マルチメディア通信と分散処理 33－4
（１９８７．５．２１）

ユーザフレンドリィなプロトコル検証システムの構成

岡崎直宣、 相沢茂喜、 高橋薫、 白鳥則郎、 野口正一
（東北大学電気通信研究所）

情報通信システムの開発を効率的に推進するためには、設計されたプロトコルの仕様の論理的矛盾や曖昧さを検出し、それを設計者に適切に指摘するプロトコル検証システムの開発が必須の条件となっている。本論文では、ユーザフレンドリィ性を重視したプロトコル検証システムを与える。本検証システムでは、プロトコルは仕様記述言語NESDELを用いて仕様化され、その検証は従来のパータベーション解析に基づいた検証法EXPAを一般化したEXPA IIを用いて行われる。検証の結果として、デッドロックなどのプロトコル上の論理誤りは、検証用表現から対応する元のNESDEL表現上でユーザに提示される。従って、本検証システムでは、従来の検証システムとは異なり、ユーザは理解性に乏しい検証用表現や検証結果の表現を知る必要はなく、理解性に優れた仕様記述表現だけを用いてプロトコルを記述し、検証することが可能となっている。その結果、検証システムのユーザインタフェースを向上させ、プロトコル開発における生産性を向上させる。

A User-Friendly Protocol Verification System

Naonobu OKAZAKI, Shigeki AIZAWA, Kaoru TAKAHASHI, Norio SHIRATORI, Shoichi NOGUCHI
Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Sendai, 980 Japan

This paper presents a user-friendly protocol verification system and its applications. In this system, a protocol is specified by using the protocol specification language NESDEL, and it is verified by EXPA II which is an extension of the protocol verification method EXPA based on the traditional perturbation analysis. By the provision of the bidirectional transformations between NESDEL expressions and EXPA II expressions, this system can directly show logical errors in the protocol such as deadlocks on the original NESDEL expression. The user needs not to know the detailed expressions for verification and its results. Thus, it is sufficient for a user to know only the specification language, consequently, the interface between the user and the verification system can be highly improved.

1. まえがき

情報通信システムの大規模化、通信サービスの拡大に伴い、プロトコルや通信関係のソフトウェアも複雑化･大規模化し、その開発コストも増大する傾向にある。この問題を解決する一つの方策は、(1)プロトコルの仕様化、(2)プロトコルの検証、(3)プロトコルの自動インプリメンテーション、(4)適合性の試験、等を総合的に支援する支援環境を構築し、ユーザに提供することである。筆者らは、従来より、上述した支援を行うためのシステムとして、IDESSと呼ばれる支援システムを開発してきている。

本論文では、IDESSの中で、プロトコルの検証をユーザフレンドリィな形で支援する機能を遂行するシステム、即ち、ユーザフレンドリィ･プロトコル検証システムについて述べる。

近年、プロトコルの検証を機械的に行うシステムが開発されており[1],[2],[3]、これらは仕様記述言語で表現されたプロトコルを検証用の表現に変換し、その上で検証を行い、その結果を検証用表現の形でユーザに提示するものとなっている。これらのシステムでは検証の結果を元の仕様記述言語上に直接反映させる機能を有していない。そのため、検証用表現で与えられたプロトコルの論理誤りを、ユーザ自身で元の仕様記述言語上で確認し、訂正するという作業が必要となる。またこれらのシステムはその検証法として、従来のパータベーション解析[4]に基づいており、論理誤りの検出は可能であるが、誤りの原因などを自動検出することは可能となっていない。従って、ユーザ自身でそれを調べることが必要となる。

本論文では、上述の難点を解決するため、(1)プロトコル検証の結果を直接に仕様記述言語上の表現に反映し、(2)プロトコルの論理誤りのみならず、その原因を同時に検出しユーザに提示する、ユーザフレンドリィ性を重視したプロトコル検証システムを構成する。本システムでは、プロトコル仕様記述言語としてNESDEL[5]、検証法として、従来のパータベーション解析に基づいた検証法EXPA[6]を一般化したEXPAIIを開発し用いている。つまり、NESDELで表現されたプロトコルは、EXPAIIの表現に変換され、検証される。検証結果は、EXPAIIの表現から逆変換され、対応する元のNESDEL表現上でユーザに提示される。

以下本論文では、まずプロトコルと通信ソフトウェアの総合的な開発支援システムであるIDESSについて概観する。そして次に、本論文の目的である、IDESS中の構成要素としてのユーザフレンドリィ･プロトコル検証システムについて述べる。

2. IDESS： プロトコルと通信ソフトウェアの開発支援システム

IDESSの目的は、プロトコルや通信ソフトウェアの開発を容易にする環境をユーザに提供することである。図1に示すように、IDESSは9つの構成要素から成る。

(1) NESDEL[5]： プロトコルの仕様記述を目的とした言語である。プロトコル規定の理解性に重点を置いて開発した言語となっており、そのため、有向グラフ表現を主体とする。

(2) IDL[10]： 有限状態機械の概念を反映した通信ソフトウェア向きのプログラミング言語である。

(3) ESTELLE[7]： ISO標準の言語ESTELLEを上記のIDLと同じ位置づけでとらえ、通信ソフトウェアのもう1つの記述形式としてIDESSの中に取り込んでいる。

(4) 知的NESDELエディタ： NESDEL表現に対する編集や検証の機能をユーザに提供する。

(5) NESDEL-to-IDL変換 ： NESDELで与えられたプロトコル仕様からそれに対応する通信ソフトウェアのIDL表現を生成する。

(6) NESDEL-to-ESTELLE変換：NESDELで与えられたプロトコル仕様からそれに対応する通信ソフトウェアのESTELLE表現を生成する。

(7) IDLコンパイラ ： IDLプログラムを既存の言語(C、アセンブラ)に翻訳する。

(8) ESTELLEコンパイラ：ESTELLE記述を既存の言語(C、アセンブラ)に翻訳する。

(9) プロトコル検証 ： NESDELで表現されたプロトコル仕様の検証を行う。

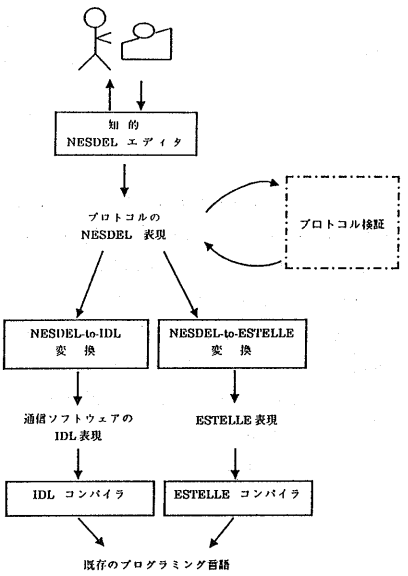

図1 IDESSの全体構成

3. プロトコル検証システムの構成

本システムの目的は、プロトコルの検証機能をユーザフレンドリィな形でユーザに提供することである。そのため、プロトコルの仕様化には理解性に重点を置いて開発された仕様記述言語NESDEL[5]を採用する。また、プロトコル検証は従来のパータベーション解析[4]に基づいた検証法EXPA[6]を一般化したEXPAIIを開発し、これを用いる。本システムでは、プロトコルの論理誤りのみならず、EXPA及びEXPAII検証の特長である"逆パータベーション"によって、システム状態の自動分類、デッドロックなどの異常なシステム状態を引き起こす原因となるシステム状態の自動検出を行い、プロトコル設計における誤りなどを適切にユーザに提示する。

本システムは次の3つの構成要素から成る(図2参照)。

(1) NESDEL-EXPAII変換 : NESDELで表現されたプロトコルを検証用表現であるEXPAII表現形式に変換する。

(2) EXPAII検証 : 変換されたEXPAII表現を入力とし、デッドロックなどの論理誤りや、その原因について検証を実行する。そして検証結果を出力する。

(3) EXPAII-NESDEL変換 : 検証の結果を検証用のEXPAII表現から元の仕様記述用のNESDEL表現に変換する。

上記(3)によって、ユーザはプロトコル検証に関して仕様記述言語だけを知っていれば十分であり、検証法やその表現形式などの詳細を知る必要はない。その結果、本システムは、従来のシステム[1],[2],[3]に比べ、ユーザフレンドリィ性が高いものとなっている。また、EXPAIIが誤りだけでなく、その原因となるシステム状態も検出可能であることから、その提示によって、プロトコルの訂正においてユーザに強力な支援を与える。

以下では、本システムで用いるプロトコル検証法EXPAII、プロトコル仕様記述言語NESDELについて述べ、そして、プロトコル検証用表現を得るためのNESDEL-EXPAII変換、検証結果を元のNESDEL表現に反映させるためのEXPAII-NESDEL変換について述べる。最後に本検証システムの適用例を与える。

4. プロトコル検証法 EXPAII

EXPAIIは、筆者らが開発したプロトコル検証法EXPAの入力に関する制限を緩和し、一般化した検証法である。EXPAの詳細については文献(6)を参照されたい。ここでは、EXPAの一般化した点を述べる。

プロトコルモデルに関する諸定義は、次に述べる一般化した点を除いて、すべて文献(6)に従うものとする。プロトコルはその機能を実行する論理的な機能モジュールである

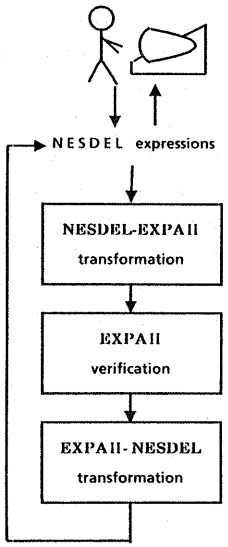

図2 プロトコル検証システムの構成

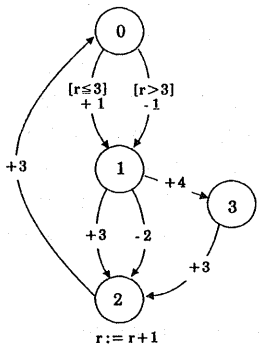

図3 変数付状態遷移表現

"プロセス"によって規定される。EXPAではプロセスの表現として変数を許さない状態遷移表現を考えている。本論文で与えるEXPAIIでは、図3に示すような変数およびそれに関する述語、処理を許した状態遷移表現(これを"変数付状態遷移表現"と呼ぶ)を入力として許すようにEXPAを一般化している。例えば、図3において、プロセスの状態0から1へのアーク上のラベル"[r>3]-1"は、述語'r>3'が真であるとき、メッセージ1を送信して、状態0から状態1へ遷移可能であることを意味する。また各ノードは、図3の状態2の'r:=r+1'のように、述語の判定にかかわる変数の処理を含んでも良い。このようなプロセスの変数付状態遷移表現のことを"EXPAII表現"と呼ぶ。EXPAIIはEXPAII表現を入力として受け付ける。従って、EXPAIIではプロセスの表

現力が豊かになり、EXPAの場合よりプロセスの記述が簡単になるという利点がある。その結果、後述するNESDEL-EXPAII変換とEXPAII-NESDEL変換のアルゴリズムの設計が比較的容易となる。

EXPAIIの出力は、システム状態の到達可能性グラフで与えられる。ここで、システム状態とは、プロセスの状態とチャネルの状態から成る合成状態であり、プロセス群とチャネル群から構成されるシステム全体の状態を表す。到達可能性グラフは、プロトコルで規定された動作に従って構成されるシステム状態の系列を網羅的に表したものである。

到達可能性グラフの分析によって、EXPAIIでは次の論理誤りを自動検出できる。

(1) デッドロック状態
(2) 受信不能状態
(3) チャネル・オーバフロー
(4) 非実行可能遷移

また、ユーザあるいはプロトコル設計者が誤りの原因を分析するのに有益となる情報として、

(a) 臨界状態
(b) 臨界状態からの誤りシーケンス
(c) ループを構成するシステム状態のシーケンス

を、与えることが可能である。ここで、臨界状態とは、正常状態でかつ不適状態への遷移を持つシステム状態である。正常状態とは、初期状態から到達可能でかつ最終状態へ到達可能なシステム状態、不適状態とは、初期状態から到達可能で、かつどの最終状態にも到達可能でないシステム状態である。

5. プロトコル仕様記述言語 NESDEL

NESDELは理解性と厳密性の観点に基づいて、筆者らが開発したプロトコル仕様記述言語である。ここでは、NESDELの設計思想と、簡単なプロトコルのNESDELによる記述例を与えることによってNESDELの概要を示す。NESDELの仕様は、"有向グラフ部"と"プリミティブ部"から成る。有向グラフ部は、プロトコルの論理的な制御の流れをグラフ的に表現し、プリミティブ部は有向グラフ部の中で使用される変数、メッセージ、オペレーション等の詳細を定義する。NESDELでは曖昧さを避けるために、プリミティブ表現を導入している。プリミティブ表現は一種の高級プログラミング言語的表現である。NESDELの特長は、有向グラフ表現とプリミティブ表現を統合することによって、プロトコルを厳密に規定し、かつ規定内容の理解性を高めることにある。

図4にNESDELによる記述例を示す。これは3回迄再送を許す簡単なプロトコルを記述した例である。NESDELの有効性は、実際のプロトコル、例えばX.25、HDLC等の記述を通して確認されている。NESDELの詳細については文献(5)を参照されたい。

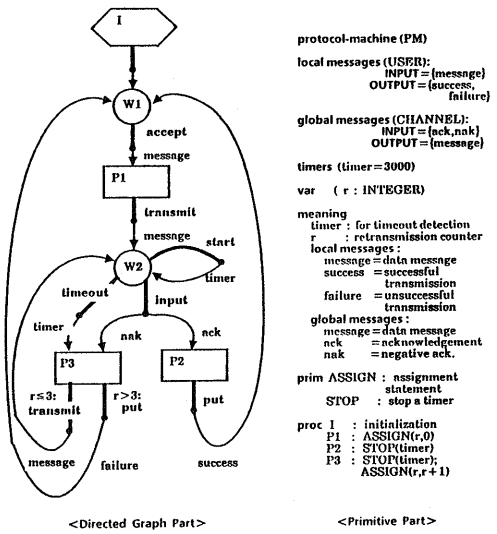

図4 NESDELによる仕様記述例

本論文では、NESDELで記述されたプロトコルの有向グラフ部を"NESDEL表現"と呼ぶ。この時、NESDEL表現は、各枝が[述語、オペレーション、メッセージ]の形のラベルを持つ有向グラフG=<V,E>として定義される。ここで、VとEはそれぞれノードと枝の集合を表し、オペレーションとは、accept(上位層からの入力)、put(上位層への出力)、input(相手からのメッセージの受信)、transmit(相手へのメッセージの送信)、start(タイマーの起動)、timeout(タイマーの時間切れ)、のうちのどれかまたは空である。

6. NESDEL-EXPAII 変換

本章では、プロトコルの仕様記述表現であるNESDEL表現を検証用のEXPAII表現に変換するアルゴリズムを与える。

パータベーション解析に基づいた検証法[2),(3),(4),(6),(9)]では、その入力として変数を許さない状態遷移表現を要求する。しかしながら、NESDEL表現のみならず、ESTELLE[7]やSDL[8]などにおけるように、プロトコル仕様としての状態遷移表現は、一般に変数付きである。変数付状態遷移表現を変数のない表現に変換する場合、変数の値による状態の分類などを考えねばならず、そのアルゴリズムの設計は容易ではなく、従来人手によって行われていた。

本論文では、EXPAを変数付状態遷移表現も受け入れられるよう一般化したEXPAIIを開発することによってこの問題を解決し、NESDEL表現とEXPAII表現との間の変換アルゴリズムを設計する。従って、ユーザは仕様記述表現を検証用表現に変換する手間から解放され、ユーザインタフェースが向上する。以下では、この変換アルゴリズムを与える。

【NESDEL-EXPAII 変換アルゴリズム】

G=<V,E>をNESDEL表現とする。このアルゴリズムはGを入力とし、EXPAII表現G'=<V',E'>を出力する。

ステップ1 (初期設定)

V'←V、E'←∅(空集合)とする。(注意：NESDEL表現のノードは代入や演算などの変数の処理を持って良い。このような変数の処理は、V'←Vの実行においてノードと一緒にV'に移される)

ステップ2 (E内の枝の処理)

Eから枝eを取り出し、eのラベルに従って次のいづれかを実行し、ステップ3へ行く。

① もしラベルが[述語,transmit,message]ならば、それを[述語,負整数]に変える。E'←E'∪{e}とする。

② もしラベルが[述語,input,message]ならば、それを[述語,正整数]に変える。E'←E'∪{e}とする。

③ もしラベルのオペレーション部が"start"ならば、この枝はE'には加えない。

④ もしラベルのオペレーション部が"accept"、"put"、"timeout"、"空"のいづれかならば、このラベルの述語部だけをeのラベルとし、E'←E'∪{e}とする。

ステップ3 (終了判定)

もしE=∅ならば停止し、さもなければステップ2へ行く。 □

NESDEL-EXPAII 変換アルゴリズムにおいて、入力であるNESDEL表現Gとアルゴリズムの出力であるEXPAII表現G'の違いは上位層との通信およびタイマーに関する枝とラベルにだけある。これらの違いは、EXPAIIが検出可能な論理エラーとは独立であることが容易に分かる。即ち、次の命題が成立する。

[命題] n個のプロセスから成る通信システムを考える。G_1, ---, G_nを各プロセスのNESDEL表現とし、変換アルゴリズムによって得られるEXPAII表現をG_1', ---, G_n'とする。そのとき、G_1', ---, G_n'上で論理エラーが存在するならばG_1, ---, G_n上でも同じ論理エラーが存在し、またその逆も言える。 □

この命題より、NESDEL表現のプロトコルを検証するにはEXPAII表現を検証すれば良いことが分かる。

7. EXPAII - NESDEL変換

パータベーション解析に基づいた従来の検証システムでは、その検証結果をシステム状態の到達可能性グラフ等で与えていた。このような表現は、仕様記述言語表現に比べ理解性に劣るため、ユーザが分析するのは容易でない。従って、ユーザインタフェースを向上させるには、検証結果を仕様記述言語、即ち、NESDEL上で得ることが望ましい。

本章では、前述のような到達可能性グラフの形で与えられた検証結果を分析し、検証用のEXPAII表現上で対応する箇所を見つけ、さらに、それを仕様記述言語上に変換するためのEXPAII - NESDEL変換アルゴリズムを与える。この変換アルゴリズムの設計も、NESDEL-EXPAII変換アルゴリズムと同様、EXPAをEXPAIIへ一般化したことにより容易になっている。

EXPAII - NESDEL変換アルゴリズムは2つのステージから成る。1つは"分解ステージ"であり、EXPAIIの検証によって得られるシステム状態やそれらの遷移列を、図3のようなEXPAII表現で記述された各プロセスの状態の遷移列に分解する。もう1つは"変換ステージ"であり、これは分解ステージの出力を入力とし、EXPAII表現に基づいたプロセスの状態遷移列を対応するNESDEL表現上の状態遷移列に変換する。以下でこの変換アルゴリズムを与える。ここでNを通信システムにおけるプロセスの数とする。

【EXPAII - NESDEL変換アルゴリズム】

ステップ1 (分解ステージ)

このステージは入力により次の2つに分けられる。

<1.1：システム状態の分解> 入力としてシステム状態が与えられたならば、EXPAII表現上で各プロセスに対応する状態を見つける。

<1.2：システム状態列の分解> システム状態の列、$S_1 \rightarrow \cdots \rightarrow S_k$が与えられたとする。その時、以下の実行で$A_i$にはプロセスi($1 \le i \le N$)のEXPAII表現上での状態の遷移列が入る。

① q_iをS_1の中でプロセスiに対応する状態とする。その時、すべてのiに対し、$A_i \leftarrow q_i$。

② j=1からk-1まで次を繰り返す。もし$S_j \rightarrow S_{j+1}$がプロセスiによって引き起こされたものであれば、$A_i \leftarrow A_i \cdot (q'_j)$。ここで、$q'_j$は$S_{j+1}$の中でプロセスiに対応する状態であり、"・"は記号列の連結を意味する。

(注意：もし$A_i = q \cdot (q_1 j_1) \cdot (q_2 j_2) \cdot \cdots \cdot (q_m j_m)$ならば、これは、$S_{j_1} \rightarrow S_{j_1+1}$はプロセスiの$q \rightarrow q_1$なる状態遷移によって引き起こされたものであり、$S_{j_2} \rightarrow S_{j_2+1}$はプロセスiの$q_1 \rightarrow q_2$によって引き起こされたものであり、等を意味する)

ステップ2 (変換ステージ)

このステージは分解ステージの出力を入力としてとる。以下のアルゴリズムで、$A_i=q\cdot(q_1,j_1)\cdot(q_2,j_2)\cdot\cdots\cdot(q_m,j_m)$は入力であり、$B_i$には対応するNESDEL表現の状態遷移列が入る。下記の①、②をすべてのi($1\leq i\leq N$)に対し実行する。

① プロセスiのNESDEL表現上でqに対応する状態を見つけ、それをB_iにセットする。

② $J=j_1,j_2,\cdots j_m$ に対し、次を実行する。プロセスiのNESDEL表現上でq_Jに対応する状態uを見つける。そして、$B_i\leftarrow B_i\cdot(u,J)$とする。 □

システム状態は、EXPAII表現における各プロセスの状態およびチャネルの状態を合成したものなので、ステップ1の分解ステージは確かに、システム状態を個々のプロセスのEXPAII表現上での成分に分解する。また、前章のNESDEL-EXPAII変換から、"start"をラベルとして持つ枝を除けば、NESDEL表現とEXPAII表現はそのノードと枝の接続関係が同じであることが分かる。従って、NESDEL表現とEXPAII表現の間で互いに対応する状態及び状態遷移を探すことは機械的にできる。以上より、ステップ2の変換ステージも正確にNESDEL表現上の対応する状態を見つけることができる。

8. 適用例

上述したEXPAII検証法、NESDEL-EXPAII変換、EXPAII-NESDEL変換から成るプロトコル検証システム(図2)を実際に東芝AS3000(UNIX)の上に実現した。本システムは、(1)検証対象となるNESDEL表現、(2)EXPAIIによる検証結果(到達可能性グラフ)、(3)検証結果の元のNESDEL表現上への反映、をマルチウィンドゥ環境の下でユーザに提示する。

次に、本システムの実際の適用例を与える。ここで使用するプロトコルの例(図5)は、デッドロック、受信不能、非実行可能遷移の論理誤りを含むものである。以下ではこの例を通して、本システムがプロトコルの論理誤りおよびその原因を、ユーザにいかに提示するかを述べる。図5はシステムへの入力であるNESDEL表現を示す。システムはNESDEL-EXPAII変換を用いて、それらをEXPAII表現に変換し、検証を行う。

図6は、5つのウィンドゥから構成される検証結果の表示である。左上のウィンドゥは検証結果の概要を表示している。この例では、エラーシーケンスが3個あり、そのうち、2個がデッドロック、残りが受信不能であることを示している。エラーシーケンス1のデッドロックに対応する番号とし

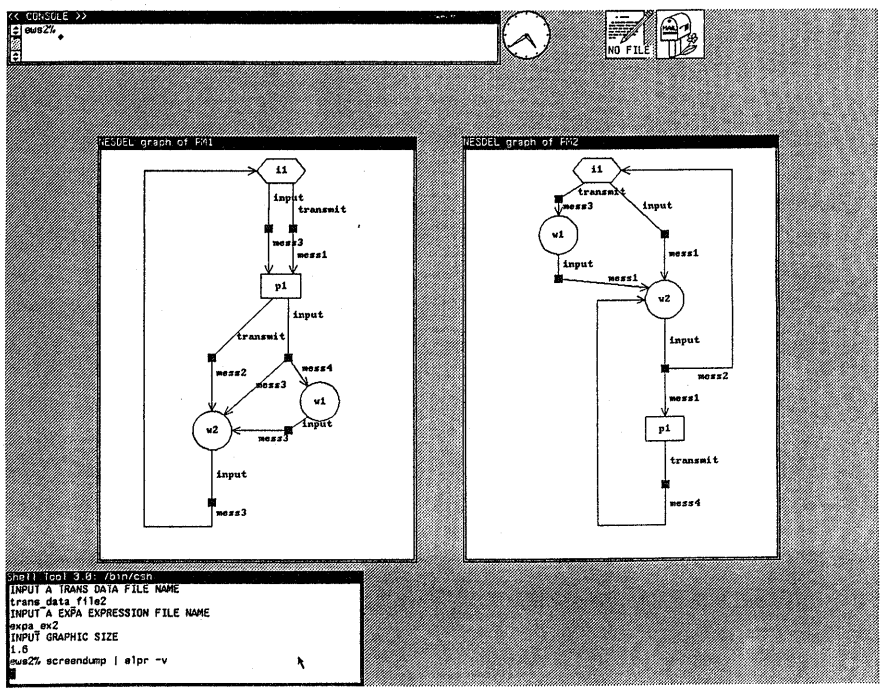

図5 プロトコルのNESDEL表現

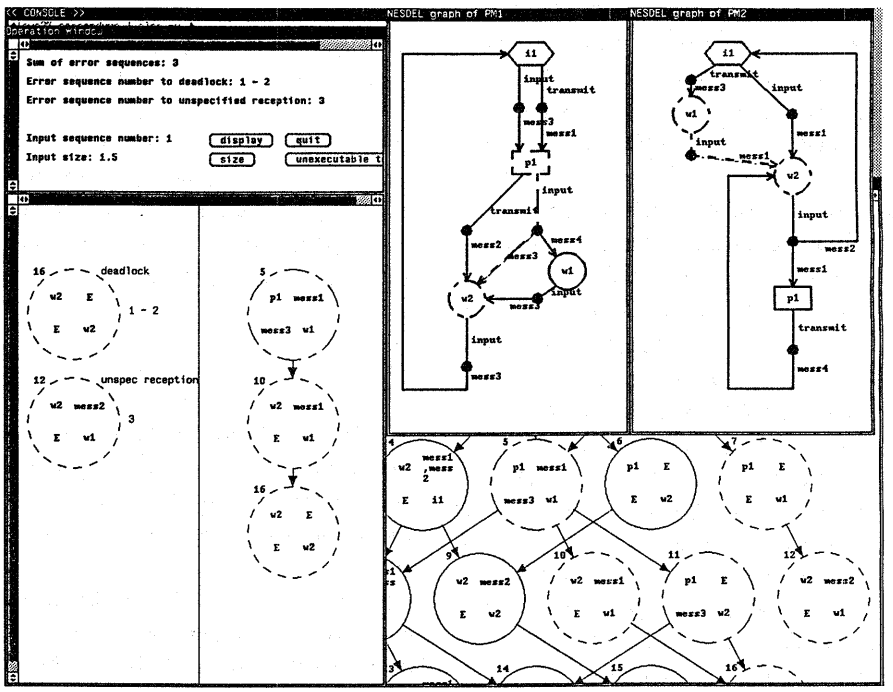

図6　検証結果の表示

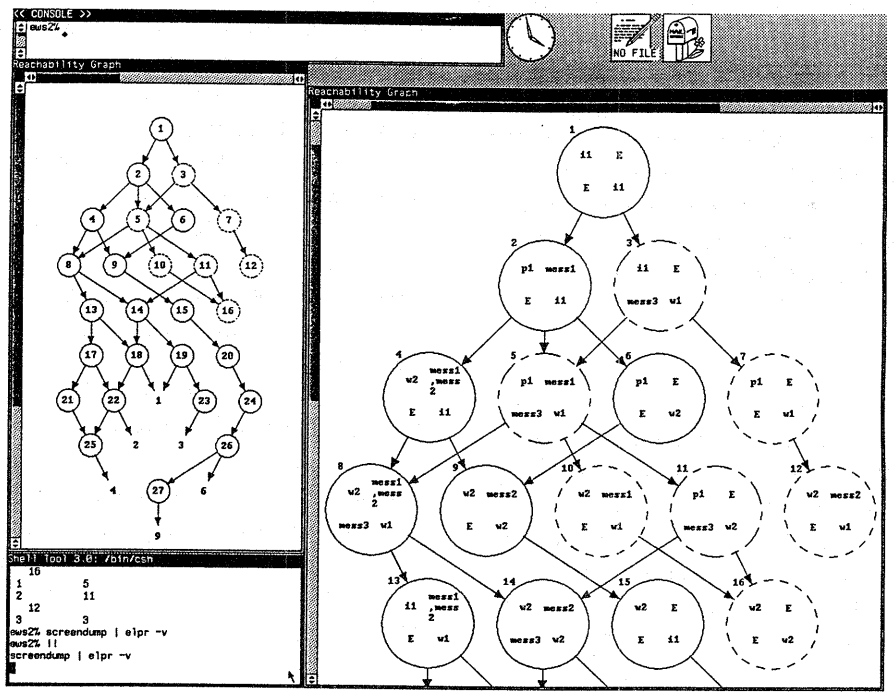

図7　到達可能性グラフの表示

て1を入力し、ボタンメニュー上の"display"を押下すると、他の4つのウィンドゥが表示される。右上の2つのウィンドゥは、EXPAII-NESDEL変換によって得られたNESDEL表現上での誤り(デッドロック)の指摘を表している。ここで、破線で示されたノードおよびアーク列がデッドロックに至る誤りシーケンスを表しており、誤りを引き起こす原因となるノード(左側のノードP1、右側のノードW1)から最終的に誤りとなったノード(左側及び右側のノードW2)までをカラー表示を用いた視認性の高い形でユーザに指摘している。また同時に、この誤り(デッドロック)及びそれに至るシーケンスのより詳細な情報として、これらが到達可能性グラフ上(右下のウィンドゥ)のどのシステム状態及びシステム状態の系列に対応するかを表示することによって(左下のウィンドゥ)、ユーザによる誤り分析の容易化を図っている。左下のウィンドゥの左側の状態16はデッドロックを示し、これに至るシーケンスを同ウィンドゥの右側に、システム状態番号のシーケンス(5-10-16)を用いて示している。このシーケンスは、元の到達可能性グラフ上(右下のウィンドゥ)でも見ることができる。また、右下のウィンドゥを改めて表示したものを図7に示す。同図において、左のウィンドゥは各システム状態の詳細には触れず、到達可能性グラフの全体構成をユーザに見通し良くコンパクトに表示する。一方、右のウィンドゥは、システム状態の内容を含めたより詳細な表示を行っている。これら2つのウィンドゥ間の対応は、システム状態に付された識別番号によってなされる。

なお、実際のシステムでは、ユーザの理解性を高める観点から、誤りに関した情報はすべてカラー表示を用いている。ここでは、論文投稿上の制約からカラー表示が使用できないため、破線を使用した。本システムはC言語を用いて作成し、各構成要素のプログラムサイズは、EXPAII検証が約2200行、NESDEL-EXPAII変換が約940行、EXPAII-NESDEL変換が約1500行となっている。

9. むすび

本論文では、ユーザフレンドリィなプロトコル検証システムを構成し実現した。本検証システムの特長は、NESDEL-EXPAII変換およびEXPAII-NESDEL変換の機能によって、ユーザは仕様記述言語の上でユーザフレンドリィに検証を実行しその結果を知ることができる点にある。これによって、ユーザは仕様記述言語だけを知っていれば十分であり、検証用の表現や検証結果の表現を知る必要はない。従って、ユーザによる検証結果の分析の容易化に大きく貢献する。また前章で示したように、検証の結果はグラフィカル表示かつ色変化を用いてユーザに提示されるため、結果に対するユーザの理解性を高めたものとなっている。

今後の課題として、生成される状態数の削減がある。これについては、従来の手法、例えば文献(9)の手法、を採用すると状態数が半減する。また、NESDELの階層的仕様記述の機能を効果的に用いることによりさらに状態数の削減が可能であり、現在検討中である。

文　献

(1) N.Shiratori, J.Gohara and S.Noguchi : "A New Design Language for Communication Protocols and a Systematic Design Method of Communication System", Proc. of the 6th International Conference on Software Engineering, pp.403-412 (Sept. 1982).

(2) T.P.Blumer and D.P.Sidhu : "Mechanical Verification and Automatic Implementation of Communication Protocols", IEEE Trans. Software Eng., SE-12, 8, pp.827-834 (Aug. 1986).

(3) K.K.Sabnami and A.M.Lapone : "PAV-Protocol Analyzer and Verifier", Proc. of IFIP 6th International Workshop on Protocol Specification, Testing and Verification, pp.(2-1)-(2-25) (June 1986).

(4) C.H.West : "General Technique for Communication Protocol Validation", IBM J. Res. Develop., 22, 4, pp.393-404 (July 1978).

(5) 白鳥, 高橋, 野口 : "NESDEL : プロトコル向き仕様記述言語とその応用", 情処学論, 26, 6, pp.1136-1144 (Nov. 1985).

(6) 白鳥, 郷原, 野口 : "EXPA : パータベーション解析に基づく通信プロトコルの検証法", 情処学論, 26, 3, pp.446-453 (May 1985).

(7) ISO : "ESTELLE - A Formal Description Technique Based on an Extended State Transition Model", ISO/DP 9074 (1985).

(8) CCITT : "Functional Specification and Description Language (SDL)", CCITT Recommendations Z.100-Z.104 (1984).

(9) J.Rubin and C.H.West : "An Improved Protocol Validation Technique", Computer Networks, 6, pp.65-73 (Dec. 1982).

(10) 高橋, 白鳥, 野口 : "通信ソフトウェア向き超高級プログラミング言語IDLとその適用", 情報処理, 26, 11, pp.1378-1387 (Nov. 1985).