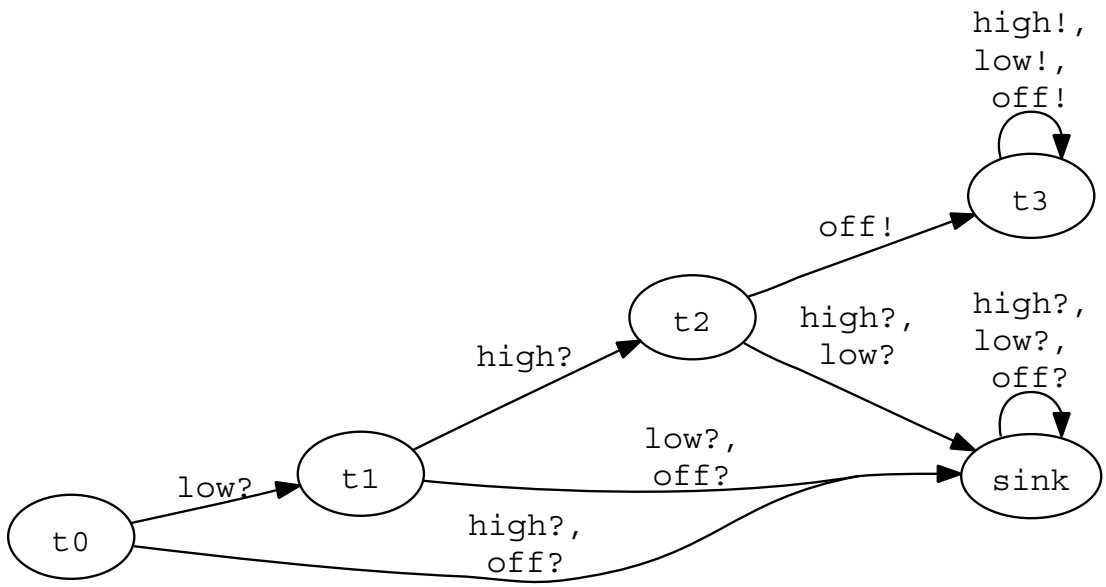


図 3.5. 安全性 ϕ_3 の反例を禁止する 4VTS

s_{j+1} を満たすような遷移を，FV_π に追加する．

3. 状態 s_n と遷移 $s_{m-1} \xrightarrow{a_b}_{\mathbf{f}} s_m$ を FV_π に追加する．
4. 各 $a \in A(L)$ に対して，$s_m \xrightarrow{a}_{\mathbf{f}} s_m$ を満たす遷移を，FV_π に追加する．
5. 新たな吸収状態 s_s を FV_π に加え，各 $a \in A(L)$ に対して，s_s の自己閉路 $s_s \xrightarrow{a}_{\mathbf{c}} s_s$ を追加する．
6. これまで追加した s_s と s_m 以外の各状態 s_j (ただし，$j \in \{0, \ldots, m-1\}$) に対して，各 $a \in A(L)$ についての出遷移 $s_j \xrightarrow{a}_{\mathbf{c}}$ が存在しなければ，吸収状態への遷移 $s_j \xrightarrow{a}_{\mathbf{c}} s_s$ を FV_π に追加する．

例 19. 3 段階スイッチの例を再び考える．図 3.5 に，$\pi_{\phi_3} = [low,\ high,\ off]$ から構成した 4VTS $FV_{\pi_{\phi_3}}$ を示す．状態 t_0 が初期状態であり，これは上の手続きのステップ 1 で追加されたものである．π_{ϕ_3} の最初の 2 つの事象列 $[low,\ high]$ が P_0 をなすので，ステップ 2 において状態 t_1, t_2 と遷移 $t_0 \xrightarrow{low}_{\mathbf{c}} t_1$, $t_1 \xrightarrow{high}_{\mathbf{c}} t_2$ が $FV_{\pi_{\phi_3}}$ に付加される．π_{ϕ_3} の最後の事象 off が反例の最後の事象 a_b に相当するので，ステップ 3 で状態 t_3 とそこへの入遷移 $t_2 \xrightarrow{off}_{\mathbf{f}} t_3$ が生成される．また，t_3 の自己閉路はステップ 4 で得られる．$sink$ が吸収状態であり，ステップ 5 と 6 で自身の自己閉路を含む t_3 以外の各状態からの出遷移が追加される．

■

性質が活性の場合の 4VTS の構築

性質 ϕ が活性の場合の反例を禁止する 4VTS の構築法も，安全性の場合と同様の考え方に基づく．ただし，活性の場合は，2.4 節で説明した公平な選択をモデル検査に際して仮定することがある．そこで，ϕ に対して公平な選択を仮定する場合とそうでない場合に分けて考察する．まず，公平な選択を仮定する場合を考える．この場合，2.4 節で説明したように，反例 $\pi = PC^{\omega}$ の P は，$TA(\neg\phi)$ の受理状態を含む終了状態集合に達する軌跡であり，C はその終了状態の集合を通過する閉路である．次に，公平な選択を仮定しない場合，π は終了状態に限らず，受理状態を含む強連結成分を通過する軌跡であればよいので，P はその強連結成分に達する軌跡で，C はその強連結成分の各状態を通過する閉路である．したがって，公平な選択を仮定する場合もそうでない場合も，同様の方法を用いて 4VTS を構成することができる．

P の実行，および C の有限回の繰り返しは，必ずしも ϕ に違反し，禁止されるべき事象列となるわけではない．ゆえに，C が無限回繰返し実行される軌跡のみを禁止するようにモデルを構成すればよい．このことは，安全性のときのように C において全ての事象を禁止遷移でモデル化することは，活性の場合は強すぎる仮定であることを意味する．そこで，C の各事象を，その実行の可能性が少なく，有限回の実行のみが許されると解釈される不確実遷移でモデル化する．他の軌跡は，安全性の場合と同様に可能遷移によってモデル化する．

以上より，構成される 4VTS において，π は不確実軌跡によって表され，また，それ以外の軌跡は可能軌跡で表される．

以上の議論により，π を禁止する 4VTS FV_{π} を以下の手順で構築する．

1. FV_{π} に初期状態 s_0 を追加する．
2. $P = [a_0,\ a_1, \ldots, a_{m-1}]$ に対して，状態 $s_1, \ldots, s_m$ と $\forall j \in \{0, \ldots, m-1\}.s_j \xrightarrow{a_j}_{\mathbf{c}} s_{j+1}$ を満たすような遷移を，FV_{π} に追加する．
3. $C = [b_0,\ b_1, \ldots, b_{n-1}]$ に対して，以下の手続きを実行する．
   (a) $n > 1$ ならば，状態 $s_m, \ldots, s_{m+n-1}$ と $\forall j \in \{m, \ldots, m+n-2\}.s_j \xrightarrow{b_{j-m}}_{\mathbf{u}} s_{j+1}$ を満たす遷移を FV_{π} に追加する．
   (b) 遷移 $s_{m+n-1} \xrightarrow{b_{n-1}}_{\mathbf{u}} s_m$ を FV_{π} に追加する．
4. 新たな吸収状態 s_s を FV_{π} に加え，各 $a \in A(L)$ に対して，s_s の自己閉路 $s_s \xrightarrow{a}_{\mathbf{c}} s_s$ を追加する．
5. これまで追加した s_s 以外の各状態 s_j (ただし，$j \in \{0, \ldots, m+n-1\}$) に対して，各 $a \in A(L)$ についての出遷移 $s_j \xrightarrow{a}_{\mathbf{c}}$ と $s_j \xrightarrow{a}_{\mathbf{u}}$ のいずれも存在しなければ，吸収状態への遷移 $s_j \xrightarrow{a}_{\mathbf{c}} s_s$ を FV_{π} に追加する．

例 20. 3 段階スイッチの性質 ϕ'_3 を考える．図 3.6 は，反例 $\pi_{\phi'_3} = [low,\ high\ (high)^{\omega}]$ より構成した 4VTS である．初期状態は u_0 で，安全性の場合と同様にステップ 1 で生成される．ステップ 2 において，$P = [low,\ high]$ は，u_0 から u_2 までの可能遷移に変換される．ステップ 3(b) により，$C = [high]$ は，u_2 において自己閉路をなす不確実遷移として表現される．ここ

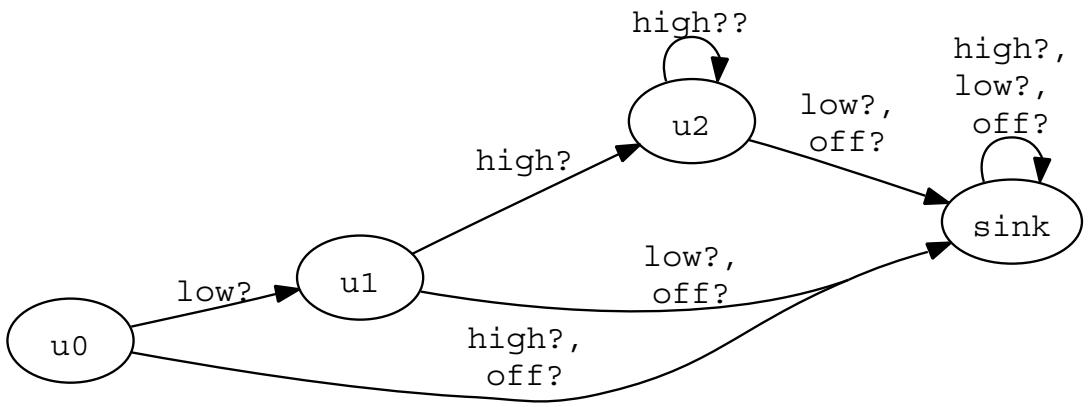


図 3.6. 活性 ϕ_3' の反例を禁止する 4VTS

で，遷移ラベルの後ろに 2 重疑問符「??」を付加した遷移は，不確実遷移を表す．なお，この例では $n = 1$ なので，ステップ 3(a) は条件を満たさず，実行されることはない．最後に，$sink$ が吸収状態として追加され，π_{ϕ_3} 以外の全ての軌跡が，$sink$ への可能遷移として表現される．

■

3.3.3　合併によるモデル修正

これまで，基盤モデルから構成した 4VTS FV_Φ と，反例 π を禁止する 4VTS FV_π を求める方法を述べた．続いて，これらを用いて対象モデル L を修正する方法を述べる．

まず，定義 11 の合併演算を用いて，FV_Φ と FV_π を合併する．その結果得られるモデルは，$FV_{mod} = FV_\Phi{+}{+}FV_\pi$ と表される．以降，FV_{mod} を合併モデルと呼ぶ．合併演算の規則より，FV_{mod} は FV_Φ と FV_π の禁止遷移を保存し，禁止遷移と合併されなければ必須遷移も保存する．また，FV_π の不確実遷移は，合併対象となる FV_Φ の遷移が可能遷移であるならば，FV_{mod} の対応する遷移も不確実遷移である．よって，性質 ϕ が安全性のときは，FV_{mod} は反例 π，つまり実行が禁止されるべき事象列を禁止遷移によって表現する．同様に，ϕ が活性のときには，無限回実行が禁止される反例の閉路を不確実遷移によって表現する．それ以外の Φ の各要素を満たす軌跡は，必須遷移または可能遷移で与えられる．FV_{mod} の可能遷移と不確実遷移は，その実行が起こる可能性の高さを表すと解釈される遷移であるので，FV_{mod} は，修正の結果得られるべきモデルの候補を表すと考えられる．

ここまでの議論を整理すると，FV_{mod} は以下に列挙する特徴を持つ．

- FV_{mod} の必須遷移と可能遷移のみを実行した軌跡は，FV_Φ の必須遷移と可能遷移のみを実行した軌跡である．すなわち，FV_Φ は，FV_{mod} において必須遷移と可能遷移のみを実行する軌跡を保存する．言い換えると，FV_{mod} の必須軌跡と可能軌跡は，それぞれ FV_Φ の必須軌跡と可能軌跡であるので，FV_Φ は FV_{mod} の必須軌跡と可能軌跡を保

存する．したがって，FV_{mod} 上のこれらの軌跡は Φ の要素である全ての性質を満たす．

- FV_{Φ} の必須遷移と可能遷移に加えて，禁止遷移を実行した軌跡は，FV_{mod} 上でもやはり禁止遷移を実行する軌跡である．さらに，検証対象の性質 ϕ が安全性の場合，反例 π は FV_{mod} 上でも禁止遷移を実行する．以上の議論により，FV_{mod} は，FV_{Φ} において禁止遷移を実行する軌跡を保存する．言い換えると，FV_{Φ} の禁止軌跡は FV_{mod} 上でも禁止軌跡であり，また，安全性に対する反例 π を部分列として含む軌跡も FV_{mod} 上で禁止軌跡である．
- FV_{mod} の必須遷移と可能遷移に加えて，不確実遷移を実行した軌跡は，FV_{Φ} において必須遷移と可能遷移を実行した軌跡，またはそれらに加えて禁止遷移を実行した軌跡となる．特に，ϕ が活性の場合，π が FV_{Φ} 上で必須遷移と可能遷移を実行する軌跡ならば，FV_{mod} 上では，必須遷移と可能遷移を実行するのみならず，不確実遷移を無限回通過する軌跡または禁止遷移を実行する軌跡で表される．言い換えると，π が FV_{Φ} 上で可能軌跡ならば，FV_{mod} 上では不確実軌跡または禁止軌跡で表される．

以上より，FV_{mod} 上で π は不確実軌跡あるいは禁止軌跡で表現される．そして，それ以外の軌跡の集合は FV_{Φ} と同一である．ゆえに，FV_{mod} は Φ の各性質を満たす軌跡から π を除いた軌跡を表現したモデルであると解釈できる．

以上の議論に基づき，開発者は，FV_{mod} の可能遷移と不確実遷移のうち，開発対象システムにおいて望ましい振る舞いを表すとみなされるものを選択し，必須遷移に変更する．FV_{mod} の不確実遷移は有限回の実行のみが許される遷移であると解釈するので，修正作業の際に，開発者は，FV_{mod} の不確実遷移によって構成される閉路が有限回実行されるように，必要に応じて FV_{mod} の状態と遷移を展開する．開発者が開発対象システムに不要あるいは実行されてはいけないと判断した可能遷移と不確実遷移は，修正後のモデルの遷移として存在してはいけない．よって，禁止遷移に変換することで，その通過を許さないようにする．ただし，必須遷移は，FV_{Φ} と FV_{mod} の各状態において禁止遷移以外の唯一の遷移であることを意味し，禁止遷移は π または Φ の各性質を満たさない軌跡である．ゆえに，FV_{mod} の必須遷移と禁止遷移は，修正後のモデルにおいてその必要性が確定しているので，これらの遷移を他の遷移に変更してはいけない．

以上の遷移の選択，変更操作は，FV_{mod} から禁止軌跡，不確実軌跡を除き，可能軌跡の中から開発者が必要と判断した軌跡を残すことに相当する．ただし，必須軌跡は修正によって得られるモデルに含まれなければならないので，FV_{mod} から除いてはならない．

LTS L_{mod} が修正の結果得られるモデルであるとすると，以上のような開発者による意思決定を FV_{mod} に対して行った後，L_{mod} は次の手続きによって求められる．

1. FV_{mod} に必須遷移のみを残し，それ以外の全ての遷移を取り除く．
2. 初期状態から到達不可能な全ての状態を取り除く．

L_{mod} は，以下に述べる特徴を持つ．

- FV_{Φ} は FV_{mod} の必須軌跡と可能軌跡を保存するので，FV_{Φ} の構成法より，そのような

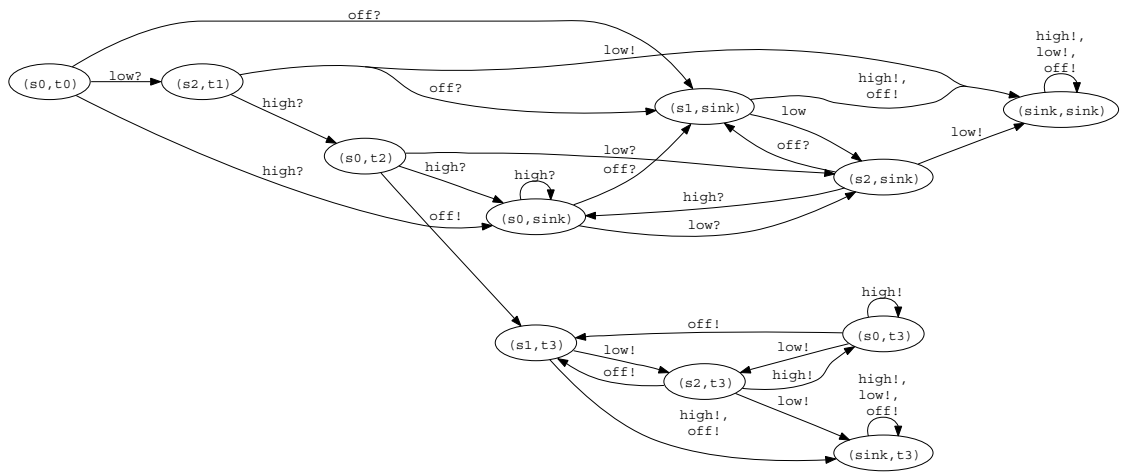


図 3.7. ϕ_3 について合併したモデル

軌跡は領域知識 Φ に含まれる各性質を満たす．よって，FV_{mod} の可能遷移を必須遷移に変更したことで得られる L_{mod} 上の軌跡もまた，Φ の各性質を満たす．また，FV_{mod} の可能遷移を禁止遷移に変更した場合は，モデルから Φ の各性質を満たし，かつ，変更した遷移を実行する軌跡を除くことを意味する．ただし，L_{mod} が領域知識を満たすためには，基盤モデルの条件 $Tr(L_{mod}) \subseteq Tr(L_\Phi)$ を維持するように遷移の変更を施す必要がある．そのためには，例えば，必須軌跡の削除をするような遷移の変更を実施してはいけない．

- FV_{mod} 上で必須軌跡と可能軌跡の集合は，領域知識を満たす全ての軌跡を表す．よって，ϕ が領域知識と矛盾しなければ，ϕ を満たす必須軌跡あるいは可能軌跡を含む．また，上の議論より，FV_{mod} は FV_Φ から π を除いたモデルと解釈されるので，L_{mod} は ϕ を満たさない軌跡を除いたモデルとみなせる．

得られた L_{mod} は，領域知識から反例を除き，$Tr(L_{mod}) \subseteq Tr(L_\Phi)$ を満たすように遷移を変更したモデルであるが，ϕ に違反する軌跡を依然として含む可能性がある．そこで，最後に，L_{mod} に対して再度 ϕ のモデル検査を実施する．$L_{mod} \models \phi$ であるか，もしくは，L_{mod} が開発関係者間で予め定めた基準を満たしているならば，修正プロセスを終了する．後者の基準の例には，状態数の爆発に対処するための状態数の制限が挙げられる．L_{mod} が両者を共に満たしていない場合は，基盤モデル L_Φ を $L_\Phi = L_{mod}$ とし，その 4VTS FV_Φ を更新して，再度モデル修正手続きを実行する．4VTS を構築する反例には，L_{mod} に対して検査を行った結果得られる反例を用いる．以上のように，反復的に修正手続きを実施することによって，性質を満たし，かつ，開発者にとって望ましいモデルを求めることができる．

例 21. 3 段階スイッチの例について，L_1 の修正を考える．$FV_{mod} = FV_\Phi{+}{+}FV_{\pi_{\phi_3}}$ を図 3.7 に示す．図で，(s_0, t_0) は FV_{mod} の初期状態を意味する．状態 $(sink, sink)$, (s_0, t_3), (s_1, t_3), (s_2, t_3), $(sink, t_3)$ に入る全ての遷移は禁止遷移である．この図から分かるように，反例 π_{ϕ_3}

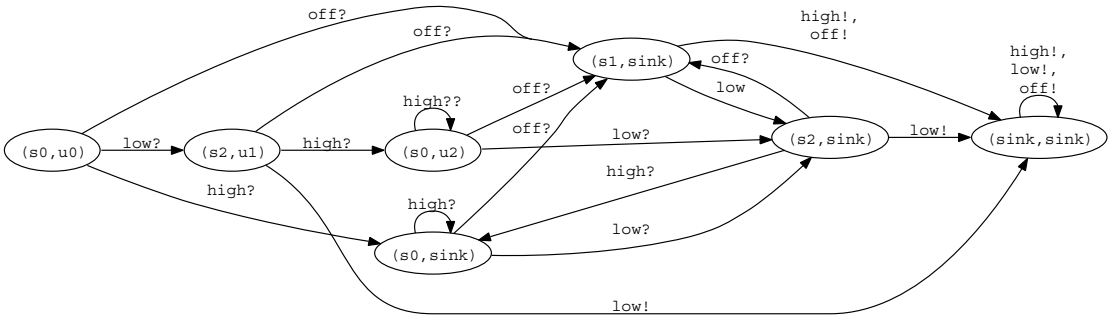


図 3.8. ϕ_3' について合併したモデル

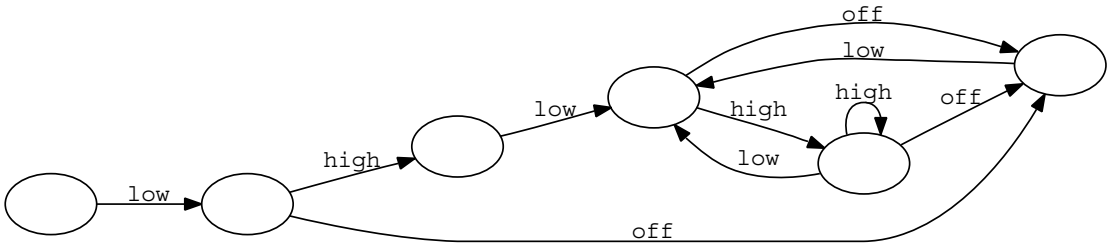


図 3.9. モデル L_1 の ϕ_3 についての修正結果

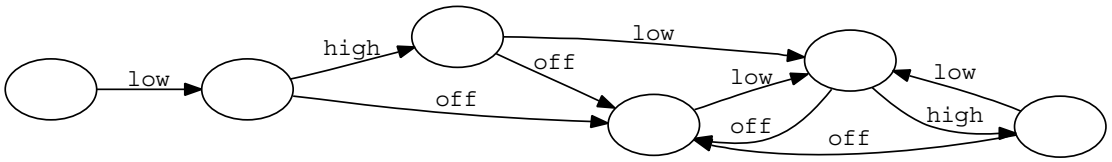


図 3.10. モデル L_2 の ϕ_3' についての修正結果

の 3 番目の事象 *off* は，(s_0, t_2) から (s_1, t_3) への禁止遷移となっているので，π_{ϕ_3} は FV_{mod} で実行することが禁止されている．そして，反例以外の対象領域の知識を表す性質 $\phi_1 \wedge \phi_2$ を満たす軌跡は，必須遷移と可能遷移によって記述されている．

同様に，L_2 について，図 3.8 に $FV_\Phi{+}{+}FV_{\pi_{\phi_3'}}$ を示す（初期状態は (s_0, u_0)）．状態 (s_0, u_2) は，不確実遷移の自己閉路を持つ．これは，反例 $\pi_{\phi_3'}$ の閉路を表しており，開発者はこの自己閉路を何回実行するかを決定しなくてはいけないということを，モデル上で表現している．

これらの 2 つのモデルについて，開発対象システムにおいて望ましい振る舞いをなす遷移を開発者は選択し，必須遷移に変更して修正後のモデルを得る．図 3.7 に対してこの変更操作を行った結果求められた L_1 の修正モデルを図 3.9 に示す．同様に，図 3.8 に変更操作を行い，求められた L_2 の修正モデルを図 3.10 に示す．図 3.9 と図 3.10 のどちらのモデルも，入遷移を持たず，*low* による出遷移のみを持つ左端の状態が初期状態である．図 3.10 において公平な選択を仮定した場合，このモデル中の事象 *off* を遷移ラベルとする遷移が無限回選択される軌跡が検証対象となるので，このモデルは ϕ_3' を満たすことが分かる．よって，修正手続きは

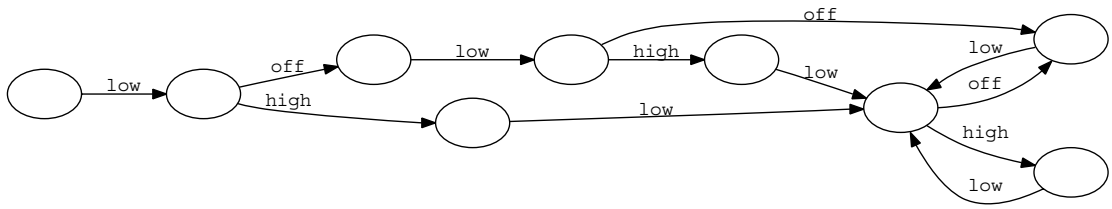


図 3.11. モデル L_1 の ϕ_3 についての 2 回目の修正結果

成功し，終了する．しかしながら，図 3.9 のモデルは依然として ϕ_3 を満たさない．これは，*high* の後に *high* と *low* のみを実行する軌跡がモデル中に存在するためである．よって，モデル検査を行うことにより，新たな反例 [*low*, *off*, *low*, *high*, *off*] が得られる．そのため，この反例と図 3.9 を用いて，再び修正プロセスを実行する必要がある．この例の場合，ϕ_3 を満たす新しいモデルが，2 回目の修正プロセスを実施した結果求められた．修正結果である LTS を図 3.11 に示す（左端の状態が初期状態）．

■

3.4　事例研究

本節では，開発したモデル修正法を適用した電子レンジシステムの事例 [25] について報告する．この事例が紹介されている文献 [25] において，モデルは状態と遷移に事象がラベル付けされたクリプキ構造で与えられている．そこで，われわれはモデルの各状態のラベルについては考慮せず，遷移ラベルに注目することで，本手法が対象とする LTS を構築した．図 3.12 に作成した電子レンジシステムの LTS を示す．図 3.12 で，初期状態は最も上に描かれた状態である．この電子レンジシステムは，初期状態，すなわち時点零において扉が開いた状態にある．次に，扉を閉める（*closeDoor*）か，または，扉を閉じることなく調理を開始する（*startOven*）かのどちらかを実行する．前者は正常な実行手順であり，その後調理を開始することで，ユーザは内容物を安全に調理した後に扉を開けて取り出すことができる．一方，後者の場合は，安全を確保するために，調理を実行せずに扉が閉じられるのを待つ．

この事例に本手法を適用するに際しては，モデルの合併およびモデル検査を実行するためのソフトウェアツールとして MTSA [38] のプロトタイプを活用した．ただし，モデル修正のために，4VTS を構築する作業，そして，遷移の種類を変更して LTS を抽出する作業は手作業で行った．

われわれは，FLTL 式で記述した以下のような性質の集合を想定した．

1. 電子レンジの扉は，加熱中，リセット処理中，および調理中には閉じられている．
2. 電子レンジシステム（の調理室）は，調理している間，熱せられている．
3. 調理中は，電子レンジに対して調理開始を実行することはない．

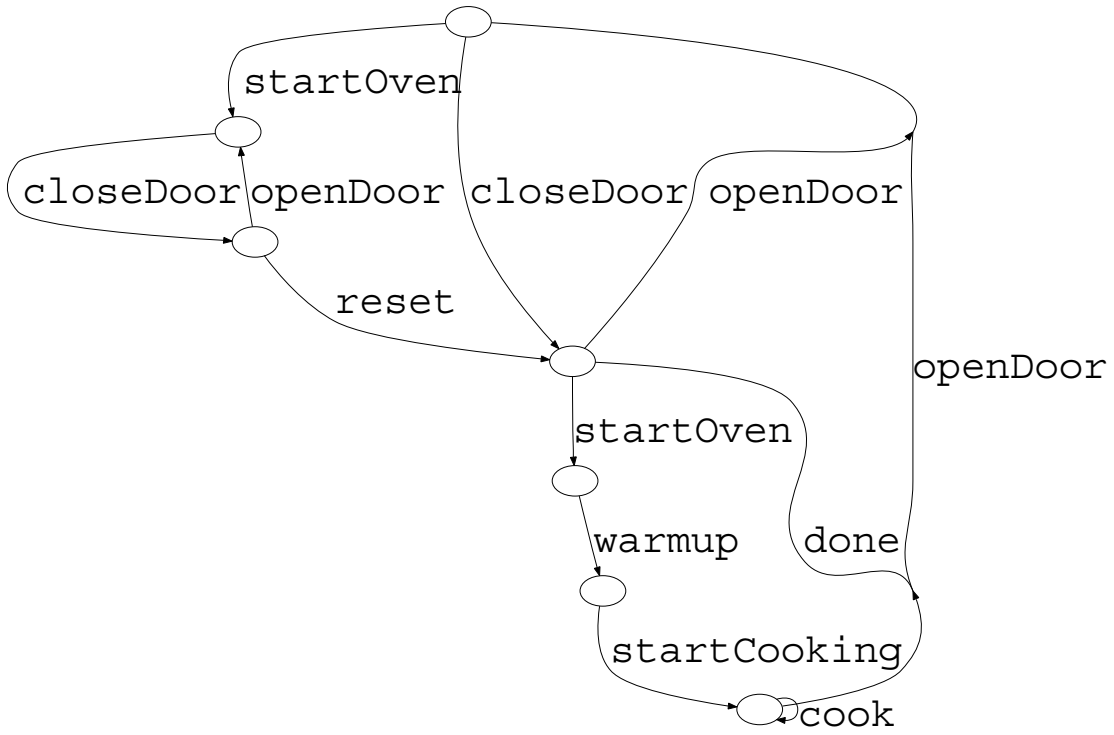


図 3.12. 電子レンジシステム

以上の性質 1 から 3 は安全性として定式化した．最初に，各性質についてモデル検査を実行して，図 3.12 のモデルにおいて性質 1 から 3 が満たされることを確認した．そこで，これらの安全性は，事例研究において領域知識とした．

続いて，同様に，以下の性質 4 と 5 を用意した．

4. 調理が行われる（*cook*）次の時点において，扉が開かれることはない．
5. 扉が開いている間に電子レンジが調理を開始すると，リセットされるまでの間，調理開始中の状態に留まって加熱を行う状態に進むことはない．

性質 4 は安全性として，そして性質 5 は活性として FLTL で定式化した．しかしながら，図 3.12 のモデルは性質 4 と，公平な選択を仮定しない場合に性質 5 も満たさず，それぞれについて反例が得られた．そこで，提案手法を用いて，図 3.12 を，性質 4 と 5 を満たすように修正した．ただし，性質 1 から 3 は図 3.12 の領域知識として用いることで，修正の結果得られるモデルもまた性質 1 から 3 を満たすように修正を施した．

修正作業の実施手順は以下の通りである．

1. 性質 1 から 3 を領域知識とみなして基盤モデルを構築した．
2. 基盤モデルと性質 4 についての反例を用いて，図 3.12 を，性質 4 を満たすように修正した．
3. ステップ 2 で得られた修正モデルを改めて性質 5 についての修正対象モデルとみなし，

性質 4 を新たに対象領域知識に加えた．そして，性質 5 を満たすようにモデルを修正した．

性質 4 と 5 についての修正工程において，反復毎に求められる LTS を求める基準として，遷移の種類の変更後に得られる LTS と修正前の元のモデルとの構造的な類似性と，各反復で得られる反例同士の類似性を用いた．ただし，これらの類似性は定量的に求めたのではなく，人手で判断した．このようにして修正を行った結果，得られた性質 4 と 5 を満たす修正モデルを図 3.13 に示す．ステップ 2 の性質 4 を満たすための修正には 2 回の反復が，ステップ 3 の性質 5 を満たすための修正には 3 回の反復がそれぞれ必要であった．以上の結果，提案手法を適用したことにより，性質 1 から 5 を全て満たすモデルを求めることができた．

なお，修正作業の負荷を緩和するために，以下のような発見的な最適化措置を講じた．

- 基盤モデルから構成した 4VTS の記述に禁止遷移を含めなかった．定義 11 より，この 4VTS の禁止遷移と反例モデルの遷移を合併結果は常に禁止遷移となる．また，基盤モデルの 4VTS においてある事象についての遷移が存在しないことを，LTS や MTS と同様に，その遷移の実行が禁止されていると解釈することができる．そして，禁止遷移を含む 4VTS を用いてモデルの合併演算を行った結果得られる合併モデルにおいて，禁止遷移を実行しない軌跡の集合は，禁止遷移を除いた 4VTS から得られる合併モデル上の軌跡の集合と同一であると解釈をすることができる．したがって，基盤モデルから構成した 4VTS から禁止遷移を除いても，開発者による修正作業に影響はない．
- 上に述べた性質に加えて，修正候補となる 4VTS の規模を抑えるためのいくつかの性質を基盤モデルに導入した．例えば，電子レンジの扉を閉じるという事象が連続して発生することはない（常に，*closeDoor* の次の時点には *closeDoor* とは異なる事象が起こる）という性質が挙げられる．これは，対象領域についての新たに知識を追加することに相当する．したがって，基盤モデルと反例から構成した 4VTS の合併モデルに対して開発者が行う修正作業が容易になる．

3.5　モデル修正法の改善

3.4 節の事例研究において，われわれは修正対象モデルと類似した修正モデルを合併モデルから選択した．これは，修正対象モデルを元にして修正後のモデルが得られるので，両者のグラフ構造は互いに類似するという想定に基づいている．この想定は，先行研究 [37, 107] でも用いられており，修正モデルを得るための支援としてモデル間の類似性を導入することは有効であると思われる．しかしながら，事例研究では，類似度を数値的に測定する指標を用いることはせず，経験的に判断した．そこで，本節では，グラフ間の類似度を数値的に評価する手法の導入によるモデル修正法の改善について論じる．

グラフ間の類似度は，主にパターン認識の分野で構造に基づく類似度の指標に関する研究が行われており，以下のようなグラフ間の距離が提案されている．

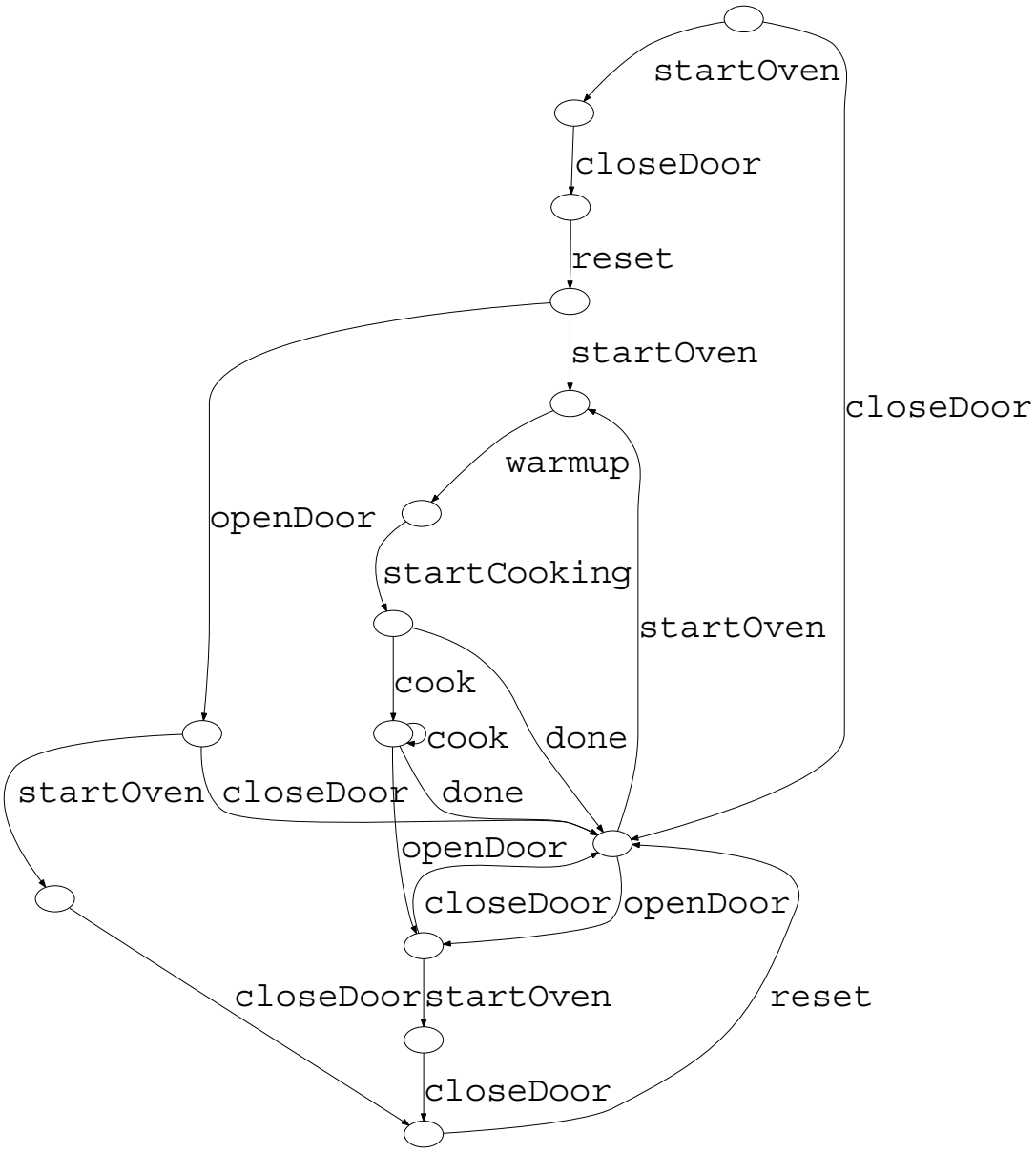


図 3.13. 電子レンジシステムの修正モデル

- **グラフ間の編集距離 [17]**：一方のグラフを他方のグラフに変換するために必要な編集操作のコストの最小値である．ここで，グラフの編集操作とは，頂点と辺についての挿入，置換，削除である．
- **最大共通部分グラフに基づく距離 [18]**：比較する 2 つのグラフの頂点数の最大値に対するこれらの最大共通部分グラフの頂点数の割合を，1 から引いた値によって定義される．直観的には，最大共通部分グラフとは，2 つのグラフのどちらの部分グラフとも同じ構造を持つグラフ，つまり，2 つのグラフに共通に存在する部分グラフの内，最も頂点の数が多いものである．この距離は，最大共通部分グラフの頂点数が大きい（比較対象グラフが同じ構造を共有している）ほど，また，比較する 2 つのグラフの頂点数が少

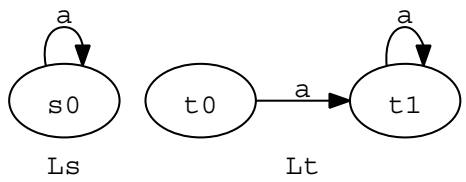


図 3.14. 重み付き量化模倣性の有効性

ないほどそれらの距離が近いと判定する．また，この研究を拡張した距離が以下に示すように提案されている．

- 最小共通拡大グラフの規模（ここでは辺の数と頂点数の和）と最大共通部分グラフの規模との差 [40]．ここで，最小共通拡大グラフとは，直観的には比較する 2 つのグラフを共に部分グラフとして含む最小のグラフである．
- 最大共通部分グラフに基づく距離において，比較する 2 つのグラフの頂点数の最大値ではなく，2 つのグラフの和に相当するグラフが持つ頂点の数を用いる距離 [102]．この変更点をより正確に述べると，両者の頂点数の和からそれらの最大共通部分グラフの頂点数の差を取った値である．

以上のグラフ間の距離は，グラフ間の静的な構造の違いを定量化した指標である．しかしながら，本研究で用いる LTS はシステムの振る舞いを記述することを目的とする．システムの振る舞いは LTS が表現する軌跡の集合によって表現されるが，この軌跡の集合はグラフの構造に依存するわけではない．したがって，上に挙げた指標が正確に LTS の振る舞いの類似性や距離を反映しているとはいえない．このような考え方を基に，Sokolsky, Kannan, Lee は，模倣関係 [99, 5] に基づくグラフ間の類似度を提案した [92]．模倣関係とはグラフの 2 つの頂点間で定められる反射的かつ推移的な二項関係である．直観的には，一方の頂点からはじまる振る舞いが，他方の頂点からはじまる振る舞いをステップ毎に模倣するときに成り立つ頂点間の二項関係をいう．Sokolsky, Kannan, Lee が提案した類似度は，2 つの頂点の間に模倣関係が成り立つならば類似度が最も高いと判定し，そうでない場合，類似度が低いと判定する．これは，振る舞いについての類似性をより反映した指標であると思われる．

3.5.1　模倣関係に基づく類似度

ここでは，文献 [92] の議論に基づいて，Sokolsky 等が提案した類似度である重み付き量化模倣性 (q-模倣性) とモデル修正技術におけるその有効性を述べる．まず，振る舞いについての類似度の必要性を議論するため，以下の例を考える．

例 22. 図 3.14 に 2 つの LTS L_s と L_t を示す（初期状態をそれぞれ s_0, t_0 とする）．上で挙げた 2 種類の構造に基づく距離は以下の通り求められる．

- グラフ間の編集距離：L_s を L_t に変更するには，例えば，以下に示す 2 種類の編集操作

が考えられる．

– 状態 t_1 とその自己閉路を追加し，次に，s_0 の自己閉路を t_1 への遷移に変更する．

– 状態 t_0 と t_0 から s_0 への遷移を追加し，初期状態を t_0 に変更する．

このような各編集操作に必要なコストの和を，おのおのの操作列について求めて，それらの最小値を求めると，その値が編集距離である．

- 最大共通部分グラフに基づく距離：初期状態を考慮した場合の L_s と L_t の最大共通部分グラフは状態 s_0（あるいは t_0）のみから成り，辺のないグラフである．よって，両者の相違は L_t の状態 t_1 にあると判断される．

一方，L_s と L_t は同一の軌跡を表現した LTS であるので，同一の振る舞いを示すと考えられる．よって，システムの振る舞いの相等性に注目するならば，両者は同値であると判断されることが望ましい．しかしながら，以上の議論より，これまで提案されている構造に基づく距離に従うと両者の距離は 0 でなく，それゆえ相違があると判定される．このことから，構造に基づく距離を尺度とすると，L_s と L_t の振る舞いの相等性を適切に表現できないことが分かる．この点を解決するには，L_s と L_t の振る舞いを反映した尺度を基にして類似度を測定する必要がある．

■

上の例で述べた LTS の振る舞いについての関係に模倣関係が知られている [99, 5].

LTS $L =< S, A, \Delta, s_0 >$ において，二項関係 $R \subseteq S \times S$ が以下の条件を満たすとき，R を模倣関係という：$(s,t) \in R$ かつ $s \xrightarrow{a} s' \in \Delta$ ならば，ある $t' \in S$ に対して $t \xrightarrow{a} t' \in \Delta$ が存在し，かつ，$(s',t') \in R$ である．状態 s と t の間に模倣関係 R が存在するとき，t は s を模倣するという．また，s が t を模倣し，かつ，t が s を模倣するとき，s と t は模倣等価である，あるいは模倣等価性を満たすという．

ここでは，1 つの LTS の状態間の模倣関係を述べたが，この関係は 2 つの LTS の状態間の関係に拡張される．LTS $L_i =< S^i, A^i, \Delta^i, s_0^i > (i = 1, 2)$ において，模倣関係 $R \subseteq S^1 \times S^2$ を以下のように定める：$(s,t) \in R$ かつ $s \xrightarrow{a} s' \in \Delta^1$ ならば，ある $t' \in S^2$ に対して $t \xrightarrow{a} t' \in \Delta^2$ が存在し，かつ，$(s',t') \in R$ である．加えて，$(s_0^1, s_0^2) \in R$ ならば，L_2 は L_1 を模倣するという．L_1 が L_2 を模倣し，かつ，L_2 が L_1 を模倣するとき，L_1 と L_2 は模倣等価である，あるいは模倣等価性を満たすという．

図 3.14 では $R = \{(s_0, t_0), (s_0, t_1)\}$ であり，$(s_0, t_0) \in R$ なので L_t は L_s の振る舞いを模倣する（同様に，L_s も L_t の振る舞いを模倣する）．よって，模倣関係を用いることで，LTS 間の振る舞いの類似性を的確に反映できると考えられる．この模倣関係に基づいて定められた類似度が q-模倣性 [92] である．

ある媒介変数 $0 < p < 1$ に対して，q-模倣性を次のように定義する [92]．全ての状態 s_1, s_2

に対して，以下の条件を満たす関数 $Q_p : S \times S \to [0,1]$ を q-模倣性と呼ぶ．

$$Q_p(s,t) = \begin{cases} NS(s,t) & s \text{ の出遷移が存在しない場合,} \\ (1-p)NS(s,t) + \frac{p}{n}(\Sigma_{s \xrightarrow{a} s' \in \Delta} \max_{t \xrightarrow{b} t' \in \Delta}(LS(a,b)Q_p(s',t'))) & s \text{ の出遷移が存在する場合.} \end{cases}$$

ここで，$NS : S \times S \to [0,1]$ は状態間の類似度，$LS : A \times A \to [0,1]$ は事象間の類似度であり，n は状態 s からの出遷移数である．本論文では，全ての状態 s, t に対して $NS(s,t) = 1$ とする．また，全ての事象 a, b に対して，$a = b$ ならば $LS(a,b) = 1$ とし，$a \neq b$ ならば $LS(a,b) = 0$ とする．このとき，いかなる $0 < p < 1$ に対しても，$Q_p(s,t) = 1$ となるための必要十分条件は，t が s を模倣することである [92].

模倣関係と同様に，この指標もまた，2 つのモデルの状態間の指標に自然に拡張できる．2 つの LTS $L_i =< S^i, A^i, \Delta^i, s_0^i >$ $(i = 1, 2)$ の状態間の q-模倣性を $Q_p : S^1 \times S^2 \to [0,1]$ とする．ここで定める Q_p の定義式は，上の式に現れる $s \xrightarrow{a} s' \in \Delta$ と $t \xrightarrow{b} t' \in \Delta$ をそれぞれ $s \xrightarrow{a} s' \in \Delta^1$ と $t \xrightarrow{b} t' \in \Delta^2$ とすることで得られる．初期状態間の q-模倣性 $Q_p(s_0^1, s_0^2)$ は，L_1 と L_2 の間の q-模倣性である．

以下の例が示すように，q-模倣性は，2 つのモデルの振舞いについての類似性を表すことができる．

例 23. 図 3.14 に関して，$p = 0.5$ として q-模倣性を計算すると，$Q_{0.5}(s_0, t_0) = Q_{0.5}(s_0, t_1) = 1$ である（同様に，$Q_{0.5}(t_0, s_0) = Q_{0.5}(t_1, s_0) = 1$ である）．この例より，$Q_{0.5}(s_0, t_0) = 1$ なので，L_t の振る舞いが L_s の振る舞いを模倣するという事実を q-模倣性が反映していることが分かる．同様に，$Q_{0.5}(t_0, s_0) = 1$ なので，L_s の振る舞いもまた L_t の振る舞いを模倣することが q-模倣性から理解される．

■

Sokolsky 等は q-模倣性の計算を線形計画問題に帰着させられることを示した [92]．よって，GNU Linear Programming Kit(GLPK) [2] などの線形計画問題用の既存の求解器を用いて自動的に求めることができる．

3.5.2　類似度を用いたモデル修正手続き

ここでは，q-模倣性をわれわれが提案するモデル修正手続きで活用する手法を述べる．この基本的な考え方は，4VTS で表現された基盤モデル FV_Φ と反例から構成したモデル FV_π との合併モデル FV_{mod} に対して行われる開発者による遷移の選択を支援するために，修正対象モデル L と合併モデルとの間の q-模倣性に基づく類似度を計算することにある．そのために，合併モデル FV_{mod} から禁止遷移を除いたモデルを LTS とみなすことにより（L_{FV} と書く），類似度を算出する．加えて，合併モデルと修正対象モデルの各状態間の対応付けを，両者の並列合成を計算することで求める．これらの手続きによって，合併モデルの各状態が修正対象モ

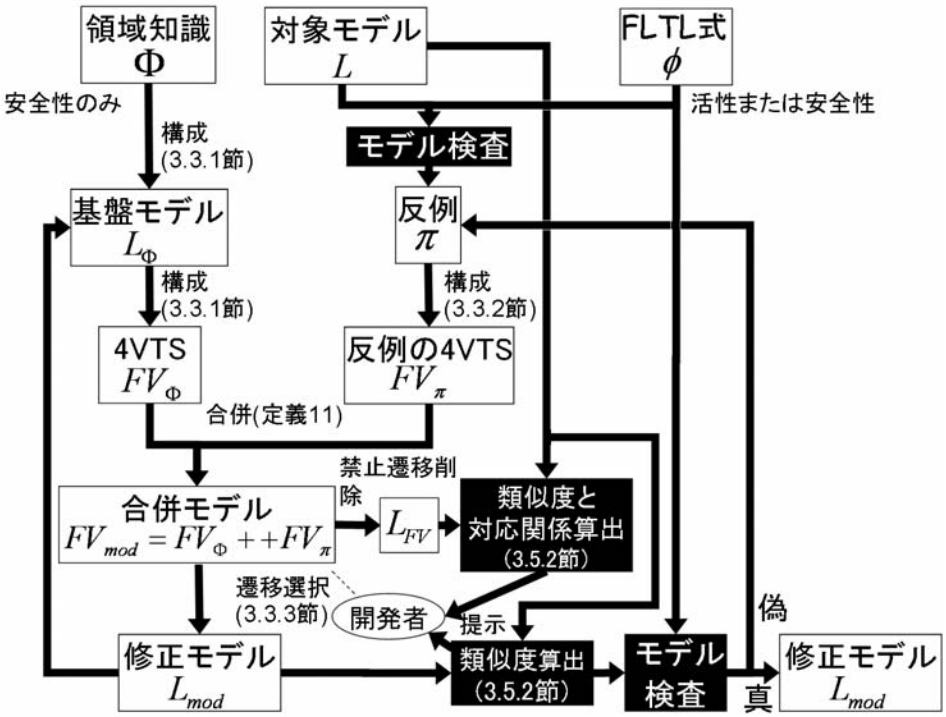


図 3.15. 改善したモデル修正プロセス

デルのどの状態と対応付けられるか，という情報を開発者に提供できる．最後に，修正作業の結果得られる LTS と修正対象モデルとの類似度を算出して開発者に提示する．これは，複数の修正候補が考えられるときに，開発者が最も適当なモデルを選択するための 1 つの判断基準を与える．

類似度を用いたモデル修正プロセスの実行の流れを図 3.15 に示す（図中の記号は図 3.3 に基づく）．図 3.3 の修正プロセスとの違いは次の通りである．図 3.15 には，合併モデル FV_{mod} から構成したモデル L_{FV} と修正対象モデル L との類似度を算出して開発者に提示する手続きと，修正モデル L_{mod} と L との類似度を同様に提示する手続きを含む．

類似度の活用

われわれは，q-模倣性を用いた 2 つの LTS の各状態間の類似度を新たに導入する．導入する類似度は関数 $Sim_p^q(s,t) : S \times S \to [0,1]$ であり，以下のように定義される．

$$Sim_p^q(s,t) = (1-q)Q_p(s,t) + qQ_p(t,s)$$

ここで，q は $0 < q < 1$ なる媒介変数である．本論文では $p = q = 0.5$ とし，$Sim(s,t) = Sim_{0.5}^{0.5}(s,t)$ とする．この類似度は模倣等価性を特徴付ける．

定理 1. 全ての状態 s,t に対して $NS(s,t) = 1$ とし，全ての事象 a,b に対して，$a = b$ ならば $LS(a,b) = 1$ とし，$a \neq b$ ならば $LS(a,b) = 0$ とする．このとき，いかなる $0 < p < 1, 0 < q < 1$ に対しても，$Sim_p^q(s,t) = 1$ となるための必要十分条件は，s と t が模倣等価であるこ

とである．

■

証明. q-模倣性の定義より，$0 \leq Q_p(s,t) \leq 1$ かつ $0 \leq Q_p(t,s) \leq 1$ なので，$Sim_p^q(s,t) = 1$ となるための必要十分条件は $Q_p(s,t) = 1$ かつ $Q_p(t,s) = 1$ である．また，文献 [92] 定理 2 より，$Q_p(s,t) = 1$ となるための必要十分条件は，t が s を模倣することであり，同様に，$Q_p(t,s) = 1$ となるための必要十分条件は，s が t を模倣することである．したがって，模倣等価性の定義より，$Q_p(s,t) = 1$ かつ $Q_p(t,s) = 1$ となるための必要十分条件は，s と t が模倣等価であることである．以上より，定理が成り立つ．

■

上の定理から $Sim(s,t)$ が最大値 1 のとき，また，そのときに限り，2 つの状態は模倣等価である．すなわち，$Sim(s,t)$ は，2 つの状態が模倣等価なときに両者が等価と判定し，そうでないときには類似性が低いと判定する．

2 つの LTS についての q-模倣性を用いることで，2 つの LTS $L_i =< S^i, A^i, \Delta^i, s_0^i >$ $(i = 1,2)$ に対してこの類似度を拡張できる：$Sim_p^q(s,t) : S^1 \times S^2 \rightarrow [0,1]$．また，初期状態 s_0^1 と s_0^2 との間の類似度 $Sim_p^q(s_0^1, s_0^2)$ によって L_1 と L_2 との間の類似度を定める．上の場合と同様に，状態 $s \in S^1, t \in S^2$ に対して，$Sim(s,t) = Sim_{0.5}^{0.5}(s,t)$ と書くことにする．

例 24. 図 3.14 において $Sim(s_0,t_0) = Sim(s_0,t_1) = 1$ である．よって，初期状態間の類似度が 1 なので，L_s と L_t の類似度は 1 となる．この結果から，L_s と L_t が模倣等価性の観点から等価なモデルであると判断される．

■

3.3.3 節で示した修正プロセスでは，修正対象モデルを手掛かりとすることなく修正を行うので，合併モデルと修正対象モデルとの関連が明らかでない．したがって，修正対象モデルの振る舞いと一致させるだけでよく，修正の必要のない遷移や状態を合併モデル上で同定する作業が開発者に委ねられてしまう．ゆえに，効率的に修正手続きを実行することが困難である．そこで，両者の各状態間の $Sim(s,t)$ を開発者に提示することによって，対応する状態間における振る舞いの類似度を示すことができる．それゆえ，開発者は修正の手掛かりとして修正対象モデルとの状態間の関連に関する情報が得られるので，その情報を用いて遷移の選択を効率的に実行することができる．

また，提案する反復型修正プロセスの実行によって得られる LTS で記述された修正モデルと，修正対象モデルとの類似度 $Sim(s,t)$ も計算する．これは，修正モデルは修正対象モデルに基づいて構成されるので，両者が類似しているという想定に基づく．このように修正モデルについて類似度を求めることは，対象モデルに複数の修正候補が考えられるときに，開発者が最も適当なモデルを選択する手掛かりを与える．

並列合成の活用

類似度の計算に加えて，合併モデル FV_{mod} と修正対象モデル L の各状態間の対応関係を両者の並列合成によって求める．これは，両者が共有する事象を同時に実行し，互いの状態の対応付けを計算することに相当する．

ここで，状態間の対応付けを次のように定める．修正対象モデル L の状態集合を S，L_{FV} の状態集合を S_{FV} とする．L と L_{FV} の並列合成 $L \parallel L_{FV}$ の状態集合 $S \times S_{FV}$ のうち，初期状態から到達可能な状態の集合を $S' \times S'_{FV} \subseteq S \times S_{FV}$ とするとき，状態 $(s, s_{FV}) \in S' \times S'_{FV}$ を s と s_{FV} の対応付けと呼ぶこととする．一般に，この定義のもとでは，L の 1 つの状態は L_{FV} の複数の状態と対応し，逆に，L_{FV} の 1 つの状態は L の複数の状態と対応する．

モデル修正手続き

以上で論じた手続きは，以下の手順によって実行される．

1. 3.3 節のステップ 1 から 4 を実行し，合併モデル FV_{mod} を求める．
2. FV_{mod} から禁止遷移を除き，その結果初期状態から到達不可能となる状態も除く．そして，他の必須，可能，不確実遷移を区別せずに LTS の意味での遷移としたモデルを LTS L_{FV} とし，以下の手順を実行する．
   (a) L_{FV} と修正対象モデル L の並列合成を求め，状態間の対応付けを求める．
   (b) L_{FV} の各状態 t と L の各状態 s との $Sim(s,t)$ を計算する．
3. 求めた類似度と状態の対応付けの情報を用いて，3.3 節のステップ 5 を実行する．
4. 修正モデル L_{mod} と L との類似度を算出し，その結果をもとにして 3.3 節のステップ 6 を実行する．

以下に，類似度を用いたモデル修正の指針を示す．

1. モデルの誤りと関連しない遷移の選択を行う際には，修正対象モデルとの類似度が増加するようにモデルの修正作業を実施する．これは，修正対象モデルに対して修正する必要がない振る舞いは，合併モデルにおいても修正対象モデルと同一の振る舞いを示すことが望ましいという想定に基づく指針である．したがって，この指針に従って得られる修正モデルの振る舞いは，修正対象モデルの振る舞いに類似することとなる．ただし，修正対象モデルの誤った振る舞いを修正するためには，修正モデルが表す振る舞いを修正対象モデル上の対応する振る舞いから変更する必要がある．この場合には，修正対象モデルとの類似度が低下するように合併モデルの遷移を選択しなければならない．加えて，基盤モデルの構成に用いる領域知識は，システムが満たすべき全ての性質や要求を表現しているとは限らず，開発者は，領域知識で性質として明確に定式化されていない要求や仕様に基づいて合併モデルから遷移を選択しなければならないことがある．その場合にも，修正対象モデルとの類似度が低下するように遷移の決定を行うことがある．修正対象モデル上で，誤りがあって修正すべき遷移 $\delta = s \xrightarrow{a} t$ は，反例が通過する遷移

である．δ に相当する合併モデル上の遷移 $\delta' = s' \xrightarrow{a} t'$ は，以下の情報を手掛かりとして同定できる．状態 s と対応付けられる合併モデルの状態が s' であり，s' からの出遷移において，事象 a を持つ遷移が可能遷移，不確実遷移あるいは禁止遷移である．

2. 修正対象モデルの 1 つの状態と対応し，かつ，類似性が認められる合併モデルの状態が複数あるならば，それらの各状態から起こる振る舞いは修正モデルにおいて互いに類似すると考えられる．そこで，対応する合併モデルの各状態からの振る舞いが互いに模倣等価になるように修正モデルの遷移を選択する．合併モデルのそのような状態の 1 つからの出遷移に対する意思決定作業を実施した場合を考える．その結果，必須（あるいは禁止）遷移に変更した遷移と同じ事象をラベルに持つ遷移が，未だ意思決定を行っていない残りの状態の出遷移に可能遷移として存在するならば，そのような遷移を同様に必須（あるいは禁止）遷移に変更する．

   対象となるシステムが大規模な場合，領域知識に含まれる安全性の数もまた多くなること，もしくは，反例の規模も大きくなることがありうる．その結果，領域知識から構成する基盤モデルや反例から構成するモデルが大規模になる可能性がある．この場合は，この指針を活用することにより，開発者による合併モデルに対する遷移選択作業の負荷を軽減することができる．この指針によって，合併モデルの複数の状態から始まる振舞いが，互いに類似するように遷移の選択がなされる．したがって，同時に複数の状態を修正することが可能になる．ただし，この指針を支援するソフトウェアツールによって，開発者はより容易に遷移選択ができるようになるであろう．修正作業を支援するソフトウェアツールについては，3.6 節で議論する．

例 25. 3 段階スイッチモデル L_2（図 2.2）と活性 $\phi'_3 = \mathbf{F}\mathit{off}$ を考える．既に述べたように，基盤モデルと反例モデルの合併モデルは図 3.8 のように得られる．この合併モデルから禁止遷移を除き，禁止遷移を除いたことによって到達不可能となる状態 $(sink, sink)$ を除いたモデルを LTS とみなす．この LTS の各状態と L_2 の各状態と類似度を Sim によって計算すると，表 3.1 の通りになる．既に述べたように，モデル間の類似度は，初期状態間の類似度によって定義される．この例では，その値は $Sim(s_0, (s_0, u_0)) = 0.82$ である．

表 3.1. L_2 と合併モデルの状態間の類似度

		L_2		
		s_0	s_1	s_2
合併モデル	(s_0, u_0)	0.82	0.88	0.8
	(s_2, u_1)	0.5	0.94	0.83
	(s_0, u_2)	0.82	0.88	0.8
	$(s_0, sink)$	0.82	0.88	0.8
	$(s_1, sink)$	0.97	0.5	0.5
	$(s_2, sink)$	0.5	0.94	0.83

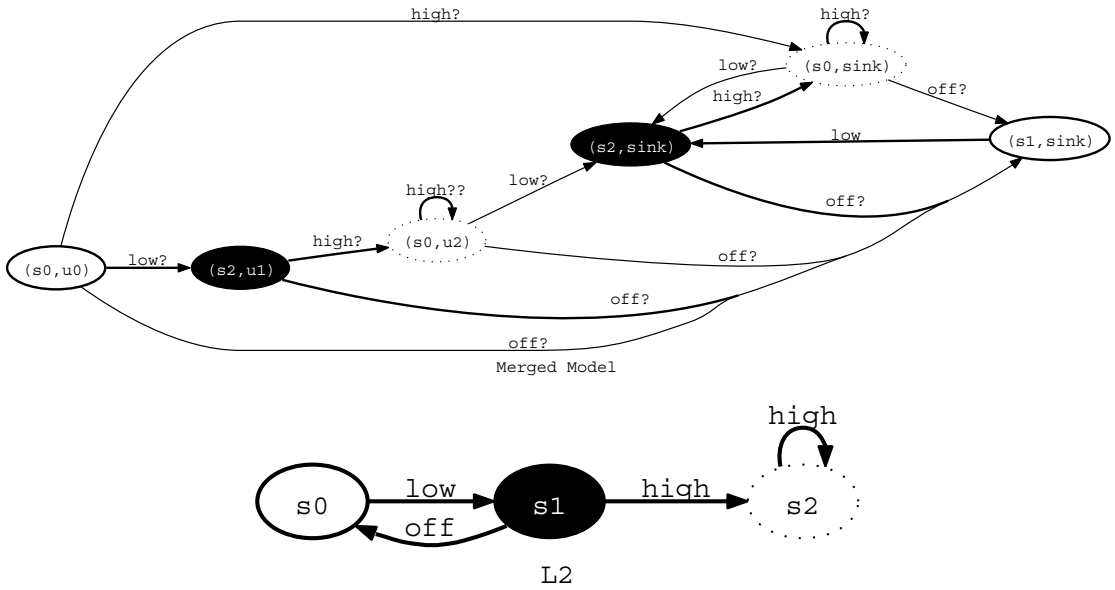


図 3.16. L_2 と合併モデルの状態間の対応付け

L_2 の初期状態 s_0 と合併モデルの初期状態 (s_0, u_0) の類似度が 1 より低い値となっている原因の 1 つには，合併モデルの初期状態 (s_0, u_0) に，L_2 の初期状態 s_0 からの事象 *low* を持つ出遷移に相当する可能遷移に加えて，事象 *off*，*high* を持つ出遷移が存在することがある．図 3.16 に合併モデルと L_2 の各状態の対応付けを図示する．両者において，同じ装飾がなされた状態が互いに対応付けられる．ただし，合併モデルにおいて，簡単のため禁止遷移を除いてある．図 3.16 に示す対応付けと既に求めた類似度によって，開発者は合併モデルに対して以下のような判断が可能である．

- **L_2 の状態 s_1 の出遷移に対応する合併モデルの遷移選択**：合併モデルの状態 $(s_2, u_1), (s_2, sink)$ は共に L_2 の状態 s_1 に類似していると判断できる．したがって，修正モデルを得るために，これらの状態について開発者は以下のように，修正の指針 1 と 2 に従って遷移の選択を行うことが可能である．
  - **状態 (s_2, u_1) の出遷移の選択**：状態 (s_2, u_1) からの出遷移 $(s_2, u_1) \xrightarrow{high}_{\mathbf{c}} (s_0, u_2)$ と $(s_2, u_1) \xrightarrow{off}_{\mathbf{c}} (s_1, sink)$ を，それぞれ $(s_2, u_1) \xrightarrow{high}_{\mathbf{t}} (s_0, u_2)$ と $(s_2, u_1) \xrightarrow{off}_{\mathbf{t}} (s_1, sink)$ に変更することで，L_2 の s_1 の出遷移を保存することができる．
  - **状態 $(s_2, sink)$ の出遷移の選択**：状態 $(s_2, sink)$ からの出遷移 $(s_2, sink) \xrightarrow{high}_{\mathbf{c}} (s_0, sink)$ と $(s_2, sink) \xrightarrow{off}_{\mathbf{c}} (s_1, sink)$ を，それぞれ $(s_2, sink) \xrightarrow{high}_{\mathbf{t}} (s_0, sink)$ と $(s_2, sink) \xrightarrow{off}_{\mathbf{t}} (s_1, sink)$ に変更することで，L_2 の s_1 の出遷移を保存することができる．

  以上の修正を行った結果得られるモデルと修正対象モデル L_2 との類似度は，修正作業前と変わらず 0.82 である．
- **L_2 の状態 s_2 の出遷移に対応する合併モデルの遷移選択**：合併モデルの状態

$(s_0, u_2), (s_0, sink)$ は，L_2 の状態 s_2 と対応付けられる．ただし，これらの状態からは，L_2 の状態 s_2 と異なり，事象 low，off を遷移ラベルとする出遷移を持つ．このことが，修正対象モデルの状態との類似度が 1 より低い値となる原因の 1 つとなっている．

- **状態 (s_0, u_2) の出遷移の選択**：状態 (s_0, u_2) は，不確実遷移 $(s_0, u_2) \xrightarrow{high}_{\mathbf{u}} (s_0, u_2)$ を出遷移として持つ．よって，L_2 の s_2 を修正を施す必要があると解釈できる．そこで，$(s_0, u_2) \xrightarrow{high}_{\mathbf{u}} (s_0, u_2)$ を禁止遷移としてモデルから除く．次に，性質 ϕ_3' を満たすためには，遷移 $(s_0, u_2) \xrightarrow{off}_{\mathbf{c}} (s_1, sink)$ を $(s_0, u_2) \xrightarrow{off}_{\mathbf{t}} (s_1, sink)$ に変更すればよい．また，修正対象モデルの状態 s_2 はスイッチが high にあることを表現した状態である．ここで，off 以外のもう 1 つのスイッチの状態 low へ遷移することが必要ならば，修正対象モデルの状態 s_1 に相当する合併モデルの状態に遷移できなければならない．これをモデル上で実現するために，修正の指針 1 に従って，可能遷移 $(s_0, u_2) \xrightarrow{low}_{\mathbf{c}} (s_2, sink)$ を必須遷移 $(s_0, u_2) \xrightarrow{low}_{\mathbf{t}} (s_2, sink)$ にする．以上の修正によって，修正対象モデル L_2 との類似度は，修正 1 の指針で述べたように，0.82 から 0.790 へと低下する．
- **状態 $(s_0, sink)$ の出遷移の選択**：上と同様の考察を行うことにより，$(s_0, sink)$ からの出遷移 $(s_0, sink) \xrightarrow{off}_{\mathbf{c}} (s_1, sink)$ を $(s_0, sink) \xrightarrow{off}_{\mathbf{t}} (s_1, sink)$ に変更する．また，可能遷移 $(s_0, sink) \xrightarrow{low}_{\mathbf{c}} (s_2, sink)$ もまた必須遷移 $(s_0, sink) \xrightarrow{low}_{\mathbf{t}} (s_2, sink)$ に変更する．一方，$(s_0, sink)$ と (s_0, u_2) は共に s_2 と対応付けられるので，両者から始まる振る舞いは同一であることが自然である（修正の指針 2）．そこで，$(s_0, sink) \xrightarrow{high}_{\mathbf{c}} (s_0, sink)$ を禁止遷移としてモデルから除く．この修正作業の結果，修正対象モデル L_2 との類似度は 0.786 となる．

ここでは，反例が示すモデルの誤りを修正したため，修正対象モデルとの類似度が 0.82 から 0.786 に低下している．

- **L_2 の状態 s_0 の出遷移に対応する合併モデルの遷移選択**：最後に，初期状態 (s_0, u_0) は，$(s_1, sink)$ と同様に s_0 に対応付けられる．$(s_1, sink)$ からは事象 low をラベルに持つ出遷移があるが，(s_0, u_0) からは他の事象をラベルに持つ出遷移も存在するため，類似度が 1 より低い値になっている．そこで，s_0 の出遷移を保存して L_2 との類似度を高めるように，$(s_0, u_0) \xrightarrow{low}_{\mathbf{c}} (s_2, u_1)$ を必須遷移 $(s_0, u_0) \xrightarrow{low}_{\mathbf{t}} (s_2, u_1)$ とし，他の出遷移を禁止遷移として除く．以上の結果得られるモデルと L_2 との類似度は 0.93 である．

この結果得られる最終的な修正モデルは，図 3.10 に示したモデルである．前節の事例研究のように状態間の対応付けと類似度の情報がない場合は，合併モデルと修正対象モデルとの関連付けが明らかでないため，開発者は修正作業を行う際に修正対象モデルを活用することができない．以上に示した情報を提供することで，開発者によるモデルの遷移の選択作業を支援をすることができる．

最後に，図 2.2 の L_2 と図 3.10 のモデルとの間の類似度を計算する．その値は既に求めたとおり，0.93 である．

■

3.5.3 改善手法の実装と事例研究

ここでは，モデル修正法の改善手法について作成した実装とそれを使った事例研究について報告する．

実装

われわれは，Java 言語によって，3.3 節で述べた提案手法およびその改善手法の以下の手続きを実装して自動化した．

- 3.3.1 節に従って，基盤モデルから 4VTS を構成する手続き．ただし，入力となる基盤モデルは MTS として表現されていることとする．この入力モデルは，MTSA ツール [38] を用いることで構成できる．ただし，3.4 節で行った最適化と同様の考え方に従い，構成するモデルには禁止遷移を含めない．
- 反例から 3.3.2 節に従って 4VTS を構成する手続き．このモデルにも，基盤モデルと同様に禁止遷移を含めない．
- 上で構成した基盤モデルと反例の 4VTS に対する定義 11 の合併演算を実行して，合併モデルを構成する．
- モデル間の類似度を線形計画法の求解器 GLPK を用いて算出する．
- 並列合成演算により，2 つのモデルの状態間の対応付けを計算する．

この実装は，MTS で表現された基盤モデル，反例，修正対象モデルを入力とし，合併モデルと，合併モデルと修正対象モデルの状態間の対応付けと類似度を計算し，出力する機能を有する．また，この機能とは独立に，修正モデルと修正対象モデルとの類似度を計算するために，LTS または 4VTS で記述されたグラフ間の類似度と対応付けを算出する機能を備えている．計算した類似度は，対応付けのある状態間の類似度と，そうでない状態間の類似度に分類して出力する．

事例研究

作成したプログラムを用いて以下の事例について改善手法の有効性を調べた．

- **3.4 節の電子レンジシステム**：検証を行う性質，基盤モデルに用いる領域知識，修正プロセスの 1 回目の修正に用いる反例は，3.4 節と同一とする．
- **ウェブを用いた電子メールシステム [97]**：システムが満たすべき活性を定め，検証を行った．その結果得られる反例を用いて，モデルの修正を行った．
- **鉱山用排水ポンプ制御システム [97]**：文献 [97] で作成された LTS モデルを元にして本事例研究用のモデルを作成した．続いて，同文献において列挙された 13 の安全性の中で，作成したモデルが満たさない性質を 1 つを定めて修正を行った．

各事例研究の詳細を以下で報告する．

電子レンジシステム　基盤モデルと反例モデルとの合併モデルと，修正対象モデルとの類似度は 0.85 だった．

まず，性質 4 について電子レンジシステムを修正する際に，1 回目の修正において合併モデルから 2 つの戦略を用いて修正モデルを求めた．各戦略に対して，3.5.2 節で挙げた 2 つの修正の指針を適用して修正を実施した．第 1 の戦略は，性質違反を起こす遷移をモデルから削除することで修正モデルを得るというものである．その結果，1 回の反復で性質 4 を満たすモデルが得られた．このモデルは，3.4 節の修正作業によって得られた修正モデルとは異なる振る舞いを示すモデルだった．一方，第 2 の戦略は，性質違反となる事象が発生しないように，修正対象モデルに遷移を追加することで修正モデルを求めるというものである．その結果，追加すべき遷移を的確に決定するために 2 回の反復が必要となったが，3.4 節の修正結果と同一の振る舞いを示す修正モデルが得られた．ただし，性質 4 について以上のようにモデル修正を行うときには，図 3.12 のモデルを修正対象モデルとして，合併モデルならびに修正モデルとの類似度を算出した．合併モデルの各状態が，修正対象モデルのどの状態に相当するかの判断は，両者の対応付けと算出した類似度を用いて行った．

この修正では，モデル修正の指針により，修正結果と修正対象モデルとの類似度が，合併モデルと修正対象モデルとの類似度 0.85 より増加するように修正を行った．2 つの戦略に基づく修正作業で得られた性質 4 を満たすそれぞれのモデルから修正モデルを決定するために，各修正モデルと修正対象モデルとの類似度を用いた．第 1 の戦略に基づく修正モデルと修正対象モデルとの類似度は 0.9981 であり，一方，第 2 の戦略に基づいて 1 回修正を行った結果得られた LTS の類似度は 0.999 だった．後者のモデルは，修正対象モデル上で反例の実行のみを禁止し，他の軌跡は実行できるように遷移を選択することで得られたため，修正によって得られたモデルの振る舞いと修正対象モデルの振る舞いとが，前者に比べて互いに類似していたと考えられる．そこで，後者を修正モデルとして後者を選択した．第 2 の戦略に従った場合，2 回目の修正によって求めた性質 4 を満たす修正モデルと，修正対象モデルとの類似度は 0.9984 であった．

次に，性質 4 の修正結果をもとにして性質 5 の修正を改善手法を用いて行い，図 3.13 のモデルが得られるか調べた．その結果，修正プロセスを 1 回実行することで，図 3.13 のモデルと同一の振る舞いを示す修正モデルが得られた．ただし，性質 5 についてモデル修正を行うときには，性質 4 についての修正結果のうち，上で得られた 3.4 節の修正結果と同一の振る舞いを示す修正モデルを改めて修正対象モデルとして，合併モデルおよび修正モデルとの類似度を算出した．得られた修正モデルと修正対象モデルとの類似度は，0.983 であった．

事例研究において，類似度を以下のように活用してモデル修正作業を実施した．

- 3.4 節では，修正モデルの選択に際して経験的な方法のみを用いた．本節で行った性質 4 についての修正プロセスにおいて，上で述べたように修正対象モデルとの類似度を用いることで，修正モデルを選択するための指針に従って修正作業を実施した．また，ここで用いる類似度はモデルの振る舞いに基づいているので，モデルの構造や規模に影響されずに状態間の類似度を測定することができた．特に，合併モデルと修正対象モデル

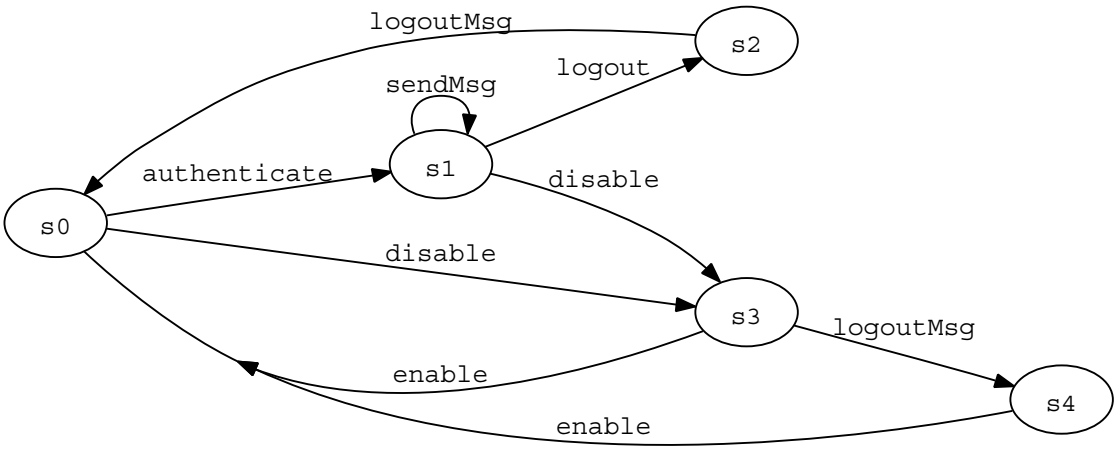
との対応関係とそれらの類似度を定量的に把握することができた．

- 性質 4 と 5 のどちらの修正においても，合併モデルの 1 つの状態に対して，修正対象モデルの複数の状態が対応し，また，その類似度が約 0.8 という値を示す場合があった．この場合，合併モデルの該当する状態が修正対象モデルの複数の状態の情報を含んでいると考えられる．しかし，モデル修正を行う際には，合併モデルの 1 つの状態に修正対象モデルのただ 1 つの状態が対応している方が，合併モデルから修正モデルに必要な遷移を選択する作業を効率的に実施することができる．そこで，合併モデルにおいて該当する状態とその状態に対する入遷移と出遷移を，修正対象モデルの対応する状態の数だけ複製し，各状態に対応すると解釈して遷移の選択を行うという発見的な手段を講じた．
- 上とは逆に，修正対象モデルの 1 つの状態と対応する合併モデルの状態が複数ある場合には，既に述べた指針に従って，遷移ラベルが互いに同じになるように，該当する合併モデルの各状態の出遷移を選択した．特に，実行が禁止されるべき遷移について，以下の手順で合併モデルの各状態からの出遷移を選択した．
  1. 合併モデルの 1 つの状態において，反例モデルが示す性質違反となる事象をラベルに持った遷移を禁止遷移とした．
  2. 同じ修正対象モデルの状態と対応する他の状態についても，同じ事象をラベルに持つ遷移を禁止遷移とした．

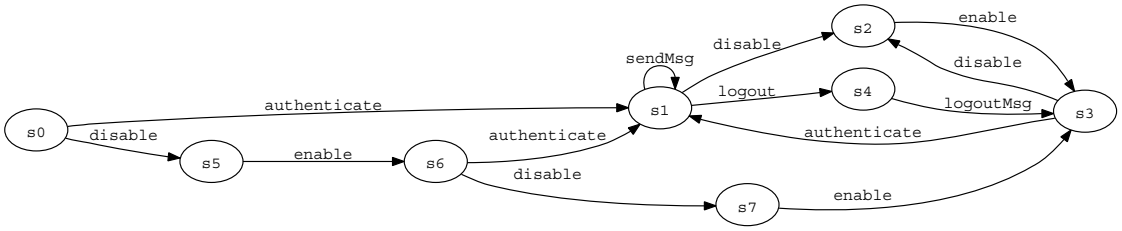
  その結果，性質 5 に対する修正作業について，3.4 節と異なり，1 回の修正作業で性質違反の原因を取り除くことができた．合併モデルの関連する遷移に対する意思決定を互いに独立に行う必要がなくなるので，修正対象モデルとの類似性を考えない場合と比べて修正作業の効率化が実現されたと思われる．
- 性質 5 の修正に関しては，修正対象モデルのいずれの状態とも対応しない状態が合併モデルに存在した．この状態と修正対象モデルの各状態との類似度は $0.6 - 0.8$ の値となった．この状態は修正対象モデルから逸脱した振る舞いを表現する状態であり，修正対象モデルの多くの状態と類似性があると解釈できる．したがって，状態の追加が必要な修正作業を行う際にこのような状態が有用であると考えられる．ただし，電子レンジシステムの事例に関しては，その状態に到達する遷移を修正モデルから全て除く必要があった．それゆえ，到達不可能な状態として削除した．

これらは既に述べたモデル修正作業の指針の妥当性に加えて，新たに指針に加えるべき事項を表していると考えられる．そこで，このような指針を更なる事例研究を重ねて収集，整理することでガイドラインを作成することができるであろう．

ウェブメールシステム　対象となるウェブメールシステムは，文献 [97] で扱われている．利用者はシステムにログインし，メッセージの送信を行った後にログアウトする．ログアウト時には，その旨を表示して利用者に伝える．ただし，システムが適切に利用されていない時には，システムの管理者は利用者の利用権限を停止することができる．その場合，管理者が利用権限を再度与えない限り，利用者はシステムにログインをしてメールを送信することができない．



性質 MsgAfterLogout を満たさない修正対象モデル



修正によって得られるモデル

図 3.17. ウェブメールシステム

われわれは，まず，検証対象となるウェブメールシステムのモデルを文献 [97] を元にして以下の手順で作成した．

1. 文献 [97] に従って，ウェブメールシステムの MTS モデルを MTSA ツールを用いて構成した．このモデルは，文献 [97] で取り上げられた 8 個の安全性を満たし，かつ，システムが実行すべきシナリオを表現する．
2. この MTS の可能遷移と必須遷移を LTS の遷移に変更して，LTS で記述されたモデルを構成した．構成したモデルを図 3.17 の上図に示す（初期状態を s_0 とする）．

上の手順で構成された LTS に対して，新たに満たすべき活性 MsgAfterLogout を定めた．この活性は，利用者がログアウトした後に，はじめてシステムはログアウトしたことを表すメッセージを利用者に表示することを主張する性質である．MsgAfterLogout は以下の FLTL 式で表される．

$$\texttt{MsgAfterLogout} = (\neg\texttt{LoggedIn} \wedge \neg \mathit{logoutMsg})\mathbf{U}(\texttt{LoggedIn} \wedge \mathbf{F}(\neg\texttt{LoggedIn} \wedge \mathit{logoutMsg}))$$

ここで，LoggedIn は，利用者がウェブメールシステムにログインしていることを表す流動で，$\texttt{LoggedIn} =< \{\mathit{authenticate}\}, \{\mathit{logout}, \mathit{disable}\}, \mathbf{f} >$ と定義される [97]．また，$\mathit{logoutMsg}$ はシステムが利用者によるログアウト処理の成功を表示するメッセージを出力する事象である．

次に，われわれは，作成した LTS について性質 `MsgAfterLogout` のモデル検査を実施した．ただし，この事例研究では，公平な選択を仮定して活性の検証を行った．検証の結果，LTS が `MsgAfterLogout` を満たさず以下の反例が得られた．

$$[disable, logoutMsg(enable, disable, logoutMsg)^{\omega}]$$

そこで，得られた反例を用いて，`MsgAfterLogout` を満たすように図 3.17 の上図のモデルを修正するために，提案するモデル修正法の改善手法を適用した．領域知識には，文献 [97] に示されている 8 個の安全性から構成したモデルを用いた．この安全性には，例えば，利用者がメッセージを送信するのはログイン中に限るという性質を表す FLTL 式がある．モデルの修正に必要なモデル修正プロセスの反復回数は 2 回だった．修正した誤りは，利用者が一度もウェブメールシステムにログインしていないにも関わらず，システムがログアウト成功メッセージを表示することを許す遷移である．この遷移がモデルに存在すると，システムの管理者が利用者の権限を停止した後に権限を回復させた際に，ログアウト成功メッセージを表示してしまう．この誤りを修正することによって，`MsgAfterLogout` を満たすモデルが得られた．以上の手続きによって得られた修正モデルを図 3.17 の下図に示す．

この事例では，合併モデルと修正対象モデルとの類似度は 1.0 だった．それゆえ，修正モデルと修正対象モデルとの類似度が合併モデルと修正対象モデルとの類似度よりも低下するように，合併モデル上で遷移の選択を繰返す必要があった．1 回目の修正において，反例が通過する遷移を 2 箇所削除した．これらは，隣接する遷移，すなわち，同じ状態への入遷移と出遷移だったので，両者を続けて削除することで，修正対象モデルとの類似度は 0.999 となった．2 回目の修正において，新たに得られた反例が示す誤りを実行可能にする遷移をまず削除した．さらに，その遷移が出遷移となる状態と対応付けられる修正対象モデルの状態を求め，求めた状態と高い類似度を示す合併モデルの状態集合を求めた．そして，その集合に含まれる各状態の出遷移のうち，削除した遷移と同一の遷移ラベルを持つものを削除した．その結果，最終的な修正モデルと修正対象モデルとの類似度は 0.96 となった．

この事例研究によって，修正対象モデルにおいて修正の必要のない遷移を合併モデル上で選択する際に，修正対象モデルとの類似度を増加させるような修正が必ずしもなされるとは限らないことが分かった．しかしながら，性質を満たすために修正をする必要のない遷移を誤って修正することで，必要以上に修正対象モデルとの類似度を低下させることのないように修正作業をすることが一つの指針となる．状態間の対応付けや類似度の情報は，そのような不適当な修正を防ぐための手掛かりを開発者に提示することができる．

鉱山用排水ポンプ制御システム　これは，鉱山内に充填する汚水を排水するポンプの動作を制御するシステムである [97]．システムにおいて，汚水の水位は低，中，高の 3 段階で検出される．システムは汚水の水位を監視して，水位が中や高のときにはポンプを起動して排水を促し，水位が低になるとポンプを停止する．ただし，鉱山内にはメタンガスが発生することがあり，ポンプを爆発させる恐れがある．そこで，システムはメタンガスの状況も監視して，メタンガスが存在しないときのみポンプを起動する．汚水の水位が中や高でメタンガスが存在する場合には，システムはポンプを停止した後に危険を知らせる警告灯を点灯させる．汚水の水位

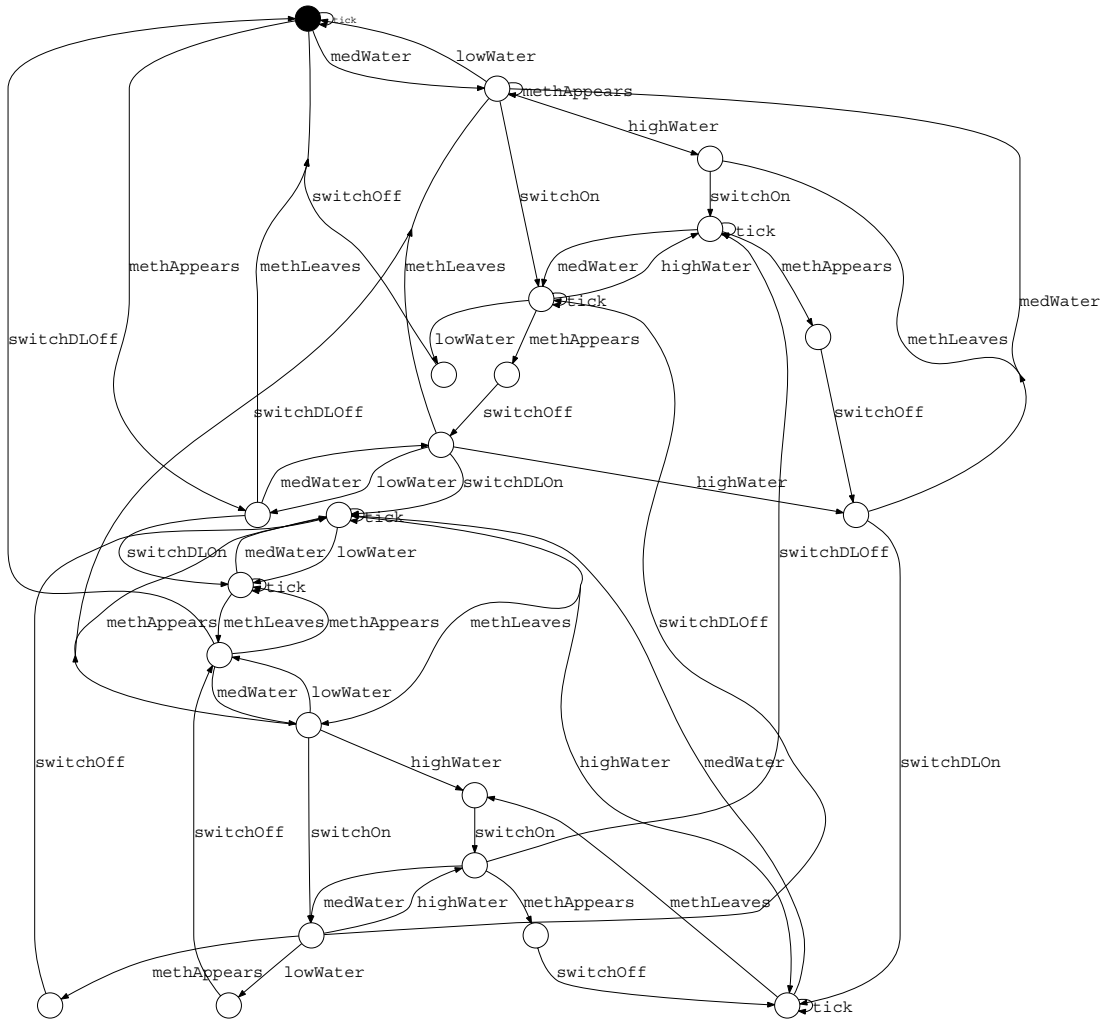


図 3.18. 鉱山用排水ポンプ制御システム

が低の場合であっても，メタンガスが発生するとシステムは警告灯を点灯させて危険を通知する．

このシステムが満たすべき性質として，文献 [97] に 13 の安全性が挙げられている．それらの安全性の 1 つは，「汚水の水位が低であるか，または，メタンガスがある場合には常にポンプを停止する」ことで，以下の FLTL 式で表される [97].

$$\texttt{EmergentOff} = \mathbf{G}((\texttt{LowWater} \vee \texttt{MethanePresent}) \Rightarrow \mathbf{X}\neg\texttt{PumpOn})$$

ここで，`LowWater`，`MethanePresent`，`PumpOn` はそれぞれ，水位が低いこと，メタンガスが発生していること，排水ポンプが起動していることを表す流動である．

われわれは，まず，文献 [97] を基にして図 3.18 に示す制御システムのモデルを作成した．図中で，黒丸で示されている状態が初期状態である．このモデルは，13 の安全性のうち 11 の性質を満たすが，それ以外の `EmergentOff` を含む 2 つの性質を満たさない．特に，`EmergentOff` に違反したのは，図 3.18 のモデルに含まれている 2 箇所の誤りが原因であっ

た．これらの誤りは，汚水の水位が中のときにメタンガスが発生した場合，システムがポンプを起動する，という軌跡を実行することを許してしまう．モデル検査器 LTSA を用いた結果，得られた以下の反例はこの性質違反の証拠を指示していた．

$$[medWater, methAppears, switchOn]$$

ここで，$medWater$, $methAppears$, $switchOn$ はそれぞれ，水位が中になる，メタンガスが発生する，排水ポンプを起動するという事象である．性質 `EmergentOff` を満たすように，われわれは図 3.18 のモデルを修正した．

われわれは，提案手法を実行するために 11 の安全性を領域知識とした．領域知識には，例えば，「排水ポンプの動作中には，システムはポンプの起動操作を実行することが決してない」[97] という性質が含まれていた．ウェブメールシステムと同様に，MTSA を用いて領域知識を表す MTS モデルを作成し，反例，図 3.18 のモデルと共にプロトタイプツールに入力して，合併モデルおよび図 3.18 との類似度を求めた．求めた合併モデルを用いて提案する修正プロセスを実行した．

この事例研究では，2 箇所の誤りを同時に修正したのではなく，1 箇所ずつ修正した．1 箇所目の修正の終了後，2 箇所目の誤りを修正するために修正プロセスを引き続き実行した．その結果，構築した合併モデルには 2 箇所目の誤りを適切に修正できない合併モデルが得られたため，1 箇所目の誤りの修正が不適切であったことが見出された．例えば，この不適切な修正によって，デッドロックを引き起こす状態を持つ修正モデルが求められた．この状態に到達する軌跡は，メタンガスが発生した場合のシステムが行うべき振る舞いを表していたため，モデル上でシステムの正しい振る舞いを記述する必要があると判断された．しかしながら，デッドロック状態からは選択すべき遷移が存在しないため，適切な振る舞いを表す遷移を選択することができない．そのため，1 箇所目の誤りを修正した工程を遡って不適切な修正の要因を調査して，修正モデルの候補を再度作成して修正作業を完了した．

ここで述べたような本事例研究で行われた不適切な修正は，手作業による人為的な選択作業に起因する．具体的には，その原因は以下のようにまとめられる．

- 1 箇所目の誤りを修正する際に，2 箇所目の誤りを正す遷移を削除してしまった．この不適切な修正作業は，図 3.18 の複数の状態と高い類似度の値を持つ合併モデルの 1 つの状態の入出遷移を選択する際に発生した．モデル修正作業の指針に従うと，このような場合は合併モデルの状態と，その状態の入遷移と出遷移を複製しなければならない．この指針に従えば，状態を複製することで，修正対象モデルにおいて対応する各状態の振る舞いを分離することができる．それゆえ，誤りを含む状態の振る舞いと，それ以外の状態の性質を満たす正しい振る舞いを別個に扱うことができる．しかし，修正の結果得られるモデルの規模が大きくなったため，本事例ではその作業を適切に行うことができなかった．そこで，この不適切な修正の是正は，修正工程を遡った調査により状態と遷移を改めて複製して行った．
- 合併モデルの複数の状態が図 3.18 の 1 つの状態と対応付けられる場合，合併モデルの各状態の振る舞いが互いに類似するように修正する必要があった．しかしながら，この

指針に従った修正を行わなかったため，修正モデルの同一の状態と対応する，または類似度の高い状態間で同様な振る舞いを示す修正モデルを，合併モデルから求めることができなかった．詳しく述べると，修正対象モデルの同一の状態と対応する合併モデルの複数の状態について，1 つの状態の出遷移には，修正モデルに必要と判断された事象が含まれていた．しかし，それ以外の他の状態の出遷移にはその事象が存在しなかったため，各状態の振る舞いを一致させることができなかった．また，合併モデルの類似する状態間において，以上のように修正モデルに必要な振る舞いを実行できる状態とそうでない状態が存在したため，これらの類似度の値に違いが見出された．この類似度の違いは，状態間の振る舞いの不一致を指摘していると考えられる．この現象は，状態間の類似度を参照しながら合併モデルから遷移を選択している作業を行っている過程で発見した．そこで，1 箇所目の誤りを修正した際の合併モデルの遷移の選択を再度行い，合併モデルの類似する状態間で振る舞いが同一になるように修正することで，この問題点を解決した．

修正工程の再実行の結果得られた最終的な修正モデルを図 3.19 に示す．ただし，図 3.18 と同様に初期状態を黒丸で示す．このモデルは，`EmergentOff` を満たすのみならず，図 3.18 が満たさなかったもう 1 つの安全性も満たす．すなわち，文献 [97] で挙げられている全ての安全性を満たすモデルである．

3.6　考察と課題

本節では，本章の手法が持つ利点と制約を議論する．加えて，今後の研究課題について述べる．

提案手法の利点と制約

3.4 節，3.5.3 節の事例研究によって，われわれが開発した修正法の有効性だけでなく，その利点もまた見出した．その利点を以下に論じる．

- 提案したモデル修正法は，既存のモデル検査器の結果を直接利用できる．従って，提案手法を用いることで，ソフトウェア開発において，モデル検査器が用いられる多くの工程で，システムの振る舞いを記述するモデルの修正が可能である．特に，LTS のような状態遷移機械がシステムの振る舞いを記述するためによく用いられるソフトウェア開発プロセスにおける要求分析，設計工程において有効と思われる．
- 提案手法は，安全性と活性の両方の性質に対して同一の修正プロセスを提供する．安全性と活性の違いは，与えられた反例から 4VTS を構築する方法の違いにあるのみにある．われわれの方法が採用しているモデル合併法についての先行研究 [97] は，安全性にのみ注目しているが，提案手法が活性の場合にも対象モデルの修正ができることを，事例研究を通して確認した．また，活性の場合は，公平な選択を仮定したモデル検査の結果に対しても，本手法が適用可能である．

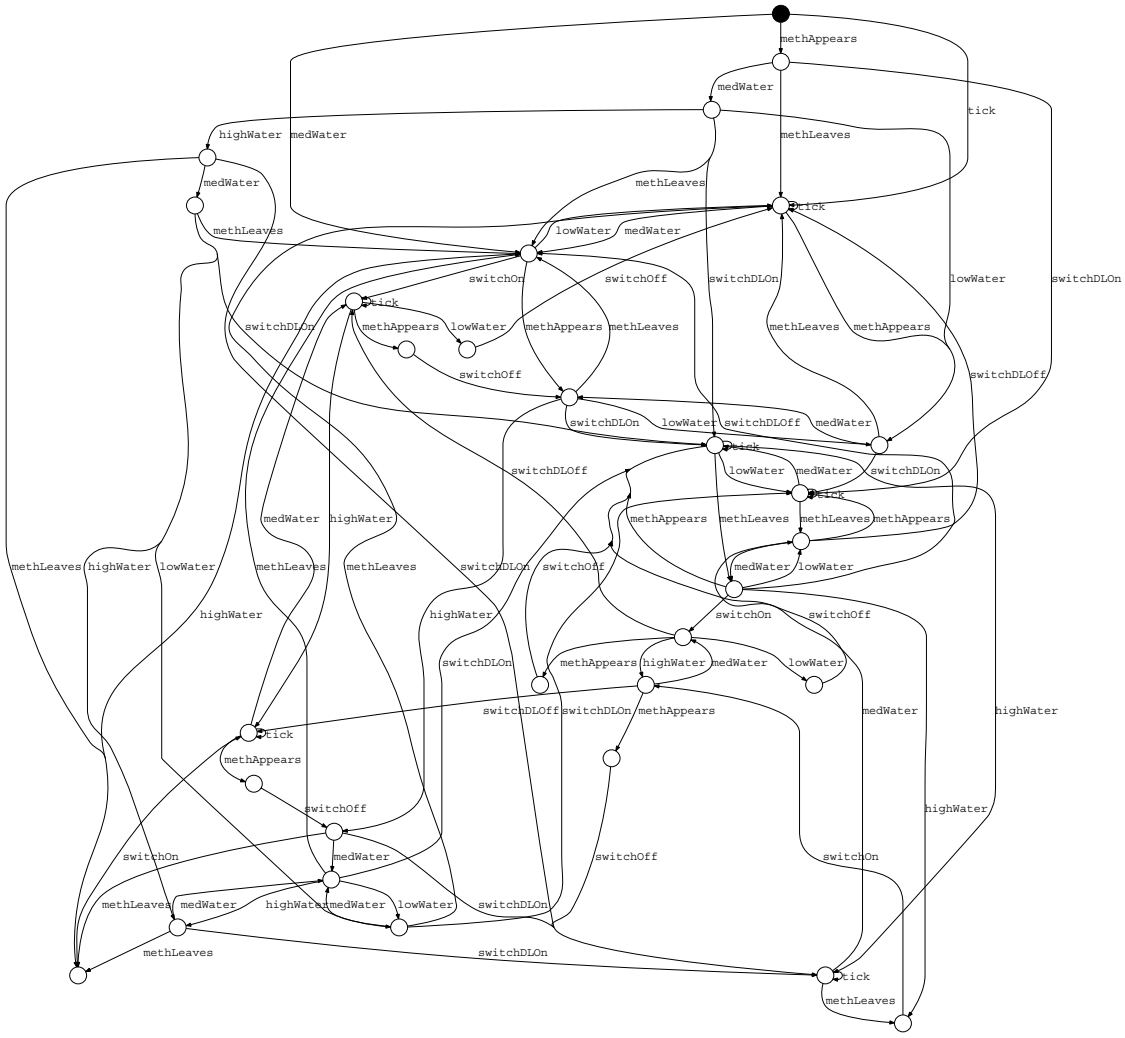


図 3.19. 鉱山用排水ポンプ制御システムの修正モデル

- 提案手法は，4VTS で記述したモデルを活用し，規則に基づいたモデル合併技術に基づく方法である．これらの手段によって，開発者による遷移の選択，つまり修正モデルの抽出作業を支援することができる．
- 提案手法は 4VTS で記述されたモデルの修正候補を開発者に提示するので，開発者との対話的方法である．ゆえに，開発者が修正過程に関わることで，対象領域に関して意味のないモデルが修正後のモデルとして構成される恐れが少ない．

しかしながら，提案手法には適用上の制約や条件が存在することも分かった．以下，各要点を列挙して項目ごとに論じる．

- 提案手法は反例を禁止するモデルを構成することで，基盤モデルが持つ反例の情報を開発者に提供する．しかし，反例のどこに性質違反の原因があるかの判断は開発者自身が行う必要がある．システム規模が大きく，反例が非常に長い事象列からなる軌跡の場合，その作業は困難である．よって，反例や対象モデルにおいて性質違反となる箇所を

自動的に発見する手法があるのが望ましい．この問題点の克服のため，4 章と 5 章において，反例を用いた誤り特定手法を提案する．

- 開発対象領域の知識を表現する基盤モデルは，知識が形式的に記述されていること，すなわち，既にモデル上で満たされる互いに無矛盾な安全性の集合が与えられているという前提に基づいて構築される．このことは，この集合の要素である 2 つの安全性が互いに矛盾した FLTL 式で定式化された場合，対象領域の知識が満たす軌跡が存在しないことを意味する．したがって，この集合から構成される基盤モデルは，いかなる軌跡も表現せず，ただ 1 つの状態のみからなるモデルとなる．このような基盤モデルから得られる 4VTS と，その 4VTS と反例モデルとの合併モデルは，あらゆる軌跡が禁止遷移で表現されたモデルである．よって，性質を満たし，禁止遷移を実行しない軌跡がモデル上に存在しないので，提案手法を適用することができない． さらに，与えられた安全性の集合が修正を行う性質と矛盾する場合には，反復回数が非常に大きくなるか，あるいは，修正作業が行うことができない恐れがある．しかしながら，矛盾する性質や知識を開発者が想定することは，問題領域を開発者が正しく理解していないこと，または，知識の定式化を誤ったことを示していると考えられる．Holzmann が文献 [54] で指摘しているように，開発者が意図した性質を論理式で正確に定式化するには熟練が必要であるので，後者の場合は特に注意が必要である．したがって，これまでの性質の抽出過程，あるいは性質を記述した FLTL 式を再度見直す必要があるだろう．
- 提案手法の効果は，修正モデルである LTS を構築する際に，開発者による適切な意思決定がなされるかどうかに依存する．例えば，図 3.8 の 4VTS において，*off* とラベル付けられた可能遷移を全て禁止遷移に変更したと仮定する．その結果得られる LTS には，*off* という事象を持つ遷移が存在しないので，性質 ϕ_3' を満たすことは決してない．それゆえ，以前の反復において，不適当な遷移の変更がなされたならば，開発者は過去の反復に遡って，以前行った遷移の変更が適切であったか調査しなければならない．しかしながら，3.5 節で導入した合併モデルと修正対象モデルの類似度を計算するステップは，合併モデルと修正対象モデルとの状態間の関連付けを提供する．3.5.3 節で考察したように，状態間の類似度を活用することで，以下の利点が得られたと考えられる．
  - 以前の修正工程で行われた不適切な修正を指摘あるいは防止できる．
  - 修正モデルのある状態と類似した合併モデルの各状態について，一貫した修正を行う手掛かりを提供する．

  ただし，本章で行った事例研究において，合併モデルからの遷移の選択は全て手作業で行った．それによって起こりうる不適当な修正作業の実施を防止するには，遷移の選択を支援する対話型のソフトウェアツールを提供することが 1 つの解決策である．このツールには，次のような機能が必要だろう．
  - 合併モデルを図式的に開発者に提示するだけでなく，開発者がツール上でこの図式を用いて修正を行うことができる．また，この項で述べた図 3.8 の例にあるような不適切な修正が行われたときに開発者にその旨を通知する．
  - 修正対象モデルの複数の状態と合併モデルの 1 つの状態が対応付けられる場合，合

併モデルの関係する状態と遷移を複製したモデルを開発者に提示する．

- 合併モデルの複数の状態と修正対象モデルの 1 つの状態が対応する場合，合併モデル上のその中の 1 つの状態を修正した場合に，その修正を残りの状態に反映させることができる．
- 修正履歴を保存する．この機能は，不適切な修正が行われた場合に工程を遡ってその原因を調査することを可能にする．

不適切な修正作業が行われてしまう原因には，修正対象システムが大規模であり，その結果，基盤モデルや反例から構成したモデルが大規模になってしまうことが挙げられる (3.5.2 節の修正の指針 2)．したがって，上で挙げた機能を持つソフトウェアツールを用いることで，開発者は，大規模なシステムに対して効率的に適切な修正作業を行うことができると考えられる．

- 提案手法は，モデル検査の結果，反例が 1 つ与えられていると想定している．既存のモデル検査器は，反例が存在する場合にはそのうちの 1 つを提示するので，これは妥当な仮定であると思われる．しかし，単一の反例は，性質違反の全ての原因を表すことはできない．よって，複数の性質違反の原因が存在する場合は，それらを個別に表現する複数の反例から 4VTS を構築する方が望ましいであろう．特に 3.4 節の事例研究において，われわれは修正に必要な反復回数が大きくなりすぎないようにするために，以前の反復で得られた反例も参考にして修正後の LTS を求めた．
- 本手法は，複数のプロセスからなる並行システムの修正については考慮されていない．実用的には並行システムの修正ができることが望ましいが，本手法はシステム全体の振る舞いに対する修正法であるため，方法論の拡張が必要である．解決のためには，以下の手段により，プロセス毎の個別の修正工程に帰着させるという方法が考えられる．
  - プロセスごとに領域知識を抽出して，提案手法を適用する．
  - 従来のモデル検査の結果得られる反例をプロセスごとの軌跡に分割して，提案手法を適用する．
- 提案手法は，反例が，活性の場合は無限長投げ縄型，安全性の場合は有限長であることを前提としている．2.4 節で論じたように，この想定は，既存のモデル検査器が採用している反例探索技術に基づく．投げ縄型以外の形状の反例を生成する研究は，計算木論理の検証において行われており，木に類似した構造の反例を生成する手法が提案されている [29]．このように，多様な形状をした反例を提示するすることで，開発者は，単一の構造の反例を用いるよりも，より効率的に誤りを修正することが可能となると思われる．提案したモデル修正法を，与えられた多様な形状の反例に適用するためには，3.3.2 節で説明した反例から 4VTS を構築する手続きを，反例の形状に対応して変更すればよい．したがって，投げ縄型以外の形状の反例に対しても提案手法は適用可能であると考えられる．

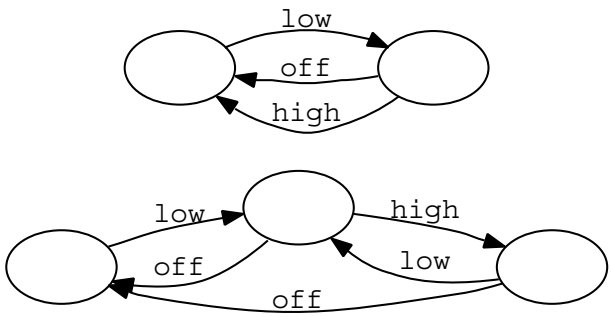


図 3.20. 図 3.11（上図）と図 3.10（下図）の最小化モデル

今後の課題

モデル修正法についての今後の課題は多くある．

これまで，提案手法の基盤モデルと反例モデルの構築，モデル合併演算，類似度の算出をソフトウェアにより自動化した．今後は，これらを統合することにより，開発者との対話的なモデル修正器を実現することが課題である．ただし，モデル修正器を作成するには，既に述べたモデルの修正を合併モデルの図式の操作によって実現する，などの機能が必要である．特に，モデルの規模が大きい場合に，効率的な修正を可能にする機能を実現しなければならない．さらに，より実用的な事例に適用して，その結果を評価することで，提案手法の利点および改善すべき点を明らかにしていく必要がある．

提案手法により修正したモデルは，修正前のモデルよりも状態数や遷移数が増加した複雑なものとなる恐れがある．実際，図 3.9，図 3.11, ならびに図 3.13 に示したモデルは，状態数がそれぞれ図 2.2 と図 3.12 のモデルよりも増加した規模の大きいモデルである．これは，モデルの合併を実施することが原因で起こる現象である．修正モデルの複雑化によって，特に次の反復が必要な場合は，基盤モデルの複雑化を引き起こす．よって，各反復の結果，できるだけ規模の小さい（すなわち，状態数，遷移数が少ない）LTS モデルが得られるのが望ましいであろう．修正モデルの複雑化を緩和するための手段として，各反復の実施後に得られた修正モデルの最小化を実行することが考えられる．LTS の最小化には，双模倣性 [25, 78, 79, 99, 5] に基づく最小化アルゴリズムが提案されている [69]．直観的には，2 つの LTS が双模倣であるとは，初期状態からの振る舞いを互いにステップごとに模倣することができる関係であり，振る舞いについての同値関係である．双模倣性に基づいて最小化したモデルを新たな基盤モデルや修正モデルとすることで，より効率的に修正を行うことができると考えられる．モデルの最小化を組込んだ修正プロセスの有効性も，更なる研究により評価されるべき課題である．

例えば，3 段階スイッチ L_1 と L_2 の修正結果である図 3.11 と図 3.10 を双模倣性に基づいて最小化したモデルを図 3.20 に示す．ここで，どちらのモデルについても最も左の状態が受理状態である．この図から分かるとおり，最小化を実施したモデルの振る舞いは，最小化を行う以前のモデルよりも開発者が理解しやすいものとなっている．

3.5 節において，われわれは，Sokolsky 等が提案した q-模倣性 [92] を用いたモデル間の類似度を導入して，提案手法の改善を図った．Sokolsky 等は同じ文献 [92] において，双模倣性に基づく LTS 間の類似度である量化双模倣性を提案している．量化双模倣性は，2 つのグラフが双模倣であるときに最も類似していると判定する類似度の尺度である．文献 [92] によると，量化双模倣性は確率過程ゲームによって計算できる．その計算量に関しては，グラフの規模に対して指数時間より悪くはないという結果が得られている．一方，本研究では，線形計画問題を解くことで計算が可能な q-模倣性を採用し，模倣等価性に基づく類似度を定義した．しかしながら，一般に，2 つのグラフが双模倣ならばそれらは模倣等価でもあるが，逆は成立しない [99, 5]．よって，模倣等価ではあるが双模倣ではないグラフの組合せが存在するので，われわれが採用した類似度と量化双模倣性の間にはモデル間の類似性の評価値が異なることがある．ゆえに，モデル修正の観点から，2 つの類似度の尺度のこのような相違が与える影響を明らかにする必要がある．そして，その結果に基づいて，どちらの指標を採用するのが適当かを更なる事例研究によって評価する必要があるだろう．

3.7 関連研究

モデル修正法と関連する研究は多く行われている．本節において，本研究との関連を議論する．

最も関連する研究に，本研究と同様にモデル検査の結果に基づく修正法の提案がある [37, 107]．ただし，本研究と異なり，対象となるモデルはクリプキ構造で記述されていることが前提である．これらの研究は，それぞれ LTL 式と CTL 式の検査が対象であり，性質を満たさない元のモデルを修正して，性質を満たす正しいモデルを構成する．修正モデルは，元のモデルとの構造的な類似度を基準として自動的に構成する．しかしながら，これらの研究が提案する手法には開発者が参加しないため，開発対象システムの問題領域において望ましいモデルが修正の結果得られるとは限らない．われわれが開発した手法は，開発者や対象領域にとって望ましいモデルを得ることが特徴である．

モデル合併は，2 つのモデルからより洗練されたモデルを構成する演算である．Uchitel と Chechik が提案する合併法 [98] では，モデルは MTS [68, 56] で記述される．モデルの合併演算は，並列合成と類似した合成規則で定義されている．Uchitel 等は，この合併技術を用いて，安全性とメッセージ系列図で記述されたシナリオから，MTS で記述されたモデルを構築するアルゴリズムを提案した [97]．この研究の目的はモデルの誤りを特定することではなく，要求を満たすモデルを合成することである．与えられた性質が安全性であるならば，この方法によって性質を満たすモデルを構築することが可能である．われわれが開発した修正法は，この研究のアイデアを用いている．しかしながら，活性を表す MTS を構築することは一般にできないので，この研究を活性に適用することは困難であり，方法論の拡張が必要である．他の合併手法は，発見的手法に基づいた方法 [83] や，モデルのグラフ構造に注目した方法である [90]．

モデル合成法は，与えられた性質を満たすモデルを構成する手法である [87, 108]．Piterman

等の方法 [87] は μ 計算 [64] とゲームを用いてモデルを合成する．しかし，適用可能な性質は LTL の特定の形式を具体化した式で表されたものに限られるという制約がある．一方，Ziller と Schneider が提案する方法 [108] は，μ 計算の式で記された性質を満たすモデルを，記号的モデル検査により構成するというものである．この技術には，構成したいモデルの潜在的な振る舞いが，全て予め与えられているという前提がある．

モデル検査に関連して近年研究が活発に行われている分野に，2.1.4 節で述べたモデルの抽象化技法がある．これは，モデル検査時に探索する状態空間を減らすために，モデルを圧縮する方法である．抽象化技法を導入することにより，モデル検査における大きな問題点である状態空間爆発を回避することができる．この抽象化技法を用いることで，モデル検査を効率化する手法が提案されている [24]．この方法は，抽象化したモデルにモデル検査を実施し，得られた反例を元に，抽象化したモデルと元のモデルでモデル検査の結果が同じになるように，反復的にモデルの抽象化関数を更新する方法である．われわれのモデル修正法は，この方法と同様の考え方を元にしており，反復的な方法を採用している．

ソフトウェアテストにおいては，テストで発見されたプログラムにある虫の修正を自動化する手法が提案されている [4]．この研究は，遺伝的プログラミング技術を用いて対象のプログラムを修正し，より多くのテストに合格するプログラムを発見する試みである．

3.8　まとめ

本章では，モデル検査に基づいたモデル修正問題の概念を定め，それを解決するための方法を提案した．提案した手法は，反復的な修正プロセスである．提案手法は，反例に基づく方法であるので，性質が安全性の場合も活性の場合も同様に適用することが可能である．加えて，活性については公平な選択を仮定した検査にも適用できる．また，領域知識を用いた対話的な手法であるため，開発者が対象領域において望ましいと判断するモデルを構築することもできる．

続けて，われわれは，提案する修正法の改善のためにモデル間の類似度を導入する手法を議論した．この改善手法は，開発者にモデル修正に有用な情報を提供することができる．

これまでなされてきた研究の多くが注目しているのは，主に検証手法やその効率化であり，性質を満たしていないと判定されたモデルの修正技術にはあまり研究上の関心が集まることがなかった．しかし，特に要求分析や設計においては，モデルを用いて構築すべきシステムを理解するということがしばしば行われる．この場合，性質を満たす正しいモデルが得られなければ，それに基づいて正しいシステムを構築することもまたできない．それゆえ，提案する手法が，産業界においてモデル検査技術を活用した要求分析，設計工程の実施に貢献すると思われる．

第 4 章

反例を用いた誤り特定法

3 章では，モデル検査の結果に基づいて，反例を用いたモデルの修正技法について論じた．開発したモデル修正法は，反例のモデルと基盤モデルを合併したモデルを修正の候補として開発者に提供するが，反例のどの部分が性質違反の原因であるかの判断は，開発者が行わなければならない．既に論じたように，複数のプロセスからなる並行システムのような複雑な振る舞いを示すシステムの場合には，この作業は開発者にとって大きな負担となり，信頼性の高いソフトウェアの開発の障害となる．ただし，モデル検査器が提供する反例は，対象モデルの誤りや正すべき箇所を明らかにするための手掛かりとなるので，それが指し示すモデル上の誤りは，モデルの修正作業においても開発者にとって有用な手掛かりとなる．よって，開発者によるモデル修正作業を支援するためには，このようなモデル上の修正すべき誤りや反例の修正候補を特定する手法を確立することが望ましい．

これまで，モデル検査の結果に基づいて，反例の説明を生成する手法や，対象の誤りを特定する手法は数多く提案されている [7, 49, 19, 48, 47, 62, 93, 41]．これらの研究は，プログラムを対象とした手法 [7, 49, 19, 48, 47, 62] と，論理回路を対象とした手法 [93, 41] に分けられる．既存手法の目的は，プログラムや論理回路中の誤った構成要素を発見する手法である（例えば，プログラム中の代入文や，論理回路を構成する素子）．以上の先行研究については，5.6 節で議論し，ここでは概要のみを述べる．

プログラムを対象とした既存の手法のうち，文献 [7, 49, 48, 47] は，安全性の違反に対してのみ適用可能な方法である．安全性に対しては，有限長の反例を調べるだけでよく，軌跡に含まれる誤った事象を自動的に特定するのは容易である．しかしながら，活性に対する誤りの特定は，安全性に対して誤りを特定する問題よりも困難な課題である．活性に対してモデルやプログラムの誤りを特定するためには，無限長の反例を分析する必要がある．文献 [7, 49, 48, 47] の手法は，軌跡が有限長であることを前提としているので，無限長の軌跡に適用するのは困難である．先行研究には活性を取り扱うことを目標としたものもある [19, 62]．しかし，計算量が非常に大きい [19]，活性のうち特定の形式の論理式で表される性質のみを対象としている [62] 等の制約がある．

論理回路の誤り特定を目標とした研究の多くは，論理回路の特性に注目しているので，LTS で記述されたモデルに適用するのは難しい [93, 41]．

そこで，本章では，モデルが性質を満たさない場合に，モデルの誤り箇所と反例の修正候補を自動的に計算して開発者に提示する新しい手法である LLL-F（Lightweight error Localization for Labeled transition systems - First version）を提案する [109]．LLL-F は，投げ縄型の形状をなす無限長の反例が分析できないという先行研究の問題点を解決する．本手法を用いることで，開発者によるモデル修正を支援することができる．

LLL-F のアイデアは次の通りである．

1. 反例と最も類似した正例を求める．ここで，正例は性質を満たす軌跡であり，反例と同様に投げ縄型で無限長の軌跡であるとする．
2. 求めた正例と反例を比較し，両者の違いを抽出する．この違いは，反例が性質を満たすように修正されるべき事象を表していると考えられる．LLL-F はその違いを引き起こす遷移を誤りの候補として，また，各誤り候補に対応する正例を誤りの説明として開発者に提示する．

このアイデアを実現するために技術的に解決すべき課題を以下に示す．

1. 求めるべき正例の集合を与える方法の確立．
2. 軌跡間の類似度を表現するための，軌跡間の距離の指標の確立．ただし，モデルの誤りの特定を適切に行うことができる指標でなくてなくてはならない．
3. 課題 1 と 2 が解決した場合に，与えられた正例の集合から適切な正例を探索する手法の確立．

課題 1 は，検証した性質と等価なテスタオートマトン（TA）を構築し，これを正例の集合とすることによって解決できる．

課題 2 の解決のため，LLL-F は軌跡間の編集距離を距離の指標とする． ただし，有限長文字列の編集距離を扱うために反例を接頭辞と閉路に分割し，以下のような発見的な基準を導入する．まず，反例の接頭辞と編集距離が最小の事象列を，求める正例の接頭辞とする．加えて，反例の閉路と編集距離が最小で，その接頭辞に続く事象列を正例の閉路とする．

課題 3 は，性質の TA 上で正例を探索する問題を，有向グラフ上の古典的な最短経問題に帰着させることで解決する．ただし，正例の接頭辞は，性質の TA の受理状態で終わる事象列とする．また，LLL-F は，反例が有限長であり，正例が無限長となる安全性の検証にも適用できる．

LLL-F の主な利点は以下の通りである．

- LLL-F は，活性と安全性を含む全ての FLTL 式で記述可能な性質に対して適用することができる．また，既存のモデル検査器の出力を直接用いることができる．前者は，任意の FLTL 式は，その式と等価な TA に変換できるという事実 [45] に基づく．
- LLL-F は LTS で記述されたモデルに含まれる誤りの特定を目標としているので，要求分析，設計工程において活用できる．特に，反例の修正候補となる正例という有用な情報を開発者に提示できる．

- LLL-F において，正例の集合は性質の TA で与えられるので，開発者がそれを明示的に与える必要がない．
- LLL-F は，既に多くの研究成果が存在するグラフ探索アルゴリズムやモデルの合成技術を用いたモデルに基づく方法であり，自動化が容易である．特に，上で述べたような発見的な基準に基づいて正例を探索するので，探索を高速に行うことができ，実用的な性能を達成できる．

まず，4.1 節で，提案する誤り箇所特定および修正候補提示法 LLL-F を説明する．ここで述べる手法は活性に対して適用される手法なので，安全性に適用できる方法を 4.2 節で議論する．

4.3 節では，LLL-F を自動化したプロトタイプツールの実装について報告し，続いて 4.4 節においてそのプロトタイプツールを使って行った事例研究の結果を述べる．事例研究で評価を行った項目は次の 2 種類である．

- 7 種類のシステムについてモデルを構成し，LLL-F を実行する．そして，その結果が適切にモデルの誤りを指摘しているかを調べる．
- LLL-F の実行性能に影響を与える要因である反例，および性質の TA の規模を変化させ，それぞれの場合について LLL-F の実行時間の変化を調べる．

4.5 節にて，手法についての考察を行い，今後の課題も明らかにする．最後に，4.6 節で本章をまとめる．

4.1 誤り特定手続き

本節では，提案する誤り特定，反例修正候補提示法 LLL-F を説明する．LLL-F の鍵となるアイデアは，性質を満たす軌跡（正例）のうち，反例と類似し，かつその反例を修正するものを求める点にある．正例が反例の修正候補であることの基準は文字列間の編集距離 [71, 82, 33] とし，反例からの編集距離が最小の正例を探索する．この正例は，性質違反の原因である事象のみが反例と異なると考えられるので，両者の差分により誤り箇所の候補を発見できる．ただし，無限長の反例で編集距離を直接扱うことは困難なので，反例を有限長である接頭辞と閉路の 1 周期に分割して，それぞれの編集距離が最小となる正例を探索する，という発見的な手法を導入する．

LTS で記述された検証対象のモデルを $L =< S, A, \Delta, s_0 >$，FLTL で記述された性質を ϕ とする．ただし，$L \not\models \phi$ である．P，C を有限長の事象列とし，反例を $\pi = PC^\omega \in Tr(L)$ と表す．正例の集合として，$TA(\phi)$ が受理する全ての軌跡を取る．ただし，性質 ϕ と等価な TA $TA(\phi) =< S_t, A_t, \Delta_t, t_0, S_t^{acc} >$ の事象の集合 A_t において，$A_t \subseteq A$ とする．本論文では，2 重深さ優先探索法に基づき，正例は，有限長事象列 P'，C' に対して $\tau = P'C'^\omega$ の投げ縄型構造をなすとする．ここで，P' は $TA(\phi)$ の受理状態に達する事象列で，C' はその受理状態から始まり，その状態を無限回通過する閉路である．π を修正する正例は，P との編集距離が最

小の接頭辞 P' を持つ正例のうち，C との編集距離が最小の閉路 C' を持つものを，$TA(\phi)$ が受理する軌跡の集合から探索して求める．

2 つの文字列の編集距離は，一方の文字列を他方の文字列に変換するのに必要な編集コストの最小値で表される [82, 33]．本論文で想定する操作は以下の 3 種類とし，各操作の編集コストは互いに等しく 1 とする．

- **置換操作**：一方の文字列の文字を異なる文字に置き換える．
- **削除操作**：一方の文字列の文字を削除する．
- **挿入操作**：一方の文字列に文字を挿入する．

A の要素である各事象を文字とみなすと，L の軌跡は文字列と考えられるので，軌跡に現れる事象の削除，異なる事象への置換，事象の挿入を軌跡への編集操作とする．

LLL-F では，P と C からそれぞれ重み付き遷移系（WTS）と呼ぶモデルを構成する．WTS は LTS を拡張して，各遷移に重みを付加した状態遷移機械である．

P の WTS が表す事象列の集合は，P に任意の回数の編集操作を適用した文字列の集合である．各編集操作は WTS の遷移関係で表され，その編集コストは遷移関係に対する重みで表す．まず，接頭辞 $P = [a_0, \ldots, a_{m-1}]$ に対しては，状態 p_i と遷移 (p_i, a_i, p_{i+1}) $(i = 0, \ldots, m-1)$ を生成する．以上で生成した遷移は反例と同一の軌跡を表すので，重みを 0 とする．次に，これらの遷移を，編集操作に相当する以下の 3 種類の遷移によって拡充する．

1. **置換操作**：状態の組 (p_i, p_{i+1}) に対して，重み 1 を持つ遷移 (p_i, a, p_{i+1}) を生成する．ここで，$a \notin A - \{a_i\}$ である．この遷移は，a_i を a に置換する編集操作を表す．
2. **削除操作**：状態の組 (p_i, p_{i+1}) に対して，重み 1 を持つ遷移 (p_i, ϵ, p_{i+1}) を生成する．この遷移は，a_i を削除する編集操作を表す．事象 ϵ は事象の削除を表す特殊な事象である．
3. **挿入操作**：状態 p_i に対して，重み 1 を持つ自己閉路 (p_i, a, p_i) を生成する．この遷移は，$a \in A$ を p_i（a_{i-1} と a_i との間）に挿入する編集操作を表す．

また，P の WTS は，P の事象列の末尾を指し示す状態（末尾状態）を持つ．P あるいは P を編集した事象列を，P の WTS の初期状態から末尾状態までに通過する遷移に付加された事象からなる有限長の事象列により表す．その事象列が通過する各遷移に伴う重みの総和が反例からの編集距離である．

閉路 C からも P のときと同様に WTS を構築する．C の WTS は P の WTS と同じ特徴を持つが，正例は $TA(\phi)$ において Büchi の受理条件を満たす軌跡でなければならない．そのためには正例の閉路は空列であってはならないので，C を編集した事象列が空列とならないように WTS を構築する．

$TA(\phi)$ と P の WTS との積を，$TA(\phi)$ の受理状態に達し，かつ P に編集操作を適用した事象列の集合を表すように定める．同様に，$TA(\phi)$ と C の WTS との積で，$TA(\phi)$ の受理状態を含む強連結成分で，かつ C に編集操作を適用して得られる事象列の集合を表す．加えて，編集コストが各遷移の重みとなるように積を定めることで，正例 $\tau = P'C'^{\omega}$ の探索を，両方

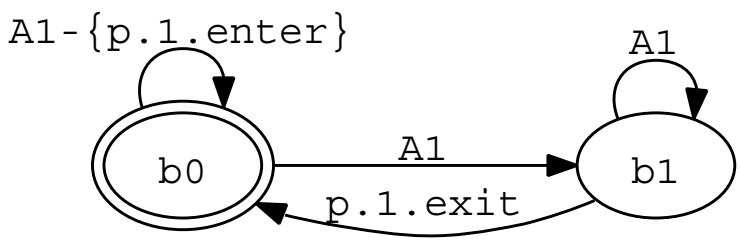


図 4.1. TA(EXIT_1)

の有向グラフに対する 2 段階の最短路探索問題 [33] に帰着させる．$TA(\phi)$ と P の WTS との積に対する第 1 の最短路探索により正例の接頭辞 P' を求め，$TA(\phi)$ と C の WTS との積に対する第 2 の最短路探索によって正例の接頭辞 C' を求める．

π と τ との差分は誤っている可能性のある事象を指し示す．LLL-F は，求めた全ての正例に対して反例との違いを見つけ，π を τ に変換するために修正されるべき π の事象を遷移ラベルとして持つ遷移を，誤りの候補とする．LLL-F の出力は，反例を修正した正例と，対応する誤りの候補である．

誤り特定，反例修正候補提示法への入力は，対象のモデル L，検査する性質 ϕ（もしくはその TA $TA(\phi)$），性質に対する反例 π である．以上をまとめると，LLL-F は次の手順で実施される．

1. 反例 $\pi = PC^\omega$ の P と C を WTS に変換する．
2. P の WTS と性質の TA $TA(\phi)$ との積を求める．そして，その積モデルに対する最短路探索により $TA(\phi)$ の各受理状態に相当する状態までの事象列 P' を得る．
3. C の WTS と $TA(\phi)$ との積を求める．そして，$TA(\phi)$ の各受理状態に相当する状態を始点とし，その受理状態に戻る事象列 C' を，その積モデルに対する最短路探索により得る．
4. π と正例 τ との差分を求めて，モデル上の誤り箇所を特定する．

4.1.1 節から 4.1.5 節までで，各ステップの詳細を説明する．なお，今後の議論では，以下の例を用いて提案手法の説明を行う．

例 26. 2 章で挙げた並行システムにおける性質 EXIT_1= $\mathbf{G}(p.1.enter \Rightarrow \mathbf{F}p.1.exit)$ の検証の例を考えよう．図 4.1 に EXIT_1 の TA TA(EXIT_1) を示す．この TA(EXIT_1) が受理する軌跡の集合が正例の集合である．われわれは，この軌跡の集合から，反例との編集距離に基づいて正例を探索する．b_0 が初期状態，2 重丸が受理状態を表す．また，図 2.3 の事象の集合を A_1 とする．

この図において，状態 p，q をそれぞれ出状態，入状態とし，事象の集合 $A \subseteq A_1$ の全ての要素を事象とする遷移集合が存在する場合，それを (p, A, q) と表す．すなわち，状態 p，q について，全ての $a \in A$ に対して，$(p, a, q) \in \Delta_1$ ならば，$(p, A, q) = \{(p, a, q) \in \Delta_1 | a \in A\}$ と書き表す．ここで，Δ_1 は，TA(EXIT_1) の遷移関係を表す．

例えば，(b_0, A_1, b_1) は集合 $\{(b_0,\ p.\{1,2\}.enter,\ b_1),\ (b_0,\ p.\{1,2\}.exit,\ b_1),\ (b_0,\ p.\{1,2\}.mutex.up, b_1), (b_0, p.\{1,2\}.mutex.down, b_1)\}$ を表す．同様に，$(b_0, A_1 - \{p.1.enter\}, b_0)$ は遷移の集合 $(b_0,\ A_1,\ b_0)$ から $(b_0,\ p.1.enter,\ b_0)$ を除いた残りの遷移集合を意味する．つまり，遷移の集合 $\{(b_0,\ p.2.enter,\ b_0),\ (b_0,\ p.\{1,2\}.exit,\ b_0),\ (b_0,\ p.\{1,2\}.mutex.up,\ b_0),\ (b_0,\ p.\{1,2\}.mutex.down,\ b_0)\}$ の省略表記である．

■

4.1.1　接頭辞モデル

本節では，π の接頭辞 P から WTS を構成する．まず，WTS を各遷移に対して重みを持った LTS として定義する．

定義 14.（**重み付き遷移系 (WTS)**）重み付き遷移系（WTS）W を次のように定める：$W =< S_w, A_w, \Delta_w, q_0, \zeta, M_w >$．ただし，各要素は次のように定められる．

- S_w は状態の有限集合である．
- A_w は事象の有限集合である．
- $\Delta_w \subseteq S_w \times A_w \times S_w$ は遷移関係である．
- $q_0 \in S_w$ は初期状態である．
- 全関数 $\zeta : \Delta_w \to \mathcal{R}$（$\mathcal{R}$ は実数の集合）は各遷移に付加される重みである．
- $M_w \subseteq S_w$ は末尾状態の集合である．

■

以下では，軌跡，経路，入状態，出状態，終端状態などの LTS に関する用語を WTS に関しても同様に用いる．

反例の接頭辞 P を以下のように WTS に変換する．P の WTS は，P と P に対して編集操作を適用した有限長事象列を表すモデルである．

定義 15.（**接頭辞モデル**）A を対象モデルの事象の集合とし，長さ m の事象列 $P = [a_0,\ a_1, \ldots, a_{m-1}]$ が与えられているとする．ただし，$a_i \in A\ (0 \leq i < m)$ である．このとき，WTS $W_P =< S_P, A_P, \Delta_P, p_0, \zeta_P, M_P >$ を接頭辞モデルと呼ぶ．ただし，各要素を以下の通り定める．

- $S_P = \{p_i | 0 \leq i \leq m\}$,
- $A_P = A \cup \{\epsilon\}$,
- $M_P = \{p_m\}$,
- $\Delta_P \subseteq S_P \times A_P \times S_P$ は遷移関係であり，次のように定義される．$\Delta_P = \{(p_i, a, p_i) | 0 \leq i \leq m, a \in A\} \cup \{(p_i, a, p_{i+1}) | 0 \leq i < m, a \in A_P\}$,
- $\delta_P \in \Delta_P$ に対して，$\delta_P = (p_i, a_i, p_{i+1})\ (0 \leq i < m)$ の場合は $\zeta_P(\delta_P) = 0$ とし，その

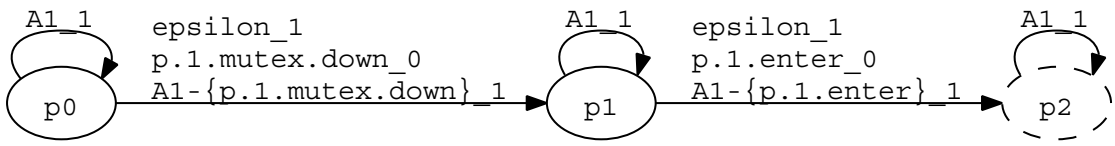


図 4.2. 接頭辞モデル W_P

他の場合は $\zeta_P(\delta_P) = 1$ とする．

■

W_P の状態 p_i は，P の文字 a_i に対して何らかの編集操作を適用することを表す．ただし，末尾状態 p_m は，事象列 P に対して編集操作を適用した結果得られる事象列の末尾を表す状態である．各遷移は，以下に述べる通り，編集操作を意味する．

- 置換操作：状態 p_i から隣接する状態 p_{i+1} への遷移 (p_i, a, p_{i+1}) は，事象 a_i を a に置換する操作である．
- 削除操作：a_i から特殊な事象 ϵ への置換操作を表す遷移 (p_i, ϵ, p_{i+1}) は，a_i を削除する操作である．
- 挿入操作：p_i の自己閉路 (p_i, a, p_i) は，a_{i-1} と a_i との間に事象 a を挿入する操作である．

ただし，状態 p_0，状態 p_m の自己閉路はそれぞれ P の先頭，および末尾への挿入操作である．各操作の編集コストは遷移に付加する重みで表す．既に述べた通り，置換，挿入，削除操作の重みは 1 である．

例 27. 図 4.2 に反例 $\pi_{\texttt{EXIT\_1}}$ の接頭辞 [*p.1.mutex.down*, *p.1.enter*] から構成した接頭辞モデル W_P を示す．図で p_0 が初期状態で，点線の楕円 p_2 が末尾状態であり，事象 ϵ は *epsilon* で表す．図の表記法は図 4.1 と同様であるが，各遷移の重みを事象の後ろに記す．例えば，(p_0, A_1, p_0) や $(p_1, A_1 - \{p.1.enter\}, p_2)$ の事象の後の値 1 は，これらが表す全ての遷移の重みが同じ値 1 であることを表す．なお，これらの 2 種類の遷移は，それぞれ $\pi_{\texttt{EXIT\_1}}$ の接頭辞の先頭への事象の追加（挿入）操作と，2 つ目の事象 *p.1.enter* を異なる事象に置換する操作を表す．また，$(p_0, p.1.mutex.down, p_1)$ と $(p_1, p.1.enter, p_2)$ は，反例の接頭辞の各事象 *p.1.mutex.down* と *p.1.enter* を自身に置換する操作であり，反例の接頭辞に対して置換操作，削除操作のどちらの編集操作も行われないことを表している．したがって，この場合の遷移の重みは 0 である．

■

4.1.2　最短路探索による接頭辞探索

本節では，反例の接頭辞 P との編集距離が最小で，性質 ϕ の TA $TA(\phi)$ の受理状態に達する事象列を求める方法を述べる．接頭辞モデル W_P と $TA(\phi)$ の積 $TA(\phi) \bowtie W_P$ を求めることで，この事象列を求める問題を有向グラフ上の最短路問題 [33] に帰着させる．積を求める演算 $\bowtie$ は，2 つの LTS の並列合成演算を WTS と LTS に対する演算に拡張したものである．直観的には，$TA(\phi) \bowtie W_P$ は，$TA(\phi)$ の各遷移を，P に適用される編集操作のコストでラベル付けしたものである．各編集操作の編集コストは各遷移の重みで表される．また，$TA(\phi)$ の受理状態と W_P の末尾状態から構成される状態を $TA(\phi) \bowtie W_P$ の末尾状態とすると，この末尾状態が $TA(\phi)$ の受理状態に相当し，P を編集した事象列が到達すべき状態になる．すなわち，$TA(\phi) \bowtie W_P$ の初期状態から末尾状態までの重みの合計が最小の経路が，編集距離が最小の経路である．したがって，この経路を通過する事象列が正例の接頭辞である．この経路は，Dijkstra 法 [36] による最短路探索で求めることができる．

定義 16.（**WTS と TA の積**）WTS $W =< S_w, A_w, \Delta_w, q_0, \zeta, M_w >$ と TA $L =< S_t, A_t, \Delta_t, t_0, S_t^{acc} >$ において $A_t \subseteq A_w$ とする．両者の積もまた WTS であり，以下のように表される．

$$L \bowtie W =< S_t \times S_w, A_w, \Delta'_w, (t_0, q_0), \zeta'_w, S_t^{acc} \times M_w >$$

ここで，$\Delta'_w \subseteq (S_t \times S_w) \times A_w \times (S_t \times S_w)$ は遷移関係であり，以下に定める Δ'_{w_1},Δ'_{w_2} により，$\Delta'_w = \Delta'_{w_1} \cup \Delta'_{w_2}$ で表される．

- $\Delta'_{w_1} = \{((s_t, s_w), a, (s'_t, s'_w)) | (s_t, a, s'_t) \in \Delta_t, (s_w, a, s'_w) \in \Delta_w\}$
- $\Delta'_{w_2} = \{((s_t, s_w), a, (s_t, s'_w)) | s_t \in S_t, (s_w, a, s'_w) \in \Delta_w, a \notin A_t\}$

また，$\delta_w = ((s_t, s_w), a, (s'_t, s'_w)) \in \Delta'_w$ に対して，$\zeta'_w : \Delta'_w \to \mathcal{R}$ は $\zeta'_w(\delta_w) = \zeta((s_w, a, s'_w))$ である．

■

上の定義において, TA と WTS の積の末尾状態は, TA の受理状態であり, かつ, WTS の末尾状態であるような状態と定義されている．よって，正例の接頭辞を求めるには，$TA(\phi) \bowtie W_P$ の初期状態から各末尾状態までの最短路を探索すればよい．最短路は状態列なので，全ての最短路に対して，各状態を通過して得られる事象列を求める．

探索の効率化と正例の組合せ数の爆発を防ぐため，以下の前処理を $TA(\phi) \bowtie W_P$ に行う．これは，最短路探索において不要な遷移と状態を削除する処理であり，初期状態から各状態への最短路を保存する．

1. 全ての状態の自己閉路を除く．
2. 出状態と入状態が等しく，事象のみが異なる遷移が複数ある場合，重みが最小のもの 1

つ以外は全て削除する．

3. 末尾状態でない終端状態とそこへ入る遷移を全て除く．
4. 状態と遷移が削除された場合はステップ 3 に戻り，そうでない場合は終了する．

定義 16 において，$A_t \subseteq A_w$ を仮定しているが，この条件を導入した理由は次の通りである．この条件を満たさず，LTS の並列合成と全く同様に WTS と TA の積を定めると，両者の積の末尾状態への事象列には，重みが未定義となるものが存在してしまう．このような遷移は編集操作が適用できないことを表すので，$A_t \subseteq A_w$ となることが必要である．接頭辞モデルと性質の TA の観点から述べると，TA で受理状態に達する任意の事象列に対して，WTS の末尾状態への事象列が存在することが必要である．接頭辞に対して適用しうる編集操作を L の事象の集合に対して定義すれば，接頭辞モデルの事象の集合は L の事象の集合となる．また，$TA(\phi)$ は L について成り立つべき主張を表すので，$TA(\phi)$ の事象の集合は，L の事象の集合の部分集合としてよい．ゆえに，実用上は定義 16 の仮定を満たすようにモデルを構築することができると考えられる．

例 28. 反例 $\pi_{\texttt{EXIT\_1}}$ の接頭辞を考える．図 4.1 の $TA(\texttt{EXIT\_1})$ と図 4.2 の接頭辞モデル W_P との積 $TA(\texttt{EXIT\_1}) \bowtie W_P$ を図 4.3 に示す．図 4.2 と同様に，末尾状態を点線の楕円で表す．この積の末尾状態は (b_0, p_2) であるが，これは図 4.1 の受理状態に相当し，かつ，図 4.3 の末尾状態に相当する．したがって，正例の接頭辞を求めるには，このモデルに対して，初期状態 (b_0, p_0) から末尾状態 (b_0, p_2) までの最短路探索を行えばよい．この結果，最短路として，距離 1 の状態列 $[(b_0, p_0), (b_1, p_1), (b_0, p_2)]$ が得られる．この状態列を通過する事象列を求めることで，編集距離が最小の事象列の例として [*p.1.mutex.down*, *p.1.exit*] が得られ，これが求める正例の接頭辞である．これは $\pi_{\texttt{EXIT\_1}}$ の接頭辞の 2 番目の事象 *p.1.enter* を *p.1.exit* に置換したもので，編集距離は最小の 1 である．

■

一般に，編集距離が最小であるような最短路は複数存在する．複数の最短路が存在する場合は，それら全てについて各状態を通過する事象列を求める．

例 29. 図 4.3 において，状態列 $[(b_0, p_0), (b_0, p_1), (b_1, p_2), (b_0, p_2)]$ もまた距離 1 であり，末尾状態への最短路である．このとき得られる事象列は，[*p.1.mutex.down*, *p.1.enter*, *p.1.exit*] である．これは，$\pi_{\texttt{EXIT\_1}}$ の接頭辞の事象 *p.1.enter* の後に *p.1.exit* を挿入操作によって追加した事象列である．この事象列も正例の接頭辞となる事象列である．

■

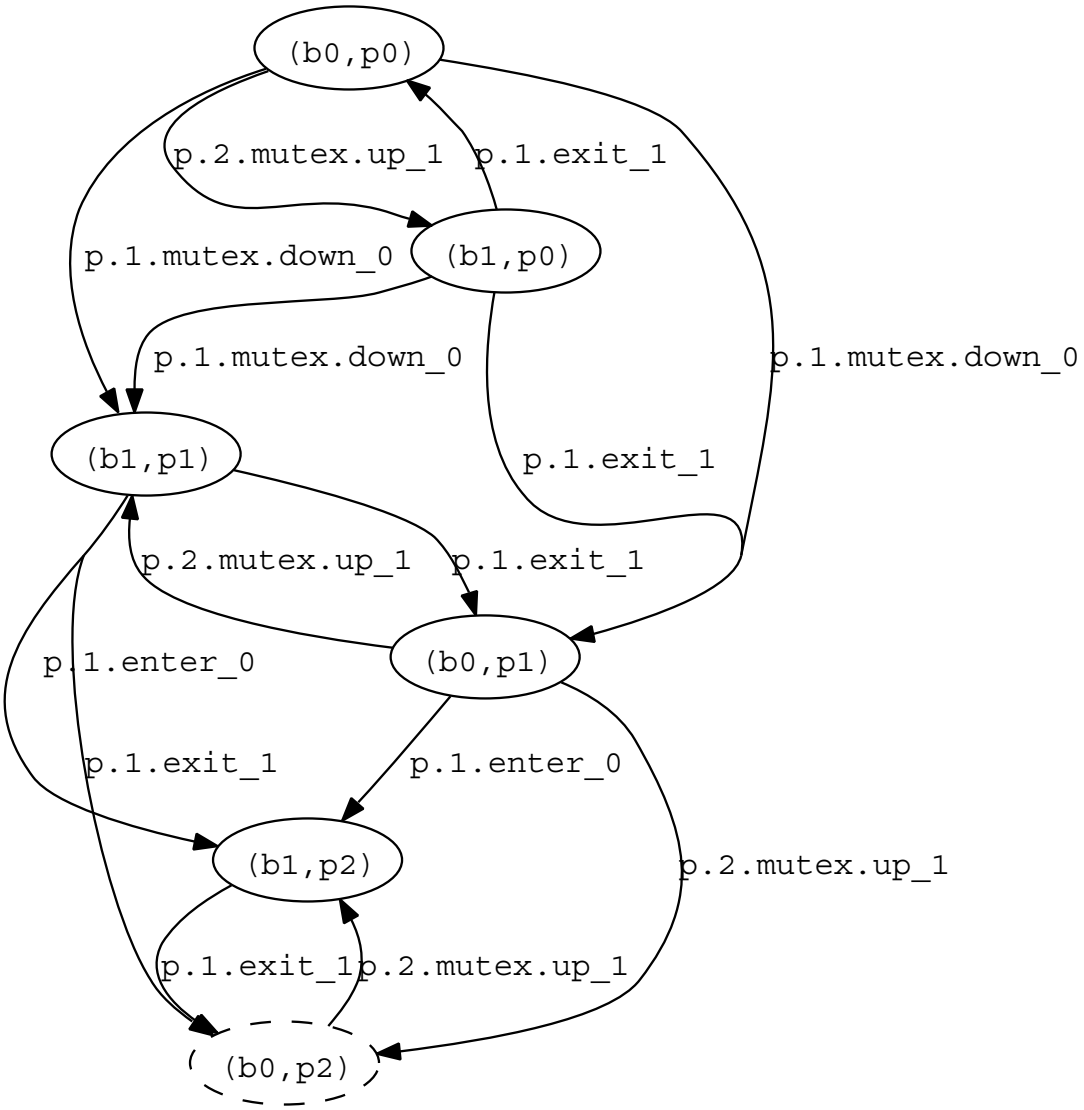


図 4.3. $TA(\texttt{EXIT\_1}) \bowtie W_P$

4.1.3　閉路モデル

反例の閉路の事象列 C との編集距離が最小の事象列を求めるために，接頭辞の場合と同様に C から WTS を構成する．ただし，先に論じた通り，探索する正例は無限長なので，削除操作の適用によって閉路に空列が得られないようにしたい．そこで，文字列長未満の回数の削除操作のみが可能なように WTS を構成する．

定義 17. （閉路モデル）A を対象モデルの事象の集合とし，長さ n の事象列 $C = [a_0,\ a_1, \ldots a_{n-1}]$ が与えられているとする．ただし，$a_i \in A$ $(0 \leq i < n)$ である．このとき，WTS $W_C =< S_C, A_C, \Delta_C, c_0, \zeta_C, M_C >$ を閉路モデルと呼ぶ．ここで，W_C の各構成要素は以下の通り定められる．

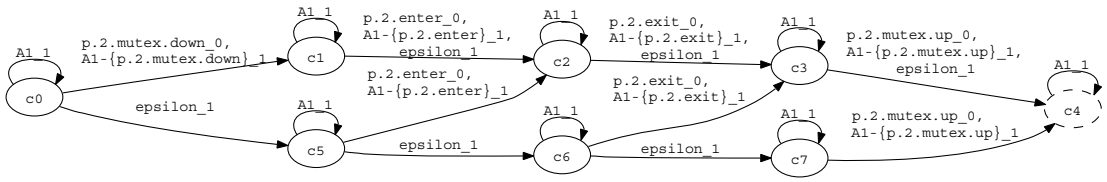


図 4.4. 閉路モデル W_C

- $S_C = \{c_i | 0 \leq i < 2n\}$,
- $A_C = A \cup \{\epsilon\}$,
- $M_C = \{c_n\}$,
- $\Delta_C \subseteq S_C \times A_C \times S_C$ は遷移関係であり，次のように定義される．$\Delta_C = \Delta_C^1 \cup \Delta_C^2 \cup \Delta_C^3 \cup \Delta_C^4$, ただし，$\Delta_C^1$, Δ_C^2, Δ_C^3, Δ_C^4 は次の通りである．
  - $\Delta_C^1 = \{(c_i, a, c_i) | 0 \leq i < 2n, a \in A\}$,
  - $\Delta_C^2 = \{(c_i, a, c_{i+1}) | 1 \leq i < n, a \in A \cup \{\epsilon\}\}$,
  - $\Delta_C^3 = \{(c_0, \epsilon, c_{n+1}) | 1 < n\} \cup \{(c_{i+n}, \epsilon, c_{i+n+1}) | 1 \leq i < n-1\}$,
  - $\Delta_c^4 = \{(c_0, a, c_1) | a \in A\} \cup \{(c_{i+n}, a, c_{i+1}) | 1 \leq i < n, a \in A\}$.
- $\delta_C \in \Delta_C$ に対して，以下のいずれかの場合は $\zeta_C(\delta_C) = 0$ とし，その他の場合は $\zeta_C(\delta_C) = 1$ とする．
  - $\delta_C = (c_i, a_i, c_{i+1})$ $(0 \leq i < n)$,
  - $\delta_C = (c_{i+n}, a_i, c_{i+1})$ $(1 \leq i < n)$.

■

閉路モデルは，削除操作の実行可能回数が閉路の事象列長未満となるように接頭辞モデルを拡張したものである．各遷移の意味は以下の通りである．

- 置換，削除操作：遷移 (c_i, a, c_{i+1}), (c_{i+n}, a, c_{i+n+1}), (c_{i+n}, a, c_{i+1}) は，事象 a_i を a に置換する操作である．特に，$a = \epsilon$ のとき，a_i を削除する操作である．ただし，(c_0, ϵ, c_{n+1}) は a_0 の削除を表す．
- 挿入操作：c_i の自己閉路 (c_i, a, c_i), (c_{i+n}, a, c_{i+n}) は，a_{i-1} と a_i との間に事象 a を挿入する操作である．ただし，(c_0, a, c_0) と (c_n, a, c_n) は，事象列の先頭と末尾に事象 a を追加する操作である．

例 30. $\pi_{\texttt{EXIT\_1}}$ の閉路 [*p.2.mutex.down*, *p.2.enter*, *p.2.exit*, *p.2.mutex.up*] の閉路モデル W_C を図 4.4 に示す．$\pi_{\texttt{EXIT\_1}}$ の閉路の文字列長が 4 なので，図 4.4 は定義 17 で $n = 4$ の場合のモデルである．図の末尾状態や各遷移の表示方法は図 4.2 の場合と同様であり，c_0 が初期状態，c_4 が末尾状態である．定義 17 の Δ_C^1 からは，挿入操作を表す各状態の自己閉路が構成される．Δ_C^2 は C に含まれる各事象の置換操作と削除操作を隣接する状態への遷移として定める．この場合は，c_1 から c_2，c_2 から c_3，c_3 から c_4 への遷移として表される．Δ_C^4 が構成する遷

移は，c_0 から c_1 への遷移，および c_5 から c_2，c_6 から c_3 および c_7 から c_4 への遷移である．それ以外の削除操作のみを表す遷移は Δ_C^3 により構成される．Δ_c^4 により得られる遷移には置換操作のみ含まれ削除操作は含まれないので，各事象の削除が行われず，置換のみが許される遷移を表している．これは，先に述べた削除回数の制限についての条件を満たすために必要である．図 4.4 より，c_0 から c_4 への全ての事象列で，削除操作（事象 ϵ）が現れるのは高々 3 回である．よって，削除操作の適用により空列が得られることはない．

■

4.1.4　閉路探索

求めた接頭辞に続き，かつ，反例の閉路に対して編集距離が最小の事象列を求める方法は，接頭辞の場合とほぼ同様である．ただし，$TA(\phi) \bowtie W_C$ では正しく閉路を求めることはできない．TA が受理する軌跡の閉路が開始される状態は，前節で求めた接頭辞により達する受理状態である．また，閉路はこの受理状態を無限回通過しなければならない．そこで，以下のように $TA(\phi)$ の初期状態と受理状態の集合を，前節で得られた接頭辞が到達する受理状態となるように変更する．

$TA(\phi) =< S_t, A_t, \Delta_t, t_0, S_t^{acc} >$ とし，M_w を接頭辞モデル W_P の末尾状態の集合とする．$TA(\phi) \bowtie W_P$ の末尾状態のうち，最小距離で到達する状態を $(s_t^{acc}, q_M) \in S_t^{acc} \times M_w$ とする．このとき，W_C との積を計算する前に，$TA(\phi)$ に対して次の手続きを実行する．

1. $TA(\phi)$ の初期状態を s_t^{acc} に変更する．
2. 求める事象列が閉路をなすようにするために，s_t^{acc} を受理状態とし，それ以外の受理状態を受理状態の集合から除く．

$TA(\phi)$ に上述の操作を適用することで得られる TA は $TA'(\phi) =< S_t, A_t, \Delta_t, s_t^{acc}, \{s_t^{acc}\} >$ である．

$TA'(\phi) \bowtie W_C$ に対して，4.1.2 節の方法により，反例の閉路との編集距離が最小の事象列を探索する．そして，4.1.2 節で求めた各接頭辞に対して，対応する全ての閉路を加えて正例とする．

例 31. $\pi_{\texttt{EXIT\_1}}$ の場合，求めた正例の接頭辞により到達する $TA(\texttt{EXIT\_1})$ の状態は b_0 であるので，$TA'(\texttt{EXIT\_1})$ の初期状態は図 4.1 の b_0 である．$TA'(\texttt{EXIT\_1})$ と図 4.4 の閉路モデルの積を図 4.5 に示す．初期状態は一番上に配置されている状態 (b_0, c_0) であり，末尾状態は (b_0, c_4) である．この積に対して，接頭辞の場合と同様に，(b_0, c_0) を始点とする最短路探索を行うことによって，(b_0, c_4) への距離 0 の最短路 $[(b_0, c_0), (b_0, c_1), (b_0, c_2), (b_0, c_3), (b_0, c_4)]$ が得られる．よって，正例の閉路は $\pi_{\texttt{EXIT\_1}}$ の閉路と同一の事象列 [*p.2.mutex.down*, *p.2.enter*, *p.2.exit*, *p.2.mutex.up*] となる．以上より，既に求めた接頭辞と組合わせることにより，$\pi_{\texttt{EXIT\_1}}$

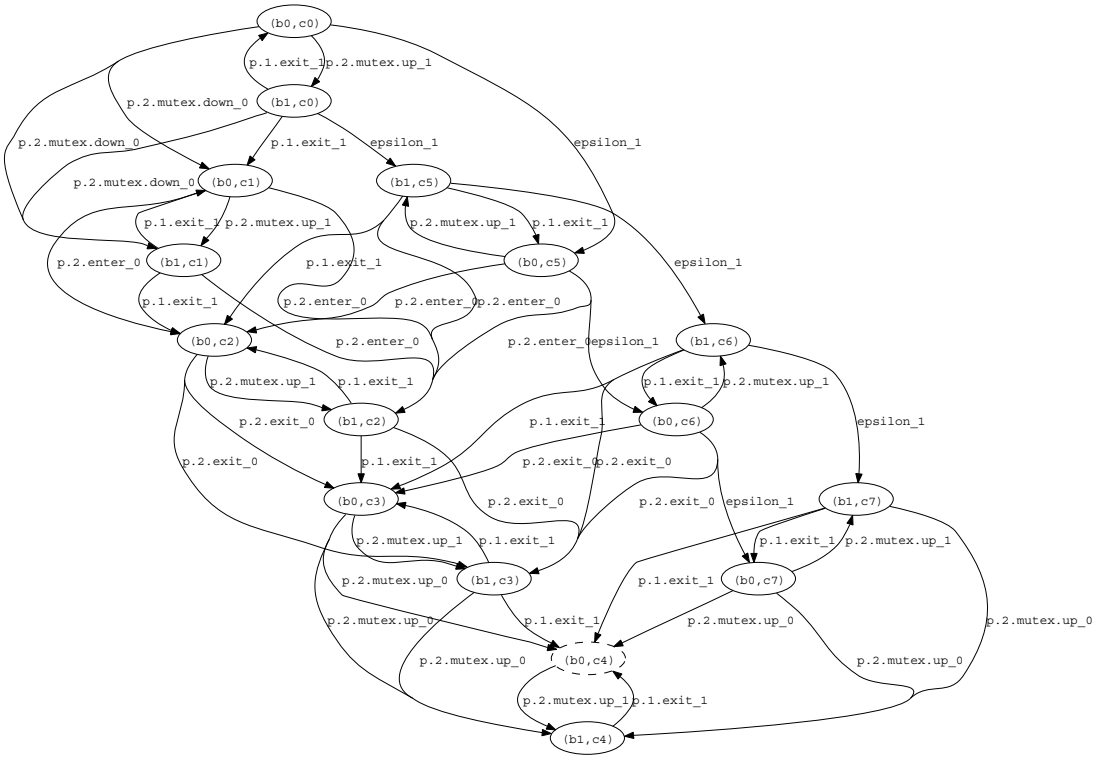


図 4.5. $TA'(\texttt{EXIT\_1}) \bowtie W_C$

を修正する正例は，以下のように求められる．

$$\tau_{\texttt{EXIT\_1}} = \quad [p.1.mutex.down,\ p.1.exit\ (\ p.2.mutex.down,\ p.2.enter,\ p.2.exit,\ p.2.mutex.up)^{\omega}]$$

LLL-F が生成する代表的な正例を表 4.1 に示す．これらはいずれも，反例の接頭辞中に現れる p.1 が危険領域に入る事象 *p.1.enter* について修正を施す正例である．例えば，表 4.1 の上の正例は，p.2 が危険領域に入る *p.2.enter* が反例に現れる前に，*p.1.exit* によって p.1 を危険領域から脱出させる軌跡である．

■

表 4.1. 正例の探索結果

反例	$[p.1.mutex.down,\ p.1.enter\ (p.2.mutex.down,\ p.2.enter,\ p.2.exit,\ p.2.mutex.up)^{\omega}]$
正例	$[p.1.mutex.down,\ p.1.enter,\ p.1.exit\ (p.2.mutex.down,\ p.2.enter,\ p.2.exit,\ p.2.mutex.up)^{\omega}]$
	$[p.1.mutex.down,\ p.1.exit\ (p.2.mutex.down,\ p.2.enter,\ p.2.exit,\ p.2.mutex.up)^{\omega}]$

4.1.5　誤りの同定

モデル L に含まれる誤り箇所を同定するために，反例 π と正例 τ の差分を求める．われわれが差分に注目するのは次の理由による．π と τ との間で一致しない事象が，π を τ に変換するために修正すべき事象であり，直接的あるいは間接的に性質違反の原因を指し示していると考えられる．一方，π と τ との間で一致する事象は，性質を満たすために修正する必要のない事象なので性質違反の原因ではない．

L が複数のプロセスで構成される並行システムのとき，π と τ で一致せず，修正すべき事象を遷移ラベルとして持つ遷移をプロセスごとに誤りの候補として特定する．しかしながら，L を構成するプロセスには，修正すべき事象に対応する遷移を持たないものが存在することがある．このようなプロセスに対しては，π と τ とで一致しない事象より前に，そして最後に実行される遷移を，性質違反の発生原因とみなして誤りの候補とする．以上のような遷移を発見するというアイデアは，求めた遷移が他のプロセスによる修正すべき事象の実行を引き起こすという想定に基づく．

本節では，反例を $\pi = [a_0, a_1, \ldots]$ とし，システム L が r 個のプロセスの並列合成で構成されるとする．各プロセスを $L^h{=}< S^h, A^h, \Delta^h, s_0^h >$ (ただし，$0 \leq h < r$，$A = \bigcup_{0 \leq h < r} A^h$) と書くこととする．

以下，反例に置換，削除操作が適用された場合と，挿入操作が適用された場合に分けて，システムを構成する各プロセスの誤りを発見する手法を説明する．

反例に置換操作，削除操作が行われた場合

反例中の事象 a_d に置換操作あるいは削除操作が適用されたとき，事象 a_d を不整合事象と呼ぶ．このとき，L^h に含まれる誤りの候補は，$a_d \in A^h$ のとき不整合事象 a_d に相当する遷移である．そうでない場合には，すなわち $a_d \notin A^h$ のとき，a_d の前に実行される L^h の最後の遷移を誤りの候補とする．形式的には，LLL-F は以下の条件を同時に満たす遷移 $(s, a_j, t) \in \Delta^h$ を求める：

- $0 \leq j \leq d$,
- $a_j \in A^h \wedge \forall l \in \mathcal{N}.(j < l \leq d \Rightarrow a_l \notin A^h)$.

$0 \leq j \leq d$ なる全ての j に対して $a_j \notin A^h$ ならば該当する遷移が存在しないので，遷移を返さない．

例 32. 反例 $\pi_{\texttt{EXIT\_1}}$ と $\tau_{\texttt{EXIT\_1}}$ を考える．$\pi_{\texttt{EXIT\_1}}$ の 2 番目の事象 *p.1.enter* が $\tau_{\texttt{EXIT\_1}}$ において *p.1.exit* に置換されているので，両者の差分を求めることにより，*p.1.enter* が不整合事象であることが分かる ($d = 1$)．LLL-F はこの不整合事象を用いて図 2.3 の p.1，p.2，Sema の各プロセスに対して誤りの候補をそれぞれ求める．プロセス p.1 に対して，不整合事象 *p.1.enter* は p.1 が危険領域に入ることを表す事象であり，この事象を持つ遷移を誤りの候補とする．し

たがって，LLL-F は不整合事象 *p.1.enter* を持つ遷移 $(1, p.1.enter, 2)$ を p.1 の誤りの候補とする．これは，p.2 が危険領域に入る前に p.1 が危険領域に入ってはいけないと解釈されるので，この性質違反の原因についての情報を表すとみなせる．そこで，LLL-F は誤り候補の遷移 $(1, p.1.enter, 2)$ と共に，その誤り候補を説明する正例 $\tau_{\texttt{EXIT\_1}}$ を開発者に提示する．

セマフォプロセス Sema に対する誤り候補を次に考える．しかしながら，不整合事象 *p.1.enter* は Sema の事象の集合に含まれない．そこで，$\pi_{\texttt{EXIT\_1}}$ において *p.1.enter* より前に，そして最後に実行される Sema の事象を求める．この条件を満たす事象は，事象 *p.1.mutex.down* である．したがって，セマフォプロセスが *p.1.enter* を引き起こす原因となる事象は *p.1.mutex.down* であると解釈されるので，LLL-F はこの事象を伴う遷移 $(0, p.1.mutex.down, 1)$ を誤りの候補として報告する．これは，*p.2.mutex.down* の前に *p.1.mutex.down* が生起されるため，p.1 と p.2 が同時に危険領域に入ることができるセマフォの排他制御機構の誤りを表す．$\tau_{\texttt{EXIT\_1}}$ は，p.1 が危険領域にいる間に p.2 が無限回危険領域に出入りすることが誤りの原因で，p.2 が危険領域に入るときに p.1 が危険領域にいてはいけないことを示唆する．以上により，セマフォプロセスの誤りがモデル上の誤りとして適当であることが分かる．

プロセス p.2 に対しては LLL-F は *p.1.enter* より前に実行される遷移が存在しないので，誤りの候補は存在しない．よって，LLL-F はプロセス p.1 と Sema に対してそれぞれ $(1, p.1.enter, 2)$ と $(0, p.1.mutex.down, 1)$ を誤り候補として求める．

以上の誤りの候補を手掛かりとして，開発者はモデル上の誤りを特定することが可能である．並行システムの例では，LLL-F によりセマフォプロセスの誤りである遷移 $(0, p.1.mutex.down, 1)$ が求められるので，誤りの特定が容易になる．

■

反例に挿入操作が行われた場合

反例中の事象 a_{d-1} と a_d との間に事象の挿入操作が適用された場合には，LLL-F は，L を構成する各プロセス L^h において，挿入された事象を「挟み込む」遷移の組を誤りの候補として開発者に提示する．この遷移の組は次の手順で求められる．

1. $0 < d$ ならば, 前項の手続きにおいて a_{d-1} を不整合事象とみなすことにより, $a_{d-1} \in A^h$ のとき，a_{d-1} を実行する遷移を返す．$a_{d-1} \notin A^h$ のとき，a_{d-1} の前に実行される L^h の最後の遷移を返す．形式的には，LLL-F は以下の条件を同時に満たす遷移 $(s, a_j, t) \in \Delta^h$ を求める：
   - $0 \leq j \leq d-1$,
   - $a_j \in A^h \wedge \forall l \in \mathcal{N}.(j < l \leq d-1 \Rightarrow a_l \notin A^h)$.

   $d = 0$ あるいは $0 \leq j \leq d-1$ なる全ての j に対して $a_j \notin A^h$ ならば該当する遷移が存在しないので，遷移を返さない．
2. $a_d \in A^h$ のとき a_d を実行する遷移を返す．そうでない場合，すなわち $a_d \notin A^h$ のと

き，a_d の後に実行される L^h の最初の遷移を返す．形式的には，LLL-F は以下の条件を同時に満たす遷移 $(s, a_j, t) \in \Delta^h$ を求める：

- $j \geq d$,
- $a_j \in A^h \wedge \forall l \in \mathcal{N}.(d \leq l < j \Rightarrow a_l \notin A^h)$.

$d \leq j$ なる全ての j に対して $a_j \notin A^h$ ならば該当する遷移が存在しないので，遷移を返さない．

この手続きのステップ 1 では，上で述べた置換操作や削除操作の場合と同様に，LLL-F は，各プロセスに対して，a_{d-1} を実行する遷移，あるいは，a_{d-1} の前に最後に実行され，そのプロセスの事象集合の要素であるような事象を持つ遷移を抽出する．ステップ 2 では，a_d，もしくは，それ以降に最初に現れる L^h に含まれる事象集合の要素の事象をラベルとして持つ遷移を返す．これらの遷移の組は挿入された事象を挟み込む事象と遷移を各プロセスに対して求めるので，開発者は，モデルのどこに正例に追加された遷移があるべきかについての情報を知ることができる．

例 33. 先の並行システムに対して活性 `EXIT_1` を検証した例について，表 4.1 に示した正例のうち，以下の軌跡を考える．

[*p.1.mutex.down*, *p.1.enter*, *p.1.exit* (*p.2.mutex.down*, *p.2.enter*, *p.2.exit*, *p.2.mutex.up*)$^\omega$]

この正例は，反例 $\pi_{\texttt{EXIT\_1}}$ の 2 番目の事象 *p.1.enter* と 3 番目の事象 *p.2.mutex.down* との間に事象 *p.1.exit* を挿入した軌跡である．

上の手続きのステップ 1 を適用すると，p.1, p.2，および Sema のそれぞれについて，事象 *p.1.enter* あるいは，その前に最後に実行される事象が得られる．このような事象は，p.1 と Sema についてはそれぞれ *p.1.enter* と *p.1.mutex.down* である．よって，それぞれについて p.1 と Sema の対応する遷移は，$(1, \mathit{p.1.enter}, 2)$, $(0, \mathit{p.1.mutex.down}, 1)$ となる．一方，事象 *p.1.enter* が実行される時点では p.2 のいかなる遷移も実行されていないので，p.2 には該当する遷移が存在しない．

続いて，ステップ 2 を各プロセスに適用する．プロセス p.1, p.2，および Sema それぞれについて，事象 *p.2.mutex.down* あるいは，その後に最初に実行される事象が抽出される．p.1 に関しては，$\pi_{\texttt{EXIT\_1}}$ において *p.2.mutex.down* 以降はどの遷移も実行されない．したがって，p.1 には該当する遷移は存在しない．一方，p.2 と Sema については，*p.2.mutex.down* が求める事象である．よって，それぞれの対応する遷移 $(0, \mathit{p.2.mutex.down}, 1)$ と $(1, \mathit{p.2.mutex.down}, 2)$ が開発者に提示される．

以上をまとめると，各プロセスについて，以下に示す誤り候補が求められる．

- p.1 の誤りの候補は $(1, \mathit{p.1.enter}, 2)$ である．これは，挿入された事象 *p.1.exit* は *p.1.enter* の後に現れなくてはいけないことを表すと解釈される．
- p.2 の誤りの候補は $(0, \mathit{p.2.mutex.down}, 1)$ である．これは，挿入された事象 *p.1.exit* は *p.2.enter* の前に現れなくてはいけないことを表すと解釈される．

- Sema の誤りの候補は $(0, p.1.mutex.down, 1)$ と $(1, p.2.mutex.down, 2)$ である．これらは，挿入された事象 *p.1.exit* は両者の間に実行されなくてはいけないことを表すと解釈される．

■

最後に，LLL-F が発見した全ての誤りの候補の遷移から，開発者はモデル L の誤った振る舞いを適切に指摘した遷移を決定する．LLL-F は誤り候補のみならず，各誤り候補に対応する正例も開発者に提供する．正例は反例がどのように性質違反を回避しているかを示しているので，開発者による決定作業を支援することができる．5.6 節で挙げる Chaki 等の方法 [19] に代表される先行研究と同様，正例を開発者に提示することは LLL-F の重要な特徴である．その上，各プロセスに対して誤り候補である遷移も提示することで，開発者は誤ったプロセスを容易に同定することができる．このような複数プロセスからなるモデルの誤り特定手法は，関連研究では扱われていないので，先行研究と LLL-F を区別する特徴と考えられる．

例 34. セマフォプロセス Sema の排他制御機構の誤りは，LLL-F が表 4.1 の 2 つの正例に対して列挙した遷移（例 32，例 33）によって指摘されている．そして，2 つの正例が Sema の誤った振る舞いを回避，訂正する方法を示している．

■

4.2　有限長の反例への適用

前節の誤り候補と対応する正例の探索手法は，投げ縄型の無限長反例を対象としている．しかし，性質が安全性の場合，2.4 節で述べた通り，多くのモデル検査器では反例は有限長の軌跡で与えられる．そこで，本節では，前節の手法を用いて，有限長の反例に対する正例を探索する手法を説明する．ただし，正例の集合は性質の TA が Büchi の意味で受理する軌跡の集合なので，正例は投げ縄型無限長軌跡で，有限長の事象列 P', C' に対して $\tau = P'C'^{\omega}$ となることに注意する．

2.4 節の議論から，有限長の反例は接頭辞のみから構成されると解釈できる．よって，有限長の事象列 P を用いて反例を $\pi = P$ と表せる．求める正例の接頭辞 P' は，P と編集距離が最小で，TA の受理状態に達する事象列とする．すなわち，P の接頭辞モデルを定義 15 を用いて構成し，4.1.2 節の方法で P' を求める．一方，正例の閉路 C' を 4.1.4 節の手法で求めるために，空列である反例の閉路に 1 回以上の挿入操作を行って得られる文字列の集合が C' の候補となるように閉路モデルを構成する．このようにして C' を求めるのは，C' が TA の受理状態を始点として同じ状態を終点とする事象列でなければならないからである．なお，閉路に対する 1 回の挿入操作は，C が空事象 ϵ からなると考え，それを他の事象に 1 回置換すると解釈してもよい．

A を対象モデルの事象の集合とするとき，構成する閉路モデルは WTS W'_C =<

$S'_C, A, \Delta'_C, c_0, \zeta'_C, M'_C >$ である．W'_C の構成要素を以下の通り定める．

- $S'_C = \{c_0, c_1\}$,
- $M'_C = \{c_1\}$,
- $\Delta'_C = \{(c_0, a, c_1) | a \in A\} \cup \{(c_1, a, c_1) | a \in A\}$,
- 全ての $\delta_C \in \Delta'_C$ に対して $\zeta'_C(\delta_C) = 1$.

W'_C の c_0 から c_1 への遷移が 1 回の挿入操作を，c_1 の自己閉路が任意回数の挿入操作を表す．よって，W'_C は，空列に 1 回以上の挿入操作を行って得られる事象列の集合を表す．W'_C に 4.1.4 節の手法を適用すれば，C' の候補が得られる．

例 35. 図 2.3 の CSys について，全ての時点で p.1 と p.2 が同時に危険領域にいることはない，という以下の安全性を検証する [74].

$$\texttt{MUTEX} = \mathbf{G}\neg(\texttt{CRITICAL.p.1} \wedge \texttt{CRITICAL.p.2})$$

ここで，p.1 と p.2 が危険領域にいることをそれぞれ流動 `CRITICAL.p.1` と `CRITICAL.p.2` で表す．2.3 節でこれらの流動を定義したが，再度以下に示す．

$$\texttt{CRITICAL.p.1} =< \{p.1.enter\}, \{p.1.exit\}, \mathbf{f} >$$
$$\texttt{CRITICAL.p.2} =< \{p.2.enter\}, \{p.2.exit\}, \mathbf{f} >$$

p.1 が危険領域にいる間に p.2 も危険領域に入ることが可能なため，図 2.3 のシステムは `MUTEX` を満たさない．反例は以下の有限長の軌跡で表される．

$$\pi_{\texttt{MUTEX}} = [p.1.mutex.down,\ p.1.enter,\ p.2.mutex.down,\ p.2.enter]$$

この反例の 2 番目の事象 *p.1.enter* によってプロセス p.1 が危険領域に入る．ところが，p.1 が事象 *p.1.exit* により危険領域から出て行かないうちに，p.2 が事象 *p.2.enter* によって危険領域に入ってしまう．つまり，$\pi_{\texttt{MUTEX}}$ の最後に生起される事象 *p.2.enter* が性質違反の原因を表している．よって，この誤りを指摘する正例を求めることが必要である．

図 4.6 に `MUTEX` の TA $TA(\texttt{MUTEX})$ を示す．図 4.6 の初期状態は b_0 である．状態 b_2 は吸収状態であり，この状態に達しない軌跡のみが `MUTEX` を満たす．

上の反例に本節の方法を適用することで，誤りを指摘する正例を求めることができる．まず，定義 15 よりこの反例 $\pi_{\texttt{MUTEX}}$ の接頭辞モデルは図 4.7 に示したようになる．ここで，図 4.7 の接頭辞モデルの初期状態は p_0 であり，末尾状態は p_4 である．

次に，図 4.6 の TA と図 4.7 の接頭辞モデルとの積に対して最短路探索を行うことにより，正例の接頭辞を求める．両者の積を図 4.8 に示すが，探索の始点となる初期状態は (b_0, p_0) である．また，求めるべき最短路の終点となる末尾状態は $(b_0, p_4), (b_1, p_4), (b_3, p_4)$ の 3 箇所である．この図では各終点の状態に対して複数の最短路が存在するが，その 1 つは，(b_0, p_0) から (b_0, p_4) への状態列 $[(b_0, p_0), (b_0, p_1), (b_3, p_2), (b_3, p_3), (b_0, p_4)]$ である．この経路を通過する事象を抽出することで，反例との編集距離 1 の事象列 [*p.1.mutex.down*, *p.1.enter*, *p.2.mutex.down*, *p.1.exit*] が正例の接頭辞として求められる．

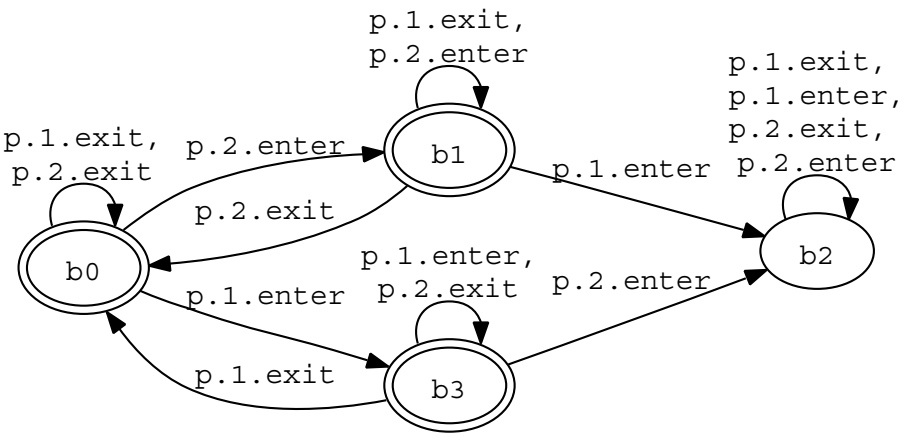


図 4.6. $TA(\texttt{MUTEX})$

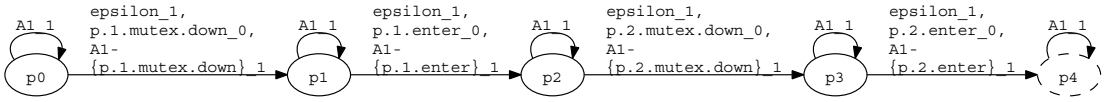


図 4.7. $\pi_{\texttt{MUTEX}}$ から構成した接頭辞モデル

一方，閉路モデルは，本節で述べた構成法に従うと図 4.9 で表されるモデルとなる．図で初期状態は c_0 であり，末尾状態は c_1 である．c_0 から c_1 への遷移の集合 (c_0, A_1, c_1) は空列である反例の閉路への事象の挿入操作を表す．よって，初期状態から末尾状態に達するまでに，少なくとも 1 回の挿入操作が実行されることが分かる．c_1 の自己閉路は，直前に挿入された事象の後に事象を追加する操作を表している．よって，4.1.2 節の方法によって得られる正例の閉路が空列とはならず，Büchi の受理条件を満たすことがこの例から分かる．

正例の接頭辞の探索の結果，到達した $TA(\texttt{MUTEX})$ の状態は b_0 である．そこで，4.1.4 節で論じたように，$TA(\texttt{MUTEX})$ と閉路モデルとの積を求める前に，受理状態が b_0 のみになるように $TA(\texttt{MUTEX})$ を変更する．ただし，この例では初期状態を変更する必要はない．この受理状態を変更した $TA(\texttt{MUTEX})$ を改めて $TA'(\texttt{MUTEX})$ とすれば，$TA'(\texttt{MUTEX})$ と図 4.9 の積を用いることで，正例の閉路が求められる．この積を図 4.10 に示す．このモデルの初期状態は (b_0, c_0) であり，また，末尾状態は (b_0, c_1) であるので，最短路は距離 1 の状態列 $[(b_0, c_0), (b_0, c_1)]$ である．したがって，得られる正例の閉路は，*p.1.exit* である．これは，空列である反例の閉路に，事象 *p.1.exit* を追加したものであると解釈することができる．

以上より，以下の正例が得られる．

$$[p.1.mutex.down,\ p.1.enter,\ p.2.mutex.down,\ p.1.exit\ (p.1.exit)^{\omega}]$$

この正例に適用された編集操作をまとめると，この正例は，反例の最後の事象 *p.2.enter* を *p.1.exit* に置換し，さらに，閉路をなす事象 *p.1.exit* を追加した軌跡である．

最後に，この正例と $\pi_{\texttt{MUTEX}}$ に 4.1.5 節の方法を適用して，モデルの誤り候補を求める．性

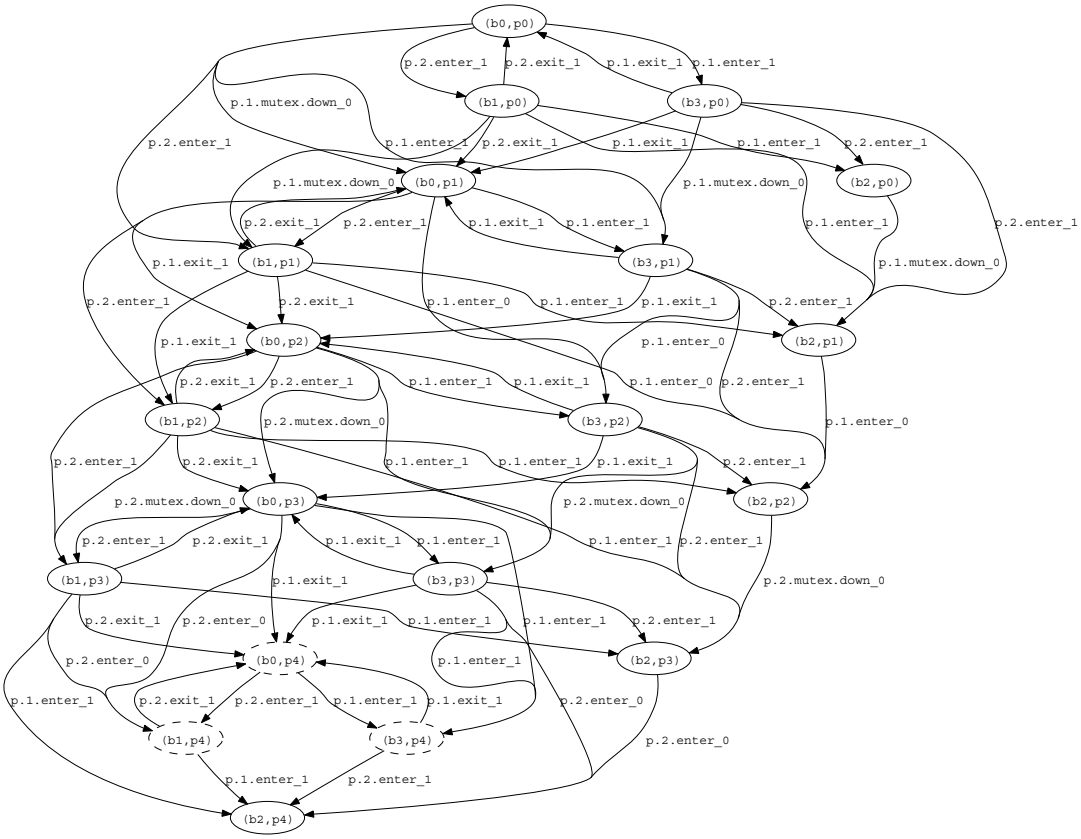


図 4.8. $TA(\texttt{MUTEX})$ と MUTEX の接頭辞モデルとの積

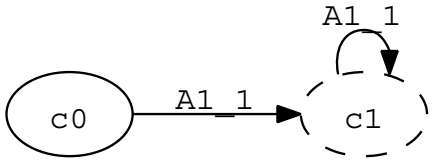


図 4.9. $\pi_{\texttt{MUTEX}}$ から構成した閉路モデル

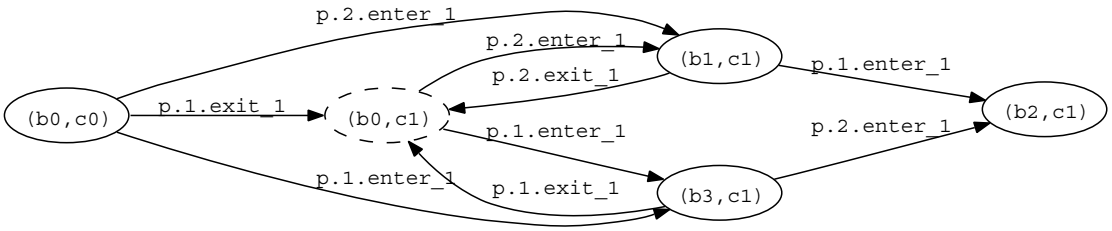


図 4.10. $TA'(\texttt{MUTEX})$ と MUTEX の閉路モデルとの積

質 MUTEX に関しては，反例の *p.2.enter* が *p.1.exit* に置換されているので，不整合事象は *p.2.enter* である．したがって，p.1 と Sema については *p.2.enter* の前に最後に実行される

反例中のそれぞれの事象 *p.1.enter* と *p.2.mutex.down* が，また，p.2 については不整合事象 *p.2.enter* が性質違反の要因の候補を表している．よって，p.1，p.2，Sema に対して，それぞれの遷移 $(1, p.1.enter, 2)$，$(1, p.2.enter, 2)$，$(1, p.2.mutex.down, 2)$ が対応するプロセスの誤りの候補となる．一方，正例は，p.1 が危険領域に入っている間に，p.2 が危険領域に入る事象 *p.2.enter* が生起してはいけないことを示唆する．これらの遷移と対応する反例から，*p.1.mutex.down* の後に *p.2.mutex.down* が生起されること，すなわち p.1 と p.2 がいずれも危険領域に入ることができるセマフォの排他制御機構の誤りを同定することができる．

なお，正例の閉路を構成する事象 *p.1.exit* は，反例中に現れる性質違反の原因となる事象を直接には指し示さないので，`MUTEX` に違反する誤りを指摘するためには必ずしも重要ではない．

■

4.3 実装

われわれは，4.1 節, 4.2 節で述べた LLL-F を自動化したプロトタイプツールを Java 言語で実装した．プロトタイプは，モデル，性質の TA，および反例を入力とし，モデルを構成する各プロセスに対する誤り候補である遷移と，対応する正例，適用された編集操作列を出力する．モデルおよび性質の TA は，Aldebaran 形式 [1] によって記述した拡張子".aut"を持つファイルとする．ただし，Aldebaran 形式には受理状態を表す構文が用意されていないので，性質の TA の記述において，受理状態は遷移ラベルの先頭の文字に"@"を付加した自己閉路を持つ状態で指し示す．これは，Giannakopoulou と Magee [45] が導入した記法である．

実行効率を高めるために，プロトタイプツールには次のような発見的な最適化処理を加えた．

- TA と WTS との積を求める演算と最短路探索を同時に実行する．LLL-F は，TA と WTS の積を最初に求め，その結果得られた WTS に対して最短路探索を行って正例の接頭辞を求めた．しかしながら，その手続きを直接実現すると，積の WTS と同じ状態空間を 2 回探索する必要がある．つまり，まず，積を求めるために 1 回探索を行い，続けて，その結果に対して最短路を求めるために 1 回探索を行わなくてはならない．しかし，実用性を考えると，モデルの状態数が大きい場合は 1 回の探索で実行できることが望ましい．そこで，TA と WTS の積を求めてからグラフ探索を実行するのではなく，初期状態から経路探索を実行しながら必要に応じて積モデルを構築していくことで，積を求める演算と最短路探索を同時に実行することができる．この場合，積の WTS がなす状態空間の探索回数が 2 回から 1 回に削減される．
- 同一の経路（状態列）を通って得られる正例が複数ある場合は，それらを全て出力するのではなく，代表例のみを出力する．これは，同じ編集操作を反例に適用した得られた代表的な正例だけを出力することで，冗長な結果が得られるのを防ぐためである．
- 複数プロセスからなるモデルにおいて，反例と正例の比較から同定された修正すべき事

象からは，各プロセス上で反例を実行して，そのプロセスの誤り候補となる遷移を求める．また，LLL-F は，性質の TA と，反例の接頭辞あるいは閉路から構成した WTS との積上で正例を探索するので，誤り候補を求める手続き以外では，対象となるシステムのモデルを利用しない．以上より，各プロセスを並列合成したシステム全体の振舞いを表すモデルを求める必要がないので，多くのプロセスからなる大規模なモデルに対しても LLL-F を適用できる．

4.4　事例研究

本節では，実装したプロトタイプツールを用いて実施した事例研究の結果を報告する．

4.4.1　有効性の評価

各事例研究は，以下の手順で実施した．

1. システムの振る舞いを表現する一つもしくは複数の並行動作するプロセスからなる LTS モデルを構成する．
2. モデルが満たさない性質を用意し，FLTL で記述する．
3. モデル検査器 LTSA [74] を用いて性質の検証を行い，反例と性質の TA を出力として得る．
4. システムのモデル，性質の TA，および反例を入力として実装したプロトタイプツールを実行する．
5. ツールの実行結果である正例と誤り候補のモデルの遷移を調査し，正例が誤りを適切に説明しているか，出力された遷移が適切に誤りを指摘しているか検討する．

実施した事例研究は以下の 7 つである．

- 電子レンジシステム（MOvn） [25]
- キャッシュコヒーレンスプロトコル（AFS-1） [104]
- 図 2.3 の並行システム（CSys）
- 鉱山用排水ポンプ制御システム（MPmp） [97]
- 分散データベース（DDb1, DDb2, DDb3） [74]

各事例に対するプロトタイプツールの実行条件を表 4.2 に示す．表 4.2 には，各モデルの規模，検証した性質の TA の規模，および LTSA が出力した反例の規模を記載してある．複数プロセスからなるモデルの規模は，各プロセスを並列合成した結果得られるシステム全体の振舞いを表すモデルの規模を示している．DDb3 モデルの規模は，ヒープメモリサイズの制限により計算することができなかったので，表の該当箇所を”-”で示す．DDb3 モデルの状態数と遷移数はそれぞれ 200 万以上および 6000 万以上である．性質が安全性の場合には反例が有限長となるので，表において反例の閉路の長さを”-”と記述している．

表 4.3 に，各事例研究を行った結果，LLL-F が生成した正例の数と実行時間をそれぞれ示す．各事例に対して，われわれはプロトタイプツールを 10 回実行し，その平均をツールの実行時間とする．ただし，実行時間は提案手法の実行に要する時間のみである．すなわち，入力を読み込む時間や，出力のための文字列を整形する時間，整形した文字列をファイルに出力するのに要する時間を含まない．実行時間は，2GB RAM を備えた 3.4GHz Pentium 4 (JDK 1.6.0) 上で計測した．

LLL-F は正例をシステムのモデル上ではなく，検証対象の性質の TA 上で探索する．表 4.3 に示した実行結果より，モデルの規模が非常に大きい DDb3 のようなシステムに対しても実用的な時間で実行できる手法であることが分かる．

以下，各事例に対して LLL-F を適用した結果を述べる．

MOvn

MOvn は，電子レンジシステムの振る舞いを表し，性質 `HEAT` は，全ての時点で，調理開始されるならいずれ加熱されることを表す．この事例について，LLL-F はモデルの誤りを適切

表 4.2. 事例研究を行ったシステム，性質の TA，および反例の規模

システム		TA の規模		反例の規模
名称	状態数/遷移数	性質名	状態数/遷移数	接頭辞長/閉路長
MOvn	7/21	`HEAT`	7/91	4/4
AFS-1	16/21	`VALID`	4/28	5/-
CSys	16/32	`MUTEX`	4/16	4/-
		`EXIT_2`	6/99	5/4
MPmp	22/56	`EMG`	2/30	3/4
DDb1	160/402	`QUIS`	10/897	12/1
DDb2	6460/18537	`QUIS`	10/890	26/33
DDb3	-	`SAFE`	452/33900	18/-

表 4.3. 各事例に対して LLL-F が生成した正例の数と実行時間

システム名	性質名	正例数	誤りを指摘した正例数	実行時間 [sec]
MOvn	`HEAT`	60	4	0.23
AFS-1	`VALID`	8	2	0.04
CSys	`MUTEX`	7	7	0.03
	`EXIT_2`	6	3	0.19
MPmp	`EMG`	9	1	0.07
DDb1	`QUIS`	20	1	0.18
DDb2	`QUIS`	1	0	0.72
DDb3	`SAFE`	82	11	30.30

に説明する正例を生成することができた．

HEAT は，電子レンジが調理を開始したならば，いずれレンジ内が加熱されるということが常に成立することを主張する応答性 [39] である．応答性は，2.1.2 節で述べたように，活性に分類される性質のパターンの 1 つである．上で述べた反例は，電子レンジの扉が開いているときに，電子レンジが調理を開始したにもかかわらずレンジ内が過熱されることのない軌跡を表していた．LLL-F が出力した正例は，以下の通り分類された．

- 扉が開いている間に電子レンジが調理開始した後に，レンジ内が加熱されるという系列が繰返される．
- 扉が開いている間は電子レンジが調理開始することがないという系列が繰返される．

誤りを指摘するのに適当な正例は，LLL-F が出力した 2 種類の正例のうち，後者の場合であった．

AFS-1

AFS-1 はクライアントとサーバで保持するデータの一貫性を保証するプロトコルで，性質 VALID は，常に，サーバのデータが有効ならばクライアントのデータも有効であることを主張する安全性である．AFS-1 に対して LLL-F を適用した結果得られたモデルの誤り候補には，性質違反の要因を正しく指摘したものが含まれていた．モデル検査器から与えられた反例が示すのは，サーバのデータが有効であるが，クライアントのデータが無効であるという現象だった．LLL-F によって得られた正例は，VALID を成り立たせるための反例の修正点を 2 つの可能性を指摘していた．

- クライアントのデータを有効とする．
- サーバのデータを無効にする．

以上は，性質を満たさない要因として，以上の 2 種類が考えられることを示しており，これを元にサーバが持つ性質違反の原因を発見した．

CSys (MUTEX)

図 2.3 の CSys について成り立つべき性質 MUTEX は，4.2 節で述べた安全性である．LLL-F によって得られたセマフォプロセスについての誤り箇所候補や正例は，前節で論じたようにセマフォ機構の誤りを指摘していた．すなわち，図 2.3 の性質 MUTEX で得られた誤り箇所候補は，セマフォプロセス Sema の遷移 $(1, p.2.mutex.down, 2)$ と $(0, p.1.mutex.down, 1)$ だった．これらは，いずれも前節で示したセマフォ機構の誤りを表す．

LLL-F が生成した正例は，次のように分類することができた．

- p.1 が危険領域に入っている間に，p.2 が危険領域に入らない．
- p.2 が危険領域に入る前に，p.1 が危険領域に入らない．

これらの正例はモデル上の異なる箇所を誤りとして指摘していたが，いずれも p.1 と p.2 が同

時に危険領域に入ることを禁止する正例であった．

CSys (EXIT_2)

性質 EXIT_2 は，危険領域に入った p.1 と p.2 は共にいずれ危険領域を脱するという活性で，EXIT_1 の拡張である．反例は，EXIT_1 の場合と同様に，p.2 が危険領域にいる間に p.1 が無限回危険領域に出入りすることを表していた．LLL-F を適用した結果，得られた正例は，いずれも p.1 が無限回危険領域に入る前に p.2 が危険領域を出ることを要請するものだった．また，p.2 が危険領域に入る遷移 $(1, p.2.enter, 2)$ と，それを可能にするセマフォプロセス Sema の誤り $(1, p.2.mutex.down, 2)$ が誤り候補として報告された．それらの情報から，Sema の排他制御機構の誤りが指摘されていることを確認した．

MPmp

MPmp は鉱山内に充填する汚水を排水するポンプ制御システムのモデルである．検証した性質 EMG は，鉱山内にメタンガスが発生した場合には，緊急時の対応として機器の警告灯を点灯させるという応答性である．反例は，鉱山内にメタンガスが発生しているにも関わらず，警告灯が点灯しないという軌跡であった．LLL-S の結果得られる正例は，以下の通りに分類されるものだった．

- メタンガスが発生した後に警告灯が点灯するように反例を修正した軌跡.
- メタンガスが発生しないように反例を修正した軌跡.

後者の正例は緊急事態が発生しないことを表しているので，モデルが持つ誤りを指摘したものではなかった．よって，前者の正例から緊急時に警告灯を点灯しないモデルの誤りを特定した．

DDb1

DDb1 は 3 つのデータベースノードを持ち，これらのノードが環状のネットワークを形成する分散データベースシステムのモデルである．そして，1 つのノードのデータが更新されると，他のノードにもその変更が順に伝播されて，残りのノードのデータも更新される．ただし，DDb1 は，データの更新が行われるのは 1 回限りであり，コントローラプロセスによって制御されている．検証した活性 QUIS は，各ノードが，いつの時点からもいずれ全て不活性（データの更新が行われない状態）になるという性質である．したがって，この性質によって，各ノードのデータの更新が正しく完了し，今後更新が発生しない状態になるかどうかを検証することができる．反例は，3 つのノードの内 2 つのみが不活性になるが，残り 1 つが不活性にならないことを表していた．

LLL-F が生成した正例は，全てのノードが不活性になるように反例を修正した軌跡だった．ただし，その中には，反例中で不活性になることのないノードが，データの更新が行われる前に不活性になることを表す軌跡も 1 件含まれていた．このような正例は性質の論理式は満たすが，更新が 1 回行われるという対象領域の条件を満たしていないので，性質違反の原因を適

切に指摘していない正例だった．一方，正例の中には，コントローラが全てのノードが不活性になることを確認せずに終了する，という性質違反の適切な原因も含まれていた．この原因によって起こるコントローラの誤った振る舞いを回避する正例から，われわれはコントローラの誤りを発見した．

DDb2

これは，DDb1 と同様の構成のネットワークであるが，データの更新は一度限りではなく，任意回数実行できるモデルである．検証した性質 `QUIS` は DDb1 と同じ性質を記述した論理式である．この反例も，DDb1 の事例と同様に 3 つのノードの内 2 つのみが不活性になることを表していた．

しかしながら，LLL-F はモデルの誤りを指摘する正例を生成することができなかった．これは，LLL-F の方法論上の制約に起因する．LLL-F が求める正例の接頭辞は，性質の TA の初期状態から受理状態に到達する事象列ではなくてはならず，また，その閉路は，その受理状態を繰返し通過する事象列でなければならない．したがって，接頭辞が受理状態で終わらない正例を発見することができない．これは，LLL-F が求める正例の接頭辞が TA の受理状態である，という正例の形状についての仮定，ならびに LLL-F が想定する正例と反例の距離に起因する問題点である．5 章において，この問題点について詳しく述べ，LLL-F を精緻化することでそれを克服する方法を提案する．

DDb3

この事例も，DDb2 と同様の分散ネットワークである．性質 `SAFE` はデータの一貫性を保証する安全性であり，全てのノードが不活性であるならば，同一の値をデータとして各ノードが保持することを表す．反例は，全てのノードが不活性となった際に，保持する値が異なるノードが 1 つ存在することを表していた．LLL-F によって得られた正例が表す意味は以下の 2 通りのいずれかだった．

- 不活性とならないノードが存在する．
- 全てのノードが不活性となったときに，全ノードの値が等しくなる．

LLL-F の出力から，値が異なるデータを保持するノードが不活性とならないことを表す正例を選択した．この事例は出力された正例の数が他の事例より多いので，手作業で適当な正例を選択する必要があった．そこで，正例間の順位付けによって開発者が誤りを効率的に特定する仕組みが望まれる．

4.4.2　実行性能の評価

続いて，われわれは，性質の TA の規模，反例の接頭辞，および閉路の長さをそれぞれ変化させた場合に，プロトタイプツールの実行時間がどのように変化するかを調査した．調査に用いた事例は MPmp で，性質として `EMG` を用いた．`EMG` の TA の規模のみを変化させた場合の

プロトタイプツールの実行時間の変化を表 4.4 に示す．同様に，反例の接頭辞長のみを変化させた場合のプロトタイプツールの実行時間と，反例の閉路長のみを変化させた場合のプロトタイプツールの実行時間を図 4.11 にそれぞれ示す．表 4.4 において，規模の異なる EMG の TA を作成した方法は次の通りである．

1. MPmp が満たす安全性を記述した論理式を複数用意して，それらのうちのいくつかと EMG との論理積を作成した．
2. 作成した論理積を LTSA で TA に変換した．

一方，反例の接頭辞長と閉路長に関しては，表 4.2 の反例を実験に用いた．ただし，反例の接頭辞長あるいは閉路長を変化させるために，閉路を構成する事象列を，接頭辞あるいは閉路内に展開して反例を再構成した．実行時間の計測法と計測環境は表 4.3 の場合と全く同様である．

表 4.4 の結果より，LLL-F は数十の状態と 1000 以上の遷移を持つ TA に関して実用的な時間で実行可能であることが分かる．この結果は，表 4.3 の結果が示す傾向と一致する．図 4.11 によると，反例の接頭辞長が実行時間に与える影響は，閉路を構成する事象列の長さが与える影響とほぼ同程度であることが分かる．加えて，この結果より，LLL-F は，は接頭辞と閉路が長い場合にも，実用的な性能が得られていると考えられる．

表 4.4. MPmp システムにおける性質 EMG を用いたときの Büchi オートマトンの規模に対する LLL-F の実行時間

TA の規模		実行時間
状態数	遷移数	[sec]
4	58	0.08
8	176	0.53
11	216	0.24
14	302	0.34
29	534	0.28
29	650	3.33
35	751	1.18
35	855	3.99
52	931	0.46
50	1190	1.25
64	1327	0.64
64	1471	0.66

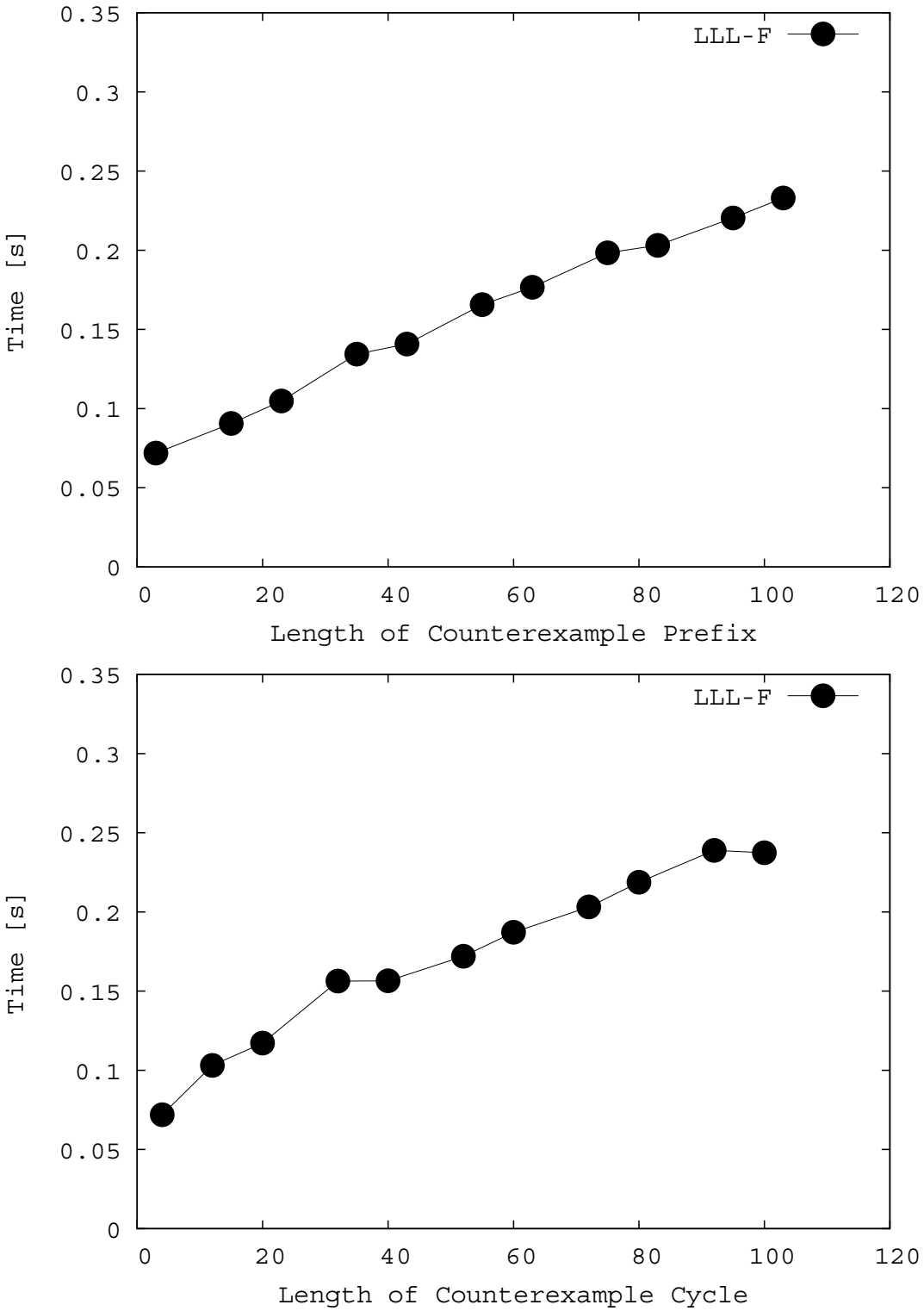


図 4.11. 反例の接頭辞長に対する LLL-F の実行時間（上）と閉路長に対する LLL-F の実行時間（下）

4.5 考察と課題

計算時間

LLL-F において，正例を求めるための時間が計算時間の多くを占めると考えられる．ここでは，LLL-F が正例を求めるのに必要な計算時間を見積もる．反例を $\pi = PC^\omega$ とし，性質 ϕ の TA を $TA(\phi) = (S_\phi, A_\phi, \Delta_\phi, u_0, S^a_\phi)$ とする．ここで，π について $|P| = m$，$|C| = n$ とし，$TA(\phi)$ について $|S_\phi| = v_\phi$，$|S^a_\phi| = v^a_\phi$ とする．

正例の探索時間は，ϕ の TA と接頭辞モデル W_P，閉路モデル W_C との積の規模，ならびにその積モデル上の探索に要する時間に支配される．接頭辞モデルの構成法より，W_P は m 個の状態を持つので，積 $TA(\phi) \bowtie W_P$ の状態数は $v_\phi m$ である．よって，接頭辞探索に必要な時間は，初期状態からの最短路探索に必要な時間 $O(v_\phi m \log(v_\phi m))$ となる．

次に，$TA(\phi)$ の各受理状態に対する 1 回の閉路探索に必要な計算時間を見積もる．定義より W_C の状態数は $2n$ であるので，$TA'(\phi) \bowtie W_C$ の状態数は $2v_\phi n$ である．ゆえに，閉路を求めるための 1 回の最短路探索には $O(v_\phi n \log(v_\phi n))$ 時間必要である．正例の閉路探索は，$TA(\phi)$ の受理状態の数と同一の v^a_ϕ 回繰返されるので，全ての閉路探索に必要な時間は $O(v^a_\phi v_\phi n \log(v_\phi n))$ である．

最後に，$m \approx n$ と仮定すれば，正例を探索するために必要な時間は，全ての閉路を探索する時間に支配される．以上より，正例を求めるための時間は $O(v^a_\phi v_\phi n \log(v_\phi n))$ であり，この時間が誤り特定アルゴリズムの実行に支配的である．

5 章で述べる手法 LLL-S は，後で議論するように，$O(v^a_\phi v_\phi n^2 \log(v_\phi n))$ の計算時間が正例の探索に必要である．ゆえに，LLL-S は LLL-F の精緻化を図った方法であるが，LLL-F は軌跡間の距離と正例の形状について発見的な手法を導入することで，LLL-S に対して正例探索に必要な計算時間の短縮が実現されている．

LLL-F の適用可能性について

本章で提案した誤り特定法 LLL-F は反例が示す誤りを求める．また，性質は，充足不能な LTL 式でない限り，その性質と等価な TA を構築することが可能である．加えて，本手法は，グラフ探索手法に基づき正例を探索することで，Büchi の受理条件を満たす正例を求める．よって，性質の論理式の形式を問わず実行可能な方法であるので，安全性，活性を含む全ての FLTL 式に適用可能な手法である．したがって，汎用性の高い方法であるといえよう．

公平性について

本章では，活性については，2.4 節で説明した公平な選択を仮定しない場合のモデル検査のみを対象として，誤りを特定する方法を議論してきた．しかし，提案した方法 LLL-F は，反例が表す誤りの要因を発見する手法である．したがって，公平な選択を仮定した場合においても，LLL-F によって，与えられた反例を修正する正例を求め，誤りを特定する手続きを実行することができる．以上の議論より，公平な選択を仮定している場合も，全く同様に LLL-F を

適用することが可能である．

反例と求める正例の形状について

LLL-F は，3 章のモデル修正法と同様に，活性の検証に対しては反例が無限長の投げ縄型軌跡，安全性の場合には有限長の軌跡であることを前提とする．この前提を満たさないような形状をした反例に対して LLL-F が適用できるか議論する．

- LLL-F は，投げ縄型反例に対して，性質を満たす投げ縄型正例を求める．本論文で正例の形状を投げ縄型とする利点は，以下の通り挙げられる．
  - 反例と正例の違いやモデル上の誤り候補を，反例に対する編集操作を基に効率的に求めることができる．
  - 反例と同様の形状をなす正例を開発者に示すことにより，開発者は，反例が示す誤りの要因を直観的に理解することができる．

  特に，後者に関連して，投げ縄型でない形状の反例に対しては，反例の形状に応じて，探索すべき正例の形状を適切に定める必要がある．
- LLL-F は，投げ縄型反例の接頭辞と閉路を分割して，編集操作によって拡充したモデルをそれぞれについて構成する．ゆえに，探索すべき正例の形状に応じた WTS モデルを構築しなければならない．そのために，入力として与えられた反例の形状を調べて，適切なモデル構築手続きを適用する必要がある．

LLL-F の制約について

本章の提案手法 LLL-F は，反例と類似した正例を発見的に求める手法である．LLL-F は，投げ縄型無限長軌跡の接頭辞と閉路については，有限長事象列間の編集距離を距離の尺度としている．しかしながら，投げ縄型無限長軌跡間に対して統一的に適用できる距離の尺度を定義していない．また，本手法が発見する正例は，接頭辞が性質の TA の受理状態に到達する事象列であり，閉路はその受理状態を無限回通過する閉路であるという特徴がある．すなわち，LLL-F は，正例と反例の距離と，正例の形状に関する発見的手法に基づいて，反例と類似した正例を求める方法である．したがって，本手法は，接頭辞が受理状態に達しない正例を求めることができないという制約がある．それゆえ，精確な誤り特定を行うために必要な正例を発見できない場合がある．われわれが行った事例研究においても，1 件の事例について性質違反の原因を正しく指摘した正例が求められなかった．5 章において，以上の問題点について詳しく議論を行い，この制約を解決し，投げ縄型無限長軌跡間の距離を導入することで方法論の精緻化を図った手法 LLL-S を提案する．

今後の課題

本章で提案した方法 LLL-F では，正例と反例とを比較することで，正例に変換するために編集操作を適用した反例の事象を求めて，モデル中の誤りを特定する．しかし，われわれが扱う命題は性質に現れる流動なので，反例や正例によって各命題の真理値がいかに変化するかを

情報として開発者に提示することが望ましい．つまり，単に正例と反例の事象の違いを分析するだけでなく，流動の真理値の違いや性質の論理式の真理値も考慮した差分を求めることで，誤り特定の精度を高めることができると考えられる．モデル検査器 LTSA [74] には反例中の流動の値の変化を表示する機能が備えられているので，その機能を正例に対しても実装することで実現することが可能であると考えられる．

反例の解釈についてのさらに詳細な情報を提供するためには，正例との差分に加えて，反例が性質をなぜ満たさないのか，あるいは正例が性質をなぜ満たすのかについて証明をそれぞれ構築する手法を適用することが考えられる．LTL のモデル検査において，モデルが性質を満たす理由の説明を生成する研究は，Peled と Zuck により実施されている [86]．これは，LTL 式の否定の一般化 Büchi オートマトンと公理を用いて証明を導出する手法である．一般化 Büchi オートマトンは，受理状態の集合が存在しない代わりに，受理集合という状態の冪集合を導入した Büchi オートマトンの一種である．一般化 Büchi オートマトンは，受理集合の全ての要素を訪問する軌跡を受理する．つまり，受理集合の全ての要素（状態の集合）について，各集合に含まれる少なくとも 1 つの状態を通過する軌跡を受理する．LTL 式の一般化 Büchi オートマトンは，LTL 式を Büchi オートマトンに変換するための中間生成物として作られることが多い [43, 44, 25, 5]．この場合，オートマトンの各状態には，検証対象の LTL 式の部分式の集合がラベル付けられている．この部分式の集合は，直観的には，その状態において成り立つ論理式の集合である．Peled と Zuck の方法は，モデルと一般化 Büchi オートマトンの各状態にラベル付けした論理式の集合を調べながら，公理を用いて演繹的に証明を導出するので，自動化が難しいという問題点がある．しかしながら，正例と反例に限れば，両者をオートマトン上で追跡して，通過する状態に付けられたラベルの集合を分析すればよいので，手続きの単純化ができると思われる．この方法により作成した証明は，正例が性質を満たし，また，反例が性質を満たさない根拠を提示できるので，開発者が両者を理解するための支援をすることが期待される．

LLL-F の現在のプロトタイプは，性質の TA を既存のモデル検査器を用いてユーザが作成して入力しなければならない．ここで想定しているモデル検査器は LTSA である [74]．LLL-F のプロトタイプを実用的なソフトウェアツールとするためには，性質の論理式を入力として TA を自動的に生成する機能を今後実現することが望ましい．そのためのアルゴリズムは既に提案されているだけでなく [44, 45]，LTSA において実現されている．そして，この機能が実現されることで，さらに，以下の機能もまた実現できる．

- モデル検査機能とモデル上の誤り特定機能の統合．従来のモデル検査器は反例を出力するが，それに加えて，LLL-F を用いて正例と誤りの候補を出力することが可能となる．これにより，モデルが性質を満たさない場合に，従来のモデル検査器よりも有益な性質違反に関する情報を開発者に提供することができる．
- 誤り特定法と先に論じた証明作成機能との統合．Giannakopoulou 等による Büchi オートマトン構成法 [44] は，一般化 Büchi オートマトンを最初に作成し，それを Büchi オートマトンに変換する方法である．よって，正例と反例についての証明の構築のある

程度の自動化ができると考える．

最後に，より多くの事例，特に，現実の開発事例へ本手法を適用し，有効性を評価する必要がある．中でも，LLL-F が想定している正例の形状や，反例と正例との編集距離に関する発見的手法がどの程度誤り特定作業に影響を与えるかを調査しなければならない．

4.6　まとめ

既存のモデル検査器は，モデルが性質を満たさない場合には反例を開発者に提示する．本論文では，反例からモデルの誤り箇所を発見し，反例に対する修正候補を提示する方法 LLL-F を提案した．LLL-F は，反例の接頭辞と閉路についてそれぞれ編集距離が最小となる正例を，有向グラフ上の最短路探索法を用いて求める．

LLL-F は，特別な技術を前提とせず，これまで確立されたグラフ探索アルゴリズムを用いているので自動化が容易な方法である．そして，FLTL 式で記述可能な全ての性質に適用できる．また，本手法は，所望の性質を満たさないモデルを，性質を満たすように改編するモデル修正法 [66] の支援に用いることが期待できる．モデル修正法については 3 章で議論した．

第 5 章

軌跡間の距離を用いた誤り特定法の改善

4 章では，反例と類似した正例を求めることで，モデルの誤りを特定する手法 LLL-F を提案した．LLL-F は求める正例の形状と，反例と正例の距離について発見的な仮定を導入して，正例を求める手法である．LLL-F は実用上効率的な手法であるが，反例と正例との間の類似度の基準である無限長軌跡間の距離を明確に定義していない．したがって，4.3 節の事例研究で述べたように，LLL-F を用いるとモデルの誤りを適切に指摘した正例を生成できない場合がある．

本章では，これらの点を解決し，LLL-F を精緻化した正例の生成手法, および，これを用いたモデルの誤り特定手法である LLL-S（Lightweight error Localization for Labeled transition systems - Second version）を提案する．5.1 節で，上記の課題を明らかにする．そして，先行研究で提案された正例の探索手法を用いても指摘した問題点を解決できないことを示す．したがって，問題点を解決するためには，誤り特定手法の精緻化および改善が必要である．

続いて，5.2 節において，問題点を解決する誤り特定手法 LLL-S を説明する．この節の最初に，無限長の軌跡である反例と正例の類似度を表す距離を，有限長の文字列間の距離である編集距離を用いて定義する．本章で定義する距離は，先行研究が採用した距離と異なり，無限長の軌跡の接頭辞と閉路を区別して取り扱うことに特徴がある．LLL-S は，この距離が最小となる正例を，LLL-F と同様に有向グラフ上の最短路探索問題を解くことによって求める．そのため，性質の TA と反例から構成したモデルの積を，探索を実行する有向グラフとする．反例から構成したモデルは，反例の各事象を表現する遷移を持つ．続いて，反例の各事象に対する置換，削除，挿入操作を表す遷移を拡充することによってモデルを構成する．ただし，LLL-F とは異なり，反例の接頭辞と閉路に対して個別にモデルを作るのではなく，両者を編集した事象列を 1 つのモデルで表現する．このモデルと性質の TA との積上で最短路探索を 2 段階実行することで，正例を求めることができる．

われわれは，Java 言語により LLL-S を自動化したプロトタイプツールを実装した．このプロトタイプツールには，実行性能を高めるために LLL-S にいくつかの最適化処理を組込んだ．そこで，5.3 節で実装した最適化処理を紹介する．

続いて，このプロトタイプツールを用いて事例研究を行った．この事例研究の結果を 5.4 節で報告する．事例研究で評価を行った項目は，4 章と同様に次の 2 種類である．

- 7 種類のシステムについてモデルを構成し，LLL-S を実行する．そして，その結果が適切にモデルの誤りを指摘しているかを調べる．
- LLL-S の実行性能に影響を与える要因である反例，および性質の TA の規模を変化させ，それぞれの場合について LLL-S の実行時間の変化を調べる．

次に，5.5 節で，LLL-S と関連する事項について議論する．この節の議論には，正例探索に必要な計算量，本章で提案した距離関数や公平性を仮定したモデル検査への適用についての議論に加えて，LLL-S に関する今後の課題も含まれている．特に，正例探索に必要な時間は，精緻化を行ったために LLL-F よりも大きくなるが，実用的な性能であることを示す．

5.6 節において，本論文で提案した誤り特定法 LLL-F と LLL-S の関連研究を議論する．モデル検査の結果に基づきモデル上の誤りを特定する手法に注目した先行研究はあまり見当たらない．この節では，主にプログラムやハードウェアの誤りに注目した方法を論じる．

最後に，5.7 節で本章のまとめを行う．

5.1　既存手法の問題点

4 章では，正例を用いてモデルの誤りを特定する手法 LLL-F を提案した．本節ではその問題点を並行システム CSys を例に挙げて説明する．また，指摘する問題点は，先行研究で採用されたアイデアを用いても解決できないことを同じ例で示す．

LLL-F への入力は，対象のモデル，性質の TA，性質をモデル検査した結果得られる反例 PC^ω である．ただし，P, C を有限長の事象列とすると，P は接頭辞，C は閉路である．求める正例は，反例と同様に $P'C'^\omega$ の形状をなすと仮定する．ここで，P', C' は有限長の事象列である．

1. 次の 2 条件を満たす正例の接頭辞 P' を求める．
   - 反例の接頭辞 P と編集距離が最小である．
   - 性質の TA の初期状態から受理状態に到達する事象列である．
2. 次の 2 条件を満たす正例の閉路 C' を求める．
   - 反例の閉路 C と編集距離が最小である．
   - 性質の TA において，上で到達した受理状態を始点として，その受理状態に到達する事象列である．
3. 反例と正例の差分を求めて誤りを特定する．

この方法は，有限長文字列間の編集距離を用いて反例と類似した正例を求めるが，無限長軌跡を扱うための発見的手法を導入している．しかしながら，反例と正例，あるいは正例間の類似性の基準を定めた無限長軌跡間の距離が，明確に定義されないという問題点がある．それゆえ，反例に対して，適切に誤りの修正が実施されたとみなせる正例を生成できない場合があ

る．先行研究において，反例と類似した正例を求めるために，軌跡間の距離が編集距離に基づいて定義されている [19, 48]．しかし，これらの研究が採用する距離は，無限長軌跡が持つ接頭辞と閉路を区別しない指標である．そのため，先行研究のアイデアを用いたとしても，誤りを適切に指し示す正例を発見できない場合がある．

例 36. 本節における論点を明らかにし，LLL-F と先行研究の限界を説明するため，2 章で挙げた並行システム CSys（図 2.3）と性質 $\texttt{EXIT\_1} = \mathbf{G}(p.1.enter \Rightarrow \mathbf{F}p.1.exit)$ の検査の例を再び考える．表 5.1 に，EXIT_1 の反例 $\pi_{\texttt{EXIT\_1}}$，ならびに，EXIT_1 を満たす正例 $\tau^1_{\texttt{EXIT\_1}}$，$\tau^2_{\texttt{EXIT\_1}}$，$\tau^3_{\texttt{EXIT\_1}}$，$\tau^4_{\texttt{EXIT\_1}}$ を示す．

われわれが扱う反例 $\pi = PC^\omega$ と正例 $\tau = P'C'^\omega$ との間の類似度の最も簡単かつ直感的な指標として，事象列 PC を $P'C'$ に変換するのに必要な編集距離が考えられる．ただし，この編集距離は C と C' の繰り返しを無視する．この距離の基本的なアイデアは，Chaki 等の手法 [19] や Groce 等の手法 [48] で用いられているプログラム実行列間の距離と同様である．本節では，この距離を $d_e(\pi, \tau)$ と表すことにする（各編集操作のコストを 1 とする）．

$\tau^1_{\texttt{EXIT\_1}}$ は，事象 *p.1.exit* を $\pi_{\texttt{EXIT\_1}}$ の接頭辞の 2 番目の事象 *p.1.enter* の直後に挿入することで得られる正例である．$\pi_{\texttt{EXIT\_1}}$ を $\tau^1_{\texttt{EXIT\_1}}$ に変換するのに必要な編集操作は 1 回の挿入操作のみであるので，$d_e(\pi_{\texttt{EXIT\_1}}, \tau^1_{\texttt{EXIT\_1}}) = 1$ である．反例を性質を満たすように修正するためには少なくとも 1 回の編集操作が必要であるので，d_e によって $\tau^1_{\texttt{EXIT\_1}}$ は $\pi_{\texttt{EXIT\_1}}$ と最も近い正例であると判断できる．$\tau^1_{\texttt{EXIT\_1}}$ によって，開発者は，プロセス p.1 は p.2 が危険領域に入る前に危険領域を出なければならないということが理解できる．したがって，$\tau^1_{\texttt{EXIT\_1}}$ は，Sema プロセスは p.1 と p.2 による危険領域への排他的なアクセスを適切に制御できていないことを示しており，編集操作が行われなかった $\pi_{\texttt{EXIT\_1}}$ 内に現れる他の事象は，システムが EXIT_1 を満たすには変更が不要であることを示している．前章で示した方法に従って，$\pi_{\texttt{EXIT\_1}}$ と $\tau^1_{\texttt{EXIT\_1}}$ とを比較することで，Sema プロセスの誤った遷移の候補として，$(0,\ p.1.mutex.down, 1)$ と $(1,\ p.2.mutex.down, 2)$ が抽出される．これらの遷移は，Sema において事象 *p.1.exit* と最

表 5.1. EXIT_1 に対する反例（$\pi_{\texttt{EXIT\_1}}$）と正例（$\tau^1_{\texttt{EXIT\_1}}$，$\tau^2_{\texttt{EXIT\_1}}$，$\tau^3_{\texttt{EXIT\_1}}$，$\tau^4_{\texttt{EXIT\_1}}$）

反例	$\pi_{\texttt{EXIT\_1}} =$	[*p.1.mutex.down*, *p.1.enter* (*p.2.mutex.down*, *p.2.enter*, *p.2.exit*, *p.2.mutex.up*)$^\omega$]
正例	$\tau^1_{\texttt{EXIT\_1}} =$	[*p.1.mutex.down*, *p.1.enter*, *p.1.exit* (*p.2.mutex.down*, *p.2.enter*, *p.2.exit*, *p.2.mutex.up*)$^\omega$]
	$\tau^2_{\texttt{EXIT\_1}} =$	[*p.1.mutex.down*, *p.1.enter* (*p.2.mutex.down*, *p.1.exit*, *p.2.exit*, *p.2.mutex.up*)$^\omega$]
	$\tau^3_{\texttt{EXIT\_1}} =$	[*p.1.mutex.down*, *p.1.enter*, *p.1.exit* (*p.2.mutex.down*, *p.2.enter*, *p.2.exit*)$^\omega$]
	$\tau^4_{\texttt{EXIT\_1}} =$	[(*p.1.mutex.down*, *p.1.enter*, *p.1.exit*, *p.2.mutex.down*, *p.2.enter*, *p.2.exit*, *p.2.mutex.up*)$^\omega$]

も近い事象を求めることで得られる．これらの遷移は，Sema が，*p.1.mutex.down* によって p.1 が危険領域に入ることを許可した後で *p.1.mutex.up* によって p.1 を脱出させることなく，*p.2.mutex.down* によって p.2 が危険領域に入れるようにしてしまっているという誤りの原因を示す．

一方，もう 1 つの正例 $\tau^2_{\texttt{EXIT\_1}}$ は，$\pi_{\texttt{EXIT\_1}}$ の 4 番目の事象 *p.2.enter* を *p.1.exit* に置換した結果得られる正例である．したがって，$d_e(\pi_{\texttt{EXIT\_1}}, \tau^2_{\texttt{EXIT\_1}}) = 1$ である．$\pi_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ との両者の違いは次のことを意味すると解釈される．すなわち，$\pi_{\texttt{EXIT\_1}}$ が EXIT_1 を満たすためには，$\pi_{\texttt{EXIT\_1}}$ の閉路に *p.2.enter* が含まれてはいけない．ゆえに，p.1 が危険領域にいる間に p.2 はそこに無限回入ってはいけないことを $\tau^2_{\texttt{EXIT\_1}}$ が明らかにしている．$\pi_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ とを比較することで，Sema プロセスの遷移 $(1, \mathit{p.2.mutex.down}, 2)$ が p.2 を危険領域に入ることを許す誤った遷移として抽出される．

$\tau^3_{\texttt{EXIT\_1}}$ は，$\pi_{\texttt{EXIT\_1}}$ の 2 番目の事象 *p.1.enter* の直後に *p.1.exit* を挿入し，また，*p.2.mutex.up* を削除することで得られる正例である．よって，少なくとも 2 回の編集操作が $\pi_{\texttt{EXIT\_1}}$ を $\tau^3_{\texttt{EXIT\_1}}$ へと変換するには必要であり，$d_e(\pi_{\texttt{EXIT\_1}}, \tau^3_{\texttt{EXIT\_1}}) = 2$ となる．これらの編集操作のうち，前者は $\tau^1_{\texttt{EXIT\_1}}$ を求めるための操作と同一であり，$\pi_{\texttt{EXIT\_1}}$ が EXIT_1 を満たすように直すために必要な修正操作である．しかしながら，後者の削除操作は，性質を満たすように反例を修正するためには，不必要な操作である．それにも関わらず，後者の削除操作を開発者に提示することで，$\pi_{\texttt{EXIT\_1}}$ において *p.2.mutex.up* は生起してはいけない事象であり，性質違反の原因を指し示していると開発者が誤解する可能性がある．このような反例に含まれる性質違反の要因と無関係な編集操作を含む正例は，誤り特定手法の結果として出力されないことが望ましい．距離 d_e により，$\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ は $\tau^3_{\texttt{EXIT\_1}}$ よりも $\pi_{\texttt{EXIT\_1}}$ に近い正例であると判定されるので，$\tau^3_{\texttt{EXIT\_1}}$ がモデルの誤り特定に使われることはない．

最後の正例 $\tau^4_{\texttt{EXIT\_1}}$ もまた $\pi_{\texttt{EXIT\_1}}$ が示す誤りの特定には不適当である．閉路の繰り返しを無視するならば，$\tau^4_{\texttt{EXIT\_1}}$ は $\tau^1_{\texttt{EXIT\_1}}$ と同一の事象列であるが，$\tau^1_{\texttt{EXIT\_1}}$ の接頭辞が閉路に移された軌跡である．すなわち，$\tau^4_{\texttt{EXIT\_1}}$ は $\pi_{\texttt{EXIT\_1}}$ を構成する事象列に 1 回の編集操作を適用することで得られるので，$d_e(\pi_{\texttt{EXIT\_1}}, \tau^4_{\texttt{EXIT\_1}}) = 1$ である．しかしながら，$\tau^4_{\texttt{EXIT\_1}}$ は以下のどちらにも解釈可能なので，開発者が性質違反の原因を決定するために有用な情報を提供できない．

- $\pi_{\texttt{EXIT\_1}}$ の接頭辞は無限回繰り返されるべきである．
- $\tau^4_{\texttt{EXIT\_1}}$ の接頭辞には誤りが含まれず，$\tau^1_{\texttt{EXIT\_1}}$ の場合と同様に，*p.1.exit* だけが EXIT_1 の違反を修正するのに重要な性質違反を引き起こす事象である．

よって，$\tau^4_{\texttt{EXIT\_1}}$ を用いて $\pi_{\texttt{EXIT\_1}}$ の性質違反の適切な原因を指摘することができない．ゆえに，類似度の指標である距離関数によって，$\tau^4_{\texttt{EXIT\_1}}$ は $\tau^1_{\texttt{EXIT\_1}}$ ほど $\pi_{\texttt{EXIT\_1}}$ と近い正例ではないと判定されるのが望ましい．ところが，$d_e(\pi_{\texttt{EXIT\_1}}, \tau^4_{\texttt{EXIT\_1}}) = 1$ であるので，d_e を用いると，$\tau^1_{\texttt{EXIT\_1}}$ と同じく $\tau^4_{\texttt{EXIT\_1}}$ は $\pi_{\texttt{EXIT\_1}}$ と最も近い正例であると判定される．この問題の原因は，d_e は，無限長の軌跡を構成する接頭辞と閉路とを区別せずに編集距離を計算することにある．他の先行研究 [61, 106, 7, 49, 88, 31, 47] は軌跡が有限長であると仮定しているので，同様の問題点を持っている．

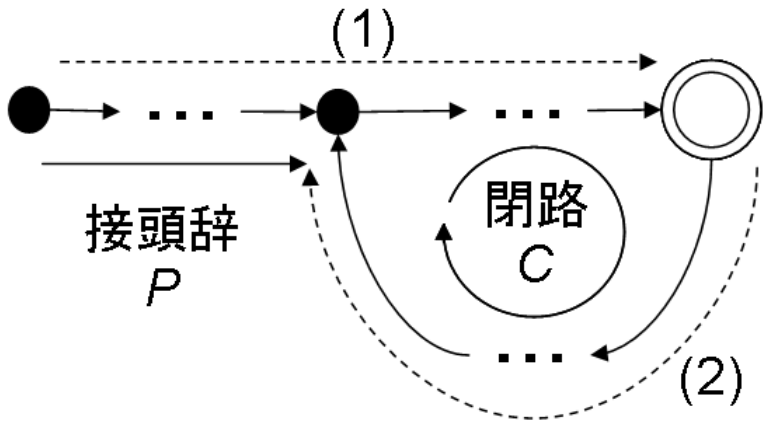


図 5.1. 正例の形状

以上の議論より，モデルの誤りを特定するためには，正例として $\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ を求めることが望ましいが，$\tau^3_{\texttt{EXIT\_1}}$ と $\tau^4_{\texttt{EXIT\_1}}$ は除外されることが望ましいと結論できる．そのために，$\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ が $\pi_{\texttt{EXIT\_1}}$ と最も距離が近いと判定され，$\tau^3_{\texttt{EXIT\_1}}$ や $\tau^4_{\texttt{EXIT\_1}}$ は距離がより遠いと判定される指標が必要である．

LLL-F は無限長軌跡間の類似度を測定できる距離を定義しておらず，$\tau^1_{\texttt{EXIT\_1}}$ を発見することができるが，$\tau^2_{\texttt{EXIT\_1}}$ を求めることはできない．この原因は，LLL-F において正例を求める手続きにある．LLL-F には，正例の接頭辞が性質のテスタオートマトンの受理状態に到達する事象列であるという前提がある．しかし，一般に性質を満たす軌跡は図 5.1 に示す形状をなすので，接頭辞が受理状態で終わるとは限らない．この例の場合のテスタオートマトン $TA(\texttt{EXIT\_1})$ は図 4.1 で表される．$\tau^1_{\texttt{EXIT\_1}}$ の接頭辞 [*p.1.mutex.down*, *p.1.enter*, *p.1.exit*] は受理状態 b_0 に到達するが，$\tau^2_{\texttt{EXIT\_1}}$ の接頭辞 [*p.1.mutex.down*, *p.1.enter*] は状態 b_1 に到達し，b_0 には達しない．ゆえに，誤り特定の精度を高めるためには，$\tau^2_{\texttt{EXIT\_1}}$ が求められるように LLL-F を拡張する必要がある．

■

上で例示した課題の解決のために，図 5.1 に示す形状をなす投げ縄型正例を，反例との定量的な距離に基づいて，類似した正例を求める手続きが必要である．ただし，この距離は，軌跡の接頭辞と閉路を区別して扱う指標でなくてはならない．以下では，これらを克服し，無限長文字列間の距離に基づき LLL-F を精緻化した正例を求める手法 LLL-S を提案する．

5.2　編集距離に基づく類似正例の生成

本節では，提案するモデルの誤り特定法である LLL-S の詳細を説明する．まず，反例と正例の類似度を計量することのできる投げ縄型の無限長軌跡間の距離を，編集距離に基づいて定

義する．LLL-S はここで定義した距離が反例と最小となる（すなわち，反例と最も類似する）正例をモデルを用いて求める．次に，LLL-F と同様に，反例と求めた正例との違いを比較することでモデルの誤りの候補を発見する．この方法によって，前節で挙げた問題点を解決することができる．

LLL-S への入力は，システムの振る舞いを記述した LTS $L = (S, A, \Delta, s_0)$，FLTL 式で記された性質 ϕ，反例 $\pi = PC^\omega$ の 3 種類である．ただし，$L \not\models \phi$ とする．また，$0 \leq m, 1 \leq n$ なる m, n に対して，$P = [a_0,\ a_1, \ldots, a_{m-1}]$ と $C = [b_0,\ b_1, \ldots, b_{n-1}]$ は有限長事象列である．性質 ϕ のテスタオートマトンを $TA(\phi) = (S_\phi, A_\phi, \Delta_\phi, u_0, S^a_\phi)$ と記す．ただし，以下の議論において $A_\phi \subseteq A$ とする．求める正例 τ は，$\tau = P'C'^\omega$ の形状をなす．

A の要素である各事象を文字とみなすならば，L 上の軌跡は無限長文字列と考えることができる．LLL-F の場合と同様に，以下の 3 種の編集操作に関して 2 つの有限長文字列 s_1, s_2 の間の編集距離を $d(s_1, s_2)$ とする [71, 82].

- **置換操作**：一方の文字列の文字を異なる文字に置き換える．
- **削除操作**：一方の文字列の文字を削除する．
- **挿入操作**：一方の文字列に文字を挿入する．

既に述べた通り，2 つの有限長文字列 s_1, s_2 間の編集距離 $d(s_1, s_2)$ は，一方の文字列を他方に変換するのに必要な最小の編集コストである．本章において，置換，削除，挿入の各編集操作のコストを 1 とする．次に，d を用いて π と τ との間の類似度の指標である距離 D を定義する．

本論文では，正例や反例は投げ縄型の無限長の軌跡を仮定しているので，PC^ω の形状をなす全ての文字列の集合を考える．2 つの文字列 $\pi_1 = P_1C_1^\omega$ と $\pi_2 = P_2C_2^\omega$ についての相等関係を以下のように定める．

> π_1 と π_2 が等しいのは，π_1 と π_2 が接頭辞と閉路を区別して同一の文字列のとき，すなわち，$P_1 = P_2$ かつ $C_1 = C_2$ のときであり，かつそのときに限る．

すると，この相等関係は明らかに反射律，対称律，推移律を満たす同値関係である．文字列間の類似度の指標を編集距離に基づき定めるために，両者を接頭辞と閉路を構成する有限長の事象列に分割して距離を定義する．反例と正例の接頭辞間の編集距離と両者の閉路間の編集距離をそれぞれ求め，その和を反例と正例の距離 D とする．

定義 18.（**軌跡間の距離**）Σ を文字集合，P, C, P_1, C_1, P_2, C_2 を Σ 上の有限語の正規表現であるとする．このとき，Σ 上で PC^ω の形の全ての無限語の集合を $\mathcal{PC}^\omega_\Sigma$ で表す．さて，$\pi_1 = P_1C_1^\omega, \pi_2 = P_2C_2^\omega \in \mathcal{PC}^\omega_\Sigma$ を無限長文字列とする．このとき，π_1 と π_2 の距離を $D : \mathcal{PC}^\omega_\Sigma \times \mathcal{PC}^\omega_\Sigma \to \mathcal{R}$ によって，以下のように定義する．

$$D(\pi_1, \pi_2) = d(P_1, P_2) + d(C_1, C_2)$$

■

定理 2. PC^ω の形の全ての無限語の集合 $\mathcal{PC}^\omega_\Sigma$ 上で文字列間の相等関係を上のように定めると，D は $\mathcal{PC}^\omega$ 上の距離関数である．すなわち，以下の条件を全て満たす．

1. 任意の $\pi_1 = P_1C_1^\omega, \pi_2 = P_2C_2^\omega \in \mathcal{PC}^\omega_\Sigma$ に対して，$D(\pi_1, \pi_2) \geq 0$.
2. $\pi_1 = P_1C_1^\omega, \pi_1 = P_2C_2^\omega \in \mathcal{PC}^\omega_\Sigma$ において，$D(\pi_1, \pi_2) = 0$ となるのは，π_1 と π_2 が上で定めた意味で等しいときであり，かつそのときに限る．
3. 任意の $\pi_1 = P_1C_1^\omega, \pi_2 = P_2C_2^\omega \in \mathcal{PC}^\omega_\Sigma$ に対して，$D(\pi_1, \pi_2) = D(\pi_2, \pi_1)$.
4. 任意の $\pi_1 = P_1C_1^\omega, \pi_2 = P_2C_2^\omega, \pi_3 = P_3C_3^\omega \in \mathcal{PC}^\omega_\Sigma$ に対して，$D(\pi_1, \pi_3) \leq D(\pi_1, \pi_2) + D(\pi_2, \pi_3)$.

■

証明．条件 1 編集距離は距離関数であるので [82]，$d(P_1, P_2) \geq 0$ かつ $d(C_1, C_2) \geq 0$ である．よって，$D(\pi_1, \pi_2) \geq 0$ である．

条件 2 $D(\pi_1, \pi_2) = 0$ ならば，$d(P_1, P_2) = 0$ かつ $d(C_1, C_2) = 0$ である．$d(P_1, P_2) = 0$ かつ $d(C_1, C_2) = 0$ となるのは，$P_1 = P_2$ かつ $C_1 = C_2$ のときであるので，$D(\pi_1, \pi_2) = 0$ ならば π_1 と π_2 は上で定めた意味で等しい．逆に，π_1 と π_2 が上記の意味で等しいならば，$d(P_1, P_2) = 0$ かつ $d(C_1, C_2) = 0$ である．よって，$D(\pi_1, \pi_2) = 0$ である．

条件 3 $d(P_1, P_2) = d(P_2, P_1)$ かつ $d(C_1, C_2) = d(C_2, C_1)$ なので成り立つ．

条件 4 $D(\pi_1, \pi_2) = d(P_1, P_2) + d(C_1, C_2)$, $D(\pi_2, \pi_3) = d(P_2, P_3) + d(C_2, C_3)$ において，編集距離の性質より，

$$
\begin{aligned}
d(P_1, P_3) &\leq d(P_1, P_2) + d(P_2, P_3) \\
d(C_1, C_3) &\leq d(C_1, C_2) + d(C_2, C_3)
\end{aligned}
$$

よって，

$$
\begin{aligned}
D(\pi_1, \pi_3) &= d(P_1, P_3) + d(C_1, C_3) \\
&\leq (d(P_1, P_2) + d(C_1, C_2)) + (d(P_2, P_3) + d(C_2, C_3)) \\
&= D(\pi_1, \pi_2) + D(\pi_2, \pi_3).
\end{aligned}
$$

■

例 37. 表 5.1 によると，$D(\pi_{\texttt{EXIT\_1}}, \tau^1_{\texttt{EXIT\_1}}) = D(\pi_{\texttt{EXIT\_1}}, \tau^2_{\texttt{EXIT\_1}}) = 1$ である．一方，$D(\pi_{\texttt{EXIT\_1}}, \tau^3_{\texttt{EXIT\_1}}) = 2$，$D(\pi_{\texttt{EXIT\_1}}, \tau^4_{\texttt{EXIT\_1}}) = 5$ である．したがって，編集距離に基づいた軌跡間の距離として D を用いると，$\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ が $\pi_{\texttt{EXIT\_1}}$ と最も距離が近く類似した正例であり，$\tau^3_{\texttt{EXIT\_1}}$ と $\tau^4_{\texttt{EXIT\_1}}$ はそうではないと判定されるので，前節の考察より D が本論文における類似度として適当である．これは，先行研究で導入された距離 d_e と異なり，D が軌跡の接頭辞と閉路の距離を区別していることから得られる利点である．

■

5.2.1 節で LLL-S の概要を述べ，続いて，5.2.2 節から 5.2.5 節でその詳細を説明する．

5.2.1　LLL-S の概要

本節では，LLL-S が実施する誤り特定手法の概要を説明する．

4 章の議論より，性質 ϕ を満たす正例の集合は，$TA(\phi)$ が受理する軌跡の集合によって与えられる．この点を利用して，LLL-S は π と最も類似した正例 τ，すなわち，$D(\pi, \tau)$ が最小となる正例 τ を $TA(\phi)$ 上で探索する．その後，π と τ に対して 4.1.5 節の方法を適用して誤りを特定する．τ が $TA(\phi)$ において Büchi の受理条件を満たすように，2 段階の手続きによって τ を求める（図 5.1 参照）．

1. $TA(\phi)$ の初期状態 u_0 から受理状態 $s_\phi^a \in S_\phi^a$ に到達する事象列を探索する．求めた事象列は図 5.1 の事象列 (1) となる．
2. s_ϕ^a を始点とし，u_0 から s_ϕ^a までの経路上の状態に戻る事象列を探索する．このステップで得られる事象列は図 5.1 における事象列 (2) である．

探索対象モデル（WTS）構成法の概要　正例を求めるために，まず，反例 π から WTS W_π^A を構築する．W_π^A には，4 章と同様に π に対する編集操作とそのコストを埋め込む．π は PC^ω の投げ縄型構造であるので，W_π^A は，P に相当する非閉路有向道と，それに続く C に相当する閉路によって構成される．接頭辞 $P = [a_0, \ldots, a_{m-1}]$ に対しては，状態 p_i と遷移 (p_i, a_i, p_{i+1}) $(i = 0, \ldots, m-1)$ を生成する．また，閉路 $C = [b_0, \ldots, b_{n-1}]$ に対しては，状態 c_i と遷移 (c_i, b_i, c_{i+1}) $(i = 0, \ldots, n-1)$ を生成する．ただし，c_n は c_0 と同一の状態であるとする．以上で生成した遷移は反例と同一の軌跡を表すので，重みを 0 とする．

これらの遷移を，編集操作に相当する以下の 3 種類の遷移を追加することによって拡充する．

1. **置換操作**：状態の組 (p_i, p_{i+1}) に対して，重み 1 を持つ遷移 (p_i, a, p_{i+1}) を生成する．ここで，$a \notin A - \{a_i\}$ である．この遷移は，a_i を a に置換する編集操作を表す．同様に，状態の組 (c_i, c_{i+1}) に対して，重み 1 を持つ遷移 (c_i, b, c_{i+1}) を生成する．ただし，$b \notin A - \{b_i\}$ である．
2. **削除操作**：状態の組 (p_i, p_{i+1}) に対して，重み 1 を持つ遷移 (p_i, ϵ, p_{i+1}) を生成する．この遷移は，a_i を削除する編集操作を表す．特殊な事象 ϵ は，前章と同様に事象の削除を表す事象である．同様に，状態の組 (c_i, c_{i+1}) に対して，重み 1 を持つ遷移 (c_i, ϵ, c_{i+1}) を生成する．
3. **挿入操作**：状態 p_i に対して，重み 1 を持つ自己閉路 (p_i, a, p_i) を生成する．この遷移は，$a \in A$ を p_i（a_{i-1} と a_i との間）に挿入する編集操作を表す．同様に，状態 c_i に対して，重み 1 を持つ自己閉路 (c_i, b, c_i) を生成する．ただし，$b \in A$ である．

続いて，作成したモデル W_π^A と $TA(\phi)$ について，積モデル $W_\bowtie = TA(\phi) \bowtie W_\pi^A$ を構築する．積を求める演算 $\bowtie$ の定義は定義 16 に従う．積モデルを用いると，距離 D に関して反

例 π と最も近い ϕ の正例を探索する問題は，有向グラフ $W_{\bowtie}$ 上で初期状態を表す頂点から始まって，$TA(\phi)$ の 1 つの受理状態に対応する頂点を経由し，閉路を構成するように有向道を閉じた頂点への最短路探索問題に帰着させられる．ただし，$TA(\phi)$ の受理状態に相当する頂点は閉路に含まれなければならない．

正例探索アルゴリズムの概要　正例を探索する問題を解くために，Dijkstra 法に代表される古典的な単一始点の最短路探索アルゴリズムを利用することができる．最短路探索アルゴリズムは 2 段階に分けて実行される．第 1 のステップにおいて，初期状態を表す頂点から受理状態に対応する各頂点までの最短路を求める．続いて，第 2 のステップでは，初期頂点から到達可能な受理状態に対応する各頂点に対して，その頂点から始まって初期頂点から受理状態を表す頂点への最短路上にある各頂点への最短路探索問題を解く．第 2 のステップにおいて，正例の閉路を構成する経路を求めることができる．以上の議論により，$W_{\bowtie}$ で受理状態に相当する頂点（末尾状態数）の数を v_a とするならば，正例を求めるために最短路探索問題を $v_a + 1$ 回解く必要がある．

誤りの発見手続き　LLL-F と同様に，τ と π との違いを求めることで，誤りの候補を指摘することできる．モデルの誤り候補を求める方法には，4.1.5 節の方法を用いる．このようにして，L が複数のプロセスから構成される場合，LLL-S は各プロセスについて，π を τ に変換するために修正すべき π 中の事象を引き起こす遷移を求めて開発者に提示する．

5.2.2　反例の WTS モデルの構築

本節では，反例 π から WTS を構築する手法を述べる．この WTS は，$TA(\phi)$ の受理状態 $s^a_\phi \in S^a_\phi$ に到達する事象列を求めるために活用される．

π から構築する WTS は 2 つの部分モデルからなる．

- π の接頭辞 P から構成される部分モデル．この部分モデルは，P をその各事象に対する編集操作で拡充したモデルで，P に編集操作を適用した結果得られる事象列とその編集コストを表す．この P から構成される WTS を通過する有限長の軌跡によって，正例の接頭辞 P' が得られる．
- π の閉路 C から構成される部分モデル．この部分モデルは，C をその各事象に対する編集操作で拡充したモデルで，C に編集操作を適用した結果得られる事象列とその編集コストを表す．この C から構成される WTS を通過する有限長の軌跡によって，正例の接頭辞 C' が得られる．また，得られる事象列が閉路をなすようにするために，この部分モデルは有向閉路として構成する．

WTS は，P から構成される部分モデルに C から構成される部分モデルを連結することで構築され，π の各事象の生起順序を保存する．末尾状態の集合は，C から構成する部分グラフの全ての状態からなる．末尾状態をこのように定めるのは，WTS における有限長の軌跡が，初期状態から始まりその集合の要素で終わることを表現するためである．末尾状態は，$TA(\phi)$ の

受理状態で終わる事象列，すなわち図 5.1 の事象列 (1) を探索するために使われる．

以上より，反例 π の WTS $W_\pi^A = (S_w, A \cup \{\epsilon\}, \Delta_w, q_0, \zeta_w, M_w)$ は以下のように構成される．

- $S_w = \{p_i | 0 \leq i < m\} \cup \{c_i | 0 \leq i < n\}$.
- $m \neq 0$ ならば $q_0 = p_0$ とし，そうでない場合は $q_0 = c_0$ とする．
- $\Delta_w \subseteq S_w \times (A \cup \{\epsilon\}) \times S_w$ は遷移関係であり，$\Delta_w = \Delta_p \cup \Delta_b \cup \Delta_c$ とする．ここで，Δ_p，Δ_b，Δ_c は以下のように定められる．
  - $\Delta_p = \{(p_i, a, p_{i+1}) |\ 0 \leq i < m-1, a \in A \cup \{\epsilon\}\} \cup \{(p_i, a, p_i) | 0 \leq i < m, a \in A\}$.
  - $m \neq 0$ ならば，$\Delta_b = \{(p_{m-1}, a, c_0) | a \in A \cup \{\epsilon\}\}$，そうでない場合は，$\Delta_b = \emptyset$.
  - $\Delta_c = \{(c_i, a, c_{i+1}) | 0 \leq i < n-1, a \in A \cup \{\epsilon\}\} \cup \{(c_{n-1}, a, c_0) | a \in A \cup \{\epsilon\}\} \cup \{(c_i, a, c_i) | 0 \leq i < n, a \in A\}$.
- 各 $\delta \in \Delta_w$ に対して，以下の場合のいずれかを満たす場合は $\zeta_w(\delta) = 0$ とする．
  - $\delta = (p_i, a_i, p_{i+1})$ $(i = 0, \ldots, m-2)$,
  - $\delta = (p_{m-1}, a_{m-1}, c_0)$,
  - $\delta = (c_i, b_i, c_{i+1})$ $(i = 0, \ldots, n-2)$,
  - $\delta = (c_{n-1}, b_{n-1}, c_0)$.

  そうでない場合，$\zeta_w(\delta) = 1$ とする．
- $M_w = \{c_i | 0 \leq i < n\}$.

接頭辞 P から構成されるモデルは，状態 $p_i (0 \leq i < m)$, c_0 と遷移関係 Δ_p, Δ_b から構成される部分モデルである．ただし，Δ_b は，P の末尾 a_{m-1} を表す遷移およびその事象に対する置換，削除操作を表す遷移の集合である．遷移 (p_i, a_i, p_{i+1}) $(i = 0, \ldots, m-2)$ と (p_{m-1}, a_{m-1}, c_0) は P の各事象を表す．よってその重みは 0 である．他の遷移は，P に対するコスト 1 の編集操作を表し，以下のように解釈される．

- 置換操作：遷移 (p_i, a, p_{i+1}) (ただし，$0 \leq i < m-1, a \in A - \{a_i\}$) と (p_{m-1}, a, c_0) (ただし，$a \in A - \{a_{m-1}\}$) は，それぞれ事象 a_i と a_{m-1} を他の事象 a に置換する操作を表す．
- 削除操作：遷移 (p_i, ϵ, p_{i+1}) (ただし，$0 \leq i < m-1$) と (p_{m-1}, ϵ, c_0) は，それぞれ事象 a_i と a_{m-1} を P から削除する操作を表す．
- 挿入操作：自己閉路 (p_i, a, p_i) (ただし，$0 \leq i < m, a \in A$) は，事象 a_{i-1} と a_i の間に事象 a を挿入する操作を表す．

閉路 C からは状態 $c_i (0 \leq i < n)$ と遷移関係 Δ_c からなる部分モデルが構成される．接頭辞の場合と同様に，遷移 (c_i, b_i, c_{i+1}) $(i = 0, \ldots, n-2)$ と (c_{n-1}, b_{n-1}, c_0) は C の各事象を表す．特に，C の末尾から先頭への遷移 (c_{n-1}, b_{n-1}, c_0) によって反例の閉路を表現する．これらの遷移の重みは 0 となる．他の遷移は編集操作を表すので，重みをそれぞれ 1 とする．

- 置換操作：遷移 (c_i, b, c_{i+1}) (ただし，$0 \leq i < n-1, b \in A - \{b_i\}$) と (c_{n-1}, b, c_0) (

ただし，$b \in A - \{b_{n-1}\}$) は，それぞれ事象 b_i, b_{n-1} を他の事象 b に置換する操作を表す．

- 削除操作：遷移 (c_i, ϵ, c_{i+1}) (ただし，$0 \leq i < n-1$) と (c_{n-1}, ϵ, c_0) は，それぞれ事象 b_i と b_{n-1} を C から削除する操作を表す．
- 挿入操作：自己閉路 (c_i, b, c_i) (ただし，$0 \leq i < n, b \in A$) は，事象 b_{i-1} と b_i の間に事象 b を挿入する操作を表す．

P から構成されるモデルの初期状態 p_0 から c_0 に至る有限長事象列が P' である．同様に，C から構成されるモデルの状態 c_0 から各状態 c_i $(i = 1, \ldots, n-1)$ を経由して c_0 に戻る有限長事象列が C' である．初期状態 p_0 からの有限長事象列が末尾状態 c_i に到達するならば，これは事象列 $[a_0, \ldots, a_{m-1}, b_0, \ldots, b_{i-1}]$ に編集操作が適用された事象列である．この事象列が通過する遷移の重みの合計が編集コストである．編集コストが 0 の場合は，編集操作が適用されていない事象列 $[a_0, \ldots, a_{m-1}, b_0, \ldots, b_{i-1}]$ が得られる．末尾状態は，閉路 C の各状態である．それらのうち，$TA(\phi)$ の受理状態に相当する末尾状態が，5.2.3 節で行われる最短路探索の目的地となる．

例 38. 図 5.2 に CSys の例で示した反例 $\pi_{\texttt{EXIT\_1}}$ から構成した WTS $W^{A_1}_{\pi_{\texttt{EXIT\_1}}}$ を示す．このモデルは，上述の WTS 構築アルゴリズムにおいて $m = 2$, $n = 4$ とすることで得られる．$m \neq 0$ なので，図において初期状態は p_0 である．末尾状態は，点線で記された状態 c_i $(i = 0, \ldots, 3)$ である．図において，遷移の記法は 4 章に従う．

$W^{A_1}_{\pi_{\texttt{EXIT\_1}}}$ において，$\pi_{\texttt{EXIT\_1}}$ の接頭辞から構成される部分モデルは，状態 p_0, p_1, c_0 と p_0, p_1 からの出遷移からなる．重み 0 の遷移 $(p_0, \mathit{p.1.mutex.down}, p_1)$ と $(p_1, \mathit{p.1.enter}, c_0)$ はそれぞれ P の事象 *p.1.mutex.down* と *p.1.enter* を表す．遷移の集合 $(p_0, A_1 - \{\mathit{p.1.mutex.down}\}, p_1)$ は，$\pi_{\texttt{EXIT\_1}}$ の接頭辞の 1 番目の事象 *p.1.mutex.down* が他の事象に置換されるということを意味する．また，遷移の集合 $(p_1, A_1 - \{\mathit{p.1.enter}\}, c_0)$ は接頭辞の 2 番目の事象 *p.1.enter* に対する置換操作を意味している．同様に，遷移 (p_0, ϵ, p_1) と (p_1, ϵ, c_0) はそれぞれ事象 *p.1.mutex.down* と *p.1.enter* の削除である．自己閉路の集合 (p_0, A_1, p_0) と (p_1, A_1, p_1) は P のそれぞれ先頭への追加操作，および，*p.1.mutex.down* と *p.1.enter* の間への挿入操作である．同じようにして，状態 $c_i (i = 0, \ldots, 3)$ と，各状態からの出遷移によって，C から構成される部分モデルが得られる．

■

5.2.3　受理状態への事象列の探索

本節では，性質 ϕ の TA $TA(\phi)$ の受理状態 $s^a_\phi \in S^a_\phi$ で終わる事象列を探索する．求める事象列が，反例 π との距離が最も近くなるようにするために WTS W^A_π を用いる．まず，W^A_π と $TA(\phi)$ の積 $W_\bowtie = TA(\phi) \bowtie W^A_\pi$ を計算する．そして，この事象列を求める問題を積モデル $W_\bowtie$ 上の最短路探索問題に帰着させる．積を求める演算 $\bowtie$ には，4 章の定義 16 の演算を用

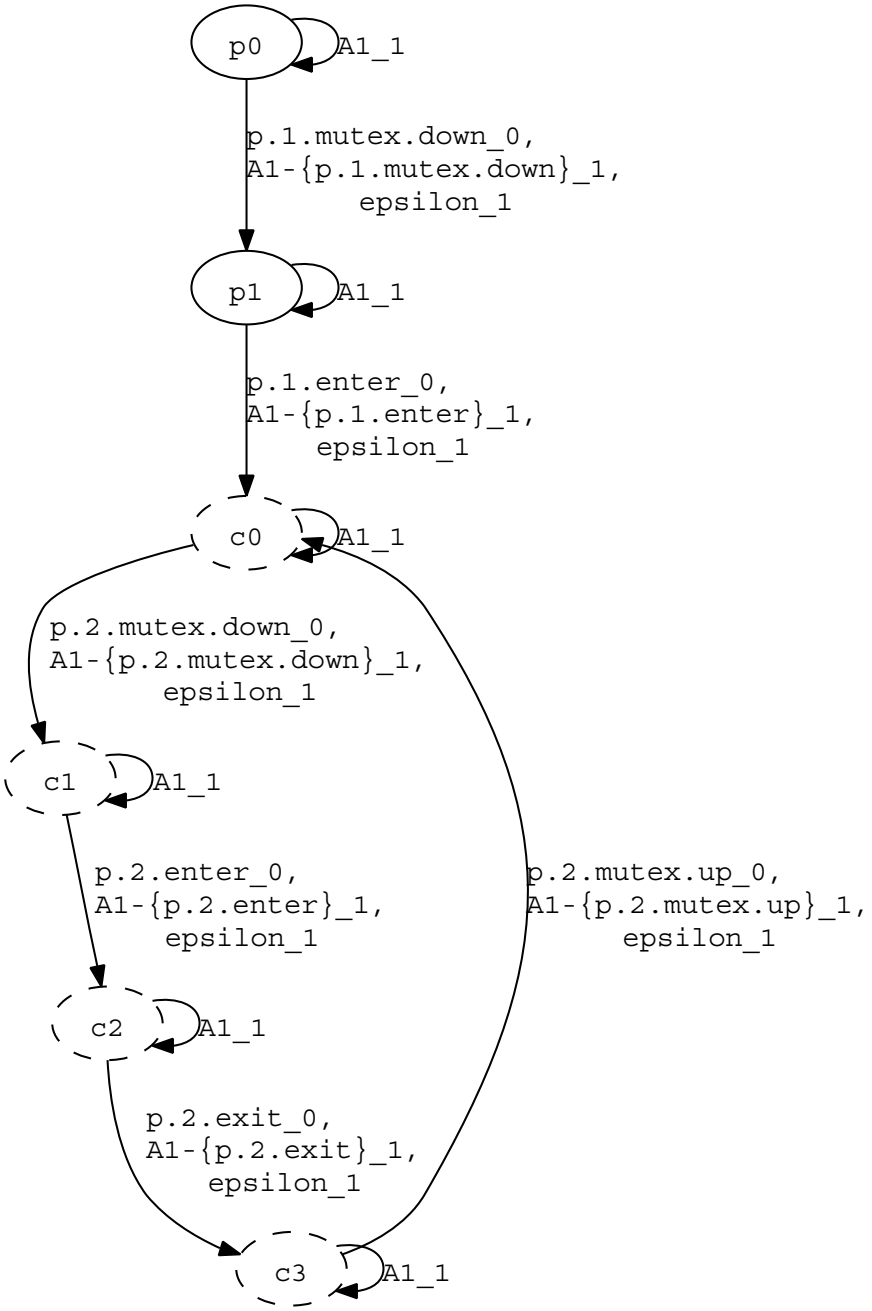


図 5.2. $\pi_{\mathrm{EXIT\_1}}$ の WTS

いる．

直観的には，積 $W_{\bowtie} = TA(\phi) \bowtie W_{\pi}^{A}$ は，π に適用される編集操作のコストを $TA(\phi)$ の各遷移にラベル付けした WTS と解釈される．$W_{\bowtie}$ の末尾状態 $(s_{\phi}^{a}, c^{M}) \in S_{\phi}^{a} \times M_{w}$ は $TA(\phi)$ の受理状態 s_{ϕ}^{a} であり，かつ，W_{π}^{A} の末尾状態 c^{M} である状態を意味する．$W_{\bowtie}$ の初期状態 (u_0, q_0) から (s_{ϕ}^{a}, c^{M}) への最短路を求めることで，s_{ϕ}^{a} で終わる事象列が得られる．このような最短路は Dijkstra 法 [36] によって計算することができる．この最短路を通過する事象列は

$H = [x_0, \ldots, x_{n_1-1}]$ (ただし，$\forall 0 \leq i < n_1.x_i \in A \cup \{\epsilon\}$) の構造を持つ．なお，積 $W_{\bowtie}$ に最短路探索を実施する際には，探索の効率化のために 4.1.2 節に挙げた前処理を実行する．

LLL-S は，$TA(\phi)$ の各受理状態に相当する全ての末尾状態への最短路を $W_{\bowtie}$ 上で計算する．続いて，各最短路を通過する事象列を求めることで，正例の部分列を得る．

例 39. 図 4.1 の TA $TA(\texttt{EXIT\_1})$ と図 5.2 の WTS $W^{A_1}_{\pi_{\texttt{EXIT\_1}}}$ を考える．これらの積を図 5.3 に示す．表示の簡潔化のために，図には正例 $\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ の探索に必要な事象のみを記述した遷移を記し，それ以外の遷移については重みのみを記述する．初期状態は (b_0, p_0) である．

初期状態 (b_0, p_0) から末尾状態 (b_0, c_0) への最短路の 1 つは，状態列 $[(b_0, p_0), (b_0, p_1), (b_1, c_0), (b_0, c_0)]$ である．この各状態を通過する事象列は，$TA(\texttt{EXIT\_1})$ の受理状態 b_0 への $\tau^1_{\texttt{EXIT\_1}}$ の部分列 $H^1 = [\ p.1.mutex.down,\ p.1.enter,\ p.1.exit]$ である．H^1 は WTS $W^{A_1}_{\pi_{\texttt{EXIT\_1}}}$ の状態 c_0 に相当する状態までの事象列なので，$\tau^1_{\texttt{EXIT\_1}}$ の接頭辞である．

一方，(b_0, p_0) からもう 1 つの末尾状態 (b_0, c_2) への最短路は $[(b_0, p_0), (b_0, p_1), (b_1, c_0), (b_1, c_1), (b_0, c_2)]$ である．よって，この状態列を通過する事象列を求めることによって，正例 $\tau^2_{\texttt{EXIT\_1}}$ の部分列 $H^2 = [p.1.mutex.down,\ p.1.enter,\ p.2.mutex.down,\ p.1.exit]$ が得られる．事象列 H^2 は，$\tau^2_{\texttt{EXIT\_1}}$ が事象 $p.1.exit$ の後に $TA(\texttt{EXIT\_1})$ の受理状態 b_0 に達することを示している．

■

5.2.4　正例の探索

本節では，正例の閉路を構成する閉じた事象列を探索して，反例と最も類似した正例を求める方法を詳しく説明する．

前節の第 1 ステップの最短路探索によって到達する積 $W_{\bowtie} = TA(\phi) \bowtie W^A_{\pi}$ の各末尾状態 $(s^a_{\phi}, c^M) \in S^a_{\phi} \times M_w$ に対して，(s^a_{ϕ}, c^M) から開始する第 2 の最短路探索を $W_{\bowtie}$ 上で実施して，各状態への最短路を求める．次に，第 1 ステップの探索で得られた状態 (u_0, q_0) から (s^a_{ϕ}, c^M) への最短路上にある各状態 $(s_{\phi}, s_w) \in S_{\phi} \times S_w$ までの事象列を求める．そして，この結果得られた事象列と前節で求めた事象列を組み合わせることで，正例が得られる．

前節に実施した初期状態 (u_0, q_0) から (s^a_{ϕ}, c^M) への第 1 の最短路探索で得られた事象列を $H = [x_0, \ldots, x_{n_1-1}]$ と書く．(s^a_{ϕ}, c^M) からの第 2 の最短路探索の結果，(s^a_{ϕ}, c^M) から始まり，(u_0, q_0) から (s^a_{ϕ}, c^M) への経路上の状態 (s_{ϕ}, s_w) までの最短路を通過する事象列は，$T = [x_{n_1}, \ldots, x_{n_2-1}]$ $(1 \leq n_1$ かつ $\forall i \in \mathcal{N}.(n_1 \leq i < n_2 \Rightarrow x_i \in A \cup \{\epsilon\}))$ と書くことができる．正例を計算するために，H から (s_{ϕ}, s_w) に到達する部分列 $[x_0, \ldots, x_{m_1-1}]$ $(0 \leq m_1 \leq n_1)$ を求める．この事象列が正例の接頭辞 $P'_{\epsilon} = [x_0, \ldots, x_{m_1-1}]$ となる．続いて，H の残り事象列と T を連結した事象列 $C'_{\epsilon} = [x_{m_1}, \ldots, x_{n_2-1}]$ を求める．P'_{ϵ} と C'_{ϵ} から全ての ϵ を除いた事象列をそれぞれ $P' = [x'_0, \ldots, x'_{m'-1}]$ と $C' = [x'_{m'}, \ldots, x'_{n'-1}]$ $(0 \leq m' \leq m_1, m' < n' \leq n_2$ かつ $\forall 0 \leq i < n'.x'_i \in A)$ とする．最後に，P' と C' を用いると，正例 $\tau = P'C'^{\omega}$ が得られ

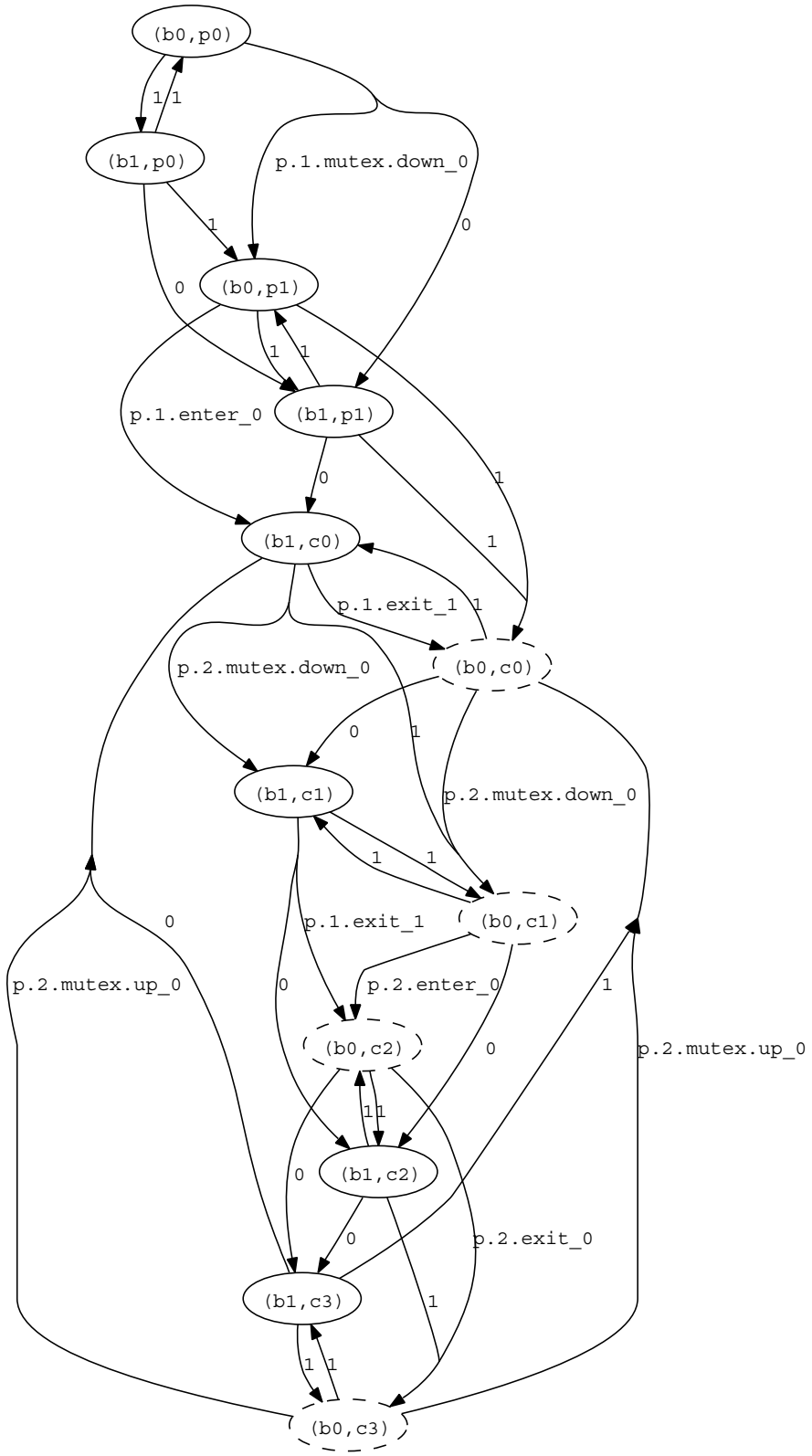


図 5.3. $TA(\texttt{EXIT\_1})$ と $W^{A_1}_{\pi_{\texttt{EXIT\_1}}}$ の積

る．ただし，τ は $TA(\phi)$ の受理条件件を満たさねばならないので，C' は空列であってはならない．よって，C' が空列であるような τ は，正例の候補から除くこととする．

以上の手続きによって求められた全ての正例のうち，第 1 ステップの探索と第 2 ステップの探索の結果求められる編集距離 (WTS 上のコスト) の和が最小となるもののみを集めることで，反例との距離 D が最も小さい正例が求められる．同一の正例が重複して求められた場合，その中の 1 つだけを残して他の正例を除く．

例 40. 正例 $\tau^1_{\texttt{EXIT\_1}}$ の探索を行う．前節の結果から，積 $W_{\bowtie} = TA(\phi) \bowtie W^A_{\pi}$ 上の第 1 ステップの探索において，末尾状態 (b_0, c_0) への事象列 H^1 が探索されたので，第 2 ステップの探索において (b_0, c_0) からそれ自身への最短路探索を実行する．その結果，得られる最短路は $[(b_0, c_0), (b_0, c_1), (b_0, c_2), (b_0, c_3), (b_0, c_0)]$ であり，この経路を通過する事象列 $T^1 = [p.2.mutex.down, p.2.enter, p.2.exit, p.2.mutex.up]$ が得られる．最後に，$H^1 T^{1\omega}$ とすることで，$\tau^1_{\texttt{EXIT\_1}}$ を求めることができる．

もう一つの例として，正例 $\tau^2_{\texttt{EXIT\_1}}$ を求めることを考える．積 $W_{\bowtie}$ の末尾状態 (b_0, c_2) は第 1 ステップの最短路探索において到達した状態であり，初期状態 (b_0, p_0) から末尾状態 (b_0, c_2) までの最短路上にある状態集合は $\{(b_0, p_0), (b_0, p_1), (b_1, c_0), (b_1, c_1), (b_0, c_2)\}$ である．正例を構成する残りの事象列の探索のため，状態 (b_0, c_2) を始点とする最短路探索を実行する．状態 (b_0, c_2) から上記の集合の要素 (b_1, c_0) への最短路を通過する事象列は，$T^2 = [\ p.2.exit, p.2.mutex.up]$ である．よって，T^2 と前節で求めた事象列 H^2 を組み合わせることにより，$\tau^2_{\texttt{EXIT\_1}}$ が得られる．

■

5.2.5　誤りの同定

モデル L 上の誤りを発見するために，求めた正例と反例を比較することで，両者の違いを引き起こす原因となる遷移を求める．この誤りを求める方法には，4.1.5 節のアルゴリズムを用いる．

例 41. $\pi_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ を用いて CSys の誤り候補を指摘することにする．$\pi_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ を比較すると，不整合事象は $\pi_{\texttt{EXIT\_1}}$ の閉路の 2 番目の事象 *p.2.enter* である．LLL-S は，この不整合事象を用いて，p.1，p.2，および Sema に対して誤り候補を求める．まず，Sema について考える．*p.2.enter* は，Sema の事象の集合の要素ではないので，誤りの原因を発見するために Sema に現れる先行する事象を調べる．*p.2.enter* より前に，そして最後に起こる Sema の事象は *p.2.mutex.down* である．LLL-S は遷移 $(1, p.2.mutex.down, 2)$ を Sema の誤りの候補として開発者に報告する．この遷移は，*p.2.mutex.down* が不整合事象 *p.2.enter* を引き起こすと解釈される．同様の方法を p.1 と p.2 に適用することにより，LLL-S は他の誤り候補として，p.1 の遷移 $(1, p.1.enter, 2)$ と p.2 の遷移 $(1, p.2.enter, 2)$ を返す．

$\pi_{\texttt{EXIT\_1}}$ と $\tau^1_{\texttt{EXIT\_1}}$ を用いた誤り特定手続きの実行結果は，4.1.5 節に $\pi_{\texttt{EXIT\_1}}$ と示したとおり

である．開発者は，これらの結果を通じて，Sema に含まれる排他制御機構の誤りを容易に発見することが可能である．

既に論じたように，LLL-F は，正例 $\tau^1_{\texttt{EXIT\_1}}$ は発見できるが，$\tau^2_{\texttt{EXIT\_1}}$ を発見することができない．したがって，LLL-F は $\tau^1_{\texttt{EXIT\_1}}$ が示す $\pi_{\texttt{EXIT\_1}}$ の接頭辞中の性質違反の不整合事象を発見すことができるが，$\tau^2_{\texttt{EXIT\_1}}$ が示す $\pi_{\texttt{EXIT\_1}}$ の閉路内に存在する誤りの候補を明確に指摘することができない．LLL-S は，距離 D に基づいて正例を発見して誤りを特定するので，反例の閉路内に含まれる誤りの候補を発見することができる手法である．以上より，LLL-F も $\tau^1_{\texttt{EXIT\_1}}$ を通じて Sema の排他制御機構の誤りを発見できるが，LLL-S の方がより精緻に誤りを指摘できる．

CSys および性質 EXIT_1 に対して LLL-S を実行することで，$\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ を含めて 11 種類の正例が発見された．ただし，同一の編集操作を同一の事象に適用することで得られる正例は等価とみなし，等価な正例の重複を除いた個数のみを考える．これらの正例は，以下の 2 種類に分類される．

- EXIT_1 の後件に含まれる *p.1.exit* が起こる正例，すなわち，p.1 が危険領域に入った後に，危険領域から出る事象が起こる正例である（10 種類）．これらの正例は，$\pi_{\texttt{EXIT\_1}}$ の *p.1.enter* の後に，*p.1.exit* が挿入された軌跡，あるいは *p.1.enter* の後に現れる事象が *p.1.exit* に置換された軌跡である．この種の正例には，$\tau^1_{\texttt{EXIT\_1}}$ と $\tau^2_{\texttt{EXIT\_1}}$ が含まれており，Sema の排他制御機構の誤りを指摘した正例が求められている．ただし，ここに分類される正例が指摘した Sema プロセスの誤りの候補には，上で述べたものの他に $(2, p.2.mutex.up, 1)$ もある．この誤り候補を指摘した正例は，p.2 が危険領域から出た後に p.1 が危険領域から出ることを表していた．この正例は，EXIT_1 を確かに満たしているが，Sema の果すべき役割である排他制御の誤りを適切に指摘しているとはいえない．
- EXIT_1 の前件に含まれる *p.1.enter* が実行されない正例，すなわち，p.1 が危険領域が入らない正例である（1 種類）．この正例は，$\pi_{\texttt{EXIT\_1}}$ の *p.1.enter* が異なる事象に置換されたもので，以下の軌跡で表される．

  $$[p.1.mutex.down,\ p.1.exit\ (\ p.2.mutex.down,\ p.2.enter,\ p.2.exit,\ p.2.mutex.up)^{\omega}]$$

  この正例が示す Sema プロセスの誤りは，$\tau^2_{\texttt{EXIT\_1}}$ のときと同様に，p.1 が危険領域に入る遷移 $(0, p.1.mutex.down, 1)$ である．ただし，$\tau^2_{\texttt{EXIT\_1}}$ が意味することは，p.2 が危険領域に入る前に p.1 は危険領域から出なければならないということと解釈される．一方，上の正例は，p.2 が危険領域に入る前に p.1 が危険領域に入ってはいけないということを表すと解釈できる．

■

上の例の議論により，同じ誤り候補を指摘する全ての正例が同じように解釈されるわけではないということが分かる．よって，誤りを適切に指摘する正例の解釈が問題領域に合致するとは限らないので，開発者は問題領域の性質を考慮して適切な正例を選択する．このように

LLL-S が提示する正例を解釈することによって，開発者は反例が性質に違反する理由を理解することができるので，その解釈に基づいて適当な誤りを選択することが可能である．したがって，LLL-S が生成する正例を，性質を定式化した論理式が表す意味によって分類することができるならば，開発者が正例をより容易に理解することの支援になると考えられる．この点については，5.5 節における「LLL-S が発見する正例の性質の意味に基づく分類」の項で詳しく議論する．

5.3 実装

われわれは，LLL-S を自動化したプロトタイプツールを Java 言語で実装した．本節では，実装について紹介する．LLL-F と同様に，プロトタイプツールは，モデル，性質の TA，および反例を入力とする．そして，LLL-S を実行し，正例，適用された編集操作列，各正例から求めた誤り候補の遷移を出力する．モデルおよび性質の TA は，Aldebaran 形式 [1] によって記述されている必要がある（拡張子".aut"を持つテキストファイルで入力する）．

実行性能を高めるために，プロトタイプツールには次のような発見的な最適化処理を実装した．

- 5.2 節で述べた通り，LLL-S は，反例から構築した WTS モデルと性質の TA との積モデルを構成し，その後，その積モデルに対して最短路探索を行うことによって正例を探索する．しかしながら，この方法は，積モデルの計算と積モデル上の最短路探索において全状態空間の探索を最初から実行しなければならず，実用的には効率的ではない．そこで，LLL-F の実装と同様に，プロトタイプツールは両者を同時行うように処理を実装している．すなわち，積モデル上を最短路探索を行う際に，必要に応じて定義 16 に基づいて積モデルの未探索の部分モデルを構成する．この方法によって，正例の探索時間の短縮のみならず，初期ノードから到達不可能なノードを生成しなくてよいという利点も得られる．
- 同一の経路（状態列）を通って得られる正例が複数ある場合は，それらのうちの代表例のみを出力する．同一の編集操作を反例に適用して得られた代表的な正例だけを出力することで，正例の重複を防ぎ，生成する正例の数を絞り込むためである．この処理については，既に 5.2.5 節で述べた．
- 5.2.3 節の第 1 ステップと 5.2.4 節の第 2 ステップの探索において，最短路探索の終点となる末尾状態を，探索の結果計算される最短距離を用いて昇順に整列する．また，第 1 ステップと第 2 ステップの探索によって最小の距離 D を持つ正例を発見すると，プロトタイプツールは正例と共にその距離を記録する．以上の手続きを利用して，次の手順で効率的に正例を生成する．
  - 第 1 ステップの探索において計算される初期状態から末尾状態までの距離が，それ以前の正例探索の結果記録されている距離 D の最小値よりも大きい場合には，この状態を始点とする第 2 ステップの探索を行わず，正例の探索を終了する．なぜな

ら，この場合，第 2 ステップの探索を実行したとしても，その結果計算される距離 D が最小となることはないからである．

  - 第 1 ステップの探索において計算される初期状態から末尾状態までの距離が，それ以前の正例探索の結果記録されている距離 D の最小値以下の場合には，引き続き第 2 ステップの最短路探索を実施する．この 2 回の探索の結果得られる距離 D が，記録されている距離 D の最小値以下の正例のみを求める．また，このとき D の最小値を更新する．

- 削除操作は，事象 ϵ への置換操作とみなすことにより，他の事象への置換操作と同様に扱う．これによって，モデル上の同一の遷移を誤り候補として指摘する冗長な正例が生成されないようにすることができる．
- LLL-F と同様に， 複数プロセスからなるモデルにおいて，反例と正例の比較から同定された修正すべき事象からは，各プロセス上で反例を実行して，そのプロセスの誤り候補となる遷移を求める．よって，各プロセスを並列合成したシステム全体の振舞いを表すモデルを求める必要がないので，多くのプロセスからなる大規模なモデルに対しても LLL-S を適用できる．

プロトタイプツールは，活性だけでなく，反例が有限長の軌跡となる安全性に対する誤り特定も実行できる．性質が安全性の場合，4.2 節で述べたように反例の閉路は空列である．よって，その空列への 1 回以上の事象の挿入操作が行われることで正例の閉路が求められるように，反例から WTS を構成する手法を書き換えればよい．もしくは，反例の閉路は事象列 $[\epsilon]$ の無限回繰返しであると考えることもできる．したがって，閉路を構成する事象 ϵ が他の事象に置換されるように WTS を構成するとみなしてもよい．したがって，LLL-S の方法論を変更することなく安全性を扱うことができる．なお，探索する正例は性質の TA 上で Büchi の受理条件を満たすので，無限長であって図 5.1 の投げ縄型構造をなす．

5.4　事例研究

本節では，実装した LLL-S のプロトタイプツールを用いて実施した事例研究の結果を報告する．

5.4.1　有効性の評価

各事例研究は，4.4 節と同様に，以下の手順で実施した．

1. システムの振る舞いを表現する一つもしくは複数の並行動作するプロセスからなる LTS モデルを構成する．
2. モデルが満たさない性質を用意し，FLTL で記述する．
3. モデル検査器 LTSA [74] を用いて性質の検証を行い，反例と性質の TA を出力として得る．

4. システムのモデル，性質の TA，および反例を入力として実装したプロトタイプツールを実行する．
5. ツールから出力された正例と誤り候補であるモデルの遷移を調査し，正例が誤りを適切に説明しているか，出力された遷移が適切に誤りを指摘しているか検討する．

事例研究は，4.4 節で取り上げた以下の 7 種類のモデルについて行った．

- 電子レンジシステム（MOvn） [25]
- キャッシュコヒーレンスプロトコル（AFS-1） [104]
- 図 2.3 の並行システム（CSys）
- 鉱山用排水ポンプ制御システム（MPmp） [97]
- 分散データベース（DDb1, DDb2, DDb3） [74]

事例研究で用いたモデルの規模，検証した性質の TA の規模，および LTSA が出力した反例の規模は表 4.2 と同一である．

表 5.2 に，各事例に対して LLL-S が生成した正例の数と実行時間を示す．実行時間は，4.4 節と同様に各事例に対して 10 回実行した平均時間である．実行時間には，入力を読み込む時間や，出力のための文字列を整形する時間，整形した文字列をファイルに出力するのに要する時間を含めなかった．また，実行時間の計測は，2GB RAM を備えた 3.4GHz Pentium 4 (JDK 1.6.0) 上で行った．

LLL-S は正例をシステムのモデル上ではなく，検証対象の性質の TA 上で探索する．よって，表 5.2 より，LLL-F ほどの性能ではないが，モデルの規模が非常に大きい DDb3 のようなシステムに対しても実用的な時間で実行できる手法であることが分かる．5.5 節で LLL-S の正例探索時間について詳しく議論する．

以下，各事例に対して LLL-S を適用した結果を述べる．

表 5.2. 各事例に対する LLL-S による正例の生成数と実行時間

システム名	性質名	正例数	誤りを指摘した正例数	実行時間 [sec]
MOvn	`HEAT`	9	0	0.29
AFS-1	`VALID`	8	2	0.05
CSys	`MUTEX`	7	7	0.04
	`EXIT_2`	12	3	0.35
MPmp	`EMG`	9	2	0.16
DDb1	`QUIS`	23	2	0.25
DDb2	`QUIS`	40	10	5.37
DDb3	`SAFE`	467	57	37.26

MOvn

この事例について，表 4.2 の条件に従うと LLL-S は適切な正例を発見することができなかった．この現象の発生原因は，LLL-S が反例の形に依存する点にある．MOvn の事例では，モデル検査器が出力した反例の接頭辞と閉路は，同一の事象列から構成されていた．すなわち，反例は PP^ω という構造だった．性質 `HEAT` に違反する原因となる誤りを P が含んでいたので，この反例を適切に修正した正例を求めるためには，反例の接頭辞と閉路の両者を編集する必要があった．しかしながら，LLL-S を実行した結果，反例の閉路のみを編集した正例が出力された．そこで，われわれは反例 PP^ω を P^ω に手作業で変換して，LLL-S を再度実行した．このとき，反例の接頭辞長が 0，閉路長が 4 である．その結果，LLL-S が生成した正例の総数は 10 で，そのうち誤りを適切に指摘した正例が 2 あった．LLL-S の実行時間は 0.24 秒だった．以上のように正例の接頭辞が閉路の部分列である時には，この接頭辞を削除することでこの問題点を解決することができた．

この修正を加えた反例に対して，LLL-S が出力した正例は以下の通り分類された．

- 扉が開いている間に電子レンジが調理開始した後に，レンジ内が加熱される．
- 扉が開いている間は電子レンジが調理を開始することがない．

前者の正例は，扉を開いた状態で調理機能が実行されることを示しているので，性質違反の原因を適切に指摘しているとはいえない．誤りを指摘するのに適当な正例は，後者の場合であった．

AFS-1

AFS-1 とその安全性 `VALID` に対して LLL-S を適用した結果得られたモデルの誤り候補は，LLL-F と同一であった．ただし，表 4.3 と表 5.2 に示すように，生成した正例の総数は LLL-F より LLL-S の方が多くなった．これは，LLL-F で探索できなかった距離 D が最小の正例を探索できたためである．同様の現象は，安全性を検証した他の事例についても当てはまった．LLL-S によって得られた正例は，LLL-F と同様に以下の 2 種類だった．

- クライアントのデータを有効とする．
- サーバのデータを無効にする．

われわれは，これらを元にサーバが持つ性質違反の原因を発見することができた．

CSys (`MUTEX`)

性質 `MUTEX` は 4.2 節で述べた安全性である．LLL-S が生成した正例は，次のように分類された．

- p.1 が危険領域に入っている間に，p.2 が危険領域に入ることがない．
- p.2 が危険領域に入る前に，p.1 が危険領域から脱する，あるいは，p.1 が危険領域に入

ることがない.

これらの正例はモデル上の異なる箇所を誤りとして指摘していたが，いずれも p.1 と p.2 が同時に危険領域に入ることを禁止する正例であった.

CSys (EXIT_2)

反例は，p.1 が危険領域にいる間に p.2 が無限回危険領域に出入りすることを表していた．LLL-S によって得られた正例の集合は，5.2.5 節で述べた EXIT_1 の正例と同様に分類された．それゆえ，これらの正例を用いることで，Sema の排他制御機構の誤りを指摘することができた.

この事例では，LLL-F は反例と距離 D に関して最も類似している正例を全て発見できたわけではなかった．この問題は，LLL-F が，接頭辞が EXIT_2 の TA の受理状態に到達しない正例を発見できない，という 5.1 節で提示した LLL-F の仮定が原因だった．それゆえ，LLL-F は反例の閉路のみを編集した正例が求められなかったが，LLL-S は D に基づいて性質違反となる閉路の事象のみを修正する正例が求められた.

MPmp

LLL-S を実行した結果得られる正例の集合は，いずれもメタンガスが発生した後に警告灯が点灯するように反例を修正した軌跡だった．このような正例は，警告灯の点灯を表す事象が閉路中に 1 箇所現れるように反例を修正することによって得られた.

一方，LLL-F は反例と距離 D に関して最も類似した正例を全て発見できたわけではなかった．この原因は CSys の性質 EXIT_2 の事例と同様に，LLL-F が探索した正例は，接頭辞が性質 EMG の TA の受理状態に到達する軌跡のみであることにあった.

また，LLL-F によって生成された適切に誤りを説明してはいない正例を，LLL-S は出力しなかった．この正例は，メタンガスが発生しないように反例を修正した軌跡だった．LLL-S がこの正例を出力しなかったのは，反例の接頭辞と閉路のそれぞれにメタンガスが発生する事象があったので，最も近い正例を得るには，それらの事象全てを修正する必要があるためだった．このように，距離 D の導入によって，誤りを発見するために不必要な正例を防止することも可能になった.

DDb1

LLL-S は，全てのノードが不活性になる正例と対応する誤り候補を生成した．これらは，不活性なノードのデータの更新を行わない軌跡，または，ノードが活性の時にコントローラが終了しないことを示す軌跡に分類された．その中には，LLL-F と同様に，コントローラが全てのノードが不活性になることを確認せずに終了するという性質違反の適切な原因が含まれていた．われわれは，性質違反を引き起こすコントローラの誤った振る舞いを回避する 2 つの正例を手掛かりとして，コントローラの誤りを発見した.

DDb2

LLL-S は，3 つのノードが同時には不活性にならないという反例が示す誤りを修正する正例を出力した．すなわち，生成された正例は，3 つのノードが全て不活性になるという軌跡であった．そのため，それらを用いてモデルの誤りを特定することができた．ただし，出力された正例には，全てのノードが同時に不活性になる時点に違いがあった．そこで，対象領域から判断して，8 件の適切な正例を選択して誤り発見の手掛かりとした．LLL-F は本章の最初に述べた問題点を持つためこの事例について誤りを特定できなかったが，距離 D とそれに基づいて反例と最も近い正例を生成することで，LLL-S はこの問題点を克服することができた．

DDb3

LLL-S によって得られた正例が表す意味は，以下の 2 通りのいずれかに分けられた．

- 不活性とならないノードが存在する．
- 全てのノードが不活性となったときに，全ノードの値が等しくなる．

よって，LLL-S が生成した 467 件の正例から，値が異なるデータを保持するノードが不活性とならないことを表す正例を選択する必要があった．この事例は，出力された正例の数が他の事例と比べて多かったので，誤りを指摘した正例を発見するために，手作業で正例を調べなくてはならなかった．ゆえに，LLL-F のときと同様に，正例間の順位付けを行うなどの仕組みを導入することによって，開発者が誤りを効率的に特定する仕組みが望まれる．

有効性の評価に関する結論

MOvn の事例において，接頭辞を修正する前の反例（表 4.2 の反例）において，LLL-F はモデルの誤りを適切に説明する正例を生成することができたが，LLL-S は適切な正例を発見することができなかった．上で議論したように，この現象の発生原因は，LLL-S が反例の形に依存する点にある．この事例では，正例の接頭辞が閉路の部分列だったので，この接頭辞を削除することでこの問題点を解決した．このような現象は，ここで行った他の事例研究からは見出されなかったが，LLL-S の実用上多く発生する問題であるかどうかさらに評価する必要があるだろう．

MPmp，CSys（性質 `EXIT_2` を検証した場合），DDb2 の各事例において，LLL-F は反例と D に関して最も類似している正例を全て発見できたわけではなかった．特に，DDb2 に関しては，発見した正例によってモデルの誤りを指摘することができなかった．既に述べたように，この問題は，LLL-F が，探索すべき正例の接頭辞が性質 `QUIS` の TA $TA(\texttt{QUIS})$ の受理状態に到達することを仮定していることが原因である．一方，LLL-S は，全ての事例について，適切にモデルの誤りを指摘する正例を生成することができた．ただし，先に論じたように，MOvn の事例については反例の形状を修正する必要があった．以上の結果により，LLL-S が導入した距離 D が，誤りを特定するために適切な指標であることが確認されたと考えられる．また，LLL-S はいずれの事例についても LLL-F より多くの実行時間を必要とするが，実用的な時間

で効率的に実行できていると思われる．以上より，LLL-S は効率的に LTS で記述されたモデル上の誤りを特定する手法であると考えられる．

5.4.2 実行性能の評価

続いて，われわれは，性質の TA の規模，反例の接頭辞，および閉路の長さをそれぞれ変化させた場合に，プロトタイプツールの実行時間がどのように変化するかを調査した．調査に用いた事例と性質は，4.4 節と同じ MPmp と `EMG` である．`EMG` の TA の規模を変化させた場合のプロトタイプツールの実行時間の変化を表 5.3 に示す．また，反例の接頭辞長のみを変化させた場合のプロトタイプツールの実行時間と，反例の閉路長のみを変化させた場合のプロトタイプツールの実行時間を図 5.4 にそれぞれ示す．表 5.3 において規模の異なる `EMG` の TA を作成した方法，図 5.4 において正例の接頭辞長および閉路長を増加させた方法は，4.4 節と同様である．また，実行時間の計測法と計測環境は表 5.2 の場合と全く同様である．

表 5.3 の結果より，LLL-F は，規模の大きな TA に関して LLL-S より性能がよいが，LLL-S もまた数十の状態と 1000 以上の遷移を持つ TA に関して実用的な時間で実行可能であることが分かる．また，表 5.3 において，LLL-S は，中規模の TA に対する実行時間と大規模な TA に対する実行時間とがほぼ等しいという結果が得られた．これは，われわれが実装した発見的な最適化処理の効果が原因である．この最適化処理とは，5.3 節で述べた処理のうち，第 1 ステップの最短路探索の結果得られた最短距離が，既に発見された正例の探索で計算された距離 D の最小値よりも大きい場合には，第 2 ステップの探索を行わないという処理である．この処理によって，最短路探索の実行を回数を減らすことができる．

表 5.3. MPmp システムにおける性質 `EMG` を用いたときの Büchi オートマトンの規模に対する LLL-S の実行時間

TA の規模		実行時間
状態数	遷移数	[sec]
4	58	0.16
8	176	0.69
11	216	0.41
14	302	0.66
29	534	0.35
29	650	1.13
35	751	0.97
35	855	1.42
52	931	0.60
50	1190	2.59
64	1327	1.00
64	1471	1.05

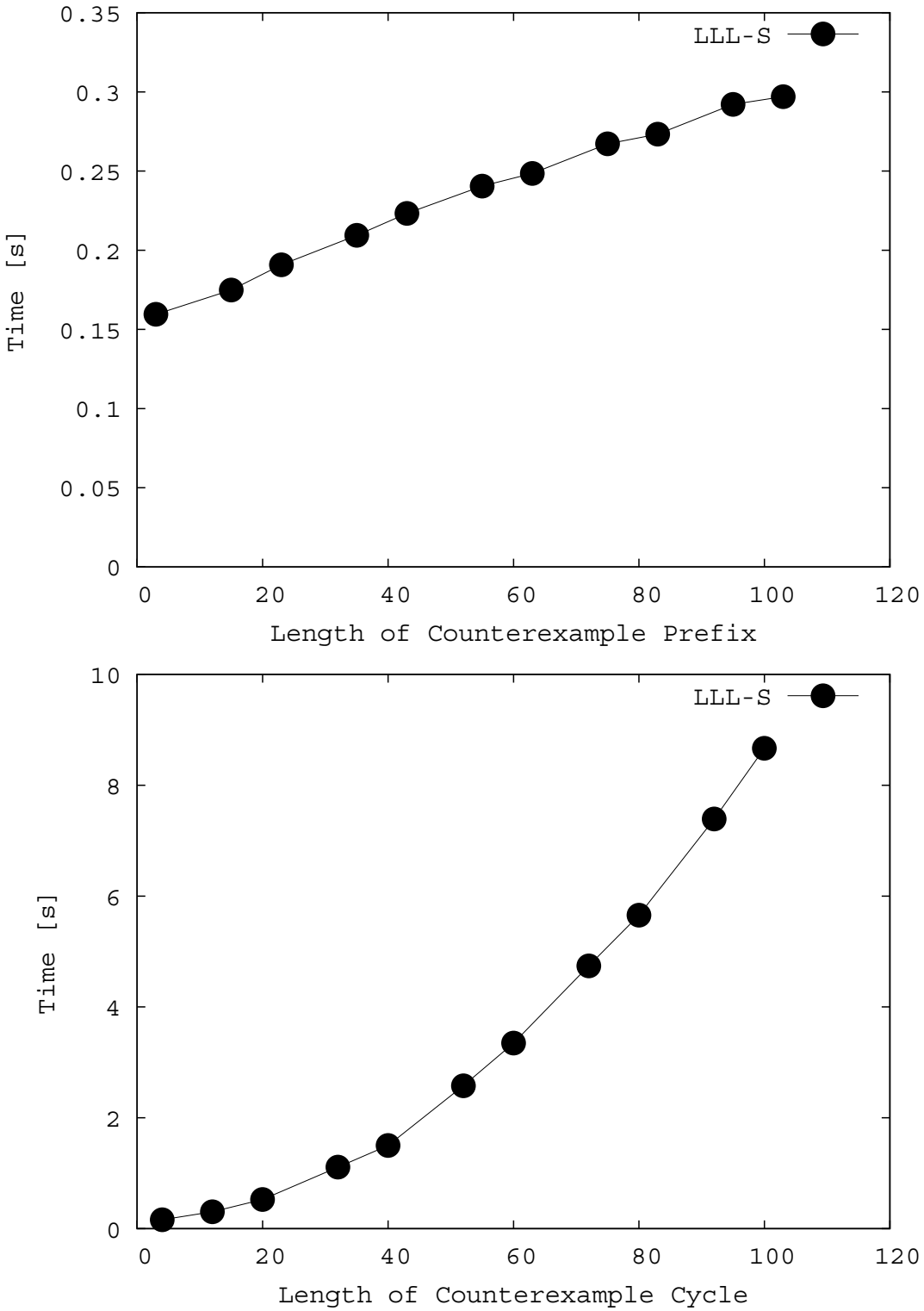


図 5.4. 反例の接頭辞長に対する LLL-S の実行時間（上）と閉路長に対する LLL-S の実行時間（下）

図 5.4 から，反例の閉路を構成する事象列の長さの方が接頭辞長よりも実行時間に与える影響が大きいことが分かる．これは，反例の接頭辞長が主に第 1 ステップの探索の実行時間に対して影響するのに対して，反例の閉路長は第 1 ステップと第 2 ステップの両方の探索に必要な時間に影響を与えるためである．特に，後で詳しく考察するように，第 2 ステップの探索の実行回数は性質の TA の受理状態数と閉路長の積に比例する．したがって，反例の閉路長は，第 2 ステップの探索回数と探索を実行する WTS の積モデルの大きさの両者に影響を与える．

図 5.4 の結果より，LLL-F は LLL-S よりも実行性能がよいが，LLL-S は接頭辞と閉路が長い場合にも，実用的な性能が得られていると考えられる．

5.5 考察と課題

本節では，LLL-S に関連する事項と今後の課題について論じる．

計算時間

LLL-F と同様に，LLL-S において，正例を求めるための時間が計算時間の多くを占めると考えられる．ここでは LLL-S が正例を求めるのに必要な計算時間を見積もる．反例を $\pi = PC^\omega$ とし，モデルの事象の集合を A，そして，性質 ϕ の TA を $TA(\phi) = (S_\phi, A_\phi, \Delta_\phi, u_0, S^a_\phi)$ と書く．ここで，π について $|P| = m$，$|C| = n$ とし，$TA(\phi)$ について $|S_\phi| = v_\phi$，$|S^a_\phi| = v^a_\phi$ とする．

正例の探索時間は，ϕ の TA $TA(\phi)$ と反例モデル W^A_π の積の規模，ならびにその積モデル上の最短路探索の実行時間に支配される．WTS W^A_π は $m+n$ 個の状態と n 個の末尾状態を持つ．したがって，積 $TA(\phi) \bowtie W^A_\pi$ の状態数は $v_\phi(m+n)$ であり，末尾状態数は $v^a_\phi n$ である．よって，5.2.3 節で述べた第 1 ステップの最短路探索に必要な時間は，$O(v_\phi(m+n)\log(v_\phi(m+n)))$ となる．

次に，第 2 ステップの最短路探索の実行に必要な時間を算出する．5.2.4 節の第 2 ステップの最短路探索は，探索の始点が $TA(\phi) \bowtie W^A_\pi$ の全ての末尾状態であるので，$v^a_\phi n$ 回実行される．ここで，第 2 ステップの最短路探索の 1 回の実行時間を見積もる．この探索は $v_\phi n$ 個の状態からなる $TA(\phi) \bowtie W^A_\pi$ の部分グラフ上で実行される．これは，探索の起点である $TA(\phi) \bowtie W^A_\pi$ の末尾状態からは，W^A_π の閉路から構成される部分グラフに相当する部分グラフのみが到達可能であるからである．ゆえに，$TA(\phi) \bowtie W^A_\pi$ の各末尾状態に対して，最短路探索アルゴリズムの実行に必要な時間は $O(v_\phi n \log(v_\phi n))$ である．したがって，5.2.4 節の第 2 ステップの最短路探索の総実行時間は，$O(v^a_\phi v_\phi n^2 \log(v_\phi n))$ である．

最後に，$m \approx n$ と仮定すれば，正例を探索するために必要な時間は，第 2 ステップで行われる全ての最短路探索に必要な時間に支配される．以上より，LLL-S が正例を求めるための時間は $O(v^a_\phi v_\phi n^2 \log(v_\phi n))$ であり，5.4 節で論じたように実用的に実行可能である．

LLL-F と LLL-S の活用法について

前項の計算量についての考察から，LLL-S は LLL-F よりも必要な計算時間が長いことが分かる．これは，方法論の精緻化を行ったことが原因である．したがって，要求分析で扱うような，対象システムの実装の詳細を捨象した規模の小さいモデルについては，LLL-S を用いると高い精度で誤りを高速で特定できる．一方，4.4 節の事例研究から，システムの具体的な設計に近い規模の大きいモデルについては LLL-F の方が高速に誤りを特定できる．ただし，事例研究を行ったシステムについては，LLL-S も実用的な性能で実行可能であるので，その違いが顕著となる事例を調べる必要があるだろう．

距離関数について

本章では，定義 18 で定めた D を無限長軌跡間の類似度の基準に用いた．無限語間の距離には既に先行研究で提案された以下の d_ω がある [94].

$$d_\omega(\sigma_1, \sigma_2) = \inf\{r^{-|w|} | w \text{ は} \sigma_1 \text{と} \sigma_2 \text{との共有接頭辞である } \}$$

ここで，$r > 1$ は実数とする．この距離は，直観的には，二つの無限語が先頭から同一の部分列を長く共有すればするほど，両者の距離が小さくなる（両者が類似している）ように定義されている．本論文ではこの距離を類似度の基準に採用しなかった．これは，この距離が注目しているのは，2 つの軌跡において最初に一致しない事象が現れるまでの両者に共通する事象列の長さのみである．したがって，以下の問題点がある．

- 二つの無限長軌跡の間で複数の事象が異なる場合，最初に不一致となる箇所以外の事象の違いを距離の値に反映させることができない．
- 軌跡の接頭辞と閉路の区別をした距離ではないので，5.1 節の問題点を解決できない．

例 42. 上で挙げた第 2 の問題点について，CSys を用いて例示する．表 5.1 の各正例 $\tau^1_{\texttt{EXIT\_1}}$, $\tau^2_{\texttt{EXIT\_1}}$, $\tau^3_{\texttt{EXIT\_1}}$, $\tau^4_{\texttt{EXIT\_1}}$ に対して，反例 $\pi_{\texttt{EXIT\_1}}$ との距離 d_ω を考える．$\pi_{\texttt{EXIT\_1}}$ と $\tau^1_{\texttt{EXIT\_1}}$ との距離は $\pi_{\texttt{EXIT\_1}}$ と $\tau^3_{\texttt{EXIT\_1}}$ との距離と等しいが，$\pi_{\texttt{EXIT\_1}}$ and $\tau^2_{\texttt{EXIT\_1}}$ との距離よりも大きくなる．したがって，$\tau^1_{\texttt{EXIT\_1}}$ を $\pi_{\texttt{EXIT\_1}}$ と最も近い正例として探索することができないので，モデルの誤りを適切に同定できない．

■

本研究と類似の考え方によって正例を求める研究に，述語抽象を実施したプログラムに対するモデル検査の結果に基づく誤り同定手法がある [19]．この手法が採用しているプログラム実行列間の類似度の指標は，無限長の軌跡を構成する有限長の事象列間の編集距離に相当する距離関数である．すなわち，この手法が用いる距離は無限長軌跡の接頭辞と閉路を区別しない指標である．したがって，この距離もまた上述の d_ω と同様に，5.1 節で示した問題点を解決できない．われわれが提案した距離 D は，反例の接頭辞間の編集距離と閉路間の編集距離の和とする点で上記の研究とは異なる．

反例と探索すべき正例の形状について

ここでは，LLL-F のときと同様に，LLL-S について反例と探索すべき形状が投げ縄型でない場合について考察する．投げ縄型でない形状を持つ反例に対して，距離の近い正例を求めるには，LLL-S では採用する距離関数が重要である．LLL-S が扱う距離 D は，軌跡が無限長投げ縄型であることが前提である（軌跡が有限長の場合は，その閉路が空列であると考える）．したがって，投げ縄型でない軌跡を扱う場合には，異なる距離の尺度を導入しなければならない．そのような距離の候補には，上で挙げた距離 d_ω が考えられる．d_ω は，軌跡の形状に拠らずに距離の算出することが可能なので，汎用性の高い尺度である．しかしながら，例 42 で示したように，モデルの誤りを特定するという観点から望ましい正例を求められない可能性がある．ゆえに，無限長軌跡の形状の特徴を反映させた指標が必要である．距離 D は，投げ縄型軌跡が持つ接頭辞と閉路という特徴を捉えた尺度であることが分かる．

公平性について

LLL-S は反例が示す誤りを発見する手法である．よって，本章の事例研究では考慮していないが，LLL-F と同様に，LLL-S は，活性の検証を行う場合にしばしば用いられる公平な選択を仮定したモデル検査に対しても適用可能である．これは，LLL-S が想定する事項は，反例が図 5.1 で表される投げ縄型構造を持つ無限長軌跡であることのみのためである．

状態空間爆発問題について

モデル検査技術に関する最も重要な研究課題の 1 つは，状態空間爆発問題の克服である．LLL-S は正例を，システムの振る舞いを記述した検証対象のモデルではなく，性質の Büchi オートマトン上で探索する．加えて，Dwyer 等による議論 [39] によると，よく知られた性質のパターンの多くは簡潔な論理式で記述できる．既に述べたように，性質 `EXIT_1` は，性質パターンの 1 つである応答パターンを具体化した論理式である．したがって，LLL-S の探索空間の大きさは，実用的にはモデル検査に必要な探索空間の大きさよりもはるかに小さい．このことは，表 4.2 に示したように，事例研究で扱ったシステムのモデルと性質の TA の規模を比較すると，MOvn の事例以外については正例探索に使う TA の規模の方が小さいことから分かる．ゆえに，LLL-S はモデル検査を実行できる環境で活用することができ，状態空間爆発問題が弱められていると考えられる．

LLL-S が発見する正例の性質の意味に基づく分類

各性質パターン [39] に対して LLL-S が生成する正例の分類について分析することは，得られた誤りの候補と対応する正例から開発者が適切に誤りを指摘しているものを絞り込むために，有用な情報を提供すると考えられる．ここでは，5.4 節の事例研究で扱った性質のモデル検査の結果得られる反例，および，LLL-S によって生成される正例が持つ特徴を考察する．生成される正例は，性質を表す FLTL 式の形式に応じて以下のように分類される．ただし，p, q は異なる論理式（時間演算子は含まない）を表すとする．

安全性 $\mathbf{G}(p \Rightarrow q)$.　この形式の性質は，VALID, SAFE である．反例は，p が成り立つにもかかわらず，そのとき q は成り立たないことを表す軌跡である．この反例に対する正例として，LLL-S は次のいずれかを満たす軌跡を出力した．

- p が成り立たない軌跡.
- p が成り立つときは，q もまた成り立つ軌跡.

事例研究の結果によると，誤りを適切に指摘する正例は常に上記のどちらか一方に属する軌跡のみではなく，問題領域に応じていずれの場合に属する軌跡もありうることが分かった．

安全性 $\mathbf{G}\neg(p \wedge q)$.　この形式の性質は，MUTEX である．この性質に対する反例は，ある時点において p と q が同時に成り立つ軌跡である．LLL-S が生成した正例は，次のいずれかを満たす軌跡だった．

- p が成り立つ間は，q が成り立たない軌跡.
- q が成り立つ間は，p が成り立たない軌跡.

これらは，p と q が共に成り立つことに誤りの要因がある場合の正例であり，少なくともどちらかが偽であることを指摘していると解釈できる．ただし，p と q のいずれにも誤りの原因が含まれていることがありうるので，対象とする問題領域に基づいていずれの正例がモデルの誤りを適切に指摘しているかを開発者が判断する必要がある．一方，他の種類の正例，例えば，p と q が同時に偽となる軌跡を LLL-S が正例として出力することはなかった．距離 D により，編集操作の回数が最小になるような軌跡のみを LLL-S は生成する．ゆえに，事例で扱った性質の場合は，p が成り立たなくなるような編集操作と q が成り立たなくなるような編集操作を適用しなくてはならないため，どちらか一方のみを編集する操作を適用する方が距離が小さくなる．以上が両者を編集する正例が生成されることはなかった理由である．しかしながら，LLL-S によって生成された 2 種類の正例によって，p と q が同時に偽となる正例が表す誤りを求めることができる．

活性 $\mathbf{G}(p \Rightarrow \mathbf{F}q)$.　この形式の性質は，本章で例として用いた EXIT_1 と，事例研究で扱った HEAT, EMG である．また，EXIT_2 はこの形式の 2 つの FLTL 式を連言によって結合した FLTL 式である．この性質の反例は，p が成り立つにもかかわらず，それ以降決して q は成り立たないことを表す軌跡である．事例研究によると，LLL-S は以下の二通りに分類される正例を生成した．

- p が成立しない軌跡.
- p 以降のある時点で q が成り立つ軌跡.

安全性 $\mathbf{G}(p \Rightarrow q)$ の場合と同様に，どちらの場合が誤りを適切に説明しているかは問題領域に依存して定まった．

活性 GFp. この形式の性質は，QUIS であり，その反例はある時点以降常に p が不成立であることを表す軌跡である．得られた正例は，p が無限回繰返し成立することを表す軌跡である（ただし，全ての時点で p が真である必要はない）．

以上の議論と事例研究の結果により，LLL-S は，モデルの誤りを特定するには十分な種類の正例を生成することができたと考えられる．他の形式の FLTL 式（例えば，性質のパターン [39] の各パターンの論理式）に関しても分析することで，LLL-S が発見する正例の特徴づけができると期待される．加えて，このような性質のパターンに応じて正例を分類し，その分類に基づいて正例を開発者に提示することにより，正例が表す意味や正例と反例の関係が開発者にとって理解しやすくなるであろう．したがって，反例が示す性質違反の要因や誤りを説明する適切な正例の絞込みが容易になり，開発者による誤り特定を支援することが可能になると思われる．反例が示す誤りの要因を説明する手法は Beer 等 [10] や Chaki 等 [19] によって LTL のモデル検査を対象として提案されているが，正例が表す意味の説明に活用できるか検討する必要がある．

今後の課題

LLL-S が発見する誤りの候補は，正例からの逸脱要因であり，モデルから反例を除くために削除または改変すべき遷移である．LLL-S が求める最短路は複数ある可能性があるので，この正例と誤りの候補は，一般に複数個得られる．開発者は，これらの誤り候補と対応する正例を用いて，モデルの誤りを特定する必要がある．そこで，特に大規模なモデルにおいて誤り候補や正例が多く生成された場合に，誤りの原因をより適当に説明していると思われる正例を選別あるいは順位付けをする仕組みを導入することが望ましい．一つの方法として，本章で用いた軌跡間の距離関数 D とは異なる基準を新たに導入して，それにしたがって D の値が等しい正例の順位付けを行うことが考えられる．このような基準には，例えば，上で挙げた無限語の距離 d_ω がある．d_ω の昇順あるいは降順に正例を行うことで，距離 D が同一の正例間の評価が可能となるであろう．ただし，この評価は実用性に基づいて行われなければならないので，事例研究の蓄積によって，このような評価が妥当であるかをさらに研究する必要がある．

もう 1 つの課題は，対象領域の知識に基づく正例の生成をすることである．例えば，5.2 節において，LLL-S が求めた正例は，セマフォの排他制御機構を正しく実現したものではない．この理由は，性質 EXIT_1 がセマフォの排他制御機構について直接言及した性質ではないことによる．同様の議論は LLL-F にも成り立つ．4.2 節で求めた正例の閉路は，*p.1.exit* を無限回反復する軌跡である．これは，プロセス p.1 が連続して危険領域から出るという操作を無限回実施することを表現しているが，扱っている対象の並行システム CSys の振る舞いが考慮されていない事象列となっている．以上の点を解決して対象領域に特化した正例を生成するためには，セマフォに関する知識や各プロセスで実行が許される振る舞いについての知識を与えることが望ましい．対象領域を考慮した正例は，開発者にとっても理解しやすく，また，反例との違いをより明確に示すことが期待される．3 章で扱ったように，対象領域の知識を FLTL 式の性質で記述したものを開発者が与えるのが最も簡潔である．本手法では性質は TA として扱わ

れるので，3 章とは異なり，領域知識の記述を安全性に限る必要はなく，いかなる FLTL 式で表された知識をも扱うことが可能である．

最後に，本論文では，編集操作に必要なコストを編集操作の種類を考慮せずに一定値 1 としている．求められる正例は，反例からの編集コストの値に影響されるので，各操作の編集コストを変化させた場合の結果を，実例を元に調査，分析することが必要である．

5.6　関連研究

本節では，本論文で提案した誤り特定法 LLL-F と LLL-S と関連する先行研究を議論する．

本研究と最も密接に関連している研究は，Beer 等が提案した反例の説明を生成する手法である [10]．この方法の目標は，クリプキ構造を対象とする LTL のモデル検査において，検査器が出力した反例を利用者が理解する作業を支援することにある．したがって，Beer 等の方法はわれわれの方法を補完する技術と考えられる．ただし，この研究は,「反例がなぜ誤っているのか」という問いに答えるが，われわれの手法は正例を利用者に提供することで,「反例がどのように修正されればよいか」という問いに答えることができる．また，Beer 等の方法は，モデルの誤りの発見について踏み込んだ議論を行ってはいない．

対象の誤りを特定することを目標とする先行研究は，主にプログラムや論理回路修正の観点から進められている．以下では，まず，特に本研究と関連が深いと思われるプログラムの誤りを発見する方法に関する先行研究を論じる．次に，論理回路を対象とする関連研究について簡単に述べる．

Groce と Visser は，C 言語プログラムを対象として，同一の誤りを引き起こすプログラム状態に達する複数の反例を用いて，プログラムの誤りとその原因を特定する方法を開発した [49]．後に，Groce 等は，1 つの反例を用いたプログラムの誤りの説明を生成，提示する方法を提案している [48]．この研究は，反例と類似の実行列を求め，両者の差分によりプログラム中の誤りを特定するというアイデアを実現している．本章におけるわれわれの誤り特定法も，基本的な考え方はこの研究に依拠する．Griesmayer 等が提案した技術は，プログラムの各変数に対応する述語を導入して，誤りのある変数を探索する手法である [47]．Ball 等は，反例を用いて C 言語プログラム中の誤りを原因ごとに特定する手法を提案している [7]．誤りの原因箇所を得るために，原因箇所を通るプログラムの実行例と通らない例との比較を行う．Chaki 等は，抽象化されていない具体的なプログラムを対象とする Groce 等の研究 [48] を拡張して，検証のために抽象化したプログラムに対する LTL 式の有界モデル検査において，抽象的な誤りの説明を生成する手法を提案した [19]．

ソフトウェアテスティングの分野でもプログラム中の誤りを特定する研究が注目されている．Zeller は，C 言語プログラムの原因–結果鎖を特定する方法を提案している [106]．これは，使われている変数の値に注目して構成したメモリグラフ間の構造的類似度を用いる手法である．その後，Cleve と Zeller はこの研究を補完する方法を提案した [31]．この方法は，誤りの原因となるプログラムの変数値が，不具合現象の発生に至るまでどのように変化していくかを特定する手法である．また，スペクトルに基づく誤り特定手法と呼ばれる一連のアプローチ

が活発に研究されている [61, 88]．これらの方法は，テスティングによってテストに合格する正しい実行列と不合格となる誤った実行列を収集し，一定の基準の下で両者を比較してプログラム中の誤りを特定する技術である．

Yilmaz 等は，Java で記述されたプログラムの誤りを発見する方法を提案している [105]．これは，プログラムの正しい実行と誤った実行が与えられていると仮定し，それらを用いて誤りの含まれるメソッドを特定する．この方法では，正しい実行に現れるメソッドの実行時間と，誤った実行において同じメソッドが実行される時間の違いを調べる．両者の違いが大きい場合，誤りを含む可能性があるとみなし，報告する手法である．しかしながら，モデルの場合，実行時間を測定することに意味があるとは考えにくく，われわれが対象とするモデルの誤りにこの考え方を用いることは困難である．

以上で挙げたプログラムの誤りの発見手法に関する先行研究は，特定の基準に基づいて反例と正例とを比較することで，対象プログラムの誤りを明確にするというアイデアを実現しており，われわれの研究と類似している．Chaki 等の研究 [19] を除くどの研究も軌跡の有限性を前提としている．例えば，文献 [49, 19, 48, 47] の手法が対象とするのは，有界モデル検査器を用いた安全性の検証である．ゆえに，特に活性の場合モデル検査器が提示する無限長の実行列を扱うことができないので，安全性以外の性質に適用できない．したがって，われわれが想定する無限長の軌跡に対して適用することが難しい．加えて，これらの研究の導入している有限長の軌跡に対する距離を，本研究が想定している無限長の軌跡に対して適用したとしても，これらの距離は，無限長の軌跡を構成する接頭辞と閉路を区別しない．したがって，特に，5.1 節で議論した問題点を解決できない．

一方，Chaki 等は，活性の検証に対する抽象化したプログラムの誤り特定問題に取組んでいる [19]．この方法では，5.1 節で述べたように，有界モデル検査の考え方を用いて，無限長実行列間の距離を有限長文字列間の編集距離によって定義している．そして，この距離に基づいて，反例と類似した正例を求める問題を充足可能性問題に帰着させている．しかし，一般に充足可能性問題は NP 完全問題である．LLL-F と LLL-S は共に古典的な有向グラフ上の最短路探索問題を解くことで正例を求めている．よって，4.5 節と 5.5 節とにそれぞれ示したように，両者は Chaki 等の手法に比べてはるかに効率的である．

文献 [49, 48, 7, 106, 61, 88, 31, 105] の手法では，対象プログラムに性質や仕様を満たす実行列が存在することが前提である．つまり，プログラムの全ての実行列が性質を満たさない場合は，適用できない方法である．LLL-F と LLL-S では正例の集合は性質の TA で与えられる．性質の TA は，性質を表す FLTL 式から自動的に構成可能なので [45]，ユーザが正例の集合を入力する必要がない．

Killian 等は，C++ 言語のプログラム用モデル検査器 MaceMC と対話型デバッガ MDB を開発した [62]．MaceMC は活性の検証を実行する機能を持つ．対象プログラムが活性を満たさない場合には，MaceMC は有限長の実行列の反例を返し，検証した性質が決して満たされることのないプログラムの状態への遷移を特定する．開発者が誤りを理解できるようにするために，MDB はこの情報を利用して，反例と共通の接頭辞を持つ正例を求め両者の比較結果を提供する．この点で Killian 等の研究はわれわれの手法と類似しているが，応答パターンと

呼ばれる仕様パターン [39] の中の特別の種類の論理式のみを対象としている．一方，これまで論じてきたように，LLL-F と LLL-S は活性を含む全ての FLTL 式に適用可能な方法である．

Wang 等の方法は，プログラムの各文の最弱事前条件を生成して，反例の発生原因を求める手法である [103]．この方法は正例を必要としないが，多くの先行研究と同様に反例の長さの有限性を前提としている．

論理回路の誤り特定を目標とした研究の多くは，回路の結線法などの論理回路の特性に注目した方法で，抽象的なモデルに直接適用するのは困難である [93, 41]．

LLL-F と技術的に関連する研究については，LLL-F と同様の考え方により，グラフを用いて正規言語の編集距離を求める方法が既に提案されている [80, 63]．われわれはこれらの手法とは独立に LLL-F を開発したが，LLL-F の対象は正規言語ではなく，FLTL の論理式を満たす軌跡の集合と反例との編集距離である．これらの軌跡は一般に，無限長の文字列である．

5.7　まとめ

本章では，LLL-F の精緻化を行った．まず，無限長の軌跡間の類似度の指標となる距離を編集距離に基づいて定義した．そして，その距離が反例と最も近い正例を求めることでモデルの誤りを自動的に特定する手法 LLL-S を提案した．4 章で提案した LLL-F と同様に，LLL-S は反例およびモデルに基づく手法である．また，LLL-S は，モデルの積を求めるモデル合成技術と古典的なグラフ探索アルゴリズム以外は特別な技術を必要としない手法なので，自動化が可能である．特に，モデル検査器が出力する無限長反例と編集距離に基づいた距離が最小となる無限長正例を発見できる．この正例は反例を性質を満たすように修正した軌跡であるとみなせる．したがって，この正例と反例とを比較することによって，開発者はモデルの誤りのみならず，誤りの原因を容易に発見できるようになる．われわれは Java 言語で LLL-S のプロトタイプを実装した．このプロタイプを用いて事例研究を実施することにより，実用的な時間で実行可能な有効な手法であることを確認した．

第 6 章

おわりに

本章では，これまで述べてきた論点について整理する（6.1 節）．本論文で提案したモデル修正法と誤り特定手法は反例を活用した手法であり，また，前提としている検証手法，ならびにモデルと性質の記法が同一である．提案手法の重要な特徴は，これらの方法はいずれも安全性や活性を含む FLTL 式で記述可能な全ての性質に適用可能な方法なことである．

最後に，今後の研究課題として，以下の点を議論する．

- モデルや性質の他の記法に対する本論文で提案したモデル修正技術の適用可能性（6.2 節）．
- モデル修正法と誤り特定手法（LLL-F, LLL-S）は互いを補う技術であることから，両者を統合したモデル修正技術の実現（6.3 節）．
- モデルの修正，変更を前提とした効率的なモデル検査技術の開発（6.4 節）．

6.1 まとめ

本論文の大きな目標は，信頼性の高いソフトウェアの開発を支援する技術を開発することにある．

ソフトウェアシステム開発プロジェクトでは，プログラムを実装する以前の要求分析，設計工程において，作るべきソフトウェアの構成や動作を予め分析，設計することで，そのあるべき姿を明確にする作業を実施する．要求分析や設計には，対象システムを抽象化した記述であるモデルがしばしば用いられる．システムの振る舞いは，ラベル付き遷移系に代表される状態遷移機械などの動的なモデルで表現される．特に，現在開発されるシステムの多くは，複数のプロセスが相互に対話をしながら並行動作する並行システムである．このようなシステムの振る舞いは複雑になりがちであり，モデルを用いたシステムの振る舞いの正確な分析が必要である．システムの信頼性を高めるためには，このモデルが実現したい機能や望ましい振る舞いを表現していることを保証するための検証が重要である．

この検証作業を形式的かつ自動的に実行することを可能にするのがモデル検査技術である．モデル検査に関しては，様々な表現能力を持つモデルや性質に対する自動検証技法の確立や，

モデルの抽象化，半順序簡約などの検査の効率化の研究がこれまで活発に進められており，ソフトウェアツールの開発による実用化にも成功している．

モデル検査技術についてシステムの信頼性の観点から考えると，検証の結果システムが所望の性質を満たさないと判定された場合が重要である．この場合，モデル，ひいてはその結果を基にして構築するシステムが誤りを含むことを意味するので，システムの信頼性を脅かすことになってしまう．よって，特に要求分析，設計工程においては，誤りを含むモデルを性質を満たす正しいモデルに修正する必要がある．検証結果に注目した先行研究は，反例の提示法の提案など一部に限られており，正しいモデルを得る方法の開発を目的とする研究は少ない．よって，システムの開発者はモデルを手作業で修正して正しいモデルを求めなければならない，という問題がある．また，モデルやプログラムの誤りを指摘する研究の多くは安全性とよばれる特定の性質に注目にしたものである．開発対象が並行システムのときは，しばしば大規模で複雑な振る舞いを示し，開発者の手作業による修正作業は困難であるため，修正作業を支援する技術の確立が望まれる．

本論文では，以上で論じたモデルの修正作業を計算機によって支援するために，以下の 2 種類の課題を解決する方法論を提案することにより，高信頼性システムを実現するための検証工程の支援に貢献した．

- **反復的モデル修正法**：モデルが性質を満たさないと判定された場合に，反例を用いて性質を満たすようにモデルを修正する半自動的な手法．
- **モデル上の誤り特定法 (LLL-F, LLL-S)**：モデルに含まれる誤りを反例を用いて自動的に発見する方法．

第 1 のモデル修正法は，修正モデルの候補を開発者に提示することで，開発者によるモデルの修正作業を支援をすることを目指している．しかしながら，モデルに含まれる誤りの要因を発見する作業は，開発者が行わなければならないという課題がある．第 2 の誤り特定法はその誤りを自動的に発見することが目標である．

両者は以下に論じるように，前提とする条件や手法の特徴に共通性がある．それゆえ，2 種類の方法は，共に同一のモデル検査技術の枠組みのもとで実施することができる．

- FLTL 式で記述された性質を，LTS 上でモデル検査を行う場合を想定している．さらに，Giannakopoulou らによる Büchi オートマトンを用いた検査法 [45] を前提とする．この検査手法の利点は，FLTL 式を満たす軌跡の集合を受理する Büchi オートマトンを LTS で表現することができる点にある．われわれが研究したモデル修正法，LLL-F, LLL-S はいずれもこの利点を活用している．具体的に述べると，モデル修正法において，領域知識の基盤モデルを構築するのに Büchi オートマトンを用いている．すなわち，領域知識の否定を表す FLTL 式のテスタオートマトンを構成し，それから基盤モデルである 4VTS を作成する．誤り特定法に関しては，性質を満たす軌跡の集合が性質の Büchi オートマトンで与えられる．
- いずれも反例を利用する方法である．多くのモデル検査器では，モデルが性質を満たさ

ない場合には反例を返す．そして，反例は性質の否定を満たす軌跡で与えられるので，性質がなぜ満たされないのかを開発者が知るために有用である．よって，性質違反を引き起こすモデルの誤りを探り，そしてその誤りを正すために，反例を用いるのは自然であるとわれわれは考える．加えて，誤り特定法が注目しているのは，反例もまた開発者による解釈が必要であるという点である．つまり，反例から誤りを知るためには，反例を開発者が理解しなければならない．そのためには，モデルが表すシステムの振る舞いを開発者が知っている必要がある．われわれが開発した誤り特定法は，反例中の性質違反の要因となる箇所を，反例と類似した正例を求めることにより抽出するので，その点を支援をすることができる．

- 提案手法では，活用する反例は無限長で，有限長の接頭辞に無限回反復する有限長閉路を付加した投げ縄型の形状の軌跡を想定している．ただし，接頭辞と閉路を構成する事象列には，接頭辞や閉路を構成する語に繰返し構造が現れないもののみを対象としている．この点については既に説明した通り，Büchi オートマトンの受理条件に基づく制約ではなく，オートマトンに基づく方法を実現したモデル検査器の多くが探索する反例の形状に基づく制約である．モデル検査器が利用する反例探索法は深さ優先探索に基づく方法であり，投げ縄型の反例を大規模なモデルに対しても効率的に探索することができる．よって，本論文での反例の形状にわれわれが設けた仮定は，実用的に妥当であると考えられる．
- FLTL 式の形式に依存せず，安全性，活性共に適用できる方法である．また，公平な選択という制約を仮定した場合の活性の検証にも対応可能である．先行研究の多くは，特定の形式で書かれた性質のみを対象としているので，提案手法の先行研究に対する大きな利点は，活性ならびに公平な選択を扱うことができる点である．活性は，モデルが望ましい振る舞いを実現しているという性質を述べることができる．しばしば用いられる活性に，2.1 節, 2.3 節に紹介した `EXIT_1` のような $\mathbf{G}(p \Rightarrow \mathbf{F}q)$ のパターンで表される応答性 [39] がある．これは，直観的には，述語 p が成り立つならばいつか（p と同時あるいはそれ以降のある時点で）述語 q が成り立つことを表す．p, q をそれぞれシステムへの入力，出力についての記述とすれば，入力に対する所望の出力を保証する性質として実用的にもよく用いられる．本論文で提案した方法は，応答性に対しても適用できるだけでなく，文献 [39] で分類されている他の性質のパターンにも適用することができるので，実用的な手法である．

6.2　他のモデル検査手法への応用

本論文では，モデルとして LTS，性質として FLTL，検査法として Büchi オートマトンを用いる方法を想定している．しかしながら，検証対象のモデルの記述にはクリプキ構造なども知られている．また，従来モデル検査技術で扱われている性質の記述法には，LTL 以外に CTL や CTL*が提案されており，そのような記法に対応した検証技術が開発されている．本節で

は，われわれが提案したモデル修正技術の他のモデル検査手法への適用可能性を考察する．

クリプキ構造で記述されたモデルへの応用

多くのモデル検査技術が対象とするモデルの記法は，状態遷移機械の中でもクリプキ構造と呼ばれるものである．ここでは，LTS からクリプキ構造にモデルの記法が変更した場合の提案した方法の適用可能性について考察する．

2 章で述べた通り，クリプキ構造は LTS と異なり，遷移ではなく状態にラベルが付加される．また，状態に付加されるラベルがその状態で成り立つ命題の集合であるので，両者の違いはラベルの付け方にあるといえる．

クリプキ構造を対象とする LTL のモデル検査法には，Büchi オートマトンを用いた方法が提案されている [101]．したがって，モデルの記法としてクリプキ構造を用いた場合にも，正例の集合に性質の Büchi オートマトンを用いることができる．さらに，既存のクリプキ構造に対するモデル検査技術によって得られる反例は，モデルの状態に付加されたラベルの集合の無限列であり，その形状は本論文で仮定した無限回の閉路とその閉路への有限の接頭辞からなる投げ縄型である．また，検証する性質が安全性のときには，反例が有限列となる．ゆえに，反例を事象列とみなすことで，本論文で扱ったような事象で構成された軌跡の場合と同様に扱うことができると考えられる．以上より，本論文で提案した手法のアイデアを拡張したモデル修正法，および誤り特定法を開発することが可能であろう．ただし，検討すべき課題の 1 つには，モデル修正法や誤り特定法において必要な，反例から構成するモデルの構築法がある．特に誤り特定法の場合，本論文では編集操作を基にモデルを構成するので，命題の集合に対する編集操作を適切に定義する必要があると思われる．

FLTL の場合は公平な選択を考えたが，LTL のモデル検査については，より一般的に定義された公平性を考えることが多い [25, 5]．公平性は，モデル上の特定の経路をモデル検査の対象から排除するように，到達すべき状態や遷移に関して定める制約である．公平な選択も，公平性の一種である．提案したモデル修正法と誤り特定法は，いずれも反例に基づくので，公平な選択の場合と同様に，公平性を考慮した場合も適用可能であると考えられる．特に，モデル修正法については，LTL 式で表現された公平性を領域知識とすることにより，適切な修正モデルを得るための有効な手掛かりとすることができるであろう．

CTL, CTL*の検査への応用

モデル検査で性質を記述するためにしばしば用いられる記法には，LTL 以外に 2 章で紹介した CTL や CTL*がある．CTL については，記号的な検査法の研究が従来進められてきた．2 章で紹介した通り，記号的な方法は，検査対象のモデル上を反復的に探索して，性質を満たす状態の集合を求める方法である．この方法は，オートマトンを用いる方法と異なり，性質の論理式をオートマトンに変換する手続きが必要ない．しかしながら，われわれが開発したモデル修正法と誤り特定法は，与えられた論理式と等価なオートマトンを自動的に構築できることが前提である．したがって，記号的な方法を元にしてモデルの修正を行う場合，領域知識を表す基盤モデルを何らかの方法で与える必要がある．同様に，誤り特定法についても，正例の集

合である性質を満たす軌跡の集合を，本論文の提案手法とは異なる方法で与えなくてはならない．CTL*に関しては，2 章で論じたとおり，CTL と LTL の検査法を組み合せた検査法が提案されているが，性質を満たす軌跡の集合を予め求めることは一般に難しいと思われる．

以上のように，CTL, CTL*の検査に対して提案した方法のアイデアを直接用いることは難しいが，以下のような発見的な解法が考えられる．

- モデル修正法について，基盤モデルの構築法の工夫が必要である．そのために，対象領域の知識を論理式ではなく，それ以外の方法で与えることが考えられる．そのような方法には，例えば，開発者が，オートマトンにより領域知識を表現するという方法があるだろう．これは，Ziller と Schneider が提案した性質を満たすモデルの抽出法 [108] と同様の考え方である．また，領域知識を必ずしも CTL 式で与える必要はなく，LTL 式によって与えることが考えられる．LTL の検証は CTL の検証技術によって実現できる [28]．この手法は，検査を行う LTL 式からタブローと呼ばれるモデルを構成し，そのタブローと検証対象モデルとの積モデル上で記号的に検査を行う．そこで，このタブローを領域知識を表現する基盤モデルとして活用することが 1 つの方法となると思われる．
- 誤り特定法については，正例の集合を用いるのではなく，対象のモデル上で正例を求めて，反例と正例の違いを調べるという方法が考えられる．これは，Groce 等の方法 [48] や Chaki 等の方法 [19] と類似したアイデアであるが，5.6 節で述べた通り，モデルに性質を満たす軌跡が存在しなければ適用できない．よって，正例の集合に相当するモデルを，対象モデルから求める方法を検討することが課題である．

6.3 モデル修正技術の統合

3 章のモデル修正法は，開発者との対話的なプロセスであるが，反例が示唆する誤りは開発者が発見しなければならないという欠点がある．続く章で述べたモデルの誤り特定手法は反例から誤りを発見する手法なので，この問題点を解決することができる．したがって，両者を組み合せることによって，統合化されたモデル修正技術を構築することが期待される．この技術が実現されると，モデル検査の結果に基づくモデルの修正を開発者がより容易に実施することが可能であると考えられる．ここでは，このような統合的なモデル修正技術を実現するために残された研究課題について議論する．

6.1 節で論じたように，われわれが提案したモデル修正法と誤り特定法はどちらも反例を活用した手法であり，また，検証手法，ならびにモデルと性質の記法が同一という前提がある．ゆえに，以上の前提を元にした両者の統合は可能であると考えられる．そのために，3 章で述べたモデル修正プロセスを誤り特定工程を含むように拡張することが必要である．以下に拡張した修正プロセスを示す．

1. 領域知識を表す安全性の集合から基盤モデルの 4VTS を構築する．

2. モデル検査を実行して，対象モデルが性質を満たすかどうかを検証する．性質を満たさないならば，反例を得る．
3. 誤り特定法を実行して，その結果から，誤りを適切に説明した正例と反例が示す誤りを開発者が選択する．
4. 反例から 4VTS を構成する．ただし，作成したモデルの可能遷移あるいは不確実遷移のうち，上のステップで得られた反例が示す誤りを禁止遷移に変更する．禁止した誤りに代わる正しい遷移は，その確度も含めて誤り特定法が出力した正例を手掛かりとして決定する．
5. 基盤モデルの 4VTS と反例から構成した 4VTS とを合併する．
6. 3.3.3 節あるいは 3.5.2 節の方法に従って修正した LTS を求める．
7. 得られた LTS から反例が存在しなくなる，あるいは，LTS が事前に定義された基準を満たせば終了する．そうでない場合は，ステップ 2 へ戻る．

上記のように変更したとき，3.3.3 節の方法を利用することで発生する問題点は，誤り特定法が発見したモデルに含まれる誤りを，修正法において直接活用することができない点である．これは，モデル修正法が修正対象のモデルを直接活用する手法ではないため，検証対象の性質以外の対象領域の知識を扱わないことが原因である．しかしながら，3.5.2 節の方法は，修正対象モデルの各状態との類似度を領域知識と反例から構築したモデルを開発者に提供する．したがって，誤り特定法が指し示す誤りを構成した 4VTS 上でも同様に開発者に指示することができると思われる．また，上記ステップ 4 に示したように，反例から構築する 4VTS に求めた正例についての情報を付加することで，修正後の LTS が持つべき性質を満たす軌跡に関する情報を開発者に提供できると思われる．よって，対象領域の知識を反映した正例の生成法の開発が重要な研究課題である．

反例のある事象を異なる事象に置換した正例が得られた場合を考える．まず，この反例中の置換が適用される不整合事象を予め禁止遷移として 4VTS でモデル化する．代わりに，性質に出現し，置換操作が適用される事象により真理値が変化する流動の開始，あるいは終了事象の集合を可能遷移として表す．このようにして生成した反例の 4VTS は，合併後のモデルにおいても禁止されるべき遷移を開発者に明示することができる．したがって，開発者による遷移を選択する修正作業の負荷の軽減を図ることが可能であろう．ただし，ここで述べた反例のモデル化技法が一般に有効であるかどうかの議論，挿入，削除が行われた場合のモデル化する手段の検討，ならびに複数の編集操作が適用された場合のモデル化の方法の検討など，研究すべき課題は多くある．最後に，この新たな修正プロセスの有効性は，今後の事例研究により評価されることが必要である．

以上のように統合したモデル修正法もまた，モデル検査に引き続いて，その結果を用いて実行することが可能である．それゆえ，モデル修正法の統一化が実現された後の更なる方向性として，モデル修正技術とモデル検査技術の統合が展望されよう．それにより，単なるモデルの正しさの検証だけでなく，モデルの正しさを保証する技術として確立することができると思われる．ここで述べた技術をモデル保証技術とよぶことにする．つまり，以下の手順を実施する

ことにより，対象のモデルの検証，修正を通じて性質を満たす正しいモデルを得ることができる技術が実現されることが期待できる．

1. 対象モデル上で既知の性質のモデル検査を実行する．モデルが性質を満たすならば，真を出力して終了する．モデルが性質を満たさないならば，反例を生成してステップ 2 へ進む．
2. 領域知識を表す論理式の集合を開発者が入力する．
3. モデル，性質，反例，領域知識を用いて，開発者と対話的に統合したモデル修正法を実行する．
4. 修正後のモデルを出力する．

モデル保証技術は，ソフトウェア開発の上流工程において重要な役割を果しうる．すなわち，要求分析，設計時において，システムの振る舞い表現するモデル作成を計算機を用いて支援するのに有効である．この結果，開発者がシステムの望ましい振る舞いを正しく反映したモデルを開発者が構成するのに貢献するので，このモデルに基づいて構築されたシステムの信頼性の向上に寄与するであろう．モデルに複数の誤りがある場合，モデル検査が提供する反例が全ての誤りを表すとは限らない．しかしながら，モデル保証技術は，反復的に実施して正しいモデルを得るので，このような場合にも有効だろう．

6.4　増進的モデル検査技術

提案したモデルの修正法において，モデルの変更，修正を実施するたびにその正しさを確認するために，開発者は検証作業を行う必要がある．既存のモデル検査技術は，モデルである有向グラフ全体の探索アルゴリズムとして実現されているので，開発者は 1 回の修正の規模に関わらずモデル全体の検証作業を，修正作業が終了するまで繰返し実行しなければならない．作成するモデルの規模が大きい場合には，性質を満たさない場合に行うモデルの修正作業は，通常モデルの一部の遷移や状態の変更 (削除や追加) にとどまり，モデル全体に及ぶ大規模な修正を行うことは少ないと考えられる．したがって，1 回の変更によって検証作業に影響を与えることのない箇所がモデルの大部分を占めるため，そのような部分に不必要な再検査を実施しなくてはならず，効率的に検証を実施することができない．よって，モデルの修正箇所と適用とされた修正操作によって影響される範囲に限定した検証を実施することで，モデル全体の検証を行うよりも効率的な検証作業の実現が可能であると予想される．

そこで，モデルの修正プロセスが繰返し実施されることを前提とした効率的な検査手法を開発することが望ましい．この技術が確立されることで，われわれが提案した修正法のモデル検査工程の高速化が達成されるので，ソフトウェアシステムに求められる高い信頼性を効率的に達成することに貢献できる．加えて，この研究の成果は，前節で論じたモデル検査技術とモデル修正技術との統合した技術の実用化に貢献することができる．ここでは，修正対象のモデルに対しては，既存のモデル検査技術を用いてモデルが性質を満たしていないかどうか検証を行い，その結果に基づいてモデルが修正されることを前提とする．その際決定された検証結果の

情報と，対象モデルに対する修正あるいは変更の情報とを用いることで，変更によって影響される範囲を集中的に検証することができるならば，変更後のモデルに対する検証は，従来のモデル全体の探索によって検証するよりも小さい状態空間を探索することで実現できることが期待される．

モデルの修正を前提としたモデル検査技術に関しては，Sokolsky と Smolka によって非交替性様相 μ 計算で書かれた性質の増進的モデル検査アルゴリズムが提案されている [91]．この方法は，状態遷移機械で記述されたモデルが性質を満たさない場合に，この変更前のモデルと，遷移の削除あるいは追加操作によるモデルの変更履歴を用いることで，変更後のモデルの効率的な検証を可能にする．しかしながら，非交替性様相 μ 計算式は全ての LTL 式を表現可能ではない [32] ので，LTL 式の検証に適用することができない．本研究は，LTL 全体の検証を対象としている点に特色があるので，この手法を適用するためには，LTL 式全体を扱うように拡張しなくてはならない．

Cohen と Kupferman は，LTL 式の増進的モデル検査手法について議論しているが [32]，実例を用いた手法の評価が行われていないので，その実用性は明らかにされていない．

Henzinger 等は，ソフトウェアモデル検査器 BLAST [11] を拡張した増進的モデル検査技術である極端モデル検査法を提案した [51]．この方法は，抽象化技術である CEGAR および述語抽象技術を拡張した手法である．2 節で述べたように，抽象化技術とは，モデルの複数の状態や遷移を 1 つの状態や遷移に統合することで，検査すべきモデルの規模を圧縮する技術のことである．極端モデル検査法において，変更前のプログラムから構成される抽象化モデルと変更後のプログラム制御フローグラフを比較し，両者に不整合がなければ変更前の検証結果を変更後の検証結果とする．モデル間の整合性は，有向グラフの模倣関係に基づいて定義される．不整合が存在するならば，抽象化モデルに対して述語の洗練を行う．極端モデル検査法は述語抽象を有効に活用しているが，BLAST が前提とする安全性の検証のみに適用される方法である．

モデルの変更を前提とした増進的モデル検査技術を確立するために解決すべき技術的な課題は，以下の通りである．

1. 変更箇所に着目したモデル検査を実現するためには，前段階で作成したモデルにおいて，性質を満たす箇所とそうでない箇所を識別できなくてはならない．これは，LTL よりも表現力のある様相 μ 計算のモデル検査手法を用いるならば可能な方法である．しかし，従来の LTL 式で書かれた性質に対するモデル検査技術は，モデルが性質を満たすことを証明するのではなく，その反例を探索するという方法を用いている．そして，反例を発見できない場合にモデルが性質を満たすと判定する．したがって，従来の LTL 式のモデル検査技術では，モデル内の性質を満たす箇所と満たさない箇所を判別することができない．この点を解決するには，LTL 式の検証を，様相 μ 計算を用いた検証問題に帰着させる，あるいは，本論文で提案したモデル誤り特定技術を用いて，性質を満たさない箇所や満たす箇所を発見するという方法が考えられるが，性能など実用性の観点を含めて慎重に検討されなければならない．

2. 第 1 の課題が解決された場合に，性質が成り立つかどうかが変更の影響を受けないモデルの部分グラフの検証を効率的に省略する手法の確立である．これは，モデル検査手続きの高性能化の鍵となる技術である．この点を解決するためには，極端モデル検査法と同様に，モデル検査技術の高速化の研究として近年注目されている抽象化技術が有効であると考えられる．まず，圧縮したモデルで検査を実施し，その後，必要に応じてそのモデルの一部を元の状態，遷移集合に復元することで検証を行う．本研究でもこの考え方に基づき，前段階のモデルにおいて，変更の影響がない状態，遷移を統合することにより，高速化を図ることが可能であると考えられる．
3. 以上の検証技術をソフトウェアツールとして実装し，実用化を図ることである．そして，実用性の評価のために，事例を用いて，実装した検証技術の実行時間を他のモデル検査技法と比較することが必要である．

謝辞

本研究を実施するにあたり，指導教官である東京大学大学院 総合文化研究科 教授，玉井哲雄先生に深く感謝申し上げます．玉井先生には，修士課程に引き続いて御指導いただきました．玉井先生は，お忙しい中でも筆者に研究上の御助言をくださいました．

また，東京大学大学院 総合文化研究科 准教授，増原英彦先生にも，玉井増原研究室合同ゼミで様々なご指摘をいただきました．先生方による御指導，御教授がなければ，本論文をまとめることはできませんでした．大いに感謝申し上げます．

玉井研究室，増原研究室の方々にも，ゼミ発表や研究室内外での議論で，多くの御助言や御指摘をいただきました．誠にありがとうございました．

学会での発表に際しては，国立情報学研究所 アーキテクチャ科学研究系 教授，中島震先生に大変お世話になりました．中島先生には，筆者の研究について話を聞いていただいただけでなく，御質問，御助言もいただきました．同じく筆者が参加した学会において，九州大学大学院 システム情報科学研究院 情報知能工学部門 教授，鵜林尚靖先生にお世話になりました．お二人に感謝申し上げます．

博士論文審査会では，玉井先生，増原先生，中島先生，さらに，東京大学大学院 情報理工学系研究科 教授 萩谷昌己先生，東京大学大学院 総合文化研究科 教授 山口和紀先生から大変有意義な御助言をいただきました．ありがとうございました．

最後に，筆者の両親や学外の友人，知人に感謝いたします．彼らの精神的な支援や励ましがなければ，本論文を完成させることはできませんでした．ありがとうございました．

参考文献

[1] Aldebaran manual page:
http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html#sect6.

[2] GNU Linear Programming Kit: http://www.gnu.org/software/glpk/.

[3] P. E. Ammann, P. E. Black, and W. Majurski. Using model checking to generate tests from specifications. In *Proceedings of 2nd IEEE International Conference on Formal Engineering Methods*, pp. 46–54, 1998.

[4] A. Arcuri. On the automation of fixing software bugs. In *Proceedings of the 30th International Conference on Software Engineering, Dcotoral Symposium.*, pp. 1003–1006, 2008.

[5] C. Baier and J.-P. Katoen. *Principles of model checking.* MIT Press, 2008.

[6] T. Ball, B. Cook, V. Levin, and S. K. Rajamani. SLAM and Static Driver Verifier: technology transfer of formal methods inside Microsoft. In *IFM 2004, LNCS*, Vol. 2999, pp. 1–20, 2004.

[7] T. Ball, M. Naik, and S. K. Rajamani. From symptom to cause: Localizing errors in counterexample traces. In *Proceedings of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 97–105, 2003.

[8] T. Ball and S. K. Rajamani. Automatically validating temporal safety properties of interfaces. In *Proceedings of the 8th International SPIN Workshop on Model checking of Software, Lecture Notes In Computer Science*, Vol. 2057, pp. 103–122, 2001.

[9] K. Beck. *eXtreme programming eXplained.* Addison-Wesley, 2000.

[10] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining counterexamples using causality. In *Proceedings of the 21st International Conference on Computer Aided Verification, Lecture Notes In Computer Science*, Vol. 5643, pp. 94–108, 2009.

[11] Dirk Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker BLAST: Applications to software engineering. *International Journal on Software Tools for Technology Transfer*, Vol. 9, No. 5, pp. 505–525, 2007.

[12] A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. *Elec-*

tronic Notes in Theoretical Computer Science, Vol. 66, No. 3, pp. 160–176, 2002.

[13] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, Lecture Notes In Computer Science*, Vol. 1579, pp. 193–207, 1999.

[14] F. P. J. Brooks. *The mythical man-month: Essays on software engineering.* Addison-Wesley, Anniversary edition, 1995.

[15] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proceedings of the 11th International Conference on Computer Aided Verification, LNCS*, Vol. 1633, pp. 274–287, 1999.

[16] G. Bruns and P. Godefroid. Generalized model checking: reasoning about partial state spaces. In *Proceedings of the 11th International Conference on Concurrency Theory, Lecture Notes In Computer Science*, Vol. 1877, pp. 168–182, 2000.

[17] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, Vol. 18, No. 8, pp. 689–694, 1997.

[18] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, Vol. 19, No. 3–4, pp. 255–259, 1998.

[19] S. Chaki, A. Groce, and O. Strichman. Explaining abstract counterexamples. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 73–82, 2004.

[20] S. Chandra, P. Godefroid, and C. Palm. Software model checking in practice: An industrial case study. In *Proceedings of the 24th International Conference on Software Engineering*, pp. 431–441, 2002.

[21] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No. 4, pp. 371–408, 2003.

[22] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification, Lecture Notes In Computer Science*, Vol. 2404, pp. 359–364, 2002.

[23] E. M. Clarke. The birth of model checking. In *25 Years of Model checking, Lecture Notes In Computer Science*, Vol. 5000, pp. 1–26, 2008.

[24] E. M. J. Clarke, O. Grumberg, S. Jha, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, Vol. 50, No. 5, pp. 752–794, 2003.

[25] E. M. J. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* MIT Press, 1999.

[26] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop, Lecture Notes*

In Computer Science, Vol. 131, pp. 52–71, 1981.

[27] E. M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: Algorithmic verification and debugging. *Communications of the ACM*, Vol. 52, No. 11, pp. 74–84, 2009.

[28] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design*, Vol. 10, No. 1, pp. 47–71, 1997.

[29] E. M. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pp. 19–29, 2002.

[30] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science*, Vol. 2988, pp. 168–176, 2004.

[31] H. Cleve and A. Zeller. Locating causes of program failures. In *Proceedings of the 27th International Conference on Software Engineering*, pp. 342–351, 2005.

[32] G. Cohen and O. Kupferman. Incremental LTL model checking. *1st Int. Workshop on Semantics and Verification of Hardware and Software Systems*, 2003.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2nd edition, 2002.

[34] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, Vol. 1, No. 2–3, pp. 275 – 288, 1992.

[35] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 2nd edition, 2002.

[36] E. W. Dijkstra. A note on two problems in connection with graphs. In *Numerische Mathematik*, pp. 269–271, 1959.

[37] Y. Ding and Y. Zhang. A logic approach for ltl system modification. In *Proceedings of the 15th International Symposium on Methodologies for Intelligent Systems, Lecture Notes in Computer Science*, Vol. 3488, pp. 435–444, 2005.

[38] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel. MTSA: The modal transition system analyser. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 475–476, 2008.

[39] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering*, pp. 411–420, 1999.

[40] M. Fernández and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, Vol. 22, No. 6-7, pp. 753–758, 2001.

[41] G. Fey, S. Staber, R. Bloem, and R. Drechsler. Automatic fault localization for

property checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 2, No. 6, pp. 1138–1149, 2008.

[42] M. Fowler. *UML distilled: A brief guide to the standard object modeling language.* Addison-Wesley, 3rd edition, 2004.

[43] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pp. 3–18, 1995.

[44] D. Giannakopoulou and F. Lerda. From states to transitions: Improving translation of ltl formulae to buchi automata. In *Proceedings of the 22nd IFIP WG 6.1 International Conference Houston on Formal Techniques for Networked and Distributed Systems, Lecture Notes In Computer Science*, Vol. 2529, pp. 308 – 326, 2002.

[45] D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In *Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 held jointly with 9th European Software Engineering Conference*, pp. 257–266, 2003.

[46] D. Giannakopoulou, J. Magee, and J. Kramer. Checking progress with action priority: Is it fair? In *Proceedings of the 7th European Software Engineering Conference held jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 511–527, 1999.

[47] A. Griesmayer, S. Staber, and R. Bloem. Automated fault localization for C programs. *Electronic Notes in Theoretical Computer Science*, Vol. 174, pp. 95–111, 2007.

[48] A. Groce, S. Chaki, D. Kroening, and O. Strichman. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer*, Vol. 8, No. 3, pp. 229–247, 2006.

[49] A. Groce and W. Visser. What went wrong: Explaining counterexamples. In *Proceedings of the 10th International SPIN Workshop on Model Checking of Software, Lecture Notes on Computer Science*, Vol. 2648, pp. 121–135, 2003.

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

[51] T. A. Henzinger, R. Jhala, R. Majumdar, and M. A. Sanvido. Extreme model checking. *Verification: Theory and Practice, Lecture Notes in Computer Science*, Vol. 2772, pp. 332–358, 2004.

[52] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, Vol. 12, No. 10, pp. 576–580, 1969.

[53] C. A. R. Hoare. *Communicating sequential processes.* Englewood Cliffs, N.J. ; London, Prentice-Hall International, 1985.

[54] G. J. Holzmann. The logic of bugs. *ACM SIGSOFT Software Engineering Notes*,

Vol. 27, No. 6, pp. 81–87, 2002.

[55] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison-Wesley, 2004.

[56] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Proceedings of the 10th European Symposium on Programming Languages and Systems, Lecture Notes In Computer Science*, Vol. 2028, pp. 155 –169, 2001.

[57] D. Jackson. A direct path to dependable software. *Communications of the ACM*, Vol. 52, No. 4, pp. 78–88, 2009.

[58] M. Jackson. *Problem frames: Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[59] I. Jacobson, G. Booch, and J. Rumbauch. *The unified software development process*. Addison-Wesley, 1999.

[60] R. Jhala and R. Majumdar. Software model checking. *ACM Computing Surveys*, Vol. 41, No. 4, pp. 21:1–21:54, 2009. Article 21.

[61] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, pp. 467–477, 2002.

[62] C. Killian, J. W. Anderson, R. Jhala, and A. Vahdat. Life, death, and the critical transition: finding liveness bugs in systems code. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, pp. 243–256, 2007.

[63] S. Konstantinidis and P. V. Silva. Computing maximal error-detecting capabilities and distances of regular languages. Technical report, CMUP 2008-28, 2008.

[64] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, Vol. 27, No. 3, pp. 333–354, 1983.

[65] J. Kramer, J. Magee, and S. Uchitel. Software architecture modeling & analysis: a rigorous approach. *Formal Methods for Software Architectures, Lecture Notes in Computer Science*, Vol. 2804, pp. 44–51, 2003.

[66] T. Kumazawa and T. Tamai. Iterative model fixing with counterexamples. In *Proceedings of the 15th Asia-Pacific Software Engineering Conference*, pp. 369–376, 2008.

[67] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, Vol. 19, No. 3, pp. 291–314, 2001.

[68] K. Larsen and B. Steffen. A modal process logic. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pp. 203–210, 1988.

[69] D. Lee and M. Yannakakis. Online minimization of transition systems (extended abstract). In *Proceedings of the 24th annual ACM Symposium on Theory of computing*, pp. 264–274, 1992.

[70] E. Letier, J. Kramer, J. Magee, and S. Uchitel. Deriving event-based transition systems from goal-oriented requirements models. *Technical Report 02/2006, Imperial College*, 2006.

[71] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, Vol. 10, No. 8, pp. 707–710, 1966.

[72] N. G. Leveson. *Safeware: System safety and computers.* Addison-Wesley, 1995.

[73] N. G. Leveson. The role of software in spacecraft accidents. *American Institute of Aeronautics and Astronautics Journal of Spacecraft and Rockets*, Vol. 41, No. 4, pp. 564–575, 2004.

[74] J. Magee and J. Kramer. *Concurrency: State models & Java programming.* John Wiley & Sons, 2nd edition, 2006.

[75] W. H. Maisel, M. O. Sweeney, W. G. Stevenson, K. E. Ellison, and L. M. Epstein. Recalls and safety alerts involving pacemakers and implantable cardioverter-defibrillator generators. *Journal of the American Medical Association*, Vol. 286, No. 7, pp. 793–799, 2001.

[76] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification.* Springer-Verlag, 1992.

[77] S. P. Miller, M. W. Whalen, and D. D. Cofer. Software model checking takes off. *Communications of the ACM*, Vol. 53, No. 2, pp. 58–64, 2010.

[78] R. Milner. *Communication and concurrency.* Prentice-Hall, 1989.

[79] R. Milner. *Communicating and mobile systems: The π-calculus.* Cambridge University Press, 1999.

[80] M. Mohri. Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, Vol. 14, No. 6, pp. 957–982, 2003.

[81] S. Nakajima and T. Tamai. Behavioural analysis of the Enterprise JavaBeans^{tm} component architecture. In *Proceedings of the 8th International SPIN Workshop on Model checking of Software*, pp. 163–182, 2001.

[82] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, Vol. 33, No. 1, pp. 31–88, 2001.

[83] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *Proceedings of the 29th International Conference on Software Engineering*, pp. 54 – 64, 2007.

[84] E. Nuutila and E. Soisalon-Soininen. On finding the strongly connected components in a directed graph. *Information Processing Letters*, Vol. 49, No. 1, pp. 9 –14, 1994.

[85] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proceedings of the 11th International Conference on Automated Deduction, Lecture Notes In Computer Science*, Vol. 607, pp. 748–752, 1992.

[86] D. Peled and L. Zuck. From model checking to a temporal proof. In *Proceedings of the 8th International SPIN Workshop on Model checking of Software, Lecture Notes in Computer Science*, Vol. 2057, pp. 1–14, 2001.

[87] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. In *Proceedings of the Conference on Verification, Model Checking, and Abstract Interpretation, Lecture Notes in Computer Science*, pp. 364–380, 2006.

[88] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Proceedings of the 18th IEEE International Conference on In Automated Software Engineering*, pp. 30–39, 2003.

[89] W. W. Royce. Managing the development of large software systems. In *Proceedings of IEEE WESCON*, pp. 1–9, 1970.

[90] M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Requirements Engineering*, Vol. 11, No. 3, pp. 174–193, 2006.

[91] O. V. Sokolsky and S. A. Smolka. Incremental model checking in the modal μ-calculus. In *Proceedings of the 6th International Conference on Computer Aided Verification, Lecture Notes In Computer Science*, Vol. 818, pp. 351–363, 1994.

[92] O. Sokolsky, S. Kannan, and I. Lee. Simulation-based graph similarity. In *Proceedings of 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science*, Vol. 3920, pp. 426–440, 2006.

[93] S. Staber, B. Jobstmann, and R. Bloem. Finding and fixing faults. *Lecture Notes in Computer Science*, Vol. 3725, pp. 35–49, 2005.

[94] L. Staiger. ω-languages. *Handbook of Formal Languages*, Vol. 3, pp. 339–387, 1997.

[95] T. Tamai. Social impact of information system failures. *Computer, IEEE Computer Society Press*, Vol. 42, No. 6, pp. 58–65, 2009.

[96] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Journal of Mathematics*, Vol. 5, No. 2, pp. 285–309, 1955.

[97] S. Uchitel, G. Brunet, and M. Chechik. Synthesis of partial behaviour models from properties and scenarios. *IEEE Transactions on Software Engineering*, Vol. 35, No. 3, pp. 384–406, 2009.

[98] S. Uchitel and M. Chechik. Merging partial behavioural models. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 43–52, 2004.

[99] R. J. van Glabbeek. The linear time - branching time spectrum I. *Handbook of Process Algebra, Elsevier*, pp. 3–99, 2001.

[100] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pp. 249–263, 2001.

[101] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science*, pp. 332–344. IEEE Computer Society Press, 1986.

[102] W. D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, Vol. 22, No. 6-7, pp. 701–704, 2001.

[103] C. Wang, Z. Yang, F. Ivancic, and A. Gupta. Whodunit? Causal analysis for counterexamples. In *Proceedings of 4th International Symposium on Automated Technology for Verification and Analysis, Lecture Notes in Computer Science*, No. 4218, pp. 82–95, 2006.

[104] J. M. Wing and M. Vaziri-Farahani. A case study in model checking software systems. *Science of Computer Programming*, Vol. 28, pp. 273–299, 1997.

[105] C. Yilmaz, A. Paradkar, and C. Williams. Time will tell: Fault localization using time spectra. In *Proceedings of the 30th International Conference on Software Engineering*, pp. 81–90, 2008.

[106] A. Zeller. Isolating cause-effect chains from computer programs. In *Proceedings of the 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 1–10, 2002.

[107] Y. Zhang and Y. Ding. CTL model update for system modifications. *Journal of Artificial Intelligence Research*, Vol. 31, No. 1, pp. 113–155, 2008.

[108] R. Ziller and K. Schneider. Combining supervisor synthesis and model checking. *ACM Transactions on Embedded Computing Systems*, Vol. 4, No. 2, pp. 331–362, 2005.

[109] 熊澤努, 玉井哲雄. モデルに基づく誤り特定と反例修正候補の提示. 鵜林尚靖, 岸知二（編）, ソフトウェアエンジニアリング最前線 2009, pp. 55 – 62. 社団法人 情報処理学会ソフトウェア工学研究会, 近代科学社, 2009.

[110] 高井利憲, 古橋隆宏, 尾崎弘幸, 大崎人士. 環境ドライバを用いたモデル検査による検証事例. Technical Report PS-2007-010, 独立行政法人 産業技術総合研究所システム検証研究センター, 2007.

[111] 玉井哲雄. ソフトウェア工学の基礎. 岩波書店, 2004.

[112] 玉井哲雄. ソフトウェア工学の 40 年. 情報処理, Vol. 49, No. 7, pp. 777–784, 2008.

[113] T. デマルコ (著), 高梨智弘, 黒田順一郎 (訳). 構造化分析とシステム仕様 - 目指すシステムを明確にするモデル化技法. 日経 BP 社, 1994.

[114] 独立行政法人 情報処理推進機構ソフトウェアエンジニアリングセンター. ソフトウェア開発データ白書 2008. 日経 BP 社, 2008.

[115] 中島震. モデル検査法のソフトウェアデザイン検証への応用. コンピュータソフトウェア, Vol. 23, No. 2, pp. 72–86, 2006.

[116] 林晋. プログラム検証論. 共立出版, 1995.

[117] G. J. マイヤーズ (著), 国友義久, 伊藤武夫 (訳). ソフトウェアの複合 / 構造化設計. 近

代科学社, 1979.

[118] I. ヤコブソン, M. クリスターソン, P. ジョンソン, G. ウーバガード (著), 西岡利博, 渡邊克宏, 梶原清彦 (監訳). オブジェクト指向ソフトウェア工学 OOSE : use-case によるアプローチ. アジソン　ウェスレイ・トッパン, 1995.