データベース・システム 84−14
（１９９１．７．１８）

# データベースサーバシステムの制御方式に関する一考察

根岸　和義
（株）日立製作所　システム開発研究所

データベースサーバシステムにおいてそのプロトコルの正当性の検証は重要な課題である。本報告では、有限オートマトン（ＦＳＡ）により記述された複数のプロトコルマシンで定義されているプロトコルの検証方式に関して述べる。各ＦＳＡの組合せのＦＳＡを分析して、不当状態（コミット、ロールバックの混在した状態）や行き止まり状態（所期の最終状態に遷移できない状態）が発生しないことを検証し、何台のサーバマシンを用意すれば実機における動作確認が可能であるかを示した。ＦＳＡの組合せに際して状態の直積でなく、異なる状態の組合せのみを考えることにより、サーバマシンの台数とは独立にすべての台数のサーバに対して上記の検証を可能とした。

# A Study of Control Methods for Database Server Systems

Kazuyoshi Negishi

Systems Development Lab., Hitach Ltd.
1099 Ohzenji Asao-ku Kawasaki-si 215, JAPAN

Verification of communication protocol is important for database server systems. This paper discribes a verification method of the protocol. That protocol is defined by number of finite state automata(FSA). We examine these FSA to exclude invalid states which include both commit state and rollback state, and to exclude halt states which has no transition path to final states. We show how many server machines are needed for verification of the protocol with real machines. Reduced FSA is used to analize the protocol independently of the number of database servers.

## １．まえがき

近年、大型システムの発達とともに、計算機システムの処理能力に対するニーズも増大している。これに対処する手段の一つとして、複数の計算機システムを通信回線により連結し、互いのデータベースを相互にアクセス可能としたデータベースサーバシステムに対するニーズも高まっている。また、異機種の計算機システムによりデータベースサーバシステムを構成する際に、その相互接続を保証するプロトコルの標準化がＯＳＩのＴＰ[2]等として推進されている。

従来、このプロトコルの定式化には、主に有限状態オートマトン（ＦＳＡ）によるプロトコルマシンとその直積によるＦＳＡが用いられてきた[1]。そこでは、ｎを固定値とした、１対ｎのモデルが使用されている。

従来の方式の問題点は、１対ｎのモデルに対するプロトコルの検証能力が不十分であった点である。ｎを固定した場合、それをＦＳＡとして、不正状態（各ノードのコミット、ロールバックが混在した終了状態）を生じないこと、および行き止まり状態（終了状態への遷移ができない　状態）を生じないことをモデルで検証し、実機で確認することはできるが、すべてのｎについて、ＦＳＡによる検証、および実機による確認を行うことは困難である。また、プロトコルの検証においてｎ台の個々の状態をすべて検証する必要はなく、状態の組合せ方の各々に着目して検証すれば十分な場合が多い。

そこで我々は、ＦＳＡによりモデル化し、プロトコルマシンの状態の直積でなく、状態の種別を考え、状態の組合せによるＦＳＡを解析する方策を考案した。この方式によれば、ノード数ｎとは独立にプロトコルの検証を行うことができる。

我々の対象とするシステムは、クライアント、サーバモデルにより表現されるデータベースサーバシステムである。クライアントのマシンは、全体を制御するメインマシンとサーバ対応の個別マシンから成る。各ノードは、独立にジャーナルを取得し、ノード障害に備えるシステムである。このシステムにおいて、各プロトコルマシンをＦＳＡの状態遷移により、モデル化した。プロトコルマシンのうちサーバマシン対応に設定される部分をまずサーバ直積マシンとして統合し、このマシンとシステム全体を統括するプロトコルマシンとのメッセージのやりとりを組合せマシンとしてモデル化し、不当状態への遷移を組合せマシンによりチェックする方策を取った。

本報告のシステムの特徴は、１対ｎのプロトコルマシンの状態組合せマシンによるシステムのモデル化と、その検証方策の策定である。なお、具体的システムにおける１対ｎのプロトコルマシンの定義方法、および障害回復に着目したその検証法則に関してはすでに報告済みである[4]。本報告では、任意台数のサーバにおけるプロトコルの検証の統一的方策を示す。

## ２．モデルの概要

我々は評価方式を明確にするため、サーバ、クライアント間のトランザクション処理に着目し、下記のように簡略化したモデルを検討した。本報告で想定するプロトコルマシンの構成を図１に示す。システムの詳細はすでに報告済み[4]であるので、ここでは概要のみを示す。システムはクライアントノード及びｎ台のサーバノードより構成される。クライアントノードには、全体を統括するメインマシンと、サーバノードに対するインタフェイスとなる個別マシンがある。一方サーバノードにはサーバマシンがある。各プロトコルマシンはメッセージにより互いに

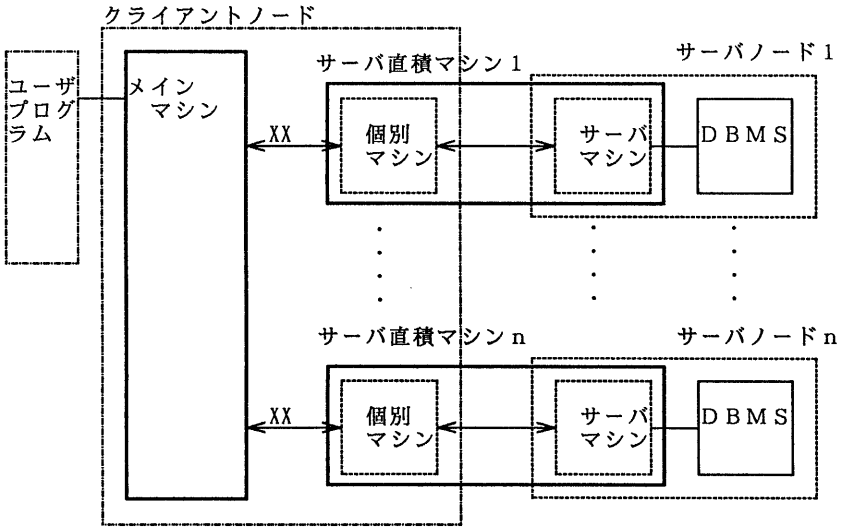


図１　プロトコルマシンの関連図

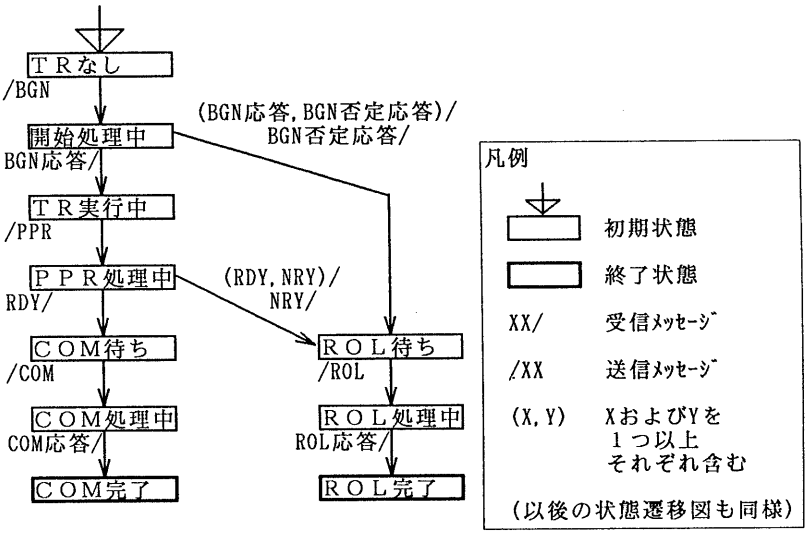


図２　メインマシンの状態遷移図

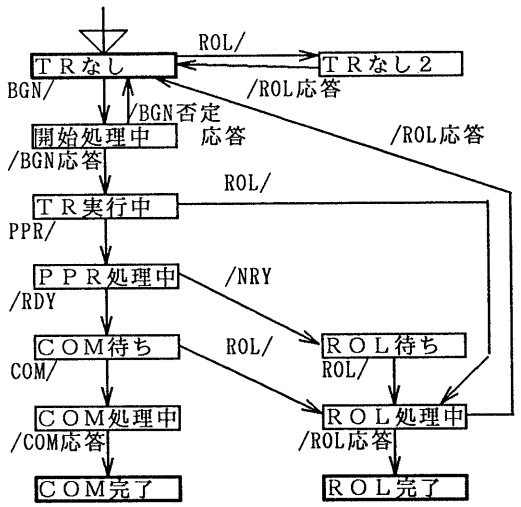


図３　サーバ直積マシンの状態遷移図

連絡する。

２．１　メッセージの伝達モデル

メインマシンと個別マシン、個別マシンとサーバマシン間のメッセージは各プロトコルマシン間で同時には一つのみ存在する（半二重通信方式を想定）。また、メッセージの消失、追越は想定しない。ノード障害の発生時、受信されていないメッセージは削除される。

メインマシンと個別マシンの間では、メインから個別には常に同一の内容を全個別マシンへ送信する。これは、トランザクションの処理開始時に関連する全ノードへトランザクション開始を連絡する方式を想定している。また、トランザクションに関連しないノードに関してはモデル化の対象外とする。また、個別マシンから、メインマシンへのメッセージは、下記のいずれかとする。

（１）全個別マシンから同一のメッセージを受信する。

（２）全個別マシンから二種類のメッセージのいずれかを受信する。ただし、各メッセージを

少なくとも一つの個別マシンから受信するものとする。

２．２　プロトコルマシンの詳細

メインマシンの状態遷移図を図２に示す。個別マシンとサーバマシンの直積を考え、メインマシンとのやりとりに関連のない部分を統合したものが、サーバ直積マシンである。図３にサーバ直積マシンの状態遷移図を示す。この図において、開始状態は「ＴＲなし」状態であり、終了状態は「ＣＯＭ完了」状態、「ＲＯＬ完了」状態、「ＴＲなし」状態のいずれかである。

３．プロトコルの正当性検証

次に、メインマシンとサーバ直積マシンとの組合せマシンを考える。図４の例でこれを説明する。まず、初期状態は、メインマシン、サーバ直積マシンとも「ＴＲなし」状態であるから、組合せマシンの初期状態は（「ＴＲなし」，（「ＴＲなし」））である。ここで、（Ｓ０，Ｓ１，．．，Ｓｍ）は、Ｓ０がメインマシンの状態、Ｓ１，．．Ｓｍがｍ通りのサーバ直積マシンの状態の組合せを表す。メインマシンからサーバ直積マシンへＢＧＮメッセージを同時に送信すると組合せマシンの状態は、（「開始処理中」，「開始処理中」）となる。次に、サーバマシンがダウンすることなく開始処理を終了すると全てのサーバ直積マシンからＢＧＮ応答が返り、組合せマシンは状態（「ＴＲ実行中」，「ＴＲ実行中」）に遷移する。しかし、いくつかのサーバノードでダウンが発生した場合、当該サーバ直積マシンはＢＧＮ否定応答を返すから、組合せマシンの状態は、（「ＲＯＬ待ち」，「ＴＲ実行中」，「ＴＲなし」）に遷移する。また、すべてのノードでダウンが発生した場合、すべてのノードがＢＧＮ否定応答を返すので、組合せマシンの状態は、（「ＲＯＬ待ち」，「ＴＲなし」）に遷移する。以下同様にして、図６が得られる。ここで、組合せマシンの終了状態は、

（「ＣＯＭ完了」，「ＣＯＭ完了」）
（「ＲＯＬ完了」，「ＲＯＬ完了」）
（「ＲＯＬ完了」，「ＲＯＬ完了」，
　　「ＴＲなし」）
（「ＲＯＬ完了」，「ＴＲなし」）

のいずれかである。従って、「ＴＲなし」と「ＲＯＬ完了」を同一視すれば、この組合せマシンにおいて、プロトコルの規約に矛盾する不正状態への遷移は発生しない。アルゴリズム１がこの組合せマシンを生成する手順を示す。また、定理１よりこのマシンが、不正状態（正常でない状態）を生じないなら、対応する直積マシンにおいても任意のｎに対して不正状態への遷移は発生しない。

次に、組合せマシンの各状態に対して直積マ

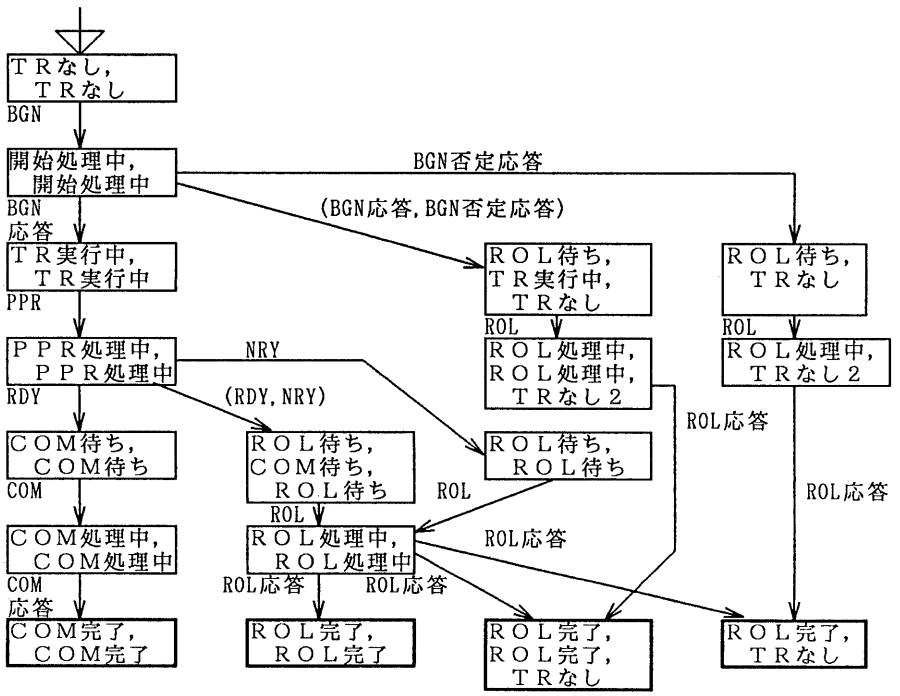


図４　組合せマシンの状態遷移図

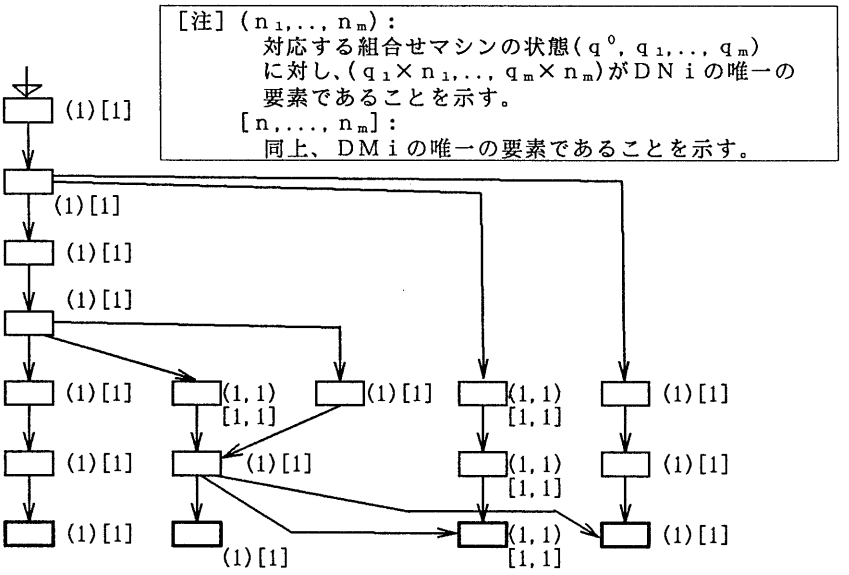


図５　アルゴリズム２の評価結果

シン上で当該状態に対応する状態に遷移するために最小限何台のサーバマシンが必要かをアルゴリズム２を使用して分析する。アルゴリズム２に従い、各ノードの

DNi：ノードiに対応する直積マシンの状態で初期状態から遷移可能な状態の集合

DMi：ノードiに対応する直積マシンの状態で終了状態へ遷移可能な状態の集合

を求める。

for all q∈DNi
　　∃q'∈DMi（q⊃q'）

ならノードiは行き止まり状態ではない。ノードiにおけるDNiの要素のうち最小ノード数をdiとした時、Max（di）が実機でのテストに必要なノード数である。

本例題に上記のアルゴリズム１を適用した結果を分析の結果を図５に示す。図５ではステップ２のDNiを（）により表し、DMiを[]により表す（記述法は図５の注記の通り）。この例では、行き止まり状態がないことがわかる。図５のDNiの状態数の最大値が２であることから本プロトコルマシンでは最小限２台のサーバマシンがあれば全ての組合せを検証できることがわかる。

ここで、DMiを定義する過程で、ノードの障害発生時にしか遷移しないものが含まれている可能性がある。これを避けるにはアルゴリズム１において、直積マシンの遷移として正常時の遷移のみを対象とするようアルゴリズムを変更すればよい。ここでは、簡単のためその可能性は考慮しない。

４．DNi，DMiの考察

DNiの持つ意味を以下に考察する。組合せマシンのノードiについて、これに対応する直積マシンのn＝１～∞のすべてのノードのうち、初期状態から移行可能なノードに対応する状態の組合せの集合を考える。この集合の要素のうち包含関係⊃に関して、他の要素を包含しない要素（最小要素）の集合がDNiである。

DMIは逆に、終了状態に移行可能なノードに対して上記を行ったものである。

しかし、このような考え方でDNi，DMiを求めるのでは、間接的に直積マシンを扱っており、組合せマシンを考えた意味が少ない。そこで、付録に示すアルゴリズムでは、直積マシンを考慮せず、組合せマシンの状態を扱う方策を示した。

５．まとめ

本報告では、データベースサーバシステムのプロトコルマシンモデルに対して、プロトコルの検証をノード別の状態ではなく、状態の種別に対して行うことにより、ノード数と独立にプロトコルが行き止まり状態を生じないことの検証を行うアルゴリズムを示した。さらに、実機においてプロトコルの検証に必要なノード数を求めるアルゴリズムを示した。

本方式により、データベースサーバシステムのように１対nの関係にあるプロトコルシステムのnと独立したプロトコル検証が可能となった。また、プロトコル実証時の必要なノード数を求めることができた。

参考文献
[1] Dale Skeen,"Formal Model of Crash Recovery in a Distributed System",IEEE Trans. on Software Eng. Vol.SE-9,No.3 May 1983 pp219-228.
[2] ISO/IEC DIS 10026 Information Processing Systems-Open Systems Interconnection-Distributed Transaction Processing
[3] J.E.ホップクロフト,J.D.ウルマン,"言語理論とオートマトン",サイエンス社(1971)
[4]根岸他,"分散データベースシステムの制御方式に関する一考察",信学技法 Vol.90, No.146,1990年7月,pp29-38.

付録 記号の定義とアルゴリズム、定理

文献［３］の記法に基づき、ｎ台のサーバ直積マシンを下記で表す。

$M=(X, Q, \delta, q_0, F)$

ここで、

Ｘ：マシンの入力記号の集合
Ｑ：マシンの状態の集合
δ：マシンの状態遷移関数の集合
ｑ：マシンの初期状態
Ｆ：マシンの終了状態の集合

とする。

次に、Ｘ，Ｑ，δにアルファベット順を仮定し、ＡをＸ，Ｑ，δのいずれかに固定した時、

$(A)^n$：Ａの要素の長さｎの列

とする。

長さｎの列$a_1..a_n \in (A)^n$に対して、

$U(a_1..a_n)=$（$a_1..a_n$のうち重複するものを除きアルファベット順に並べ換えたもの）

なる関数を考える。次に、

$(A)^+$：Ａの要素の列の集合

とした時、Ｕを拡張して、

$U((A)^+)=\{U(a_1..a_n) \mid a_1..a_n \in (A)^+\}$

とする。これを用いて、Ｍと同様にメインマシンを下記で表す。

$M^0=(X^0, Q^0, \delta^0, q_0^0, F^0)$

ここで、

$X^0=U((X)^+)$

である。

$M^0$とＭはその入出力メッセージをすべて入力記号と見なすことにより、ＦＳＡを定義している。以下の直積マシン、および組合せマシンでは、$M^0$とｎ個のＭが各々$U(x^1..x^n)$，および$x^1..x^n$を入力した時、メッセージの送受信が成立したと見なす。

サーバ直積マシンとメインマシンのノード数ｎにおける直積マシンを下記のように定義する。

$M^{pn}=(X^{pn}, Q^{pn}, \delta^{pn}, q_0^{pn}, F^{pn})$

ここで、

$X^{pn}=(X)^n$
$Q^{pn}=Q^0\times(Q)^n$
$\delta^{pn}: Q^{pn}\times X^{pn}\rightarrow Q^{pn}$
$q_0^{pn}=q_0^0 q_0..q_0$
$F^{pn}=F^0\times(F)^n$

である。ただし、

$\delta^{pn}(q_i^0 q_i^1..q_i^n, x^1..x^n)=(q_j^0 q_j^1..q_j^n)$

となるのは、

$\delta^0(q_i^0, U(x^1..x^n))=q_j^0$ かつ、
$\delta(q_i^k, x^k)=q_j^k$ （ｋ＝１～ｎ）

の時である。

また、サーバ直積マシンとメインマシンの組合せマシンを下記のように定義する。

$M^s=(X^s, Q^s, \delta^s, q_0^s, F^s)$

ここで、

$X^s=U((X)^+)$
$Q^s=Q^0\times U((Q)^+)$
$\delta^s: Q^s X^s\rightarrow Q^s$
$q^s=(q_0^0 q_0)$
$F^s=F^0\times U((F)^+)$

である。ただし、

$\delta^s(U(q_i^0 q_i^1..q_i^n), U(x^1..x^n))=U(q_j^0 q_j^1..q_j^n)$

となるのは、あるノード数ｎにおいて

$\delta^{pn}(q_i^0 q_i^1..q_i^n, x^1..x^n)=q_j^0 q_j^1..q_j^n$

が存在する時に限る。

組合せマシンの状態遷移を直積マシンを考えずに求めるアルゴリズムを以下に示す。

［アルゴリズム１］

組合せマシンのすべてのノードｉの状態：$(q_i^0 q_{i1}..q_{im}) \in Q^s$

に対して、下記により状態遷移を定義する。

(1) $q_i^0$を左辺とするクライアントマシンの状態遷移の集合を$G(q_i^0)$とする。

(2) $q_{ik}$を左辺とするサーバ直積マシンの状態遷移の集合を$G(q_{ik})$とする。$G(q_{ik})$のすべての部分集合（空集合を除く）の集合を$G(q_{ik})^+$とする。

(3)クライアントマシンとサーバ直積マシンの間で記号$X=x_1..x_r$を入出力し、ノードｊの状態$(q_j^0 q_{j1}..q_{jm})$に遷移する状態遷移の集合$G_{ijx}$を考える。

$G_{ijx}=\{\{\delta^0\}, G^1..G^m\} \mid$
$\delta^0\in G(q_i^0)$,
$G^k\in G(q_{ik})^+$,
$\delta^0(q_i^0, x)=q_j^0$,
$U(G^1..G^m$のすべての記号$)=x$,
$U(G^1..G^m$のすべての右辺$)=q_{j1}..q_{jm}\}$

(4)このような$G_{ijx}$の存在する時、

$\delta^s(q_i^0 q_{i1}..q_{im}, x)=q_j^0 q_{j1}..q_{jm}$

とする。

ここで、本報告では、初期状態から遷移可能な直積マシンの状態においては、送信可能な記号列ｘに対してはこれを受信する状態遷移が必ず存在することを仮定する。このような状態遷移が存在するかどうかをチェックするように本方式のアルゴリズムを拡張することは可能であるが、それは本報告の対象外とする。

次に、不正状態と行き止まり状態を下記のように定義する。

不正状態：個別のサーバ直積マシンの終了状態の集合を複数のグループに分けた時、直積マシンの終了状態のうちこれら複数のグループの二つ以上のグループに属する終了状態を含むものを不正状態とする。

行き止まり状態：ＦＳＡにおいて、初期状態から遷移可能な状態で、その状態から終了状態への遷移ができない状態を行き止まり状態とする。

この時、以下の定理が成立する。

［定理１］

組合せマシンにおいて、不正状態（正常でない状態）を生じないなら、対応するすべての直積マシンでもこのような状態は発生しない。

［証明］

直積マシンにおいて、初期状態から状態$q_i^0 q_i^1 . . q_i^n$への状態遷移が存在する時、組合せマシンにおいて初期状態から対応する状態$U(q_i^0 q_i^1 . . q_i^n)$への状態遷移が存在する。

定義より、直積マシンの状態及び状態遷移に対応して、組合せマシンの状態および状態遷移が必ず存在する。従って、直積マシンにおいて、初期状態から不正状態への遷移が存在するなら、それに対応して直積マシンにおいて、初期状態から対応する不正状態への遷移が存在する。

［証明終］

直積マシンＭの状態$q^{pn} \in Q^{pn}$と、$q \in Q$に対して、

Ｎ（$q^{pn}$，ｑ）＝$q^{pn}$中のｑの個数

を定義する。また、$q^0 \in Q^0$に対して、

Ｎ（$q^{pn}$，$q^0$）＝１

とする。次に、$q^{pn}$と$q^{pm}$の包含関係$\supseteq$を以下のように定義する。

$q^{pn} \supseteq q^{pm}$とは、

for all $q \in q^{pn}$，Ｎ（$q^{pn}$，ｑ）$\neq 0$

Ｎ（$q^{pn}$，ｑ）$\geqq$Ｎ（$q^{pm}$，ｑ）

である。また、

$q^{pn} \supseteq q^{pm}$かつ$q^{pn} \neq q^{pm}$の時、

$q^{pn} \supset q^{pm}$であるとする。

サーバ直積マシン任意のｎ個とメインマシンを組合せた場合、組合せマシンに行き止まり状態が存在しない場合、直積マシンにも行き止まり状態が存在しないかどうかをチェックするために、ＤＮｉ，およびＤＭｉを求める以下のアルゴリズムを考案した。

［アルゴリズム２］

［ステップ１］

組合せマシンのノード毎に探索済み状態集合Ｗｉ（初期値$\phi$）、および探索対象状態集合Ｖｉを用意する。

［ステップ２］

組合せマシンの初期状態０で、Ｖ０＝｛（$q_0^0$，$q_0$）｝とする。

［ステップ３］

ノード０を対象に以下の処理［ステップ４］を行う。

［ステップ４］

もし、for all $q \in$Ｖｉ（$\exists q' \in$Ｗｉ（$q \supseteq q'$））なら何もしないで終了。

それ以外なら、Ｕ＝｛ｑ｜$q \in$Ｖｉ，$\neg \exists q' \in$Ｗｉ（$q \supseteq q'$）｝を求める。

ＶｉＵＷｉの包含関係$\supset$に付いての最少の部分集合を新たなＷｉとする。

ﾉｰﾄﾞｉから入力記号Ｘによるノードｊへの遷移各々に付き［ステップ4-1］～［ステップ4-3］を行う。

［ステップ4-1］

状態遷移ｉ→ｊ（入力記号Ｘ）を生成する規則集合$G^k_{ijX}$の各々につき下記の［ステップ4-1-1］を行う。

［ステップ4-1-1］

$q \in$Ｕなる要素ｑの各々につき下記の(5-1-1-1)を行う。

［ステップ4-1-1-1］

ｑと$G^k_{ijX}$の規則の左辺の状態を比較し、各記号毎に、

①ｑの同一記号の個数が$G^k_{ijX}$の左辺の状態の記号の個数より多いとき、規則をすべて適用して、さらに余った記号に適用可能な規則を任意に適用してできた、すべての右辺の組合せを遷移先の状態とする。

②ｑの記号の個数が$G^k_{ijX}$の中の規則の左辺の状態より少ないとき、当該記号についてはｑに関係なく規則をすべて適用したものとして、遷移先の状態を定める。

を行い、生成した遷移先の状態の集合を求める。

［ステップ4-2］

遷移先の状態の集合のうち包含関係$\supset$につき最少の部分集合をＶｊとする。

［ステップ4-3］

ノードｊを新たなノードｉとして、［ステップ４］を再帰的に実施する。

［ステップ5］

上記がすべて終了すると、ＤＮｉ＝Ｗｉである。

上記の遷移規則の右辺と左辺の記号を反対にし、初期状態のかわりに終了状態の集合を用いることにより、ＤＭｉを求めることができる。

上記のアルゴリズム終了の保証は下記のようにして得られる。

［ステップ４］でなにもせず再帰的サーチを無限回繰り返さなければ、ノード数は有限であり、このアルゴリズムは有限ステップで終了。

［ステップ４］でさらに処理をするということは、

$\exists q \in$Ｖ

（$\neg \exists q' \in$Ｗｉ（$q \supseteq q'$））

であり、たはＷｉに加えられる。

このような入れ換えはＱの要素数が有限であること、および個別の要素$q \in$Ｑに関するＮ（ｑ，ｑ）の小さいものを追加していくことより、無限回繰り返すことはできない。従って、アルゴリズムは有限ステップで終了する。