階層化プロトコルを実現するプログラムの検証論理の生成法

荒木　哲郎　　高田　賢次　　吉武　静雄
（　電電公社　横須賀電気通信研究所　）

1. まえがき.

プロトコルの標準化がＩＳＯ等[1]で進むにつれて、プロトコルをインプリメントした個々の製品が、標準プロトコルに正しく準拠していることを検証するシステム（検証システムと呼ぶ）の開発が必要となってくる。最近、このような検証システムの研究・開発が、Euronet[2]、NBS[3]，ＮＰＬ[4]等で行われている。

検証システムを開発する上で二つの重要な問題がある。一つは、プロトコル製品を網羅的に検証する為の検証シーケンスを系統的に求める手法を研究することで、例えばBochmann[5]，斎藤等[9]の研究がある。他の一つは、実際の検証システムの上で、このような検証シーケンスを具体的に実現する方法を求めることである。本論文では主に後者の問題を扱っており、検証シーケンスを汎用的な検証システム上で実現する為の体系的な手法を提案する。時に、汎用的な検証システムとして、種々のプロトコル製品との通信を容易にする為のプロトコル実行部と、主にプロトコルエラーに関する検証シーケンスの生成・検証を司どる検証論理部から成るシステムを考え、その上で実現されるべき検証論理（プロトコル製品を検証する為に必要な手順）を、双対なオートマトンの概念を用いて生成する方法を述べる。

2. プロトコル製品の検証モデル

プロトコル製品（計算機、端末等）が標準プロトコルに準拠しているか否かを検証する検証システムのモデルについて述べる。

検証システムと検証対象製品の一般的なモデルを、ＯＳＩの参照モデル[1]に基づいて図１に示す。ここで検証システムは、各検証対象製品と通信回線を介して通信を行いながら、また各製品をブラックボックスと見なしてその入出力を観測することによって検証を行う。検証システムは二つのコンポネントから構成される。一つは各製品との通信を容易にする為に必要なプロトコル実行部である。ここで検証システムと検証対象製品の各々のプロトコル実行部の関係は、一方が送信側のとき他方は受信側になることである。もう一つのコンポネントは、検証論理部と呼ばれ、主にプロトコルエラーに関する検証シーケンス（すなわち検証システムのプロトコル実行部では生成できないもの）を生成・検証するものである。

検証システムにおいて、プロトコル実行部と検証論理部を結合する為のモデルを図２に示す。ここでは制御がレイヤ間インタフェースのサービスアクセスポイントで分岐され（このようなメカニズムをディレールポイントと呼ぶ）、検証論理部へ送られる。

検証の完全化を期す為に、検証対象製品内に検証用モジュールを組込む方法が提案されているが[5][6][9]、多様な異機種製品を検証する場合には、製品毎にプログラムを準備するわずらわしさがあり、実用的な観点からは短期間に検証を行う上で大きな阻害要因となる。本論文では、このような検証用モジュールを、製品内に一切組込まない方法で検証することとする。但し、検証対象製品側には、製品を操作する為に必要なアプリケーションプログラムやオペレーションは存在するものとする。

また各検証対象製品を検証する際の

検証の実行単位としては、初期状態から始まり再び初期状態に戻る一つのサイクル(検証サイクルと呼ぶ)を考え、各状態遷移アークが少くとも一回通過する方法で求まる検証サイクルの集合に対して検証を実施することとする。

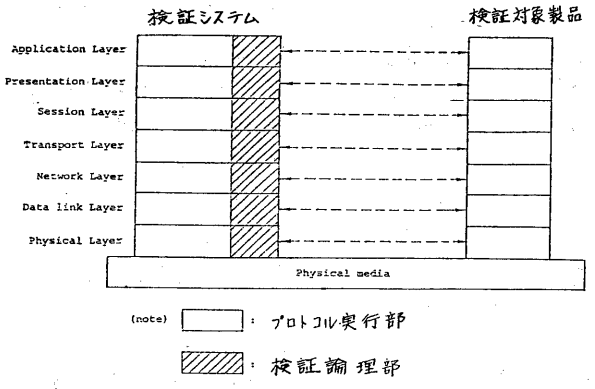


図1. OSI参照モデルに基づく検証システムモデル

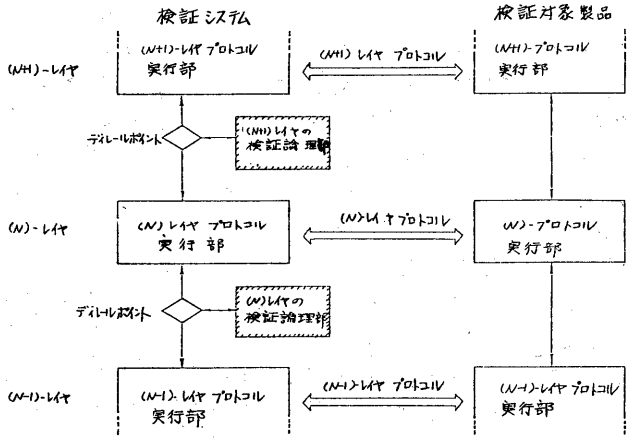


図2. 検証論理部の実現モデル

3. 検証システムが実現すべき状態遷移構造.

ここでは、2.で述べた検証システムが、検証対象製品を網羅的に検証する為に、実現しなければならない状態遷移構造を求める。

3.1. プロトコルモデル

図1において各レイヤは一つのエンティティから構成される。同位レイヤのエンティティは、同位のプロトコルを用いて互いに通信する。このような各プロトコルエンティティは、次の5組から成るオートマトンによって定義される。

$M=(Q, I, O, \delta, W)$

ここで、

Q: 有限な状態の集合、
I: 有限な入力シンボルの集合、
O: 有限な出力シンボルの集合
δ: $Q\times I\to Q$, 状態遷移関数
W: $Q\times I\to O$, 出力関数
q_0: Mの初期状態 ($q_0\in Q$)

である。

また、Mの表現として次のような遷移グラフも用いる。各頂点をMの一つの状態に、また各アーク(遷移アークと呼ぶ)を一つの遷移に対応づけ、その遷移を引きおこす入力と遷移結果の出力によって各遷移アークをラベル付けする。

本論文で考えるオートマトンは強連結で、次の二つのタイプの入力/出力シンボルを有すものと仮定する。

(1) 外部シンボル: 通信回線を介して、同位レイヤのエンティティ間で送受信されるコマンド/レスポンスである。

(2) 内部シンボル: (1)以外のシステム内部で生じるイベントで、例えばコンファメーション/インディケーション等である。

このうち、内部シンボルは本検証に関して重要なインパクトを与えないから、一様に特別なシンボル"*"で代表して表わすことにする。すなわち、入力集合I, 及び出力集合Oは各々、

$I=\{i_1, i_2, \cdots, i_{m-1}, *\}$.

$O = \{o_1, o_2, \cdots\cdots, o_{t-1}, *\}$
と表わされ、i_j ($1 \le j \le m-1$)、o_k ($1 \le k \le t-1$) はコマンド／レスポンスである。明らかに、上記(1)、(2)の組合せより、全ての遷移アークは、4つのカテゴリー即ち、*/*, */o, i/*, i/o に分類される。

3.2. 双対オートマトン

3.1で述べたプロトコルエンティティを表わすオートマトンMに対して、それと双対なオートマトン$\overline{M}$を次のように定義する。

$M = (Q, I, O, \delta, w)$ に対して、$\overline{M} = (\overline{Q}, \overline{I}, \overline{O}, \overline{\delta}, \overline{w})$ が双対なオートマトンと呼ばれるのは、写像$\alpha = (\alpha_Q, \alpha_I, \alpha_o)$の下で、$\overline{M}$が以下の条件を満たすときである。すなわち、

$\alpha_Q : Q \to \overline{Q}$ は1対1写像
$\alpha_I : I \to \overline{O}$ は恒等写像
$\alpha_o : O \to \overline{I}$ は恒等写像であり、
任意な$q \in Q$, 任意な$i \in I$ 及び $w(q, i) = o$ となるある$o \in O$に対して、

(i) i又はoが内部シンボルの時：
Mと$\overline{M}$の間に次式が成り立つ。
$\alpha_Q(\delta(q, i)) = \overline{\delta}(\alpha_Q(q), \alpha_o(o))$、及び、$\alpha_I(i) = \overline{w}(\alpha_Q(q), \alpha_o(o))$
(上式の関係を、図3に示す)。

(ii) iとoが共に外部シンボルの時：
特定の状態$\overline{q'} \in \overline{Q}$ ($\overline{q'}$はQの中には存在しない状態)に対してMと$\overline{M}$の間に次式が成り立つ。
$\overline{\delta}(\alpha_Q(q), *) = \overline{q'}$、かつ $\overline{\delta}(\overline{q'}, \alpha_o(o)) = \alpha_Q(\delta(q, i))$ 及び、
$\alpha_I(i) = \overline{w}(\alpha_Q(q), *)$ かつ
$\overline{w}(\overline{q'}, \alpha_o(o)) = *$.

上述した双対オートマトン$\overline{M}$は、ちょうどもとのオートマトンMに対して入力と出力の関係が逆になっている(一方が送信側のとき、他方は受信側となる関係)。このような双対オートマトンの概念は、Zafiropulo によって示された

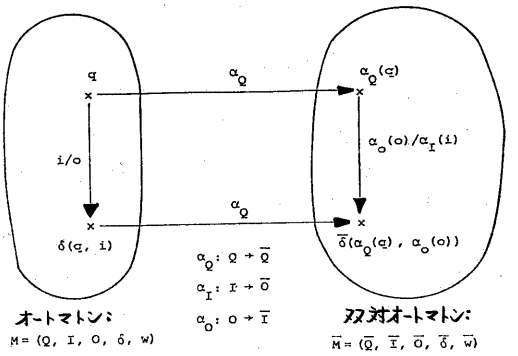


図3. オートマトン M, $\overline{M}$ 間の双対条件

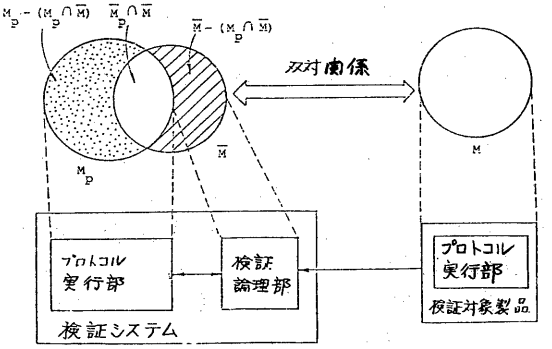


図4 オートマトン M, $\overline{M}$, M_P 間の関係

duologue[8]の概念と類似なものである。また双対オートマトンは、プロトコル仕様で規定されている通信相手側の状態遷移とは異なっている。即ち、一般にプロトコル仕様では、エラーの受信に関する遷移動作は規定されているが、エラーの送信に関する遷移動作は規定されない。検証対象製品を網羅的に検証する為には、このようなエラーの送信も検証システム側で実現しなければならないが、双対オートマトン$\overline{M}$には必要なエラー送信動作が全て含まれている。

以上のことから次の事が示される。
"検証システムが、オートマトンMを実現する検証対象製品を網羅的に検証できるのは、検証システムが双対オートマトン$\overline{M}$を実現するときで

ある。〃

また検証システムにおいて、図1で示されたプロトコル実行部で実現されるオートマトン（Mpと表わす）と双対オートマトンの関係は、一般に図4のようになる。同図において、

(i) $Mp \cap \overline{M}$: プロトコル実行部で実現可能な双対オートマトンの遷移部分、

(ii) $\overline{M} - (Mp \cap \overline{M})$: 双対オートマトンの中で検証論理部で実現しなければならない遷移部分、

(iii) $Mp - (Mp \cap \overline{M})$: 検証対象製品がオートマトンMで許されていない遷移を実行したとき、検証システムのプロトコル実行部で検出される遷移部分

である。

一般に、$Mp \cap \overline{M}$ 及び $\overline{M} - (Mp \cap \overline{M})$ はそれぞれ正常な遷移及び検証システムからのプロトコルエラーの送信に対応する遷移を表わす。ここで $Mp \cap \overline{M}$ 及び $\overline{M} - (Mp \cap \overline{M})$ の各々に含まれる各遷移アークを、それぞれPアーク及びLアークと呼ぶことにする。

3.3. 双対オートマトンの導出手順.

ここでは双対オートマトン並びにLアークの集合（検証論理部で実現すべき遷移アークの集合）を導出する手順を示す。

ステップ1. オートマトンMの各状態 q_i を、$\overline{q}_i$ で置きかえる。

ステップ2. 各遷移アーク i_k/o_l を次のように置き換える。

(i) $i_k = *$ かつ $o_l = *$ のとき、
$*/* \Rightarrow */*$

(ii) $i_k = *$ かつ o_l が外部シンボルのとき、
$*/o_l \Rightarrow o_l/*$

(iii) i_k が外部シンボルかつ $o_l = *$ のとき、
$i_k/* \Rightarrow */i_k$

(iv) i_k 及び o_l が共に外部シンボルのとき、i_k/o_l を、$*/i_k$ と $o_l/*$ に分解し、それらの遷移アーク向け新しい状態を定義する。

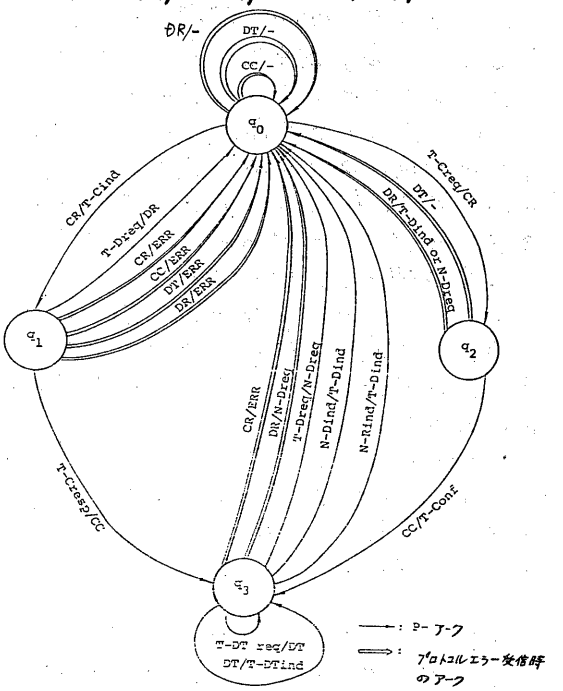
ステップ1及び2より双対オートマトン$\overline{M}$が得られる。

ステップ3. $\overline{M}$の初期状態 $\overline{q}_0$ を、Mpの初期状態に対応づける（$q_0 = \overline{q}_0$）。

ステップ4. q_0から出発又は終了する遷移パス（連続した遷移アークの列）が、Mpと$\overline{M}$間で一致するならば、これらのパスによって通過される状態 $\overline{q}_i$ を、Mpの対応する状態でおきかえる。またこれらのパスを、$\overline{M}$から取り除く。

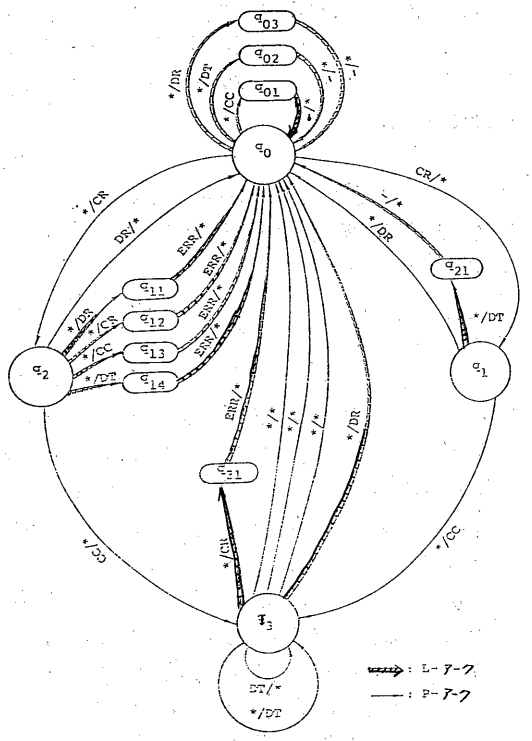
ステップ3及び4より、Lアークの集合が求まる。

《例》 ISOのOSIトランスポートプロトコル（クラス0）[ISO 8073]に対する双対オートマトンを、上記の手順に従って求めると図5のようになる。同図でハッチを施した部分（Lアーク）及び実線部分（Pアーク）は各々、$Mp \cap \overline{M}$ 及び $\overline{M} - (Mp \cap \overline{M})$ に対応し、二重線は $Mp - (Mp \cap \overline{M})$ に当たる。



(a) オートマトン M（検証システム及び検証対象製品のプロトコル実行部）

図5. ISOトランスポートプロトコル（クラス0）[ISO 8073]のオートマトンモデルと双対オートマトン（1/2）



(b) 双対オートマトン $\bar{M}$

```
  Legend: 1) Input Events
*T_Creq  = T_Connect_Request
*T_Dreq  = T_Disconnect_Request
*T_DTreq = T_Data_Request
 CR      = Connect_Request PDU
 CC      = Connect_Confirm PDU
 DT      = Data_Request
 DR      = Disconnect_Request PDU
*N_Dind  = Network_Disconnect_Indication
*N_Rind  = Network_Reset_Indication
   2) Output Events
*T_Cind  = T_Connect_Indication
*T_Dind  = T_Disconnect_Indication
*T_Cconf = T_Connect_Confirm
*T_DTind = T_Data_Indication
 ERR     = Error PDU
*N_Dreq  = Network_Disconnect_Request
where "*": internal symbol
"-": an output event means that no output is generated
   3) State
q0: Idle
q1: Wait_for_TSAP_Accept_Resp
q2: Wait_for_TPDU_Connect_Conf
q3: Data Transfer
```

図5. ISOトランスポートプロトコル(クラス0)のオートマトンモデルと双対オートマトン (2/2)

検証対象製品を検証する上で必要となる検証サイクルを以下に定義する。Mにおいて、初期状態q_0から出発し、q_0で終るもので途中でq_0を通過しないパス(一つのunilogue[8]に字価)を検証サイクルと呼ぶ。特にM及び$\bar{M}$において、全ての遷移アークが少くとも一回通過する方法で得られる検証サイクルの集合を、それぞれP_M及び$P_{\bar{M}}$と書く。ここで、$P_{\bar{M}}$は検証対象製品を網羅的に検証するとき、検証システムによって実現されるべき検証サイクルの集合を示す。このような$P_{\bar{M}}$は、[8]等の手法を用いて求めることができる。

$P_{\bar{M}}$は次の二つのカテゴリに分類される。

カテゴリ1. 一つの検証サイクルが、P-アークのみから構成されるものでP-サイクルと呼ぶ(図6-(a))。

カテゴリ2. 一つの検証サイクル内に少くとも一つのL-アークを含むものでL-サイクルと呼ぶ(図6-(b))。また一つのL-サイクル内で、連結したL-アークの集合をL-パスと呼ぶ。

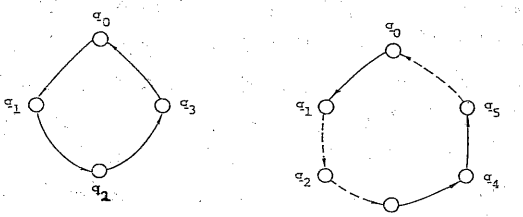


図6. 検証サイクルの2つのカテゴリー

4. 検証論理の導出方法.

ここでは3.で求めたL-アークの集合を実際に検証論理部で実現する方法を述べる。

4.1. 検証サイクルと検証論理

3.2.の議論より、プロトコル実行部は常に一つのP-サイクルを実行し、また全てのP-サイクルはプロトコル実行部で検証される。またL-サイクルは、検証論理部とプロトコル実行部

間で実行される。すなわち、一つのL-パスは、P-サイクルのある遷移(P-アーク)がプロトコル実行部で生じたとき、検証論理部で実行され、またL-パスが検証論理部で終了したとき、再びプロトコル実行部でP-サイクルを実行する。

次に一つのL-パスを実行する検証論理を一つのオートマトンと見なして次のように定義する。

$T=(S, O_1\times O_2, I_1\times I_2, g, h)$

ここで、Sは状態の集合、$O_1\times O_2$は入力シンボルの集合(O_1は検証システムのプロトコル実行部からの出力集合、O_2は検証対象製品からの出力集合)、$I_1\times I_2$は出力シンボルの集合(I_1は検証システムのプロトコル実行部への入力集合、I_2は検証対象製品への入力集合)、g及びhは各々遷移関数及び出力関数を表わす。また検証の開始状態、終了状態、及び中止状態をそれぞれS_0、S_f及びS_z(いずれもSの元)と表わす。

4.2. 検証論理の基本的な生成手順.

Pの一つのL-サイクル及び一つのP-サイクルをそれぞれC_L及びC_Pで表わす。C_L内の各Lパスの開始頂点及び終了頂点をそれぞれg_s, g_eとし、検証論理Tの任意な入力及び出力シンボルをそれぞれ(O_{1g}, O_{2k})及び$(i_{1\ell}, i_{2m})$とする。ここで、$O_{1g}\in O_1$, $O_{2k}\in O_2$, $i_{1\ell}\in I_1$, 及び $i_{2m}\in I_2$。検証論理Tを生成する為の基本的な手順を以下に示す。但し、O_{1g}, 及び O_{2k} はTで同時に生じないものとし、また検証システムのプロトコル実行部は正しく動作するものと仮定する。

ステップ1. C_L内の任意なL-パスが次の条件を満たしているかチェックする。

(i) g_sで終了するP-アークが存在しないとき(例えば$g_s=g_0$)、C_L内のg_sで開始するP-アークのタイプは*/心であること。

(ii) g_sで終了するP-アークが存在するとき、C_L内でg_sで終了するP-アークは、C_P内のg_sにおいても存在すること。

(本ステップは、一つのLパスを実現する為に、C_Pを如何に選ぶかを示す。もし条件が満たされなければ、別のC_Pが選ばれる。)

ステップ2. P-アーク上の任意な入力シンボル(O_{1g}, O_{2k})に対して、

(i) g_sで終了するP-アークが存在し、かつO_{1g}又はO_{2k}がg_sで終了するP-アークの出力シンボルの時、

ⓐ 初期状態S_0をS_1にする。

ⓑ 入力/出力を、$(O_{1g}, -)/(-, O_{1g})$又は$(-, O_{2k})/(-, i_{2m})$とする。ここで、"-"は入力及び出力が存在しないことを示し、又i_{2m}はL-パス内の最初のL-アークの出力シンボルを表わす。

(ii) g_sで終了するP-アークが存在せず、かつO_{1g}, 又はO_{2k}がg_sから出発するP-アークの出力シンボルであるとき、

ⓐ S_0をS_1又はS_zにする。

ⓑ 入力/出力を、$(O_{1g}, -)/(-, i_{2m})$又は$(-, O_{2k})/(i_1^+, i_2^+)$とする。ここで$i_1^+$及び$i_2^+$は検証を中止する為の出力シンボルを表わす。

(iii) O_{1g}又はO_{2k}がg_sで開始又は終了するP-アークの出力シンボルでなく、かつC_P内のP-アークに対する出力シンボルであるとき、

ⓐ S_0をS_0のままとする。

ⓑ 入力/出力を$(O_{1g}, -)/(-, O_{1g})$又は$(-, O_{2k})/(O_{2k}, -)$とする。

(iv) O_{1g}又はO_{2k}がC_P内のP-アークの出力シンボルでないとき、

ⓐ S_0をS_zとする。

ⓑ 入力/出力を、$(-, O_{2k})/(i_1^+, i_2^+)$とする。

(本ステップは、L-パス内のL-アークの開始を如何に行うかを述べている。特に(i)又は(ii)の場合には、検証システムのプロトコル実行部は、状態 δs, 又は $\delta s'$, $\delta s''$ で停止している。ここで $\delta s'$ は δs の直前の状態を、又 $\delta s''$ は δs の直後の状態を表わす。)

ステップ3. (i) L-パス上の入力シンボル($O_{1\delta}$, O_{2k})の系列に対して、

ⓐ 状態 S_u を S_{u+1} にする($u=1, 2, \cdots, n-1$)。但し、$S_k = S_f$(最後のL-アークにあたる)。

ⓑ 入力/出力を、(-, -)/(-, i_{2m}) 又は(-, O_{2j})/(-, i_{2m})とする。

(ii) 任意の S_u 及びL-パス上に現われない任意の入力シンボル($O_{1\delta}$, O_{2k})に対して、

ⓐ S_u を S_z とする。

ⓑ 入力/出力を、(-, O_{2j})/(i_1^+, i_2^+)とする。

(本ステップは、各L-パスを実際に検証論理として実現する方法を示す。)

ステップ4. ステップ3-(i)に対して、

ⓐ 状態 S_{f+v} を S_{f+v+1} とする

($v=0, 1, \cdots\cdots$)

ⓑ 入力/出力を(-, -)/(i_{1e}, -)又は($O_{1\delta}$, -)/(i_{1e}, -)とする。ここで $O_{1\delta}$ 及び i_{1e} は δs 又は $\delta s'$, $\delta s''$ から出発し、δe で終るP-アーク上の入力及び出力シンボルである。

(本ステップは、検証システムのプロトコルの実行部の状態を、δs, $\delta s'$ $\delta s''$ から δe へ移す方法を示す。)

《例》 上記手順に基づいて、図5で示したプロトコルに対する検証論理Tの生成例を図7に示す。ここでP-サイクル(C_P)及びL-サイクル(C_L)としては次のものを考える。

C_L: (δ_0) */CR · */CR · ERR/* (δ_0)

C_P: (δ_0) */CR · CC/* · DT/*, */* (δ_0)

また上記に示した検証論理Tは、Tの検証が開始する前に予めキューに蓄えておき、Tが開始されるとき、検証システムの制御部によってTはキューから順に取出され、検証論理部に設定されて検証を開始する。

5. 実験結果と評価.

ここでは我々が開発した検証システムを用いた検証実験の結果と本論文で提案した手順の適用性等について述べる。

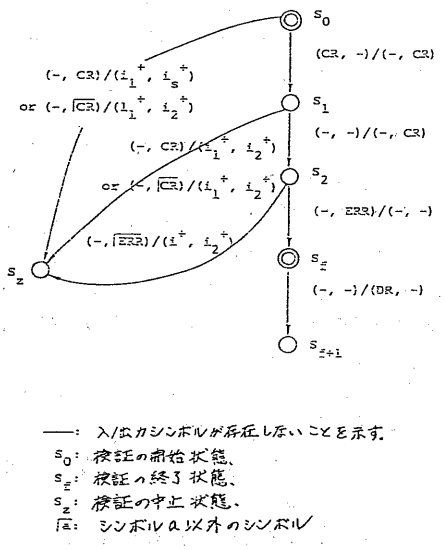


図7. 図5に対する検証論理の例

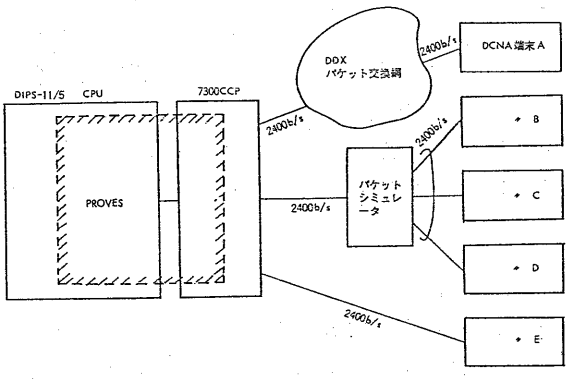


図8 PROVES を用いた検証実験構成

のCNAプロトコル[10]に基づいてインプリメントした製品が開発されるのに伴い、これらの製品を検証する為のDCNA製品検証システム（PROVES）[11]の開発を進めている。我々は今回異なる製造メーカによって開発されたDCNA製品を対象に、PROVESを用いて図8の構成で検証実験を行った。その結果は次の通りである。

(1) 本実験を行うにあたって必要な検証論理は基本的には全て本文の手順に従って作成でき、そのプログラムサイズは平均で49ステップであった。

(2) 各DCNA製品は、平均112個の検証サイクルにより検証され、1検証サイクルあたりの検証時間は約2分であった。

(3) 各DCNA製品は、自社の試験を受けていたにも拘らず、PROVESは1製品あたり、7個の誤りを検出した。これらの誤りの約半数はDCNA仕様の解釈誤りに帰因するものであり、この種の誤りは自社内の試験では検出しにくいものである。

(4) 検証論理として生成すべき部分は、検証全体の中で約40%（全体の遷移アークの中でレーアークの占める割合）を占め、重要な役割を担っている。

(5) 本実験において、製品側でのアプリケーションプログラムやターミナルオペレータによって製品内部のイベントを人為的に生成すること（ビジー条件等）により、検証システム側と製品側の状態不一致の問題を適宜解決でき、結果として約90%の遷移アーク（*/* タイプの遷移アーク部分を除く）を検証できた。

6. おわりに.

本論文では、検証システムにおいて必要な検証論理を体系的に生成する為の手順を、双対オートマトンの概念に基づいて示した。また実験によりこの手順が実用的であること、及び検証論理が全体の検証の中で約40%と占め、残りの60%がプロトコル実行部（図4）で行われていることを確認した。

また本論文で述べた検証システムでは検証すべき遷移アークの約90%を検証できることがわかった。

最後に本研究において御指導頂いた電電公社横須賀通研、苗村憲司氏及び河岡司氏に感謝する。

〔文献〕

[1] "Reference Model of Open System Interconnection", ISO/DIS 7498, 1982.

[2] K. Weaving: "Europe Reference and Test Center", Comp. comm., 3. No.5, 1980

[3] J.S. Nightingale: "Protocol Testing using a Reference Implementation", Proc. 2nd. International Workshop for Protocols, Protocol Specification, Testing, and Verification, Idyllwild, CA, Noth-Holland Pub. Co. New York, 1982.

[4] J.P. Ansart: "A Protocol independent system for testing protocol implenentation", ibid.

[5] B. Sarikayi and G.V. Bochmann: "Some Experience with Test Seguence Generation for Protocol", ibid.

[6] D. Rayner: "A system for Testing Protocol Implementation", Comp. Net. 6, 6, 1982

[7] ISO/CCITT: "Draft Transport Protocol Specification", Dec. 1981.

[8] P. Zafiropulo et al: "Protocol Validation by Duologue-Matrix Analysis", IEEE Trans. COM-26, No. 8, 1978

[9] 斉藤他："オートマトンモデルによるHDLCプロトコル製品検証の一方式"，信学論(D)，J63-D.8，1980.

[10] I. Toda: "DCNA Higher Level Protocols", IEE Trans. COM-28, No. 4, 1980.

[11] T. Kawaoka et.al: "A Method for Verifying Layered Protocol Products and Its Application Products", Proc. ICCC, 1980.