通信ソフトウェア仕様検証技術の

効率向上に関する研究

若原　恭

通信ソフトウェア仕様検証技術の
効率向上に関する研究

若原　恭

目 次

第1章　序論

1.1　研究の背景

情報社会の発展に伴って情報通信は高度化･多様化の一途をたどっており，電話網による音声主体の通信からインターネットによるマルチメディア情報通信に大きく変遷しようとしている．このような情報通信の実現の核となる通信ソフトウェアの開発量･保守量は増大する一方である．その結果，通信ソフトウェアの開発･保守における作業の効率化と品質の向上が一層重要な課題となってきた．この課題の解決には，通信ソフトウェアの開発･保守工程の全体にわたる技術の改革が必要である．しかし，開発･保守の初期の段階で作成する(要求)仕様がその後の全作業工程の基本となりまた要となるので，正確な仕様の作成を支援する仕様検証技術が特に重要である．

これまで仕様検証技術については様々な研究開発が進められてきたが，通信ソフトウェアに適した仕様検証技術についての研究開発は十分ではなかった．

通信ソフトウェアの中で重要な位置を占めるものの一つに通信プロトコルがある．一般に，通信プロトコルの仕様は大規模でかつ複雑であり，誤りを含むことが少なくなく，その結果通信システムの動作に重大な支障を来すことがある．従って，プロトコル仕様に含まれる誤りを検出するプロトコル検証は必須である．コンピュータを利用して機械的にプロトコル検証を行う技術に関しては1970年代に研究が開始し，これまでに検証の原理とそれに基づく多くの検証法が研究されてきた．そして，これらの研究成果を応用したプロトコル検証ツールが開発され実際に使用され始めている．市販されている汎用的なプロトコル検証ツールもある．しかし，これらの検証技術では，検証処理の効率が低く，プロトコル規模が大きくなると検証に必要なメモリ量や処理量が指数関数的に急激に増大するため，多くの実用的規模のプロトコルが検証できないという問題がある．そして，この問題が解決されていないため，市販プロトコル検証ツールも必ずしも広く実用されている状況にはない．

通信ソフトウェア仕様の検証には，プロトコル検証だけでは十分ではない．しかし，これまでのところ総合的かつ体系的な検討は少なく，通信ソフトウェア仕様の特徴を考慮した効率的な仕様検証技術は未だ確立されてはいない状況にある．例えば，通信ソフトウェア仕様は一般に規模が大きいため，通常一部分づつ段階的に作成される．しかし，このような仕様の段階的作成を効率よく支援する検証技術はあまり開発されておらず，特に，完全でない仕様を対象とした検証については検討がほとんどなされてこなかった．また，通信ソフトウェア仕様の検証はある程度人手による必要があるが，仕様は大規模でかつ複雑な場合が多く，検証の前提となる仕様の理解自体が容易でない場合が少なくなかった．しかし，理解が容易でない通信ソフトウェア仕様の人手による検証を効率よく支援する技術の研究も十分には取り組まれていなかった．更に，理解が容易でない通信ソフトウェア仕様の人手による検証作業自体を効率よく支援する技術についての検討も十分ではなかった．

このように通信ソフトウェア仕様の検証に関しては，プロトコル検証の効率を大幅に向上できる技術，及び，通信ソフトウェア仕様向きの検証機能を持った新しい効率よい技術の開拓が必要な状況にあった．

1.2 研究の目的

1.1で述べた通り，通信ソフトウェア仕様の検証については多くの問題が存在しており，情報社会の発展に伴う通信ソフトウェアの役割の増大に伴って，このような問題の解決が一層重要になってきた．本論文の目的は，これらの問題の解決を図ることにあり，具体的には，次の通り，二つに大別できる．

本論文の第1の目的は，信号送受信の過不足やデッドロック等，通信ソフトウェア仕様の要となるプロトコルの動作仕様に関する論理的な誤りを検出するプロトコル検証の効率を大幅に向上し，これまでは不可能であった実用的規模のプロトコルの検証を可能とする技術を確立することにある．即ち，規模が大きなプロトコル，特にプロトコルの処理を司るプロセスが多い現実的なプロトコルに含まれる誤りの完全な検出を可能にするプロトコル検証技術を確立することにある．基本的には，プロトコル動作を模擬してすべての誤りを検出するという方針を採用し，これまでの検証法においてボトルネックとなっていた所要メモリ量と処理量の大幅な削減を可能にする技術の開拓を目標とした．

本論文の第2の目的は，原始要求に対する通信ソフトウェア仕様の充足性に関する検証を効率よく実現する新しい技術の開拓にある．特に，大規模な通信ソフトウェア仕様を一気に作成することは稀であり，通常一部分づつ段階的に作成していくため，途中段階の仕様は当然不完全なものとなる．そこで，このような不完全な仕様の段階的な機能検証を可能にする技術の開拓を目標とした．また，大規模で複雑な通信ソフトウェア仕様の機能や動作は，作成した技術者自身でも理解が容易でなくなることが少なくないことから，与えられた仕様をできる限り簡明な表現に自動的に変換して検証を容易にすることによって検証の効率を向上する技術の開拓を目標とした．更に，機能に関する要求条件を入力し，与えられた仕様がそれを満たしているか否かを効率よく自動的に検査できる技術の開拓を目標とした．

1.3 論文の構成

本論文は1.2で述べた目的で遂行した研究成果をとりまとめたものである．

まず，第2章で，本論文に関連あるこれまでの研究状況の概要を述べ，本論文における研究の位置づけを明確にする．

次に，第3章で，通信ソフトウェアの開発保守における仕様検証技術の役割と意義を明らかにする．即ち，通信ソフトウェアの特徴に基づくライフサイクルのモデルを提示する．そして，このモデルに基づいて通信ソフトウェア仕様作成支援技術を総合的に論じ，その一部である仕様検証技術の位置づけを明確にする．更に，検出すべき仕様誤りを整理し，この結果をベースに仕様検証技術を体系的に論じ，具体的な検証技術の概要を提示する．

次に，第4章で，通信プロトコル仕様の論理に関する検証の効率を大幅に向上できる新しい検証技術を提案し，その原理とアルゴリズムを明らかにするとともに，その効率向上の効果を論じる．具体的には，プロトコル検証問題を定義した後，プロトコル検証の効率を向上する次の4件の技術を論じる．

①プロセスを削減することによって，検証効率を向上する方法．

②プロトコル動作の模擬のために行うプロトコル仕様の展開において，プロセス遷移を一括処理することによって検証効率を向上する方法．

③プロトコル仕様をプロセス毎にアサイクリック状に展開することによって，検証効率の向上を図る方法．

④検出すべき誤りの性質に応じて段階的に検証することによって，検証効率を向上する方法．

これらのプロトコル検証技術を，具体的なプロトコル仕様に適用し評価した結果を論じ，所要メモリ量や計算処理量が大幅に削減できることを示し，これら検証技術の有効性を実証する．また，これらの新検証技術によって，現在国際通信に使用されている大規模な通信システムのプロトコルの検証が可能になったことを示す．

第5章では，プロトコル検証以外の新しい通信ソフトウェア仕様検証技術を論じる．具体的には，まず，与えられた不完全な通信ソフトウェア仕様をプログラムに自動的に変換して，このプログラムを実行しその結果を確認することによって検証を行うプロトタイピング技術を提示し，その方式を具体的に論じた後，実験を通してこの新技術の有効性を実証する．次に，人手で作成した通信ソフトウェア仕様を，それと機能的に等価でかつより簡明な表現に自動的に変換することによって仕様検証の効率向上を図る新しい検証技術とその有効性を論じる．更に，通信ソフトウェア仕様が満たすべき機能要件を仕様要素の実行順序に関する条件で与え，通信ソフトウェアを構成する各プロセスの仕様における仕様要素の実行系列（フロー）を機械的に検査することによって，仕様が機能要件を満足しているか否かを効率よく検証する技術を論じ，その効果を明らかにする．

第6章で，本論文の結論を述べる．

本文の第4章と第5章で示した各種通信ソフトウェア仕様検証技術を総合的に実装した通信ソフトウェア仕様検証支援システム ESCORT(Environment for Specifying Communications Software Requirements)を開発した．付録で，ESCORT の概要とその適用例を説明する．

第2章　研究の位置づけ

2.1　通信ソフトウェアのモデルと仕様記述言語

通信ソフトウェアには，交換機ソフトウェア，通信ノード用ソフトウェア等の通信網を利用して端末同士を接続するための基本的なソフトウェアや端末（コンピュータ）同士の間で情報を交換するためのファイル転送ソフトウェア，遠隔ログインソフトウェア，メール転送ソフトウェア等のアプリケーションソフトウェア等，多くの種類がある．本論文では，このような通信ソフトウェアの仕様に対する検証技術を議論するが，対象とする通信ソフトウェアとその検証技術を明確にするため通信ソフトウェアのモデル化を行う．一般に通信ソフトウェアに適用できるモデルには，抽象データ型に基づく代数的モデル，データフローモデル，有限状態機械モデル，CSP(Communicating Sequential Processes)，CCS(Calculus of Communicating Systems)等がある[1]~[4]．

本論文では，拡張された有限状態機械で通信ソフトウェアをモデル化し，通信ソフトウェア仕様の記述言語には，国際電気通信連合-電気通信セクタ ITU-T（旧国際電信電話諮問委員会 CCITT）において国際標準として勧告化された通信システム用記述言語 SDL(Specification and Description Language)を採用した[5]~[11]．

拡張有限状態機械モデルを採用した理由は，その基本要素である状態・遷移・事象・動作が，実際の通信ソフトウェアの基本要素である安定状態・別の安定状態への変遷・信号の送受信等のイベント・処理内容にそれぞれ対応していてモデルへの適合性が高く，通信システムで重要となる処理の効率性の観点からも拡張有限状態機械モデルが有効であり，また，状態を単位として通信ソフトウェアの動作や記述がモジュール化でき理解性や記述性の観点からも優れているからである[1]．実際，拡張有限状態機械モデルに基づいて構築される通信ソフトウェアは極めて多い．一方，代数的モデル，CSP，CCS 等は，研究分野では比較的広く取り扱われているが，記述・理解の両面において難点があるため，また，現実的には数学的な厳密性が強く要求されることは稀であるため，研究分野で取り上げられているほど広くは実用されてはいない．

一方，SDL を採用した理由は，SDL が有限状態遷移機械モデルに基づいており，SDL には通常のプログラミング言語と同様なテキスト形式の表現 SDL/PR（Phrase Representation）に加え，図形式の表現 SDL/GR(Graphical Representation)があるため理解性に優れているうえ，国際標準として文法やセマンチックス等が明確に定義されていて参考書籍も充実してきているからである[5]~[14]．ただし，SDL は欧米では古くからかなり広く実用されてきたものの，日本語の詳細な教科書や解説書が最近まで発行されていなかったこともあって，我が国においては必ずしも広く実用化されてきたわけではなかった．また，SDL が実用される場合，現実的には，必ずしも文法に照らして正しく記述されるわけではなく，SDL/GR を用いて非形式的に記述されることが多い．なお，SDL のデータ型には抽象データ型が採用されているが，これは SDL/PR にのみ規定されている．

国際標準の仕様記述言語には，SDL 以外に，LOTOS(Language Of Temporal Ordering Specification)と Estelle とがあり，両言語ともに国際標準化機構 ISO で制定された[9]．LOTOS は，各プロセスをブラックボックスと見做し，プロセスの外から観測できるプロセス間での送受信の時間順序に基づいて仕様を記述する言語で，特に ISO で定めた OSI（開放型システム間相互接続）プロトコルを規定するために開発された．データ型には SDL と同様の抽象データ型を採用しており，図式表現 G-LOTOS(Graphical LOTOS)もある．現実的には，OSI プロトコルの規定に実用されてはいるが，これ以外にはあまり実用されてはいない．しかし，研究分野では，数学的な厳密さがあるため学術的に扱い易いこともあって広く利用されている．一方，Estelle はプログラミング言語 Pascal の言語機能を，有限状態機械モデルに適合するよう拡張したもので，LOTOS と同様に主として OSI プロトコルの規定に使用するため開発された．Estelle は仕様記述言語というよりもプログラミング言語に近く，比較的特長が少ないうえ，有限状態機械に基づく国際標準言語 SDL が既に存在していることもあって，あまり実用されていない．

2.2　通信プロトコル仕様の論理検証技術

通信ソフトウェアの重要な役割の一つは通信プロトコルの処理にある．通信プロトコルの仕様（以下，単にプロトコルと呼ぶことがある）には主として信号の送信や受信に関する規定が記述されるが，このような信号の送受信に係わる論理的な誤りを機械的に検出する通信プロトコル仕様の論理検証（以下，単にプロトコル検証と呼ぶ）に関する研究は 1970 年代に開始した．以来多くの研究が行われ，様々なプロトコル検証法やその改善法が研究開発されてきた[15]~[19]．これらの検証法は，大別して 4 つの方法に分類できる．

第 1 のプロトコル検証法では，信号の送信や受信等のプロトコルの動作を逐一模擬することによって，プロトコルに係わるシステム全体の動作と状態を表すグローバルな状態をすべて列挙し，グローバル状態とそれらの間のグローバル遷移関係を表すグローバル状態遷移図を導出する．そして，グローバル状態・グローバル遷移やそれらに関連するデータ等の網羅的な検査によって誤りを検出する[20]~[22]．この方法は，到達可能木解析法とも呼ばれている．

しかし，このままではプロトコルを司る実体であるプロセスやプロセスが持つ状態・遷移の増加に伴って列挙すべきグローバル状態とグローバル遷移の個数が急激に増大し，検証に要するメモリ量及び処理量が指数関数的に増加するので，規模が大きなプロトコルは実際には検証できなくなってしまう．つまり，小規模なプロトコル仕様しか検証できず，多くの現実的なプロトコルの検証は不可能であるという問題があった．この問題は「状態数爆発問題」と呼ばれることがある．このプロトコル検証法は，プロトコル論理検証の原理を与え，以降のプロトコル検証の研究開発の出発点になったという意味で重要な研究成果であった．

この基本的なプロトコル検証法がもつ状態数爆発問題の解決を図るため，検証の対象となるプロトコルに対して，プロセス数を 2 個に限定したり，ループ状の遷移を許さない等何らかの制約

を設けることによって効率を改善する方法が検討された[23],[24]．検証プロトコルに制約を設けるのではなく，検証処理自体の効率化を図るため，同一の繰り返しとなるグローバル状態遷移系列の列挙をできる限り回避するような工夫も考えられた[25]．更に，信号の送受信を入れ替えれば同一のグローバル状態遷移系列になる場合は１回だけの検査だけで済ませてしまう方法，個々のプロセスを中心としたローカルな検査によってプロトコル全体の検証を実現する方針とし各ローカル検証では当該プロセス以外のプロセスの仕様をできる限り縮退させておく方法等が考案された[26]．これらの手法によって60～80%程度削減できるという報告がある[26]～[31],[18]．

しかし，これらの検証法の改善効果には限界があり，特にプロセスが多いプロトコルを検証する場合には所要メモリ量と処理量がやはり大きくなり，プロトコル検証が不可能になるという問題がある．

第２のプロトコル検証法では，プロセス毎の動作とそれに伴う他プロセスの最小限の動作を模擬し管理するプロセス状態遷移図を作成し，それを検査することによって誤りを検出する[32]．一方，この方法を，プロセスの個数と同数のCPU(コンピュータ)で並列に同時処理できるように拡張して効率の向上を図る方法も考案された[33]．このような新しい原理に基づく検証法によって，プロセスがある程度多くなっても検証が可能となったが，検証可能なプロトコル仕様の規模はやはりかなり限られており，総合的には，第１の検証法と同様な状態数爆発問題がある．このプロトコル検証法は，第１のプロトコル検証法とは原理が全く異なるもので，比較的早い時期に考案されたにも拘わらず，その動作には必ずしも理解が容易でない点があり，またその効果を定量的に明示した論文が初期には発表されなかったため，あまり注目を受けず，論文が引用されることも比較的少なかった．

第３のプロトコル検証法では，プロトコル動作を逐一網羅的に模擬するのではなく，多数の実行可能な動作の中から一部の動作のみを選択し模擬する．一部の動作を選択する方法には，①ランダムにプロトコル動作を選択する方法[34]～[36]，②実際に発生する状態遷移の確率に応じて選択する，つまり，よく起きるプロトコル動作を主として検証の対象にする[37]～[39]，③導出するグローバル状態遷移図に含まれる各プロセスの遷移の個数ができる限り等しくなるように動作を選ぶ[40]，等の方法がある．しかし，これらの方法では，同じ動作を２度以上検査することがあり，返って効率が悪い場合もある．また，現実的には，滅多にしか生じない動作こそ計算機によって機械的な検証を済ませておきたいという要求がある．更に，例えばループ状の遷移があるとやはり計算量は極めて多くなる．

このように第３の方法には，一部の動作のみを検査するという比較的簡単な原理によって検証処理量を容易に削減できかつその削減程度を比較的柔軟に制御することができる．しかしながら，この方法では，個々の問題に加え，全体として，すべての誤りを検出することができないうえ，検査がどの程度網羅できたかの判定が容易でないという致命的な問題がある．

第１の検証法では，グローバル状態を導出する毎に，既に列挙したグローバル状態のいずれかと一致するか否かを判定するため列挙したグローバル状態をすべて蓄積しておき比較判定する必

要があり，これが所要メモリ量と所要計算処理量の支配的要素となる．第４のプロトコル検証法は，この事実に着目し，その蓄積用メモリ量と関連処理の量を削減することを主眼に考案されたもので，ハッシング関数を通してグローバル状態の一致を判定する[18],[40]～[44]．即ち，グローバル状態同士が一致しているか否かを直接比較して判定するのではなく，グローバル状態をハッシング関数に入力して処理しその結果得られる値同士が一致するか否かで判定する．この結果，導出したグローバル状態が列挙済みのグローバル状態と完全には一致しなくても，そのハッシング関数の出力値が同一であればグローバル状態が一致していると誤って判定し，それ以降のプロトコル動作の検査を中止することがある．この方法はSupertraceと呼ばれ，ハッシング関数の出力種類数をメモリ総量で制限することによって，使用メモリ量を一定に押さえられるという特長がある．その効果については，ハッシング関数の適切な設計によって，グローバル状態が100万個以上ある場合でもその90%は実質的にカバーでき，検証に必要なメモリ量は最大99%程度削減できる，つまり検証効率が100倍向上できると報告されている[40],[42]．しかし，本来は検査し続ける必要がある場合でも検査を止めてしまうことがあるため，第３の検証法と同様，誤りをすべて検出することは不可能であるという致命的な問題がある．なお，このような致命的な問題があるものの，実用的な規模のプロトコルを検証するための現実的なプロトコル検証法が他に存在しないため，このSupertrace法はかなり実用性のある手法と考えられてきた．

なお，以上とは異なる検証アプローチとして，いったん検証が完了したプロトコルを少し変更した場合，最初から完全に検証し直すのではなく，変更前の検証結果を保存しておき，これを活用することによって，更新されたプロトコルを効率よく検証する方法の検討もある[44]．しかし，この方法を利用する前に，与えられたプロトコルを最初から完全に効率よく検証する方法がやはり必要である．

以上の通り，これまで研究し開拓されてきたプロトコル検証法には総じて検証効率が不十分であり，実用されている大規模なプロトコルの完全な検証は極めて困難であるという問題があった．例えば，11個のプロセスから構成され各プロセスの状態数，遷移数がそれぞれ37～2，57～2である交換機間共通線信号方式No.7の電話ユーザ部TUPの基本部は，一部プロセスの規模が大きいため，所要メモリ量が極めて大きくなり，第１の基本的な検証法では完全な検証が不可能であった．また，状態数が6で遷移数が12の状態遷移図で表されるプロセスで各ノードの機能がモデル化されるトークンリングLAN用プロトコルについては，第１の基本的な検証法ではノード数が10個強までの場合しか検証できなかった．また，Supertraceで初めて実用プロトコルサイズの検証が可能になったとの実験報告もあり[18]，実用規模のプロトコルを完全に検証できるためには，第１の最も基本的な検証法に比較して少なくとも100倍以上の検証効率の向上が必須であると考えられる．

本論文では，これまでに開発されてきた上述の様々なプロトコル検証法が持つ問題を解決し，これまでは検証が不可能だった大規模プロトコル，特にプロセスが多い現実的なプロトコルに含まれる誤りの完全な検出を可能にすることを目的として，これまでに開発されてきた検証法に比

較して検証効率が大幅に優れたプロトコル検証技術を開拓することにした．具体的には，プロトコル動作を模擬するよう状態遷移図を展開していき，その結果を検査することによってすべての誤りを検出するという方針を採用し，検証効率を少なくとも 100 倍以上向上させることを目標として，この展開状態遷移図の規模，即ちそれに含まれる状態と遷移の個数を，誤り検出に支障のない範囲で最小限に留めることが可能な新しい検証手法を開拓することとした．具体的には，以下に示す4手法を考案した．

①プロセスの削減：検証効率を向上するため，タイマ管理プロセスやリソース管理プロセス等の通信相手プロセスが1個に限られるプロセスを，与えられたプロトコル仕様から削除して検証する[45]．（4.3）

②プロセス状態遷移の一括化：第1に分類される検証法をベースに，誤り検出に支障のない範囲で，プロセス状態遷移を徹底的に省略すると同時に，一括化してまとめる．これによって，列挙するグローバル状態の個数とグローバル状態間遷移の個数を直接的に大幅に削減する[46],[47]．（4.4）

③プロセス仕様のアサイクリック展開：第2に分類される検証法をベースにして，プロセス毎にプロトコル仕様を展開していくが，展開の停止の仕方を徹底的に厳しくし，アサイクリック状に展開する．これによって，展開状態数を大幅に削減する[48],[49]．（4.5）

④検出誤りの性質に応じた段階的処理：検出誤りを，グローバル状態の列挙が本質的に必要か否かという観点から分類し，それぞれに必要最低限の処理を2段階に分けて実施することによって，プロトコル全体の検証を実現する[50]．（4.6）

上述の4手法のうち①プロセスの削減に関連して，タイマ管理プロセスを削除してプロトコル検証する方法の提案があるが[44]，この方法では，タイマ管理プロセスとの間の送受信は常に実行可能であるものとして検証する．しかし，実際にはそのような送受信は常に実行可能であるとは限らない．従って，この方法による検証は正確ではなく，実際には存在しない誤りを出力することがある．これに対して本論文で提案する①プロセス削減法ではこのような不正確な仮定は設定しないため，正確なプロトコル検証が可能である．このような意味から，このプロセス削減法のアプローチは，プロトコル検証に関する過去の研究とは全く異なるもので，極めてユニークなものである．

以上で説明したプロトコル検証法に類似した技術として，ハードウェア回路の故障検出や診断に応用される「有限状態機械の試験技術」がある．この試験技術は，有限状態機械をブラックボックスと捉え，それに適切な入力を与えて出力を外部から検査することによって，その状態遷移の定義に従って有限状態機械が動作するか否かを決定するものである[51]~[53]．この試験技術のポイントは，直接観測できない内部状態を推定することにあり，試験対象の有限状態機械がとり得るすべての状態とその時間軸上の組合せを考慮して，初期状態・最終状態や状態遷移関数と出力関数を決定するのに必要な最小限の入力系列をテスト系列として導出することに基本がある[54]．

一般に有限状態機械の入力系列は多数存在する．特に状態数が多く規模が大きな有限状態機械

に対する入力系列は極めて多くなり，最短のテスト系列を導出するための計算量は膨大な値となる[55],[56]．そこで，この問題を解決しテスト系列を効率よく導出する方法に関し多くの研究が行われている．例えば，テストの対象となる有限状態機械が持つすべての状態とその遷移をカバーするテスト系列を用いて故障を検出する状態図アプローチ[57]とその改善法[58]，有限状態機械に含まれるフィードバックループを時間軸上で展開し，全体としてループのない組合せ回路として取り扱う方法[59]等がある．また，いかなる入力系列に対しても到達しない状態を事前に検出することによって，テスト系列の導出に無駄となる計算処理を回避する方法の研究もある[60]．このように多くの研究がなされてはいるが，故障を確実に検出する方法の効率化には限界があるため，現実的な方法としては，実用性の観点からできるだけ速やかに故障を検出するという方針を採用し，ランダムなパターンのテスト系列を用いる方法もある[52],[53]．

このような試験技術は，プロトコルに係わる論理的な誤りを検出するプロトコル検証法に比較すると，一般に多くの状態の組合せを列挙して検査するという点で類似している．しかし，有限状態機械の試験では基本的には一つの有限状態機械を対象としていて時間軸上での状態の組合せを考慮しているが，プロトコル検証法では2個以上の（拡張）有限状態機械（プロセス）を対象にしているため，時間軸上での状態の組合せに加え複数個の有限状態機械が持つ状態の空間軸上の組合せも考慮する必要がある．また，試験技術では故障の検出に必要な入力系列の導出が目的であり，例えば与えられた状態遷移の定義から特定の状態に到る入力系列の導出に必要な状態とその系列をすべて列挙するが，プロトコル検証ではプロトコルに係わる誤りをすべて導出することを目的としているため，複数の有限状態機械が持つ状態に加え有限状態機械相互の間における情報の授受に関する状態の組合せを含めてすべて列挙することが基本となる．このような理由から，特に有限状態機械（プロセス）が多くなるほど列挙すべき状態の組合せの個数は多くなり，現実的には，これに起因する状態数爆発の問題を解決することがプロトコル検証法の実用化における重要な鍵となっている．そして，本論文ではこの状態数爆発問題を解決するため，誤りの検出に支障がない範囲で無駄な列挙をできる限り回避する実用性の高い方法を論じているが，これらの方法は試験の効率向上技術とは本質的に異なるものである．

2.3　通信ソフトウェア仕様の意味検証技術

従来，通信ソフトウェアに対する原始要求がその仕様に正しく反映されているか否かを検査することによって誤りを検出する意味検証については，通信ソフトウェアを含む一般のソフトウェア向けに開発されてきた様々な方法が利用されてきた．しかし，多くの場合，完全な検証つまり仕様に含まれている誤りをすべて検出することは実質的に不可能と考えられており[1]~[3]，現実的には与えられた仕様に対し種々の手法を適用し，できる限り効率よく誤りを検出するよう検証作業が実施されている．このような背景から，通信ソフトウェア仕様の意味検証には，多様な検証法が考案され研究されてきた．これらの検証法は，仕様を記述するためのモデルや言語に依存

するものが少なくないが，2．1で述べた通り，本論文では，通信ソフトウェアを拡張有限状態機械でモデル化し，仕様記述言語としてITU-Tの通信システム用仕様記述言語SDLを採用した．

仕様記述言語SDLは拡張有限状態遷移モデルに基づき，各状態における動作については手続き的な処理記述を採用している．このようなSDLで記述された仕様の意味検証のための主要な方法については，

①仕様をインタプリティブに解釈して又は仕様から(プロトタイプ)プログラムを作成して実行しその実行結果を検査する方法，

②何らかの着目点に従って仕様を編集加工し人手によるレビューを容易にする方法，

③仕様が満たすべき要求条件を何らかの形式で具体的な表明として表し，そのような表明が成立するか否かを検査する方法

に大別できる[6],[61],[62]．

(1)仕様の実行による意味検証

仕様を実行することによって実施する意味検証を本論文ではプロトタイピングと呼ぶ[63]~[65],[2]．SDLは基本的には状態遷移の記述をベースとする手続き型言語であり，SDLのモデルが持つ意味は明確に定義されている[5]．このため，SDLで記述された仕様を実行可能なプログラムに変換することは比較的に容易である．実際SDLからプログラミング言語のCやC++等に変換するツール，及び変換されて得られるプログラムを実行することによって仕様を検証するツールがいくつか開発されている[66]~[86]．このようなツールを総称してプロトタイピングツールと呼ぶことにする．しかし，これまでに開発されてきたプロトタイピングツールには，次に示す問題がある．

①一般に通信ソフトウェア仕様は規模が大きく複雑になることが多いため，通常段階的・部分的に作成していく．このため，部分的に作成された不完全な仕様を最大限プロトタイピングし，作成された範囲内で最大限の検証を行うことが望ましいが，このような機能は実現できていない．

②一般にプロトタイピングでは実行を多数回繰り返すことが必要になるが，毎回多くの入力データを準備する必要があり，このための作業量が膨大で煩雑になってしまうことが多い．

そこで，本論文ではこれらの問題を解決することを目的として，新しいプロトタイピング手法を開発することにした[87]~[90]．（5．2）

(2)仕様の編集加工によるレビューの容易化

現実的な意味検証には，ウォークスルーやインスペクションあるいはこれらに準じたレビュー方法を採用することが多い[3],[91]．しかし，通信ソフトウェアは一般に規模が大きく複雑であるためレビューは通常多量の作業を必要とし容易でない．従って，このようなレビューを容易にする方法は極めて重要である．しかし，レビューに関するこれまでの研究では，主眼はレビューの進め方や運用方法に置かれることが多く，実施時期・対象とするドキュメント・レビューチームの

構成・参加メンバの役割などの明確化が行われてきた．この結果，仕様の編集加工によってレビューの容易化を図る技術的な方法については，あまり体系的な検討はなされていない．

本論文では，このような方法の一つとして，与えられた仕様を，機能的にはそれと等価であるがより簡明でレビューが容易となる仕様に変換する方法をとり上げる．このような簡明化に適用できる手法としては，有限状態機械に基づくSDLで仕様がモデル化され記述されることを前提にすると，順序機械の最適化がある[92],[51]．この最適化によって，等価な状態を一つにまとめることができ，仕様規模を小さくすることが可能となる．

しかし，例えば通信ソフトウェア仕様には多くの判断分岐要素が含まれているため制御の流れが複雑であり，そのような制御の流れを単純にすることが望ましく，順序機械の最適化だけでは不十分である．そこで，本論文では，このような観点から，過去には全く検討されてこなかった新しい仕様簡明化法を開拓することにした[93]~[97]．（5．3）

(3)仕様に対する要求条件の充足性の判定

通信ソフトウェア仕様に対する要求条件を具体的に表した表明の充足性を検査する方法についての研究は少なくない．例えば，仕様記述法が代数的な抽象データ型に基づき，データ型とそれに対する演算及び演算に関して成立する公理によって仕様が記述される場合は，公理を表す等式関係を書き換え規則とみなすことによって，表明を表す命題が仕様から導出できるか否かを判定することで意味検証が可能となる[3],[4],[98]．また，述語論理に基づく仕様記述法の場合，証明システムが存在するケースもあるので，仕様を直接実行することが原理的に可能となる[3],[4],[98]．一方，従来からある古典的な論理に対し時間に関する概念を扱う時間演算子を付加した時相論理は，通信システムがいつかは所定の動作を完了する等の安全性や生存性等に関する性質の記述と検証が可能である[99],[100]．しかし，これらの方法は，一般に難解と言われている抽象データ型や述語論理に基づいているうえ，命題の導出や仕様実行には通常極めて多くの計算量を必要とするため，実用性が低いという問題がある．

要求条件を具体的に表した表明の充足性を検査する方法のうち，SDLのような手続き型言語に適用可能なものとしては，データフロー解析を応用した手法がある[101],[102]．これらの手法はもともとプログラムの検証のために開発されたものであるが，これらの応用によって，仕様要素を実行順序に従って逐一トレースし，その過程で表明の充足性を検査し判定することができる．しかし，これによって充足性が判定可能な表明は，一つの実行系列（フロー）に含まれる仕様要素に関する条件に限定されるという問題がある．実際の仕様では，複数個の実行系列にまたがった仕様要素間の条件の充足性の判定が必要になることが多い．

そこで，本論文では，複数個の実行系列に係わる要求条件の充足性の判定を可能とすることによって，上述の問題をすべて解決する新しい意味検証法を開拓することにした[103],[104]．（5．4）

第3章　通信ソフトウェア仕様の検証

3.1　まえがき

従来，通信ソフトウェアの生産性と品質の向上を図ることを目的として，その初期の段階で作成する(要求)仕様に含まれる誤り(仕様誤り)を検出する仕様検証技術について様々な研究が行われてきた．効果的な仕様検証技術を論じるためには，それによって検出すべき仕様誤りを明確にすることが必要不可欠である．しかし，これまでの研究では仕様誤りについての議論が十分ではなく，仕様誤りが体系的に明らかになった状況にはなかった．

そこで，本章では，通信ソフトウェア仕様検証技術を論じるための基礎として，仕様検証技術の位置づけを論じてその意義を明らかにし，仕様誤りと仕様検証技術全体の概要を体系的に整理する．具体的には，3.2で，通信ソフトウェアのライフサイクルモデルを提示し，3.3で，ライフサイクルにおいて仕様検証の位置づけを論じることによって，その意義を明らかにする．そして，3.3で，仕様誤りを体系的に整理し，構文誤り・論理誤り・意味誤りに大別できることを示す．これら仕様誤りのうち，特に論理誤りと意味誤りを検出するための検証技術を論じ，仕様検証技術の全体像を明らかにする．

3.2　通信ソフトウェアのライフサイクル

一般の情報処理ソフトウェアのライフサイクルモデルに関しては，既に多くの検討が行われている．その古典的な代表的モデルの一つは，開発・保守のための作業工程を，①要求定義(分析・仕様作成)，②基本設計，③詳細設計，④コーディング，⑤試験(単体試験・結合試験・過負荷試験・安定化試験・総合試験)，⑥運用・保守の6工程に分割整理し，各工程で作業結果の検証を完全に行い前工程に後戻りすることを許さないというウォーターフォールモデルであろう[2]．

このウォーターフォールモデルを通信ソフトウェアにそのまま適用することもできるが，特に開発・保守の面からは，通信ソフトウェアには，機能の追加や変更が絶え間なく要求されること，つまり，常に保守が行われるという特徴がある．このような特徴を考慮した通信ソフトウェアのライフサイクルモデルを図3.2.1に示す[105],[106]．この図で，新規の通信ソフトウェアの開発は，原始要求の分析・仕様化から運用に到る細い矢印で表現される．また，通信ソフトウェアに対する機能の追加や変更等の保守は，運用から新しい原始要求の分析・仕様化を経て，再び運用に到るループ状の太い矢印によって表現される．このように，図3.2.1のモデルでは，最初の要求分析・仕様化から始まって，その後基本設計・詳細設計……運用という半永久的なループ状の遷移によって通信ソフトウェアのライフサイクルが表現される．ただし，保守の内容によっては，一部の工程がスキップされることがある．

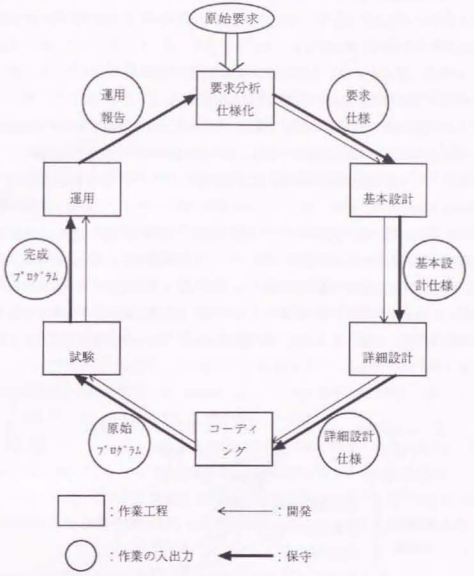


図3.2.1　通信ソフトウェアのライフサイクルモデル

3.3　通信ソフトウェア仕様検証の位置づけ

3.2で，図3.2.1を参照して述べた通り，要求分析・仕様化の目的は，通信ソフトウェアに対する原始要求を分析して具体化し，その結果を(要求)仕様としてとりまとめることにある．このような仕様には，一般に，以下に示す性質が要求される[106],[107]．

①一義性：記述内容があいまいでなく，一通りに解釈できること．

②理解性：記述内容が容易に理解できること．

③正当性：原始要求を完全に満たしていること．

④一貫性(無矛盾性)：記述された内容に内部矛盾がないこと．

⑤完全性：すべての処理条件が記述されていること．

⑥最小性：冗長な記述がないこと．

仕様の作成を支援する仕様作成支援技術は，このような要求条件を満たす仕様の作成を支援するために必要な技術である．具体的には，

(1)あいまいな原始要求を分析し具体化する(要求)分析技術．

(2)具体化された要求仕様を実際に表現する仕様記述技術．

(3)記述された仕様に誤りや冗長があった場合，それらを検出し除去するための検証・最適化処理，部分的に記述された仕様を一つにまとめる合成処理等の仕様処理技術

等が考えられる．これらの仕様作成支援技術と仕様に対する要件との関係を図３．３．１に示す．

以上の仕様作成支援技術は，通信ソフトウェアの開発・保守における生産性と品質の向上にいずれも重要である．本論文では，それらの中でも特に重要な仕様検証技術を検討の対象としている．仕様検証技術が特に重要である理由は，図３．２．１からも分かるように，仕様は，それ以降の設計～運用に到る全開発工程及び運用開始後の全保守工程における諸作業の基本であり，もし仕様に誤りが残っていれば，それ以降の作業に完全な無駄と手戻りが発生するからである．そして，このような誤りの検出が遅くなるほど，その修正にはより多くの時間とコストが必要になるからである．

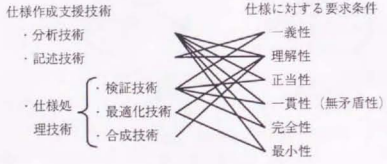


図３．３．１　仕様作成支援技術と仕様に対する要件の対応

３．４　通信ソフトウェア仕様検証のあり方

３．４．１　仕様誤りと検証

仕様検証は，与えられた仕様に誤りがあった場合，それを検出することである．従って，検証を論じるためには，検出対象となる誤りを明確にする必要がある．表３．４．１に示す通り，誤りは，構文誤り，論理誤り，意味誤りに分類することができる[1],[6],[108]．

ただし，ここでは，以下の前提を設けた．

①その通信ソフトウェアが持つ社会的な価値や影響等，技術的に評価することが困難な仕様誤りは対象外とする．

②前節で示した仕様要件のうち，正しさに関するもの，即ち一義性，正当性，一貫性及び完全性を劣化させる要因をすべて仕様誤りと考える．

要求仕様の一義性は，それを記述する言語に依存する．即ち，仕様記述言語があいまいさのないように定義されていれば，その言語に従って正しく記述した仕様については，その一義性が保証される．従って，あいまいさのない仕様記述言語の使用を前提にすると，一義性を劣化させる仕様誤りは，記述された仕様がその仕様記述言語の構文規則(文法)を満たすか否かを検査することによって検出できる．

検証技術の面からは，誤りであることを判定するための基準が予め既知であるか否かが大きな意味を持つ．これは，誤り判定基準が既知であれば，原理的には，与えられた仕様がその判定基準を満たすか否かを機械的に検査することによって誤りを検出できるからである．一方，誤り判定基準が既知でない場合には，このような機械的な検査は一般に困難となる．

一般に，一貫性と完全性は，それらを具体化することによって誤りの判定基準が明らかになる．例えば，「タイマのセットを実行した場合は，それ以降において必ずタイマのリセット処理を実行する必要がある」は，一貫性の例である．また，「入力条件に記述漏れがない」は，完全性の例である．そして，これらの誤り判定基準は明らかである．

一方，正当性については誤りの判定基準が一般には既知ではない．なぜなら，判定基準となるべき原始要求が通信ソフトウェア毎に固有で，必ずしも明確ではないからである．

本論文では，一貫性や完全性を劣化させる誤りを論理誤りと呼び，正当性を劣化させる誤りを意味誤りと呼ぶ．そして，論理誤り・意味誤りを検出することをそれぞれ論理検証・意味検証と呼ぶ．

以下では，技術的に困難な論理検証と意味検証について論じる．

３．４．２　仕様の論理検証

論理誤りは，その判定基準が明確に定義されるため，原理的には機械的な検出が可能である．本論文では，通信ソフトウェアを，次の３条件を満たす複数のプロセスによってモデル化する．

①各プロセスの動作は有限状態機械で表現される．

②各プロセスは，他プロセスから信号を受信することによって状態が遷移する．

③各プロセスは，状態の遷移に伴って必要な処理(含信号送出)を実行する．

以下では，このモデルを拡張有限状態機械 EFSM(Extended Finite State Machine)と呼ぶ．

本論文では「プロセス」という用語を頻繁に用いるが，情報処理の分野では，コンピュータ上での実際のプログラムの一連の制御の流れを意味し，各“プロセス”は他“プロセス”とは独立した動作をとるものとしている[3]．このような意味で「プロセス」を用いる場合には“プロセス”

と記述することにする．しかし，本論文での「プロセス」は，原則として，このような実装上の物理的なものを表すのではなく，実装の前の段階でそのベースとなる仕様を作成するために用いることを目的として，抽象化されたモデルとしての論理的な機能単位を表すものとする．即ち，本論文でいうプロセスを実際の通信システムにおいて実装する場合，ソフトウェアでなく，ハードウェアで実現してもよい．そして，ソフトウェアで実現するときは，複数の“プロセス”で実現してもよいし，逆に複数のプロセスをまとめて1個の“プロセス”で実現してもよい．換言すると，本論文では，通信ソフトウェアの仕様を抽象的な拡張有限状態機械を利用して規定することにしており，通信ソフトウェア工学やプロトコル工学における慣習に従って，そのような機械をプロセスと呼んでいる[1],[12]．

本論文の第4章では通信プロトコル仕様の論理検証（プロトコル検証）を論じるが，そこではプロトコルを司る複数のプロセスによってプロトコルを形式的に定義する．通信においては様々な信号（情報）の授受が行われるが，通常各信号を自由に送受できるわけではなく，何らかの規定（制約）があり，ある信号の送受はそれまでに授受してきた信号の履歴に依存する．このような規定がプロトコルである．一般に拡張有限状態機械によって，事象の履歴に応じた動作を容易に規定することができる．このため，信号の送受を事象に対応させることによって，拡張有限状態機械に基づく複数個のプロセスで，過去の動作の履歴に依存する動作を持つプロトコルを適切に表現することができる．

実際，電話用交換システム，INMARSAT 海事衛星通信システム，パケット交換システム，ISDN 通信端末等実際の通信ソフトウェアや通信プロトコルの仕様の記述では，このような拡張有限状態機械によるモデル化を利用し，そのような機械をプロセスと呼んでいることが多い[18],[109]~[111]．

このような定義によるプロトコルでは，あるプロセスが表す拡張有限状態機械は，他のプロセスから見たときの動作仕様を表しているが，実装の仕方を規定するものではない．しかし，あるプロトコルについて検証を行った場合，そのプロトコルに従って実装された通信システムのプロトコルを間接的に検証したことになる．ここで，プロトコルに従って実装された通信システムとは，その通信システムがプロトコルの各プロセスに対応する部分を持っていて，それらの外部に対する動作が，プロセスが表す拡張有限状態機械の動作と一致することを言う．このような一致を達成するための有力な方法の一つは，仕様からプログラムを自動的に作成する技術を応用することである[67]．

拡張有限状態機械モデルでは，通信ソフトウェアの処理は状態遷移に伴って時系列的に実行される．従って，論理誤りを判定するための基準は，状態あるいは処理の時系列上の実行可能性（到達可能性）及び実行順序に関する論理規則と考えることができる．例えば，すべての状態及び処理が時系列上で連続的に実行可能であること，各状態で記述された信号入力に記述漏れがないこと，ある処理を実行する前に必ず他の処理を実行すること等が誤り判定のための論理規則の例となる．

論理検証は与えられた仕様がこのような規則を満たすか否かを逐一検査することによって実

現でき，このような処理は一般に計算機で自動処理することが可能である．論理規則は予め分かっていると先に述べたが，一般に論理規則は経験によって得られるものである．従って，有用な論理規則が得られた場合知識ベースに蓄えておき，他の通信ソフトウェアの開発や保守でも容易に再利用できるようにしておくことが重要である．同時に，論理検証法では，このような論理規則の追加に柔軟に対応できることが望ましい．

3.4.3 仕様の意味検証

意味誤りについては，誤り判定基準が個々の通信システムに依存するため，機械的な検出が一般に困難であり，技術課題も多い．従って，どのような通信システムに対しても適用でき，かつ，意味誤りをすべて機械的に検出できる普遍的な意味検証法を確立することは容易でなく，要求者との対話等人手によらざるを得ない部分があると考えられる．

意味検証法は，大きく次の3つのアプローチに分類できる．

①仕様をプログラムに変換して又はインタプリティブに解釈して実行し，その結果を検査する方法．実行方法には，プログラムと同様にテストデータを用いた実行，変数等をそのまま処理するシンボリック実行等がある．

②なんらかの着目点に従って仕様を編集加工して，レビューを容易にする方法．具体的には，別の表現への変換，仕様の一部分の抽出，指定された関係や条件を満たす仕様要素群の出力等が考えられる．

③もとの要求をなんらかの形式で具体的な表明として表現し，その表明が成立するか否かを検査する方法．

しかし，一般に，仕様を実行させるために多くのデータ類を準備する手間が大きい，仕様の規模が大きく複雑であることが多いため部分抽出のための着目点が必ずしも明らかでない，もとの原始要求が必ずしも明確でない，等の問題があり，意味検証は一般に困難である．上述の3アプローチは，いずれも原始要求の一部のみを検査し確認するもので，単独で完全な意味検証を行うことは困難であり，相互に補完すべきものと考えられる[1]~[3]．

表3．4．1　仕様誤りの分類

仕様誤り	誤りの定義	誤りの性質	誤りの例	関連する仕様要件
構文誤り	・仕様を記述する言語の文法に対する違反	・記述方法／言語に依存 ・構文誤りを含む仕様の計算機処理は不可能	○文法誤り	・一義性
論理誤り	・誤り判定基準が予め明確に定義でき，多くの通信システムに共通なもの（除構文誤り）	・記述方法／言語に独立 ・仕様からのみで誤り検出が可能	○内部矛盾 ・信号の入出力（送受信）の対応不一致 ・手続き呼び出しのパラメータの不一致 ・信号・データ等の２重宣言 ○不完全誤り ・宣言されたが使用されないデータ，信号等 ・宣言されずに使用されるデータ，信号等 ・値の設定前に参照されるデータ，信号等 ・行き先が不明な信号出力（送信） ○実行不可能誤り ・デッドロック ・実行条件の漏れ ・分岐条件の漏れ ・実行アクセス不可能な仕様部分	・一貫性 ・完全性
意味誤り	・誤り判定基準が個々の通信システムに固有で，予め明確には定義できないもの	・記述方法／言語に独立 ・仕様からのみでは誤り検出は不可能	・要求と異なる機能 ・不十分な性能 ・使い勝手の悪さ	・正当性

3．5　むすび

本章では，まず通信ソフトウェアのライフサイクルモデルを提示し，通信ソフトウェアの生産性と品質の向上に対し，仕様検証が極めて大きな意義を持つことを明確にした．また，仕様に対する要求条件を論じ，要求条件のうち，一義性・正当性・一貫性・完全性を劣化させる要因を仕様誤りと位置づけ，そのような仕様誤りを，誤りであることの判定基準が既知であるか否か等の観点から整理し，構文誤り・論理誤り・意味誤りに分類した．また，誤り判定基準は誤りの検出方法に密接に関係しており，この分類に応じて，それらを検出する検証技術の基本原理を明らかにした．

これらの結果は，第４章と第５章で論じる仕様検証効率向上技術の基礎をなすものである．

第４章　通信プロトコル仕様の論理検証

4．1　まえがき

通信ソフトウェアにおいて，通信プロトコルは大きな役割を持つ．そして，一般にプロトコルは規模が大きくかつ複雑であることが多く，その仕様に誤りが含まれていることが少なくない．このような通信プロトコル仕様に含まれる論理誤りを検出するプロトコル論理検証(プロトコル検証)については，これまでの研究によって多くの方法が考案され開発されてきた．しかし，２．２で述べた通り，これまでに開発されたプロトコル検証法では検証効率が低く，実用的な規模のプロトコルを正確に検証することは困難であった．

本章では，この問題を解決することを目的として，プロトコル検証の効率を向上する新しい技術を提案し論じる．具体的には，まず４．２で，プロトコル検証の定義を明らかにする．次に，４．３で，検証すべきプロトコルの規模を削減することによって検証効率を向上する方法を論じる．また，４．４～４．６の各節で，検証処理自体の効率を向上できる新しいプロトコル検証法をそれぞれ提示し，その有効性を論じる．

4．2　通信プロトコル仕様のモデルと検証

本論文では，３．４．２で述べた通り，通信プロトコル仕様に従って通信する機能の実体をプロセスと呼び，信号を送受することによって状態を遷移していく２個以上のプロセスでプロトコルをモデル化し，各プロセスを拡張有限状態機械でモデル化する．

本論文で扱うプロトコルのモデルの形式的な定義は次の通りである．

［定義４．２．１］（プロトコル）

プロトコルを，Q_i，o_i，M_{ij}，succ（$1 \leq i, j \leq N$）を要素とする４項組$<<Q_i>, <o_i>, <M_{ij}>, succ>$で定義する．ここで，Nはプロセスの個数，$Q_i$はプロセスｉの状態の集合，$o_i$はプロセスｉの初期状態，$M_{ij}$はプロセスｉからプロセスｊに送る信号の集合であり，$M_{ii} = \phi$（空集合）とする．また，succは各プロセスが信号を送受することによって遷移する先の状態を表す関数で，一般に$succ(s_i, x)$は，プロセスｉが状態s_iにあるとき信号ｘの送信($x \in M_{ij}$，$i \neq j$)または受信($x \in M_{ji}$，$j \neq i$)によって遷移した後の状態を表す．

［定義終］

図４．２．１にプロセスが２個の場合のプロトコルの一例を示す．この図で，○は状態を表し，特に◎は初期状態を表す．状態間の矢印は遷移を表す．矢印に付けられたラベルは，符号が－のときは遷移に伴って送信する信号を表し，符号が＋のときは遷移に伴って受信する信号を表す．この図で，プロセス１の初期状態はs_0で，この状態において信号ａをプロセス２に送出すると状

態 s_1 に遷移する．その後，プロセス２から送出された信号 b を受信すると状態 s_2 に遷移する．そして，状態 s_2 で信号 c をプロセス２に送出すると状態 s_3 に遷移する．一方，プロセス２の初期状態は t_0 で，この状態において，信号 b をプロセス１に送出すると状態 t_1 に遷移し，信号 a 又は c をプロセス１から受信すればそれぞれ状態 t_2，t_1 に遷移する．そして，状態 t_1 において信号 a をプロセス１から受信すればやはり状態 t_2 に遷移する．

図４．２．１　プロトコルの例（初期状態＝s_0，t_0）

本プロトコルモデルでは，以下の仮定をおく．

①各プロセスが信号の送受によって遷移する先の状態は，その信号と元の状態に応じて一意に決まる．即ち，非決定的な遷移はない．

②各プロセスは相手プロセス別に受信用バッファを所有しており，あるプロセスが別のプロセスに信号を送信すると，この信号は直ちに該当するバッファに蓄積され，受信側プロセスの受信によってバッファからその信号が取り出される．このようなバッファの容量は有限であるが，十分大きくあふれることはない．

③各バッファは FIFO(First In First Out)型のキューであり，各バッファの信号の受信順序は送信順序と同一である．

④各プロセスは，空でないバッファを複数もつ場合，それらのうち任意のバッファの先頭の信号を受信することができる．

以下でプロトコル検証問題を形式的に定義するが，その前に，プロトコルに係わるシステム全体の状態を表すグローバル状態や定義済み遷移等の基本的な概念や用語を定義する．

［定義４．２．２］（グローバル状態）

プロトコルに従って通信するシステム全体のグローバル状態を，S と C の２項組<S, C>で定義する．ここで，$S=(s_1, s_2, \cdots, s_N)$，$C=(c_{11}, c_{12}, \cdots, c_{NN})$ であり，$s_i(1 \leq i \leq N)$ はプロセス i の状態，$c_{ij}(1 \leq j \leq N)$ はプロセス i から送信されたがバッファ内にあって未だプロセス j が受信していない信号の系列で，特に j=i の場合の $c_{ij}=c_{ii}=\phi$ とする．バッファがすべて空のグローバル状態を安定グローバル状態または単に安定状態と呼ぶ．特に，各プロセスの初期状態を含む安定状態を初期グローバル状態と呼び，G_0 で表す．また，グローバル状態相互の間の遷移をグローバル遷移と呼ぶ．

［定義終］

［定義４．２．３］（グローバル状態の到達可能性）

グローバル状態 G において，一つの信号を送信あるいは受信することによって別のグローバル状態 G′に遷移するとき，G と G′の関係を G⇒G′で表す．初期グローバル状態からこのような遷移を繰り返すことによって到達するグローバル状態 G，即ち $G_0 \Rightarrow \cdots \Rightarrow G$ を満たす G は到達可能であると定義する．

［定義終］

本論文では，既に“送信”と“受信”という用語を使ってきたが，関連する用語を含め以下にあらためて定義する．

［定義４．２．４］（送信／受信，定義済み送信／受信，定義済み遷移）

プロセス i（$1 \leq i \leq N$）の状態 s_i と信号 x から構成された順序対$<s_i, x>$を，$x \in M_{ij}(i \neq j)$であれば送信（遷移），$x \in M_{ji}(j \neq i)$であれば受信（遷移）と呼ぶ．与えられたプロトコルに $succ(s_i, x)$ が定義されているとき，$<s_i, x>$を定義済み送信／受信または定義済み遷移と呼ぶ．

［定義終］

定義済み送信$<s_i, x>$（$x \in M_{ij}, i \neq j$）は，s_i を要素として含む到達可能なグローバル状態が存在するとき実行可能である．また，定義済み受信$<s_i, x>$（$x \in M_{ji}, j \neq i$）は，s_i を要素として含みかつ該当するバッファの先頭に信号 x を蓄積している到達可能グローバル状態が存在するとき実行可能である．

以上の諸定義に基づき，本論文で扱うプロトコル検証問題を次の通り定義する．

［定義４．２．５］（プロトコル検証問題）

与えられたプロトコルにおいて，次に示す３種類の誤りをすべて検出することをプロトコル検証問題と定義する．

①信号定義の過剰：プロトコルに含まれているが実行不可能な送受信．（実行不可能送受信）

②信号定義の不足：プロトコルには含まれていないが実行可能な受信．（未定義受信）

③デッドロック：どのプロセスについても状態遷移が不可能である安定状態．

［定義終］

図４．２．２に，図４．２．１に示したプロトコル例におけるこれら３種類のプロトコル誤りを示す．例えば，プロセス１，２がそれぞれ初期状態 s_0，t_0 にあるとき，プロセス２が信号 b をプロセス１に送出すると，信号 b がプロセス１の受信バッファに蓄積されプロセス１はこの信号 b を受信することが可能となる．しかし，図４．２．１にはこの受信動作の記述がないため，プロセス１における信号 b の受信が未定義受信となる．図４．２．２ではこれを点線の矢印で示す．また，プロセス２が初期状態 t_0 で信号 c を受信すると状態 t_1 に遷移するよう記述されているが，この受信はあり得ない．なぜなら，図４．２．１によれば，プロセス１が信号 c を送出できるためには，プロセス１がその前にプロセス２に信号 a を送出しその後プロセス２から信号 b を受信する必要があるが，それにはプロセス２は少なくとも状態 t_1 に遷移していなければならないからである．

更に，両プロセスが初期状態にあるとき，プロセス１がプロセス２に信号ａを送出して状態 s_1 に遷移し，プロセス２がこの信号ａを受信して状態 t_2 に遷移したとする．このとき，両プロセスともに信号を送出することができないだけでなく，両受信バッファともに蓄積されている信号が存在しないため信号を受信することもできないため，デッドロックとなる．

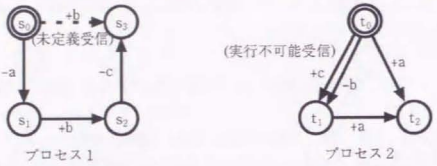


図４．２．２　図４．２．１のプロトコルの誤り（デッドロック＝(s_1, t_2)）

一般に人手で作成したプロトコル(仕様)は様々な誤りを含むことが少なくない．このような誤りのうち，機械的に検出することが可能で代表的な誤りが，定義４．２．５に示す論理誤りである．プロトコルは信号の送受に関する規約であり，例えば，一般に勝手に信号を送信したり受信したりすることはできないように規定される．つまり，ある時点で送受できる信号は特定のものに限られることになる．従って，あるプロセスが他プロセスに信号を送信すると規定してあれば，後者のプロセスではその信号を受信できるような記述がなされているべきである．そうでなければそのプロトコルは誤りと考えられる．このような誤りが定義４．２．５の未定義受信である．一方，プロトコルで規定されている送受信は当然実行できるように規定されているはずである．なぜなら，実行しない送受信を規定することは無駄であり，そのような規定はないと考えてよいからである．つまり，プロトコルの中の規定で実行できないものがあれば，それは誤りと考えられる．これが定義４．２．５の実行不可能送受信である．定義４．２．５のデッドロックがあった場合，プロトコル動作は停止し処理ができなくなってしまう．一般にこのようなデッドロックはあってはならないもので，これも誤りである．

このように，定義４．２．５で示した３種の誤りは，一般にどのような通信プロトコルにおいてもあってはならないものと考えられ，与えられたプロトコルにこれらの誤りがあった場合に検出する意義は大きい．特に，プロトコルの規模が大きくなると，これらの誤りを人手で検出することは指数関数的に困難となっていくため，コンピュータを使って効率よく検出できる検証技術が重要となる．

なお，デッドロックについては，定義４．２．５と異なり，実行可能な遷移を持たないグローバル状態と定義することもできる．この定義によるデッドロックのうち，安定状態であるデッドロックは定義４．２．５のデッドロックと同一であり，安定状態でないデッドロックは，定義４．２．５では②の未定義受信に含まれる．即ち，デッドロックの定義がこのように異なる場合でも，定

義４．２．５によればいずれかのプロトコル誤りとして検出されるので，定義４．２．５でも問題はない．

一般に定義４．２．５のプロトコル検証問題を解くアルゴリズムは必ずしも存在しないことが知られている．しかし，プロセス間のバッファに蓄積される信号数が有限であり，従って到達可能なグローバル状態の個数が有限なプロトコルについては検証法が存在する[20],[21]．そこで，本論文では，上述の仮定②で述べた通り，検証法が存在するこのようなプロトコルを検討の対象とする．

４．３　プロセスの削減によるプロトコル検証の効率向上

４．３．１　概要

一般に，プロトコル検証法においてはプロトコル規模，特にプロセス数が増大すると検証に必要となる処理量やメモリ量が指数関数的に増えていくという問題がある．そこで，本節では，与えられたプロトコルに含まれるプロセスを一部削除し，その結果プロセス数が減少したプロトコルに対して，既に確立しているプロトコル検証法または４．４以降で提示する新プロトコル検証法を適用することによって，もとのプロトコル全体の検証を行う方法を提示する[45]．

一般にプロセスを削減して検証した場合，①削除したプロセス自体の検証と，②残ったプロセスに含まれる信号送受信のうち，削除したプロセスとの間で送受する信号の扱いが問題となる．この２問題を解決するため，本論文では，送受信する相手プロセスが１個のみで，かつ規模が比較的小さくそれ自体の検証が人手でも困難でないプロセスのみの削除を考える．そのようなプロセスの例としては，タイマやリソースを管理するプロセス等がある．このうち，信号を送受する相手プロセスが１個に限定されているプロセスを内部プロセス又は子プロセスと呼び，相手プロセスを親プロセスと呼ぶ．また，両プロセスの間で送受する信号を内部イベントと呼ぶ．各子プロセスは親プロセスを１個のみ持ち，その親プロセスとのみ内部イベントを授受する．一般に，一つのプロセスは複数の子プロセスを持つことができる．親プロセス・子プロセスという概念は相対的なものであり，一般に子プロセスが別の子プロセスの親プロセスになることもある．

子プロセス削減によるプロトコル検証の基本方針は次の通りである．

①子プロセスを含む形で与えられたプロトコル仕様を，子プロセス及び内部イベントを全く含まず，それ以外の送受信の動作に関して等価なプロトコル仕様に変換する．（削除した子プロセス自体の検証は人手等で別途行うか，または，既に検証が別途完了済みであるものとする．）

②①の変換において，内部イベントの実行可能性を検査することによって，内部イベントに関する誤りを検出する．

③変換して得た等価なプロトコル仕様に，既に確立しているプロトコル検証法を適用する

ことによって，全送受信の実行可能性等を検査し，もとのプロトコルを検証する．

4.3.2 内部イベントと規則

例えば，タイマはいったんセットされると，それに伴って指定された時間を経過した後にタイムアウト信号を発生する．タイマは，タイムアウト信号を発生する以前にストップされると，セットされていない初期の状態に戻りタイムアウト信号を発生しなくなる．このようなタイマ機能を子プロセスとみなすと，セット，ストップ，およびタイムアウトの３つの内部イベントがある．タイマ機能を持つプロセスの仕様を図４．３．１に示す．図４．３．２は，このような内部イベントを含む（親）プロセス仕様の一例である．これらの内部イベントの間にはその実行順序に関して一定の制約があり，例えば，タイムアウトはセット後でないと実行されない．

一方，あるリソースを管理する機能は，リソースがその容量一杯に使用された状況に到ると，例えばリソース・ビジー信号を親プロセスに送出し，リソースに予め定めた量の余裕ができるとその旨通知するリソース・フリー信号を親プロセスに送出するという動作を行う．この二つの内部イベントには，それらが全く任意に発生するのではなく，必ず交互に発生するという制約がある．

タイマプロセス，リソース管理プロセスのいずれの例においても，送受信相手プロセスは１個に限られており，そのプロセスの子プロセスとなる．これら子プロセスの機能は比較的単純であり，これらは多くのプロトコル仕様に共通して使用され，その動作はプロトコル仕様には記述しないことも少なくない．また，これら子プロセスの例から分かるように，内部イベントは一般に全くランダムに送受されるものではなく，その順序にはある一定の制約がある．本論文では，このような制約を以下では単に規則と呼ぶ．

一般に，親プロセスの到達不可能な状態で定義されている内部イベントはすべて実行不可能である．親プロセスの到達可能な状態で定義されている内部イベントのうち内部送信イベントは常に実行可能であるが，内部受信イベントは必ずしも実行可能ではない．本論文では，親プロセスのある状態が到達可能で内部受信イベントを持てば，初期状態からその状態に到る親プロセスの信号系列に含まれる内部イベントの系列が規則を満たすか否かに従って，その内部受信イベントの実行可能性が判定できるものとする．つまり，到達可能状態で定義された内部受信イベントの実行可能性は，初期状態からその状態に到るすべての信号系列を検査することによって判定できるものとする．

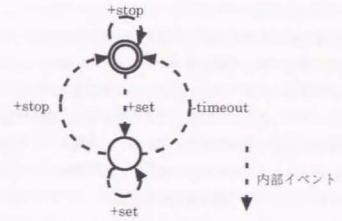


図４．３．１　タイマ処理用プロセス(子プロセス)

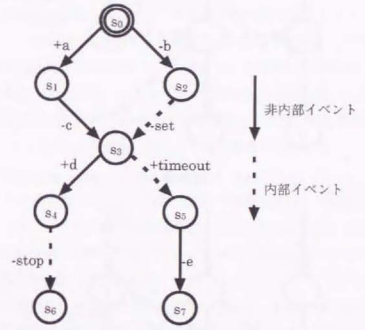


図４．３．２　タイマ機能に関連した内部イベントを含む(親)プロセスの例
（初期状態＝s_0）

4.3.3 プロセス削減検証法の原理

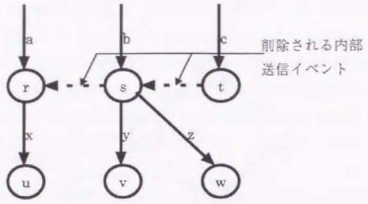
一般にタイマ機能やリソース管理機能等の各子プロセスが送受する内部イベントの種類は通常少なく，子プロセス自体の処理内容も単純である．従って，子プロセス仕様の作成は容易であり，子プロセス自体を検証する必要性は極めて小さい（実際に作成・記述しないこともある）．この考え方に基づき，以下では，与えられたプロトコル仕様に含まれている子プロセス仕様を削除して検証する方法を提示する．その基本方針は次の通りで，この検証法をプロセス削減検証法と呼

ぶ.

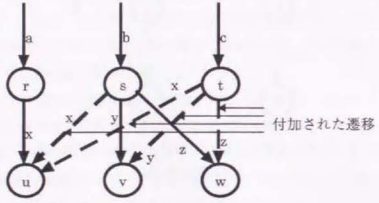
①内部イベントを含む形であたえられたプロトコル仕様を，内部イベントを全く含まず，それ以外の送受信の動作に関して等価なプロトコル仕様に変換する（この変換の結果，内部イベントのみから構成される子プロセスはすべて削除される）.

②この変換においては，内部イベント以外の送受信はすべて実行可能であるものとして，内部イベントの実行可能性を判定することによって，内部イベントに関する誤りを検出する.

③既に確立しているプロトコル検証法を，変換して得た等価なプロトコル仕様に適用し，内部イベントを含めた全送受信の実行可能性を検査し，もとのプロトコルを検証する.



(a) 内部送信イベント削除の前の仕様



(b) 内部送信イベント削除の後の仕様

図4.3.3 内部送信イベントの削除

子プロセスを削除する方法は自明であるので，以下では，内部イベントを含んだプロトコル仕様のうち，親プロセスから内部イベントを削除し，内部イベントを含まない等価なプロトコル仕様に自動的に変換する方法を主として示す．その基本原理は次の通りである.

(1)内部送信イベントの削除

内部送信イベントは，内部プロセスの仕様に依存せずその遷移元状態において常に実行可能である．そこで，図4.3.3の例に示す通り，各内部送信イベントをすべて削除し，削除した内部イベントの遷移元状態に対して，その内部イベントのみをその状態から実行することによって遷移可能だった各状態が持つ遷移（非内部イベント）と，ラベルおよび遷移先状態が同一の遷移をすべて付加する．また，削除した各内部イベントの遷移先状態に遷移する遷移が，削除した内部イベントの他になければ，この遷移先状態へはいずれの状態からも遷移不可能となるので，この遷移先状態とそこから出ている遷移をすべて削除する.

例えば，図4.3.3(a)の例では，削除すべき内部送信イベントは状態sとtにそれぞれ1個づつある．状態sが持つ内部送信イベントを削除する場合，その内部送信イベントを実行した後は状態rに遷移していたはずで，この状態rが持つ遷移を実行できたはずである．このような遷移は，この例ではラベルがxで遷移先が状態uの遷移1個のみであり，従って，図4.3.3(b)に示す通り，これと同等な遷移を状態sに付加する．一方，状態tが持つ内部送信イベントを削除する場合，その内部送信イベントを実行した後は状態sに遷移していたはずであり，この状態sが持つ遷移を実行できたはずである．このような遷移は，ラベルがyで遷移先が状態vの遷移とラベルがzで遷移先が状態wの遷移に加え，状態sが持っていて今削除した内部送信イベントを実行して状態rに遷移した後実行できた遷移，つまりラベルがxで遷移先が状態uの遷移がある．そこで，状態tについては，図4.3.3(b)に示す通り，これら3個の遷移を付加する.

(2)内部受信イベントの削除

内部受信イベントが実行できるためには，4.3.2で述べた内部イベントの実行に関する規則を満たす必要がある．例えば，タイマに関する内部受信イベントの一つタイムアウトは，セットという促進イベントを送出した後では実行可能であるが，セットを送出していない場合は実行不可能であり，両者の場合に分けて扱う必要がある.

このように，一般に各内部受信イベントが実行できるか否かは，その状態に到る遷移系列において実行した内部イベントに依存する．そこで，内部受信イベントを持つ状態は，その内部イベントが実行できるか否かに応じて二つの状態に分け，内部受信イベントが実行でき得る状態にのみその内部受信イベント以降の仕様を与え，その後，上述の(1)で述べた内部送信イベントの削除と同様な方法でその内部受信イベントを削除する．一般には，内部受信イベントを持つ状態だけでなく，それに到る前のすべての状態についても，内部イベントの実行状況に応じて複数の状態に分ける必要がある．内部受信イベントが複数種類あるときは，このような分割をすべての内部受信イベントについて行う．図4.3.4に，内部受信イベントの削除の例を示す.

以上の原理に基づき，内部受信イベントの実行可能性に応じて状態を分割するためには，初期状態から各状態に到るすべての遷移系列について内部イベントの実行状況を順次検査していき，その検査結果に応じてその状態を複数の状態に分割していけばよい．以下では簡単化のため，初期状態から各状態に到る系列に含まれる内部イベントの実行状況とそれに基づく内部イベントの実行可能性を検査した結果，例えば，セットを実行済みでタイムアウトが実行可能であること，

セット後更にタイムアウトを実行済みでタイムアウトが実行不可能であること等を単に検査結果と呼ぶことにする．本論文では，この検査結果の種類数は有限であると仮定する．

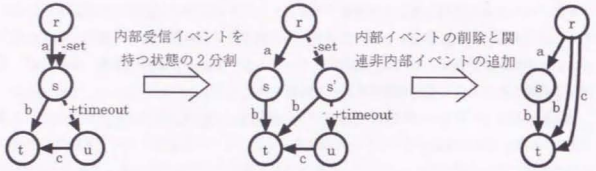


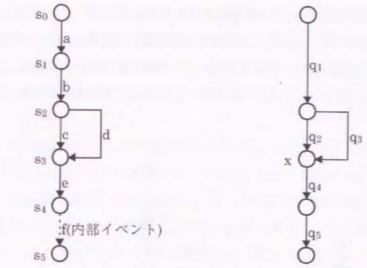
図４．３．４　内部送受信イベントの削除の例

本論文で提案するプロセス削減検証法では，与えられたプロトコル仕様に含まれる各プロセス仕様に対し，初期状態からの信号系列をすべて順次検査し，検査結果に応じて各状態を複数の状態に分割することによって，プロセス仕様を木状に展開していく．このような木状展開を機械的に実行する方法を簡明に示すため，以下では部分系列，実行系列グラフという概念を導入し，実行系列グラフの木状展開という形でプロセス仕様の木状展開を論じる．

プロセス仕様に含まれる送受信の実行順序に従った系列を実行系列と呼び，実行系列を，遷移の流れが分岐あるいは合流する状態，および内部受信イベントを持つ状態で分割したものを部分系列と呼ぶ．

例えば，図４．３．５(a)のプロセス仕様の部分系列を，状態と遷移の系列で表現すると，$[s_0(a)s_1(b)s_2]$，$[s_2(c)s_3]$，$[s_2(d)s_3]$，$[s_3(e)s_4]$，$[s_4(f)s_5]$となる．これらの部分系列をそれぞれq_1，q_2，q_3，q_4，q_5で表わすと，図４．３．５(a)のプロセス仕様は同図(b)のように表現することができる．このようにプロセス仕様を部分系列のみで表わした図を実行系列グラフと呼ぶ．図４．３．５(a)のプロセス仕様の実行系列は，同図(b)の実行系列グラフから，$[q_1, q_2, q_4, q_5]$および$[q_1, q_3, q_4, q_5]$となる．

プロセス仕様がループ状の実行の流れを含む場合は，実行系列の個数は無限となる．例えば，図４．３．６の実行系列グラフにおける実行系列は，$[q_1, q_2, q_4]$，$[q_1, q_2, q_3, q_2, q_4]$，…，$[q_1, (q_2, q_3)^{i-1}, q_2, q_4]$，…$(i=1, 2, \cdots)$となる．ここで，$(\ )^i$は，（　）内の系列のｉ回の繰り返しを表す．



(a)プロセス仕様　　(b)実行系列グラフ

図４．３．５　プロセス仕様とその実行系列グラフの例

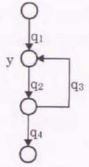


図４．３．６　ループを含む実行系列グラフの例

プロセス木状展開の基本は，与えられたプロセス仕様を，部分系列を単位に初期状態から検査すると同時に実行系列グラフとして木状に展開し，その結果もとのプロセスと等価な木を導出することにある．以下では，このような木導出のための木状展開アルゴリズムを示す．

［アルゴリズム４．３．１］（実行系列グラフの木状展開）

以下の手順で，実行系列グラフを木状に展開する．

step1　初期状態を表す頂点(根)を設定する．

step2　初期状態から始まる部分系列を表す枝を頂点にすべて付加する．

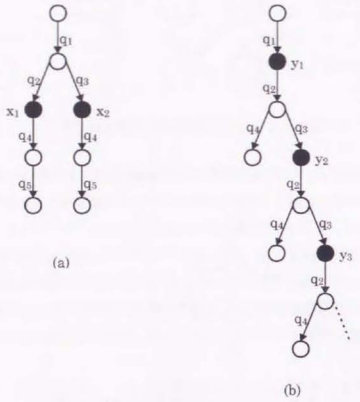
step3　作成された木における各端点(葉)に到る枝に対し，それに対応する部分系列に引き続く部分系列を表す枝をその端点にすべて付加する．

step4　step3による枝の付加を可能な限り繰り返す．

［アルゴリズム終］

実行系列グラフの木状展開によって得られるグラフを展開木と呼ぶことにすると，図４．３．５

(b)と図4．3．6に示した実行系列グラフの展開木はそれぞれ図4．3．7(a)と(b)になる．両実行系列グラフはいずれも実行の流れの合流点を1個ずつ含むが，展開木ではその合流点 x, y に対応する節点を●で表し，x_1, x_2, y_1, y_2, y_3…と記号を付けた．図4．3．7(a)で，節点 x_1 と x_2 以降の部分木は完全に同一である．図4．3．7(b)では，節点 y_i (i=1, 2, …) 以降の部分木がすべて同一である．



●：実行系列上での合流点

図4．3．7　図4．3．5(b)と図4．3．6の実行系列の展開木

以上から明らかなように，展開木の性質に関し以下の補題が成立する．

[補題4．3．1]

展開木には以下の性質がある．

①各実行系列は，頂点から端点に到るパスで表される．

②実行系列グラフにおける実行の流れの合流点は，展開木では複数個の節点に対応する．一つの合流点 x に対応する節点を $\{x_i\}$ (i=1, 2, …) とすれば，節点 x_i 以降の部分木は i に依らずすべて等しい．

[補題終]

プロセス仕様の実行系列グラフの木状展開において，実行系列グラフ上の一つの合流点に対応

する状態を二度検査したとき検査結果が同一であればそれ以降の検査結果も同一となるので，それら二つの状態に対応する展開木の節点は一つに統合できる．つまり，展開木において異なる実行系列上にこのような二つの節点が存在する場合は，それらの節点を一つに統合した後，以降を引き続き検査し木状展開すれば十分である．ただし，このような節点の統合処理の結果，展開木は完全な木ではなくなる．

内部イベントの削除は，このような部分系列を単位としたプロセス仕様の木状展開と並行して実行することとする．部分系列が内部イベントを含む場合，部分系列の定義によって，内部イベントは1個で，その部分系列の先頭に位置し，部分系列の中では遷移の流れの分岐や合流はない．従って，先に述べた内部イベント削除の原理(1)と(2)に基づいて部分系列内の内部イベントを削除すると，削除した内部イベントの遷移先状態へはいずれの状態からも遷移不可能となるので，この遷移先状態も削除し，その状態が持つ遷移の遷移先状態を，削除した内部イベントの遷移元状態，つまり，部分系列の最初の状態に変更する．ただし，部分系列が内部イベントのみから構成される場合，つまり，内部イベントの遷移先状態が部分系列の最後の状態に等しい場合には，その内部イベントとその遷移先状態が削除されることによって，木状展開して付加した部分系列全体が削除される．このとき，その部分系列をいったん付加した展開木上の節点においては，プロセス仕様上でその部分系列に続く部分系列も引き続き付加して木状展開する対象とする必要がある．

4．3．4　プロセス削減のためのプロセス木状展開

上述した実行系列グラフの木状展開を応用して，与えられたプロセス仕様における状態を，内部イベントの実行可能性に応じて順次分割していく「プロセス仕様の木状展開」のアルゴリズムは次の通りである．

[アルゴリズム4．3．2]（プロセス仕様の木状展開）

以下のステップに従って，与えられたプロトコル仕様に含まれる各プロセス仕様を木状展開する．

step1　与えられたプロセス仕様を，部分系列ごとに分解する．

step2　与えられたプロセス仕様の初期状態を，木状展開プロセス仕様の初期値(初期状態)とする．

step3　もとのプロセス仕様において，初期状態から始まる部分系列のうち実行可能な内部送信イベントまたは送信から始まるものを，未処理部分系列の集合Σにすべて登録する．ただし，これら各部分系列の最初の状態(初期状態)における検査結果を，内部イベントを一切実行していないという初期値に設定する．

step4　Σより任意の未処理部分系列を1個抽出してqとし，これを木状展開プロセス仕様上に付加する．部分系列qの最初の状態における検査結果をもとに，qの最後の状態における検査結果を導出してW(q)とする．

step5　木状展開プロセス仕様において部分系列 q に内部送信イベントが含まれる場合，その内部イベントとその遷移先状態をすべて削除する．更に，削除した内部イベントの遷移先状態が部分系列 q の最後の状態に等しい場合は，木状展開プロセス仕様上でその部分系列 q をいったん付加した(step4)節点において，もとのプロセス仕様上で部分系列 q に続くすべての部分系列を付加し木状展開できるものとする．また，削除した内部イベントの遷移先状態が部分系列 q の最後の状態に等しくない場合は，削除した内部イベントの遷移先状態が持っていた遷移の遷移元状態を，削除した内部イベントの遷移元状態に変更する．

step6　部分系列 q の最後の状態に到る別の部分系列 q'が木状展開プロセス仕様上に既に存在して W(q)＝W(q')を満たすならば，木状展開プロセス仕様上で二つの部分系列 q，q'の最後の状態を一つの状態に統一して step8 に進む．このような q'が存在しないならば step7 へ進む．

step7　もとのプロセス仕様において部分系列 q に続く部分系列のうち，実行不可能な内部受信イベントから始まるものを除き，Σに加える．

step8　Σに未処理部分系列が残っているか否か検査し，もし残っていれば step4 に戻る．否のときは終了する．

［アルゴリズム終］

本アルゴリズムにおいて各部分系列を検査した結果，実行可能であると判定された内部受信イベントが定義されていないとき，内部イベントに関する信号定義の不足（未定義受信）とする．また，木状展開されたプロトコル仕様に含まれない内部送受信イベントは実行不可能であり，信号定義の過剰となる．

図４．３．７(a)では，節点 x_1 での検査結果と節点 x_2 での検査結果が同一であれば，例えば，節点 x_1 を節点 x_2 に統合し部分系列 q_4 以降は１回のみ木状展開する．また図４．３．７(b)では，例えば節点 y_2 と y_1 の検査結果が同一であれば，節点 y_2 以下の部分木は不要でそれ以降の木状展開を行う必要はなく，y_2 を y_1 に統合する．節点 y_2 と y_1 の検査結果が異なる場合でも，節点 y_3 の検査結果が節点 y_1 または y_2 の検査データと一致していれば，節点 y_3 以下の木状展開は不要となり，節点 y_3 を y_1 または y_2 に統合する．検査結果のとり得る値の組合せは有限と仮定したため，節点 y_1，y_2，y_3…の検査結果がすべて異なるということはあり得ず，この木状展開は必ず有限時間で停止する．

本アルゴリズムに関して，以下の定理が成立する．

［定理４．３．１］（アルゴリズム４．３．２の正当性）

内部イベントを含むプロトコル仕様が（必要であれば子プロセスを陽に記述しそれに追加することによって）検証可能であれば，もとのプロトコルにアルゴリズム４．３．２を適用して得られるプロセス仕様木状展開結果を検証することによって，もとのプロトコルを正しく検証することが可能である．即ち，定義４．２．５のプロトコル誤りをすべて検出することが可能で，この処理は必ず有限時間で停止する．

［定理終］

［定理４．３．１の証明］

まず，アルゴリズムの停止性について述べる．木状展開アルゴリズム４．３．２はΣ内の未処理部分系列が０になったとき停止する．一般に，Σ内の未処理部分系列は，step6 において W(q)＝W(q')を満たす部分系列 q'が存在しないため step7 を実行したとき有限個数だけ増加していく．しかし，与えられたプロセス仕様の部分系列の個数は有限であるため，q の最後の状態に到る別の部分系列 q'が存在しないという条件が無限に続くことはない．また，検査結果のとる値の組合せは有限であると仮定したため W(q)≠W(q')が無限に続くことはあり得ない．従って，Σ内の未処理部分系列は有限時間で必ず０個となり，この木状展開処理は必ず停止し，アルゴリズム４．３．２も停止する．

次に，定義４．２．５の誤りを有限時間で漏れなく検出できることを示す．既に述べた通り，木状展開されたプロセス仕様はもとのプロセス仕様に比較して，内部イベントの実行可能性に基づいて各状態と遷移を分割したものであり，従って，内部イベントを除きもとの実行系列をすべて含むが，もとのプロセス仕様に含まれていなかった余分な実行系列を含むことはない．更に，もとのプロトコル仕様のグローバル状態数が有限であれば，木状展開されたプロトコル仕様のグローバル状態数も有限となり，検証処理の停止性が保証され，プロトコル検証が可能となる．ただし，実行不可能送受信については，一般にもとのプロセス仕様の一つの状態が木状展開プロセス仕様では複数の状態に木状展開されるので，それら木状展開された状態全体を通して全く実行不可能であるもののみを最終的に実行不可能と判定する必要がある．

［証明終］

例えば，図４．３．２のプロトコル例に対し，アルゴリズム４．３．２を適用した結果は図４．３．８となる．即ち，状態 s_3 は二つの状態 s_3'と s_3''とに分かれ，タイマのセットを経ないで状態 s_3 に到る実行系列 $s_0 \to s_1 \to s_3$ は s_3'に，またセットを経て状態 s_3 に到る実行系列 $s_0 \to s_2 \to s_3$ は s_3''に到る．s_3'にはタイムアウト処理以降の実行系列を与えない．内部イベントをすべて削除することにより最終結果として図４．３．８が得られる．なお，図４．３．２で，例えば，$s_3 \to s_5 \to s_7$ の遷移がなかった場合は，アルゴリズム４．３．２の適用によって，図４．３．８の状態 s_3''において，つまりもとの状態 s_3 において実行可能なタイムアウト受信の遷移が不足であることを検出できる．

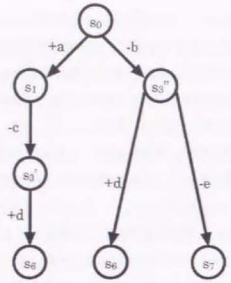


図4．3．8　図4．3．2のプロセスの木状展開結果

4．3．5　プロトコル削減検証法の評価

以上で提示した検証法は，与えられたプロトコル仕様から子プロセスを削除して，与えられたプロセス仕様をプロセスごとに木状展開し，その結果に対して従来から提案されているプロトコル検証法を適用するものである．ここで適用するプロトコル検証法を基本検証法と呼ぶことにする．プロセス木状展開を適用しないで，つまり，基本検証法をプロトコル仕様に直接適用してプロトコル検証を行う方法をここでは従来法と呼ぶことにする．

本項では，検証効率について提案検証法と従来法との比較評価を行う．評価の尺度としては，検証処理量（時間）やメモリ量が考えられるが，提案法・従来法ともに同一の基本検証法を適用することにすれば，一般に提案法によって検証処理量を削減できれば所要メモリ量も削減できる傾向にある．そこで，以下では検証処理量についてのみ論じることとする．

（1）　検証処理量の削減の定性的評価

ここでは，基本検証法には，広く知られている到達可能木解析法，つまり，初期グローバル状態 G_0 から出発して実行可能なプロセス遷移を逐一すべて実行していくことによってグローバル状態とグローバル遷移をすべて導出していくという方法をとりあげ，これを適用した場合の提案検証法の処理量について考察する．

従来法に比較して提案法ではプロセス木状展開処理部分が多いが，この処理はプロセスごとに実行するもので処理量は一般に少ない．特にプロセス数が多いような大規模プロトコルに対してはその処理量は基本検証法の処理量に比較すると十分少ない．そこで，以下では基本検証法の処

理量についてのみ考察する．

与えられたプロトコルのグローバル状態とグローバル遷移の総数をそれぞれ N_s，N_t とすると，到達可能木解析法の検証処理量は，$C=N_t\cdot\log_2(N_s!)/N_s$ に比例するものと考えることができる（次節4．4参照）ので，以下ではこのCを検証処理量と考える．

提案検証法は，従来法に比較して，以下の①～③の特性を持っている．

①子プロセスが記述されている場合はそれらが削除されるため，プロセス数Nが小さくなる．

②各プロセスに含まれる内部イベントがすべて削除され，これに伴っていずれの状態からも遷移が不可能となるプロセス状態が発生しこれを削除できることがある．更に，従来法では検証用の特別な遷移を余分に追加することが必要な場合があるが[45]，提案法ではそのような遷移の追加はない．これらは，従来法に比較してプロセス遷移数とプロセス状態数を減少させる効果を持つ．

③内部イベントの削除に伴って，非内部イベント遷移が追加されることがある．また，内部受信イベントの実行可能性に応じて状態が分割されるため，状態が多くなるとともに，遷移も増加する．これらは，従来法に比較してプロセス遷移数とプロセス状態数を増加させる効果を持つ．

以上の特性のうち，①によって，従来法に比較して，プロセス総数が子プロセスの個数分だけ減少し，NからN'（N'<N）に変化するものとする．また，②と③によって，一般にプロセス i（$i=1,2,\cdots,N$）の状態数と遷移数がそれぞれ u_i，v_i 倍に変化するものとする．

一般に内部イベントを含むプロトコルに対しては，$N'<N$ が常に成立するので，$u_i\leq 1, v_i\leq 1$（$i=1,2,\cdots,N$）が成立し，プロセス木状展開によって各非子プロセスの状態と遷移が増加せずプロトコル規模が拡大しなければ，各プロセスの状態と各信号受信バッファ内信号系列の組み合わせで定義されるグローバル状態，および，各グローバル状態が持つ実効可能なプロセス遷移であるグローバル遷移の総数は一般に減少する．つまり，従来法におけるグローバル状態とグローバル遷移の総数をそれぞれ N_s，N_t とし，提案法におけるグローバル状態とグローバル遷移の総数をそれぞれ N_s'，N_t' とすれば，$N_s'<N_s$，$N_t'<N_t$ となり，提案法の検証処理量C'は従来法の検証処理量Cに比較して小さくなる．

しかしながら，一般には $u_i\leq 1$，$v_i\leq 1$（$i=1,2,\cdots,N$）は必ずしも成立するとは限らない．両不等式が成立する例として，タイマ処理について考察する．タイマに関する仕様では，図4．3．2において $s_0\rightarrow s_1\rightarrow s_2$ の実行系列部分を削除したもの，即ち，

①タイムアウト処理（$s_3\rightarrow s_5$）を持つ状態（s_3）に遷移するときは必ずそのタイムアウトに対応するタイマのセット処理（$s_2\rightarrow s_3$）が実行されており，

②その状態（s_3）において，タイムアウト以外の信号を受信した場合の処理（$s_3\rightarrow s_4$）が実行されれば，その後にそのタイマのストップ処理（$s_4\rightarrow s_8$）が必ず実行され，更に，

③これら各内部イベントを実行した後の状態（s_2，s_5，s_8）が持つ遷移の個数が高々1であるか，または，その状態に遷移する遷移がその内部イベント以外に存在しない．

という3条件を満たすよう規定されていることが多い．このような仕様をプロセス木状展開すると，タイマのセット，ストップ，タイムアウトの各内部イベントがすべて削除されて遷移が減少するが，これに伴って状態が増えることはなく，$u_i \leq 1$，$v_i \leq 1 (i=1,2,\cdots,N)$となる．その木状展開結果は，図4．3．8の右半分のみ（$S_0 \rightarrow S_3'' \rightarrow \cdots$）となる．一般に，与えられたプロトコル仕様に含まれるこのようなタイマ処理部分に対してのみプロセス木状展開法を適用することによって，プロトコル規模を確実に縮小させることが可能で，検証処理量を削減することができる．

実際，このようなタイマ処理は実用されているプロトコル，例えば，ITU-T勧告のISDN用ユーザ網インタフェース・レイヤ3プロトコル，ITU-T勧告の共通線信号方式No.7の電話ユーザ部TUP(Telephone User Part)やトランザクション機能部TCAP (Transaction Capability Part)等に含まれており，その個数は少なくない．この検証処理量削減効果の具体的な定量評価については，次項で論じる．

以上の説明から理解できるように，一般に，与えられたプロトコルにおける子プロセスと内部イベントが多いほど，削減されるプロセスとプロセス状態やプロセス遷移の個数が多くなり，提案検証法の検証処理量削減効果は大きくなる．

（2） 検証処理量の削減の定量的評価

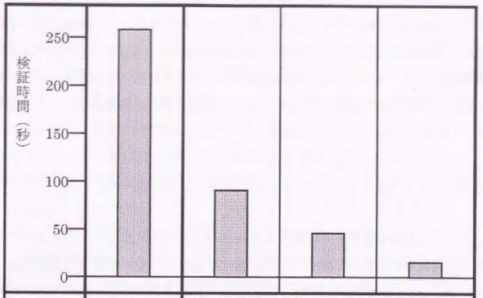
提案検証法の実用的効果を評価するため，プロセス木状展開アルゴリズムを実現するツールを作成し，実験評価を行った．この実験では，基本検証法として，到達可能木解析法よりも一般に効率が優れているアサイクリック展開法（4．5参照）を採用し，これに基づく検証ツールとプロセス木状展開ツールとを組み合わせ様々なプロトコル仕様の検証を行った．その結果の一部を表4．3．1に示す．表4．3．1の検証で用いたプロトコルは，ITU-T勧告Q.931に基づくISDN用ユーザ網インタフェースのレイヤ3プロトコルで，勧告にある呼制御機能のうち，処理状態の問い合わせ(status inquiry)処理，呼設定信号(setup)の分割転送処理(overlapping)，および，データリンク管理処理の各機能を除外している．このプロトコルは，ユーザ側の端末処理プロセス，網側のプロトコル処理プロセスと呼処理プロセスの計3個のプロセスから構成されている．このうち，端末処理プロセスとプロトコル処理プロセスにはそれぞれ5個のタイマが含まれている．

検証に使用した計算機は，処理能力が10MIPS程度のワークステーションであり，検証を効率よく進めるため，各プロセスが初期状態を遷移して他の状態に移った後初期状態に戻るように遷移してきた場合には，それ以降の部分については遷移を実行しないことにした．

従来法においては，例えば端末処理プロセスは5個のタイマプロセスを含むが，これらタイマプロセスは同時にセットされることがないため，それらをまとめて1個のタイマプロセスとして共用することができる．プロトコル処理プロセスについても同様である．実際，このように各非子プロセスごとにそれに含まれるタイマプロセスを1個にまとめることによってプロセス総数は13から5に減少し，検証処理時間を280秒から258秒に削減できたので，表4．3．1ではタイマ

プロセスをこのようにまとめた場合の結果を示している．

表4．3．1 プロセス削減検証法による検証時間の削減効果
（＊，＃：ユーザ側端末処理プロセス，網側プロトコル処理プロセス，網側呼処理プロセスの状態数（＊）と遷移数（＃）を表す）



検証法	従来法	本論文による方法(プロセス削減法の適用)		
		1プロセスにのみ適用	別の1プロセスにのみ適用	全(2)プロセスに適用
プロセスの個数	5	4	4	3
各プロセスの状態数と遷移数	47,135,26＊ 78,235,53＃	47,111,26＊ 78,220,53＃	24,135,26＊ 54,235,53＃	24,111,26＊ 54,220,53＃

一方，提案検証法については，タイマ機能を含む2個のプロセスのうち一方のみにプロセス木状展開を適用した場合，両プロセスにプロセス木状展開を適用した場合の計3ケースについての検証結果を示す．3ケースのうち，最初の2ケースでは，タイマプロセスをやはり1個にまとめている．なお，プロセス木状展開ツールには，各プロセス仕様に対して，予め決められた形式（文法）を満たしているか，すべての初期状態からたどれるように遷移が定義されているか，送信のみによるループ状遷移があるため検証不可能でないか等の論理検査を行う機能や，冗長な状態と遷移を除去する最適化機能，例えば，2個の状態が持つ（複数個の）遷移が完全に同一な場合，即ち，それら遷移のラベルと遷移先状態がそれぞれ同一の場合，この2個の等価な状態を1個の

状態に統合し，同一の遷移をそれぞれ統一する機能も含まれている．しかし，これらの機能の処理時間を含めプロセス木状展開処理時間は常にほぼ一定で約1分程度であったため，表4．3．1の検証処理時間は1分程度の処理時間を除いた値とした．

表4．3．1から分かるように，この例では，プロセス木状展開によって，各プロセスの状態数と遷移数はいずれも減少している．これは，プロセス木状展開によって状態が分割され状態と遷移が若干増加したが，各プロセスに含まれるタイマ処理のための内部イベントが多く，これらを削除することによって状態と遷移が大きく減少したためである．その結果，従来法に比較して，提案法では一部のプロセスに対してのみプロセス木状展開を適用した場合でも検証処理時間が1/3～1/5 程度に小さくなっている．特に，全プロセスにプロセス木状展開を適用した場合には，従来法に比較して検証処理時間は16分の1以下になっており，本提案法による検証処理量削減効果は極めて大きい．

4．3．6　まとめ

本節では，プロトコル検証の効率向上を図るため，与えられたプロトコルから子プロセスを削除し，親プロセスの各状態を内部イベントの実行可能性に応じて複数個の状態に分け，その結果に基づいて内部イベントをすべて削除して検証するというプロセス削減検証法を提示した．そして，この検証法によってプロセス数や信号数を削減し効率よく検証できることを定性的に論じた後，実用されているプロトコルを対象とした検証実験の結果を示すことによって，従来法に比較して検証時間を大幅に削減できることを具体的に明らかにした．例えば，タイマプロセスを1個削減すれば検証処理量は1/3～1/5程度になり，タイマプロセスを2個削減すれば検証処理量は10数分の1になる．

一般に通信プロトコルではタイマ処理に基づく誤り監視・回復機構が必須であり，頻繁にタイマ処理が利用される．また，リソース管理に伴う内部イベントを使用することも少なくない．従って本論文で提示した方法は，このような子プロセスを持つ通信プロトコルの検証の効率向上に極めて有効である．

子プロセスはこれらタイマプロセスやリソース管理プロセスに限るわけではない．原理的には，

①信号（内部イベント）を授受する相手プロセス（親プロセス）が1個であること，かつ，

②子プロセス自体の検証が必要でない，

という2条件を満たすようなプロセスすべてに適用できる．従って，例えば，既に別途完成済みのプロセスが上述の条件①を満たせば，このようなプロセスすべてに本展開法を適用して削除することが可能となる．

本節は，内部イベント処理を含むプロトコル検証を行うための，いわば前処理法を提案したものであり，この前処理によって形式的には内部イベントを含まないプロトコルに変形した後，従来から提案されているプロトコル検証法を適用してプロトコルを検証するという考え方に基づいている．ここで適用できるプロトコル検証法については特別な制約がないため，本プロセス木状展開法は適用範囲が広いという特徴も持っている．

4．4　プロセス状態遷移の一括処理によるプロトコル検証の効率向上

4．4．1　概要

本節では，効率よくプロトコル検証を行うことが可能な新しい検証法を提案する．本提案検証法は，基本的には，従来法の一つである到達可能木解析法に基づくものであるが，プロトコル動作を表すグローバル状態遷移図を展開し作成していく際，誤り検出に支障のない範囲で，実行可能なプロセス遷移を可能な限り一括処理する[46],[47]．具体的には，次に示す3処理を行う．

①同一のグローバル状態に到る遷移は，できる限り1個となるよう削除する，

②同時に実行可能な遷移のうち誤り検出に支障のない遷移を必ず実行することによって遷移をできる限り一括する，

③誤り検出に支障がなければ送信遷移に対応する受信遷移も同時に実行する．

このように状態遷移をできる限りまとめて一括処理することによって，生成するグローバル状態遷移図に含まれる状態と遷移を最小限に留めて誤りの有無を検査することに，提案検証法の特徴がある．

4．4．2　プロセス状態遷移一括処理の原理

定義4．2．5のプロトコル検証問題は，一つの信号の送信または受信によるプロセス遷移を初期グローバル状態から逐次実行していき，到達可能なグローバル状態と実行可能なグローバル遷移をすべて含むようなグローバル状態遷移図を作成して検査するという方法（到達可能木解析法）によって解くことができる[20],[21],[22]．この方法を，本節では従来法とも呼ぶ．

従来法での誤り検出の具体的方法は次の通りである．まず，信号定義の過剰はグローバル状態遷移図を完成した後，それに含まれるプロセス遷移が実行可能なプロセス遷移となるので，もとのプロトコル仕様には含まれるがグローバル状態遷移図には含まれないプロセス遷移を実行不可能遷移として検出する．次に，信号定義の不足は，各グローバル状態で実行可能な受信をすべて導出し，もとのプロトコル仕様に含まれないものを未定義受信として検出する．更に，導出した安定グローバル状態のうち実行可能遷移を持たないものをデッドロックとして検出する．

これに対して，本節で提案するプロトコル検証法の原理は，各グローバル状態において，定義4．2．5で示した誤りを検出するのに支障ない範囲で，実行可能なプロセス遷移を最大限一括してまとめたものをグローバル遷移とし，これを用いてグローバル状態遷移図を作成して検査することにある．具体的には，各グローバル状態において，次に示す三つの処理を順次適用して実行

することによってグローバル遷移を導出する．

①直ちに実行可能なプロセス遷移の組み合わせによってグローバル遷移候補を導出し，その結果，各グローバル状態に到る遷移を１個のみとする．

②冗長なグローバル遷移候補を削除することによって，同時に実行可能な遷移のうち誤り検出に支障のないものを必ず実行することにする．

③グローバル遷移候補の中に送信が含まれている場合，誤り検出に支障がなければその送信に対応する受信遷移を一括処理してグローバル遷移を最終的に決定する．

そして，このようなグローバル遷移を初期グローバル状態から順次逐一実行し，必要最小限のグローバル情態遷移図を作成する．図４．４．２は，図４．４．１のプロトコル例に対して本検証法を適用した結果で，以下この具体例を用いてこれら３処理を詳しく説明する．

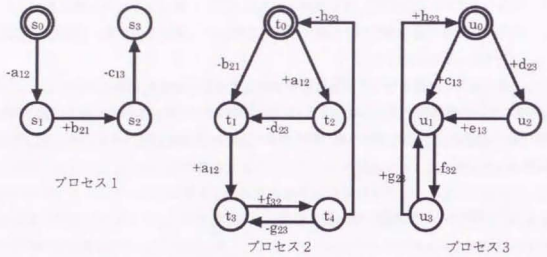


図４．４．１　プロトコルの例（初期状態＝s_0, t_0, u_0）

(1)グローバル遷移候補の導出

[定義４．４．１]（グローバル遷移候補の導出）

一つのグローバル状態において，直ちに実行可能なプロセス遷移が１個のみであれば，そのグローバル状態からのグローバル遷移はそのプロセス遷移のみとする．直ちに実行可能なプロセス遷移が複数個ある場合，このようなプロセス遷移のすべての組み合わせのうち，各プロセスで実行する遷移が高々１個であるような組み合わせすべてを，そのグローバル状態からのグローバル遷移とする．ここで，直ちに実行可能なプロセス遷移とは，信号の送信，および，受信バッファの先頭に蓄えられている信号の受信である．

このようにして導出されるグローバル遷移には，直ちに実行可能なプロセス遷移を持つプロセスの遷移を持たないものもある．このようなグローバル遷移を実行して得られるグローバル状態以降で更にグローバル遷移を導出する場合は，もとのグローバル状態でそのようなプロセスが持っていた直ちに実行可能なプロセス遷移を除外して導出することとする．

[定義終]

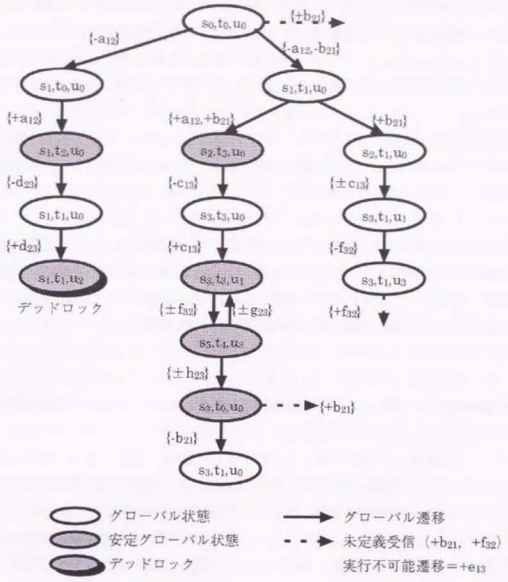


図４．４．２　図４．４．１のプロトコル例に状態遷移一括法を適用した結果

例えば，図４．４．１のプロトコル例で，初期グローバル状態(s_0, t_0, u_0)（本節では簡単化のため，グローバル状態の記述においてバッファ内の蓄積信号系列を省略する．）において直ちに実行可能なプロセス遷移は，プロセス１の$-a_{12}$およびプロセス２の$-b_{21}$である（ここで，遷移を表す$-a_{12}$等のサフィックスは，信号の送受信プロセスをそれぞれ表す）．これらの組み合わせによるグローバル遷移は，$|-a_{12}|$，$|-b_{21}|$，$|-a_{12}, -b_{21}|$の３通りあり，それぞれを実行すると，新グローバル状態として，(s_1, t_0, u_0)，(s_0, t_1, u_0)，(s_1, t_1, u_0)が得られる．実際には，次項で述べるように，これら３個のグローバル遷移のうち，$|-b_{21}|$は冗長であると判定されて削除され，生成され

るグローバル遷移・グローバル状態はともに2個となる．図4．4．2では，その最終結果を示している．

初期グローバル状態(s_0, t_0, u_0)からグローバル状態(s_1, t_0, u_0)へのグローバル遷移 $|-a_{12}|$ は，初期グローバル状態で直ちに実行可能な遷移$-b_{21}$を持つプロセス2の遷移を含まない．従って，グローバル状態(s_1, t_0, u_0)以降においてグローバル遷移を導出する場合には，プロセス2の遷移$-b_{21}$が除外され，その結果，グローバル状態(s_1, t_0, u_0)からのグローバル遷移は，図に示す通り，$|+a_{12}|$のみとなる．

到達可能性解析法において，そのグローバル遷移導出法を定義4．4．1の処理で置き換えることによって，一つの新しいプロトコル検証法が得られる．この検証法に関し，以下の補題が成立する．

[補題4．4．1]

到達可能性解析法において，定義4．4．1の処理によって得られるグローバル遷移を用いてグローバル遷移図を作成することによって，定義4．2．5のプロトコル誤りをもれなく検出することができる．ただし，デッドロックは，定義4．4．1の後半の処理によって除外されたプロセス遷移を復活しても実行可能な遷移を持たない安定グローバル状態とする．

[補題終]

[補題4．4．1の証明]

到達可能性解析法に定義4．4．1の処理を適用しても定義4．2．5の誤りを検出し損なうことがないことを示す．

定義4．4．1の処理では，実行可能となったプロセス遷移について，各々を実際に実行する場合と実行しない場合とに明確に分け，実行するものはそれらを同時に実行してグローバル遷移を導出する．このため，実行可能な遷移に関しすべての場合を包含している．従って，到達可能性解析法に定義4．4．1の処理を適用することによって実行不可能な定義済み送受信が検出不可能となることはない．同様な理由から，定義4．4．1の処理は到達可能性解析法と同様にすべての到達可能なグローバル状態を列挙できる．従って，検出不可能となる実行可能未定義受信やデッドロックはない．以上から，補題4．4．1が成立する．

[証明終]

この証明から分かるように，到達可能性解析法に定義4．4．1の処理を適用すると，到達可能性解析法と同一の全グローバル状態を列挙できる．しかし，定義4．4．1の後半の処理によってプロセス遷移が削減され，その結果到達可能性解析法に比較してグローバル遷移数が減少する．特に，各プロセスについていったん到達したプロセス状態に戻ることがないプロトコルでは，グローバル状態遷移図は木状となり，各グローバル状態に到達するプロセス遷移（及びその）系列は1通りのみとなる．

(2)冗長なグローバル遷移候補の削除

[定義4．4．2]（冗長なグローバル遷移候補の削除）

定義4．4．1の処理によって得られるグローバル遷移のうち，次の2条件を満たすプロセスの遷移を含まないグローバル遷移を削除する．

①そのプロセスがその状態で持つ定義済み遷移がいずれも直ちに実行できる遷移か，またはこの状態が持つ他の遷移を実行しない限り実行し得ない受信遷移のいずれかであり，かつ直ちに実行可能な遷移を1個以上持つこと．

②そのプロセスはその状態で他の各プロセスから信号を受信することがない，あるいは受信することが可能でかつ最初の信号が特定できること．

[定義終]

例えば，図4．4．1のプロトコル例で，初期グローバル状態(s_0, t_0, u_0)において，プロセス1は定義4．4．2の2条件を満たす．なぜなら，プロセス1の状態 s_0 における定義済み遷移は$-a_{12}$のみであり，これは直ちに実行可能で条件①を満たす．一方，プロセス1は状態 s_0 でプロセス2から信号 b_{21} を受信することが可能であるが，プロセス2がプロセス1へ送信する信号はこの b_{21} のみである．また，プロセス3はプロセス1への送信を持たないので，プロセス1は状態 s_0 でプロセス3から信号を受信することはない．即ち，条件②も成立する．従って，初期グローバル状態(s_0, t_0, u_0)からのグローバル遷移は，図4．4．2に示す通り，直ちに実行可能な遷移$-a_{12}$と$-b_{21}$の3通りの組み合わせ $|-a_{12}|$ ， $|-b_{21}|$ ， $|-a_{12}, -b_{21}|$ から，プロセス1が持つ直ちに実行可能な遷移を含まない $|-b_{21}|$ を除外した， $|-a_{12}|$ と $|-a_{12}, -b_{21}|$ の2通りとなる．

到達可能性解析法に定義4．4．1と4．4．2の処理を適用することによってもう一つの新しいプロトコル検証法が得られる．この検証法に関し，次の補題が成立する．

[補題4．4．2]

到達可能性解析法において，定義4．4．1と4．4．2の処理を適用することによって得られるグローバル遷移を用いてグローバル情態遷移図を作成し，これを検査することによって定義4．2．5の誤りをすべて検出することができる．

[補題終]

[補題4．4．2の証明]

補題4．4．1の方法に定義4．4．2の処理を追加しても定義4．2．5の誤りを検出し損なうことがないことを示す．

定義4．4．2の処理の追加によって実行可能な定義済み遷移が実行不可能となることはないので，実行可能な定義済み遷移を検出し損なうことはない．

一般に，あるグローバル状態$G=\langle\langle s_i\rangle, \langle c_{ij}\rangle\rangle$ $(i, j=1, 2, \cdots, N)$において，プロセスk $(k=1, 2, \cdots, N)$の遷移を含まないグローバル遷移を削除すると，その削除されたグローバル遷移を実行した後のグローバル状態，即ちプロセスkの状態 s_k を含む G 以外のグローバル状態が削除される．このためプロセスkの状態 s_k で実行可能な未定義受信を検出し損なうことがあり得る．しかし，定義4．4．2では，特に条件②が成立するプロセス，即ち他プロセスから最初に受信できる信号を特定できるプロセスについてのみそのようなグローバル遷移を削除するので，実行可能な未定義受信を

検出し損なうことはない．

また，一般に定義４．４．２によってグローバル遷移が一部削除されるのでグローバル状態も一部削除される．削除されたグローバル遷移をｘとすると，ｘを含むグローバル遷移が他に必ず存在し，これをｘ'とする．ｘの実行によって到達するグローバル状態は削除されるが，このグローバル状態はｘ'を実行して到達するグローバル状態に必ず到達することが可能であり，デッドロックとならない．従って，デッドロックを検出し損なうことはない．以上から，補題４．４．２が成立する．

［証明終］

(3)送受信遷移一括処理

［定義４．４．３］（送受信遷移一括処理）

各グローバル状態Ｇにおいて定義４．４．２と４．４．３の処理を実行した結果得られるグローバル遷移のうち，送信遷移を含むものについて次の３条件がすべて成立する場合，この送信遷移による信号を受信する遷移をそのグローバル遷移に含める．このような処理を送受信遷移一括処理と呼ぶ．

①その送信の信号を受信する遷移がもとのグローバル状態Ｇで定義されており，かつその信号受信バッファが空であること．

②①で示した受信を持つプロセスは，その状態で他に定義済み遷移を持たない，あるいは他に定義済み遷移を持つ場合，この受信を実行しない限りそれら遷移を実行し得ないこと．

③①で示した受信を持つプロセスは，その信号を送るプロセス以外の各プロセスからその状態で信号を受信することがない，あるいは受信することが可能でかつその最初の信号を特定できること．

［定義終］

例えば，図４．４．１と図４．４．２の例で，定義４．４．１と定義４．４．２の処理によって，安定グローバル状態 $G=(s_3, t_3, u_1)$ におけるグローバル遷移は $|-f_{32}|$ のみとなる．ところが，この信号を受信するプロセス２は，これを受信する遷移$+f_{32}$以外に実行できる遷移を持たないので定義４．４．３の条件①と②を満たす．更にプロセス２は状態 t_3 でプロセス１から信号を受信することがなく，またプロセス３から最初に受信する信号は f_{32} と特定できるので条件③を満たす．従って，グローバル状態(s_3, t_3, u_1)からのグローバル遷移 $|-f_{32}|$ は，プロセス２の受信遷移$+f_{32}$を統合した $|\pm f_{32}|$ となる．

到達可能性解析法に定義４．４．１～４．４．３の処理を適用することによって得られる検証法が本節で提案するものである．この検証法に関し次の補題が成立する．

［補題４．４．３］

到達可能性解析法に定義４．４．１～４．４．３の処理を適用することによって得られるグローバル遷移を用いてグローバル遷移図を作成し，これを検査することによって定義４．２．５の誤りをすべて検出できる．ただし，実行可能な未定義受信は次の方法で検出する．即ち，各グローバル状態において，

①送信が定義されている場合，その信号を受信するバッファが空でかつその信号の受信が定義されていないとき，および

②信号受信バッファに未処理信号が残っていてその先頭の信号の受信が定義されていないとき，

これらの受信を実行可能な未定義受信として検出する．

［補題終］

［補題４．４．３の証明］

到達可能性解析法に定義４．４．１～４．４．３の処理を適用しても定義４．２．５の誤りを検出し損なうことがないことを示す．

定義４．４．３の処理はいわば実行可能な受信処理を早く実行するものであり，他の実行可能な遷移の実行を妨げることはない．従って，実行可能な定義済み遷移を検出し損なうことがないのは明らかである．また，条件③が成立するので，一括処理される受信を持つプロセスのその状態で実行可能な受信をもれなく検出でき，実行可能な未定義受信を検出し損なうことはない．更に，定義４．４．３の処理は受信のみを早く実行するので，この処理によって安定グローバル状態が削除されることはない．従って，デッドロックを検出し損なうことはない．以上から，補題４．４．３が成立する．

［証明終］

４．４．３　プロセス状態遷移一括処理のアルゴリズム

前項に示した原理に基づくプロトコル検証法のアルゴリズムを次に示す．

［アルゴリズム４．４．１］（プロセス遷移一括処理）

step1　処理対象となるグローバル状態を表す変数Ｇを初期グローバル状態 G_0 に，また未処理グローバル状態の集合を表す変数Σを ϕ（空集合）に初期設定する．

step2　Ｇで実行可能な未定義受信を検出する．

step3　Ｇで直ちに実行可能な遷移があるか否か検査し，否のときはstep7へ進む．

step4　Ｇにおいてグローバル遷移の候補をすべて導出し（定義４．４．１），冗長なグローバル遷移候補を除去する（定義４．４．２）．更に，送受信遷移一括処理によってグローバル遷移を最終決定する（定義４．４．３）．

step5　得られた各グローバル遷移を実行し，新しいグローバル状態を導出する．導出したグローバル状態のうち，既にstep2の処理を施したものを除いてΣに追加する．

step6　Σから未処理グローバル状態を１個取り出してＧとしstep2へ戻る．このようなＧが存在しないときはstep8へ進む．

step7　Ｇがデッドロックか否か検査し（補題４．４．１），デッドロックの場合Ｇをデッドロックとして登録する．

step8　実行した送受信と定義済み送受信を比較し，実行不可能な定義済み送受信を導出し停止する．

[アルゴリズム終]

本アルゴリズムの step4 の処理では，定義4．4．2と定義4．4．3で示したように，各グローバル状態において，

①定義された受信が実行し得ないか否か．

②特定のプロセスがその状態で他プロセスから信号を受信することがないか否か．

③あるプロセスが他プロセスから最初に受信する信号を特定できるか否か．

を判定する必要がある．これらの具体的判定法は以下の通りである．

あるグローバル状態Gで定義された受信は，

①それを直ちに実行することができない，かつそれに対応する送信がG以降に定義されていない，即ちGから該当するプロセスの遷移をその遷移方向にたどっていったときそのような送信が存在しない場合，および

②該当する信号受信バッファに別の信号が存在する場合，

実行し得ないと判定する．

また，あるプロセスに対する送信がそのグローバル状態以降全く定義されていない場合，そのプロセスはその状態で他プロセスから信号を受信し得ないと判定する．

一般に，あるグローバル状態 $G=<<s_i>,<c_{ij}>>$ において，プロセス k が状態 s_k でプロセス i から最初に受信する信号を特定できるための条件は，

①グローバル状態Gにおけるプロセス i の各定義済み遷移がプロセス k への送信かまたは実行不可能のいずれかである，または

②プロセス i からプロセス k あて信号受信バッファが空でない，

のいずれかが成立することとする．

アルゴリズム4．4．1は正当であり，次の定理が成立する．

[定理4．4．1]（アルゴリズム4．4．1の正当性）

アルゴリズム4．4．1は定義4．2．5で示した誤りを有限時間ですべて検出することができる．即ち，プロトコル検証を正しく行う．

[定理終]

[定理4．4．1の証明]

補題4．4．1～4．4．3によって，アルゴリズム4．4．1は定義4．2．5の誤りをすべて検出できることがいえる．一方，本アルゴリズムは，高々すべての到達可能なグローバル状態を検出し列挙したとき停止するが，4．2で述べた仮定によって到達可能なグローバル状態は有限であるため，本アルゴリズムは必ず有限時間で停止する．以上から，本定理が成立する．

[証明終]

4．4．4　プロセス状態遷移一括処理の評価

（1）　検証処理量の理論的評価

(a)検証処理量の評価尺度

提案検証法における処理はその大部分がグローバル状態遷移図の作成，即ちアルゴリズム4．4．1の step2～step7 の処理にある．このうち一つのグローバル状態における step2～step4，step6，step7 の各処理量は，グローバル状態遷移図の大きさに依らずほぼ常に一定であると考えられる．しかし，step5 では，いったん導出したグローバル状態が既に step2 の処理を施したものか否かを判定するが，この判定にはその時点までに作成されたグローバル状態遷移図に含まれる全グローバル状態と比較する必要があり，step5 の処理量はグローバル状態遷移図の拡大に伴って増加していく．従って，グローバル状態遷移図が大きな大規模プロトコルを評価対象にすると，アルゴリズム4．4．1の処理量は step5 の処理量で代表することができる．

step5 の判定に2分探索法を採用することにすると，その処理量は $\log_2$(グローバル状態数)に比例するが，この式の(グローバル状態数)は，step2 の処理を施していない新グローバル状態の検出とともに増加していく．つまり，1回の判定に要する処理量の平均値は，グローバル状態の総数を N_s として，

$$\sum_{x=1}^{N_s} \{\log_2(x)/N_s\} = \log_2(N_s!)/N_s$$

となる．step5 においていったん導出されるグローバル状態の個数はグローバル遷移の総個数に等しい．従って，提案検証法の処理量は，グローバル遷移の総数を N_t として，

$$C=N_t \cdot \log_2(N_s!)/N_s$$

にほぼ比例すると考えられる．この性質は従来法においても全く同様である．

以上の考察から，検証処理量の一般的な評価尺度として上述の C が一つの候補と考えられる．しかし，一つのグローバル状態遷移図に含まれるグローバル遷移数とグローバル状態数は，プロトコル仕様に含まれるプロセス遷移と状態の個数やそれらの間の相互関係に大きく影響を受け，しかもその関係は複雑である．このため，これらの値を計算することは一般に容易でなく実際的でもない．従って，これらの値に直接依存する C はそのままでは検証法を比較評価するための好ましい尺度とは言えない．

そこで，ここではプロトコル仕様の複雑な影響を受けない検証処理量の尺度を新たに導入する．一つのグローバル状態が持つ直ちに実行可能なプロセス遷移のみを実行することによって導出されるグローバル遷移とグローバル状態の個数の期待値を考えることとし，これらの値をそれぞれ n_t，n_s とする．ただし，直ちに実行可能な各プロセス遷移を実行して到達するプロセス状態は互いに異なるものとする．一般に，検証結果として得られる最終グローバル状態遷移図は，一つのグロ

ーバル状態に対し n_t 個の新グローバル遷移とそれらの遷移先状態として n_s 個の新グローバル状態を付加するという操作を繰り返し実行した結果とみなすことができる．しかし，先述した通り，この繰り返し回数はプロトコル仕様に依存する．ここでは検証法自体を比較評価することを目的とするので，検証処理量の尺度として，この繰り返し回数を 1 とした場合の処理量 $c=n_t \cdot \log_2(n_s!)/n_s$ を採用することとする．以下では，一つのグローバル状態Gにおいてプロセス $i\,(i=1,2,\cdots,N)$ が持つ直ちに実行可能な遷移の個数を m_i として，n_t と n_s の値を導出する．

(b)従来法における検証処理量

従来法においては，初期グローバル状態 G_0 から出発して実行可能なプロセス遷移を逐一すべて実行していくことによってグローバル状態を順次導出していく．

グローバル状態 G が持つ直ちに実行可能な $\Sigma m_i\,(i=1,2,\cdots,N)$ 個のプロセス遷移のうち $k(=0,1,\cdots,N-1)$ 個のプロセス $i_1, i_2, \cdots, i_k$ の遷移をそれぞれ 1 個ずつ実行することによって到達するグローバル状態を $G_l(k)\,(l=1,2,\cdots)$ で表すことにする．これらの遷移を $a_{i1}, a_{i2}, \cdots, a_{ik}$ とすると，一般に，$G_l(k)$ は $a_{i1}, a_{i2}, \cdots, a_{ik}$ の実行順序に依存しない．従って，$k>0$ のとき，そのような $G_l(k)$ は $t_k=m_{i1} \cdot m_{i2} \cdot \cdots \cdot m_{ik}=\Pi m_i\,(i=i_1, i_2, \cdots, i_k)$ 個ある．各 $G_l(k)$ より出る遷移のプロセスは $i_1, i_2, \cdots, i_k$ 以外のいずれかであるので，その個数は $\Sigma m_i\,(i \neq i_1, i_2, \cdots, i_k)$ である．その結果，$G_l(k)\,(l=1,2,\cdots,t_k)$ 個のグローバル状態全体より出る遷移の個数は，$k>0$ のとき，

$$\sum_{(i_1, i_2, \cdots, i_k)} \left| \prod_{i=i_1, i_2, \cdots, i_k} m_i \cdot \sum_{i \neq i_1, i_2, \cdots, i_k} m_i \right|$$

となり，$k=0$ のとき，$\Sigma m_i\,(i=1,2,\cdots,N)$ となる．ここで，$k>0$ の場合の式で最初の Σ は，$i=i_1, i_2, \cdots, i_k$ のすべての組み合わせについての総和を表す．従って，

$$n_t=\sum_{k=1}^{N-1} \left[\sum_{(i_1, i_2, \cdots, i_k)} \left| \prod_{i=i_1, i_2, \cdots, i_k} m_i \cdot \sum_{i \neq i_1, i_2, \cdots, i_k} m_i \right| \right] + \sum_{i=1}^{N} m_i$$

となる．

一方，n_s は各プロセスのプロセス遷移を高々 1 個含む組み合わせ総数より 1 だけ小さいので，

$$n_s=\prod_{i=1}^{N} (m_i+1)-1$$

となる．

(c)提案検証法の検証処理量

提案検証法についての n_t，n_s の上限値（安全側）をそれぞれ，n_t'，n_s' とし，これらの値を以下に導出する．

まず，定義4．4．1の処理を到達可能性解析法に適用した結果得られる検証法における n_t' と n_s' を導出する．グローバル状態Gが持つ $\Sigma m_i\,(i=1,2,\cdots,N)$ 個のプロセス遷移の遷移先状態はすべて互いに異なると仮定したので，先述した通り，グローバル状態 G からこのようなプロセス遷移のみを用いて導出されるグローバル状態遷移図は木状となり，このグローバル遷移数 n_t' はグローバル状態数 n_s' と等しくなる．このグローバル状態数 n_s' は，やはり先述した通り，従来法を適用した場合に得られるグローバル状態数 n_s と等しい．従って，

$$n_t'=n_s'=\prod_{i=1}^{N} (m_i+1)-1$$

である．

次に，定義4．4．2の処理を加えたときの n_t' と n_s' の値は，定義4．4．2の条件①と②を満たす確率を計算することによって，以下の通り求めることができる．グローバル状態 G が持つ定義済み送信が直ちに実行できる確率は 1.0 である．Gが持つ受信遷移が実行できないと判定できる確率は安全のため 0 とする．一つの実行遷移が特定のプロセスの送信または受信である確率は，送信と受信が同数であり，かつ各プロセスの送受信動作が独立であると仮定すると，$1/(2N)$ となる．従って，Gが持つ受信遷移が直ちに実行できる確率は，プロトコルの誤りが十分少ないと仮定すると，その信号受信バッファに信号が 1 個以上蓄積されている確率に等しく，送信を実行した後直ちにそれに対応する受信を実行するものとすれば，この値は $1/(2N)$ 以上となる．その結果，m_i 個の実行可能な遷移を持つプロセス i が定義4．4．2の条件①を満たす確率は，それらの遷移がいずれも直ちに実行可能である確率 $[1/2\{1+1/(2N)\}]^{m_i}[m_i]$ となる．ここで，$[m]$ は，$m \geq 1$ であれば 1 を表し，$m=0$ であれば 0 を表すものとする．

一方，条件②を満たす確率は，そのプロセスが持つ $(N-1)$ 個の信号受信バッファすべてに信号が蓄積されている確率に等しいものとすると，前述の結果を利用して，$\{1/(2N)\}^{N-1}$ となる．従って，プロセス i が条件①と②の両方を満たす確率は，$Q_i=[1/2\{1+1/(2N)\}]^{m_i}[m_i] \cdot \{1/(2N)\}^{N-1}$ で，このときプロセス i の遷移はいずれのグローバル遷移にも必ず含まれることとなる．従って，

$$n_t'=n_s'=\prod_{i=1}^{N} \{Q_i m_i+(1-Q_i)(m_i+1)\}-1$$

$$Q_i=[1/2\{1+1/(2N)\}]^{m_i}[m_i] \cdot \{1/(2N)\}^{N-1}$$

となる．

次に，定義4．4．3の処理を更に加えた場合の n_t' と n_s' の値を求める．ある遷移が送信である確率を先述の通り 1/2 とすると，グローバル状態Gでは平均 $\Sigma m_i/2\,(i=1,2,\cdots,N)$ 個の送信がある．プロトコル誤りが十分少ないものと仮定すると，定義4．4．3の条件①を満たす確率は 1.0 となる．条件②が成立する確率は，m_i 個の直ちに実行可能な遷移を持つプロセス i がこの受信以外に遷移を持たない確率に等しいものとしてこれを q とする．すると，$m_i \geq 2$ のとき $q=0$，$m_i=1$ のとき $q=1$ である．また，一つのプロセスが条件③を満たす確率は，定義4．4．2の条件②と同様，$\{1/(2N)\}^{N-1}$ である．一つの送信に一つの受信を一括した場合は結果としてその受信がなくなるためそのグローバル遷移の先のグローバル状態において直ちに実行可能な遷移が等価的に 1 個減ることになる．

以上によって，定義4．4．3の処理の追加により，グローバル状態Gにおいてプロセス i(i=1,2,…,N)が持つ直ちに実行可能な遷移は等価的に m_i 個から $m_i/2\cdot q\,|1/(2N)|^{N-1}$ 個だけ減少する．

(d)提案検証法の評価

これまでの結果を用いることによって，本検証法と従来法とを比較評価することができる．本検証法における c'，n_t'と n_s'および従来法における c，n_t と n_s の数値例を表4．4．1に示す．本表の例では，提案検証法によって検証処理量が約 1/2～1/4 に減少している．

以上で示したように，ここで提案した新しいプロトコル検証法は，従来法に比較して，グローバル状態とグローバル遷移の個数を直接減少させるとともに実行可能な遷移を等価的に削減することが可能で，その結果，検証処理量が大幅に減少する．この削減効果は，N や m の値が大きいほど即ちプロトコルの規模が大きいほど，特にプロセス数N が大きいほど顕著である．

（2） 検証処理量の実験評価

提案検証アルゴリズムに基づいてプロトコル検証ツールを試作した．そして，このツールを動作させてその結果を検査することによって，定義4．2．5の誤りを検出できることを確認した．この検証ツールはプログラミング言語 C で記述されており，その大きさはコメントや保守のためのルーチン等を含め約5.1K ステップである．

本検証アルゴリズムの検証処理量を評価するため，この検証ツールを用いて様々な実験を行った．その結果の一部を表4．4．2に示す．この表で実験対象としたプロトコルは ITU-T 勧告 X.25 のパケット通信用呼設定・解放プロトコルおよび ITU-T 勧告共通線信号方式 No.7 の電話ユーザ部 TUP の呼処理基本部（回線のリセットやブロック等の回線管理機能と2重補足等の異常処理を除く）であり，それらの規模も併せて示した．これらのプロトコルは，従来法では検証することができなかったものであるが，表4．4．2から分かるように，本提案検証法によって検証が可能となった．このように，本プロトコル検証法は，実用的規模の通信プロトコルに対しても十分適用可能で検証することができる．

表4．4．1 検証処理量の例（理論値）

		ケース1	ケース2	ケース3	ケース4
ケース条件	プロセス数N	2	5	2	5
	遷移数 m_i(i=1,2,…,N)	1	1	5	5
従来法	検証処理量尺度 c	3.4	291	228	378,936
	グローバル遷移数 n_t	4	80	60	33,000
	グローバル状態数 n_s	3	31	35	7,725
提案法	検証処理量尺度 c'	1	113	133	89,280
	グローバル遷移数 n_t'	2	31	35	7,725
	グローバル状態数 n_s'	2	31	35	7,725

表4．4．2 検証実験結果

項目	項目の内訳	X.25 プロトコル	共通線信号方式 No.7
被検証プロトコルの規模	プロセス数	2	11
	プロセス遷移数	21, 21	572
	プロセス状態数	7, 7	372
検証結果	検証処理時間	1s	38m10s
	グローバル遷移数	57	24,494
	グローバル状態数	26	20,136

4．4．5 まとめ

本節では，可能な範囲でプロセスの遷移を一括した必要最小限のグローバル遷移を用いて作成したグローバル状態遷移図によって，プロトコル仕様に含まれる誤りを検出する新しいプロトコル検証法を提案した．本検証法の特長は，同様な誤りを検出する従来の代表的プロトコル検証法である到達可能性解析法に比較して，生成するグローバル状態・グローバル遷移ともにその個数が著しく削減されており，その結果検証処理量が大幅に減少していることにある．この効果については理論的にまた実験的にも論じた．例えば，理論的には，少なくとも検証処理量を 1/2～1/4 程度以下に削減できることを示した．また，実験的な評価では，従来法では検証できなかった実用的規模のプロトコルが，提案検証法によって検証できることを具体例で示した．

4.5 プロセス状態遷移図のアサイクリック展開法によるプロトコル検証の効率向上

4.5.1 概要

本節では，前節4.4で提案し論じた検証法とは原理が全く異なる新しいプロトコル検証法とそのアルゴリズムを示す．

一般に，2.2で述べた通り，プロトコルの動作を模擬する方法には，全プロセスの状態と動作を並行して完全に網羅的に列挙していく方法（以下，グローバル展開法と呼ぶ）と，各プロセスの動作を中心に模擬し他プロセスについては最小限の動作のみ模擬し管理していく方法（以下，プロセス展開法と呼ぶ．これは4.3で述べたプロセス木状展開とは全く異なるものである．）がある．そして，前節4.4で論じた新プロトコル検証法は，前者のグローバル展開法に基づくものである．本節4.5で提案し論じる新プロトコル検証法は，後者のプロセス展開法に基づくものである[48],[49]．

プロセス展開に基づく本プロトコル検証法では，各プロセスの状態遷移図を，そのプロセスが到達可能な状態をすべて含みかつそれらの状態で実行可能な送受信遷移をもれなくすべて含むよう，アサイクリック状に展開していく．そして，それ以上展開を続けても既に得た展開結果の（一部の）繰り返しとなった場合に展開を停止する．つまり，プロセス毎の状態遷移図の展開において，等価な展開状態が得られた場合，それ以降は同じ展開の繰り返しとなるので展開を停止する．このような等価な展開状態は，一つの遷移系列について展開を進めた場合に得られる場合，即ち，等価な展開状態が先祖と子孫の関係にある場合と，そのような関係はなく，全く異なる2個以上の遷移系列を実行した後に到る展開状態が互いに等価になる場合の2通りがある．このような展開に伴って，定義4.2.5で述べた全プロトコル誤りを検出する．

4.5.2 プロセス状態遷移図のアサイクリック展開

本項では，与えられたプロトコルに含まれる各プロセスの状態遷移図をアサイクリック状に展開する方法を具体的に述べる．一般に，与えられた各プロセス仕様の状態と遷移は，アサイクリック展開された状態遷移図（以下，展開状態遷移図と呼ぶ）では複数個の状態と遷移にそれぞれマッピングされる．

図4.5.1にプロトコルの一例を示す．また，そのアサイクリックなプロセス展開結果を図4.5.2に示す．図4.5.2で，楕円と矢印は，アサイクリック展開状態遷移図における状態と遷移をそれぞれ表す．これらの図で，例えば，図4.5.1のプロセス1の状態0は，図4.5.2のプロセス1の状態0.0，0.1，0.2，0.3の4個の展開状態にマッピングされる．そして図4.5.1のプロセス1のループ状の遷移系列−2，+3，−4は，図4.5.2の0.0から1.6に到る−2.0，+3.0，−4.1，−2.2にマッピングされる．

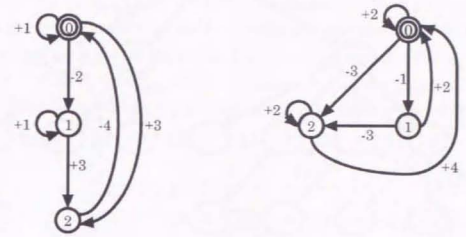


図4.5.1 プロトコルの例（初期状態＝0）

[定義4.5.1]（アサイクリック性）

プロトコルの状態遷移図がアサイクリックであるための必要十分条件は，各状態が初期状態から到達可能で，かつループ状の遷移系列が存在しないことである．状態遷移図がアサイクリックなプロトコルはアサイクリックであるという．

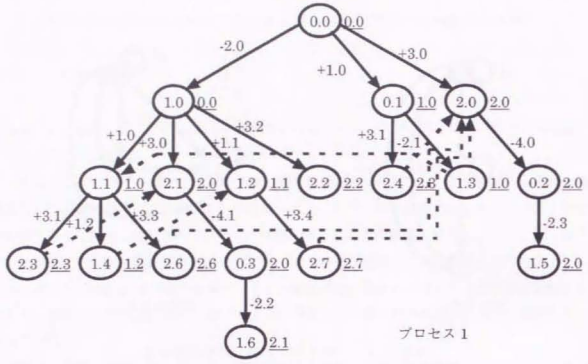
[定義終]

[定義4.5.2]（プロトコルのアサイクリック展開）

与えられたプロトコルのアサイクリックな展開（この展開を「アサイクリックなプロトコルへの変換」とも呼ぶ）は，与えられたプロトコルの各状態，各状態遷移そして各信号が，アサイクリックなプロトコルの一般には複数個の等価な展開状態，展開（状態）遷移，展開信号にそれぞれマッピングされる対応づけをいう．アサイクリックなプロトコルにおける等価な状態，遷移，信号は，元の状態名，遷移名，信号名の後にドット（.）に続く数字（0から開始する）で識別する．

[定義終]

与えられたプロトコルから，実行可能な送受信遷移をすべて漏れなく含むようなアサイクリック展開状態遷移図を作成するためには，各状態で実行可能な送受信を確定するための必要十分条件を明確にする必要がある．



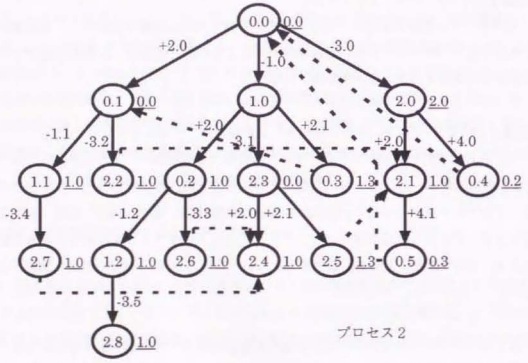


図4.5.2　図4.5.1のプロトコル例をプロセス展開した結果

[定義4.5.3] (FROM, TO, L, R) (図4.5.3参照)

XとYをそれぞれプロセスkの展開初期状態 $o_k.0 \in Q_k$ から状態 $s_k.n \in Q_k$ $(k=1,2,\cdots,N)$ に到る信号系列，プロセスiの展開初期状態 $o_i.0 \in Q_i$ から状態 $t_i.m \in Q_i$ $(i=1,2,\cdots,N)$ に到る信号系列とする．

信号系列Xにおいてプロセスiから受信した最後の信号を $FROM_i(s_k.n,X)$ とする．また，信号系列Yにおいてプロセスkに送出した最後の信号を $TO_k(t_i.m,Y)$ とする．さらに，プロセスkが信号系列Xに含まれる信号を送信又は受信することによって状態 $s_k.n$ に到達するために，プロセスiが少なくとも到達している必要がある状態を $L_i(s_k.n,X)$ とする．この $L_i(s_k.n,X)$ を状態 $s_k.n$ の最小状態と呼ぶ．また，プロセスkが信号系列Xに含まれる信号を送信又は受信することによって状態 $s_k.n$ に到達するために，プロセスiが $L_i(s_k.n,X)$ に到達するまでに送受信する信号の系列を $R_i(s_k.n,X)$ とする．

[定義終]

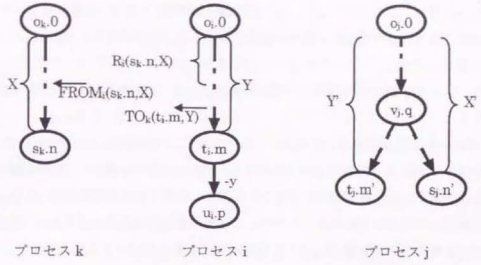


図4.5.3　FROM, TO, L, Rの説明図

例えば，図4.5.2で，FROM, TO, L, Rの例を示すと，以下の通りとなる．

$FROM_2(1.2,(-2.0)\cdot(+1.1))=1.1,$

$TO_1(1.1,(+2.0)\cdot(-1.1))=1.1,$

$FROM_2(2.7,(-2.0)\cdot(+1.1)\cdot(+3.4))=3.4,$

$TO_1(2.7,(+2.0)\cdot(-1.1)\cdot(-3.4))=3.4,$

$L_2(2.7,(-2.0)\cdot(+1.1)\cdot(+3.4))=2.7,$

$R_2(2.7,(-2.0)\cdot(+1.1)\cdot(+3.4))=(+2.0)\cdot(-1.1)\cdot(-3.4),$

$L_1(2.7,(+2.0)\cdot(-1.1)\cdot(-3.4))=1.0,$

$R_1(2.7,(+2.0)\cdot(-1.1)\cdot(-3.4))=(-2.0).$

図4.5.2において，各状態の右横にLの値をアンダーライン付きで示す．

[系4.5.1] (信号を受信できるための条件)

定義4.5.3においてアサイクリックなプロトコル P_a のプロセスkが状態 $s_k.n$ で，信号 $y \in M_{ik}$ を受信できるための必要十分条件は，以下の3点が成立することである．ただし，プロセスiは，

状態 $t_i.m$ で信号 y の送信遷移を持つものとする．

(1) $FROM_i(s_k.n, X)=TO_k(t_i.m, Y)$,

(2) $succ_j(t_i.m, y)=u_i.p$ を満たす $u_i.p$，及び k 以外のすべてのプロセス j に対し，

$L_j(u_i.p, Y\cdot y)\rightarrow L_j(s_k.n, X)$ 又は $L_j(s_k.n, X)\rightarrow L_j(u_i.p, Y\cdot y)$,

(3) $L_k(t_i.m, Y)\rightarrow s_k.n$.

ここで，$s\rightarrow t$ は，$succ^*(s, Z)=t$ を満たす信号系列 Z が存在することをいう．ただし，$succ^*(s, Z)$ は，$succ^+(s, Z)$ 又は s を表し，$succ^+(s, Z)$ は $succ(\cdots(succ(s, z_0), z_1), \cdots, z_n)$，$n\geqq 0$，$Z=z_0\cdot z_1\cdot\cdots\cdot z_n$ とする．

［系終］

系４．５．１は，以下に示す通り，信号送受信がFIFO型であること及び各プロセスの一つの状態遷移系列に含まれる状態相互の間の関係から証明することができる．

［系４．５．１の証明］

系４．５．１の必要性の証明は次の通りである．なお，十分性については自明であるので証明を省略する．

最初に条件(1)を説明する．定義３．１．１によって，プロセス i から k に向けたバッファはFIFO動作を行う．条件(1)が成立しないと仮定する．FROM と TO の定義から，展開状態 $o_i.0$ から $t_i.m$ に到る信号系列 Y に含まれる信号のうちプロセス k に送出された最後の信号は，$o_k.0$ から $s_k.n$ に到る信号系列 X に含まれる信号のうちプロセス i から受信した最後の信号とは一致しない．これは，信号系列 X の中でかつ状態 $s_k.n$ よりも前の状態で信号 y をプロセス k が受信すること，又は，$s_k.n$ で y 以外の信号をプロセス k が受信することを意味する．これは矛盾である．従って，条件(1)が成立する．

次に，条件(2)と(3)を説明する．$L_j(u_i.p, Y\cdot y)=t_j.m'$，$L_j(s_k.n, X)=s_j.n'$とする．$o_j.0$ から $t_j.m'$ に到る信号系列を Y'とし，$o_j.0$ から $s_j.n'$に到る信号系列を X'とする．条件(2)が成立しないとすると，X'は Y'の部分系列ではないし，また逆に Y'が X'の部分系列でもない．$o_j.0$ から $v_j.q$ に到る信号系列が X'の部分系列でありかつ Y'の部分系列でもあるような $v_j.q$ が存在する．L の定義から，$v_j.q$ から $t_j.m'$に到る信号系列を実行すると，$v_j.q$ から $s_j.n'$に到る信号系列は実行できない．逆に，$v_j.q$ から $s_j.n'$に到る信号系列を実行すると，$v_j.q$ から $t_j.m'$に到る信号系列は実行できない．これは矛盾である．従って，条件(2)が成立する．条件(3)は条件(2)のプロセス k に対する特別な一ケースであり，同様な理由によって，これも成立する．

［証明終］

例えば，図４．５．２で，プロセス１は状態 1.2 で，プロセス２が状態 1.1 で送出した信号 3.4 を受信することができる．なぜなら，以下の示す通り３条件が成立するからである．

(1) k=1 で，$s_k.n=1.2$，$X=(-2.0)\cdot(+1.1)$，また，i=2 で，$t_i.m=1.1$，$Y=(+2.0)\cdot(-1.1)$とすると，$FROM_2(1.2, (-2.0)\cdot(+1.1))=1.1$，$TO_1(1.1, (+2.0)\cdot(-1.1))=1.1$ となる．従って，$FROM_2(1.2, (-2.0)\cdot(+1.1))=TO_1(1.1, (+2.0)\cdot(-1.1))$.

(2) y=3.4 で，$u_i.p=2.7$，$L_1(2.7, (+2.0)(-1.1)(-3.4))=1.0$，$L_1(1.1)=1.1$ であり，プロセス２について $L_2(2.7, (+2.0)(-1.1)(-3.4))\rightarrow L_1(1.1)$が成立する．

(3) $L_1(1.1, (+2.0)(-1.1))=1.0$，プロセス２について，$L_1(1.1, (+2.0)(-1.1))\rightarrow 1.2$ が成立する．

この例で，プロセス１の状態 1.0 から状態 1.2 への遷移の受信(1.0, +1.1)は，プロセス２の状態 0.1 から状態 1.1 への遷移の送信(0.1, -1.1)に対応する．また，プロセス１の状態 1.0 から状態 1.1 への遷移の受信(1.0, +1.0)は，プロセス２の状態 0.0 から状態 1.0 への遷移の送信(0.0, -1.0)に対応する．

本節で提案する検証法のアルゴリズムでは，次に示す手順に従って，与えられたプロトコルの各プロセスの状態遷移図からアサイクリック状の状態遷移図を構築していく．

［アルゴリズム４．５．１］（アサイクリック状展開）

i, j, k をプロセスの識別子（$1\leqq i, j, k\leqq N$），y を信号の識別子（$y\in M_{ik}$）とする．以下に示す手順に従って，一つの送信（$t_i.y$）とそれに対応する全実行可能受信（$s_k.y$）を帰納的に順次追加することによって，アサイクリック展開図 P_a，安定状態の集合 S_G，関数 L と R を作成する．ここで，全実行可能受信（$s_k.y$）は系４．５．１によって得られる．

step1：P_a を，$P_a=(Q, o, M, succ)$とする．ここで，$Q=\langle Q_i\rangle$，$Q_i=\{o_i.0\}$，$o=\langle o_i\rangle$，$o_i=o_i.0$，$M=\langle M_{ij}\rangle$，$M_{ij}=\phi$，$(1\leqq i, j\leqq N)$ とし，$succ_i(o_i.y)$（$y\in M_{ij}$）は未定義とする．また，$S_G=\{(o_1.0, o_2.0, \cdots, o_N.0)\}$，$L=\langle L_j\rangle$，$L_j(o_i.0, \varepsilon)=o_j.0$，$R=\langle R_j\rangle$，$R_j(o_i.0, \varepsilon)=\varepsilon$ $(1\leqq i, j\leqq N)$ とする．

step(n)：P_a，S_G，L_j，R_j が得られているものとする．

step(n+1)：本ステップで得る P_a，S_G，L_j，R_j をそれぞれ $P_a'=(Q', o', M', succ')$，S_G'，L_j'，R_j'とする．

まず，P_a'と S_G'を次により導出する．

$Q'=\langle Q_i'\rangle$

$Q_k'=Q_k\cup\{succ_k(s_k.n, y)\}$

$Q_i'=Q_i\cup\{succ_i(t_i.m, y)\}$

$Q_j'=Q_j\ (j\neq i, k)$

$o'=o$

$M_{ik}'=M_{ik}\cup\{y\}$

$M_{jl}'=M_{jl}\ (j\neq i$ または $l\neq k)$（l＝L の小文字）

$succ'=succ\cup\{succ_k(s_k.n, y)\}\cup\{succ_i(t_i.m, y)\}$

$S_G'=S_G\cup\{P_a'$の作成に伴って生成された全安定状態$\}$

次に，$u_i.p=succ_i(t_i.m, y)$とし，$v_k.q=succ_k(s_k.n, y)$とする．ただし，$y\in M_{ik}$ とする．状態 $u_i.p$ と $v_k.q$ の L_j'，R_j'は次の通りとする．

$L_i'(u_i.p, Y\cdot y)=u_i.p$,

$R_i'(u_i.p, Y\cdot y)=Y\cdot y$,

$L_j'(u_i.p, Y\cdot y)=L_j(t_i.m, Y)$,

$R_j'(u_i.p, Y\cdot y)=R_j(t_i.m, Y)$,

ただし，$1 \leqq j \leqq N$ で $j \neq i$ である．また，

$L_k^*(v_k.q, X \cdot y) = v_k.q$,

$R_k^*(v_k.q, X \cdot y) = X \cdot y$,

$L_j^*(v_k.q, X \cdot y) = \max_L \{L_j(s_k.n, X),\ L_i^*(u_i.p, Y \cdot y)\}$,

$R_j^*(v_k.q, X \cdot y) = \max_R \{R_j(s_k.n, X),\ R_i^*(u_i.p, Y \cdot y)\}$,

ただし，$1 \leqq j \leqq N$ で $j \neq k$ である．ここで，$u \to v$ のときに限り $\max_L(u, v) = v$ であり，逆に $v \to u$ のときに限り $\max_L(u, v) = u$ である．また，$\max_R(U, V)$ は，U と V の間の信号系列の最長のものである．

［アルゴリズム終］

系4．5．1を用い，上述のアルゴリズム4．5．1の展開手続きに従って，一つの送信とそれに関連する実行可能な全受信をアサイクリック展開図に，一つのステップで追加する．この結果，最終的には実行可能な全送信と全受信とがこのアサイクリック展開図に含まれることになる．

定義4．5．3に示したマッピング関係が，与えられたプロトコルとそのアサイクリック展開図に存在するならば，与えられたプロトコルの定義済み送受信と安定状態は，そのアサイクリック展開図の定義済み送受信と安定状態に対応する．従って，もしアサイクリックプロトコルに未定義受信，実行不可能送受信，デッドロックが含まれていないならば，与えられたプロトコルにもこのような誤りは存在しないことになる．

以上から，与えられたプロトコルとアサイクリック状展開によって作成されるそのアサイクリックプロトコルとの間に定義4．5．3のマッピング関係が存在しかつそのアサイクリックプロトコルにプロトコル誤りが存在しなければ，元のプロトコルにも誤りが存在しないことが分かる．このように，アサイクリックプロトコルを検査することによって元のプロトコルが誤りを含むか否か判定することができる．

4．5．3　プロセス状態遷移図のアサイクリック展開の停止

上述のアサイクリック状展開手続きは，展開停止条件が示されていないため，展開が無限に続く可能性があり，それ自体では誤り検出アルゴリズムとして完全ではない．つまり，誤りを検出し損うことがないという条件下でアサイクリック展開を停止できることが必要である．停止条件として，以下に示す2種類の条件（終了条件，及びマージ条件）を考える．もしこの停止条件を適用しない場合には，適用した場合に停止する状態以降に，展開シーケンス（系列）の一部が繰り返し展開されることになる．

［定義4．5．4］（展開終了条件：タイプ1s，図4．5．4参照）

以下の3条件が成立するとき，状態 $t_i.m \in Q_i$ は状態 $s_i.n \in Q_i$ のタイプ1sのプレカーソルと呼び，状態 $s_i.n$ をタイプ1sとマークする．ただし，U，Vをそれぞれ $o_i.0$ から $t_i.m$，$s_i.n$ に到る信号系列とし，UはVの部分系列，即ち，$U = u_1 \cdot u_2 \cdots\cdots u_p$ とし，$V = v_1 \cdot v_2 \cdots\cdots v_q$ とすると，$u_p = v_p$ で，$p \leqq q$ である．

(1) $t_i.m \to s_i.n$ である．しかし，$t_i.m \neq s_i.n$ である．

(2) 任意のプロセス $j\ (1 \leqq j \leqq N)$ について，$L_j(t_i.m, U) \sim L_j(s_i.n, V)$．

(3) U内の全信号を送受信することによって到達するバッファ状態と $R_j(t_i.m, U)$ は，V内の全信号を送受信することによって到達するバッファ状態と $R_j(s_i.n, V)$ に等しい（$1 \leqq j \leqq N$）．

ここで，～は，s=t の場合に限り s.i～t.j が成立するという等価関係（i と j が等しい必要はない）を表す．

［定義終］

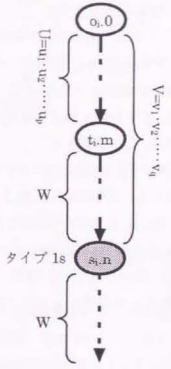


図4．5．4　タイプ1sによる展開の停止

もし，タイプ1sとマークされた状態 $s_i.n$ で展開を終了しない場合は，状態 $s_i.n$ 以降，状態 $t_i.m$ から状態 $s_i.n$ までの状態遷移系列が繰り返し展開されることになる．

図4．5．2の例で，プロセス2の状態0.5は，次に示す通り3条件が成立するのでタイプ1sとマークされる．

条件(1)：i=2，$t_i.m=0.0$，$s_i.n=0.5$ であり，$0.0 \to 0.5$ である．更に，$0.0 \neq 0.5$ であるので，条件(1)が成立する．

条件(2)：$U = \varepsilon$ で，$V = (-3.0) \cdot (+2.0) \cdot (+4.1)$，$L_i(0.0, \varepsilon) = 0.0$，$L_i(0.5, (-3.0) \cdot (+2.0) \cdot (+4.1)) = 0.3$，

$L_2(0.0, \varepsilon)=0.0$, $L_2(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))=0.5$ である．従って，プロセス1に対して，$L_1(0.0, \varepsilon)\sim L_1(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))$，プロセス2に対して，$L_2(0.0, \varepsilon)\sim L_2(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))$ となり，条件(2)が成立する．

条件(3)：$R_1(0.0, \varepsilon)$ 及び $U=R_2(0.0, \varepsilon)$ に含まれる全信号を送受信することによって到達するプロセス1から2宛のバッファ状態とプロセス2から1宛のバッファ状態をそれぞれ c_{12}，c_{21} とする．そして，$R_1(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))$ 及び $V=R_2(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))$ に含まれる全信号を送受信することによって到達するプロセス1から2宛のバッファ状態とプロセス2から1宛のバッファ状態をそれぞれ c_{12}'，c_{21}' とする．

$R_1(0.0, \varepsilon)=\varepsilon$，$R_1(0.5, (-3.0)\cdot(+2.0)(+4.1))=(-2.0)\cdot(+3.0)\cdot(-4.1)$，$U=R_2(0.0, \varepsilon)=\varepsilon$，$V=R_2(0.5, (-3.0)\cdot(+2.0)\cdot(+4.0))=(-3.0)\cdot(+2.0)\cdot(+4.0)$ である．従って，$c_{12}=\varepsilon$，$c_{21}=\varepsilon$，$c_{12}'=\varepsilon$，$c_{21}'=\varepsilon$，$c_{12}\sim c_{12}'$，$c_{21}\sim c_{21}'$ となり，条件(3)が成立する．

［定義4．5．5］（展開終了条件：タイプ2s，図4．5．5参照）

状態 $u_i.p_i$ は，次の3条件が成立する場合，タイプ2sのマークを付ける．

(1)全プロセス $i\,(1\leq i\leq N)$ について，$S_t=(\cdots, t_i.m_i, \cdots)$，$S_s=(\cdots, s_i.n_i, \cdots)$ として $t_i.m_i\sim s_i.n_i$ を満たし，かつ $succ_i^*(t_i.m_i, X_i)=s_i.n_i$ を満たす X_i が存在するという条件を満たすグローバル状態 $G_t(S_t, C_t)$ と $G_s(S_s, C_s)$ が存在する．

(2) $succ_i^*(s_i.n_i, Y)=u_i.p_i$ を満たす信号系列Yが存在する．

(3)すべての $j\,(1\leq j\leq N)$ について，以下の条件(3.1)又は(3.2)が成立する．

(3.1) $succ_j^*(s_j.n_j, Z_j)=L_j(u_i.p_i, Y')$ を満たす信号系列 Z_j が存在する．ここで，Y'は，状態 $o_i.0$ から状態 $s_i.n_i$ を経由して状態 $u_i.p_i$ に到る信号系列である．

(3.2) $s_i.n_i$ が常に $L_j(u_i.p_i, Y')$ から到達可能である．

これら3条件が成立するとき，状態 $t_i.m_i$ は状態 $s_i.n_i$ のタイプ2sのプレカーソルであると呼ぶ．

［定義終］

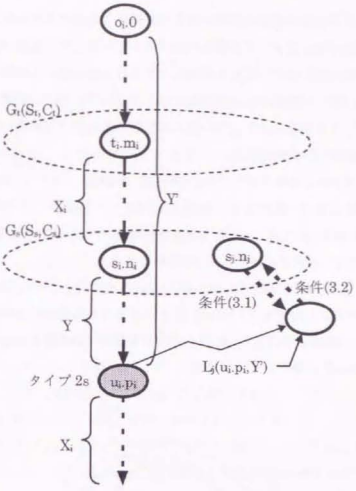


図4．5．5　タイプ2sによる展開の停止

タイプ2sとマークされた状態 $u_i.p_i$ で展開を終了しない場合，この状態 $u_i.p_i$ 以降，状態 $t_i.m_i$ から状態 $s_i.n_i$ に到る遷移系列が繰り返し展開されることになる．

例えば，図4．5．2の例で，プロセス2の状態0.5はタイプ2sともマークされる．なぜなら，次に示す通り，［定義4．5．5］の3条件が成立するからである．

条件(1)：$S_t=(0.0, 0.0)$，$S_s=(0.3, 0.5)$ とする．プロセス1に対し，0.0〜0.3であり，またプロセス2に対し，0.0〜0.5である．従って，条件(1)が成立する．

条件(2)：$s_i.n_i=0.5$，$u_i.p_i=0.5$ であるので，プロセス2に対し，$succ_2^*(0.5, Y)=0.5$ を満たす信号系列 $Y=\varepsilon$ が存在する．従って，条件(2)が成立する．

条件(3)：$L_1(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))=0.3$ で，$succ_1^*(0.3, \varepsilon)=0.3$ であるので，プロセス1に対して，$succ_1^*(0.3, Y)=L_1(0.5, (-3.0)\cdot(+2.0)\cdot(+4.1))$ を満たす信号系列 $Y=\varepsilon$ が存在する．つまり，条件(3.1)が成立する．

一般に，タイプ2sのマークがついた場合，必ずしもタイプ1sとマークされるとは限らない．

実際のプロトコルでは，複数個の異なる状態遷移系列が同一のグローバル状態に到達すること

が少なくない．これまでに述べてきた展開手続きと展開終了処理のみで展開を実施すると，このようなグローバル状態が多いほど，不必要な展開を行うことになり，展開状態遷移図に含まれる状態は指数関数的に増大していく．これを回避して，プロトコル誤りの検出を損なうことなく効率よく検証するため，新しい展開停止条件を次に導入する．この新しい展開停止条件によって，その時点までに並行して展開されてきた複数個の状態遷移系列が一つの系列にマージされる．この展開停止条件を展開マージ条件と呼ぶ．

［定義４．５．６］（展開マージ条件：タイプ 1p，図４．５．６参照）

次の２条件が成立する場合，状態 $s_i.n$ は状態 $t_j.m$ のタイプ 1p のパートナと呼び，状態 $s_i.n$ にタイプ 1p のマークを付ける．ただし，U と V をそれぞれ $o_i.0$ から状態 $t_j.m$ と状態 $s_i.n$ に到る信号系列とし，U と V は互いに相手の部分系列ではないとする．

(1)全プロセス $j(1 \leq j \leq N)$ に対し，$L_j(t_j.m, U) \sim L_j(s_i.n, V)$．

(2)全プロセス $j(1 \leq j \leq N)$ に対し，U と $R_j(t_j.m, U)$ に含まれる信号の送受信を実行して到達する全バッファ状態は，V と $R_j(s_i.n, V)$ に含まれる信号の送受信を実行して到達するバッファ状態に完全に等しい．

［定義終］

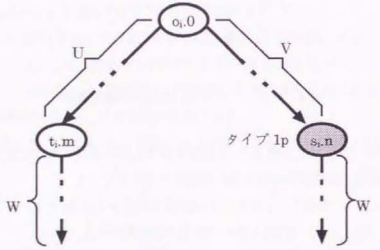


図４．５．６　タイプ 1p による展開の停止

定義４．５．６の２条件が成立する場合，状態 $t_j.m$ 以降の展開と状態 $s_i.n$ 以降の展開は同一となる．従って，タイプ 1p とマークされた状態 $s_i.n$ 以降の展開は省略できる．

例えば，図４．５．２の例で，プロセス１の状態 1.4 は，次に示す通り２条件を満たすため，状態 1.2 のタイプ 1p のパートナとなり，1p とマークされる．

条件(1)：$s_i.n=1.4$，$t_j.m=1.2$，$L_1(1.2, (-2.0)\cdot(+1.1))=1.2$，$L_1(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))=1.4$，$L_2(1.2, (-2.0)\cdot(+1.1))=1.1$，$L_2(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))=1.2$，$L_1(1.2, (-2.0)\cdot(+1.1)) \sim$

$L_1(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))$，$L_2(1.2, (-2.0)\cdot(+1.1)) \sim L_2(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))$ であり，条件(1)が成立する．

条件(2)：$U=(-2.0)\cdot(+1.1)=R_1(1.2, (-2.0)\cdot(+1.1))$ と $R_2(1.2, (-2.0)\cdot(+1.1))$ にそれぞれ含まれる送受信を実行して到達する，プロセス１から２宛のバッファ状態とプロセス２から１宛のバッファ状態をそれぞれ c_{12}，c_{21} とする．また，$V=(-2.0)\cdot(+1.0)\cdot(+1.2)=R_1(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))$ と $R_2(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))$ にそれぞれ含まれる送受信を実行して到達する，プロセス１から２宛のバッファ状態とプロセス２から１宛のバッファ状態をそれぞれ c_{12}'，c_{21}' とする．

$R_2(1.2, (-2.0)\cdot(+1.1))=(+2.0)\cdot(-1.1)$ で，$R_2(1.4, (-2.0)\cdot(+1.0)\cdot(+1.2))=(-1.0)\cdot(+2.0)\cdot(-1.2)$ となる．従って，$c_{12}=\varepsilon$，$c_{21}=\varepsilon$，$c_{12}'=\varepsilon$，$c_{21}'=\varepsilon$ で，$c_{12} \sim c_{12}'$，$c_{21} \sim c_{21}'$ である．つまり，条件(2)が成立する．

［定義４．５．７］（展開マージ条件：タイプ 2p，図４．５．７参照）

状態 $u_i.p_i$ は，次の３条件が成立するときタイプ 2p のマークが付けられる．

(1)全プロセス $i(1 \leq i \leq N)$ について，$S_t=(\cdots, t_i.m_i, \cdots)$，$S_s=(\cdots, s_i.n_i, \cdots)$ として $t_i.m_i \sim s_i.n_i$ を満たし，かつ $succ_i^*(t_i.m_i, X_i)=s_i.n_i$ を満たす X_i が存在しないという条件を満たす２個のグローバル状態 $G_t(S_t, C_t)$ と $G_s(S_s, C_s)$ が存在する．

(2) $succ_i^*(s_i.n_i, Y)=u_i.p_i$ を満たす信号系列 Y が存在する．

(3)すべての $j(1 \leq j \leq N, j \neq i)$ について，以下の条件(3.1)又は(3.2)が成立する．

(3.1) $succ_j^*(s_j.n_j, Z_j)=L_j(u_i.p_i, Y')$ を満たす信号系列 Z_j が存在する．ここで，Y'は，状態 $o_i.0$ から状態 $s_i.n_i$ を経由して状態 $u_i.p_i$ に到る信号系列である．

(3.2) $s_j.n_j$ が常に $L_j(u_i.p_i, Y')$ から到達可能である．

これら３条件が成立するとき，状態 $u_i.p_i$ は状態 $t_i.m_i$ のタイプ 2p のパートナであると呼ぶ．

［定義終］

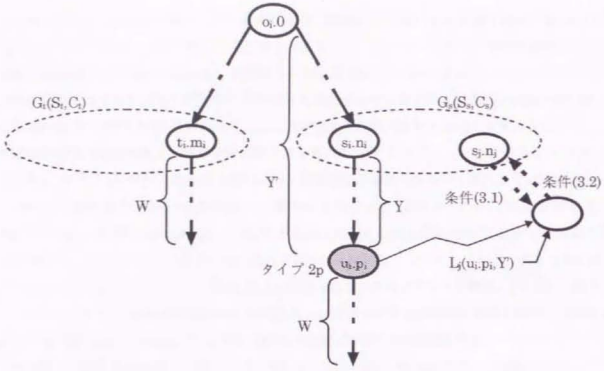


図4.5.7　タイプ2pによる展開の停止

状態 $u_i.p_i$ と状態 $t_i.m_i$ が上述の3条件を満たす場合，これら2つの状態以降の展開状態遷移系列は同一となる．従って，タイプ2pのマークが付いた状態以降の展開を省略することができる．

例えば，図4.5.2の例で，プロセス1の状態1.4は，次の通り，状態1.2のタイプ2pのパートナとなり，タイプ2pのマークも付けられる．

条件(1)：i=1で，S_t=(1.2,1.1)，S_s=(1.4,1.2)として，2個のグローバル状態 $G_t(S_t,C_t)$ と $G_s(S_s,C_s)$ が存在し，プロセス1について1.2～1.4を満たし，プロセス2について1.1～1.2となる．従って，条件(1)が成り立つ．

条件(2)：$s_i.n_i$=1.4で，$u_i.p_i$=1.4であるので，プロセス1に対して $succ_1^*(1.4,Y)$=1.4を満たす信号系列 $Y=\varepsilon$ が存在する．従って，条件(2)が成立する．

条件(3)：$L_2(1.4,(-2.0)\cdot(+1.0)\cdot(+1.2))$=1.2で，$succ_2^*(1.2,\varepsilon)$=1.2であるから，プロセス2に対して $succ_2^*(1.2,Z_j)=L_2(1.4,(-2.0)\cdot(+1.0)\cdot(+1.2))$ を満たす信号系列 $Z_j=\varepsilon$ が存在する．つまり，条件(3.1)が成立する．

この例では，タイプ1pのマークが付いた状態にタイプ2pのマークも付いたが，一般にはこのような関係はなく，両者は互いに独立である．

以上で示したプロセス単位でのプロトコルのアサイクリックな展開と展開停止条件によって，図4.5.1のプロトコルを展開した結果は図4.5.2となる．図4.5.2の各状態 $t_i.n_i$ の近くにあるアンダライン付きの状態 $t_j.n_j$ は状態 $t_i.n_i$ のL値，即ち $L_2(t_i.n_i,X_i)$ を表す．ここで，X_i は，状態 $o_i.0$ から状態 $t_i.n_i$ に到る信号系列である．

$X_j=R_j(t_j.n_j,X_i)$ とする．X_i と X_j のそれぞれに含まれる信号を送受信した結果到達する，プロセスiからj宛のバッファ状態とプロセスjからi宛のバッファ状態をそれぞれ c_{ij}，c_{ji} とする．この例の説明の範囲内において，$(t_i.n_i,t_j.n_j,c_{ij},c_{ji})$ を状態 $t_i.n_i$ のグローバル状態と呼ぶことにする．

図4.5.2の破線矢印は以下に示す2個の関係のいずれかを示す．ただし，破線矢印の矢元の状態を $u_i.p_i$ とし，矢頭の状態を $s_i.m_i$ とする．

関係1；状態 $s_i.m_i$ のグローバル状態と，状態 $u_i.p_i$ のグローバル状態が互いに等価である．ここで等価関係～は定義4.5.4に示した通りで，状態 $o_i.0$ から状態 $s_i.m_i$ に到る信号系列は，状態 $o_i.0$ から状態 $u_i.p_i$ に到る信号系列の部分系列である．実際，この関係1が成立すると，この例では，これらの状態 $u_i.p_i$ (i=1,2)はタイプ1sのマークが付けられる．例えば，プロセス2において，$(u_i.p_i,s_i.m_i)$=(0.5,0.0)，(0.4,0.0)とすると，プロセス2の信号系列(-3.0)・(+2.0)・(+4.1)における状態0.0と0.5は，互いに等価なグローバル状態(0.0,0.0,ε,ε)と(0.5,0.3,ε,ε)をそれぞれ持つ．従って，状態0.5において展開が停止する．

関係2；状態 $s_i.m_i$ のグローバル状態と，状態 $u_i.p_i$ のグローバル状態が互いに等価である．ここで等価関係～は定義4.5.4に示した通りで，状態 $o_i.0$ から状態 $s_i.m_i$ に到る信号系列は，状態 $o_i.0$ から状態 $u_i.p_i$ に到る信号系列の部分系列ではない．実際，この関係2が成立すると，この例では，これらの状態 $u_i.p_i$ (i=1,2)はタイプ1pのマークが付けられる．例えば，プロセス1において，$(u_i.p_i,s_i.m_i)$=(1.4,1.2)，(2.3,2.1)とすると，プロセス1の信号系列(-2.0)・(+1.0)・(+1.2)により到達する状態1.4と，プロセス1の信号系列(-2.0)・(+1.1)により到達する状態1.2は，それぞれ互いに等価なグローバル状態(1.4,1.2,ε,ε)と(1.2,1.1,ε,ε)を持つ．従って，これら2個の状態以降の展開はマージされる．

4.5.4　プロセス状態遷移図のアサイクリック展開法のアルゴリズム

本項では，4.5.3で原理を示した提案プロトコル検証法のアルゴリズムを示す．与えられたプロトコルに含まれるプロセスの個数をNとする．

[アルゴリズム4.5.2]（アサイクリック展開法）

導出すべきプロセス展開図全体をPとし，処理対象とする送信を表す変数をa，未処理の送信を蓄えておくFIFO型キューをWとする．

step1　Pにおける各プロセスi（i=1, 2, …, N）の初期展開状態を $o_i.0$ とする．また，Pの各初期展開状態 $o_i.0$ の最小状態を $|o_j.0|$（$1\leq j\leq N$, $j\neq i$）とする．更に，各プロセスiの初期状態 o_i が持つ送信をすべてWに蓄積する．

step2　Wから送信を一つ取り出してaとし，この送信aを持つプロセスをi（$1\leq i\leq N$）とする．Pのプロセスiの展開図において，送信aによる遷移とその先の展開状態を追加する．さらに，この展開状態についてサブルーチンTを実行する．ただし，Wが空だった場合はstep6に進む．

step3　プロセス i について，送信 a の遷移元展開状態から連続して実行可能な全受信遷移とそれらの先の展開状態を a の遷移先展開状態にコピーするというプロパゲーション処理を行う．このプロパゲーション処理は文献[32]の方法に従う．（図4.5.8と図4.5.9参照）ここで，各展開状態を導出する毎にサブルーチン T を実行する．

step4　送信 a の信号を受信するプロセスを j（$1 \leqq j \leqq N$, $j \neq i$）とする．P のプロセス j の展開図において，a の遷移元展開状態の最小状態から到達できる範囲内の各展開状態が，a の信号を受信することが可能か否か調べ，受信可能であればその展開状態に a の信号の受信遷移と遷移先の展開状態を導出し追加する．ここで，各展開状態を導出する毎にサブルーチン T を実行する．

step5　step4 で導出した各受信遷移について，その遷移元の展開状態から連続して実行可能な全受信遷移とそれらの先の展開状態を，その遷移先の展開状態にコピーするというプロパゲーション処理を行う．このプロパゲーション処理は文献[32]の方法に従う．（図4.5.8と図4.5.9参照）ここで，各展開状態を導出する毎にサブルーチン T を実行する．これらの処理の完了後，step2 に戻る．

step6　元のプロトコルで定義されている送受信のうち，プロセス展開図 P に含まれないものを実行不可能送受信とする．

サブルーチン T：導出した各展開状態に対し以下の 4 点の処理を行う．

- その展開状態が受信によって遷移し，かつその受信が元のプロトコルに定義されていない場合，その受信遷移を未定義受信とし，以下の 3 点の処理はスキップする．
- その展開状態について，安定状態の集合，及び，関数 L（最小状態）と R を導出し追加する．（アルゴリズム 4.5.1 による）
- その展開状態について，定義 4.5.4 ～ 4.5.7 の条件が成立するか否か検査し，いずれかの条件が成立する場合「等価展開状態」とマークする．
- その展開状態が「等価展開状態」とマークされておらずかつ元のプロトコルで送信を持つ場合，それら送信をすべて W に追加する．

［アルゴリズム終］

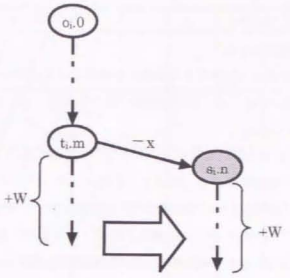


図4.5.8　－プロパゲーション

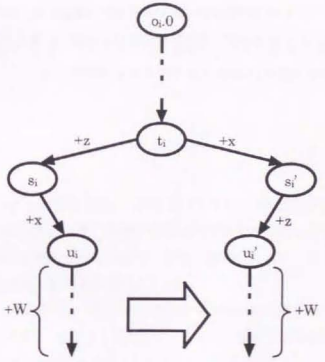


図4.5.9　+プロパゲーション

4.5.5　プロセス状態遷移図のアサイクリック展開法の評価

本節で提案したプロトコル検証法の特徴は以下の通りである．

(1)各プロセス毎に，実行可能な状態遷移と到達可能な状態とから構成される最小のプロセ

ス状態遷移図を作成する．

(2)プロセス数に制限はない．

(3)定義4．2．5で示した全プロトコル誤りを検出することができる．

プログラム言語 C を用いて本プロトコル検証法に基づく検証ツールを作成した．このプログラムの大きさは約 12,000 行である．

図4．5．10 に，プロセス状態遷移図のアサイクリック展開に基づくプロトコル検証の効果を，従来法との比較で示す．検証対象としたプロトコルはトークンリング LAN 用プロトコルで，リング内の各通信ノードは，状態数が6で遷移数が12の1個のプロセスで実現される．この図の横軸はプロセス数又は通信ノード数を表し，縦軸は，所要メモリ量の尺度としてメモリに蓄積すべき状態数と，計算処理量の尺度として検証時間(CPU 使用時間)を表す．

図4．5．10 から分かるように，従来法に比較して，所要メモリ量及び検証時間を大幅に削減できる．例えば，プロセス数が10の場合，本提案プロトコル検証法によって所要メモリ量と検証処理量は10分の1程度になる．この削減度合い，つまり本検証法による効果は，プロセスの増加に伴って増大していく．また，従来法では使用コンピュータのメモリ制限から12個以下のノード(プロセス)を持つ場合しか検証できなかったが，本提案検証法によれば 14 個までのノードを持つ場合の検証が可能であった．

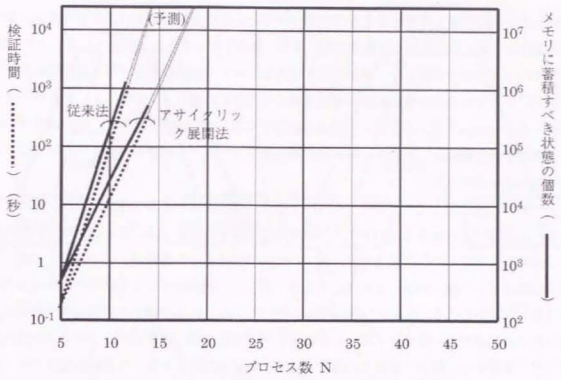


図4．5．10　アサイクリック展開に基づくプロトコル検証法による蓄積状態数(メモリ量)と検証時間(処理量)の削減効果

4．5．6　まとめ

本節では，プロセス展開法を基にした新しいプロトコル検証法を提案した．この提案検証法には，プロセス展開法に比較して，展開の停止強化を徹底的に図ったことに特長がある．即ち，プロトコル誤りの検出に支障がない範囲で，冗長な展開をなくし，これによって，検証に必要なメモリ量と処理量を削減し検証効率を向上させた．

検証効率の向上の度合いについては，具体的なプロトコル例を用いて論じた．そして，従来法に比較して，検証効率を10倍以上に向上できること，プロトコル規模，特にプロセス数が多くなるほど検証効率の向上が大きくなること等を明らかにした．

4．6　誤りの性質に応じた段階的検証処理によるプロトコル検証の効率向上

4．6．1　概要

一般に，誤り検出のために列挙し検査すべき情報は，検出すべき誤りの性質に応じて異なる．このため，性質が異なる誤りを同時に検出する場合には，それら誤りに応じて異なる情報の組み

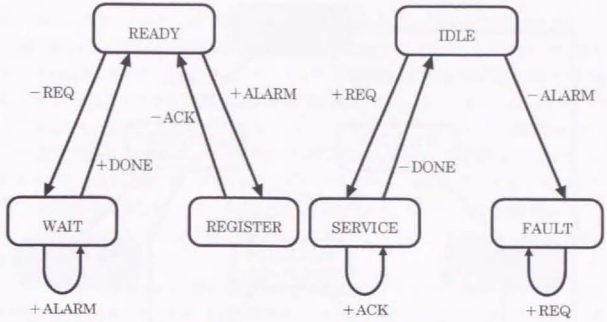
合わせをすべて列挙し検査することが必要になる．この結果，列挙すべき情報の量は，個々の誤りの検出のために列挙すべき情報量の積になり，極めて多くなることがある．そこで，本節では，検出すべきプロトコル誤りを，全プロセスを含むシステム全体の動作状態に直接関わるデッドロックと，システム全体の動作状態に直接は係わらない他の誤りとに分類する．そして，まず後者に属す誤りを検出するため各プロセス毎の動作を中心とする検査を実行し，その後全プロセス動作の同時検査を実施し，システム全体の状態を表すグローバル状態を列挙してデッドロックを検出する方法を提案する．

いずれの検査においてもプロトコル動作を表す状態遷移図を展開し作成するが，それぞれの誤り検出に支障のない範囲で，列挙する状態や状態遷移を必要最小限に留める．特に，最初のプロセス毎の動作を中心にした検査では，各プロセスの実行可能な遷移はすべて列挙するが，グローバル状態はできる限り列挙しないことにする．そして，次の全プロセス動作の同時検査では，実行可能と判定された各プロセスの遷移のみを用いたうえ，4．4で述べた状態遷移一括化法と同様，誤り検出に支障がない範囲でグローバル状態の列挙を最小限に押さえる．このような展開処理によって，展開する状態と状態遷移の総数の大幅な削減を可能とする．本提案検証法では，このような段階的な検証を行うことから，段階的プロトコル検証法と呼ぶ．

4．6．2　プロトコル検証における所要メモリ量と処理量の支配的要素の分析

定義4．2．5のプロトコル検証問題を解く方法については多くの提案がある．原理的にはプロトコルに従う動作をすべて模擬し誤りの有無を検査すればよい．2．2で述べた通り，すべてのプロトコル誤りを検出するために必要なプロトコル動作を網羅的に模擬する主な方法には，グローバル状態を順次列挙する方法[20]~[22]（以下，従来法1と呼ぶ）と，プロセス毎にプロトコル動作を模擬していく方法[32]（以下，従来法2と呼ぶ）とがある．

従来法1では，初期グローバル状態から出発して，グローバル遷移を逐一実行することによって到達可能となるすべてのグローバル状態を順次導出し，同一のグローバル状態を検出した場合にはそれ以降のグローバル遷移の実行を停止する．このようにして得たグローバル（状態）遷移図の各グローバル状態において，未定義受信・デッドロックの有無を検査する．さらに，グローバル遷移図の導出完了後，一度も実行しなかった遷移を実行不可能送受信とする．図4．6．1のプロトコル例に従来法1を適用した結果得られるグローバル遷移図を図4．6．2に示す．図4．6．2で，角が丸い長方形はグローバル状態を表し，そのうち網かけのある長方形は安定状態で，特に影付きの安定状態はデッドロックである．これらのグローバル状態では，各プロセス状態名及び各バッファ内信号系列（信号系列は（　）内に）が示されている．図4．6．1には含まれるが図4．6．2のグローバル遷移図には含まれない遷移は，プロセス2の状態SERVICEにある＋ACKであり，これが実行不可能送受信となる．プロセス2の状態FAULTで未定義受信＋ACKが検出されている．



(a) プロセス 1　　(b) プロセス 2

図4．6．1　プロトコルの例（初期状態＝READY，IDLE）

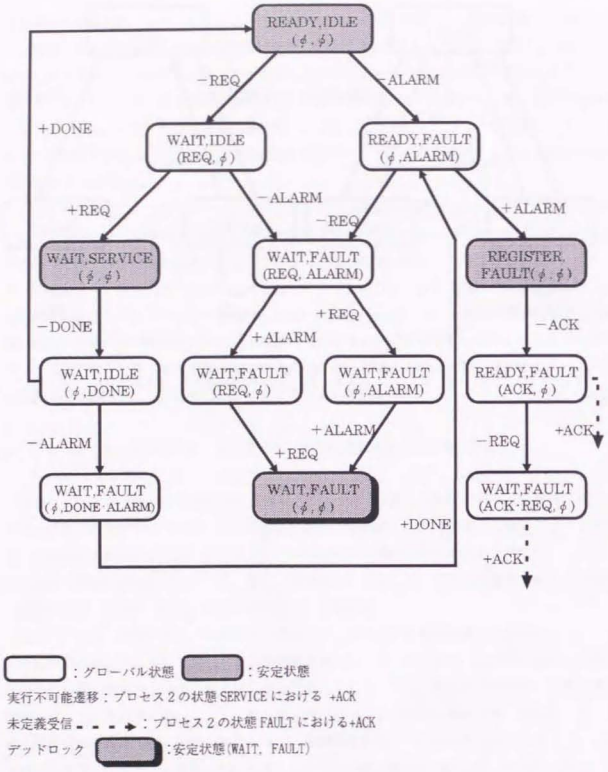


図４．６．２　従来法１を図４．６．１のプロトコル例に適用した結果得られるグローバル遷移図

従来法１の特徴はグローバル状態の導出と蓄積にあり，この蓄積量が所要メモリ量の支配的要素となる．つまり，総グローバル状態数を N_s とすると，所要メモリ量は N_s にほぼ比例する．一方，従来法１では，いったん導出したグローバル状態が既に導出したグローバル状態のいずれかに一

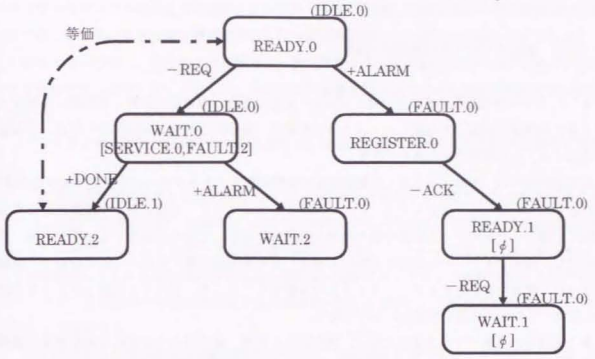
致するか否かを検査するが，この処理量が最も大きい．この検査に２分探索法を採用する場合，その処理量は，$\log_2$(グローバル状態数)に比例する（Supertrace[40]~[44]のようにハッシングを使えば，この処理量を減らすことができる）．グローバル状態数は新グローバル状態の導出と共に増加していく．つまり，この検査に要する処理量の平均値は，$\Sigma\ |\log_2 X|\ /N_s$（X=1，2，…，N_s についての総和）$=\log_2(N_s!)/N_s$ に比例する．また，いったん導出するグローバル状態の個数は，グローバル遷移の個数に等しい．従って，従来法１の総処理量は，グローバル遷移数を N_T として，ほぼ $N_T\cdot\log_2(N_s!)/N_s$ に比例すると考えられる．グローバル状態は，定義４．２．２に示した通り，一般に各プロセスの状態と各バッファの状態の組み合わせであり，従って，プロセスが増えるとグローバル状態数やグローバル遷移数は指数関数的に増大し，従来法１の所要メモリ量と処理量は急激に増加していく．

一方，従来法２ではプロセス毎に実行可能な状態遷移動作をすべて順次展開していく．その結果得られる状態遷移図をプロセス展開図と呼び，プロセス展開図における状態を（プロセス）展開状態と呼ぶ．展開状態を順次導出する過程において，他のプロセスが最低限到達しているはずの状態（これを最小状態と呼ぶ）と安定状態をすべて同時に導出していき，同じ動作展開の繰返しになった場合，つまり，等価な展開状態が既に導出されている場合それ以降の展開は停止する．そして，各展開状態の導出時に未定義受信の有無を検査し，安定状態においてデッドロックの有無を検査することによって，それぞれ誤りを検出する．さらに，展開終了後一度も実行しなかった遷移を実行不可能送受信とする．

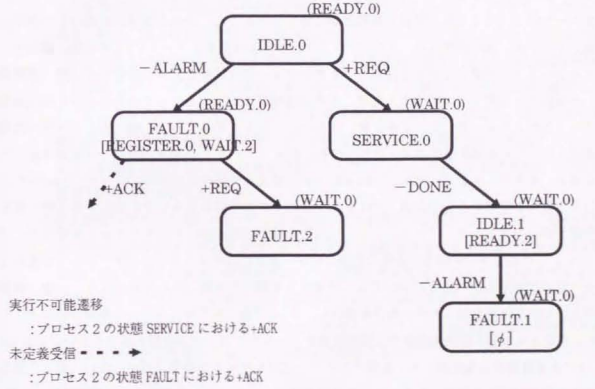
図４．６．１のプロトコル例に従来法２を適用した結果得られるプロセス展開図を図４．６．３に示す．同時に得られる安定状態は図４．６．２に示すものと同一である．図４．６．３で，角が丸い長方形は展開状態を表し，展開状態の名前には，図４．６．１のプロセス状態名の後にドット(.)と数字を付与しているが，これは一つのプロセス状態が一般に複数回展開されるので，このような複数の展開状態を区別するために数字を識別子として使用しているからである．図４．６．３で，各展開状態の最小状態を右上の（　）内に示す．この例ではプロセス数が２であるため，各最小状態は相手プロセス１個のみの状態を表す．また，展開状態名の下にある［　］内の情報は，従来法２では導出しないものであり，後で詳しく述べる．検出されたプロトコル誤りは，本質的に従来法１つまり図４．６．２と同一である．

図４．６．３のようなプロセス展開図を導出する従来法２では，各プロセス毎の展開状態と他プロセスの最小状態を導出し管理するが，プロセスが増加しても従来法１のグローバル状態ほどこれらの状態は急激には増加しない．一方，従来法２では，プロセス展開を停止するための判定とデッドロックの検出に安定状態を使用しているので安定状態をすべて導出する必要がある．この安定状態は各プロセスの状態の組み合わせであり，一般的にはプロセスが増えれば指数関数的に増加する．つまり，従来法２の所要メモリ量は，一般には従来法１より少ないものの，オーダー的にはほぼ同様で，N_s に比例する．また，総処理量はやはり従来法１より少ないものの，オーダー的には同様で，ほぼ $N_s\cdot\log_2(N_s!)/N_s=\log_2(N_s!)$ に比例する．

以上から，従来法の所要メモリ量はいずれもN_sにほぼ比例し，総処理量は$N_T \cdot \log_2(N_s!)/N_s$又は$\log_2(N_s!)$にほぼ比例する．即ち，従来法の所要メモリ量と処理量の支配的要素は，極めて多くのグローバル状態や安定状態を導出する点にある．



(a) プロセス 1



(b) プロセス 2

図 4.6.3　図 4.6.1 のプロトコル例を従来法 2・提案検証法によってプロセス展開した結果

4.6.3 段階的プロトコル検証法の原理

4.6.2での分析結果に基づき，特にプロセスが多い場合，従来法に比較して検証に必要なメモリ量と処理量を削減できる新しいプロトコル検証法（段階的プロトコル検証法と呼ぶ）を本節で提示する．その原理は次の通りである．

提案検証法では，所要メモリ量と所要処理量の支配的要素であるグローバル状態（含安定状態）をできる限り列挙しない方針とする．そして，検出すべきプロトコル誤りをその性質に応じて2種類に分類し，それぞれに必要最小限の処理を段階的に実行して別々に検出する方針とする．具体的には，検出にグローバル状態を直接は必要としない未定義受信・実行不可能送受信と，直接グローバル状態を必要とするデッドロックとに分類する．そして，これら2種類のプロトコル誤りを検出するための処理を2段階に分けて行う．

第1段階の処理をプロセス検証と呼び，従来法2と同様，各プロセスの実行可能な遷移と遷移先の状態をすべて導出することによってプロセス展開図を得ると同時に未定義受信を検出する．そして，展開終了後に実行不可能送受信を検出する．この第1段階処理では，従来法2と同様，一般に一つのプロセス状態が複数の展開状態に展開されるため，それら複数の展開状態の識別を可能とするため，展開状態名は，元のプロトコルにおける状態名にドット(.)とその後に識別子として0から1づつ更新する数字を付与したものとする．実行可能な遷移と遷移先の状態（展開状態）の導出は，送信を一つづつとりあげ，その送信遷移に対応するすべての受信遷移を順次追加していく方針とする．ただし，このとき，プロパゲーション[32]によって，既に導出されていた受信遷移とその先の展開状態がコピーされて追加される（ドット（.）以下の識別子は更新する）ことがある．例えば，あるプロセス $i(1 \leq i \leq N)$ との送信又は受信の前に別のプロセス $j(1 \leq j \leq N,\ j \neq i)$ から信号を受信する場合には，プロセス i とのその送受信の後でもプロセス j からその信号の受信を実行できるので，プロセス i との送受信を持つ状態が持つプロセス j からの受信を，プロセス i との送受信の後遷移する先の状態にコピーする．このようなプロパゲーションを含むプロセス毎の状態遷移図の展開については，従来法2を論じた文献[32]の方法に従う．一方，展開の停止については従来法2の停止条件を強化し，4.5で論じたアサイクリック展開法に従う．しかし，所要メモリ量と検証処理量の支配的要素となる安定グローバル状態を導出しないこととし，4.5にある展開停止条件のうち，安定グローバル状態を使用する条件は採用しない．更に，第2段階の処理で利用する情報として，各送信遷移の信号の受信により遷移する先の展開状態（以下，受信先状態と呼ぶ）をすべて，その送信遷移の先の展開状態に蓄えておく．

第2段階の処理をグローバル検証と呼び，第1段階の処理の結果得られる実行可能プロセス遷移を利用することによって，実行可能な各プロセスの遷移をできるだけまとめて実行した後のプロセス展開状態から構成されるグローバル状態を順次導出する．結果的には，実行可能な遷移を

持たないグローバル状態と受信遷移を含む複数の遷移（分岐）を持つグローバル状態を中心に導出する．そして，実行可能な遷移を持たない安定状態が得られた場合，これをデッドロックとして検出する．つまり，従来法2とは異なり，安定状態をすべて列挙した後その中からデッドロックを抽出することはしない．そして，実行可能な遷移を持つグローバル状態の導出を最小限に留める．

このような第2段階処理をできる限り効率よく実現するため，以下に示す特別な処理を採用する．

①第1段階のプロセス検証処理において，送信された各信号を受信した後遷移する先の展開状態（受信先状態）を保存しておくが，保存したこの受信先状態情報を第2段階の処理で利用することによって，信号を受信し遷移した後のグローバル状態を迅速に生成する．

②連続して実行可能な同一プロセスの遷移は，途中でデッドロックが発生することがないので，受信遷移の分岐がない限り送信遷移を一括して実行し，その実行後のグローバル状態を導出する．

③ある安定状態が複数の送信遷移を持つ場合，送信遷移を持つプロセスの状態が遷移しない限りこれらの送信は以降常に実行可能であり，デッドロックの検出に関しては，それらの実行順序のすべての順列組み合わせを網羅して検査する必要はない．つまり，一つの送信遷移（以下，第1の送信遷移と呼ぶ）以降のグローバル状態を導出した後，他の送信遷移（以下，第2の送信遷移と呼ぶ）以降のグローバル状態の導出で第1の送信遷移が現れてこれ以降もグローバル状態を導出し続けてデッドロックが検出されれば，既に導出した第1の送信遷移以降でも同じデッドロックが必ず検出されている．つまり，第2の送信遷移後に第1の送信遷移が現れた場合，それ以降のグローバル状態の導出は無駄となる．このような無駄を回避するため，各安定状態においていったん実行した送信遷移を「検証終了遷移」と呼んで管理する．

④第1段階のプロセス検証において，展開停止条件を満たす展開状態，つまり，他の展開状態と等価な展開状態を検出するが，このような等価展開状態が第2段階のグローバル状態導出において現れた場合，以降のグローバル状態とその遷移は繰返しとなるので，それ以降の処理は中止する．この結果，第2段階のグローバル状態導出では，いったん導出したグローバル状態を保存しておき，あらたに導出したグローバル状態と比較し同一である場合それ以降の展開を停止するという処理は必要なくなる．

4.6.4 段階的プロトコル検証法のアルゴリズム

本節では，4.6.3で示した原理に基づく提案プロトコル検証法のアルゴリズムを示す．与えられたプロトコルに含まれるプロセスの個数を N とする．第1段階のプロセス検証のアルゴリズムは次の通りである．

[アルゴリズム4.6.1]（プロセス検証＝提案検証法の第1段階の処理）

導出すべきプロセス展開図をPとし，処理対象とする送信を表す変数をa，未処理の送信を蓄えておくFIFO型キューをWとする．

step1　Pにおける各プロセスi（i=1, 2, …, N）の初期展開状態を$o_i.0$とする．また，Pの各初期展開状態$o_i.0$の最小状態を$|o_j.0|$（$1 \leq j \leq N$, $j \neq i$）とする．更に，各プロセスiの初期状態o_iが持つ送信をすべてWに蓄積する．

step2　Wから送信を一つ取り出してaとし，この送信aを持つプロセスをi（$1 \leq i \leq N$）とする．Pのプロセスiの展開図において，送信aによる遷移とその先の展開状態を追加する．さらに，この展開状態についてサブルーチンTを実行する．ただし，Wが空だった場合はstep6に進む．

step3　プロセスiについて，送信aの遷移元展開状態から連続して実行可能な全受信遷移とそれらの先の展開状態をaの遷移先展開状態にコピーするというプロパゲーション処理を行う．このプロパゲーション処理は文献[32]の方法に従う．ここで，各展開状態を導出する毎にサブルーチンTを実行する．

step4　送信aの信号を受信するプロセスをj（$1 \leq j \leq N$, $j \neq i$）とする．Pのプロセスjの展開図において，aの遷移元展開状態の最小状態から到達できる範囲内の各展開状態が，aの信号を受信することが可能か否か調べ，受信可能であればその展開状態にaの信号の受信遷移と遷移先の展開状態を導出し追加する．ここで，各展開状態を導出する毎にサブルーチンTを実行する．

step5　step4で導出した各受信遷移について，その遷移元の展開状態から連続して実行可能な全受信遷移とそれらの先の展開状態を，その遷移先の展開状態にコピーするというプロパゲーション処理を行う．このプロパゲーション処理は文献[32]の方法に従う．ここで，各展開状態を導出する毎にサブルーチンTを実行する．これらの処理の完了後，step2に戻る．

step6　元のプロトコルで定義されている送受信のうち，プロセス展開図Pに含まれないものを実行不可能送受信とし，終了する．

サブルーチンT：導出した各展開状態に対し以下の5点の処理を行う．

- その展開状態が受信によって遷移し，かつその受信が元のプロトコルに定義されていない場合，その受信遷移を未定義受信とし，以下の4点の処理はスキップする．
- その展開状態の最小状態を導出しPに追加する．（文献[32]の方法による）
- その展開状態が受信によって遷移する場合は，それに対応する送信の遷移先の展開状態に，この展開状態を「受信先状態」として保存する．
- その展開状態について以下の3条件を満たす展開状態が他に存在するか否か検査し，存在する場合「等価展開状態」とマークする．
  - ①両展開状態が対応する元のプロセス状態が同一である（ドット(.)以下の識別子を除いて展開状態名が同一である）．

  - ②両展開状態が持つ最小状態に対応する元のプロセス状態が同一である（ドット(.)以下の識別子を除いて展開状態名が同一である）．
  - ③各プロセスの初期展開状態から，両展開状態とそれぞれが持つ最小状態に到る送受信遷移を実行した後のバッファ状態が同一である．
- その展開状態が「等価展開状態」とマークされておらずかつ元のプロトコルで送信を持つ場合，それら送信をすべてWに追加する．

[アルゴリズム終]

次に，第2段階のグローバル検証ではデッドロックの検出を行う．そのアルゴリズムは次の通りであり，概略フローチャートを図4.6.4に示す．ただし，処理の対象とするグローバル状態をカレントグローバル状態又は単にカレント状態と呼ぶ．

[アルゴリズム4.6.2]（グローバル検証＝提案検証法の第2段階の処理）

カレントグローバル状態を表す変数をGとし，未処理グローバル状態を保存するためのFILO型スタックをZとする．

step1　$Z := |G_0|$（初期グローバル状態）とする．

step2　$Z = \phi$であれば終了する．$Z \neq \phi$であればstep3に進む．

step3　Zに最後に追加された要素を取り出しGとする．

step4　Gの全バッファに含まれる信号の総数が1以上であればstep5に進み，0個であればstep7に進む．

step5　Gの全バッファに含まれる信号のうち，いずれかのバッファの最後尾にある任意の信号をaとし，信号aを受信するプロセスをrとする．プロセスrが信号aの受信によって遷移する先の展開状態を，アルゴリズム1のサブルーチンTで保存しておいた受信先状態の情報を利用してすべて求め，s_1, s_2, …, s_mとする．ただし，信号aの受信遷移がまったくない場合は直ちにstep2に戻る．

step6　各j（j=1, 2, …, m）についてstep6-1からstep6-6までの処理を行うことによって，Gを基にaの受信遷移あるいはそれに連続する送信遷移を実行した後のグローバル状態を導出して順にG_1, G_2, …, G_nとし，$Z := Z \cup |G_n, \cdots, G_2, G_1|$（$G_n$から$G_1$の順序でZに追加する）によってZを更新し，step2へ戻る．

step6-1　（s_jがGと同じ遷移系列上にあるか否かの検査）

s_jの各最小状態について，その最小状態からGにおける同プロセスの展開状態に到達可能な場合，その最小状態とGにおける同プロセスの展開状態が一致する場合，またはGにおける同プロセスの展開状態からその最小状態に到達可能な場合，step6-2以降の処理を行い，さもなくばs_jについてはstep6-2以降の処理を行わず，s_{j+1}の処理へ進む．

step6-2　（連続する送信遷移の先の展開状態の導出）

s_jの持つ遷移がすべて送信遷移の場合は，s_j以降連続する送信遷移のみで到達可能でかつ途中の展開状態が受信遷移を持たない最も先端の展開状態をすべて導出し，u_1, u_2, …, u_h

とする．

step6-3 （グローバル状態の生成）

プロセス r の展開状態が s_j で，r 以外のプロセスの展開状態が，G における展開状態と s_j の最小状態のうち，より先へ進んでいる展開状態であるようなグローバル状態 G' を生成する．ただし，step6-2 において展開状態 u_1，u_2，…，u_h を導出した場合は，G' のプロセス r の展開状態は s_j ではなく u_i とし，step6-4，6-5 ではすべての i（i=1，2，…，h）の値に対して処理を行う．

step6-4 （検証終了遷移の検査）

グローバル状態 G から G' へ到る各プロセスの遷移系列をすべて調べ，初期グローバル状態から G へ到る途中のいずれかのグローバル状態において「検証終了遷移」として登録した遷移が 1 個でも存在する場合は，G' を処理の対象外とする．

step6-5 （等価展開状態の検査）

グローバル状態 G から G' へ到る各プロセスの遷移系列を G' の展開状態まで含めてすべて調べ，「等価展開状態」がある場合は，G' を処理の対象外とする．

step6-6 以上により処理の対象外とならなかった G' が Z に追加すべきグローバル状態 G_{k+1}，G_{k+2}，…，G_{k+d} である．ここで，d は s_j に関して処理の対象外とならなかったグローバル状態の個数，G_k は s_j から s_{j-1} までの処理で最後に得られたグローバル状態である．

step7 カレント状態 G は安定状態であり，G が持つ送信遷移をすべて挙げ，t_1，t_2，…，t_m とする．G が送信遷移を全く持たない場合は step9 に進む．

step8 各 j（j=1，2，…，m）について step8-1 から step8-5 までの処理を行うことによって，安定状態 G から 1 個あるいは複数の連続する送信遷移を実行した後のグローバル状態を導出して順に G_1，G_2，…，G_n とし，$Z := Z \cup \{G_n, \ldots, G_2, G_1\}$（$G_n$ から G_1 の順序で Z に追加する）によって Z を更新し，step2 へ戻る．

step8-1 （検証終了遷移の検査）

t_j が，初期グローバル状態から G に到る遷移系列の中のいずれかのグローバル状態において「検証終了遷移」として登録されていれば，t_j については step8-2 以降の処理を行わず，t_{j+1} の処理へ進む．

step8-2 （連続する送信遷移の先の展開状態の導出）

送信遷移 t_j の遷移先展開状態を s_j とする．s_j の持つ遷移がすべて送信遷移の場合は，s_j 以降連続する送信遷移のみで到達可能でかつ途中の展開状態が受信遷移を持たない最も先端の展開状態をすべて導出し，u_1，u_2，…，u_h とする．

step8-3 （等価展開状態の検査）

s_j が「等価展開状態」である場合は，s_j を処理の対象外とする．また，s_j から u_i（i=1，2，…，h）へ到るまでに「等価展開状態」がある場合は，u_i を処理の対象外とする．

step8-4 （グローバル状態の生成）

送信遷移 t_j と展開状態 s_j を持つプロセスを r とする．G におけるプロセス r の展開状態を，処理の対象外とならなかった s_j または各 u_i まで進めて，Z に追加すべきグローバル状態 G_{k+1}，G_{k+1}，…，G_{k+d} を得る．ここで，d は処理の対象外とならなかった展開状態 s_j または u_i の個数，G_k は t_j から t_{j-1} までの処理で最後に得られたグローバル状態である．

step8-5 （検証終了遷移の登録）

以上で t_j に関する処理が終了するので，t_j を G における「検証終了遷移」として登録し，t_{j+1}，t_{j+2}，…，t_m によるグローバル状態の導出時，およびそれらのグローバル状態を基にしたすべてのグローバル状態の導出時に，t_j を含む遷移系列によるグローバル状態の導出を再度行わないための判定に利用する．

step9 G をデッドロックとして登録した後，step2 へ戻る．

［アルゴリズム終］

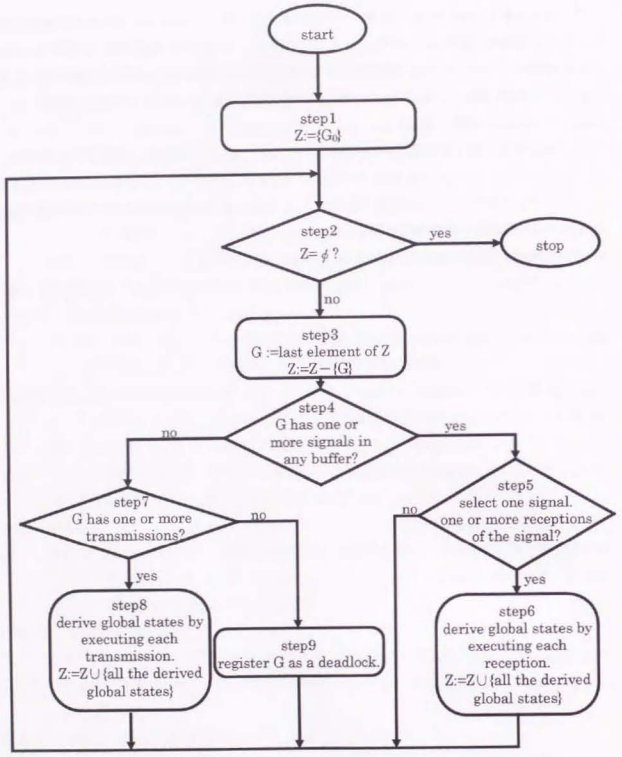


図4．6．4　提案検証法におけるグローバル検証のアルゴリズム

[定理4．6．1]（アルゴリズム4．6．1と4．6．2の正当性）

アルゴリズム4．6．1と4．6．2は正当である．つまり，両アルゴリズムによって，定義4．2．5のプロトコル誤りをすべて有限時間で検出することができる．

[定義終]

[定理4．6．1の証明]

アルゴリズム4．6．1は，4．5で論じたアサイクリック展開法に比較すると，以下の2点が異なっている．

(1)デッドロック検出のための処理がない．（未定義受信・実行不可能送受信の検出処理は同一である．）

(2)安定状態を一切導出しておらず，プロセス展開の停止条件に安定状態を利用していない．そして，この停止条件はアサイクリック展開法の停止条件のサブセットになっている．

この(2)によって，4．5のアサイクリック展開方法ではプロセス展開が停止する場合でもアルゴリズム4．6．1では停止しないことがある．しかし，アルゴリズム4．6．1のプロセス展開停止条件は，文献[32]の方法のスーパーセットとなっており，展開は必ず停止する．つまりアルゴリズム4．6．1は有限時間で終了する．また，実行可能なプロセス遷移と到達可能なプロセス展開状態はすべて列挙する．従って，未定義受信・実行不可能送受信を検出し損ねることはない．

一方，アルゴリズム4．6．2では，これらの実行可能なプロセス遷移と到達可能なプロセス展開状態を利用してグローバル状態を導出しデッドロックの有無を検査する．これらプロセス遷移とプロセス展開状態の個数は有限であり，アルゴリズム4．6．2が有限時間で停止することは明らかである．アルゴリズム4．6．2では，導出するグローバル状態を可能な範囲で最小限に抑えている．その特徴的な処理は，step4でカレント状態Gがバッファに信号を持つ場合と持たない場合に分けて処理している点と，step6とstep8においていくつかの特別な処理を実行している点にある．これらの処理がすべて有限時間で完了することは明らかであるので，これらによってデッドロックを検出し損ねることがないことを証明すればよい．

step4での場合分け処理の結果，カレント状態Gがバッファに信号を1個以上持つ場合step5に進み，バッファ内の信号を1個(a)のみとり上げて処理を進める．このため，Gが持つ送信遷移やバッファ内の他の信号の受信の実行により到達するグローバル状態を導出していないように見える．しかし，実際にはstep5において，アルゴリズム4．6．1のサブルーチンTによって得られる受信先状態を利用しており，バッファ内信号を受信するプロセス遷移はカレント状態が持つものだけでなく，それ以降のすべてのグローバル状態における全受信遷移を導出している．つまり，Gが持つ送信遷移やバッファ内の他の信号の受信を実行した後で信号aを受信した場合もすべてカバーしている．もしデッドロックがあるとすれば，それは安定状態であり，バッファ内信号をすべて受信処理した後の状態である．従って，step4，5でのこのような処理によって，デッドロックを検出し損ねることはない．

step5において，いずれかのバッファの最後尾の信号aを取り出して，基本的にはstep6でその信号の受信遷移先の状態を，遷移後のグローバル状態の一部としている．この処理は当該バッファ内の信号を連続して受信したことに等価であるが，アルゴリズム4．6．1のプロセス検証によって，このバッファに蓄積されている信号はすべて受信可能であることが分かっており，これら

一連の受信の途中でデッドロックになることはない．従って，この処理によってデッドロックを検出し損ねることはない．

一般に信号 a を受信する遷移は複数あるが，その中には，初期グローバル状態からカレント状態 G に到るまでの間でも，G 以降でも実行しないものがあり得る（G を含まない別のグローバル状態遷移系列上で実行される）．step6-1 は，そのような不用な受信を除外するための処理である．

step6-2 および step8-2 において，抽出した展開状態以降に送信遷移が続く場合，これら送信をすべて実行した後の状態をカレント状態 G の遷移先としてグローバル状態 G'を生成する．ただし，受信遷移の分岐がある場合は，その分岐を持つ状態を遷移先グローバル状態とする．これら送信は必ず実行可能であり，途中でデッドロックが生じることはあり得ない．従って，この処理によってデッドロックを検出し損ねることはない．

step6-3 において，信号 a を受信するプロセス r 以外のプロセスについて，プロセス r の遷移先状態の最小状態がカレント状態 G より進んでいれば，この最小状態を遷移先グローバル状態とする．a の受信時において，r 以外のプロセスでは，その最小状態までは必ず進んでいるので，最小状態をそのプロセスの遷移先としてもデッドロックを検出し損ねることはない．

安定状態から送信遷移処理を行った場合，step8-5 によって，その送信遷移をその安定状態における「検証終了遷移」として登録している．一般に，送信遷移は，他のプロセスの状態如何にかかわらず実行できるため，step6-4 や step8-1 において，その安定状態から出る別の送信遷移で始まる遷移系列を検査する中で，検証終了遷移を実行すると，既に検査した系列と同じ遷移の系列を再度検査することになる．このため，その先の検証を中止しても，デッドロックを検出し損ねることはない．

step6-5 および step8-3 において，カレント状態から遷移先の状態の間に等価条件を満たす展開状態が含まれている場合，以降の遷移系列は対応する等価展開状態以降の遷移系列と同一であるため，カレント状態以降を検査する必要はない．従って，step6-5 および step8-3 の処理によってデッドロックを検出し損ねることはない．

このように，step6 と step8 において複数の遷移をまとめる等の特別処理を施しているが，これらによってデッドロックを検出し損ねることはない．

以上から，アルゴリズム 4．6．1 と 4．6．2 とによって定義 4．2．5 で示したプロトコル誤りをすべて有限時間で検出できることが証明できた．

［証明終］

4．6．5　段階的プロトコル検証法の適用例

図 4．6．1 のプロトコル例に提案検証法のプロセス検証（アルゴリズム 4．6．1）とグローバル検証（アルゴリズム 4．6．2）を適用した結果を図 4．6．3 と図 4．6．5 にそれぞれ示す．

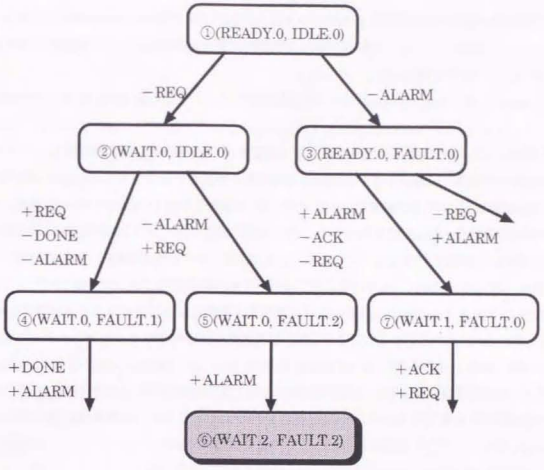


図 4．6．5　グローバル検証を図 4．6．1 のプロトコル例に適用した結果

提案検証法のアルゴリズム 4．6．1 によって得られるプロセス展開図が，従来法 2 によって得られるプロセス展開図と異なる点は，信号を受信した後遷移する先の展開状態（受信先状態）の情報が各送信遷移後の展開状態に付与されている点にある．このような受信先状態を，図 4．6．3 では送信遷移後の展開状態の名前の下の［　］内に示す．例えば，プロセス 1 の READY.0 から WAIT.0 への送信遷移－REQ の受信先状態は，プロセス 2 の SERVICE.0 及び FAULT.2 の 2 つである．アルゴリズム 4．6．1 つまり第 1 段階の処理において各プロセス状態で実行可能遷移を導出するとき，従来法 2 と同様に，未定義受信と実行不可能送受信を検出する．

アルゴリズム 4．6．2 によって，次の通り図 4．6．5 が導出され，デッドロックが検出される．まず，step1 によって初期グローバル状態①（READY.0, IDLE.0）（バッファの記述は省略した．以下同様である．）が生成され，これが未処理グローバル状態のスタック Z の要素となり，step3 で処理対象のカレントグローバル状態 G となる．このカレント状態のバッファはすべて空であるため，step4 の判定の結果 step7 に進む．step7 では送信遷移の有無を検査し，プロセス 1 の－REQ とプロセス 2 の－ALARM をこの順序で得たとする．step8 において，初期グローバル状態で送信－REQ 及び－ALARM を実行することによってグローバル状態②(WAIT.0, IDLE.0) と③(READY.0,

FAULT.0)をそれぞれ生成し，これらをスタックZに③・②の順序で追加保存する．また，step8-5によって，READY.0における信号REQの送信遷移を安定状態①における「検証終了遷移」として登録する．その後step2に戻る．

step2の後のstep3では，グローバル状態②をスタックZから取り出してカレント状態Gとする．このカレント状態ではバッファに信号REQが蓄積されているので，step4で「信号が1個以上あり」と判定してstep5に進み，受信遷移として+REQを得，信号REQの受信先状態としてSERVICE.0とFAULT.2を得る．step6でまず展開状態SERVICE.0を処理する．つまり，step6-1を経て，step6-2ではSERVICE.0の先に連続する送信－DONEと－ALARMを連続して実行し遷移した後のグローバル状態④(WAIT.0, FAULT.1)を生成する．次に，展開状態FAULT.2について処理し，グローバル状態⑤(WAIT.0, FAULT.2)を生成する．そして，生成したグローバル状態⑤と④をこの順序でZに追加する．以上で，グローバル状態②に関する処理step6が完了し，step2に戻る．

次に，step2の後のstep3でグローバル状態④をZから取り出してカレント状態Gとする．step4でGのバッファを検査すると，信号DONEとALARMが同一バッファに蓄積されているので，step5へ進み，最後尾にある信号ALARMの受信を行う．図4．6．3の通り，この信号ALARMの受信先状態は，ALARMを送信する前の展開状態IDLE.1の受信先状態READY.2以降であるが，カレント状態④のWAIT.0からこのREADY.2へ遷移するまでにREADY.2という等価展開状態が存在するので，step6-5によってこの受信先状態についての処理は中止する．

再度step2に戻り，step3でグローバル状態⑤をZから取り出してカレント状態Gとする．このGでは信号ALARMが蓄積されているので，step4を経てstep5,6に進み，グローバル状態⑥(WAIT.2, FAULT.2)を生成する．

更にstep2,3に戻って，グローバル状態⑥をZから取り出してカレント状態Gとする．このGは蓄積信号が一切ない安定状態であり，step7に進んで，いずれのプロセスも送信遷移を持たないことが判定され，step9でこのグローバル状態⑥をデッドロックとして登録する．

次に，グローバル状態③における同様な処理によってグローバル状態⑦(WAIT.1, FAULT.0)を生成した後，グローバル状態③から出る右側の遷移系列について説明する．step5の処理によって，遷移先として，バッファ内信号ALARMの受信先状態WAIT.2が求まるが，この展開状態に到るまでに安定状態①の検証終了遷移である（READY.0における）信号－REQの送信遷移を実行するので，step6-4によって，このような状態遷移系列の検証は行わない．つまり，グローバル状態③以降の右側の遷移系列の処理は中止し，step2に戻る．最後にグローバル状態⑦をカレント状態とする処理を終えると，未処理グローバル状態を持つスタックZが空となっており，グローバル検証処理が完了する．

4．6．6　段階的プロトコル検証法の評価

本項では，提案プロトコル検証を従来法と具体的に比較評価し，提案法の有効性を明らかにする．評価の尺度は検証に必要なメモリ量と処理量とし，原理的な定性的評価と，実際のプロトコル例を対象とした実験による定量的評価の２つの評価法を採用する．

（1）　原理的な定性的評価

本節で提案したプロトコル検証法は，プロセス検証とグローバル検証とから構成される．前者のプロセス検証では，4．6．2で述べた通り，プロトコル検証の所要メモリ量と所要処理量の支配的要素であるグローバル状態（含安定状態）を導出しないため，その所要メモリ量と処理量は，一般に従来法1と2に比較して少ない．一方，後者のグローバル検証では，第1段階のプロセス検証によって実行可能と判定されたプロセス遷移のみによってグローバル状態を生成するが，4．6．2で述べた通り，同一プロセスが持つ連続送信を一括して実行する等，アルゴリズム2のstep6とstep8等に示した特別な処理によって効率を向上している．また，従来法1と2では，いずれもいったん導出したグローバル状態や安定状態をすべて保存しておき，グローバル状態や安定状態を導出するごとに，保存したグローバル状態や安定状態と一致するか否か検査する必要がある．しかし，本提案検証法では，支配的要素となるこのようなグローバル状態や安定状態の保存や比較は一切不要である．以上から，特にプロセスが多い場合，従来法に比較して本提案法の所要メモリ量と処理量は原理的に大幅に少なくなる．

（2）　実験による定量的評価

評価実験には，4．5．5で示した，リング状LANに接続されたプロセス間をトークンとデータが循環するトークンリングLAN用プロトコルを使用した．実際には，トークンを持つプロセスだけがデータを送信し，自プロセスが送信したデータを受信したときは破棄し，他プロセス宛のデータは順次隣のプロセスに渡していく．各プロセスが持つ状態は6個，遷移は12個である．プロセス数は任意とすることができる．

提案検証法の効果を明確にするため，提案検証法に加え，一般に所要メモリ量・処理量が共に従来法1よりも少ないアサイクリック展開法（4．5）についても同一条件で検証を実施した．所要メモリ量と処理量をそれぞれ評価するため，保存すべき状態の個数と検証処理に要したCPU使用時間を測定した．実験評価に使用した計算機は，主メモリが256MBで処理能力が約250SPECint92のワークステーションである．

実験結果を図4．6．6に示す．図4．6．6で，提案検証法の状態数は，保存すべき安定グローバル状態数が0個であるので，プロセス展開状態の個数を表す．またアサイクリック展開法では，一般に安定状態数がプロセス状態数より多いので，図4．6．6の状態数は安定状態数を表す．アサイクリック展開法ではプロセス数Nが15以上になると，使用メモリが256MBの範囲を超え検証できなかった．実際のプログラムでは安定状態やプロセス展開状態を保存するために必要なメモ

リはプロセス数に依存するが，それぞれ 100～数 100 バイト程度である．特にプロセスが多くなると，アサイクリック展開法においてはプロセス展開状態よりも安定状態が圧倒的に多くなり，これが所要メモリ量の観点から支配的要素になり検証が不可能となる．提案検証法はアサイクリック展開法に比べると，展開停止条件が緩いためプロセス展開状態が多くなるが，安定状態を記憶する必要がないので，総合的には特にプロセスが多くなるほど提案検証法の所要メモリ量はアサイクリック展開法に比較して著しく少なくなる．

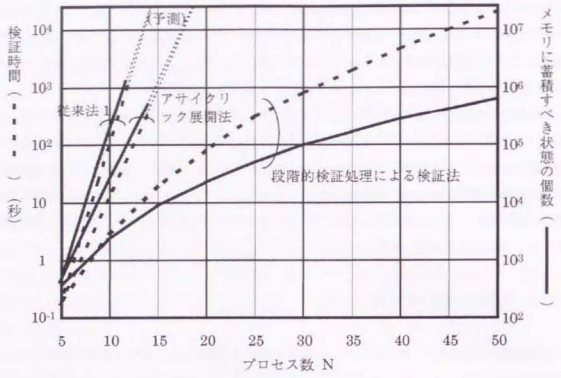


図 4．6．6　段階的検証処理によるプロトコル検証法の
蓄積状態数(メモリ量)と検証時間(処理量)の削減効果

図 4．6．6 で，提案検証法の CPU 時間はアサイクリック展開法に比較してプロセス数に係わらず小さいが，特にプロセスが多くなりプロトコル規模が大きくなるほど，その差は著しくなる．例えば，プロセス数が 12 の場合，提案プロトコル検証法によって検証処理量は従来法 1 に比較して 100 分の 1 以下になり，またアサイクリック展開法に比較して 10 分の 1 以下になる．また，所要メモリ量は，従来法 1 に比較して数 100 分の 1 程度になり，アサイクリック展開法に比較して数 10 分の 1 程度になる．この削減度合い，つまり本提案プロトコル検証法による効果は，プロセスの増加に伴って指数関数的に急激に増大していく．また，従来法とアサイクリック展開法では使用コンピュータのメモリ制限からそれぞれ 12 個・14 個以下のノード(プロセス)を持つ場合しか検証できなかったが，本段階的検証法によれば 50 個以上のノードを持つ場合の検証が可能であった．また，提案検証法では，第 2 段階のグローバル検証に要する処理時間は，第 1 段階のプロセス検証に比較すると，いずれの場合においても圧倒的に小さい．これは 4．6．4 の step6 と step8 等で採用した各種の特別処理による効果である．

以上に示した通り，提案検証法によって，従来法に比較して，所要メモリ量・処理量ともに大幅に削減することが可能となる．特にプロセスが増えてプロトコル規模が大きくなるとその差が顕著となり，提案検証法の効果が著しく大きくなる．そして，提案検証法によって，プロセスが多く従来は不可能だった大規模なプロトコルの検証が可能となる．

4．6．7　まとめ

本節では，検出に要する検査情報に応じてプロトコル誤りを 2 分し，それぞれに適した最小限の処理を適用することによって効率よく誤りを検出する新しいプロトコル検証法を提案した．具体的には，信号定義の過不足のようにグローバル状態の導出が検出に必須ではない誤りは，与えられたプロトコルをプロセス毎に展開し検査することによって検出し，グローバル状態に直接関係するデッドロックについては，前者のプロセス展開結果を利用し，最小限のグローバル状態の列挙によって検出する方法を提示した．そして，この提案検証法が従来法に比較して原理的に所要メモリ量や検証処理量が改善されることを論じるとともに，プロトコル例を適用した実験によって本提案法が所要メモリ量と処理量を大幅に削減でき，従来は検証できなかった大規模なプロトコルを検証可能にすることを実証した．具体的には，従来法に比較して，所要メモリ量と検証処理量を 1/100 以下に削減でき，プロトコル規模，特にプロセスが多いほど検証効率を大きく向上できることを示した．

4．7　むすび

本章では，通信プロトコル検証の効率を大幅に向上できる 4 件の新技術を提示しその効果を論じた．具体的には，まず，検証すべきプロトコルの規模を削減するため，信号を送受信する相手プロセスが 1 個に限定されるプロセスを削除して検証する方法を提示した．このプロセス削減法によって，例えばプロセスを 1 個削減した場合には検証処理量が 1/3～1/5 に減少し，プロセスを 2 個削減した場合には検証処理量が 10 数分の 1 に減少する．

次に，プロトコル検証自体に要する処理量・メモリ量を削減して検証効率を向上できる方法として，プロセス状態遷移一括処理法，プロセス状態遷移図アサイクリック展開法，段階的プロトコル検証法を提示した．これらの 3 検証法によって，プロトコル検証に要する処理量・メモリ量を削減し検証効率を大幅に向上でき，特に，段階的プロトコル検証法によって，検証効率を 100 倍以上に向上できる．

これら 3 個のプロトコル検証法とプロセス削減法とは独立であり，組み合せて適用することができる．その結果，例えば，段階的プロトコル検証法とプロセス削減法を組み合せて適用するこ

とによって，プロトコル検証の効率は数 100 倍以上に向上できることとなる．従って，実用されている規模のプロトコルの検証を可能にするために設定した，「検証効率の 100 倍以上の向上」という研究目標は十分に達成することができた．しかも，これらの新技術は，例えばプロセスが多くプロトコル規模が大きくなるほど，検証効率向上の効果も大きくなるという性質があり，大規模なプロトコルへの適用性が高いと言える．

第５章　通信ソフトウェア仕様の意味検証

５．１　まえがき

通信ソフトウェア仕様の検証には，プロトコル検証だけでは完全ではなく，意味検証も必要である．しかし，２．３で述べた通り，これまで研究し開発されてきた意味検証技術については，以下に示す問題があった．

①一般に大規模な通信ソフトウェア仕様は段階的に作成していくため，途中段階では不完全なものとなるが，そのような途中段階での不完全な仕様の検証を行う技術の検討はほとんどなされていなかった．

②通信ソフトウェア仕様は一般に規模が大きく複雑になることが多く，このため作成者自身でもそのような仕様の理解が困難になることが少なくない．このような仕様の理解を容易にする技術についてもほとんど検討がされていなかった．

③通信ソフトウェア仕様は，それに対する原始要求を満たす必要があるが，そのような要求の充足性を判定できる実用的な手法が開発されていなかった．

本論文では，問題①を解決するため，不完全な通信ソフトウェア仕様を効率よく段階的に検証できる新しいプロトタイピング手法を５．２で論じる．また，問題②を解決するため，与えられた仕様を，より簡明な表現に自動的に変換することによって仕様の理解性を向上し，意味検証の効率向上を図る手法を５．３で論じる．更に，問題③を解決するため，仕様が満たすべき要求条件を拡張状態遷移図で表現し，このような要求条件を与えられた仕様が満たすか否かを機械的に効率よく検査する手法を５．４で論じる．

５．２　抽象処理を用いたプロトタイピングによる不完全仕様の意味検証

５．２．１　概要

本節では，不完全な仕様を段階的に意味検証する方法として，新しいプロトタイピング法を提案する．提案プロトタイピング法では，不完全な通信ソフトウェア仕様を実行可能なプログラムに自動的に変換して実行し，その結果を出力する．通信ソフトウェア仕様の記述言語には，ITU-T の国際標準である SDL (Specification and Description Language)を採用し，自動変換して得られるプログラムの記述言語には C を採用した．本プロトタイピング技術がこれまでの類似の技術と異なる点は，例えば，変数に値が設定されていない，関数の中身が未定義である等，与えられた通信ソフトウェア仕様が不完全であることを前提としており，実行結果を確かなものと不確かなものとに明確に区別して出力することと，実行に必要なデータの最大限の自動設定を実現したことにある．これによって，部分的・段階的な仕様の作成とその検証を容易にし，その結果，通

信ソフトウェア仕様作成の総合的な効率を向上できる．

5.2.2　抽象処理を用いたプロトタイピング法の原理

(1) プロトタイピングに対する要求条件と研究課題

プロトタイピングの目的は，通信ソフトウェア仕様を基にプロトタイッププログラムを作成して実行し，実行結果を検査することによって，与えられた仕様に含まれる誤りを検出するとともにあいまいだった仕様部分を具体化させていくことにある．本論文では，特に仕様誤りを検出する検証に主目的を置く．このようなプロトタイピングでは，プロトタイッププログラムを安価で迅速に作成できることが基本要求条件となる．プロトタイピングに関する従来の検討では，この基本要求条件を満たすため，仕様からプロトタイッププログラムを効率よく作成する方法に主眼を置くことが多かった．

一般に通信ソフトウェアは規模が大きくかつ複雑であり，このため仕様の完成には多量の作業を必要とする．そこで，本論文では，効果的な意味検証を実現するため，上述の基本要求条件に加えて以下の(a)と(b)の要求条件を設定し，これら要求条件をすべて満たすプロトタイピング手法の開拓を研究課題とした．

(a)仕様の段階的な作成を支援できること

仕様作成の基本的な作業は，図5.2.1に示す通り，断片的であいまいな要求を基にして，これを具体的な仕様にまとめあげていくことであり，要求の分かっている部分を具体化し仕様として取りまとめていく作業と，それを基にして他の部分の要求を明確にするという作業を交互に繰り返す．プロトタイピングシステムでは，このような段階的な仕様作成作業のプロセスを反映した検証の支援を可能とすることが重要である．

実際，このような未完成の要求仕様をプロトタイピングしながら段階的な動作確認と仕様作成を実施することによって，一度に仕様全体を作成する方法に比較して，誤りの修正が容易になるうえ，要求そのものの明確化や具体化も効率的になる．

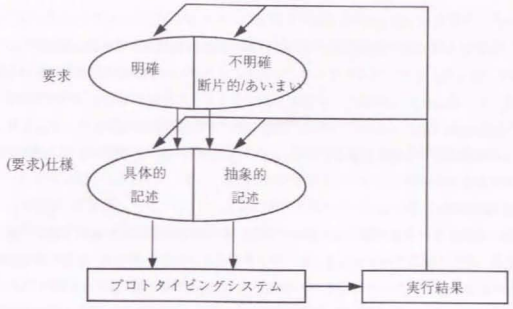


図5.2.1　要求仕様の段階的詳細化

(b)プロトタイピングに必要なマニュアル作業が少ないこと

プロトタイッププログラムの実行には様々な作業が必要になるが，そのような作業を可能な限り自動化してプロトタイピングに必要となるマニュアル作業を最小限にすることが重要である．

例えば，一般にプログラムを実行するためには多量のデータを準備する必要があり，プロトタイピングではこのような実行を多数回繰り返すことになる．いったん実行した後途中から別の部分を実行するバックトラックも少なくない．従って，プロトタイピングシステムでは，バックトラックを含むプロトタイッププログラムの実行に必要となる作業が少なく実行が容易にできることが重要である．このためには，実行に必要なデータをできる限り自動的にセットできることが望ましい．特に，いったん使用したデータは最大限再利用することとし，マニュアルでセットするデータの量を最小限にする必要がある．更に，プロトタイピングの主目的は仕様の検証にあるので，プロトタイピング実行の結果はプロトタイッププログラムでなく，仕様のレベルで評価できる必要がある．

以上の要求条件をすべて満たすプロトタイピング手法を開拓するためには，その効果を具体的に評価し明らかにすることが必要不可欠であり，それにはプロトタイピングシステムの開発とそれを用いた実験評価が重要である．そこで，本論文では，上述の要求条件を満足する手法を検討し，その検討結果を基に，プロトタイピングシステムを試作・評価する．

(2) 新しいプロトタイピングの手法とプロトタイピングシステムの設計方針

プロトタイピング手法を検討するうえで，仕様の検証対象範囲と仕様記述言語を以下の(a)と(b)の通り決定した．また，プロトタイププログラムを迅速にかつ安価に実現するという基本要求条件を満たすため，(c)に示す通り，仕様からプログラムへの自動変換によってプロトタイププログラムを自動生成することとした．更に，前項で述べた仕様の段階的詳細化の支援及びプロトタイピングの容易化という条件を満たすため，(d)～(g)に示す抽象処理等の新しい技術を考案し採用することとした[87]～[90]．

(a)機能仕様の検証

一般に通信ソフトウェアに対する要求仕様は，その静的側面を表す機能仕様と，動的側面を表す性能・特性仕様とに分けられる．本プロトタイピングシステムでは，前者の機能仕様を対象とした検証を目的とすることにした．これは，一般に性能や特性は，実行に利用するハードウェアやオペレーティングシステム等の環境条件やプログラムの記述言語やコンパイラの最適化手法等のプログラム生成条件・実行条件等，必ずしも仕様として規定されない要素に大きく依存するからである．

(b)SDL の採用

通信ソフトウェア仕様の記述言語には，2．1でも述べた通り，ITU-T が通信システムの仕様や動作を記述するために国際勧告として制定した国際標準言語 SDL(Specification and Description Language)を採用した．その理由は，SDL が，実際の通信ソフトウェアの構造と整合性のよい有限状態機械モデルをベースにしたもので，通信ソフトウェアの機能仕様を簡潔に表現できるからである．

(c)要求仕様からプログラムへの自動変換

SDL で記述された仕様は，そのモデルである有限状態機械に基づいてプログラムを実現することができる．そこで，このような原理によって仕様から C 言語で記述したプログラムを自動的に生成することにした．これによって，仕様と完全に合致したプロトタイププログラムを迅速に生成することが可能となる．

具体的には，SDL で記述された仕様における各ステートメントの意味や機能が基本的には互いに独立であることを利用し，SDL のステートメントの種別に対応した C 言語記述の小さなプログラムを予め準備しておき，仕様からプログラムへの変換に際してはこのような小プログラムに順次変換するという方法によって，これを実現することとした．この変換法は必ずしも実行効率のよいプログラムを生成することはできないが，極めて単純であり，しかも SDL の変更や修正に対して容易に対処できるという特長がある．

(d)抽象処理の採用

段階的な仕様の作成に整合した意味検証を実現するため，本論文では抽象処理と呼ぶ新しいプロトタイピング手法を考案し，採用することとした．抽象処理の原理は，まず抜けのある不完全な仕様の記述を許すことにある．これを抽象記述と呼ぶ．次に，抽象記述された仕様の中で利

用できる情報を最大限活用してプロトタイププログラムを実行し，正しい結果を最大限出力する．このような実行を抽象実行と呼ぶ．抽象記述と抽象実行の詳細は次項5．2．3で述べるが，両者を合わせて抽象処理と呼ぶ．

(e)動的な実行制御の実現

プロトタイププログラムの実行結果の検査を容易にするためには，実行の途中結果に応じて，以降の実行を自由に制御できることが重要である．そこで，変数や信号の値の設定や変更を実行途中でも可能にすることにした．更に，実行の途中結果を，実行シーケンスが分岐する各ステートメントにおいて蓄積しておくこととし，この情報を再利用することによって，実行途中で任意の分岐ステートメントの状態に戻ってバックトラックし，別の経路を実行できるようにすることとした．

(f)操作性の向上

操作性の向上には，煩雑な操作の自動化を徹底することが望ましい．そこで，以下に示す通り，実行経路の指定に関する操作の自動化を採用し，プロトタイピングに伴う操作の省力化を図った．

プログラムの実行では，通常それに必要なデータを事前に準備するが，通常は使用するか否か不明なため必ずしも使用するとは限らないデータをも準備する必要があるうえ，必ずしも検証対象となる目的の動作を実行できるとは限らない．そこで，検証対象部分を明確にかつ無駄なく指定するため，仕様のうえで実行経路を直接指定する方法を考案し，採用することとした．

しかし，すべてのプロセスについて実行経路を指定すると，矛盾が生じて操作が返って煩雑になることがある．そこで，一つのプロセスのみの実行経路をマニュアル指定することとし，他のプロセスの実行経路は可能な範囲で自動的に決定することとした．実行経路が一意に決まらない場合は，任意の経路をマニュアルで選択できることとした．また，実行経路を指定することによって値が一意に定まる変数については，これを自動的に設定することとした．例えば，実行経路の分岐点（ステートメント）では，可能であれば，実際に指定された経路を実行できるように，分岐点で参照する変数の値を自動設定する．

(g)体系的な図式出力表示

仕様作成者に分かりやすく実行結果を表示することは，プロトタイピング作業の効率向上に必要不可欠である．そこで，以下に示す通り，実行結果を仕様のレベルでかつ任意の時点で体系的に整理して出力できることとした．

(g1)プロセス単体の動作の結果

各プロセス仕様において，実行されたステートメントをその順序に従って動的に表示することとした．また元の仕様上で各ステートメントが実行された回数を表示することとした．更に，プロセス内で使われた変数と信号の送受信の履歴，及びそれらの実行時刻を記録し出力することとした．

(g2)プロセス相互にわたる動作の結果

プロセス相互間の動作の関係は，信号の送受信の履歴で表現できる．そこで，信号の送受の関係をシーケンスチャートとして表示することとした．

5.2.3 仕様の抽象処理

(1) 仕様の抽象記述

一般に，仕様は機能を単位として記述していく．SDLでは一つのまとまった機能は構文上ブロック(BLOCK)，プロセス(PROCESS)又は手続き(PROCEDURE)によって表現できるので，本プロトタイピング法では，これらを単位として分かっている部分のみを記述することを許容することにした．更に，一つの記述単位のなかでも明確に分かっているステートメントとそうでないものがあり得る．そこで，分かっているステートメントのみを記述し，必ずしも明確でないステートメントは記述しなくてもよいこととした．このような不完全な記述を抽象記述と呼ぶ．

(2) 仕様の抽象実行

抽象記述された仕様は明らかに抜けがあるため不完全であり，このような仕様を完全に実行することは不可能である．たとえ実行できたとしてもその結果の正しさについては保証できない．そこで，本論文では，以下の方法により，この問題を解決して仕様を実行することとした．

(a)文法誤りの訂正

仕様を実行するためには，まずそれが一義的に解釈できる必要があり，SDLの文法を満たす必要がある．そこで，仕様からプログラムに変換する前に仕様を検査し，SDLの文法に対する誤りをすべて検出し，原則としてマニュアルで訂正することとした．ただし，可能であれば自動的に修正することとした．例えば，チャネル(CHANNEL)，シグナルルート(SIGNALROUTE)等の宣言文は，プロセス内部の記述から自動生成することとした．また，誤った宣言文は同様な方法により自動修正することとした．

(b)抽象記述に基づく仕様の強制実行

抽象記述された仕様において記述の無い部分を参照している場合は，サブルーチンの不足したプログラムをコンパイルした後のリンケージ誤りと同様，そのままでは実行できない．しかし，本プロトタイピングシステムでは，このような誤りがあっても，以下に示す特別な処理を施すことによって強制的に実行することとした．

(b1)不足部分を参照するステートメントの処理

記述の無いプロセスや手続きを参照する信号送受信や手続き呼び出し等のステートメントは実行しないでスキップする．しかし，スキップした事実を記録しておき，プロトタイピング終了後結果の一部として出力する．これによって，プロセスや手続き等が不足していることを確認でき，不足したまま所望の動作を実行することが可能となる．

(b2)正確な出力の最大限出力

一般に，抽象記述された仕様部分又はそのような部分を参照して実行した結果は正確であると保証できない．そこで，各ステートメントで参照する変数や信号がすべて正しいと保証できる場合にのみ実行結果も正しいと判定し，このような場合にのみステートメントを実行する．この判定処理を実現するため，すべての変数と信号に対して，その値が与えられているか否か，及び，内容が正しいと保証できるか否かを表す2個のフラグを設けて管理することとした．

以上の抽象実行によって，抽象記述された仕様から生成されるプロトタイピングプログラムを実行することを可能とし，その各実行結果が正しいか否か明示することを可能とした．

5.2.4 プロトタイピングシステムの構成と実現方法

(1) プロトタイピングシステムの構成

本論文の研究で開発したプロトタイピングシステムの機能ブロック構成を図5.2.2に示す．ここで，プリプロセッサは，与えられた要求仕様の構文検査を行い，チャネル，シグナルルート等宣言文の不足部分の生成及び修正を自動的に行う．また，SDL/GRにより図式表示された要求仕様のうえでの実行経路の指定や，実行のいったん停止等の実行制御コマンドの挿入等の機能も併せ持つ．

トランスレータは，SDLで記述された仕様をC言語で記述されたプロトタイピングプログラムに変換して計算機上で実行可能にする．このプロトタイピングプログラムはプロトタイピングコントローラの下で実行される．このコントローラはプロトタイピングプログラムを前項5.2.2で述べた方法に基づき抽象実行する．そして，プロトタイピング実行時の変数や信号の履歴の収集，実行途中での信号や変数の値の代入・変更の機能を持つ．

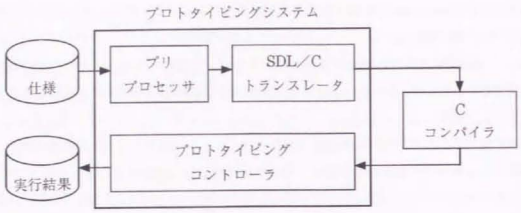


図5.2.2　プロトタイピングシステムの構成

（2）プロトタイピングシステムの実現法

本プロトタイピングシステムでは，汎用オペレーティングシステムUNIXを持つワークステーションを開発マシン並びに実行マシンとして利用した．そして，以下に示す通り，UNIXの機能を積極的に活用することによって効率よく実現した．

(a)UNIX上での仕様の実行

SDLには仕様の構成単位として，システム，ブロック，プロセスの3個の大きな枠組みがある．しかし，システムとブロックは，信号・変数のスコープのみを規定するものであり，実行単位であるプロセスをベースにしてプログラムを生成する．このようなプロセスは，UNIXの“プロセス”に1対1に対応させ，SDLの定義に従って非同期的に実行する．

(b)プロセス間通信の管理

SDLにおけるプロセス間での信号の授受は，UNIXが持つソケットと呼ばれるプロセス間通信機能のうち，バーチャルサーキット機能を利用して実現した．従って，プロセスを生成した時点で，通信するプロセス間に仮想的な通信経路を設定し，以降この通信経路を利用して信号を送受する．また，これら信号の送受の管理及びこれに基づく並列実行の管理は，プロトタイピングコントローラが行っている．

(c)バックトラック機能

本プロトタイピングシステムでは，仕様を実行する前にバックトラックポイントを設けておき，プロトタイピング実行途中でそれ以前の任意のバックトラックポイントに戻って別の経路で再開実行するという機能がある．この機能は，UNIXの子プロセス生成機能を利用して効率よく実現した．具体的には，動的実行制御のためのバックトラック機能を実現するため，実行系列が分岐するステートメント（バックトラックポイント）に到ったとき，UNIX機能を利用してUNIXの“子プロセス”を生成してそれまで実行していたUNIXの“(親)プロセス”の動作をいったん

停止し，指定された実行経路をこの“子プロセス”によって実行していく．このバックトラックポイントまで戻る場合には，それまで実行していた“子プロセス”を消滅するとともに，いったん停止していた“親プロセス”を再起動して別の“子プロセス”を生成し，それ以降の新たな経路を実行する．

5.2.5　抽象処理を用いたプロトタイピング法の評価

SDL/PRで記述された不完全な仕様の一例を図5.2.3に示す．図5.2.4は，図5.2.3の仕様におけるプロセス間の信号送受信関係を示す．この仕様例は1個のブロックから構成され，そのブロックには3個のプロセスがある．この仕様には，以下の3点の問題がある．

① プロセス間の信号経路であるシグナルルートの定義が抜けている．

② 10行目に，プロセスP1が信号sig4(i)を受信する動作が記述されているが，この信号を送信するプロセスundefの記述がない．

③ 24行目に，プロセスP2が信号sig2(i)をプロセスP1に送信する記述があるが，プロセスP1にはその信号を受信する記述がない．

このように抽象記述された仕様を本プロトタイピングツールで実行した場合，以下の通りの検証結果が得られる．

まず，プリプロセッサの処理によって，図5.2.5の4～7行目に示す通り，シグナルルート宣言文が自動的に生成された．次に，この要求仕様からトランスレータによって得られるプログラムを実行する．しかし，図5.2.5の仕様の14行目（図5.2.3の仕様の10行目）でいったん実行を停止し，信号sig4(i)が持つパラメータiへの値のマニュアル入力を促す．ツール利用者がiの値をマニュアルで入力すれば，その値をiに設定して実行を継続する．iの値を入力しない場合には，パラメータiの値は「不確か」となり，以降iを参照して処理する結果すべてが「不確か」となる．しかし，実行の途中でiに値が設定されると，それ以降は「確か」となる．図5.2.6は変数iの値をマニュアルでは与えなかったが，実行途中の19行目（図5.2.5）で値がプログラムの処理で自動的に設定される場合の変数iに関する処理結果の履歴を示す．また，図5.2.7は実行結果のシーケンスチャート例である．例えば，信号sig4の送出プロセスundefは未定義であり，プロセスP1での受信処理をスキップしたこと，プロセスP2は信号sig2を送出したが相手プロセスP1はその受信が未定義であること等を表す．

この例から分かるように，不完全な仕様に対しても，本プロトタイピングツールによれば不完全仕様に含まれる情報を最大限利用してプロトタイププログラムを実行して，正しい実行結果を最大限出力することが可能であり，これによって段階的な仕様の検証・作成が容易となる．

5.2.6　まとめ

本節では，通信ソフトウェア仕様の段階的な作成を効率よく支援する意味検証技術として，抽象処理を特徴とする新しいプロトタイピング手法を提示し具体的に論じた．

本プロトタイピング法の特徴をまとめると以下の通りとなる．

①仕様の抽象記述：プロセスや手続き(関数)等を記述単位として，要求の具体化程度に応じて部分的な記述を許容する．

②仕様の抽象実行：①で述べた記述の不完全さに応じて，仕様に含まれる情報を最大限に利用して実行し，確かな実行結果と不確かな実行結果を明確に区別して出力する．

③実行データの自動設定：実行する仕様部分は，仕様要素の実行系列で指定することとし，分岐がある場合には，その分岐条件を満たすよう変数の値等をできる限り自動的に決定する．また，一つのプロセスの実行経路の決定によって，他のプロセスの実行経路も可能な範囲で自動的に決定する．

④実行データの動的設定：実行経路の選択や変数･信号の値の設定は，仕様の実行途中でも可能である．

⑤動的な実行制御：実行の途中結果を実行系列の分岐点ですべて保存しており，これを利用することによって実行途中で任意の分岐点にバックトラックし，そこから実行を再開することが可能である．

⑥仕様レベルの実行結果の表示：実行結果として，プロセス単体ごとの実行ステートメント系列，変数の値の履歴，信号送受信の履歴，これらの実行時刻等を表示出力するとともに，プロセス相互の間の動作の表示をシーケンスチャートで表示出力する．これらの出力は，プログラムレベルではなく仕様のレベルで行い，仕様上の変数やステートメント番号を表示する．

このようなプロトタイピング技術によって，次に示す通り，通信ソフトウェア仕様の効率的な検証が可能となった．

(1)仕様の段階的な作成に対応して，不完全な仕様の記述と検証を可能にした．そして，このような不完全な仕様に含まれる情報を利用して最大限正確な検証を可能にした．

(2)多数回繰り返すプロトタイピング処理の作業は一般に煩雑になるが，その最大の要因の一つは，実行に必要な諸データの設定にある．しかし，本手法では，実行経路の指定とそれに伴う変数や関連部分の実行系列の自動決定によって，このような作業量を削減可能とした．

(3)実行途中での変数等の値の設定を可能とすることによって，実行結果に応じたダイナミックな検証が容易になる．同様に，プロトタイピングの都度最初から実行するのでなく，必要なところからバックトラックして再実行が可能なため，無駄な処理や手順を省くことができる．

以上から，本プロトタイピング法によって，不完全な仕様の検証を容易にし，要求自体の明確化並びにそれを正しく反映した仕様の作成の効率を向上させることが可能となった．


```
1   SYSTEM test_system;
2     SIGNAL   sig1, sig2(INTEGER), sig3, sig4(INTEGER);
3     BLOCK block1;
4       PROCESS P1;
5   ^^hv;                                  /* control statement for prototyping*/
6         DCL i INTEGER;
7         START;
8           NEXTSTATE state1;
9         STATE state1;
10          INPUT sig4(i) FROM undef;
11            OUTPUT sig1 TO P2;
12            TASK i:=i+1;
13            NEXTSTATE state2;
14        STATE state2;
15          INPUT sig2(i) FROM P3;
16            TASK i:=i+1;
13            STOP;
18      ENDPROCESS P1;
19      PROCESS P2;
20  ^^hv;                                  /* control statement for prototyping*/
21        DCL i INTEGER;
22        START;
23          TASK i:=1;
24          OUTPUT sig2(i) TO P1;
25          NEXTSTATE state1;
26        STATE state1;
27          INPUT sig1 FROM P1;
28            OUTPUT sig3 TO P3;
29            STOP;
30      ENDPROCESS P2;
31      PROCESS P3;
32  ^^hv;                                  /* control statement for prototyping*/
33        DCL i INTEGER;
34        START;
35          TASK i:=1;
36          NEXTSTATE state1;
37        STATE state1;
38          INPUT sig3 FROM P2;
39            OUTPUT sig2(i) TO P1;
40            STOP;
41      ENDPROCESS P3;
42    ENDBLOCK block1;
43    ENDSYSTEM test_system;
```

図5．2．3　SDL/PR記述による不完全な仕様の例

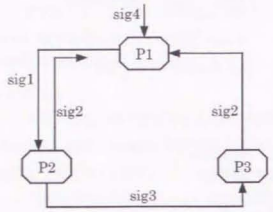


図5.2.4　図5.2.3の仕様のプロセス間相互関係図

```
 1 SYSTEM test_system;
 2   SIGNAL   sig1, sig2(INTEGER), sig3, sig4(INTEGER);
 3   BLOCK block1;
 4      SIGNALROUTE sgr0 FROM P1 TO P2 WITH sig1; /* automatically generated */
 5      SIGNALROUTE sgr1 FROM P2 TO P1 WITH sig2; /* automatically generated */
 6      SIGNALROUTE sgr2 FROM P2 TO P2 WITH sig3; /* automatically generated */
 7      SIGNALROUTE sgr3 FROM P3 TO P1 WITH sig2; /* automatically generated */
 8      PROCESS P1;
 9 ~~hv;                                  /* control statement for prototyping*/
10         DCL i INTEGER;
11         START;
12            NEXTSTATE state1;
13         STATE state1;
14            INPUT sig4(i) FROM undef;
15               OUTPUT sig1 TO P2;
16               TASK i:=i+1;
17               NEXTSTATE state2;
18         STATE state2;
19            INPUT sig2(i) FROM P3;
20               TASK i:=i+1;
21               STOP;
22      ENDPROCESS P1;
23      PROCESS P2;
```

図5.2.5　図5.2.3の仕様に対する宣言文の自動生成の結果


```
VARIABLE NAME  :i(int)          /*変数の履歴*/
  Time(sec.)  Line No.  Value
    1.080       14        0  *Undefined and Doubtful*
    1.136       16        1  *Doubtful*
    2.196       19        1
    2.204       20        2
```

図5.2.6　図5.2.3の仕様における変数iの処理結果の履歴

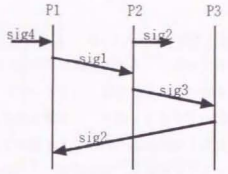


図5.2.7　図5.2.3の仕様における信号送受信の実行結果例
(sig4の送信処理と，sig2の受信処理は定義されていない)

5.3　通信ソフトウェア仕様の意味検証のための簡明化処理

5.3.1　概要

一般に通信ソフトウェア仕様の意味検証を完全に機械的に実施することは不可能であり，少なくともある程度は人手によらざるを得ない．人手による検証では，仕様が持つ意味や機能を理解することが必須である．しかし，通信ソフトウェア仕様は規模が大きく複雑なことが多く，作成者自身でも理解が困難になる場合が少なくない．従って，人手による意味検証の効率を向上させる一つの鍵は，与えられた仕様を，意味や機能は同一のまま変更しないで，より簡明な表現に変換して理解性を高めることにあると考えられる．

そこで，本節では，与えられた通信ソフトウェア仕様の表現をより簡明なものに機械的に変換する方法をいくつか提案し論じる．これらの方法によって，もとの仕様に比較して，機能的には全く同一であるが例えば含まれる要素数が少ない表現が得られる．従って，目視による仕様の検査が容易となるため，意味検証の効率を向上できる．

5.3.2 通信ソフトウェア仕様の簡明化処理の原理

仕様の簡明性や理解性についての基準や尺度は必ずしも明確になっていない．本論文では，仕様を理解するために必要となる作業量を少なくするという観点から，次の（1）～（5）に示す通り，仕様簡明化の原理を整理した[93]~[97]．但し，一般に仕様簡明化技術は，仕様を記述する言語に依存することがあるので，本論文では，2．で述べた通り，ITU-T 勧告の通信システム記述言語 SDL を採用することとした．

（1）形式簡明化

仕様は，仕様記述言語の構文規則（文法）に則って記述される．SDL は形式的言語であるが，仕様表現の自由度は高く，構文が同一でもその表現法は配置(レイアウト)や大きさに関して一般に複数通り存在する．具体的には，改行や空白（含インデンテーション）を自由に挿入してもあるいは仕様要素(シンボル)の大きさを変更しても構文的には同一であることが多い．実際には，複数通りの表現の中から仕様作成者の判断により選択されることになるが，その選択は作成者の好みや趣味に依存することがある．その結果，一つの仕様を多人数で作成する場合には，仕様全体としての一貫性が損なわれ，理解性を低下させてしまうことがある．従って，同一の構文を持つ仕様表現における配置や大きさ等の形式を可能な範囲で統一し標準化することが理解性向上のための一簡明化技術と考えられる．このような技術を形式簡明化と呼ぶ．

（2）意味簡明化

一般に同一の意味（機能・性能等）を持つ仕様の表現の構文は唯一とは限らない．このような場合には，理解性に優れた簡明な構文を選択するべきである．

SDL は手続き型言語であり，手続き型の仕様記述言語で記述された仕様を理解するためには，仕様要素の実行フローを正しく理解することが必要不可欠である．従って，仕様表現の簡明さの判断基準としては，まず，①実行フロー制御構造（仕様構造）の単純さが挙げられる．同一仕様構造を持つ仕様の中では，その表現が簡潔であるほど理解し易いと考えられることから，仕様表現の簡明さの判断基準として，更に，②冗長な仕様要素がないこと，及び，③仕様要素の個数が最小であること，が挙げられる．これらの判断基準に照らして仕様表現の簡明化を図る手法は，仕様の意味を考慮して仕様表現を処理するので，意味簡明化と呼ぶ．以下の（3）～（5）で，これら意味簡明化を具体的に論じる．

（3）仕様構造の単純化

手続き型言語における記述構造の単純化に関する検討は，これまでプログラミング言語の分野で多く行われてきた．その代表的な手法の一つは，実行フロー構造の複雑化を招く go to 機能の使用の制限である．SDL にも go to 機能（JOIN と LABEL）が含まれているため，go to 機能の使用制限は仕様構造簡明化の一手法である．図5．3．1に，その一例を示す．

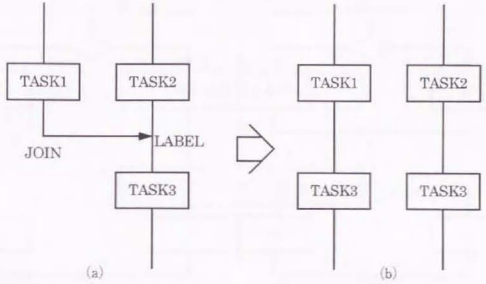


図5．3．1　go to 機能の削除

実行のフローを制御する仕様要素としては，go to 要素以外に，分岐要素（SDL の DECISION）がある．go to 要素と分岐要素の組合せの仕方によっては，実際には実行し得ない実行フローをも表現してしまい誤解を招く恐れがある．例えば，図5．3．2の仕様の例において，a，b1，b2，c1，c2，d は，それぞれ連続して実行する仕様要素から構成される仕様部分を表しており，同図（a）は，仕様部分 a を実行した後，b1 又は b2 を実行し，その後 c1 又は c2 を実行し，そして d を実行することを表す．即ち，

①a-b1-c1-d
②a-b1-c2-d
③a-b2-c1-d
④a-b2-c2-d

の4個の実行フローが存在することを表す．しかし，b1 を実行した後は c1 だけ実行し，b2 を実行した後は c2 だけ実行するものと仮定すると，同図（a）は②と③の余分な実行フローを表現することになり，仕様の誤解釈を招くことがある．このような場合には，余分で不要な実行フローを完全に削除した表現に変換することによってこの問題を解決することができる．この図の例では，同図（a）を同図（b）に変換すればよい．このような変換処理は，分岐要素の分岐条件を検査して実行不可能な実行フローを抽出し削除することによって実現できる．このように余分な実行フローを削除する方法は仕様構造簡明化の一手法であり，実行フロー最小表現化と呼ぶ．

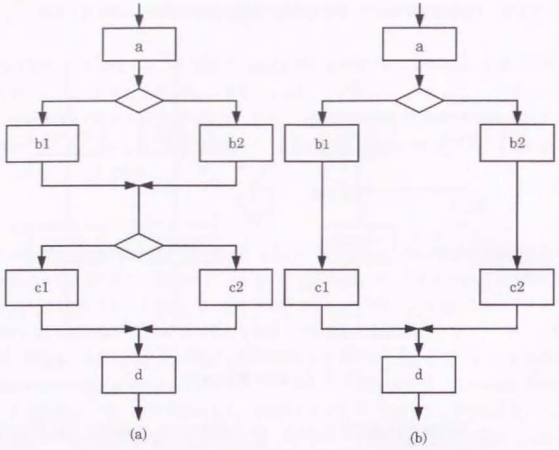


図5.3.2　仕様における実行フローの最小化の例

（4）冗長要素の削除

冗長要素の削除では，冗長な仕様要素の明確化・具体化が鍵となる．冗長な仕様要素には，まず，絶対に実行されることがない実行不可能要素がある．次に，実行可能な仕様要素のうち，その前後に実行する仕様要素との関係によって，実行しても無意味となるような無効要素がある．例えば，図5.3.3に示すように，ある変数に値を設定した後，その値を使用（参照）する前に別の値を設定する実行フローがあった場合，最初の値設定処理を記述した仕様要素は無効である．

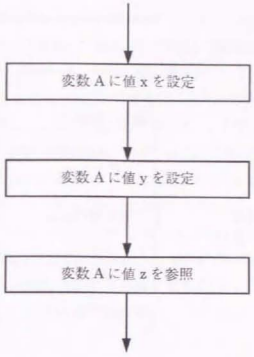


図5.3.3　無効な仕様要素の例

（5）要素数の最小化

要素数の最小化には，複数個の同値な仕様要素を一つの要素にまとめる同値仕様要素の一括化と仕様を要素数が少ない別の表現に変換する表現形式の単純化とがある．

同値仕様要素一括化には，形式的にも意味的にも同一な仕様部分を一つの手続き又はマクロにまとめて置き換える手続き化／マクロ化，及び，有限状態機械モデルに特有な処理として，形式的には異なるが意味的には同値である複数個の状態を一つにまとめる状態一括化とがある[51]．

表現形式単純化の対象となる仕様要素は，処理の場合分け機能を持つ仕様要素，即ち，分岐要素である．一般に，複数種類の分岐要素が連続して実行される場合，それらの実行順序を変更することによって，仕様要素の総数を減らし，理解し易い仕様表現に変換できる場合があり，このような変換処理が表現形式単純化である．

以上で述べた各種の仕様簡明化処理技術を図5.3.4にまとめて示す．

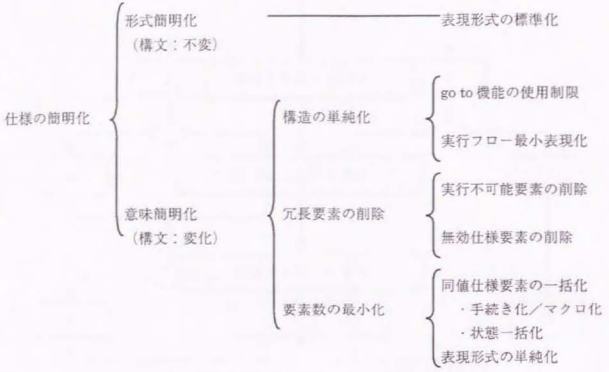


図５．３．４　仕様の簡明化技術の分類

５．３．３　通信ソフトウェア仕様の簡明化処理の実現

前項で述べた原理に基づく通信ソフトウェア仕様簡明化を支援するツールを実装し，それらを組み合わせた仕様簡明化支援システムを試作した．本試作の設計においては，以下の基本方針を採用した．

①仕様簡明化処理は計算機で行うが，仕様作成者との対話によって有効性を確認した後実行する．

②仕様簡明化支援システムは，単一の簡明化機能を持つ多数の基本ツールから構成し，仕様簡明化支援システムの運用に際しては，これらツールを必要に応じて自由に組み合せて使用できるようにする．

試作した仕様簡明化支援システムが持つ機能を，図５．３．４に基づき，以下に具体的に論じる．

（１）仕様の形式簡明化（表現形式標準化）

仕様処理のためのマンマシンインタフェースには図式表現であるSDL/GR形式が有効であるが，記述量が少ない仕様や，過去に使用したことがある仕様を部分的に修正して再利用する場合には，UNIXのvi等のテキストエディタを使ってSDL/PR形式の仕様を直接作成又は修正することも考えられる．そこで，このような場合の支援機能として，①SDL/PR清書機能と，②マクロ展開機能を開発した．なお，SDL/GRでの表現形式標準化は，SDL/PR清書機能とPR→GR変換機能を組み合せ

て使用することによって実現できるので，不要とした．

①SDL/PR清書機能は，エディタを使って自由形式で作成したSDL/PR記述の仕様の各ステートメントを１行に組み立て直し，インデンテーション等の処理を行って，SDL/PR記述の仕様を統一し見やすくする．また，SDL/GRによる図式表現の場合は，個々の仕様要素に関し特定の座標上のみの配置と大きさに制限した表現に変換して理解性を向上させる．図５．３．５にSDL/PR清書機能の適用例を示す．この機能は，UNIXの字句解析プログラムlexを用いて，SDL/PR文字列中のSDL予約語とセミコロン；で囲まれた１行文のステートメントを切り出した後（ただし，LABELには特別な処理を施す），空白・改行・タブ等の余分な文字コードを除去し，インデンテーション等を付加する処理によって実現している．インデンテーションは，ENDxxを伴う予約語（例：SYSTEM, BLOCK, PROCESS, DECISION等）が出現する毎に字下げを行い，ENDxxが出現するごとにその逆の処理を行うことによって実現している．更に，INPUTも同様に字下げを行い，同一のSTATEの下に連なるINPUTの先頭位置を揃える．

```
SYSTEM AAA; BLOCK CALL_HANDLING;
PROCESS LOCAL_CALL; STATE IDLE;
INPUT A_OFF_HOOK; DECISION IN_SERVICE;
(NO):OUTPUT OUT_OF_SERVICE TO MAINT;
NEXTSTATE OFF_HOOK_OUT_OF_SERVICE;
(YES):DECISION 'BLOCKING';
(YES): NEXTSTATE OFF_HOOK_IN_SERVICE;
(NO): TASK 'CONNECT DIGIT RECEIVER';
OUTPUT SEND_DIAL_TONE TO SSEND;
OUTPUT START_TO TO TIMER;
NEXTSTATE AWAIT_FIRST_DIGIT;
ENDDECISION; ENDDECIDION;
```

(a)元の SDL/PR 表現

```
SYSTEM AAA;
  BLOCK CALL_HANDLING;
    PROCESS LOCAL_CALL;
    STATE IDLE;
      INPUT A_OFF_HOOK;
        DECISION IN_SERVICE;
        (NO):
          OUTPUT OUT_OF_SERVICE TO MAINT;
          NEXTSTATE OFF_HOOK_OUT_OF_SERVICE;
        (YES):
          DECISION 'BLOCKING';
            (YES):
              NEXTSTATE OFF_HOOK_IN_SERVICE;
            (NO):
              TASK 'CONNECT DIGIT RECEIVER';
              OUTPUT SEND_DIAL_TONE TO SSEND;
              OUTPUT START_TO TO TIMER;
              NEXTSTATE AWAIT_FIRST_DIGIT;
            ENDDECISION;
          ENDDECIDION;
```

(b)清書された SDL/PR 表現

図5.3.5　SDL/PR 表現の清書の例

次に，②マクロ展開機能は，MACRO EXPANSION として他所で定義されている一連のステートメントを本文中に取り込む機能で，見かけ上の制御の流れを単純化して仕様を分かり易くするとともに，マクロの切り出し方を見直すためにも有効である．

（2）仕様構造の単純化

(i)go to 機能の使用制限

図5.3.1に示すように，SDL/PR 上において，二つの実行系列が JOIN と LABEL で結合されている場合，それらに共通の処理部分（TASK3）を双方の実行系列に持たせることによって JOIN を削除する．このような処理によって記述量が増えるので，一見好ましくないようにも見えるが，実行の流れが単純になるので理解性が向上するとともに実際のプログラムの制御構造との整合性も良くなるという利点がある．実際，図5.3.1（a）の形式で要求仕様を作成しても（b）の形式でプログラムを作成することを好むプログラマがある．ただし，図5.3.6のようなループが存在する場合は，展開が容易でないので，そのまま展開しないでおく．

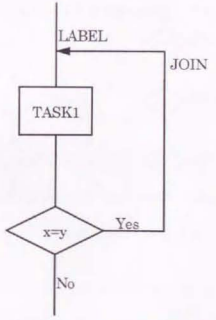


図5.3.6　削除しない go to 機能の例

(ii)実行フロー最小表現化

一つのプロセス内において，DECISION 変数から判断して実行不可能な部分系列を削除する．ここで，部分系列とは，初期ステートメントからすべてのステートメントについて実行のフローに従って辿っていったときに得られるステートメント系列（実行系列）を，実行フローの合流点と分岐点で区切ったものである．

図5.3.7に DECISION 文（ステートメント）だけを抽出して例示する．ただし，処理の複雑化

を回避するため，DECISION(変数　演算子　定数)の形の文のみをこの簡明化処理の対象とした．ここで，演算子としては，等号(=)，不等号(≠)，大小関係(>，<，≧，≦)を対象とした．また，複数の論理式をAND又はORで連結することも可能とした．この簡明化処理の手順は次の通りである．

①単なる文字列の集合として蓄積したプロセスごとのSDL/PRファイルを，初期状態から最終状態までの実行順序に従って各ステートメントの論理を追えるようなチェイン構造に変換する．

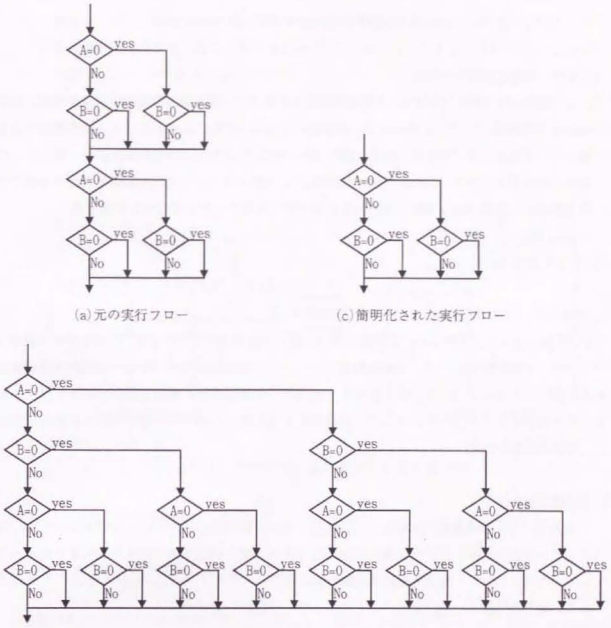
②各プロセス仕様に含まれる分岐要素DECISIONで使われている変数をすべて抽出し，X=(x1, x2, …, xn)とする．

③ある状態から次の状態までの部分を単位として，次の④以降を実施する．

④分岐要素を含む実行系列が他の実行系列と合流する場合，合流をなくし別の系列となるよう展開する（図5．3．7の(a)→(b)）

⑤DECISION変数XとLABELに注目して，④で得られたすべての実行系列を初期状態から辿り，DECISION変数のとる値の履歴に矛盾が生じて分岐条件を満たせない場合には，それ以降の部分系列を実行不可能と判断する．

⑥実行不可能な部分系列以降を仕様作成者に提示し，確認を得た後削除して，後述する同値状態一括化処理と同様な方法によって，完全に同値なステートメント(文)を一括化する．（図5．3．7の(b)→(c)）



(a)元の実行フロー

(c)簡明化された実行フロー

(b)展開された実行フロー

図5．3．7　実行不可能な実行フローの削除の例

（3）仕様冗長要素の削除

(i)無効仕様要素の削除

この機能は，5．4で述べる意味検証機能を応用することによって実現する．この意味検証機能は，仕様を記述するうえで守らねばならないPR文の実行順序関係を規則(ルール)として予め登録しておき，その規則に違反したステートメント系列を仕様誤りとして出力するものである．この意味検証機能において，無効要素を判定するためのPRステートメント実行順序関係を規則

として扱うことによって，無効仕様要素を抽出することが可能となる．

(ii)実行不可能仕様要素の削除

この機能は4．で述べたプロトコル論理検証を応用して，実行不可能な遷移処理を抽出し削除するものである．プロトコル検証は，複数個のプロセスの間にわたるシステム全体の動作を模擬して，プロセス間の信号の入出力処理，即ち，SDL/PRでのINPUT・OUTPUTステートメントの実行可能性を判定する．この判定結果を利用して，INPUTステートメント以降の実行不可能な処理を削除し，実行可能な遷移系列のみを含むプロセス毎の状態遷移仕様を作成する．

（4）仕様要素数最小化

(i)手続き化・マクロ化機能

INPUTから始まり，次のSTATEに到るまでの一連の仕様要素シーケンスを，仕様要素を1個含むグループ，2個含むグループ，3個含むグループ，…に分割し，各々のグループ内の仕様要素がその出現順序を含めて同一のものを抽出して表示し，仕様作成者の確認を経て手続き又はマクロとして一括化する．オプションとして，出現順序を入れ替えれば同一となるものも自動的に検索し，同様の処理を行う．

(ii)状態数最小化

上述の（2）仕様構造の単純化の(ii)実行フローの最小化①の要領によって作成したSDL/PRファイルを基にして，プロセス毎の状態と信号入出力及び状態遷移の関係を表にまとめる．そして，この表に（不）完全定義順序機械の簡約化手法（例えば，Paul-Ungerの手法[51]）を適用して，等価な状態の一括化を行うことによって，状態数を削減する．

(iii)表現形式単純化

分岐要素(DECISION)の実行順序を並び替えてDECISIONの一括化を行い，DECISIONの個数を削減する．一般に，DECISIONには，①ループ処理の終了の判定と，②異なる処理を実行するための場合分けの2種類の用途がある．ここでは後者の②の用途のDECISIONを処理対象とする．処理手順の概要は次の通りであり，具体例を図5．3．8に示す．また，詳細は5．3．3で論じる．

① INPUTから始まって次のSTATEに到る一連のシーケンスに，場合分けのためのDECISIONが複数個連続して存在するとき，②以降の処理を行う．

② 場合分けの条件を，DECISIONステートメントに含まれる論理変数を用いた論理式で表す．（図5．3．8（a））

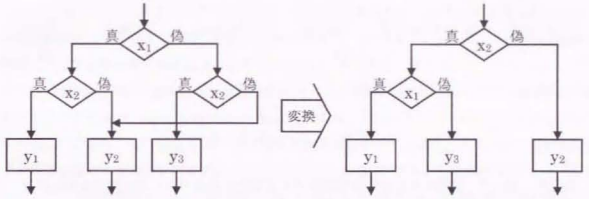
③ 多値変数に拡張したQuine-McClusky法によって，②で得た各論理式の最簡形式論理関数を

求める[112]．（図5．3．8（a）では，$y_1=x_1x_2$，$y_2=x_1x_2'+x_1'x_2'=x_2'$，$y_3=x_1'x_2$）

④ 各最簡形式論理関数に含まれる論理変数のうち，最も多くの項に含まれるものを求める．（図5．3．8（a）では，x_2）

⑤ ④で得た論理変数の真偽に応じて場合分けする（図5．3．8（a）では，$|x_1, x_1'|$ x_2, x_2'．）

⑥ ⑤で得られた各場合について，更に④の処理をくり返し実行して，次に選択すべき論理変数を可能な限り求める．（図5．3．8（b）では，x_1）



(a)変換前の仕様　　(b)変換後の仕様

図5．3．8　分岐要素の順序変更による削減の例

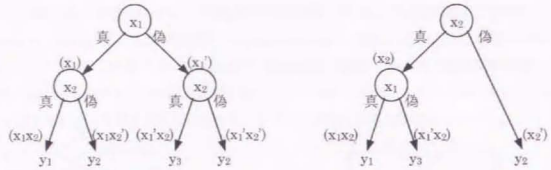
5．3．4　分岐要素の順序変更による削減

（1）　分岐要素の順序変更による仕様簡明化

通信ソフトウェア仕様を理解することが必ずしも容易でない理由の一つは，機能や動作の内容が条件によって異なる場合，そのような仕様を表す記述が複雑になりがちだからである．このような仕様を記述するのに応用される代表的な基本手法は決定木である．決定木は，出力が複数種類あるとき，それぞれに対応する条件を枝別れの連なりから構成される木によって表すもので，仕様記述への応用では，各出力が一つ一つの機能や動作に対応する．決定木の頂点と各節点は条件を判定するために必要な論理変数を持ち，一つの節点から出る複数個の枝はその節点が持つ論理変数の値に対応した条件要素を表す．つまり，決定木では頂点から各端点に到るパスによって個々の出力に対応する条件を表現している．

図5．3．8(a)，図5．3．9(a)は，プログラムの仕様記述にしばしば使われるフローチャートの例とそれに含まれる決定木をそれぞれ示す．このようにフローチャートでは，枝別れを示す菱形の分岐シンボルとそれらを結合して制御の流れを示す線分を用いて決定木が表現される．図5．3．8(a)，図5．3．9(a)において，出力は，真と偽の二つの値を持つ2個の2値論理変数x_1とx_2

の値に依存し，出力条件は4通りある．しかし，そのうち2通りについては同一の出力値 y_2 を持つので，出力は y_1，y_2，y_3 の3種類である．



(a) 図5．3．8(a)に含まれる決定木　　(b) 図5．3．8(b)に含まれる決定木

図5．3．9　決定木の例

決定木において，枝別れのための論理変数の記述順序を変更すると別の異なる木になる．つまり，枝別れのための論理変数の記述順序が不適当であると，仕様を表す決定木が複雑になり，決定木全体の理解即ち仕様の理解が困難になることがある．従って，理解し易さという観点からは論理変数の記述について最適な順序が存在し，与えられた仕様に含まれている決定木の論理変数を最適な順序となるように変更することによって仕様を分かりやすくできる．

例えば，図5．3．8(a)，図5．3．9(a)において，判定対象である論理変数 x_1 と x_2 の順序を入れ替え逆にすると，図5．3．8(b)，図5．3．9(b)がそれぞれ得られる．図5．3．8(b)，図5．3．9(b)をそれぞれ図5．3．8(a)，図5．3．9(a)に比較すると，仕様は全く同一であるが，判断分岐／節点や制御の流れを示す線分／枝が少なく，より単純で理解しやすい．この例から分かるように，決定木は頂点及び各節点に対応する条件判定に関し手続き的表現となっているので，決定木を理解するためには頂点から端点に到るパスに沿って各枝の表す条件要素を一つずつ理解していくことが必要である．従って，そのような枝が少ないほど決定木を理解することが容易になっていくものと考えられる．このような理由から，本論文では，ソフトウェア仕様の記述に応用される決定木の理解性を表す尺度として，それに含まれる枝の個数を提案する．この尺度に基づく仕様簡明化問題を，決定木を単純化する最適化問題として定義5．2．1に提示する．

[定義5．3．1]（仕様簡明化問題＝決定木の最適化問題）

与えられた決定木と同一の仕様を持ちかつ含まれる枝が最も少ない決定木を求めよ．

[定義終]

定義5．3．1で示した決定木最適化問題を直接解くには，与えられた決定木と同一の仕様を持つ決定木をすべて導出し，その中から枝が最も少ないものを選べばよい．しかし，この直接最適化法は，決定木の規模，特に，判定に使用する論理変数の種類の増大に伴って処理量が急激に増

大する．従って，直接最適化法は実用的ではなく，もっと効率の良い最適化法が必要である．

（2）決定木の特性と最適化の基本原理

一般に，決定木における各出力が選択される条件は，頂点と各節点が持つ論理変数から構成される論理関数によって表現される．与えられた決定木における条件を表すのに使われる論理変数が n 種類あるものとし，それらを $x_i(i=1,2,\cdots,n)$ とする．また，出力は m 種類あるものとして $y_j(j=1,2,\cdots,m)$ とする．出力 y_j に対応する条件を表す論理関数を f_j とする．以下では簡単化のため論理変数はすべて2値とし，論理変数の否定を'で表す．また，f_j は，例えば，Quine-McCluskey の方法によって最簡形式に簡単化され，単一の項になっているものとする．

例えば，図5．3．8(a)の決定木では，出力が y_1，y_2，y_3 となるための条件を表す最簡形式論理関数はそれぞれ $f_1=x_1x_2$，$f_2=x_1x_2'+x_1'x_2'=x_2'$，$f_3=x_1'x_2$ である．このように，一般に一つの決定木が与えられることによって論理関数 f_j がすべて与えられたことになる．逆に，与えられた論理関数からその論理変数の判定順序を決めることによって決定木を導出できる．上の例では，f_1，f_2，f_3 に含まれる二つの論理変数 x_1，x_2 の判定順序を x_1，x_2 にすると，図5．3．8(a)の決定木が得られ，また判定順序を逆の x_2，x_1 にすると図5．3．9(a)の決定木が得られる．

一般に，いくつかの最簡形式論理関数（項）のうち，論理変数 x_i またはその否定 x_i' を含むものの個数を変数の共通度と呼ぶことにする．そして，対象とする論理変数（項）すべてにそれ自身またはその否定が含まれているような論理変数を完全共通論理変数と呼ぶ．

与えられた決定木から得られる最簡形式論理関数（項）から完全共通論理変数を順次選択していくことが可能であれば，このような操作を繰り返すことによって最適な決定木が得られる．即ち次の定理が成立する．

[定理5．3．1]（決定木最適化基本原理）

与えられた決定木から得られる最簡形式論理関数（項）に完全共通論理変数が存在する場合これを選び，更に，その後も完全共通論理変数が存在して順次選ぶことを繰り返すことができて一つの決定木が得られるならば，この決定木は最適木となる．

[定理終]

この基本原理を図5．3．8(a)の決定木に適用すると以下の通り図5．3．9(a)の決定木が最適木として得られる．図5．3．8(a)の決定木の出力は y_1 から y_3 までの3通りあり，それぞれに対応する条件は，$f_1=x_1x_2$，$f_2'=x_2'$，$f_3=x_1'x_2$ であった．このとき，完全共通論理変数は x_2 のみで，x_2 を最初に選択する．この結果，条件は x_2（ケース1）と x_2'（ケース2）とに分かれる．ケース1には論理関数 f_1 と f_3 とが含まれるが，それらは選択した論理変数 x_2 を除くと，x_1，x_1' となる．ここでの完全共通論理変数は x_1 となるので，x_1 を次に選択する．その結果，ケース1は x_1x_2 と $x_1'x_2$ とに分かれるが，これらはそれぞれ f_1，f_3 に等しい．一方，ケース2は f_2 に等しい．以上の処理によって，図5．3．8(a)の決定木から図5．3．9(a)の最適木が得られる．

[定理5．3．1の証明]

一般に一つの完全共通論理変数を選択する毎に，条件判定のための論理変数を持つ節点が1個とその節点から出る枝が2本決定木に付加され，同時に条件はその変数が真の場合と偽の場合とに分かれるのでケース数が1個増える．しかも，各論理関数はその論理変数あるいはその否定を必ず含むため，これら各ケースの条件は常に与えられた m 個の論理関数（項）のいくつかの論理和に等しい．従って，このような操作を(m-1)回繰り返すことによって，与えられた m 個の論理関数と同一の条件がすべて得られ，その結果，最終的には2(m-1)個の枝を持った決定木が得られる．

一方，m 個の異なる出力を場合分けするためには，2分岐の論理判定は少なくとも(m-1)回必要であり，従って，2(m-1)個の枝が必要である．以上から，完全共通論理変数を選択し続けることによって導出される決定木は最適であり，定理5．3．1が成立する．

[証明終]

この証明から分かるように，定理5．3．1において，完全共通論理変数が複数個ある場合は，それらの選択順序は任意でよく，一般には複数個の最適木が存在する．これら最適木の枝の個数は2(m−1)であり，またその各端点は異なる出力を持つ．

（3）2値論理変数から構成される決定木の最適化

前節では，完全共通論理変数が常に存在する場合の最適化原理を示した．しかし，一般には完全共通論理変数が常に存在するとは限らない．そこで，本論文では，次に示す二つのヒューリスティックな最適化法を提案する．

①共通度の最も高い論理変数を優先して選ぶ．

②決定木をすべて導出し，それらの間で枝が最も少ないものを最適木とするが，共通度の高い論理変数を優先して選ぶ．

方法②において，決定木を網羅的に導出する場合，その途中で得られる部分木の枝数が既に得られた決定木より多いときは，それを処理し続けて決定木を導出しても決して最適とはならない．従って，分岐限定法を適用して，部分木が得られたときは，それまでに得られている決定木のうち枝数が最も少ないものと比較し，多い場合はその部分木以降の処理は中止する．このような処理の効率を向上させるためには，可能な限り早く枝数の少ない決定木を得ることが重要である．一般に決定木を導出するときに非完全共通論理変数を選ぶと，その変数およびその否定のいずれも含まない論理関数を2分することになり，その分割数に対応して決定木の枝数が出力の種類数より増えていくことになる．そこで，このような枝数の増加が少ない，即ち枝数の少ない決定木を早く得るため，共通度の高い論理変数を優先的に選び処理することとする．

以上で原理を説明したヒューリスティックな決定木最適化法①および分岐限定法を付加した②のうち，後者についての最適決定木導出アルゴリズムの概要を，論理変数の判定順序の決定部分を中心として以下に示す．ただし，出力の場合分け条件を表す論理関数(項)を要素とする集合をSとする．また，Nはその時点までに得られた決定木のうちで，枝数が最も少ないものの枝数を表す変数で，最終的には最適木の枝数を示す．更に，Tはその時点で枝が最も少ない決定木を表し，最終的には最適木を表す．

[アルゴリズム5．3．1]（最適決定木導出の近似法）

[主ルーチン]

Nに十分大きな値を設定する．与えられた決定木から出力の条件を表す論理関数(項)を導出し，それらをすべてSの要素とする．副ルーチン sub1(S)を実行する．

[主ルーチン終]

[副ルーチン sub1(S)]

Sに含まれる論理変数を共通度の高いものから順に並び替え，それらをあらためて x_1，x_2，…，x_t し，副ルーチン sub2(S, x_1)，sub2(S, x_2)，…，sub2(S, x_t)を順次実行する．

[副ルーチン終]

[副ルーチン sub2(S, z)]

zを，次に判定する論理変数として選択する．

Sに含まれる論理項のうち，zまたは z'を含む各論理項からそれぞれzまたは z'を削除することによって得られる論理項を要素とする集合をそれぞれ S_1，S_2 とする．更に，Sに含まれる論理項のうち，z·z'ともに含まない論理項をすべて S_1，S_2 の両者に対して付加する．ただし，Sがzまたは z'を要素として含む場合は，それぞれ，S_1=|1|，S_2=|1| とする．

S_1 と S_2 の要素がともに1個で，かつ，変数zの選択によって完全な決定木が生成されたならばその決定木の枝数とNとを比較し，Nより小さい場合その値をNに設定してその決定木をTとする．S_1 または S_2 の要素が複数ある場合，その時点における部分木の枝数とNとを比較し，Nより小さいときのみ副ルーチン sub1(S_1)または sub1(S_2)をそれぞれ実行する．

[副ルーチン終]

[アルゴリズム終]

これまでの説明から分かるように，アルゴリズム5．3．1によって最適な決定木を導出することが可能であり，次の定理が成立する．

[定理5．3．2]（アルゴリズム5．3．1の正当性）

アルゴリズム5．3．1は必ず停止し，その結果，最適な決定木が得られる．

[定理終]

（4） 決定木最適化法の拡張

以上で示した決定木最適化法では，論理変数がいずれも2値であるとしてきた．また，与えられた決定木から得られる最簡形式論理関数は単一の項であるとしてきた．しかし，この最適化法を，論理変数が任意の多値で，最簡形式論理関数が任意の項から構成される場合に拡張することは容易であり，以下に示す3点の変更を加えればよい．このような拡張を行った場合にも定理5．

3．1と定理5．3．2は成立する．

(1)論理変数の共通度は，その任意の値を含む論理関数の個数とする．

(2)ある変数の判定に基づく分岐後の場合分け数は，その変数のとる値の種類数に等しくする．

(3)与えられた最簡形式論理関数が複数の項を含むときは，各項を一つの論理関数とみなし処理する．

（5）決定木最適化法の評価

以上で示した最適化法を，先述した直接最適化法と比較評価した結果は次の通りである．

完全共通論理変数が常に存在する場合は，これらを順次選択することによって余分な決定木の導出を完全に回避でき，最適な決定木を極めて短時間で導出できる．

完全共通論理変数が存在しない場合には，方法①では共通度の最も高い論理変数を選ぶという単純な処理によって，必ずしも最適ではないが最適木に近い決定木を極めて短時間で導出できる．また，方法②では，分岐限定法を応用し共通度の高い変数を優先的に選ぶことによって最適な決定木を早く導出することが可能になるが，方法①よりは処理時間が大きい．そこで，論理変数が多いときは方法①を，論理変数が少ないときは方法②を採用するのが実用的であろう．

本項では，ソフトウェア仕様の理解性を向上させるための仕様最適化法を論じ，一つの仕様最適化問題を，仕様記述に広く応用されている決定木を単純化する問題として定式化するとともに，最適化のための効率よいヒューリスティックな方法を提示した．

本最適化法は決定木を応用したすべての仕様記述言語に適用可能である．このような言語には，SDL以外にも，フローチャート，PAD等，実際に使われている多くの仕様記述言語が含まれる．本最適化法によって，このような言語で記述されたソフトウェア仕様の理解性を容易に向上させることが可能で，この結果仕様検証の効率向上が図れる．

なお，以上では，決定木の最適化にあたってその論理変数の順序は任意に変更できるものと仮定したが，論理変数の持つ物理的意味からは順序変更が好ましくない場合もある．このような場合には，許される範囲内で順序を変更し最適化を実行することになる．

5．3．5　通信ソフトウェア仕様簡明化処理の評価

試作した仕様簡明化支援システムを用いて仕様簡明化の各種手法を評価した．その評価結果例を表5．3．1に示す．ここで評価対象とした仕様はメッセージ蓄積交換システムの静止画処理機能部で，端末から受信した静止画情報の同報通信処理や受信端末が閉塞の場合の代行受信等の機能を持ち，全体としてはSDL/PRで約10,000行の大きさであり，9個のプロセスから構成される．

図5．3．4の「実行不可能な実行フローの削除（実行フローの最小表現化）」と「等価状態の一括化」とをこの静止画処理機能部に適用した場合の評価結果は次の通りである．まず，静止画

処理機能部の出力制御プロセスに，「実行フローの最小表現化」を適用した．その結果，分岐要素(SDL/PRのDECISION)が33個減少し，SDL/PRでの仕様規模が1,733行から1,333行に減少した．つまり，人手で作成した仕様には冗長な分岐要素が含まれており，それを削除することによって仕様をかなり単純にできた．次に，元の静止画処理機能部の出力制御プロセスに「等価状態の一括化処理」を適用した．その結果，出力制御プロセスは2個の状態を1個の状態に簡約できる場合が3ケースあり，SDL/PR規模は1,733行から1,696行に減少した．更に，静止画処理機能部の出力制御プロセスに，「実行フローの最小表現化」と「等価状態の一括化」の両処理をこの順序で適用した場合，SDL/PR規模は1,733行が1,295行に，つまり約3/4に減少した．

別の簡明化機能の例として，「分岐要素の順序変更による削減」を，海事衛星通信システムの海岸局システムが持つテレックスサービス呼処理機能部に適用した．その結果，2ヶ所で本簡明化機能が有効に動作し，分岐要素がそれぞれ3個・4個から2個・3個に減少した．全体的にはSDLの図式表現SDL/GRでの要素数が70個から68個に減少した．

以上から分かるように，人手で作成した仕様に簡明化処理を適用することによって，仕様の規模をかなり削減することができ，これによって仕様の理解性が向上し，目視による意味検証が容易になることが確認できた．このような簡明化が適用できるということは，その簡明化結果と元の仕様部分が同一の機能であることを意味しており，この確認によってそのような意味検証が同時に達成できることにもなる．

5．3．6　まとめ

本節では，ある程度は人手による検査が必要になる意味検証の効率を向上する方法として，与えられた仕様の表現をより簡明なものに機械的に変換する手法を提案し論じた．仕様の簡明性についての基準や尺度は必ずしも明確になっていない．本節では，仕様を理解するために要する作業量を少なくするという観点から，以下の基準を設定し，これら基準に照らして簡明化を図ることとした．

①仕様記述要素の配置場所(インデンテーション)を，予め定められた整理ルールに従って特定すること（清書）．

②go toに相当する仕様要素を含まず，仕様要素の実行系列が構造化されていること．

③実行されることがない或は実行してもその結果が参照されないため冗長となる仕様要素がないこと．

④含まれる仕様要素の個数が少ないこと．

以上に示した4基準に照らし，本節では，SDLを使って人手で作成した通信ソフトウェア仕様を，より簡明で機能が同一の仕様に自動的に変換する様々な方法を具体的に示した．そして，特に含まれる仕様要素数を削減する方法について定量的な評価結果を論じ，例えば3/4程度に削減できることを示した．

このように，人手で作成した通信ソフトウェア仕様に，提案した簡明化処理を適用することに

よって，仕様の規模をかなり削減することができ，この結果，仕様の理解性が向上し，目視による意味検証が容易になる．また，このような簡明化が適用できるということはその簡明化対象部分の機能が簡明化結果と同一であることを意味しており，両者を比較して確認することによって意味検証が同時に達成できることにもなるので，本簡明化法にはそのような効果もある．

表5．3．1　仕様簡明化処理による仕様ステートメント数の低減効果例

簡明化処理 / 仕様規模(SDL/PR)	実行フローの最小表現化	等価状態の一括化	実行フローの最小表現化と等価状態の一括化
もとの仕様の規模	1,733行		
簡明化処理後の仕様の規模	1,333行	1,696行	1,295行

5．4　実行系列の検査による仕様の意味検証

5．4．1　概要

通信ソフトウェア仕様の一つの意味検証は，仕様に対する原始要求が既知の場合にそれが正しく仕様に反映され満たされていることを確認することにある．このような意味検証を機械的に実現するためには，仕様が満たすべき要求を計算機で処理できる形式で表し，そのように表現された要求を仕様が満たしているか否かを逐一検査する必要がある．

本節では，通信ソフトウェアを構成するプロセス毎の仕様に対し，それが満たすべき原始要求を仕様要素の実行順序に関する規則とみなして拡張状態遷移図で形式的に表現することとし，与えられた仕様に含まれる仕様要素の実行系列を，この拡張状態遷移図に照らしてすべて検査することによって，原始要求を満たしているか否かを機械的に自動判定する方法を提示し論じる．

5．4．2　実行系列の検査による意味検証の原理

本意味検証の目的は，通信ソフトウェアを構成する各プロセスの仕様に対し，それが満たすべき条件を与え，その条件を仕様が満たすか否かを機械的に判定することにある．仕様が満たすべき条件の例としては，タイマのセット処理を行う場合それ以降の実行系列では必ずリセット処理を実行する必要があること，交換機の動作状態が通話状態に遷移する場合は必ずその前に課金処理を開始していること，等が挙げられる．従来このような条件（表明）の充足性に基づく検証を行うためには，述語論理や時相論理等，理解性や実用性が低いと言われている論理を適用する必

要があった．このため一般には実用に到ってはおらず，実際にはこのような意味検証は通常人手に依ることとなり，検証効率が極めて低いという難点があった．

本検証法は，このような現実的な問題を解決するために考案したものである．その基本原理は，プロセス仕様が SDL のような手続き型言語で記述されていることを前提として，仕様が満たすべき条件を，仕様要素の実行順序に関する「規則」として形式化し，仕様に含まれる全実行系列を検査して規則の充足性を機械的に判定することにある[103],[104]．例えば，タイマに関する実行規則について検証するには，実行可能な仕様要素のすべての系列に対し，タイマのセット処理が実行された場合には，それ以降必ずリセットが実行されるか否か検査すればよい．ここで，仕様要素の実行可能な系列を仕様要素の実行系列，あるいは単に実行系列と呼ぶ．更に，実行の流れに沿った仕様要素列を，実行の流れの分岐点と合流点で分割したものを部分系列と呼ぶ．

図5．4．1(a)は，通信ソフトウェア仕様の一部を SDL/GR で表現した一例である．SDL/GR の各シンボルを仕様要素とみなす．この図の例に含まれる部分系列を，仕様要素の系列で表現すると，

[A, S1, B], [S2], [S3], [C, S4, D]

となる．これらの部分系列を，q1, q2, q3, q4 でそれぞれ表すと，図5．4．1(a)の仕様は図5．4．1(b)のように表現することができる．このように仕様を部分系列のみで表した図を実行系列グラフと呼ぶ．図5．4．1(a)の仕様の実行系列は，同図(b)の実行系列グラフから，

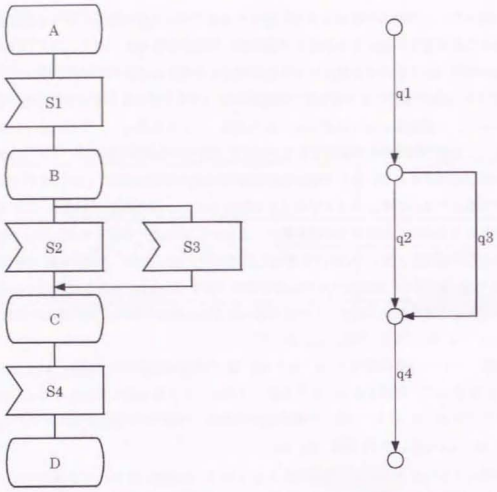
[q1, q2, q4]，及び，[q1, q3, q4]

となる．

仕様がループ状の実行の流れを含む場合は，実行系列の個数は無限となる．例えば，図5．4．2の実行系列グラフにおける実行系列は，

[q1, q2, q4], [q1, q2, q3, q2, q4],…, [q1, $(q2, q3)^{i-1}$, q2, q4],…

となる．ここで$(\ )^i$は，( )内の部分系列の i 回の繰り返しを表す．



(a) SDL/GR 表現　　(b) 実行系列グラフ表現

図5.4.1　通信ソフトウェア仕様の SDL/GR 表現と実行系列グラフ表現の例

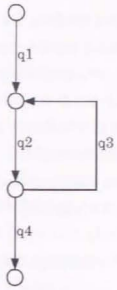


図5.4.2　ループを含む実行系列グラフの例

5.4.3　実行規則グラフ

与えられた仕様が，誤り判定基準となる実行規則を満たしているか否かを自動的に検査するには，実行規則を形式的に表し，機械的に処理できるようにする必要がある．このような実行規則の形式的な表現法として，定義5.4.1に示す拡張状態遷移図 ESTD(Extended State Transition Diagram)を提案する．実行規則を表すこの ESTD を実行規則グラフと呼ぶ．

[定義5.4.1] (実行規則グラフ)

仕様要素の実行規則を表す実行規則グラフ G は，以下の9項組(S0, S1, S2, S3, T1, T2, Q1, Q2, M)から構成される拡張状態遷移図とする．

○Si(i=0,…,3)は，それぞれ以下の検査状態を表す節点の有限集合である．

S0：検査を開始する時点での検査状態(初期検査状態又は初期状態と呼ぶ)

S1：そこに遷移した場合，直ちに実行規則違反である(誤りがある)と判定する検査状態(誤り検出状態と呼ぶ)

S2：そこに遷移した場合，①テスト条件が定義されていない，あるいはテスト条件が定義されていて成立するときは実行規則を満たす(誤りが無い)と判定し，②テスト条件が定義されていて成立しないときは実行規則違反である(誤りがある)と判定する検査状態(条件付き誤り検出状態と呼ぶ)．

S3：そこに遷移した場合，S2 と同様に誤りの有無を判定し，その後検査を終了する検査状態(検査終了状態と呼ぶ)．

○Ti(i=1, 2)は，それぞれ以下の検査状態間遷移を表す有向枝の有限集合である．

T1：検査によって検出された仕様要素に応じて実行する状態遷移．ただし，テスト条件が定義

されているときは，それを満たすときのみ遷移を実行する．

T2：別の実行系列の検査を開始するために実行する無条件遷移．

○Qi(i=1, 2)は，それぞれ以下に示すテスト条件の有限集合である．

Q1：検査状態を表す節点の集合S2及びS3の各節点に対して定義されるテスト条件．

Q2：状態遷移を表す有向枝の集合T1の各枝に対して定義されるテスト条件．

これらのテストでは，①その時点での検査状況，即ち，一つの実行系列の検査途中か，一つの実行系列の検査終了直後か，あるいは未検査実行系列が残っているかの判定，及び，②有限個の値をとる変数と定数とから構成される論理関係の判定を行うことができる．

○Mは，状態遷移を表す有向枝の集合T1とT2の各枝及び検査開始前の初期状態に対して定義され，有限個の値をとる変数及び定数とから構成される演算処理の集合である．

[定義終]

定義5．4．1の拡張状態遷移図は，次の5点において状態遷移図を拡張したものとなっている．

①各種の変数や検査状況（これらをまとめて検査情報と呼ぶ）を蓄積するメモリ機構がある．

②状態遷移を表す有向枝に対し，その遷移を実行する前に満足しなければならない任意のテスト条件を与えることができる．

③状態遷移を表す有向枝に対し，その遷移に伴って実行する演算処理を付加することができる．

④検査状態を表す節点に対し，仕様検査途中，あるいは検査終了後，正常性を判定するための任意のテストを付加することができる．

⑤一つの実行系列の検査を終了し，別の実行系列の検査の開始を表す無条件状態遷移を付加することができる．

なお，本拡張状態遷移図ESTDは，自然言語の文法を記述するために使用される拡張状態遷移網ATN(Augmented Transition Network)を，上述の④と⑤において拡張したものとみなすことができる．ATNの表現能力はチューリング機械と同等であることが知られている．ESTDはATNを部分集合として包含するので，ESTDの表現能力はチューリング機械以上である．即ち，理論的には，ESTDは，計算機で実行できるすべての処理を表現することが可能であり，更に上述の拡張点④と⑤によって複数個の実行系列にわたる仕様要素の実行規則を表現することが可能である．

前項で述べた，タイマのセット(Set)・リセット(Reset)に関する実行規則は，図5．4．3に示す実行規則グラフで表現できる．この図で，*( )は，( )内の仕様要素を除く任意の仕様要素を表す．

この実行規則に比較してやや複雑な例として，「タイマをいったんセット(Set)すれば，それ以降，そのタイマをリセット(Reset)する実行系列が少なくとも1個必要であり，かつそのタイムアウト(Timeout)処理を実行する実行系列がちょうど1個必要である．そして，セットする前にリセットが行われたり，リセット前に再セットが行われる場合は誤りである．」という実行規則を考える．このような複数個の実行系列にわたる実行規則は，通常の状態遷移図で表現することは不可能であるが，図5．4．4で示すように，ESTDによって表現することができる．この図で，[ ]

は，定義5．4．1のMに属す演算処理，‘ ’は定義5．4．1のQ1に属すテスト条件である．2重の矢印は，定義5．4．1のT2に属す無条件状態遷移で，その遷移元の検査状態に到達した場合，検査中の実行系列の検査を停止し，遷移後の検査状態に最初に遷移したときの仕様要素から始まる未検査実行系列の検査を開始するためのものである．また，nr，nt はそれぞれ検査開始時に0に初期設定された後，Reset及びTimeoutを検出するごとに1づつ増加するカウンタである．ただしその最大値は2とする．すべての実行系列を検査し終えた時点での検査状態は図5．4．4の番号が2の状態であり，nr≧1，nt=1が成立するときのみ誤りなしと判定する．このように図5．4．4の実行規則グラフは，先述したタイマのリセットとタイムアウトに関する実行規則を正しく表現している．

なお，実用的な通信ソフトウェア仕様では，複数個の実行系列にまたがった仕様要素に関する条件の充足性が誤り検出判定に必要になることが少なくない．そのような条件の例を以下に示す．

①タイマをセットした以降では，一つの実行系列にのみタイムアウト（タイムオーバ）の記述が必要で，他のすべての実行系列ではタイマのリセット（ストップ）の記述が必要である．

②各状態においてはいくつかの信号を受信した後の実行系列が記述されるが，これらの信号は互いに異なっている必要がある．

③ある仕様要素において判断分岐を行う場合，判断分岐の条件は互いに異なっている必要があるうえ，全体として漏れがあってはならない．

④いったん信号を受信したがその状態では受信せずバッファ上で先送りする場合（SDLにおけるSAVE機能），この状態に続く各状態ではそのような信号を受信処理する記述が必要である．

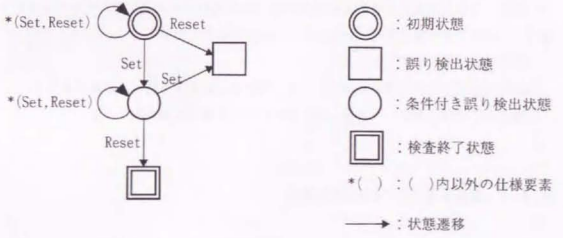


図5．4．3　実行規則グラフ(拡張状態遷移図)で表現した規則の例1（タイマに関する規則）

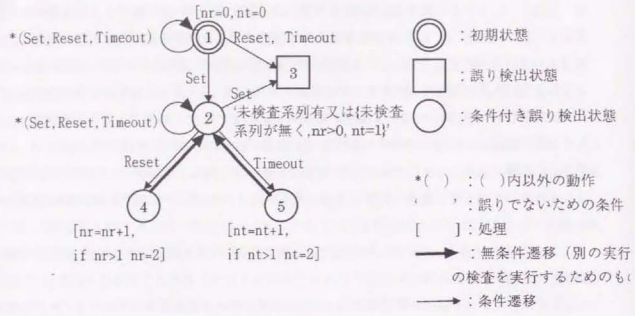


図５．４．４　実行規則グラフ(拡張状態遷移図)で表現した規則の例２（タイマに関する規則）

５．４．４　通信ソフトウェア仕様意味検証問題の形式的定義

以上で定義した実行規則グラフを用いて，通信ソフトウェア仕様の意味検証問題を次の通り定義する．

[定義５．４．２]（通信ソフトウェア仕様の意味検証問題）

通信ソフトウェア仕様の意味検証問題は，被検証仕様のすべての実行系列について，初期状態から出発して実行系列に含まれる仕様要素毎に実行規則グラフで定義された状態遷移を繰り返し実行し，次の３点が満足されているか否かを検証することである．

①誤り検出状態に遷移しないこと．

②条件付き誤り検出状態に遷移したとき，指定されたテスト条件を満足すること．

③検査終了状態に遷移したとき，指定されたテスト条件を満足すること．

[定義終]

５．４．５　実行規則に基づく仕様意味検証

定義５．４．２に従って通信ソフトウェア仕様を検証するには，原理的には，仕様に含まれるすべての実行系列を実行規則に照らして検査すればよい．しかし，実際にはこの方法を採用することはできない．なぜなら，図５．４．２で示した通り，一般には実行系列は無限にあり，この場合検査処理が有限時間では終了しないからである．

以下では，実行系列が無限にある場合でも，有限時間で検証を完了できる効率よい検証アルゴ

リスムを提示する．まず，本項では，このような検証アルゴリズムで必須となる「無限個の実行系列を有限時間で検査するためのアルゴリズム」を示す．この実行系列検査アルゴリズムは，誤り判定基準となる実行規則には依存しないものである．次項５．４．５と次々項５．４．６で，誤り検出のための処理等，実行規則に基づく処理を含む仕様検証全体のアルゴリズムを示す．

本項で示す実行系列検査アルゴリズムでは，検査対象とする実行系列を体系的に管理するため，検査木と呼ぶ実行系列の木表現を利用する．検査木の定義は次の通りである．

[定義５．４．３]（検査木）

検査木は，仕様の部分系列を表す枝から構成される木で，以下の手順で作成したものである．

s1：頂点(根)を設定する．

s2：最初に実行する仕様要素から始まる部分系列を表す枝を頂点に付加する．

s3：作成された木において，端点(葉)を含む各枝に対し，それに対応する部分系列に引き続いて実行することができる部分系列を表す枝を付加する．

s4：s3による枝の付加を可能な限り繰り返す．

[定義終]

図５．４．１(b)と図５．４．２.に示した実行系列グラフの検査木をそれぞれ図５．４．５(a)と(b)に示す．両実行系列グラフはいずれも実行の流れの合流点を１個づつ含むが，検査木ではその合流点に対応する節点を黒丸で表し，x_1, x_2; y_1, y_2,…と記号を付けた．図５．４．５(a)では，節点 x_1 と x_2 以降の部分木は完全に同一である．図５．４．５(b)では，節点 y_i (i=1, 2,…)以下の部分木がすべて同一であるだけでなく，更に，y_i と y_{i+1} の間の部分木は i に依らず同一である．

以上から明らかなように，検査木は実行系列グラフを実行の流れに沿って展開した結果であり，その性質に関して以下の補題が成立する．

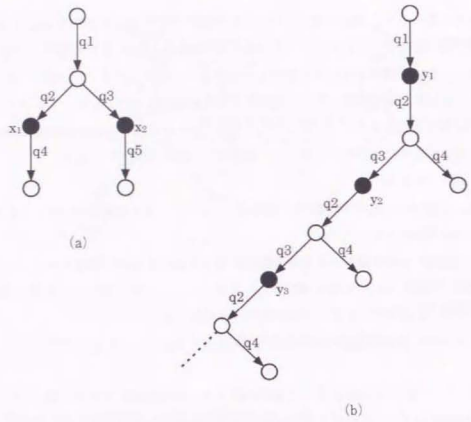


図５．４．５　図５．４．１(b)と図５．４．２の実行系列グラフの検査木

[補題５．４．１]（検査木の性質）

検査木には次の性質がある．

①各実行系列は，頂点から端点に到るパスで表される．

②仕様における実行の流れの合流点は，検査木では複数個の節点に対応する．一つの合流点に対応する節点を $|x_i|$ $(i=1, 2, \cdots)$ とすれば，節点 x_i 以下の部分木は i に依らずすべて等しい．更に，x_i の子孫が $|x_j|$ $(j \neq i)$ に含まれるとき，そのような節点のうち x_i を含む各実行系列上で x_i に最も近いものを抽出し，それらを $x_{ik}(k=1, 2, \cdots)$ とすれば，x_i と x_{ik} の間の部分木は i に依らずすべて等しい．

[補題終]

ところで，各部分系列を検査した後の検査状態と検査変数の値（これらをまとめて検査情報と呼ぶ）は，検査前のそれらの値が与えられれば唯一に決まる．従って，次の補題が成立する．

[補題５．４．２]（部分系列の検査）

一般に，複数個の実行系列が同一の部分系列を含んでおり，かつそれ以降完全に同一である場合，その部分系列を検査する直前での検査情報が同一であれば，その部分系列以降の任意の時点における検査情報も等しい．即ち，このような場合の仕様検証では，共通の部分系列以降の検査は１回行うだけで十分である．

[補題終]

補題５．４．１と補題５．４．２とから次の補題が成立する．

[補題５．４．３]（検査の停止）

一つの合流点に対応する検査木上の節点において，検査情報が一致している場合，これらの節点以下の検査は一度だけ行えば十分である．また，そのような節点で囲まれた部分木が複数個ある場合は，その内の１個の部分木についてのみ検査すれば十分である．

[補題終]

補題５．４．３から，実行系列の検査は，検査木を利用して，以下のアルゴリズムに従って検査すればよいことが言える．

[アルゴリズム５．４．１]（検査木を用いた実行系列の検査）

定義５．４．３の手順に従って検査木を構成する．そのとき，枝を１本付加するごとに，それに対応する部分系列を検査し，検査後の検査情報を導出する．その結果，以下の条件をすべて満たす場合は，定義５．４．３の手順においてその枝には以降枝を付加しないこととする．

①その部分系列に対応する枝の先の節点(端点)が，仕様上の合流点に対応する．

②その合流点に対応する別の節点が既に検査木の祖先部に含まれており，かつ両者の検査情報が完全に一致する．

[アルゴリズム終]

[定理５．４．１]（アルゴリズム５．４．１の正当性）

アルゴリズム５．４．１は，すべての実行系列の検査を正しく行う．

[定理終]

[定理５．４．１の略証]

これまでの説明から，アルゴリズム５．４．１によって仕様に含まれるすべての実行系列を検査することができるのは明らかである．

一方，定義５．４．１から，本章で扱う検証問題における検査情報は有限である．即ち，検査木において，部分系列を表す枝の各節点における検査情報の組合せは有限である．従って，アルゴリズム５．４．１の処理によって，定義５．４．３の s4 において枝の付加が無限に続くことはなく，アルゴリズム５．４．１は必ず停止する．

[略証終]

例えば，図５．４．５(a)では，節点 x_1 での検査情報と節点 x_2 での検査情報が同一であれば，節点 x_1，x_2 以下の部分系列 q4，q5 の検査はどちらか一方のみ行えば十分である．また，同図(b)では，例えば節点 y_2 と y_1 の検査情報が同一であれば節点 y_2 以下の部分木は不要で，それに対応する検査を行う必要はない．節点 y_2 と y_1 の検査情報が異なる場合でも，節点 y_2 の検査情報が節点 y_1 又は y_2 の検査情報と一致していれば，節点 y_3 以下の部分木及びその検査は不要となる．検査情報のとり得る値の組合せは有限であるから，節点 y_1，y_2，y_3，…の検査情報がすべて異なるということはあり得ない．

5.4.6 手続き内仕様検証アルゴリズム

本項では，まず実行規則が1個でかつ手続き呼び出しを含まない仕様に対する検証アルゴリズム（手続き内検証アルゴリズム）を示し，次に実行規則が複数個の場合の同様なアルゴリズムを示すが，それらの前に，仕様誤りを検出した場合の処理に関する方針を述べる．

一般に，検証処理において仕様誤りを検出した場合の処理としては，①検証処理をいったん停止し，仕様作成者が仕様を訂正した後，検証処理を再開するという処理と，②検出した仕様誤りの原因を予め推定しておき，それに応じた回復処理を自動的に実行して検証を続行するという処理の2案が考えられる．しかし，仕様誤りの原因を正しく推定することは必ずしも可能ではないので，ここでは案①を採用することとした．

この方針に則り，前項で述べた実行系列検査アルゴリズムに基づく手続き内検証アルゴリズムを以下に示す．また図5.4.6はそのフローチャートを示す．なお，本アルゴリズムは，主ルーチンと1個の副ルーチンとから構成される．

[アルゴリズム5.4.2]（実行規則が1個の場合の手続き内検証アルゴリズム）

＝主ルーチン＝

S1　初期仕様要素を含む部分系列を未検査部分系列として登録する．

S2　未検査部分系列を1個抽出する．

S3　抽出した未検査部分系列から実行順序に従って仕様要素を1個抽出する．

S4　実行規則グラフにおいて遷移を実行するためのテスト条件が定義されていればS5へ，なければ副ルーチン（実行規則での状態遷移及び誤り検査）を呼び，S6へ進む．

S5　テスト条件が成立する場合は副ルーチンを呼び，S6へ進む．成立しない場合はS7へ進む．

S6　検査処理終了フラグがセットされていれば検証を終了し，リセットされていればS7へ進む．

S7　検査した仕様要素が部分系列の最後である場合は，S8へ進む．最後でない場合は，実行規則グラフで無条件遷移が定義されていれば副ルーチンを呼びS10へ進む．無条件遷移が定義されていなければS3に戻る．

S8　アルゴリズム5.4.1の条件を満たす場合は直ちにS9へ進む．条件を満たさない場合は，検査が終了した部分系列の検査情報を蓄積するとともに，その部分系列に引き続いて実行できる部分系列を未検査部分系列として登録し，S9へ進む．

S9　実行規則グラフで無条件遷移が定義されていれば副ルーチンを呼び，S10へ進む．無条件遷移が定義されていないときはS11へ進む．

S10　検証処理終了フラグがセットされていれば検証を終了し，リセットされていればS11へ進む．

S11　未検査部分系列が残っている場合はS2に戻る．残っていない場合は，誤りなしメッセージを出力して検証を終了する．

＝主ルーチン終＝

＝副ルーチン＝（実行規則での状態遷移及び誤り検査）

SS1　実行規則グラフの遷移を実行する．演算処理があれば実行する．

SS2　遷移後の検査状態が，誤り検出状態であればSS3へ，検査終了状態であればSS4へ，また条件付き誤り検出状態であればSS5へ進む．

SS3　誤り検出メッセージを出力する．検証処理終了フラグをセットして主ルーチンへ戻る．

SS4　テスト条件が無い，又はテスト条件があってもそれが成立するときはSS6へ進み，テスト条件があってそれが不成立のときはSS3に戻る．

SS5　テスト条件が無い，又はテスト条件があってもそれが成立するときは，検証処理終了フラグをリセットして主ルーチンに戻る．テスト条件があってそれが不成立のときはSS3に戻る．

SS6　誤りなしメッセージを出力する．検証処理終了フラグをセットして主ルーチンに戻る．

＝副ルーチン終＝

[アルゴリズム終]

本アルゴリズムに関し，次の定理が成立する．

[定理5.4.2]（アルゴリズム5.4.2の正当性）

アルゴリズム5.4.2は検証を正しく行う．即ち，すべての実行規則誤りを検出し，必ず停止する．

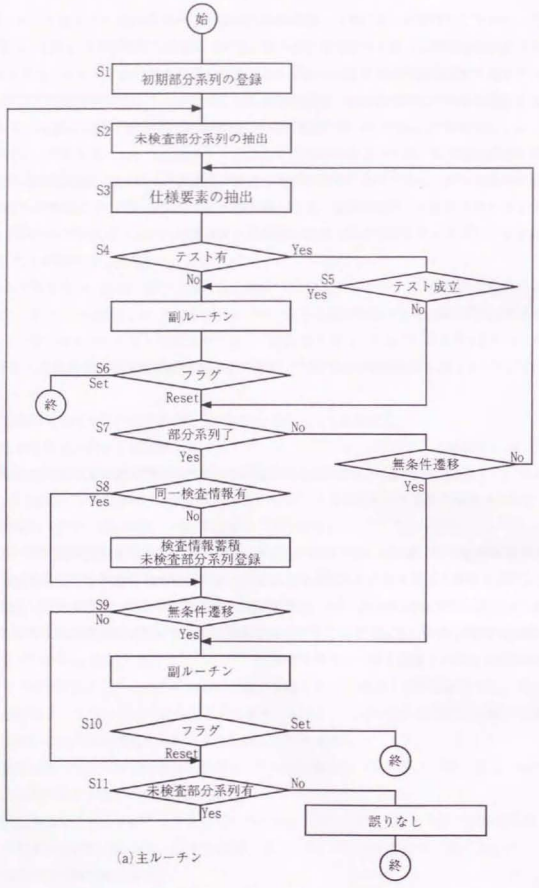
[定理終]

[定理5.4.2の略証]

定理5.4.1から，アルゴリズム5.4.2が仕様誤りをすべて検出可能で，かつその処理が必ず停止することを証明することができる．

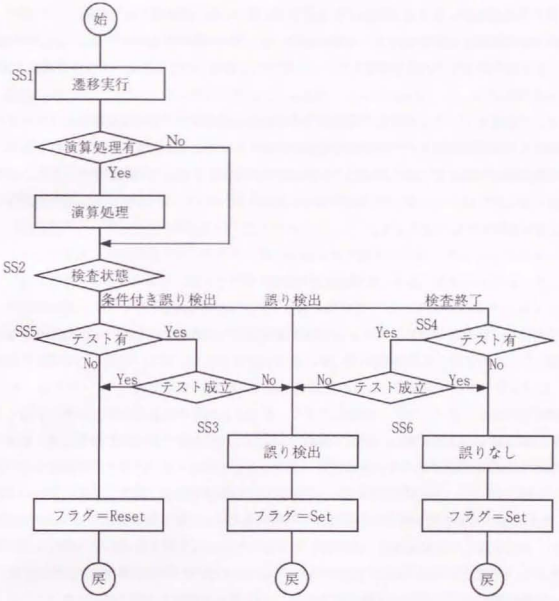
[略証終]

実行規則が複数個ある場合は，与えられた仕様の実行系列に含まれる仕様要素毎にすべての実行規則に対して同様の処理を実行すればよい．ただし，各部分系列の検査を終了するには，すべての実行規則に対する検査が終了している必要がある．例えば，図5.4.2のようなループ状の実行の流れが仕様に含まれる場合は，すべての実行規則について，同一の検査情報が再現されるまで部分系列を繰り返し検査し続ける必要がある．



(a)主ルーチン

図5.4.6. 仕様検証アルゴリズムのフローチャート(1/2)
(S#とSS#は，それぞれアルゴリズムのS#とSS#を示す)



(b)副ルーチン（実行規則での状態遷移及び誤り検査）

図5.4.6 仕様検証アルゴリズムのフローチャート(2/2)
(S#とSS#は，それぞれアルゴリズムのS#とSS#を示す)

5.4.7 手続き間仕様検証アルゴリズム

例えばSDLのような仕様記述言語では，手続き呼び出しを使用することができる．手続き呼び出しを含む仕様の検証アルゴリズム(手続き間検証アルゴリズム)は，次に示す通りである．

[アルゴリズム5.4.3]（手続き間検証アルゴリズム）

手続き内検証アルゴリズム（アルゴリズム5.4.2）に以下の処理を追加する．
手続きを呼ぶ場合，まずその時点での検査情報を用いてその手続きを検査する．次に，その結果得られる手続きの出口での検査情報を用いて，呼び出し側の元の仕様要素から検査を再開する．

[アルゴリズム終]

なお，手続きをいったん検査した場合は，その入口と出口での検査情報を対としてデータベースに登録しておき，再び同一の検査情報でその手続きを呼ぶ場合は，データベースに格納してあった検査情報を利用することによって，その手続きの再検査を省くことができる．また，手続きの入口と出口をそれぞれ仕様の流れの分岐点と合流点とみなすことによって，手続きの再帰呼び出しを含む仕様も検証可能となる．

5.4.8 実行系列検査に基づく仕様検証法の評価

本検証法を，海事衛星通信システムにおける海岸局の出力処理プロセスに適用し，その有効性を評価した．このプロセスの規模はSDL/PRで約2,000行である．また，利用した仕様実行規則は図5.4.4に示すタイマー規則で，この規則の作成には約30分を要した．

評価試験の結果，タイムアウト処理とタイマーリセット処理の記述漏れがそれぞれ1個と2個の合計3個の誤りが検出された．検証ツールによってこれらの誤りを検出するのに要した時間は操作時間を含め約5分であった．検出したこれらの誤りは別途人手で仕様を目視検査することによっても検出できた．この人手作業に要した時間は約120分である．即ち，検証のための作業効率は，人手で行う方式に比較すると本検証ツールの利用によって約((120-5-30)/120=)約71%削減できた．今回利用した仕様規則は一度作成すれば他の仕様にも適用可能であり，その作成時間を除外すると，本検証ツールによって検証時間を((120-5)/120=)約96%削減できることになる．

この例は比較的小さい規模の仕様の例であり，一般には，本検証法及びそれに基づく検証ツールによって，以下の効果が得られることを確認した．

- ○複数の実行系列に関わる条件を誤り判定基準（規則）とする意味検証が可能である．
- ○検証に使った誤り判定基準（規則）を蓄積しておくことによって，検証機能を容易に拡張することができる．
- ○人手に比べると数分の1から数10分の1の作業時間で検証でき，検証効率を大幅に向上できる．
- ○本検証法は，SDL以外に，任意の手続き型通信ソフトウェア仕様記述言語に適用できる．
- ○手続き呼び出しを含む仕様を検証することができる．
- ○ループ状の実行系列を含む仕様に対しても，有限時間で検証することができる．

5.4.9 まとめ

本節では，通信ソフトウェア仕様の意味検証を効率よく実現するため，原始要求を入力し，それが満たされているか否かを機械的に検査する方法を提案した．この方法の特徴は，次の通りである．

①原始要求を，仕様に含まれている要素の実行順序に関する規則として，拡張状態遷移図によって形式的に表し，その充足性を機械的に検査できること．

②通常の状態遷移図では表現できない「複数の実行系列にまたがる仕様要素に係わる規則」もこの拡張状態遷移図で表現でき，そのような規則の充足性も機械的に検査できること．

③規則の充足性の検査では，仕様に含まれている実行系列をすべて検査するが，ループ状の実行系列があっても検査が有限時間で完了すること．

④本検証法は，手続き呼び出しを含む任意の手続き型仕様記述言語で記述された仕様の意味検証に適用することができ，適用性が高いこと．

本意味検証法によって，従来法に比較して，検証作業時間を例えば1/20に削減することが可能となり，検証効率の大幅な向上が達成できた．

5.5 むすび

本章では，通信ソフトウェア仕様の意味検証を実現する新しい検証技術として，抽象処理を特徴とするプロトタイピング法，仕様を簡明な表現に機械的に変換する仕様簡明化法，与えられた要求を仕様が満たしているか否かを検査する実行系列検査法を提案した．そして，これらの新意味検証法によって，次に示す通り，従来は不可能であった意味検証を効率よく達成することが可能であることを示した．

①抽象処理を特徴とするプロトタイピング法によって，これまでは不可能であった「段階的な仕様の作成・検証を直接支援すること」が可能になった．即ち，不完全な通信ソフトウェア仕様を入力してプロトタイププログラムに変換し，それに含まれている正確な情報を最大限利用して実行する．そして，実行結果として，確かな部分と不確かな部分とに明確に分けて出力する．これによって，特に複数の技術者の間で共同して仕様を作成する場合に，誤りを含む仕様部分を基に他の仕様部分を作成するという悪循環を回避でき，正確な仕様作成の効率が大幅に向上できる．

②仕様簡明化法によって，与えられた通信ソフトウェアに対し，機能的には同一であるが，例えば仕様要素数がより少なく規模が小さな表現に自動的に変換することができる．この結果，特に仕様規模が大きい場合に仕様作成者自身でも困難となる仕様の理解が容易になる．また，機能的に等価でより簡明な表現の仕様を作成者が検査し理解することによって，作成した仕様の正当性を検査することも可能となる．一方，ある技術者が作成した仕様部分を基に別の技術者が別の部分の仕様やプログラムを作成するとき，仕様の理解が容易に

なり，仕様やプログラムの作成効率が向上できる．

③実行系列の検査による意味検証法によって，与えられた原始要求の充足性を容易に判定することができる．特に，複数の実行系列にまたがった仕様要素に関係する原始要求の充足性を機械的に判定できる．この機械的な判定は従来は実現できなかったが，本意味検証によって初めて達成できた．また，このような充足性の判定に要する計算機処理時間は極めて小さく，人手による場合に比較すると，数分の1から数10分の1以下に検証時間を短縮できる．

以上で示したとおり，本章で提示した意味検証法によって，従来は低かった意味検証効率を大幅に向上できることを明らかにした．

本論文は，通信ソフトウェアの仕様に含まれる誤りを検出する仕様検証技術に関するものであり，その目的は，仕様誤りのうち論理誤りと意味誤りを検出する検証法の効率を向上させる技術の確立にある．この目的を達成するため，まず，通信ソフトウェア仕様の中で重要な位置を占めるプロトコル論理に関する誤りを検出する4件の新しいプロトコル検証技術を考案し技術確立した．次に，通信ソフトウェア仕様の意味誤りを検出する新検証技術として，抽象処理を特徴とするプロトタイピング法，仕様を理解し易い簡明な表現に変換する仕様簡明化法，及び仕様が満たすべき条件の充足性を判定するため仕様に含まれる実行系列を検査する方法の3手法を考案し技術確立した．そして，これらの新技術によって，通信ソフトウェア仕様検証の効率が大幅に向上できることを明らかにした．

本論文によって得られた主要な具体的成果は以下の通りである．

(1)プロトコル検証の効率向上法

プロトコル検証に関し，次の(1-1)～(1-4)に示す通り，従来法に比較して検証効率が飛躍的に優れ，検証に必要なメモリ量・処理量の両者について大幅な削減を可能とする新しい検証技術を確立した．

(1-1)プロセス削減法

本手法は，タイマやリソース管理のためのプロセスのように，1個のプロセス（親プロセス）との間でのみしか信号の授受を行わないプロセス（子プロセス）がある場合，親プロセスに特別な処理を施し，子プロセスを削除してプロトコル検証を行うものである．このような処理によって，プロセスを削減してプロトコル規模を縮小することが可能となり，その結果，検証に必要となるメモリ量と処理量を大幅に削減できる．

(1-2)プロセス状態遷移の一括処理法

本手法では，プロトコル動作を検査するためのグローバル状態遷移図の展開において，誤り検出に支障のない範囲で，プロセス状態遷移をできる限りまとめて一括処理する．このような一括処理によって，生成するグローバル状態遷移図に含まれる状態と遷移等を直接削減し，その結果，検証に必要となるメモリ量と処理量を大幅に削減できる．

(1-3)プロトコル仕様のアサイクリック展開法

本手法では，プロトコル動作を検査するため，各プロセスの状態遷移図をアサイクリック状に展開することとし，その際他のプロセスに関する情報の管理は最小限に留める．プロセス状態遷移図の展開において，等価な展開状態が得られた場合，それ以降は同じ展開の繰り返しとなるので，等価な展開状態以降では，一つの等価状態からのみの展開を実行し他の等価状態からの冗長な展開を省略することによって，展開規模を小さくする．これによって，プロトコル検証に必要なメモリ量と処理量を大幅に削減できる．

(1-4)段階的検証法

本手法では，検出すべきプロトコル誤りを，プロトコル検証における所要メモリ量と処理量の支配的要素であるグローバル状態に直接係わるデッドロックと，グローバル状態に直接は係わらない他の誤りとに分類し，分類された誤り種別毎に検証を段階的に分けて実施する．まず第 1 段階目の検証では，後者に属す誤りを検出するため，上述した(1-3)プロトコル仕様のアサイクリック展開によって各プロセス毎の動作を中心とする検査を実行する．第 2 段階目の検証では，全プロセス動作の同時検査を実施し，グローバル状態を列挙してデッドロックを検出する．いずれの段階の検証においてもプロトコル動作を表す状態遷移図を展開し作成するが，それぞれの誤り検出に支障のない範囲で，列挙する状態や状態遷移を必要最小限に留める．特に，プロセス毎の動作を中心にした第 1 段階の検証では，原則としてグローバル状態は列挙しないこととする．そして，第 2 段階の全プロセス動作の同時検査では，第 1 段階の処理で実行可能と判定された各プロセスの遷移のみを用いたうえ，(1-2)プロセス状態遷移の一括処理と同様に，グローバル状態とグローバル遷移の列挙を最小限に押さえる．このような 2 段階の展開処理によって，所要メモリ量と処理量の支配的要素となっていたグローバル状態と遷移の総数を大幅に削減する．

以上の 4 手法のうち，(1-1)プロセス削減法は，他の 3 方法とは独立なので同時に組み合わせて利用することが可能であり，その結果，検証効率は相乗的に向上できる．

これらの新プロトコル検証法によって，従来法に比較して，検証効率を大幅に向上でき，所要メモリ量と検証処理量を飛躍的に削減することが可能となった．例えば，プロセス削減法によって，タイマプロセスを 1 個削除すれば所要処理量は 1/3～1/5 に減少し，タイマプロセスを 2 個削除すれば所要処理量は 10 数分の 1 に減少する．また，段階的検証法によって，従来のプロトコル検証法に比較して，所要メモリ量と検証処理量の両者とも 100 分の 1 以下に削減できる．段階的検証法には，検証すべきプロトコルの規模が大きいほど，特にプロセスが多いほど，従来法に比較した所要メモリ量・検証処理量の削減効果が大きくなるという特長がある．総合的には，両手法を組み合せることによって，検証効率を数 100～10,000 倍以上向上させることが可能となった．

本検証法によって，従来法に比較して検証効率を 100 倍以上向上させるという当初の研究目標は十分に達成できた．そして，例えばプロセスが多く従来は検証できなかった大規模な実用的プロトコルの検証を可能とした．実際，これまでは検証不可能であった，国際海事衛星通信システム B 及び M，国際 SDH 伝送路障害切り替えシステム，国際電話用新交換台システム等の現在実用化されている国際通信システムのプロトコル開発において，本論文によって確立したプロトコル検証技術を適用した結果，これらをすべて検証でき，その有効性を実証できた．

(2)通信ソフトウェア意味検証の効率向上

次に，通信ソフトウェア仕様の効率よい意味検証を実現するため，従来にはなかった検証機能を持つ新しい検証技術を確立した．具体的には，以下の(2-1)～(2-3)に示す通りである．

(2-1)抽象処理を用いたプロトタイピングによる検証

本検証法では，変数の値が設定されない，手続きの内容が記述されていない等，不完全な通信ソフトウェア仕様を入力し，それをプログラムに自動変換して最大限に実行する．そして，実行

結果を，正しいものと必ずしも正しくないものとに明確に区別して出力する．また，実行開始にあたって，実行すべき仕様部分を指定することが可能で，この指定によって分岐判断のための変数の値の自動設定や実行すべき他の仕様部分の自動決定を可能な限り行う．これによって，通信ソフトウェア仕様の部分的･段階的な作成を効率よく支援する意味検証が可能となった．

(2-2)仕様簡明化による検証

与えられた通信ソフトウェア仕様を入力し，それと等価でかつより簡明な表現に自動変換する様々な手法を開発した．特に，検査すべき仕様の規模を削減する手法として，無効な実行系列の削除，分岐要素の順序変更による仕様要素の削減等を考案し，実際の通信システムに適用してその評価結果を論じた．その結果，考案手法によって，例えば人手で作成した仕様の規模を 3/4 程度に縮小した簡明な表現に変換することが可能となり，目視で意味検証する場合の検査作業量をかなり削減でき，検証効率を大幅に向上できるようになった．

(2-3)実行系列の検査による検証

与えられた通信ソフトウェア仕様が満たすべき条件を，仕様要素の実行順序に関する規則として与え，仕様がその規則を満たすか否かを効率よく判定する手法を開発した．この手法によって，従来は不可能であった，複数の実行系列に係わる規則に対する機械的な検証も可能となり，人手による目視検証に比較して，例えば検証作業時間を 1/20 以下に低減でき，検証効率の大幅な向上を可能とした．

以上の通り，本論文で確立した新しい仕様検証技術によって，通信ソフトウェア仕様の検証効率を大幅に向上することが可能となった．近年情報社会はグローバル化･マルチメディア化が著しく進んでおり，開発･保守すべき通信ソフトウェアは増大する一方である．本論文で確立した通信ソフトウェア仕様検証技術が，このような発展を支える礎石の一つになるものと期待される．

以上

付録　通信ソフトウェア仕様検証支援システム ESCORT

付録1　ESCORT の設計方針

本論文で論じた通信ソフトウェア仕様検証技術を中心として通信ソフトウェア仕様作成支援システム ESCORT (Environment for Specifying Communications Software Requirements)を構築した[105]~[107]．その設計における基本方針は以下の通りである．

①仕様記述言語には，本論文のベースとした拡張状態遷移機械との親和性が高く，国際標準化された SDL(Specification and Description Language)を採用する．

②仕様検証機能には，プロトコル論理検証，プロトタイピング，仕様簡明化，実行系列検査の各機能を具備する．

③仕様検証以外に，SDL のテキスト表現 PR(Phrase Representation)から SDL の図式表現 GR(Graphical Representation)への自動変換機能等を持つ．

④各種機能は，独立したツールとして実現し，それらの使用順序は特に規定せず，任意の順序で使用できることとする．

付録2　ESCORT の構成

ESCORT の機能構成図を図付1に示す．また，ESCORT が持つ諸ツールの一覧を表付1に示す．仕様を処理するための形式は SDL のテキスト表現 PR であり，すべてのツールの入力は SDL/PR 形式の仕様となる．

ESCORT を用いて通信ソフトウェア仕様を作成する手順は以下の通りである．まず，UNIX の vi 等のテキストエディタによって SDL/PR 形式で仕様を作成する．次に SDL の構文検査ツールによって SDL の文法誤りを検出して訂正する．次に，任意の検証ツールを利用して論理誤り･意味誤りを検出し訂正する．ただし，プロトコル検証，実行系列検査，プロトタイピング，簡明化のいずれのツールについても，構文検査を最初に行っており，文法誤りを訂正してから検証を実行する．

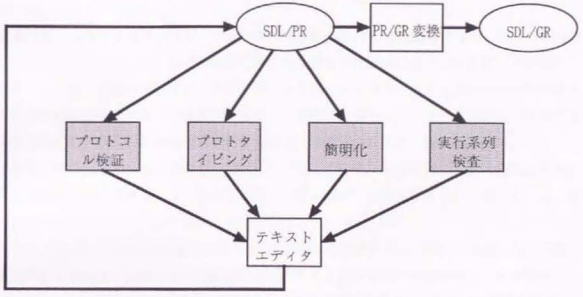
プロトコル検証ツールでは，未定義受信等のプロトコル誤りを検出した場合，初期グローバル状態から誤りに到るまでに各プロセスが実行した遷移の系列をすべて出力することができ，これを利用することによって誤りの訂正が容易になる．本文では述べなかったが，本ツールには，信号受信バッファに蓄積される信号数が予め定めた数より大きくなった場合その旨を出力する機能がある．4．2で述べた通り，信号受信バッファ内信号数が有限でない場合には，検証が有限時間で停止しない可能性があるため，そのような停止性の確認にこの機能を利用することができる．

プロトタイピングツールでは，実行する仕様部分を指定することができる．その指定は，SDL/PR 上で実行系列が分岐する仕様要素に対し特別のコメントを追加挿入して分岐後の実行系列を指定することによる．SDL 表現の仕様をプログラムに変換して実行をいったん開始すると，任意の時点

で割り込んで停止することができ，その時点までの実行結果を出力することができる．この出力には，信号授受のシーケンス，変数の履歴，実行した仕様要素(SDL/PR 文等)等が含まれる．また，停止した部分から実行を再開することも，更には実行済み実行系列の任意の分岐点から別の実行系列を実行することもできる．

仕様簡明化ツールでは，いったん簡明化処理した仕様部分を元に戻す(undo)機能があり，両者を比較して確認しながら順次簡明化処理を進めることもできる．つまり，仕様の検証と簡明化を並行して実施することができる．

仕様要素の実行系列検査ツールでは，人手で作成した規則を与えられた仕様が満たしているか否かを検査するが，規則に示された条件に合致した仕様要素も出力するので，検索ツールとして利用することもできる．また，表付1にある通り，このツールは，論理検証･意味検証の両方に利用することができる．



図付1　通信ソフトウェア仕様作成支援システム ESCORT の構成

表付１　ESCORT が持つ主なツール一覧

分類		ツール名	ツール機能概要
SDL/PR･GR 表現の作成		PR/GR 変換	PR 表現から GR 表現への変換
		PR 清書	PR 表現の清書（インデント付与等）
構文検証		PR 構文検査	PR 表現の構文検査
論理検証	プロセス内	実行可能性検査	PR 文の実行可能性（系列の連続性）の検査
		実行系列検査	規則に照らした PR 文の実行系列の検査
	プロセス間	プロトコル検証	実行不可能送受信･未定義受信･デッドロックの検出
意味検証	要求照合	実行系列検査	規則に照らした PR 文の実行系列の検査
	部分抽出	PR 要素検索	PR 文の検索
	関係に基づく編集	同一・類似ルーチン検索	同一／類似 PR 文の系列の検索

謝辞

東京大学教授斉藤忠夫博士には，本論文のとりまとめに際し，終始懇切なる御指導と御鞭撻を賜りました．謹んで深謝の意を表します．また，研究内容について貴重な御討論と御示唆をいただきました，東京大学教授青山友紀博士，同教授石塚満博士，同助教授相田仁博士，同助教授相澤清晴博士，同助教授瀬崎薫博士に深く感謝いたします．

東京大学名誉教授・学術情報センター長猪瀬博博士には，筆者が通信に関する研究を志すきっかけを与えていただき，在学中並びに卒業後を通じて終始暖かい御指導と御鞭撻を賜りました．ここに深甚の謝意を表します．

本研究の遂行並びにとりまとめの機会を与えていただきました，株式会社ＫＤＤ研究所所長村谷拓郎博士，国際電信電話株式会社取締役・元国際電信電話株式会社研究所所長村上仁己博士，株式会社ＫＤＤ研究所副所長鈴木健二博士，同副所長山本英雄博士，同取締役渡辺文夫博士に深く感謝いたします．

株式会社オーエスアイプラス社長・元国際電信電話株式会社交換部森弘道部長には，本研究の契機を与えていただくとともに国際電信電話株式会社研究所在籍時に懇切な御指導と御鞭撻をいただきました．深く感謝いたします．東京通信ネットワーク株式会社部長・元国際電信電話株式会社企業通信部安藤純利次長，早稲田大学教授・元国際電信電話株式会社研究所所長浦野義頼博士，株式会社ＫＤＤ研究所主席研究員山崎泰弘博士，国際電信電話株式会社グローバルマルチメディア事業部池田佳和審議役，株式会社ＫＤＤ研究所小西和憲主席研究員，国際電信電話株式会社技術企画部部長松本潤博士には，本研究の遂行に際し暖かい御指導と励ましを頂きました．深く感謝いたします．

国際ケーブル･シップ株式会社運行本部次長・元国際電信電話株式会社研究所乗越雅光主任研究員，広島市立大学教授・元国際電信電話株式会社研究所主任研究員角田良明博士，株式会社ＫＤＤ研究所グループリーダ小田稔周博士，同研究所伊藤篤主任研究員には，本研究の遂行にあたり，多大な御助言・御協力と有益な御討論をいただきました．深く感謝いたします．株式会社ＫＤＤテレコムネットワーク課長・元国際電信電話株式会社研究所新田文雄主査，株式会社ＫＤＤ研究所宇都宮栄二担当主査，通信コンサルタント・元国際電信電話株式会社研究所斉藤博徳研究員には，本研究の遂行に御協力いただくとともに有益な御討論をいただきました．深く感謝いたします．

本研究の遂行にあたり，通信ソフトウェア仕様検証支援システムの開発において御協力いただきました，株式会社三菱総合研究所並びに日本通信建設株式会社の関係者の皆様に厚く御礼申し上げます．

参考文献

[1] 白鳥 則朗編："通信ソフトウェア工学"，培風館，(1995年7月)

[2] 松本 吉弘："ソフトウェア工学"，3ソフトウェア要求定義，pp.77-134，丸善 (1992-10)

[3] 情報処理学会："情報処理ハンドブック"，5編プログラミング2章計算モデルとプログラミング言語・6編ソフトウェア工学3章検査・検証，pp.637-649,728-737

[4] 稲垣，坂部："抽象的データタイプの代数的仕様記述法の基礎"，情報処理，Vol.25，No.1，～Vol.25，No.7 (1984)

[5] ITU-T勧告Z.100～Z.104(1988/1992)

[6] 若原 恭："SDL 言語の特質と処理系の現状と動向"，情報処理，29，1，pp.23-34 (1988年1月)

[7] 若原，角田，伊藤，長谷川："通信システム仕様記述言語 SDL"，国際通信の研究，137，pp.88-98(1988-07)

[8] F.Belina and D.Hogrefe:"The CCITT Specification and Description Language SDL", Computer Networks and ISDN systems, 16, pp.311-341(1988/1989)

[9] 若原 恭（水野 忠則編）："プロトコル言語"，カットシステム，2．3SDL，(1994年7月)

[10] 若原，長谷川訳："仕様記述言語SDL"，カットシステム，(1996年11月)

[11] R.Saracco, J.R.W.Smith and R.Reed:"Telecommunications Systems Engineering using SDL", North Holland, Amsterdam(1989)

[12] K.J.Turner:"Using FDTs", John Wiley & Sons, p.431 (1993)

[13] G.I.Parkin and S.Austin:"Overview: Survey of Formal Methods in Industry", Proc., 6th, FORTE'94, pp.189-203(1994)

[14] G.Csopaki and K.J.Turner:"Modelling Digital Logic in SDL", Proc. RORTE X / PSTV XVII, pp.367-382(November 1997)

[15] 河岡，友永，高橋："プロトコルの記述法と検証法"，情報処理，20，7，pp.612-621 (1979-07)

[16] M.Sajkowski:"Protocol Verification Techniques: State quo and Perspectives", Proc. IFIP 4th workshop on Protocol Specification, Testing, and Verification, Skytop Lodge(June 11-14, 1984)

[17] R.E.Miller:"Protocol Verification: The first Ten Years, The Next Ten Years; Some Personal Views", Proc. 10th Int. Workshop on Protocol Specification, Testing and Verification, pp.199-225 (1990)

[18] G.J.Holtzmann:"Protocol Design:Redefining the State of the Art", IEEE Software, 9, 1, pp.17-22(January 1992)

[19] J.C.Coebett:"Evaluating Deadlock Detection Methods for Concurrent Software", IEEE Trans. on Soft. Eng., SE-22, 3, pp.161-180(March 1996)

[20] C.H.West:"General Technique for Communications Protocol Validation", IBM J. Res. Devel., 22, 4, pp.393-404 (July 1978)

[21] P.Zafiropuro:"Protocol Validation by Duologue Matrix Analysis", IEEE Trans. Commun., COM-26, 8, pp.1187-1194 (Aug. 1978)

[22] P.Zafiropuro, C.H.West, H.Rudin, D.D.Cowan and D.Brand:"Toward Analyzing and Synthesizing Protocols", IEEE Trans. on Commun., COM-36, 4, pp.651-661(Sept. 1988)

[23] M.Itoh and H.Ichikawa:"Protocol Verification Algorithm using Reduced Reachability Analysis", Trans. IECE, E66, 2, pp.88-93 (Feb. 1983)

[24] 大友，荒川，平川："縮退到達可能性解析における網羅性について"，信学技報 SSE95-68,IN95-39,CS95-88(1995)

[25] K.C.Tai, H.F.Ho and G.H.Chen:"Protocol Validation Using a Pumping-based Approach", Proc. COMPSAC'91, pp.339-344(1991)

[26] G.J.Holtzmann, P.Godefroid and D.Pirottin:":"Coverage Preserving Reduction Strategies for Reachability Analysis", Proc. 9th Int. Workshop on Protocol Specification, Testing and Verification, pp.349-363 (June 1989)

[27] S.Lam and A.Schankar:"Protocol Verification via Projections", IEEE Trans. on Soft. Eng., SE-10, 4, pp.325-342(July 1984)

[28] 若原，角田，乗越："分割原理に基づくプロトコル論理検証法"，信学技報，SE84-154，pp.55-60(1985-03)

[29] E.M.Clarke, D.E.Long and K.L.McMillan:"Compositional Model Checking", Proc. IEEE Annual Symposium on Logic in Computer Science, pp.353-362 (1989)

[30] B.Algayres, V.Coelho, L.Doldi, H.Garavel, Y.Lejeune and C. Rodriguez:"VESAR: a Pragmatic Approach to Formal Specification and Verification", Computer Networks and ISDN Systems, 25, pp.779-790(1993)

[31] G.Singh and H.Liu:"Validating Protocol Composition for Progress by Parallel Step Reachability Analysis", Proc. RORTE X / PSTV XVII, pp.239-250(November 1997)

[32] D.Brand and P.Zafiropuro:"On Communication Finite-State Machines", J. ACM, 30, 2, pp.323-342 (April 1983)

[33] M.C.Yuang and A.Kershenbaum:"Parallel Protocol Verification Using The Two-Phase Algorithm", Proc., COMPSAC'89, pp.184-192(1989) and Proc. 9th Int. Workshop on Protocol Specification, Testing and Verification, pp.339-353(June 1989)

[34] J.Rudin and C.H.West:"An Improved Protocol Validation Technique", Computer Networks, 6, pp.65-73(1982)

[35] M.Gouda and Y.T.Yu :"Protocol Validation by Maximal Progress State Exploration", IEEE

Trans. Commun., COM-32, 1, pp.94-97 (January 1984)

[36] C.H.West:"Protocol Validation by Random State Exploration", Proc. IFIP 6th workshop on Protocol Specification, Testing, and Verification(1986)

[37] N.F.Maxemchuk and K.Sabani:"Probabilistic Verification of Communication Protocols", Proc. IFIP 7th workshop on Protocol Specification, Testing, and Verification(1987)

[38] D.D.Dimitrijevic:"Dynamic State Exploration in Quantitative Protocol Analysis", Proc. 9th Int. Workshop on Protocol Specification, Testing and Verification, pp.327-338 (June 1989)

[39] P.Y.M.Chu and M.T.Liu:"Procedure for Probabilistic Protocol Verification and Evaluation", IEEE Trans. on Commun., COM-40, 7, pp.1183-1191(July 1992)

[40] G.J.Holtzmann:"Practical Methods for the Formal Validation of SDL Specifications", Computer Communications, 15, 2, pp.129-134(March 1992)

[41] G.J.Holtzmann:"Algorithms for Automated Protocol Verification", AT&T Tech. J., pp.32-44(January/February 1990)

[42] G.J.Holtzmann and J.Petti:"Validating SDL Specifications: an Experiment", Proc. 9th Int. Workshop on Protocol Specification, Testing and Verification, pp.317-326 (June 1989)

[43] G.J.Holtzmann:"Design and Validation of Computer Protocols", Prentice Hall, Software Series (1991)

[44] G.J.Holtzmann:"Design and Validation of Protocols: a tutorial", Computer Networks and ISDN Systems, 25, pp.981-1017 (1993)

[45] 若原，角田："内部イベントを含む通信プロトコルの検証"，信学論(B-I)，J74-B-1, 10, pp.721-732 (1991-10)

[46] 若原，角田，乗越，小田："グローバル状態遷移一括処理によるプロトコル論理検証法の提案"，信学技報，SE83-167, pp.9-12(1984-03)

[47] 若原，角田："プロセス状態遷移一括処理によるプロトコル検証"，信学論(B)，J71-B, 12, pp.1446-1455 (1988-12)

[48] Y.Kakuda, Y.Wakahara and M.Norigoe："A new Algorithm for Fast Protocol Validation"，Proc. IEEE COMPSAC'86, pp.228-236 (October 1986)

[49] Y.Kakuda, Y.Wakahara and M.Norigoe :"An Acyclic Expansion Algorithm for Fast Protocol Validation", IEEE Trans. Soft. Eng., SE-14, 8, pp.1059-1070 (Aug.1988)

[50] 若原，新田，宇都宮，小田，斎藤："プロセス検証とグローバル検証による高速プロトコル検証"，信学論(B-I)，J81-B-I, 6, pp.378-390 (1998年6月)

[51] 当麻，内藤，南谷："順序機械"，岩波書店(1983)

[52] 古屋，玉本："フォールト・トレランス入門"，オーム社 (1988)

[53] 当麻，南谷，藤原："フォールトトレランスシステムの構成と設計"，槙書店 (1991)

[54] F.C.Hennie:"Fault Detecting Experiments for Sequential Circuits", Proc. 5th annual Symp. on Switching Theory and Logic Design, Princeton (1964)

[55] P.Goel:"Test Generation Costs Analysis and Projections", Proc. 17th Design Automation Conf., pp.77-84(1980)

[56] I.Pomeranz and S.M.Reddy:"Test Generation for Multiple State-Table Faults in Finite-State Machines", IEEE Trans. Computers, vol.46, no.7, pp.783-794(July 1997)

[57] P.G.Kovijanic:"A New Look at Test Generation and Verification", Proc. 14th Design Automation Conf., pp.58-63(1978)

[58] T.Yamada, M.Saisho and Y.Kasuya:"Test Generation Method for Highly Sequential Circuits", Proc., COMPCON, pp.104-107(1979)

[59] H.Kubo:"A procedure for Generating Test Sequences to Detect Sequential Circuit Failures", NEC Res. Dev. No.12, pp.69-78(October 1986)

[60] H.C.Liang, C.L.Lee and J.E.Chen:"Identifying Invalid States for Sequential Circuit Test Generation", IEEE Trans. Computer-Aided Design, vol.16, no.9, pp.1025-1033(September 1997)

[61] K.E.Cheng:"A Requirements Definition and Assessment Framework for SDL", Computer Networks and ISDN Systems, 28, pp.1703-1715(1996)

[62] D.Hogrefe:"Validating SDL systems", Computer networks and ISDN Systems, 28, pp.1659-1667(1996)

[63] 有沢　誠："ソフトウェアプロトタイピング"，近代科学社 (1986)

[64] S.D.Urban, J.E.Urban and W.D.Dominick:"Utilizing an Executable Specification Language for an Information System", IEEE Trans. on Soft. Eng., SE11, 7, pp.598-605 (July 1985)

[65] B.W.Boem, T.E.Gray and T.Seewaldt:"Prototyping versus Specifying - A Multiproject Experiment", IEEE Trans. on Soft. Eng., SE10, 3, pp.290-303(March 1984)

[66] L.N.Jackson, C.J.Fidge, R.S.V.Pascoe and P.H.Gerrand:"Computer-aided Program Generation from System Specifications", Proc. ISS84, 33A4.1-7(1984)

[67] 伊藤，市川："通信分野における自動プログラミング"，情報処理，28, 10, pp.1405-1411 (1987-10)

[68] E.Vefsnmo:"DASOM-An SDL Tool", Proc., SDL'87, pp.35-42, Amsterdam(1987)

[69] U.Johansen and E.Vefsnmo:"Automatic Program Generation of SDL Specifications: Principles and Solutions", Proc., SDL'87, pp.349-359, Amsterdam(1987)

[70] J.Karlsson and L.Mansson:"Using SDL as Specification and Design Language and Ada as Inmplementaion Language", Proc., SDL'87, pp.339-348, Amsterdam(1987)

[71] E.Unruh:"SCAN, an Expert System for the Analysis of SDL Specifications", Proc., SDL'87,

pp.137-146, Amsterdam(1987)

[72] L.N.Jackson, K.E.Cheng, T.S.Choong, R.S.V.Pascoe and G.J.Chain:"MELBA at the Age of Eight: An automatic Code Generation System", Proc., SDL'87, pp.371-381, Amsterdam (1987)

[73] H.Kossman:"An Integrated Set of Tools for Software Design based on SDL, Proc., SDL'87, pp.147-155, Amsterdam(1987)

[74] 小山田："SDL に基づく交換ソフトの設計支援"，電子情報通信学会，交換・通信ソフトウェア仕様記述言語の標準と技術展望専門講習会資料，pp.67-76，(1988-06)

[75] 増井："SDL 支援システムの適用"，電子情報通信学会，交換・通信ソフトウェア仕様記述言語の標準と技術展望専門講習会資料，pp.77-86，(1988-06)

[76] 山崎："知識ベースを用いた交換ソフト仕様処理"，電子情報通信学会，交換・通信ソフトウェア仕様記述言語の標準と技術展望専門講習会資料，pp.97-105，(1988-06)

[77] M.Atlevi:"SDT-The SDL Design Tool", Proc., FORTE'88, pp.55-60(1988)

[78] 宗森，田中，佐藤，勝山，水野："SDL ビジュアルウォークスルーシミュレータ" 情報処理学会ソフトウェア工学研究会資料，59-3(1988)

[79] V.Encontre:"GEODE:An Industrial Environment for Designing Real Time Distributed Systems in SDL", Proc., SDL'89, pp.105-115, Amsterdam(1989)

[80] N.Kikuchi, Y.Shigeta, K.Miyake, W.Tanaka and M.Nabeta:"An Integrated System Development Method and Support System bsed on SDL and C++", Proc., SDL'89, pp.135-144, Amsterdam(1989)

[81] 長谷川，野村："SDL と C を組み合わせた通信プログラム仕様の記述法およびその処理系"，電子情報通信学会春季全国大会，B329(1989)

[82] B.Algayres, V.Coelho, L.Doldi, H.Garavel, Y.Lejeune and C. Rodriguez:"VESAR: a Pragmatic Approach to Formal Specification and Verification", Computer Networks and ISDN Systems, 25, pp.779-790(1993)

[83] G.Gries:"SDL++-A Toolset for the Object-Oriented Development of C++ Software", Proc. of SDL'93, pp.119-128, Darmstadt(1993)

[84] B.Algayres, Y.Lejeune, F.Hugonnet and F.Hantz:"The AVALON Project: A VALidatiON Environment for SDL?MSC Descriptions", Proc. of SDL'93, pp.221-235, Darmstadt(1993)

[85] Ek Anders:"Verifying Message Sequence Charts with the SDT Validator", Proc. of SDL'93, pp.237-249, Darmstadt(1993)

[86] 市川，高見："通信ソフトウェアの自動生成技術"，電子情報通信学会誌，Vol.77, p.522 (1994)

[87] 若原，伊藤："通信ソフトウェア用プロトタイピングシステムの基本構想"，信学技報，SE85-132, pp.37-42(1985-12)

[88] Y.Wakahara and A.Ito："Prototyping System for Telecommunications Software Based on Abstract Execution of Requirements Specifications"，Proc. of SDL'87, pp.315-325, Amasterdam(April 1987)

[89] Y.Wakahara and A.Ito："Prototyping System for Telecommunications Software Based on Abstract Execution of Requirements Specifications",Computer Networks,13,2,pp.119-128 (1987)

[90] 若原，伊藤："通信ソフトウェア仕様検証のためのプロトタイピングシステム "，国際通信の研究，138，pp.26-34(1988-10)

[91] 国友　義久："効果的プログラム開発技法"，近代科学社 (1988)

[92] T.L.Booth: "Sequential Machines and Automata Theory"，John Wiley & Sons(1967)

[93] 若原，乗越，角田："通信ソフトウェア要求仕様の最適化処理"，信学技報，SE85-89, pp.25-30(1985-09)

[94] 若原　恭："ソフトウェア仕様における決定木表現の一最適化"，信学論(D)，J71-D，9, pp.1887-1890 (1988-09)

[95] 若原，伊藤，宇都宮："通信ソフトウェア仕様の簡明化処理"，情報処理学会，CASE 環境シンポジウム，pp.143-150(1989-03)

[96] Y.Wakahara, A.Ito, E.Utsunomiya, and F.Nitta："Simplification of Requirements Specification in SDL"，Proc. of SDL'89, pp.189-198 (October 1989)

[97] Y.Wakahara, A.Ito, E.Utsunomiya, and F.Nitta："Simplification to Enhance Comprehensibility of Communications Software Descriptions Written in a Procedural Language"，IEICE Trans. Commun., E75-B, 10, pp.942-948 (October 1992)

[98] 東野，森，谷口，嵩："代数的に記述された HDLC プロトコルの検証"，信学論(D)，J66-D, 7, pp.773-780 (1983-07)

[99] R.L.Schwartz P.M.Melliar-Smith: "From State Machine to Temporal Logic: Specification methods for Protocol Standards", IEEE Trans. on Commun., COM-30, 12, pp.2486-2496(Dec. 1982)

[100] M.G.Gouda:"Protocol Verification Made Simple: a Tutorial", Computer Networks and ISDN Systems, 25, pp.969-980(1993)

[101] L.D.Fosdick and L.J.Osterwei:"Data Flow Analysis in Software Reliability", Comput. Seurv., Vol.2, No.3, pp.305-330(1976)

[102] 花田，永瀬，安原，石田："データフロー解析を応用したオペレーション実行系列の検証"，信学論(D)，J64-D, 12, pp.1137-1144 (1981-12)

[103] Y.Wakahara and Y.Kakuda："Verification of Requirements Specifications for Telecommunications Software by the Investigation of Specifications Execution Sequences"，Proc. IEEE GLOBECOM'87, pp.17.6.1-17.6.6 (November 1987)

[104]若原，乗越，角田："仕様要素の実行系列解析による通信ソフトウェア要求仕様の検証，信学技報，SE84-49，pp.13-18(1984-08)

[105]若原，乗越，角田："交換機ソフトウェア要求仕様の検証支援システム"，信学会部門全国大会，S5-3，I-429-I-430(1984)

[106]若原，乗越，角田："通信ソフトウェア要求仕様検証支援システム"，信学技報，SE84-50，pp.19-24(1984-08)

[107]Y.Wakahara，Y.Kakuda，A.Ito and E.Utsunomiya："ESCORT: An environment for specifying communication requirements"，IEEE Software，6，2，pp.38-43 (March 1989)

[108]若原 恭："インテリジェントネットワーク INの最新動向ーサービス生成技術ー"，電子情報通信学会誌，78，7，pp.715-720 (1995年7月)

[109]ITU-T勧告X.25(1984)

[110]ITU-T勧告Q.700シリーズ(1984)

[111]ITU-T勧告Q.900シリーズ(1988)

[112]E.J.McCluskey："Minimization of Boolean Functions"．BSTJ.，35，6，pp.1417-1444(1956-11)