Vol. 34 No. 1 情報処理学会論文誌 Jan. 1993

形式言語理論に基づく通信制御プログラムの処理系列検証方法

山下 博之†

プログラムにおける各種資源アクセス処理は，資源の確保→情報の設定→情報の参照→資源の解放，というような一定の順序に従って実行されねばならない．形式言語理論によれば，このような実行順序規則の多くは正規文法で表すことができ，このとき同規則に基づく資源アクセス処理系列の集合は正規言語となることから，処理系列の実行順序規則性の検証は文の帰属問題に帰着できる．一方，正規言語は有限オートマトンと等価である．本論文ではこの点に着目し，通信制御プログラムを対象に，ソースプログラムのパス解析によりその通過し得る径路を網羅的に抽出し，各径路中の資源アクセス処理系列がその実行順序規則に基づく正規言語と等価なオートマトンに受理されることを確認することにより，タイマ，バッファ等の資源アクセス処理の実行順序の妥当性を検証する方法を提案する．本検証方法は次の特徴を有する：①ループ系列については，ループ部分を正規表現で表すとともに非ループ系列と合成することにより，実際のループ回数に依存しない検証が可能である．②同一資源に対するアクセス処理が複数のモジュールで実行される場合にも，モジュール間インタフェース情報の解析により検証可能である．また，上記方法に基づく検証システムの試作例についても述べる．本検証方法および検証システムの適用により，プログラムの信頼性・生産性向上が期待できる．

A Verification Method on Operation Sequences for Communication Control Programs Based on Formal Language Theory

Hiroyuki Yamashita†

Resource access operations should be executed in the following order: get→set→refer →release. Most of these rules can be specified with regular grammars, and operation sequences compose regular languages, which are equivalent to finite automata. This paper presents a method to verify consistency of operation sequences which access to resources such as timers, buffers, etc. for communication control programs. The verification method is composed of the two parts: (1) extracting all state transition paths by analyzing source programs specified with state transition tables, (2) verifying that resource access operation sequences in each path are accepted by the automaton correspond to their execution order rule. It is capable for verification of operation sequences in paths which include a loop and of inter-module operation sequences. An example of the verification system and a prospect of its effectiveness for improvement in reliability of programs are also shown.

1. はじめに

高品質のソフトウェアを短期間（低コスト）で開発する必要がますます高まっており，生産性・信頼性向上を目的にソフトウェア工学，また最近では知識工学の立場から種々のアプローチが行われている[1]．とりわけ，ソフトウェア開発コストの半分以上を試験コストが占めるという事実から，試験検証技術の重要性が指摘されている[2]．

そこでまず，試験検証対象について次のように分析した．一般に，通信制御プログラムをはじめとする多くのプログラムの処理は，資源に対するアクセスの集合とみることができる．資源に対するアクセス処理は，資源の確保→情報の設定→情報の参照→資源の解放，というような一定の順序規則に従って実行されなければならない．過去の通信制御プログラムにおけるバグの分析によれば，このような処理の実行順序規則に関するバグは全体の約3割を占めている．他にマクロ等の部品の使用誤りに起因するバグがほぼ同割合を占めるが，バグの影響として処理の実行順序が期待から外れることになる．したがって，この種のバグを早期に除去しておくことは生産性・信頼性向上に有効である．

従来，このような観点からの検証は机上解析により

† NTT 情報通信網研究所
NTT Network Information Systems Laboratories

行っていたが，特に通信制御プログラムではその制御フローの複雑さから，設計・製造ドキュメントである状態遷移図あるいは状態遷移表を人手で追うことは効率が悪く，またチェック漏れも生じやすい．一方，実マシン上あるいはシミュレータ上での動的試験も実施している．ところが，実マシン上での試験では環境設定が困難な項目があること等から，プログラムの全径路を網羅する試験を行うことができない．また，シミュレータ上の試験でも試験データ量が膨大となり全径路を網羅する試験は困難である．このような背景から，近年対象プログラムを実際に動作させることなくその制御フロー等を解析する静的試験を機械的に行う静的解析技術が注目され，各所で研究されている．

静的解析は，形式的に記述したプログラムあるいは仕様自体の無矛盾性を論理的に調べるアプローチと，形式的に記述した仕様とプログラムとの整合性を調べるアプローチとに大別できる．

前者のアプローチとしては，プログラムあるいは仕様を有限状態機械（Finite State Machine；FSM）等により表現し，状態到達可能性やデッドロックフリー性等の検証を行う例が数多く報告されている[3]．プロトコル検証[4),5)]も仕様検証の一種と考えられる．同様の考え方によるプログラム合成あるいはプロトコル合成の報告例もある[5]．また，形式的記述からの構造テスト用試験データ生成に関する報告例も多い[6),7)]．しかし，本アプローチには次の問題がある：①プログラム規模の増大に伴いグローバル状態が爆発的に増大し，実用的でなくなる．②いずれも，動作すなわち資源アクセス処理の実行順序の妥当性の検証については触れていない．③特定の記述言語を前提としており適用範囲が限定されていたり[3]，仕様を検証用の言語記述に変換する必要があり効率が悪かったりする[4]．④仕様検証については，たとえ仕様の正しさが証明されても，その仕様に基づき作成されたプログラムが正しいとは言えない．これはプログラムが仕様から自動合成される場合にも言え，合成方法の正当性も示されなければならない．

後者のアプローチとして，システム設計仕様をFSMで記述し，プログラムの解析に基づき生成した試験データを与えて受理するか調べる方法[8]，プログラム中にコメントの形で記述したオペレーション実行順序規則を用いてプログラムのコンパイル時に検証する方法[9]が提案されている．しかし，プログラム開発に用いるシステムの動作（機能）仕様と，複数機能に含まれる資源アクセス処理の実行順序規則（いわば構造仕様に該当）とは，一般に一致しないため，文献8)の方法を資源アクセス処理の実行順序の妥当性の検証には適用できない．また，文献9)の方法は，実行順序規則としてFSMにより記述可能なクラスのみを扱っており，制約のより緩いクラスへの適用性に関する考慮がなされていない．

本論文では，通信制御プログラムにおける各資源のアクセス処理の実行順序の妥当性に関し，**形式言語理論**（formal language theory）に基づき，次の点に着目した上記後者のアプローチに属する検証方法を提案する．まず，プログラムの径路は，ソースプログラムに対してパス解析技術[10]を適用することにより網羅的に抽出することが可能である．ここで，資源ごとに定められたそのアクセス処理の実行順序規則（≠プログラム設計仕様）の多くは**正規文法**（regular grammar）で表すことができ，同規則に従う処理系列の集合は**正規言語**（regular language）を構成する．ところが，正規言語と**有限オートマトン**（finite automata）とは等価である[11]．したがって，処理系列の検証問題は，形式言語理論における文の帰属問題，すなわち処理系列の実行順序規則に基づく正規言語と等価なオートマトンへの受理問題に帰着することができる．以上より，本論文で提案する検証方法は次の手順で行う：

① ソースプログラムのパス解析により，その径路を網羅的に抽出する，

② 各径路において実行される処理をアクセスする資源ごとに集め，それぞれ処理系列とする，

③ 仕様とプログラムとの整合性，すなわち，各処理系列が資源ごとの実行順序規則に対応するオートマトンに受理されることを確認する．

本検証方法は，形式言語理論に基づくため，実行順序規則を同理論における文法のクラスに対応させて拡張可能であり，広範囲に適用できる．

また，上記検証方法に基づく検証システムを試作した．試作システムは，通信制御プログラムにおけるほとんどの資源アクセス処理に対して適用可能である．

本論文では，まず，第2章で，処理系列の検証問題を定義するとともに，対象とする通信制御プログラムの記述規約に対する要求条件と実行順序規則のクラスについて述べる．その後，第3章で，上記手順に従った検証方法を詳説する．第4章では，試作した検証システムの概要について説明し，その適用効果の見通しを述べる．さらに，第5章では，実行順序規則のクラ

スの拡張について考察する.

2. 検証問題の分析

2.1 通信制御プログラムの処理系列

通信制御プログラムは通常プロトコル・レイヤに対応して階層化されており，1つのプロトコル・レイヤにおける送信や受信等の各機能を実現する部分を，本論文では“モジュール”と呼ぶ.

本論文で対象とする通信制御プログラムはイベント駆動型であり，このようなプログラムの規定要素は，状態，トリガ，および状態遷移に伴う処理である．1回の状態遷移に伴う処理の単位を“ルーチン”と呼ぶ[12]．次々に入力するトリガによってプログラムは状態遷移するが，この遷移する状態の順序付けられた列をプログラムの“径路”という.“処理系列”とはプログラムの径路に対応するルーチンの列のことである．また，ルーチンはそれぞれ独立した資源に対するアクセスの集合である．以降では，1つの資源に対するアクセスを“操作”と呼ぶ．操作は通信制御処理の最小単位である．検証対象となる処理系列とは，具体的にはプログラムの径路中のある資源に対する操作の列である．なお，後述のように，トリガをも操作に含めて扱う場合がある.

以下では，プログラムの径路と処理系列について例を用いて説明する.

通信制御プログラムの基本動作は，高位レイヤからの指示あるいは低位レイヤからの非同期報告等に基づいて所定の処理を行った後にその結果を報告することであり，その処理は状態遷移制御により実現される．遷移条件（トリガ）となるものは，高位レイヤからの指示，低位レイヤからの報告，タイムアウト等である．通信制御プログラムの簡単な動作例を**図1**に示す．同図には，対象レイヤ n において，状態 s1 で高位レイヤ $n+1$ からの指示を受けて4回の状態遷移を伴う処理を行った後に処理結果を報告して再び状態 s1 に至るまでの径路を示してある．図1の例では，状態 s1→s2→s3→s4→s1 の順に遷移を行う径路において，4個のルーチンが実行される．なお，同図ではタイマおよびモジュール間インタフェースに関する操作のみを記してある．同図に示すプログラムの径路において，タイマに関する処理系列を抽出すれば，次のようになる：

タイマ始動　タイマ停止　タイマ始動　タイマ停止

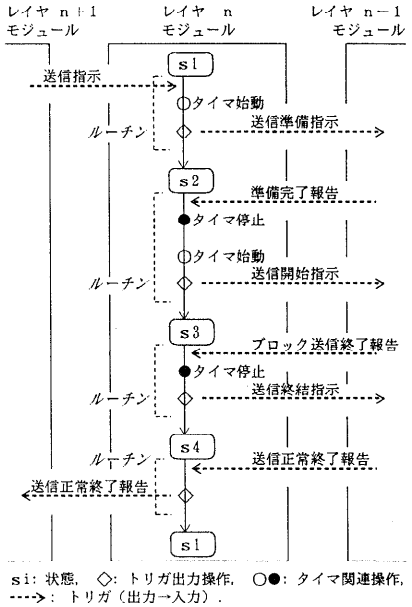


図1 通信制御プログラムの動作例
Fig. 1 Execution flow example of communication control program.

2.2 処理の実行順序規則

通信制御プログラムの各操作は，①資源確保，②情報設定，③情報参照，④資源解放，の4種に分類可能である．そして一般に，一連の処理において各操作は，①→②→③→④というような一定の順序で実行されねばならない．この分類と順序は資源により異なり，たとえばタイマに関しては，タイマ始動が②に，タイマ停止およびタイマアウトが③に，それぞれ分類され，タイマを始動した後，それを停止するかまたはタイムアウトが発生するかの順で実行されねばならない．図1の例では，タイマ始動→タイマ停止→タイマ始動→タイマ停止の順で実行されることから，上記順序に合致する．このような操作の実行されるべき順序を，本論文では処理の“実行順序規則”と呼ぶ.

各資源に対する操作の分類として，我々の経験により得られた，レイヤ1およびレイヤ2の通信制御における例を**表1**に示す[13),14)]．同表における“変数”とは，プロトコル規定に従って設定および参照が繰り返されるものであり，“フラグ”は変数と同様であるが，参照が条件判断に用いられる点が異なる．また，“カウンタ”も変数と同様であるが，設定は初期設定（0クリア）と1加算，参照は限界値との比較という特殊なものであることから，ここでは分けて挙げた．なお，

表 1　通信制御における資源対応の実行順序規則
Table 1　Execution order rules for each resource in communication control.

No.	資源	例[13),14)]	操作／トリガ		種別	実行順序規則	分類
1	タイマ	・肯定応答タイマ ・Pビットタイマ ・REJ タイマ	a	始動	設定	{a(b+c)}*	A
			b	停止	参照		
			c	タイムアウト[注1]	参照		
2	バッファ	・送信データバッファ ・受信データバッファ	a	確保	確保	(ab)*	A
			b	解放	解放		
3	変数／フラグ	・送信状態変数 ・DATA フラグ	a	設定	設定	(ab*)* [注2]	B
			b	参照	参照		
4	カウンタ	・再試行カウンタ	a	初期設定	確保	{a(bc)*}*	D
			b	増減	設定		
			c	チェック	参照		
5	高位モジュール	・NW レイヤエンティティ	a	指示受付け[注1]	設定	(ab)*	A
			b	報告出力	参照	($ab_1^*b_2$)* [注3]	C
6	低位モジュール	・MAC レイヤテンティティ	a	指示出力	設定	(ab)*	A
			b	報告受付け[注1]	参照	($ab_1^*b_2$)* [注3]	C

[注1]　入力トリガである.
[注2]　実行前に初期設定されている場合，b(*ab*)* となる.
[注3]　b_1, b_2 はbの操作／トリガに属する個々の異なる操作／トリガを表す.

初期化時／ネゴシェーション時に設定し以降参照のみのもの，必要時点で他資源の値に基づき生成・設定するものは，挙げていない．同表に示すうちの多くは他レイヤにおいてもあてはまると考える．また，通信プロトコルのレイヤに対応したモジュール構成を有し，高位モジュールからの指示を受け付け，所定の処理を実行した後その結果を報告するようなモジュール間インタフェースを採用する通信制御プログラムの場合には，指示出力／受付け，報告出力／受付けをアクセス操作と考えることにより，高位／低位モジュールをも資源とみなすことができる．したがって，表1には，高位／低位モジュールをも含めて示してある．なお，同表には，各資源対応の実行順序規則をも示しているが，その表記は後述の形式的な方法に従っている．

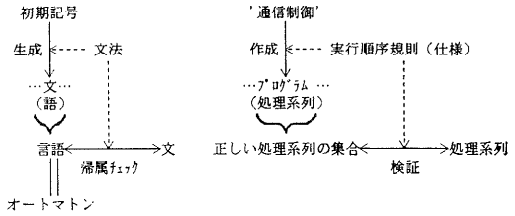
2.3　処理系列の検証問題

プログラムの各径路について資源ごとの処理系列が定められた実行順序規則に従って実行されることを確認することを，本論文では“処理系列の検証”という．

ところで，プログラムの作成は形式言語[11)]における文の生成と対応付けることが可能である．すなわち，“**初期記号**”（たとえば‘通信制御’）から始めて，仕様に対応する実行順序規則（“**書換え規則（生成規則）**”あるいは“**文法**”）を用いて段階的に詳細化（“**書換え**”）することにより，記述の最小単位である操作（“**記号**”）から成るプログラム（“**文**”あるいは“**語**”）を作成する．ここで，資源ごとのアクセスのみに着目すれば，プログラムは処理系列ということになる．逆に，プログラムの検証は，作成されたプログラム（文）が実行順序規則（文法）に従っていることを確認することである．また，文法により生成される文の集合は“**言語**”と呼ばれ，言語に対応するのは実行順序規則に基づき作成される正しい処理系列の集合ということになる．したがって，処理系列の検証問題は，特定の文がある言語に属するか否かを調べる文の認識問題あるいは帰属問題に帰着できる（**図2**参照）．

形式言語理論によれば，言語はその文法における制限の程度により4つのクラスに分類されるが，各クラスの言語はそれぞれ対応するクラスのオートマトンと等価であることが知られている．すなわち，文の認識とは，対象言語と等価なオートマトンにより受理されることを確認することにほかならない．

以上より，処理系列の検証は次の2段階により行うことができる．まず，ソースプログラムを解析しその



(a) 文の帰属問題 (b) 処理系列の検証問題
(a) Sentence membership problem. (b) Operation sequences verification problem.

図 2 処理系列の検証問題と文の帰属問題との対応
Fig. 2 Correspondence of operation sequences verification problem with sentence membership problem.

径路を網羅的に抽出する．次に，各径路における資源ごとの処理系列がその実行順序規則に基づく言語と等価なオートマトンに受理されることを確認する．

2.4 前　　提

本節では，処理系列の抽出および検証の効率化の観点から，対象通信制御プログラムの記述規約に対する要求条件を述べる．なお，ソースプログラムのパス解析に基づく検証であるため，逐次型の手続き言語により記述されているとする．また，対象とする実行順序規則(文法)，すなわち言語のクラスについて述べる．

(1) プログラムの記述規約に対する要求条件

本論文で対象とする通信制御プログラムは，次の規約に従って記述されているものとする：

(a) モジュールは独立した複数の機能より成り，資源へのアクセスが各機能内で閉じている．

(b) 1ルーチンは1セグメントから成る．なお，“セグメント”とは，一般に，分岐および合流なしに連続して実行されるステートメントのかたまりをいう．

(c) 各操作に対応するプログラム要素において，当該操作の有無が決定的であるものとする．ここでいう“決定的”なプログラム構造とは，ソースプログラム上で操作ありの径路と操作なしの径路とが異なることを意味する．

(d) イベント，操作および操作対象が，プログラム要素と1:1に対応する．なお，“プログラム要素”とは，状態，トリガ，ステートメント，マクロ，サブルーチン等のようにソースプログラムを構成する機能単位である．

上記要求条件(a)は，プログラムの径路抽出における確実な抽出範囲と実用的な径路数のための条件である．複数機能単位(部分集合)で処理を共有する場合のように，いわゆる“**連結性**”のある通信制御プログラムに対しては，以降の径路の抽出を打ち切る“**カット**”により対処可能である．もし実際の動作上はあり得ない径路が検証の結果チェックアウトされた場合，当該径路の詳細なレビューにより問題がなければ無効径路として検証対象から除外すればよい．

ルーチン内に分岐および合流が存在する場合には，対象通信制御プログラムを拡張し，分岐点および合流点を状態と，分岐条件をトリガと，それぞれみなすことにより容易に対処可能である．このようにして得られる拡張された通信制御プログラムは，抽出径路の網羅性に関する上記要求条件(b)を満たす．

検証の厳密性に関する上記要求条件(c)に関し，資源に対するアクセス操作の有無が実行時に決定する径路の場合には，たとえば，操作ありおよびなしの両ケースについて検証後，詳細な机上レビューを行う．

要求条件(d)は，処理系列の抽出および検証のEDP化に関するものであり，各種条件とプログラム要素との対応付けが重要となる．すなわち，径路抽出範囲については，特定の状態における特定の入力トリガで始まり特定の命令／マクロの実行で終わる範囲，というようにプログラム要素で記述できなければならない．また，分岐や合流がある場合には，上述の拡張を行うために，分岐点，分岐条件，および合流点とプログラム要素との対応が明示されている必要がある．資源アクセス操作のインプリメント方法としては，後述のようにマクロ等のプログラム要素と一対一に対応する場合のほかに，複数の資源に対するアクセス操作が1つの“複合マクロ”となっている場合，複数の命令により1つの資源に対するアクセス操作がなされる場合，等，種々考えられる．このような場合にも，複合マクロについては，そのマクロを複数の資源アクセス操作と対応付ければよい．また，複数命令については，対象資源アクセス操作を構成命令列で置き換えた実行順序規則表現とすればよい．このようにして検証後，机上レビューにより詳細なチェックを行う．

以上より，本項で述べた要求条件は，本論文で示す検証方法の一般性を失うものではない．なお，他の要求条件として，ループがないこと，同一資源に対するアクセスが1モジュールに閉じていること，が挙げられるが，このようなプログラムは非現実的であるため，次章で示すように検証方法の工夫により対処する．

(2) 実行順序規則のクラス

通信制御プログラムにおける資源アクセス操作の実行順序規則の多くは，経験的に，最も制限の強い文法のクラスである“**正規文法**”で表すことができ，正規文法により生成される言語は“**正規言語**”と呼ばれる[11]．また，正規言語と等価なオートマトンは，“**有限オートマトン**”と呼ばれる．本論文における検証では，この正規言語のクラスを対象とする．なお，5章でクラスの拡張について考察する．

正規言語の形式的表現方法として，一般に，“**正規表現** (regular expression)”が用いられる．資源アクセス操作の実行順序規則についても，正規表現を用いて簡明に表すことができるので，本論文では，以降，正規表現で表すこととする．資源アクセス処理の一般的な実行順序規則を正規表現で表すと，次式のようになる．(〈 〉内が1記号を表す．)

$$\left\{\left\langle\begin{matrix}\text{資源}\\ \text{確保}\end{matrix}\right\rangle\left(\left\langle\begin{matrix}\text{情報}\\ \text{設定}\end{matrix}\right\rangle\left\langle\begin{matrix}\text{情報}\\ \text{参照}\end{matrix}\right\rangle^{*}\right)^{*}\left\langle\begin{matrix}\text{資源}\\ \text{解放}\end{matrix}\right\rangle\right\}^{*} \qquad \text{【式1】}$$

【式1】は個々の資源に応じて簡単化することができ，表1には正規表現で表した各資源ごとの実行順序規則を示してある．以上より，上記前提を言い換えれば，処理の実行順序規則が正規表現で表される場合を対象とするということになる．

2.5 検証上の問題点

プログラムのすべての径路を抽出する際に最も問題となるのは径路数である．プログラム規模（通信制御プログラムの場合には状態数およびトリガ数）の増大に伴い径路数は膨大化し，たちまち実用の範囲を越えてしまう．これを避けるためには，真に検証対象とすべき径路のみを抽出する必要がある．

また，ソースプログラムを参照する静的解析においてはループの扱いが問題となる．すなわち，ループ回数は実行時でないと不明である．一般に，ループ回数はプログラムに入力されるデータに依存する．このため，ループ回数の最大・最小に対応するデータを想定して検証する方法が採られることもあるが，検証の網羅性および効率の観点からは，ループ回数に依存しない検証方法が望ましい．

もう1つの問題として，同一資源に対するアクセス処理が複数のモジュールで実行される場合には，その処理内容が検証対象モジュールのみの解析では不明であることが多く，検証は非常に複雑となることが挙げられる．とりわけ，ハードウェアでの実行が絡むと致命的である．検証効率向上のために検証のための解析範囲を極力小さくしなければならない．

次章では上記問題点の解決方法を示すとともに，具体的検証方法について述べる．

3. 検証方法

3.1 プログラム径路の抽出

プログラムの径路の抽出にはパス解析技術を適用する．本論文で対象とするイベント駆動型の通信制御プログラムにおいては，各状態で全トリガに対する遷移先状態を得るということを次々と繰り返すことにより“状態遷移系列”を得る．これを本論文では“状態遷移パス解析”と称する．

ここで，プログラムあるいはモジュール全体を対象に状態遷移パス解析を行うと，状態・トリガ数の増大に伴い抽出される系列数が膨大となり非実用的である．この問題を解決するために次の2方法が考えられる．

[方法1] 各状態での遷移先を絞り，径路を限定する．

[方法2] 状態遷移パス解析を行う範囲を細かくする．

[方法1]は，実際の通信制御プログラムにおいては処理上意味のある状態遷移の数は状態ごとに限定されるという性質を利用するものである．しかし，状態ごとに有効な遷移を行うトリガを指定する必要があること，および径路の網羅性が低下することからこの方法は好ましくない．

[方法2]はモジュールがうまく分割できれば有効な方法である．2.4節(1)のプログラムの記述規約に対する要求条件(a)から，モジュールは送信，受信等の独立した機能より成るため，これらの機能を単位として状態遷移パス解析を行えば，抽出される系列数を実用的な範囲に抑えることが可能である．したがって，[方法2]が適当である．

ここで，通信制御プログラムにおいて上記の独立した各機能は，一般に，送信関連機能に関しては高位レイヤからの指示により，受信関連機能に関しては低位レイヤからの非同期報告により，それぞれ起動される．いずれに関しても，所定の処理終了後，高位レイヤに報告を行った後に終了する．

以上の考察から，具体的な状態遷移系列の抽出に対し次の基準（開始条件および終了条件）を設ける：

【基準1】 高位レイヤからの各機能の指示イベントを受け(開始条件)，所定の処理を行った後そ

表 2 状態遷移パス解析条件
Table 2 The state-transition path analysis conditions.

状態遷移パス解析条件＼抽出基準		基準 1［注1］	基準 2
開始条件	開始状態	指示受付け可能状態 (s1)	指示を実行中でない状態
	開始トリガ	指示 (t12)	非同期事象の発生報告
終了条件		処理結果報告の出力操作 (#POST (SOK), #POST (SNG))	非同期事象発生報告の出力操作

［注1］（ ）内は図3におけるプログラム要素の例を表す.

の結果を報告する（終了条件）まで.

【基準2】 低位レイヤからの非同期事象の発生報告イベントを受け(開始条件)，所定の処理を行った後その報告を行う（終了条件）まで.

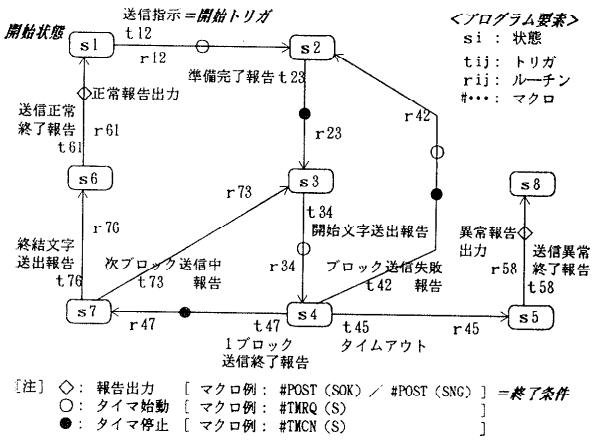
なお，高位レイヤに対し報告を行わない場合も考えられる．この場合には，待ち状態への遷移等，報告に代わる条件を状態遷移系列の抽出基準（終了条件）とすればよい.

上記基準において，開始条件は，開始状態とイベントに対応する開始トリガとから成る．状態遷移パス解析において指定する事項（状態遷移パス解析条件）は上記各基準に対して**表2**に示すようになる.

プログラムの記述規約に対する要求条件(b)および(c)から，状態遷移パス解析により，プログラムの径路を網羅的に抽出可能である．また，要求条件(d)から，状態遷移パス解析条件として，対応するプログラム要素による指定が可能である.

簡単な状態遷移図と状態遷移系列の抽出例を**図3**に示す．同図では各状態で一部のトリガに対する状態遷移しか示していないが，実際には状態遷移表の探索により全トリガを対象に状態遷移パス解析を行う．また，簡単のためタイマに関する操作のみを示してある．抽出される状態遷移系列は状態の系列であるが，これをルーチンすなわち資源アクセス操作の集合の系列である“処理系列”に置き換える．たとえば，図中の状態遷移系列 **s1 s2 s3 s4 s5 s8** (NO 2) は処理系列 **r12 r23 r34 r45 r58** に一意に置き換えられる.

なお，既に通過した状態に遷移するループを検出した場合には，その時点で当該系列を打ち切る．これを“ループ系列”と称し，図の NO 1 および NO 3 の系列が該当する．ループ系列は上記の抽出基準を満たさないが，この問題については次節で解決する.



(a) 状態遷移図の例
(a) Example of state-transition diagram.

NO	状態遷移系列	処理系列	種別
1	s1s2s3s4s2	r12r23r34r42	ループ
2	s1s2s3s4s5s8	r12r23r34r45r58	非ループ
3	s1s2s3s4s7s3	r12r23r34r47r73	ループ
4	s1s2s3s4s7s6s1	r12r23r34r47r76r61	非ループ

(b) 抽出される処理系列
(b) Extracted operation sequences.

図 3 状態遷移系列の抽出例
Fig. 3 Example for extraction of state-transition sequences.

3.2 ループ系列の扱い

ループ系列については次の問題がある.

(1) その処理が通信制御機能として完結していないため，資源アクセスが系列内で最後まで実行されないことがある.

(2) ループ回数が検証時点では不明である.

まず，(2)については，ループ部分を正規表現で表すことによりループ回数に依存しない検証が可能となる.

次に，(1)については，ループ系列と“対応する”非ループ系列とを合成することにより，ルー

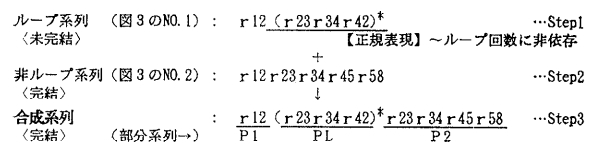


図 4　ループ系列に対する合成の例
Fig. 4　Example for compounding of loop and non-loop sequences.

プを包含する合成系列として検証する．対応する非ループ系列とは，ループに至るまでの径路が等しくループから抜け出して完結する系列のことである．

ループ系列の合成の例を**図 4** に示す．同図は図 3 の No. 1 のループ系列の合成を示しているが，これを用いてその手順を説明すると次のようになる．

Step 1　ループ系列のループ部分 [(r 23 r 34 r 42)*] を正規表現で表す．

Step 2　対応する非ループ系列として，ループに至るまでの系列 [r 12] が等しい非ループ系列 [No. 2] を捜す．

Step 3　ループに至るまでの系列[r 12]とそれ以降の系列[r 23 r 34 r 45 r 56]との間に，Step 1 で得た正規表現で表したループ部分を挿入する．

なお，合成の対象となった非ループ系列はループ回数が 0 回の場合として合成系列に包含されるので，検証対象から除外する．

以上のことから，検証対象の処理系列の一般形を次式のように表すことができる：

P1PL*P2　　【式 2】

上式で，P 1 はループに至るまでのループを含まない部分系列，PL* はループ部分の系列，P 2 はループ以降のループを含まない部分系列をそれぞれ表す．このような形で表すことにより，3. 4 節で述べるように，部分系列に分解した検証が可能となる．

ここで，多重ループの内側のループに対しては対応する非ループ系列が存在しない場合がある．この場合には，外側のループを開放し【式 2 】のような一般形を作って検証すればよい．すなわち，多重ループ系列は P 1 (PL 1 PLL* PL 2)*P 2 と表されるが，外側のループを開放した系列 P 1 PL 1 PLL* PL 2 P 2 を検証する．なぜなら，本節前半までの手続きに基づき外側のループのみの系列 P 1 (PL 1 PL 2)*P 2 も網羅的に抽出されて検証されるため，両系列の比較により，内側のループ系列 PLL 中の資源アクセス操作列は系列 PL 2 PL 1 中のそれと等価（両系列検証の際の部分オートマトンに共に受理されること）あるいは空ということになる．したがって，これらの上記多重ループ系列への代入により明らかに同系列が検証されることが言える．

また，複数のループを含む径路については，特定の状態を境として対象径路をそれぞれただ 1 個のループを含む部分径路に分割し，各部分径路に対する実行順序規則を設定することにより対処可能である．このような分割は，境界の状態が明確であり，かつ前後の実行順序規則を定式化できるという条件で可能である．

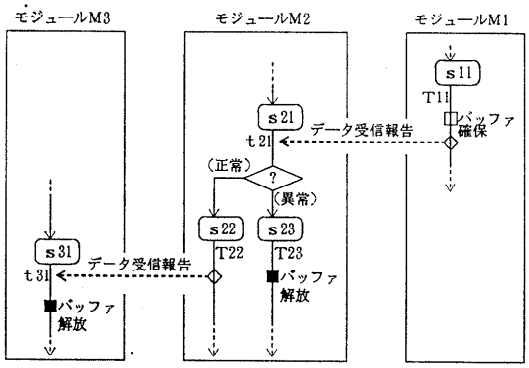
3.3　モジュール間にわたる処理の扱い

同一の資源に対するアクセスが複数（2 個）のモジュールで実行されることがある．これには次の 2 つのケースが考えられる．

〈ケース 1〉 異なるレイヤの機能を実現する 2 モジュールが，同一の資源にアクセスする場合．このケースの資源としては，送受信用データ・バッファがある（**図 5**）．

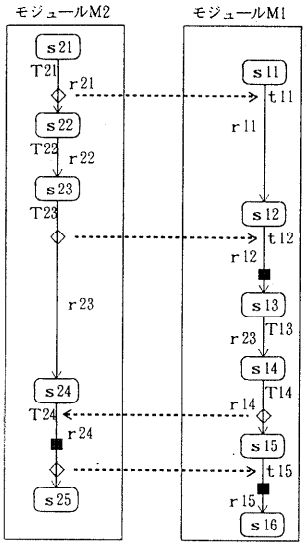
〈ケース 2〉 同一レイヤの異なる機能を実現する 2 モジュールが，相互に情報を交換しながら**コルーチン** (coroutine) 的に連携動作する場合．この例としては，全二重制御における送信および受信機能を実現する 2 モジュールがある（**図 6**）．

状態遷移パス解析はモジュール単位で行うため，抽出される系列はモジュール内に閉じている．したがって，上記のような場合には 1 モジュールの系列のみを対象に検証しても無意味である．この問題を解決するために，各ケースに応じて次に示す 2 つの方法を



[注] sij : 状態，
Tij, tij : トリガ（tijはｲﾝﾀﾌｪｰｽ情報に基づくトリガを表す），
◇ : インタフェース情報の出力操作．

図 5　モジュール間にわたる処理の例
〈ケース 1：複数モジュールで同一資源にアクセス〉
Fig. 5　Example for inter-module processing.
〈case 1: access to the same resource.〉



[注] sij: 状態, rij: ルーチン, Tij, tij: トリガ（tijは相互インタフェース情報に基づくトリガを表す）, ◇ ----> : 相互インタフェース情報の出力, ■ : 資源アクセス操作.

図 6 モジュール間にわたる処理の例
〈ケース２：コルーチン的な連携動作〉
Fig. 6 Example for inter-module processing.
〈case 2: co-operative execution.〉

採る.

（1） ケース１における問題の解決方法

この場合のモジュール間インタフェース情報は指示および報告である. 図５において, モジュール M２におけるモジュール M１ からのデータ受信報告（入力トリガ）は, M１でバッファ確保が実行されたことを暗に示している. また, モジュール M２ におけるモジュール M３ へのデータ受信報告処理は, M３ でバッファ解放が実行されることを暗に期待している. すなわち, 他モジュールにおける資源アクセス処理結果は検証対象モジュールにおいてはトリガとして入力され, また, 検証対象モジュールから出力される報告（あるいは指示）を受けて他モジュールが所定の資源アクセス処理を行う.

したがって, 入力トリガおよび報告（あるいは指示）出力処理を対応する資源へのアクセス処理に置き換えて扱うことにより, １モジュールの系列に閉じて検証を行うことが可能となる. すなわち, 図５の例では, モジュール M２ を対象とする検証において, 入力トリガ **t**21 をバッファ確保に, モジュール M３ への入力トリガ **t**31 の出力操作をバッファ解放に, それぞれ置き換えて検証する. なお, 本方法は, 他モジュールがハードウェアにより実現されている場合にも適用可能である.

（2） ケース２における問題の解決方法

ケース２は両モジュールの状態遷移パス解析が可能な場合である. 図６において, モジュール M１ の出力するインタフェース情報はモジュール M２ では入力トリガとなる. すなわち, モジュール M１ におけるインタフェース情報出力処理とモジュール M２ における入力トリガとは１対１に対応する. 同様にその逆も対応する.

したがって, 両モジュールの全出力系列について相互インタフェース情報出力処理と入力トリガとの対応を調べることにより, 連携動作する系列の組を得ることができる. この系列の組におけるルーチンを実行順に並べることにより得られる系列を検証対象とすることにより, 連携動作する両モジュールで同一の資源にアクセスする場合の検証が可能となる. すなわち, 図６の例では, 系列 **r**21 **r**11 **r**22 **r**23 **r**12 **r**13 **r**14 **r**24 **r**15 が検証対象となる.

なお, 対応する系列が存在しない場合には, 処理順序誤りの可能性があるので, 机上レビューを行う. また, 連携動作する系列中にループが存在しその対応がつかない場合には, ループ回数を固定(たとえば１回)して検証を行った後, さらに机上レビューを行う.

3.4 検証手順と例

表１の実行順序規則はＡ～Ｄの４種の基本型に分類される. 実行順序規則の基本型を, 以降, 本論文では“検証条件”と称する. 各実行順序規則（検証条件）を検証対称系列の一般形【式２】と対応づけると, **表３** に示すようになる. この対応づけにおいて, 実行順序規則における操作／トリガを表す記号 (a, b, c) から検証条件の基本型における記号 (α, β, γ) への対応表を得る. 表３は, 検証条件の基本型ごとに処理系列の部分系列 P１, PL, P２ が満たすべき条件の組合せを示すものである. それらの組合せは P１ の条件に応じ複数存在するが, いずれか１つを満たせばよい. また, 部分系列の満たすべき条件も正規表現で表される. したがって, 検証は３個の部分系列ごとに行えばよい. 部分系列はループを含まない単純な系列である. ３個の部分系列のすべてが各条件に対応する正規言語と等価なオートマトンに受理されれば, その系列の対象資源アクセス操作の実行順序に矛盾はない.

以上まとめると, 処理系列の検証は次の手順で行う.

表 3　部分系列に対する検証条件
Table 3　Verification conditions for each part of the general operation sequence form.

分類	検証条件の型	一般形における部分系列ごとの検証条件［注1］			
		No.	P1	PL	P2
A	$(\alpha\beta)^*$	1	$(\alpha\beta)^*$	$(\alpha\beta)^*$	$(\alpha\beta)^*$
		2	$(\alpha\beta)^*\alpha$	$(\beta\alpha)^*$	$\beta(\alpha\beta)^*$
B	$(\alpha\beta^*)^*$	1	$(\alpha\beta^*)^*$	$(\alpha\beta^*)^*$	$(\alpha\beta^*)^*$
		2	$(\alpha\beta^*)^*\alpha\beta^*$	$\{(\beta^*\alpha)^*\beta^*\}^*$	$\beta^*(\alpha\beta^*)^*$
C	$(\alpha\beta^*\gamma)^*$	1	$(\alpha\beta^*\gamma)^*$	$(\alpha\beta^*\gamma)^*$	$(\alpha\beta^*\gamma)^*$
		2	$(\alpha\beta^*\gamma)^*\alpha\beta^*$	$\{(\beta^*\gamma\alpha)^*\beta^*\}^*$	$\beta^*\gamma(\alpha\beta^*\gamma)^*$
D	$\{\alpha(\beta\gamma)^*\}^*$	1	$\{\alpha(\beta\gamma)^*\}^*$	$\{\alpha(\beta\gamma)^*\}^*$	$\{\alpha(\beta\gamma)^*\}^*$
		2	$\{\alpha(\beta\gamma)^*\}^*\alpha(\beta\gamma)^*$	$[\{(\beta\gamma)^*\alpha\}^*(\beta\gamma)^*]^*$	$(\beta\gamma)^*\{\alpha(\beta\gamma)^*\}^*$
		3	$\{\alpha(\beta\gamma)^*\}^*\alpha(\beta\gamma)^*\beta$	$[\gamma\{(\beta\gamma)^*\alpha\}^*(\beta\gamma)^*\beta]^*$	$\gamma(\beta\gamma)^*\{\alpha(\beta\gamma)^*\}^*$

［注1］ 検証条件の各型について，いずれかの No. の組合せを満たせばよい．

《Step 1》 状態遷移パス解析条件を指定して状態遷移パス解析を行い，プログラムの実行径路を網羅的に抽出する．また，必要に応じ2モジュールの実行径路群の中から対応する径路の組を抽出する．

《Step 2》 抽出した各径路をルーチンの系列に変換する．

《Step 3》 ループ系列を対応する非ループ系列と合成し，各系列を一般形【式2】で表す．

《Step 4》 各系列中のルーチンに検証対象の資源に関する操作，および資源によってはルーチンを起動する入力トリガ，報告／指示出力操作があれば，表1の対応する記号 (a, b, c) で置き換え，なければ削除することにより操作系列を得る．この置き換えは，2.4節のプログラムの記述規約に対する要求条件(d)から一意である．

《Step 5》 各操作系列を基本型の分類に応じて表3の記号列 (α, β, γ を使用) に置換する．

《Step 6》 各記号列の部分系列が表3の対応するオートマトンに受理されることを確認する．

タイマ操作に関する検証例を**図7**に示す．検証対象は図4でループの合成を行った系列であり，以下では図7に基づいて上記手順の《Step 4》以降を説明する．

まず，タイマ操作のあるルーチン（図3参照）について，表1に基づき，あるタイマの始動［#TMRQ (S)］があれば記号 a で，同一タイマの停止［#TMCN (S)］があれば記号 b で，同一タイマのタイムアウトをトリガに起動されるルーチンがあれば記号 c で，対象系列をそれぞれ置き換える．同一タイマの操作のないルーチンは消去し，複数あるルーチンは実行順に複数の記号で置き換える．《Step 4》次に，タイマ操作は分類Aの基本型に属し，a と α，b または c と β とが対応する．よって，記号 a を記号 α で，記号 b および記号 c を記号 β で置き換える．《Step 5》以上により部分系列 P1＝α，PL＝$\beta\alpha\beta\alpha$，P2＝$\beta\alpha\beta$ が得られるので，表3の分類Aの検証条件 No. 1 および No. 2 の各オートマトンに対して，部分系列 P1，PL，P2 がそれぞれ受理されるかを調べる．本例では，No. 2 のオートマトン［P1＝$(\alpha\beta)^*\alpha$, PL＝$(\beta\alpha)^*$, P2＝$\beta(\alpha\beta)^*$］に受理される．《Step 6》

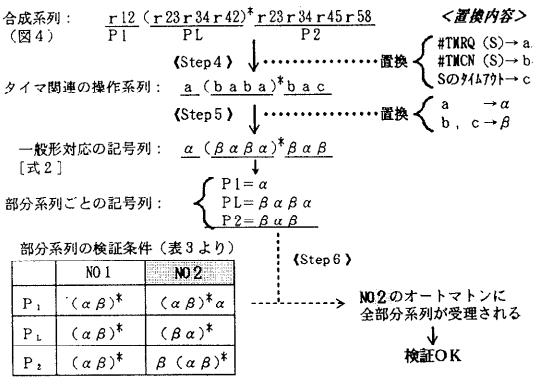


図 7　処理系列の検証例
Fig. 7　Example for the verification flow of operation sequences.

4. 検証システムの例

CCP (Communication Control Processor) 上で走行する通信制御プログラムを対象とする検証システムを試作し，その機能および性能について確認した．対象通信制御プログラム[12]は，状態遷移表，ルーチン等の設計情報がデータベース (DB) 化され，そのDBを利用した開発支援システム (ITEM) のもとで開発されており[15]，検証システムはこのITEMの一環として実現した．検証システムの記述言語はSYSL (SYStem discription Language) であり，プログラム規模は約16 kstepである．

ITEMにおいては，DB中の設計情報からソースプログラム・ファイルを自動生成するため，設計情報とソースプログラムとは等価である．対象通信制御プログラムは2.4節(1)に示したプログラムの記述規約に対する要求条件(a)～(d)を満たす．すなわち，各モジュールは送信処理，受信処理等の機能ブロックに分割でき，各ブロック間の状態遷移は基本的にない[(a)]．機能ブロックの一連の動作は，通常，高位モジュールからの指示あるいは低位モジュールからの非同期報告により始まり，高位モジュールへの処理結果の報告により終了する．また，ルーチンは，その内部で分岐および合流はなく[(b)]，各資源アクセス操作に対応する“マクロ”列により構成される[(d)]．なお，資源アクセスの有無を判定する処理と実際に資源にアクセスする処理とは異なるマクロとして明確に分離しており，前者の処理を行うマクロの実行結果として，「操作あり」および「操作なし」の2個のイベントを発生する[(c)]．

4.1 システムの機能

検証システムは，状態遷移パス解析，モジュール間インタフェース解析，系列検証，系列編集出力の4機能により構成される．

検証システムに対して状態遷移パス解析条件(表2)を教える際には，プログラム要素と初期状態，開始トリガ，終了条件との対応情報を外部から入力すればよい．また，資源アクセス操作を教える際には，プログラム中にコメント等の特別な記述を行う必要はなく，マクロと資源アクセス操作との対応情報を外部から入力すればよい．これらの対応情報はプログラム仕様にほかならない．

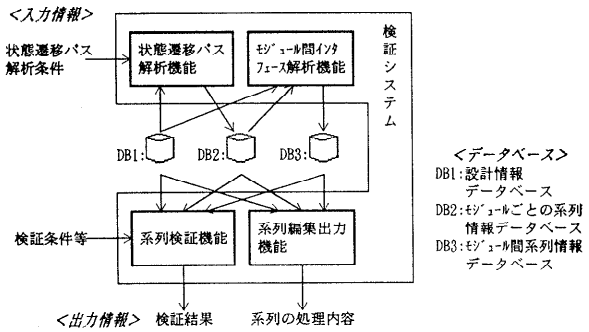


図8 検証システムの構成概要
Fig. 8 Configuration of the verification system.

(1) 状態遷移パス解析

指定する状態遷移パス解析条件に基づき設計情報DB中の状態遷移表を探索し，モジュールごとにプログラムの径路を網羅的に抽出し系列情報DBに格納する．

(2) モジュール間インタフェース解析

モジュールごとの系列情報DB中の系列群からモジュール間で相互インタフェース情報が対応する系列の組を抽出し，モジュール間系列情報DBに格納する．

(3) 系列検証

モジュールごとおよびモジュール間の系列情報DB中の系列群について，ループ系列と対応する非ループ系列との合成を行った後，指定の検証条件に基づいて検証を行う．

(4) 系列編集出力

開始状態および入力トリガを指定することにより，設計情報DBを検索して処理内容の系列を編集出力する．特定の資源アクセス操作のみを選択出力することが可能である．本機能は系列検証の結果エラーとなった系列の詳細な解析等に使用する．

検証システムの構成概要を**図8**に示す．

4.2 システムの適用効果

本システムの適用可能性を評価するため，既開発済の通信制御プログラムにおいて試験工程以降に摘出されたバグを分析した．その結果を**図9**に示す．

同図に示すように，全摘出バグのうちの約1/3が処理系列関連のバグであり，そのうちの半分が本システムにより摘出可能であることが分かった．本システムにより摘出不能なバグの主なものは，マクロを構成する機械語命令に関するものであり，本システムは適用できないものの検証方法そのものを否定するものではない．また，処理系列関連以外のバグのうちの約半分

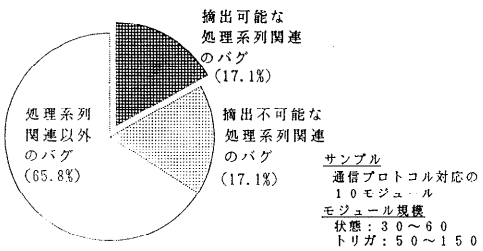


図 9　本システムによるバグ摘出可能性
Fig. 9　Estimation of error detection capability in the verification system.

を，状態遷移先の誤りあるいはルーチン選択の誤りが占める．このようなバグを含む処理系列は，資源アクセス処理の実行順序規則を満足しないことが多く，本システムの適用により間接的に摘出可能であると考える．したがって，本システムの適用により，従来試験工程以降に摘出されていたバグのうちの2～5割が同工程以前に摘出可能と考えられ，品質の早期安定化を図ることができる．

一方，プログラムの信頼性評価基準として**試験網羅率** (test coverage) がよく用いられる．通信制御プログラムの場合，通常のプログラムにおける分岐には状態遷移が相当すると考えられる．よって，各状態遷移に伴う処理（ルーチン）がセグメントに対応する．試験網羅率としては，プログラム中の全セグメントに対する試験における実行セグメントの割合である C1 網羅率，およびプログラム中の全径路（パス）に対する試験における実行径路の割合である C∞ 網羅率等が知られている[16]．このうち，C∞ 網羅率については計算不可能であるため，C1 網羅率について，現在実施している実マシン上での試験と比較する．

我々が使用する試験プログラムにおいては，実マシン上での試験における C1 網羅率は 40～80% であることが測定されている．これに対し，本検証方法においては網羅的に抽出したプログラムの実行径路を対象とするため，C1 網羅率は 100% と言える．すなわち，本システムの適用により，実マシン上での試験においては摘出できない潜在バグの摘出が可能となる．

以上より，本システムは通信制御プログラムの信頼性・生産性向上に効果があると判断できる．

5. 考　　察

これまでに述べた検証方法は，2.4節に示したように，実行順序規則として最も制限の強い正規文法のクラスを対象とするものである．ところが，このクラスには属さない実行順序規則が存在するため，本章ではクラスの拡張について考察する．

ここでは，対応する2種の資源アクセス操作である，確保および解放の実行回数が一致しているという条件を考える．この具体例としてはバッファ資源に対するアクセスがあり，確保した数のバッファを解放しなければならない．この場合の実行順序規則（検証条件）の一例は次のようになり，正規表現では表し得ない．

確保 解放＋確保 確保 解放 解放
＋確保 確保 確保 解放 解放 解放＋…【式3】

本問題に関する解決方法として，検証対象通信制御プログラムで扱う資源が有限であるという前提で，上記例のようにとにかく検証条件を表現する方法がある．この方法は，操作回数が小さい場合には本検証方法をそのまま適用できるため有効であるが，大きい場合には検証システムのオートマトン定義域がオーバフローする可能性があり，本質的な解決にはならない．

ところで，上記例の実行順序規則は，正規文法より1つ上のクラスの文法に対応する．以下では，上記問題を解決するための検証方法の拡張について述べる．

まず，1回以上の繰返しを表すために繰返し数を右肩につけ，確保〝解放〟のように表現する．このように表された規則（文法）は“**文脈自由文法** (context free grammar)”，これにより生成される言語は“**文脈自由言語** (context free language)”と呼ばれるクラスに属し，正規文法に比べ制約が緩いが，正規言語よりレベルが高い．換言すれば，正規言語は文脈自由文法で生成されるが，文脈自由言語は正規文法では生成されない．したがって，ある文が正規言語に帰属しても文脈自由言語に帰属するとは限らない．上記例では，実行順序規則を，確保*解放*というように無理やり正規表現で表し，検証対象処理系列がその有限オートマトンに受理されたとしても，それが正しいとするには不十分である．文脈自由言語と等価なオートマトンに受理されることを確認する必要がある．

文脈自由言語と等価なオートマトンは，“**プッシュダウン・オートマトン** (push-down automata)”と呼ばれる[11]．この検証は，各資源アクセス操作を記憶するスタックを導入し，対応する操作間でその出現回数の一致比較を行う等により可能となる．このようにして検証方法を拡張することにより，より広範囲の実行順序規則に適用可能となる．

ただし，実行順序規則については上記のように拡張可能となるが，検証対象であるプログラムの資源アクセス操作系列については，プログラム・ループ内で実行される場合のように操作回数が実行時に決定する場合には，記号*を用いて正規言語のクラスとして表現せざるを得ない．このとき，実行順序規則も正規表現とする必要がある．すなわち，静的検証においては，上記のようなケースに対しては，必要条件の検証のみが可能であり十分条件の検証はできず，本検証方法の適用後さらに詳細なレビューが必要となる．

6. おわりに

通信制御プログラムにおける資源アクセス処理の実行順序の妥当性を検証する方法について述べた．本検証方法は形式言語理論に基づいており，ソースプログラムのパス解析により網羅的に抽出したプログラムの径路における各処理系列が，資源ごとの処理の実行順序規則に対応する正規文法により生成される正規言語と等価なオートマトンに受理されることを確認するというものである．また，本方法を用いた検証システムを試作し，その有効性についての見通しを得た．本検証方法およびシステムの適用により，早期にバグを摘出可能となるとともに，実マシン上の試験では摘出できない潜在バグの摘出も可能となり，通信制御プログラムの信頼性および生産性の向上が期待できる．今後，同検証システムを実際に適用しその効果を評価したい．

また，本論文で述べた検証方法は，プログラムばかりでなく通信プロトコル等の仕様における処理系列の検証にも適用可能であることは容易に想像されよう．さらには，インテリジェントネットワーク（IN）において検討されている，ユーザカスタマイズド・サービス記述の検証にも適用可能であると考える．

最近，プログラムあるいは仕様の部品化が普及してきている．部品化により個々の部品そのものの信頼性は向上するが，部品の接続時における整合性の検証が重要な問題となる．整合条件の1つである資源アクセス処理の実行順序の妥当性の検証に関しては，本論文で述べた方法が有効であると考える．

なお，残された課題として，ループおよび複数モジュール処理に関する径路抽出の網羅性の確認あるいは向上，および実行順序規則のクラスの拡張等が挙げられる．

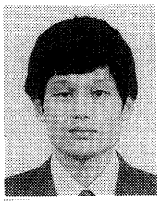
謝辞 本研究を進める上で有益なご討論をいただいた情報通信網研究所山下正秀主幹研究員ならびに同大林恵次主幹研究員に深謝します．また，日頃ご指導いただく同基本アーキテクチャ研究部和佐野哲男部長に感謝します．

参考文献

1) 宮本 勲：ソフトウェア・エンジニアリング—現状と展望—，p. 353, TBS 出版，東京 (1982).
2) Myers, G. J.: *The Art of Software Testing*, John Wiley & Sons, New York (1979). (松尾正信（訳）：ソフトウェアテストの技法，p. 192, 近代科学社，東京 (1980).)
3) 内平直志：様相論理による並行プログラムの積重ね式検証法，電子情報通信学会論文誌 D-I, Vol. J 75-D-I, No. 2, pp. 76-87 (1992).
4) 邵 峰晶，白川 理，関 浩之，藤井 護，嵩 忠雄：順序機械によってモデル化された通信プロトコルの一検証法—OSI セションプロトコルを例にして—，電子情報通信学会論文誌 D-I, Vol. J 74-D-I, No. 12, pp. 846-857 (1991).
5) 角田良明，若原 恭，乗越雅光：状態遷移系列の対応付けに基づいたプロトコル検証とプロトコル合成，昭和60年度電子通信学会情報・システム部門全国大会論文集，S 13-1 (1985).
6) 岡崎直宣，高橋 薫，白鳥則郎，野口正一：LOTOS 仕様からの効率的な試験系列の生成法，電子情報通信学会論文誌 B-I, Vol. J 74-B-I, No. 10, pp. 733-747 (1991).
7) Borzovs, J., Kalnins, A. and Medvedis, I.: *Automatic Construction of Test Sets: Practical Approach, Lecture Notes in Computer Science*, Vol. 502, pp. 360-432, Springer-Verlag (1991).
8) 田中誠一郎，宮崎義弘，手島文彰，井上勝博，三原幸博：自動検証システム ATVS の拡張，第40回情報処理学会全国大会論文集，2R-6 (1990).
9) 花田収悦，永瀬淳夫，安原隆一，石田 亨：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，電子通信学会論文誌 D, Vol. J 64-D, No. 12, pp. 1137-1144 (1981).
10) 田中善一郎，石井 茂：手工業的手法からの脱皮を目指すソフトウェア・テスト，日経エレクトロニクス，1982. 3. 15号，pp. 124-152 (1982).
11) 福村晃夫，稲垣康善：オートマトン・形式言語理論と計算機，岩波講座，情報科学-6, p. 235, 岩波書店，東京 (1982).
12) 木村正男，大林恵次，上森 明，山下博之：通信制御プログラムにおける部品化プログラミング方式，情報処理学会論文誌，Vol. 25, No. 6, pp. 1064-1071 (1984).
13) JIS ハンドブック情報処理データ通信(編)：JIS X 5105 ハイレベルデータリンク制御手順の手順要素，p. 872, 日本規格協会，東京 (1991).
14) JIS ハンドブック情報処理データ通信(編)：JIS

X 5251 ローカルエリアネットワーク論理リンク制御, p. 872, 日本規格協会, 東京 (1991).
15) 山下博之, 大林恵次, 山田俊文, 菊池康夫：設計情報データベース利用の通信制御プログラム開発支援システム (ITEM), 情報処理学会ソフトウェア工学研究会資料, 34-2 (1984).
16) 岸田孝一：ソフトウェア・テスティング, bit 誌, Vol. 13, No. 4, 6〜10, 共立出版, 東京 (1981).

(平成4年1月14日受付)
(平成4年11月12日採録)



山下 博之 (正会員)

1957年生. 1979年京都大学工学部情報工学科卒業. 1981年同大学院工学研究科修士課程(情報工学専攻)修了. 同年日本電信電話公社横須賀電気通信研究所入所. 以来, DIPS計算機の通信系ファームウェアおよびその設計・製造技術, DIPS通信制御処理装置の研究実用化, フォールトトレラント技術の研究等に従事. 現在, NTT情報通信網研究所主任研究員. 電子情報通信学会, 応用物理学会各会員.