待ち行列網シミュレーション用論理型言語とその処理系に関する研究

昭和61年12月

渡辺　尚

待ち行列網シミュレーション用論理型言語とその処理系に関する研究

昭和61年12月

渡辺　尚

序文

本論文は、筆者が大阪大学大学院工学研究科（博士後期課程通信工学専攻）に在学中に行った待ち行列網シミュレーション用論理型言語とその処理系に関する研究をまとめたもので、全文は次の７章から構成されている。

第１章は、緒論であって本研究の目的と歴史的背景並びに意義について概説する。

第２章では、待ち行列網シミュレーションの基本的構成要素と従来のモデル記述言語の特徴について述べる。まず、待ち行列網モデルはモデルの中に存在する施設間の関係を表す構造、客が施設間を渡り歩く規則を示す動作、シミュレーションの開始終了などの制御、そしてユーザに提示する統計データに関する情報である統計出力の４要素の記述から構成されることを述べる。本章ではさらにモデル記述言語として現在主として用いられている言語ＧＰＳＳおよび近年人工知能型言語として脚光を浴びている論理型言語Ｐｒｏｌｏｇとこれにシミュレーションに不可欠な時刻概念を導入したＴ－Ｐｒｏｌｏｇ、Ｃｏｎｃｕｒｒｅｎｔ Ｐｒｏｌｏｇ等の言語について述べる。

第３章において、待ち行列網シミュレーションのモデルの記述性向上を目的にして、論理型言語Ｐｒｏｌｏｇをその記述形式にした待ち行列網シミュレーション専用論理型言語ＳＩＬＱを提案する。ＳＩＬＱはまず通常のＰｒｏｌｏｇでは、実現不可能な時刻概念を、時刻に関係する状態遷移を内包した６つのタイムオブジェクト（ARRIVAL、DEPARTURE、QUEUE、SERVER、JOINT、BRANCH）によって実現する。さらに、待ち行列網モデルの定義を行うために用意した３０個の述語について詳説する。

第４章では、まずいくつかの待ち行列網モデルをＳＩＬＱで記述しプログラミングプリミティブの使用法を示す。またPrologを基本としたシミュレーション言語Ｔ－ＰｒｏｌｏｇとＳＩＬＱをモデル記述例を通じて比較し、Ｐｒｏｌｏｇが本来持っている特長をＳＩＬＱは保持しているため記述性が優れていることを示す。さらに、現在主に使用されている汎用離散型シミュレーション言語ＧＰＳＳと比較する。その結果、両者のモデル構成概念が近いこと、ＳＩＬＱではプロセスの動作等をホーン節によって定義できるためユーザの要求に柔軟に対処可能で

あること、そして待ち合わせ問題のような複数の客の流れが存在しそれらの同期を取る必要があるモデルに対してはＳＩＬＱはＧＰＳＳよりも記述性が高いこと等を示す。

第５章では、ＳＩＬＱの逐次処理系を作成する。本処理系は５つのモジュール（プロセス生成モジュール、シミュレーション制御モジュール、タイムオブジェクト実行モジュール、統計データ収集モジュール、シミュレーション結果出力モジュール）からなる。本章では、これらのモジュールの機能と動作について詳述する。

第６章では、ＳＩＬＱ処理系に並行処理技術を導入した際の効果について論ずる。ＳＩＬＱによって記述されたモデルは、構造単位及び動作単位に並列性を有する。また、シミュレーションモデル中に存在する確率的要素のため各プロセッサの処理のスケジュールが困難である。これをunknown型メッセージパッシングとして述べ、プロセッサ間の同期を規制先行制御によって行った場合の処理速度改善率について本方式の解析を行うことによって検討し、均一なモデルの場合にはその構造が複雑になるにつれ改善効果が大きくなること等を示す。

第７章は結論であり、本研究で得られた諸結果について検討を加えると共に今後の課題について述べる。

関連発表論文

A．学会論文

(A1) 渡辺,中西,真田,手塚:“規制先行制御方式を用いた非同期ジョブ並行処理システムの処理能力解析,”信学論（D）,論文,Ｊ６９－Ｄ, No.10 , PP. 1424-1433 （昭和６１－１０）.

(A2) 渡辺,中西,真田,手塚:“タイムオブジェクトを用いた待ち行列網シミュレーション専用論理型言語,”信学論（D）,論文,（採録決定）.

B．国際会議

(B1) T.Watanabe,S.Kobayashi,T.Yamaguchi,H.Nakanishi,H.Sanada, O.Kakusho and Y.Tezuka : " Simulation Language based on Logic for Queueing Network",Proceedings of PCCS KAIST , Seoul (1985-10).

(B2) T.Watanabe,H.Sanada and Y.Tezuka : "A Logic Based Network Simulation Language SILQ and Examples" , Proceedings of International Workshop on Modelling Techniques and Performance Evaluation,North-Holland, to be published , Paris (1987-3).

C．学会研究会

(C1) 渡辺,佐藤,中西,真田,手塚 :“並行処理事象型待ち行列網シミュレータＤ－ＳＳＱについて,”信学技報 ＣＳ８３－１０２ （１９８３）.

(C2) 渡辺,内尾,佐藤,中西,真田,手塚:“シミュレータＤ－ＳＳＱにおける先行制御方式の検討,”信学技報 ＳＥ８３－１０２ （１９８４）.

(C3) 渡辺,佐藤,中西,真田,手塚 :“分散型待ち行列網シミュレータＤ－ＳＳＱの処理能力解析,” 待ち行列理論とその応用研究会 京都大学数理解析研講究録 ５６４ （１９８５）.

(C4) 渡辺,小林,山口,中西,真田,角所,手塚 :“待ち行列網シミュレーション専用論理型言語について,”情処研報 Ｖｏｌ．８５ , ＭＤＰ ２６－３ （１９８５）.

(C5) 渡辺,中西,真田,手塚 : “待ち行列網解析ツールにおけるモデル記述言語の一提案,” 待ち行列理論とその応用研究会 京都大学数理解析研講究録５９６ (１９８６).

(C6) 渡辺,中西,真田,手塚:“多重世界機構を持つネットワークシミュレータの性能評価,”情処研報 Ｖｏｌ.８６, Ｎｏ.１２ ３０－８ (１９８６).

D. 学会講演発表

(D1) 渡辺,佐藤,中西,真田,手塚 :“ノード分散並行処理事象型待ち行列網シミュレータ,” 昭和５７年 電気関係学会 関西支部 連合大会 Ｇ６－１４ (１９８２).

(D2) 渡辺,佐藤,中西,真田,手塚 :“ノード分散待ち行列網シミュレータＤ－ＳＳＱについて,” 昭和５８年 電子通信学会 情報・システム部門全国大会４０３ (１９８３).

(D3) 渡辺,佐藤,山口,中西,真田,角所,手塚 :“Ｐｒｏｌｏｇによる待ち行列網シミュレーションについて,”昭和５９年電子通信学会通信 光・電波部門全国大会 ２７６ (１９８４).

(D4) 渡辺,山口,中西,真田,角所,手塚:“並行処理待ち行列網シミュレータＤ－ＳＳＱにおけるＰｒｏｌｏｇの記述性について,”昭和５９年 情報処理学会第３０回全国大会 ３Ｑ－１ (１９８５).

(D5) 渡辺,小林,山口,中西,真田,角所,手塚 :“待ち行列網シミュレーション専用言語ＳＩＬＱについて,”昭和６０年 電子通信学会情報・システム部門全国大会 ６９ (１９８６).

(D6) 渡辺,中西,真田,手塚 : “待ち行列網シミュレーション専用論理型言語の処理系について,” 昭和６１年 電子通信学会 総合全国大会 Ｓ１４－２ (１９８６).

目次

第１章

緒　論

電子計算機がこの世に誕生してから約４０年間、計算機ネットワークに代表される分散処理型システムは、ノイマンボトルネック回避による高速処理並びに地域的にかけ離れた情報源と受信者との高速情報伝達手段の提供等に大きく貢献し今日の高度情報化社会の形成に寄与してきた。これらのシステムは当初、技術者の経験と勘によって開発されてきたが、その規模の巨大化に伴い次第に開発段階で実システムの数学的モデルを構成し、達成すべき性能が得られるかどうかを解析する手法が用いられてきている。その手法の１つは待ち行列網理論による解析であり、もう１つが計算機を用いたシミュレーションである[1]。

待ち行列網理論による解析は、いまなお盛んに研究し続けられているが、対象が限定されたり、近似解析手法の精度が問題になり複雑なシステムの解析には向いていない。その一方でシミュレーションは、実システムに極めて近いモデルの性能評価が可能なこと等から重要視されてきている[2][3]。

このシミュレーションを効率的に行うために、ソフトウェア作成のみならず、モデル構築に対しても有効な概念を与えるべく様々な言語が１９６０年代後半から１９７０年にかけて発表されてきた[4][5]。その中でも、ＧＰＳＳはひときわその簡潔な概念から重要視され、いま現在最も使用されているシミュレーション専用言語である。

しかし、一般に記述性が高いと言われているＧＰＳＳでさえもプログラミングは表中のパラメータの組合せ等で行われるため、モデルが複雑化すると記述が極めて煩雑になる。また、複数の客の流れの同期を取らねばならぬモデルに対してはＧＰＳＳの内部機構を熟知せねばその作成は不可能である[4]。さらに、ＧＰＳＳの処理系は極めて低速であり、処理速度の速い利点を持つ汎用言語Fortranをその低い記述性にも拘らず用いている場合も多々ある。

この様に、最も広く用いられているＧＰＳＳはもとより他のシミュレーション

言語であっても今後益々増加するであろうシミュレーションへの要求に答えることは困難であると考えねばなるまい。この様な現状から、今後予想されうる大規模かつ高機能な分散処理型システムの効率的な性能評価を行うためにシミュレーションソフトウェア開発期間並びにシミュレーションソフトウェア実行時間の双方を軽減しうるシミュレーション言語とその処理系が渇望されている。

本研究では、まずソフトウェア開発期間に関して、待ち行列網モデルを定義するシミュレーション仮定が条件によって記述されることが多いこと並びに近年論理型Ｐｒｏｌｏｇ[6]が人工知能型言語と言われ、高い記述が評価されていること[7]を考慮し、待ち行列網モデルをＰｒｏｌｏｇ形式で定義する待ち行列網シミュレーション専用論理型言語 ＳＩＬＱ (SImulation language based on Logic for Queueing network) を提案する。

Ｐｒｏｌｏｇは、第一階述語論理中のホーン節をソフトウェアとみなす言語であり、高い記述性を持っているといわれる。これは、従来の言語がアルゴリズムすなわち、処理手順を記述していたのに対し、解くべき問題の論理的部分に注目し、事物間の論理的関係を記述することによってソフトウェア作成が可能であることが大きな要因となっている。このことは、新世代コンピュータ技術開発機構が論理型言語を核言語の一つとして選択したこと[8]からも推し量れる。

その反面、Ｐｒｏｌｏｇで記述できる世界は、単一の論理時刻に発生する事象からなる世界であるため、シミュレーションに必要な“時刻”という共通媒体を持った複数のプロセスの表現が不可能である。　そこで、　Ｔ－Ｐｒｏｌｏｇ、Ｃｏｎｃｕｒｒｅｎｔ　Ｐｒｏｌｏｇ等汎用Ｐｒｏｌｏｇに時刻概念を導入しようとする試みがいくつかなされてきた[9]-[14]が、これらではＰｒｏｌｏｇの利点を十分に生かせていない。本研究では、Ｐｒｏｌｏｇの利点を損なうことなく時刻概念を実現するためタイムオブジェクトによってプロセスの状態遷移をパッケージ化した手法をまず提案する。

次に、待ち行列網モデル定義に含まれるプロセス間の接続関係、各プロセスの動作定義などを、Ｐｒｏｌｏｇ形式で記述するための３０個のソフトウェアの最小構成要素（プログラミングプリミティブ）を用意する。そして、ＳＩＬＱの記

述性を評価するために、ＳＩＬＱによる記述例及び他の言語との比較を行う。

さらに、ＳＩＬＱ処理環境を実現するためにＳＩＬＱ逐次処理系をパーソナルコンピュータ上のＰｒｏｌｏｇを用いて構成しこの処理系の評価を実行例を用いて行う。

一般にＰｒｏｌｏｇに限らず論理型言語は、解探索の際の履歴保存等に時間を要するためシミュレーションにとって不利と考えられがちである。ＳＩＬＱ逐次処理系もこれらと同様に非常に処理速度が遅いため、これを補うべく対策が効率的なシミュレーションにとって不可欠である。そこで本研究ではさらにＳＩＬＱ並行処理系について考察する。

汎用Ｐｒｏｌｏｇの効率化に関しては（1）データ構造を改善し記憶領域への高速アクセス化[16]（2）ホーン節単位の並行処理化[17]-[20]などが考えられているが、本研究では対象を待ち行列網シミュレーションに限定し、プロセス単位の並列性に注目した並行処理によって効率改善を図ることを考える。

待ち行列網シミュレーションはソフトウェアの内部に確率的要素を含むため、複数のプロセッサの同期制御が難しい。このことをｕｎｋｎｏｗｎ型メッセージパッシングとして解釈できることを示す。そして、プロセッサ間の同期制御をプロセッサ間の一時的な矛盾の発生を許容し、可能な限り並列性を利用する先行制御方式[21]-[23]を用いた場合の処理速度改善効果についての知見を得るために本方式の近似解析を行う。その結果先行制御方式では、モデルの複雑度に対して線形に処理速度が向上するという利点を持つことを示し、ＳＩＬＱで記述されたモデルの並行処理を行った場合に得られる処理速度改善率を定量的に検討する。

第２章

待ち行列網シミュレーション

２．１　緒言

本章では、本研究の目的である高い記述性を持つ待ち行列網シミュレーション専用言語とその処理系について論議を進める上において必要な基礎概念について述べる。

まず第２節において、一般的なシミュレーションの基礎概念とユーザが対象とするシステムをシミュレーションによって性能評価する段階について述べる。

次に、現在の待ち行列網シミュレーションの状況をまず、シミュレーションソフトウェアに記述されるべき事項について述べ、ついでシミュレーションソフトウェアの作成とその処理系の２面から検討する。その中で、ソフトウェアの作成については、待ち行列網シミュレーションソフトウェアを記述する言語を汎用言語とシミュレーション言語の双方の観点から考察する。

第４節において、待ち行列網シミュレーションの効率化を目指すために、ソフトウェア作成の効率化を論理型言語Ｐｒｏｌｏｇによって図ることを示す.また、他のＰｒｏｌｏｇを基礎としたシミュレーション言語における時刻の取り扱い方法を解説し、Ｐｒｏｌｏｇの特長である宣言的記述の点からこれらの言語を検討する。

第５節では、待ち行列網シミュレーションソフトウェアの実行の効率化を並行処理によって図る際に生じるプロセッサの時刻制御に関する問題点を明らかにする。特に一時的なシミュレーション時刻の矛盾を許すプロセッサの時刻同期制御方式である先行制御方式を採用するＤ－ＳＳＱについて述べる。

２．２　待ち行列網シミュレーションの基礎

システムの性能評価とは、それを構成する要素間の条件や時刻によるシステムの状態の変化を動的に観察してそのシステムの特性や機能を知ることである。この手法としては、（１）実システムによる実験、（２）待ち行列網理論による解析、（３）シミュレーションによる解析が考えられる。対象とするシステムが巨大化し、複雑になるにつれて、（１）は当然のことながら（２）の手法によっても解析の近似精度等が問題になり困難を極める。一方、シミュレーションによるシステムの性能評価は対象とできるモデルの範囲が広い、現実に近いモデルを評価できること等の利点から大きな役割を担っている。

シミュレーションを行うに当たっては、まずモデルが必要である。モデルとは、現実のシステムに存在する物とそれが時間の経過と共にどのように変化するかの規則を表現したものである。そして、シミュレーションとは、実時刻に対応する論理時刻(シミュレーション時刻)をモデル内で考え、このシミュレーション時刻の経過によってモデルがどのように変遷するかを観察することと定義できる[2]。

このシミュレーションにおいて対象となるシステムにはシステムの状態が離散的に変化する離散系と連続的に変化する連続系がある。本研究で対象にする待ち行列網システムは、客の到着、サービスの開始、終了等によってシステムの状態すなわち、処理施設あるいは待ち行列に滞在する客の数が離散的に変化するため離散系である。従って、本研究で言及する“シミュレーション”とは以下“離散系システムのシミュレーション”を指す。

尚、この状態変化を引き起こすきっかけとなる原因を事象と呼ぶ[48]。

大型計算機のみならず、個人用コンピュータが相当な処理速度を実現している今日、シミュレーションは、大部分が計算機を用いて行われている。計算機を用いたシミュレーションは次のような手順を通して行われる。

ａ）性能を評価する上において現実のシステムの何に注目するかを明らかにする。さらにこれを対象システムの調査分析を通じて形式化する。

ｂ）シミュレーション仮定を設けることによってモデルの定義を行う。

c）モデルを表現するシミュレーションソフトウェアの記述を行う。
d）シミュレーションソフトウェアを実行する。
e）シミュレーション結果の評価、分析を行う。

シミュレーションの効率化を図る場合上記のうちb）、c）の段階においてシミュレーション言語がシミュレーションソフトウェアの作成に、また、d）の段階において言語処理系がシミュレーションの実行効率にとって重要となる。

2．3　待ち行列網シミュレーションの現状

前節で述べた待ち行列網シミュレーションソフトウェア作成並びに待ち行列網シミュレーションソフトウェア実行に関してそれぞれ独立にその現状を述べる。

2．3．1　シミュレーションソフトウェア作成の現状

シミュレーションソフトウェアの作成の前に行うシミュレーションモデルの構築についてまず考察する。

シミュレーションモデルには客や待ち行列収容施設という具体的な物の存在や関係という物質的要素と、それぞれが時刻と共にどのように変化するかという変化規則が的確に表現されていなければならない。

物質的要素とは、例えば待ち行列収容施設（以下バッファと呼ぶ）、サービス施設、客などを指し、それぞれ要素の状態、特性を示す属性を持っている.また、変化規則の表現とは、物質的要素の属性値が時刻の経過とともに変化する規則である。変化規則の表現方法としては、事象中心の表現、トランザクション（客）中心の表現、プロセス中心の表現があり、またそれぞれに対応するモデル構築法が考えられる。そして、これらのモデル構築法は単にモデルの定義のみならず、その後行うソフトウェアの作成の際にも重要な役割を果たす。

シミュレーション仮定を設け、対象とするシステムのモデルの構築を終了した後に、シミュレーションソフトウェアの作成を行う。一般にシミュレーションを

行うためには、以下が必要である。

（1）シミュレーションモデルの物質的要素とその接続関係の記述（構造）

（2）物質的要素の変化規則の記述（動作）

（3）モデルの初期状態の記述（初期状態）

（4）シミュレーション制御（制御）

（5）必要な統計処理（出力）

シミュレーション制御とは、シミュレーション時刻の更新と同一のシミュレーション時刻で処理されるべき事象、トランザクションまたはプロセスの実行順序の制御である。シミュレーション時刻の更新方式としては、ある事象に対する処理の後次に事象が生じる時刻まで更新する事象駆動型と、時刻を微少等間隔で更新する時刻駆動型がある。後者は、元来連続的な変化をするシステムを対象とする時に、用いられる方法であるが、制御が容易であることなどから離散系にも用いられている。

本研究では、定常状態におけるシステムの振舞いを対象にするため、（3）については以後考慮しないことにする。

2．3．2　シミュレーションソフトウェア記述言語

シミュレーションソフトウェアの記述には、処理速度が速い利点を持つFortran等の汎用言語とシミュレーション専用言語がある。

（1）汎用言語によるシミュレーションソフトウェアの記述

汎用言語でシミュレーションソフトウェアを記述する際にはシミュレーションモデルの構築法の決定並びにシミュレーション実行に必要な内容のすべての記述はユーザの責任において行われる。

（2）シミュレーション専用言語による記述

現在、シミュレーションソフトウェアはほとんどがこの専用言語を用いて記述されている。その理由はシミュレーション制御などを言語の機能として内包しシミュレーションソフトウェアの記述に適したモデル構築法並

びにソフトウェア作成法をユーザに提供しているためである。これによってユーザはモデルの定義に専念できるので、シミュレーションソフトウェア作成の効率化が図れる。

現在待ち行列網シミュレーションに用いらているシミュレーション専用言語としては、ＧＰＳＳ[4]、ＳＩＭＳＣＲＩＰＴ[5]、ＳＩＭＵＬＡ６７[5]等がある。

これらの中でも1961年にIBMのG.Gordon及びR.Efronらにより開発されたＧＰＳＳは、その具体的でユーザが理解しやすいトランザクション中心のモデル構築法から、現在最も普及している。しかし、ＧＰＳＳではＱＮＡテーブルまたはブロックのパラメータの指定のみにプログラミングが限定され、複雑な網を対象にしたときはユーザの要求に柔軟に対応ができない。また、複数のトランザクションの流れの同期をとらねばならないモデルに対しては、ＧＰＳＳの内部機能であるタイミングコントロールの方法を、ユーザが熟知していない限り正しいソフトウェアの記述ができないという欠点がある。

一方、1963年 Rand Corporation の H.M.Markowitz らにより開発されたＳＩＭＳＣＲＩＰＴは事象中心のモデル構築法を採用した言語であるが、汎用性に富む反面具体性に欠け、モデル構成が多様化し、その記述が難しくなる傾向があるためにＧＰＳＳほど広く用いらていない。

また、ＳＩＭＵＬＡ６７はプロセス中心のモデル構築法を採用した言語である。この言語は1964年Norwegian Computer Centerで K.Nygard, O.-J.Dahlを中心にして開発されたＳＩＭＵＬＡから、一般のシステムの記述と実現のために発展した言語である。 ＳＩＭＵＬＡ６７では同種のプロセスを１つのクラスとして定義できる等の有用な概念を提供し、今日のＳｍａｌｌｔａｌｋ[24]のようなオブジェクト指向型言語の基礎を築き上げた。しかしながら、シミュレーション言語としてはそれほど用いられてはいない。

２．３．３　シミュレーション処理系の現状

シミュレーションの実行は、多くのサンプルを採集することによってシミュレーション精度を確保するというシミュレーションの性質上多大な時間を要する。

現在前項で示したシミュレーション言語は大型計算機または、ミニコンピュータ上の逐次処理によって行われている。このため、シミュレーション実行の効率は必ずしも高いとは言えない。

２．４　待ち行列網シミュレーションソフトウェア作成の効率化

待ち行列網シミュレーションソフトウェアのモデル定義は、シミュレーション仮定を設けることによって行われるが、これは以下のものから成る。

（１）各プロセス間の接続関係

（２）各プロセスの動作の定義

（ａ）プロセスに含まれる事象の論理時刻上での順序関係

（ｂ）各事象の発生条件

これらのうちシミュレーション言語を用いてソフトウェアを作成する際に問題となるのは（２）（ｂ）である。そしてこれは条件による表現が極めて簡潔で自然であること、論理型言語Ｐｒｏｌｏｇが以下に示すような宣言的記述が可能であるという利点を持っていることを考慮し本研究ではＰｒｏｌｏｇをその記述形式とする。

Ｐｒｏｌｏｇは、第一階述語論理の中のホーン節をプログラムとみなす言語である。Ｐｒｏｌｏｇを用いるとユーザは解くべき問題について、既に知られている事実や、それらの関係を論理式によって記述するだけでよい。つまり、ユーザは解くべき問題の純粋に論理的な部分にのみ注目し、プログラムの実行状態を考慮せずにプログラミングが可能である。このためにＰｒｏｌｏｇは高い記述性を有していると考えられている。

しかし、高い記述性とは裏腹にＰｒｏｌｏｇではシミュレーションに必要な時

刻概念が扱えない。

本節では、まずＰｒｏｌｏｇの基本的な計算機構について述べ、次にＰｒｏ－ｌｏｇに時刻概念を導入した例としてＴ－Ｐｒｏｌｏｇ、Ｃｏｎｃｕｒｒｅｎｔ Ｐｒｏｌｏｇを紹介し、宣言的記述の面から検討を行う。

尚、一般にＰｒｏｌｏｇとは“ホーン節＋逐次処理系＋バックトラック”と理解されているが、本研究ではソフトウェアの記述形式としてのＰｒｏｌｏｇを意味する。

２．４．１　Ｐｒｏｌｏｇ

Ｐｒｏｌｏｇは第一階述語論理に基づく言語であり、その計算単位はホーン節である。ホーン節とは、

節　$B_1, \cdots, B_m \leftarrow A_1, \cdots, A_n.$

において以下の４種類を指す。

$B \leftarrow A_1, \cdots, A_n.$	：手続き宣言文（ｒｕｌｅ）
$B \leftarrow.$	：データ文　（ｆａｃｔ）
$\leftarrow A_1, \cdots, A_n.$	：開始文　（ｇｏａｌ）
$\leftarrow.$	：停止文　（ｈａｌｔ）

Ｐｒｏｌｏｇでは、事物に関する事実(fact)をそれぞれ述語とその引き数の組から構成された原始式で構成する。述語は必ず定項であり引き数は変項となることが許される。また、ある事実が他のいくつかの事実に依存することを表現する時に規則(rule)を用いる。規則は事物と事物の関係に関する一般的な文であり、“：－”によって連結された頭部と本体からなる。“Ａ：－Ｂ．”は“Ａが真となる条件はＢが真である”という意味を表す。

Ｐｒｏｌｏｇの実行原理は三段論法を拡張したリゾリューションと呼ばれる推論規則である。例えば、２つの節、

```
king(Y):-respects(all_man,Y).
respects(X,john).
```

から次の結論が導き出される。

```
respects(all_man,john).
king(john).
```

この例で、変数X,Yがそれぞれall_man,johnに代入されたが、この操作を単一化と呼ぶ[6]。

Ｐｒｏｌｏｇは以下の２種の非決定性を持っている。

ａ）単一化可能なrule(またはfact)が複数個存在するとき、どのrule(またはfact)を適用するかは決定されない。

ｂ）１つのgoal文の本体部に複数のgoalがある場合それらの評価順序は決定されていない。

この非決定性のため、Ｐｒｏｌｏｇによるプログラムの記述並びに検証は通常の手続き型言語とは異なりプログラムの実行順序を考えずにそれぞれの論理式の意味を考えるだけで行うことが可能である。このことをＰｒｏｌｏｇが宣言的記述が可能である利点を持つ、あるいはＰｒｏｌｏｇで記述されたプログラムが宣言的意味を持つ、と言う。

（尚、逐次処理系を仮定し、Ｐｒｏｌｏｇで記述されたソフトウェアを従来の言語と同様に逐次的に記述・検証することも可能である。）

しかしながら、この利点はＰｒｏｌｏｇでは述語の評価順序を規定できない。すなわち、シミュレーションにとって必要な時刻を共通媒体とする複数のプロセスの関係（プロセスの時刻関係）を扱えないことを意味している。

そこで、Ｐｒｏｌｏｇの宣言的記述性を生かしつつ時刻概念を導入し、複数のプロセスを扱えるようにしたいくつかの試みが成されている。

２．４．２　Ｐｒｏｌｏｇでの時刻概念の実現

２．４．２．１　時刻概念実現の手法

Ｐｒｏｌｏｇへの時刻概念導入の手法としては、以下のような４つが考えられる。

（１）中央に時刻を管理するものを想定し、これによって更新される時刻を通してプロセス間の時刻関係を記述する方法である。この様な言語の例としては、時相論理に基づく言語[14][15]などが考えられている。

（２）メッセージの送受信、リソース競合関係によってプロセス間の時刻関係を表現する。プロセス間の時刻関係はメッセージの送受信の際とリソースの束縛、開放の際に生じると考える方法であり、この言語の例としては、Ｃｏｎｃｕｒｒｅｎｔ Ｐｒｏｌｏｇ[9]、Ｐａｒｌｏｇ[13]、ＧＨＣ[25]などが考えられている。

（３）（１）および（２）を両方取り入れた言語としてはＴ－Ｐｒｏｌｏｇ[10][11]、ＴＳ－Ｐｒｏｌｏｇ[12]がある。

（４）プログラミングの対象の中に登場するプロセスの時刻に関する状態遷移が有限個のパターンに分類できる場合には、このパターンをパッケージ化したものを用いる。

（１）については、１つの論理式の中に複数の時刻に生じる事象が記述されているためにその論理式の意味は他の論理式の評価順序を考慮せねば理解できず、宣言的記述を損なう。また、（２）についてはＣｏｎｃｕｒｒｅｎｔ Ｐｒｏｌｏｇを、（３）についてはＴ－Ｐｒｏｌｏｇを例として以下述べるようにやはり宣言的記述性を損なってしまう。これらに対し、（４）では言語が用意するパッケージによってプログラミングの対象が限定されるがＰｒｏｌｏｇの宣言的記述の利点を保つことができる。本研究では待ち行列網シミュレーションに対象を限定するため、（４）の方法によって実現することを考える。この点については第３章にて詳述する。

2．4．2．2 Concurrent Prolog

Concurrent Prologは第一階述語論理に基づく並列型プログラミング言語であり、1982年 に E.Shapiroにより設計された。Concurrent Prologは読み出し専用標記（read-only annotation）“?”とコミットオペレータ（commit operator）“|”を用いて時刻概念を導入している。それぞれの機能を以下に示す。

（1）読み出し専用標記

(a) “?”の付いた変数（読み出し専用変数）は、それが未定義の間は変数以外のものと単一化しない。

(b) 読み出し専用変数は通常の変数とは常に単一化できるが、読み出し専用変数と単一化した変数はこの性質（読み出し専用）を自動的に引き継ぐ。

(c) X?の“?”はXが変数以外の項になれば自動的に消滅する。

（2）コミットオペレータ

Concurrent Prologではプログラムはガード付節（guarded clause）の並びとして表されている。ガード付節は下に示すように常に右辺にコミットオペレータ“|”を持つ節である。もし、“|”を陽に含まない節があった場合には右辺の一番左に“|”があるものとみなされる。

A：－ G | B

Aをこの節の頭部と呼ぶ。G及びBは論理的にandで結ばれた述語の並びであり、それぞれガード部、本体部と呼ばれる。コミットオペレータは、ガード部の処理が終了するまでは制御を本体部へ渡さないという逐次性の機能を持つ。

これらのConcurrent Prologのプリミティブの使用に当たってはプログラムの実行状態をユーザが考慮する必要がある。つまり、Prologの宣言的記述性を十分に生かしているとは言えない。

２．４．２．３　Ｔ－Ｐｒｏｌｏｇ

Ｔ－Ｐｒｏｌｏｇは、1982年にI.Futo らによって提案された人工知能的シミュレーションを可能とした言語である。人工知能的シミュレーションとはＰｒｏ－ｌｏｇのバックトラック機能による自動問題解決、演繹によってシミュレーションの実行を中断してモデルを修正したり、別の可能性を試すために前の状態に戻すことが可能なシミュレーションを指す。

Ｔ－Ｐｒｏｌｏｇには、論理型言語であるＰｒｏｌｏｇでシミュレーションを行うために以下の５つの概念が導入されている。

（１）プロセスの概念

シミュレーション実行中には、概念的には並列にいくつかのプロセスが存在する。プロセスはシミュレーションの初期段階と実行中に作られる。並列に進行するプロセスの同期はコルーチン制御によって行われる。

（２）リソースの概念

リソースとは、１度に１つのプロセスしか使えないようなモデルの構成要素である。リソースを使用した場合には時間が消費され、その後消去することもできる。

（３）内部時刻の概念

内部時刻はシステムによって維持され、各プロセスをスケジューリングするのに用いられる。

（４）メッセージと条件

プロセスは条件の充足やメッセージを待つことができる。条件はProlog形式で記述できる。

（５）バックトラック

バックトラックはプロセスの実行中の失敗、もしくはデッドロックや予め指定された終了時刻を越えた時に起こり、制御の選択的な経路を試すことができる。

基本的には、Ｔ－Ｐｒｏｌｏｇはこれらの概念を実現するためにＰｒｏｌｏｇ

にsend,wait_for,hold,wait,seize,release等の組み込み述語を用いている。

Ｔ－Ｐｒｏｌｏｇでは、時刻に関する組み込み述語の順序もソフトウェアの意味の一部である。すなわち、ホーン節内部に逐次性を持たせることによって時刻概念を導入しており、Ｐｒｏｌｏｇの宣言的記述が可能な利点が損なわれている。

２．５　待ち行列網シミュレーションソフトウェア実行の効率化

シミュレーションの実行には一般に膨大なコンピュータパワーを必要とし、長時間の計算機使用を余儀なくされる。このため、シミュレーションを安価に速く実行するためのなんらかの工夫が必要である。

ここでは、ソフトウェアの記述言語を特定せずに一般的な待ち行列網シミュレーションソフトウェアの並行処理という観点からシミュレーションソフトウェア実行の効率化について検討する。

２．５．１　待ち行列網シミュレーションの並行処理

高処理能力、高処理速度が得られることから様々な分野で並行処理が注目されている。しかし、現在までの並行処理が主として画像処理等、処理の実行以前に処理計画が立て易いジョブを対象にしてきており、待ち行列網シミュレーションのように高い並列性を有しているものの各処理単位内に確率的要素を含むため同期の取れていないジョブに対してはあまり並行処理は行われていない。この理由は、複数のプロセッサをできるだけ効率よく動作させるために必要な同期制御の困難さが大きな障壁となるからである。

待ち行列網モデルは、２．４に示したように構造と、動作の２面から構成されている。そして、各々の面について高い並列性を有している。従って、待ち行列網シミュレーションを並行処理する場合どちらに注目するかによって２つの方式が考えられる。

並行処理型シミュレータとしては既にＮＴＴのＮＥＷＴＳ[26]、大阪大学基礎

工学部のＨＡＳＳ－ＱＮ[27]、慶応大学のＫＤＳＳ[28]、Waterloo大学のシステム[29]等が発表されている.ＮＥＷＴＳ、ＨＡＳＳ－ＱＮは時刻駆動型シミュレータである。これらは、中央時計を設け単位時間を導入し、単位時間内の並列性を利用している。この方式は制御が簡単なため実用化が容易な反面、単位時間内に生じた事象は同時刻に生じたことになるためシミュレーション精度を上げようとすると単位時間を小さくしなければならず実行効率の低下を招く。またＫＤＳＳとWaterloo大学のシステムは事象駆動型シミュレータである。これらでは、対象とするモデルに論理的閉路が含まれる場合のデッドロック回避のために制御メッセージを導入している。この方式では、制御メッセージの到着までプロセッサの処理実行は中断されやはり処理能力向上は難しい。

この点については第６章にてプロセス間通信の新たな型としてｕｎｋｎｏｗｎ型メッセージパッシングの概念を示して詳しく述べる。

２．５．２　先行制御方式とＤ－ＳＳＱ

Ｄ－ＳＳＱ[21][22]はノード分散型の事象駆動型シミュレータであり、各プロセッサは他のプロセッサからのメッセージ受信の可能性とその論理時刻を無視して処理を続行する。Ｄ－ＳＳＱのこの時刻同期方式は先行制御方式と呼ばれている。先行制御方式は、待ち行列網が本来持っている並列性を極限まで利用するため一時的なシミュレーション時刻の矛盾を許容する方式であり、以下のアルゴリズムで示される。

（１）各プロセッサは他のプロセッサとの時刻関係を無視して処理を行う。

（２）他のプロセッサからメッセージを受信し、そのメッセージに含まれる客の到着時刻が現在のプロセッサの時刻から見て過去である場合には矛盾が生じる。これを時刻矛盾の発生という。

（３）時刻矛盾が生じた場合はプロセッサのシミュレーション時刻を客の到着すべき時刻に引き戻し、その間に実行した処理、すなわち

実行しすぎた処理を未処理の状態にする。これをキャンセルと呼ぶ。

先行制御を行うには、以下の機能が必要になる。

・各プロセッサにおいてプロセッサの履歴を保存する。

・メッセージを受信した場合に時刻矛盾発生の有無を検出する。

・時刻矛盾が生じた場合には履歴をもとにキャンセル処理を行う。

・システム全体の確定時刻を検出する。

ここで、確定時刻とは、Ｄ－ＳＳＱ中の全てのプロセッサが到達したと保証できる時刻を言う。従って、いかなるキャンセル処理であっても確定時刻以前にプロセッサ時刻を戻す必要はない。そのため、確定時刻以前の履歴は不要である。

Ｄ－ＳＳＱでは、その処理能力（同一モデルのシミュレーションを行った場合の１プロセッサシミュレータに対するＤ－ＳＳＱの処理速度比）はプロセッサ間通信が十分高速であればプロセッサ数に対して線形に増加する。つまり、１台の処理能力が小さくとも数十台のプロセッサを用いれば全体では大きな処理能力が得られる利点がある。さらに、各プロセッサの先行量をある一定値内に抑える方式（規制先行制御方式）によって処理能力を約２割改善できること、および規制先行制御方式においては処理能力を最大にする最適規制値が存在し、プロセッサが多くなればなるほど無規制時に対する処理能力改善の効果が大きくなることが実験的に示されている。

２．６　結言

本章では、待ち行列網シミュレーションの現状を待ち行列網シミュレーションソフトウェアの作成とその実行の２面から述べた。

待ち行列網シミュレーションソフトウェアの記述に関しては、汎用言語はもとより、ＧＰＳＳのようなシミュレーション専用言語であってもパラメータの指定のみによるプログラミングは複雑なモデルに対して不適当であることを述べた。

そこで、シミュレーション仮定が条件によって表現されることが多いことと論

理型言語Ｐｒｏｌｏｇの実行状態を考慮せずにプログラミングが可能である宣言的記述性が事象の発生条件の記述において有効である点に注目し、Ｐｒｏｌｏｇをソフトウェアの記述形式とすることを示した。さらに、Ｐｒｏｌｏｇに欠如している時刻概念を導入する手法について検討し、時相論理、Ｔ－Ｐｒｏｌｏｇや Ｃｏｎｃｕｒｒｅｎｔ　Ｐｒｏｌｏｇでは論理式に逐次性を持たせて時刻概念を実現しているため、宣言的記述性を十分に生かせないことを示した。そして、プログラミングの対象を限定し、プロセスの時刻に関する状態遷移をパッケージ化した方法によって、Ｐｒｏｌｏｇの宣言的記述性を保つ手法の存在を示した。

待ち行列網シミュレーションソフトウェアの実行に関しては、多大なサンプル数を要求されるシミュレーションを並行処理によって高速化する試みを示した。その中でも、プロセッサ間のシミュレーション時刻上での一時的な矛盾の発生を許容する先行制御方式によってプロセッサ数に線形に処理速度の改善を図れることを示した。

第３章

待ち行列網シミュレーション専用論理型言語ＳＩＬＱ

３．１　緒言

ＧＰＳＳ、Ｆｏｒｔｒａｎ等の現在の待ち行列網シミュレーション記述言語では、待ち行列網シミュレーションソフトウェアの記述性は必ずしも高くない。そこで本章では待ち行列網シミュレーションソフトウェアの記述性向上を目的とした待ち行列網シミュレーション専用論理型言語ＳＩＬＱ（SImulation language based on Logic for Queueing network）を提案する。

本研究では、ユーザがモデルの性質を規定するシミュレーション仮定には条件によって表現できる部分が多いこと、そしてＰｒｏｌｏｇがアルゴリズムを考慮せずにプログラミングが可能である利点を持っていることからＰｒｏｌｏｇ形式によって待ち行列網モデルを定義することを考える。

前章に述べたように、Ｐｒｏｌｏｇで表現不可能な時刻概念を実現するためにホーン節に逐次性を持たせる方法では、ユーザがアルゴリズムを考慮しなければならず、Ｐｒｏｌｏｇの宣言的記述性を損なってしまう。

そこで、プロセスの状態遷移をひとまとまりにし、パターン化したタイムオブジェクトの概念を提案し、これによってＰｒｏｌｏｇの宣言的記述が可能な特徴を保ちつつ時刻概念を導入できることを示す。

さらに、待ち行列網モデルをホーン節の集合によって定義するために必要なソフトウェアの最小構成要素（プログラミングプリミティブ）を用意する。これらを本章において明確にする。

３．２　ＳＩＬＱの設計方針

ＳＩＬＱではまず待ち行列網モデルを定義する際に規定されるシミュレーション仮定が論理的な表現によって記述可能であることとＰｒｏｌｏｇが高い記述性を持っていることを考えＰｒｏｌｏｇをシミュレーションソフトウェアの記述形式とする。次に、Ｐｒｏｌｏｇの利点を損なうことなしにシミュレーションに不可欠な時刻概念を導入し複数のプロセスを扱うために、タイムオブジェクトの概念を提案する。タイムオブジェクトは、プロセスの状態変化の組合せをパターン化したものである。タイムオブジェクトを用いるとユーザは述語の評価順序を考慮する必要がないため、Ｐｒｏｌｏｇの宣言的記述性を保つことができる。これによってユーザは、モデルがどのタイムオブジェクトに相当するプロセスで構成されているかを考えるだけで、各プロセスの動作等を定義するホーン節内部に、繁雑なプロセス間の時刻関係を記述する必要はなくなり、純粋に待ち行列網モデルの論理的性質の記述に専念できる。

ＳＩＬＱでは待ち行列網モデルを定義するために必要なモデルの構造、事象の発生条件、シミュレーション制御及び統計出力に関するソフトウェアの最小構成要素である述語（プログラミングプリミティブ）を用意する。

また、上記以外に同種プロセスの一括定義並びに階層的構造の扱いが容易であることも設計方針とした。

３．３　ＳＩＬＱにおける時刻概念の実現

３．３．１　タイムオブジェクト

待ち行列網モデルは、シミュレーション時刻という媒体を通して関係し合い同時に動作する複数のプロセスから構成されている。

本研究では、Ｐｒｏｌｏｇへ時刻概念を導入し、複数のプロセスを扱う手法としてタイムオブジェクトを提案する。タイムオブジェクトはシミュレーション時

刻を通したプロセス間の関係をプロセスの状態遷移の観点から考えたものである。

まず、オブジェクトについて述べる。オブジェクトとは、オブジェクト指向プログラミング[30][31]におけるオブジェクトと同義であり、メッセージを受信すると起動し、そのメッセージに対する処理を行い他のオブジェクトへメッセージを送信して停止する、という“物”である。

オブジェクトは次のような性質を持っている。

（1）各オブジェクトが独立に管理する内部変数を持つ。

（2）外部からはメッセージを送ることによってのみオブジェクトの内部変数の操作が可能である。

（3）オブジェクトの状態は、「メッセージを待っている状態（待機状態）」と「処理を実行している状態（実行状態）」のいずれかである。待機状態にあるオブジェクトはメッセージを受信すると実行状態になる。

これに対し、タイムオブジェクトは上記オブジェクトに時刻概念を導入したものである。つまり、上記特徴（1）（2）については同様であるが、（3）については次のように言い替えたものである。

（3’）タイムオブジェクトの状態は「メッセージを待つあるいはある条件が満たされるのを待っている状態（待機状態）」、「処理を実行している状態（実行状態）」、「ある時刻がくるまで待っている状態（停止状態）」のいずれかである。待機状態のオブジェクトはメッセージを受信またはある条件が満たされた時に、また停止状態のオブジェクトは規定された時間を経た時に他の状態に遷移する。

実行状態または停止状態にあるオブジェクトにメッセージが到着した場合は、そのメッセージはオブジェクトの前に先着順に並ぶ。

プロセス間の時刻関係とは、あるプロセスが停止している間どのプロセスが動作するかということであり、これは各プロセスがどの様な状態遷移をするかに大きく関係している。

タイムオブジェクトは待機状態、停止状態、実行状態の３状態を任意に遷移するため、プログラミングの対象を一般化すると無限個のパターンが必要になる。

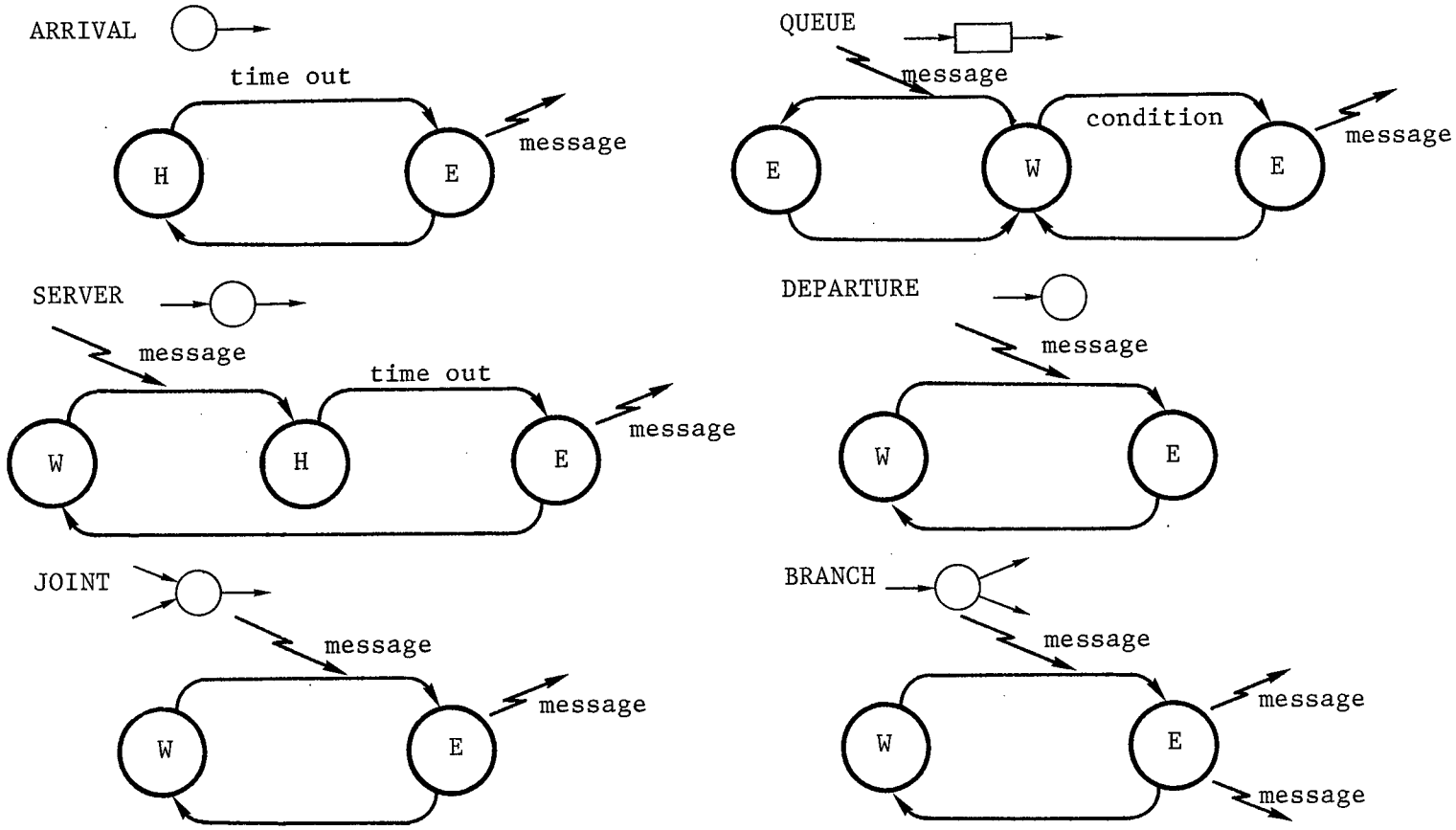


図３－１　ＱＮＳタイムオブジェクトの状態遷移

しかし、記述の対象を待ち行列網シミュレーションに限定すると待ち行列網モデルに現れるプロセスは図３－１に示す状態遷移を繰り返す６つのタイムオブジェクト(ARRIVAL,QUEUE,SERVER,BRANCH,JOINT,DEPARTURE) に限定されることがわかった。これらを待ち行列網シミュレーションタイムオブジェクト（ＱＮＳタイムオブジェクト）と呼ぶことにする。

尚、本研究で言うオブジェクトとは、同じ状態遷移の繰り返しを持つプロセス群を代表する名称である。また、客はモデル上で動く物質的要素であり、客の動きの疑似動作を行うためにオブジェクト間で送受信されるものをメッセージと呼ぶ。１つのメッセージは、一人の客の持つ履歴等の情報に送受信プロセス名を付加したものである。

３．３．２　ARRIVALオブジェクト

ARRIVALオブジェクトに属するプロセス（ARRIVALプロセス）は、実行状態になるとまず定められた形のメッセージを作り次のプロセスに送信する。その後、一定の確率分布で決定された時間が経つまで実行を停止する。

３．３．３　DEPARTUREオブジェクト

DEPARTUREオブジェクトに属するプロセス（DEPARTUREプロセス）はメッセージを受信すると実行状態になる。この時、そのメッセージで表現される客に関する統計データ収集等の処理を行った後、客を消去する。そして、次のメッセージの受信まで待機する。

３．３．４　SERVERオブジェクト

SERVERオブジェクトに属するプロセス（SERVERプロセス）はメッセージを受信すると実行状態になる。次に、そのメッセージで表される客を保持したまま予め

定められた確率分布に従う時間の間停止する。この時間が経つと、次のプロセスへ客をメッセージの形にして送信する。そして次のメッセージが到着するまで待機する。

３．３．５　QUEUEオブジェクト

QUEUEオブジェクトに属するプロセス（QUEUEプロセス）は、客を直列に並べて格納しておく施設(バッファ)を持ち、以下の二つの機能を持つ。

(a)　メッセージを受信すると実行状態になり、受信したメッセージから客を抽出し、定められた待ち行列規律に従って客をバッファに並べる。その後、待機状態になる。

(b)　バッファに客がおり、かつ予め定められた条件が満たされた場合には、待ち行列の先頭の客をメッセージとして次のプロセスに送信する。その後、待機状態になる。

３．３．６　JOINTオブジェクト

JOINTオブジェクトに属するプロセス（JOINTプロセス）には以下に示す３つのタイプがある。任意の入力経路数を持ったオブジェクトは、２入力オブジェクトの合成で機能を実現できると考え、以下では入力経路２本のオブジェクトのみを考える。

(a)　AND－JOINTオブジェクト

このオブジェクトに属するプロセスは同じシミュレーション時刻に２本の入力経路からメッセージを受信すると実行状態になり、２人の客を合成し次のプロセスへ送信する。そして、次のメッセージを受信するまで待機する。もし、片側の入力経路からしかメッセージを受信しなかった時はそのメッセージは廃棄される。すなわち、客は消滅する。

(b)　XOR－JOINTオブジェクト

片側の入力経路からメッセージを受信すると実行状態になり、同じ客を含むメッセージを作成し、次のプロセスへこのメッセージを送信する。そして、次のメッセージを受信するまで待機する。ただし、両側の入力経路から同じシミュレーション時刻にメッセージを受信した場合には両方のメッセージは共に廃棄される。

(c) OR－JOINTオブジェクト

片側の入力経路のみからのメッセージを受信、もしくは両側の入力経路から同時にメッセージを受信すると実行状態になる。次のプロセスへメッセージを送信する。そして、次のメッセージの受信まで待機する。

ただし、AND－JOINTオブジェクトのように客の合成は行わない。

実際にはOR－JOINTオブジェクトはその時点に受信したメッセージの送受信プロセス名を書き換えるだけであり、モデル上では客を単に通過させるだけである。従って特に必要性を帯びたものではない。

尚、ここで言うJOINTオブジェクトによって構成される論理が学問的に完全系を成すためには、恒真のオブジェクトが必要であるが、これは次に述べるBRANCHオブジェクトのメッセージ複写機能を用いて実現可能である。

３．３．７ BRANCHオブジェクト

BRANCHオブジェクトに属するプロセス（BRANCHプロセス）はメッセージを受信すると実行状態になる。そして、複数の出力経路から選択されたいくつかの経路へ客をメッセージとして送信する。このとき、一人の客を複写して複数の経路へメッセージとして送信することも可能である。そして、次のメッセージの到着まで待機する。

３．４ ＳＩＬＱにおけるデータ表現

ＳＩＬＱは、Ｐｒｏｌｏｇを記述形式としているため、扱える構造体はリスト

のみである。リストを用いてプロセス並びに客を以下のように表現する。

３．４．１　プロセスの表現

ＳＩＬＱでは、プロセスの識別をプロセス名によって行う。プロセス名は該当するタイムオブジェクト名と識別子を要素とするリストによって表現される。

```
( T_name , Id )
        T_name: タイムオブジェクト名
        Id:     プロセス識別子
```

図３－２　プロセス表現

例えば、タイムオブジェクト名QUEUE、識別子input-1のプロセス名は(queue, input-1)と表現される。また、識別子を変数とすることで同じ機能を持つプロセスを一括して定義することが可能である。

３．４．２　客の表現

ＳＩＬＱでは客はメッセージの形で各プロセス間を送受信される。客の構造は、図３-３に示されるようにシステムフィールドとモデルフィールドからなるリスト構造をしている。システムフィールドは客の履歴であり、その客が通ってきたプロセス名と通過した時刻を要素とするリストの集合である。ここへの書き込み等の履歴管理はＳＩＬＱ自身が行い、ユーザは読み出すことのみが許されている。一方モデルフィールドはユーザが自由に書き込み及び読み出しできる領域である。

(((T1,Pr1,Id1) , (T2,Pr2,Id2) ･･･) , MF)

system field　　model field

T : プロセスを出た時刻

Pr: タイムオブジェクト名

Id: 識別子

図３－３　客表現

３．５　プログラミングプリミティブ

ここでは、ＳＩＬＱの用意しているプログラミングプリミティブについてその分類とそれぞれの機能について述べる。

３．５．１　プリミティブの分類

ユーザはプリミティブを用いて、以下の定義をそれぞれ別々にホーン節の形で記述する。

１．待ち行列網モデルに含まれるプロセスの接続関係（構造）の定義

２．各プロセスの動作の定義

３．シミュレーション開始時刻並びに終了時刻の定義

４．統計量の定義

現在ＳＩＬＱで用意されているプリミティブは計３０個である。これらは、構造に関するプリミティブ１個、動作に関するプリミティブ１６個、開始時刻及び終了時刻に関するプリミティブ２個、統計量に関するプリミティブ１１個からなる。これら以外にも、ユーザが必要に応じて新しい述語をホーン節の形で定義することが可能である。また、これらのプリミティブは以下に示されるシステムプリミティブとユーザプリミティブの２種類にも分類できる。

（A）システムプリミティブ

ＳＩＬＱ内部で定義されているプリミティブであり、ユーザはホーン節の本体部でのみこのプリミティブを用いることができる。

（B）ユーザプリミティブ

ユーザがその動作などの定義を行うプリミティブである。実際には、ホーン節の頭部でのみ用いられ、ユーザはその引数もしくは本体部を記述することでその定義が行える。このプリミティブには、

（1）ユーザが必ず定義を行わなければならないプリミティブ

（2）ユーザが定義しなければＳＩＬＱが自ら用意しているデフォルトによってこの定義を補うプリミティブ

の２種類がある。

以下、ＳＩＬＱの提供するプログラミングプリミティブの各々についてその機能の概略を示す。尚、ＳＩＬＱプログラミングプリミティブの一覧表並びに構文を表３－１、付録１にそれぞれ示す。

３．５．２　構造の定義

構造に関するプリミティブにはユーザプリミティブの connect がある。

connect(C_Id,Ps_Id,Pr_Id)

C_Id : 接続識別子

Ps_Id: 送信プロセス名

Pr_Id: 受信プロセス名

connectは２つのプロセス Ps_IdとPr_Idを接続するプリミティブである。C_Idはこの接続の識別子である。ユーザはconnectを用いて各プロセス間の接続関係を明らかにすることでモデルに含まれているプロセスとモデルの構造を定義する。また接続識別子を用いて２つの接続されたプロセスを１つのプロセスと見なし、階層的に定義することも可能である。この有用性は１つのバッファとサーバを１つのノードと見なすような待ち行列網モデルによく現れる階層的な構造を容易に

扱える点にある。

３．５．３　動作の定義

モデルの動作の定義に際しては、各プロセスごとに専用のプロセスプリミティブと、任意のプロセスの記述に使用可能である共通プリミティブがある。

３．５．４　シミュレーション制御の定義

シミュレーション制御定義プリミティブには、シミュレーション開始時刻とシミュレーション終了時刻の２つの定義のみが用意されている。

３．５．５　統計出力の定義

ユーザは任意のシミュレーション時刻における統計データ及び統計量をこのプリミティブで定義することができる。　ユーザは、ユーザプリミティブであるdata_collect_sectionによって統計データ収集開始時刻並びに終了時刻の定義を行う。そして、１０個のシステムプリミティブで出力統計の定義を行う。なお、統計量に関するシステムプリミティブには動作に関するプリミティブと同様に各プロセス専用のプロセスプリミティブと共通プリミティブがある。

３．６　結言

本章では、待ち行列網シミュレーションソフトウェアの記述性向上を目的とした待ち行列網シミュレーション専用言語ＳＩＬＱの提案を行った。ＳＩＬＱはプロセス中心のモデル構築法に基づく言語である。Ｐｒｏｌｏｇの宣言的記述が可能である特徴を損なうことなく待ち行列網モデルをＰｒｏｌｏｇ形式で定義するために、

（1）６つのＱＮＳタイムオブジェクトを提案し、これらにプロセス間の時刻関係を内包する手法を示した。

（2）待ち行列網モデルをホーン節集合として定義するため３０個のプログラミングプリミティブを用意した。

この結果、ユーザはモデル中に含まれるプロセスがどのタイムオブジェクトに相当するかを考えることで、　ホーン節内部に時刻を持ち込むことなくモデルをＰｒｏｌｏｇ形式によって宣言的に定義できる。

表３－１　プログラミングプリミティブ

（１）　プロセスプリミティブ

（ａ）　ARRIVAL

・　客のモデルフィールドへの書き込み

append_model_field(Ar_Id,Model_Field)

Ar_Id ： ARRIVALの識別子（以下同様）

Model_Field ： 客のモデルフィールド

・　客の到着時間間隔

interval(Ar_Id,T)

T ： 客の発生時間間隔

（ｂ）　QUEUE

・　待ち行列から客が出られる条件の設定

queue_out_condition(Q_Id)

Q_Id ： QUEUEの識別子（以下同様）

・　バッファの最大容量

buffer_capacity(Q_Id,C)

C ： 最大容量

・　待ち行列規律（プライオリティ）

queue_priority(Q_Id,Hp,Lp)

Hp ： 優先度の高い客

Lp ： 優先度の低い客

・　現在待ち行列に並んでいる客

sys_customers_in_queue(Q_Id,List)

List ： 待ち行列に並んでいる客のリスト

（ｃ）　SERVER

・　サーバーが同時にサービスできる客の最大容量

server_capacity(S_Id,C)

S_Id ： SERVERの識別子（以下同様）

C ： 最大容量

・　サービス時間

service_time(S_Id,T)

T ： サービス時間

・ 現在SERVER内でサービスを受けている客

sys_customers_in_server(S_Id,List)

List : 現在サービスを受けている客

(d) JOINT

・ 客の合成 (And_JOINT)

customer_merge(J_Id,MF1,MF2,New_MF)

J_Id : JOINTの識別子

MF1,MF2 : 合成される前の客のモデルフィールド

New_MF : 合成された後の客のモデルフィールド

(e) BRANCH

・ 客の送出経路の選択

branch_selection(B_Id,Route)

B_Id : BRANCHの識別子

Route : 選択された経路の接続識別子

(2) 共通プリミティブ

・ 現在の論理時刻

sys_time(T)

T : 現在の論理時刻

・ 各プロセスを通過した客

sys_customers_record(Pr_Id,List)

Pr_Id : プロセス名 (以下同様)

List : プロセスから送信された客のリスト

・ プロセスに現在いる客

sys_customers(List)

List : プロセスに現在いる客のリスト

・ ユーザの定義した述語の書き換え

sys_change(user(Pre),user(Post))

user(Pre) : 変化前の状態を持った述語

user(Post) : 変化後の状態を持った述語

・ 客のモデルフィールドの変更

customer_change(Pr_Id,Pre_MF,Post_MF)

Pre_MF : 客の変更前のモデルフィールド

Post_MF : 客の変更後のモデルフィールド

・ シミュレーション開始時刻

start_time(St)

St : シミュレーション開始時刻

・ シミュレーション終了時刻

end_time(Et)

Et : シミュレーション終了時刻

(1) 統計データ収集区間の定義

・ 指定シミュレーション時間範囲の統計データを収集

data_collect_section(Ts,Te)

Ts : データ収集開始時刻

Te : データ収集終了時刻

(2) 出力統計の種類の定義

(a) QUEUE

・ 平均待ち行列長の出力

sys_mean_queue_length(Q_Id)

Q_Id : QUEUEの識別子 (以下同様)

・ 待ち行列長の分散の出力

sys_div_queue_length(Q_Id)

・ 最大待ち行列長の出力

sys_max_queue_length(Q_Id)

・ 平均待ち時間の出力

sys_mean_queueing_time(Q_Id)

・ 待ち時間の分散の出力

sys_div_queueing_time(Q_Id)

・ 最大待ち時間の出力

sys_max_queueing_time(Q_Id)

(b) SERVER

・ 利用率の出力

sys_utilization(S_Id)

S_Id : SERVERの識別子

(2) 共通プリミティブ

・ 各プロセスを通過した客の総数の出力

sys_customers_number(Pr_Id,MF)

Pr_Id : プロセス名

MF : 客のモデルフィールド

・ 指定プロセス間の平均遅延時間の出力

sys_mean_delay_time(Ps_Id,Pg_Id)

Ps_Id : 出発プロセス名

Pg_Id : 到着プロセス名

・ 指定プロセス間の遅延時間の分散の出力

sys_div_delay_time(Ps_Id,Pg_Id)

第４章

ＳＩＬＱによるモデル記述とその評価

４．１　緒言

本章では、ＳＩＬＱによるモデル記述手法を示し、数種の待ち行列網モデルを記述することによって、ＳＩＬＱの記述性の評価を行う。

まず第２節において、ユーザが対象とするシステムの性能評価をＳＩＬＱを用いたシミュレーションによって行う際のモデル構築並びにソフトウェア記述の方法を示す。そしてＳＩＬＱによるモデル記述例を最も基本的なＭ／Ｍ／１待ち行列モデルに帰着可能な公衆電話ボックスモデルとウィンドウフロー制御を行うパケット交換網によって示し、ＳＩＬＱでのプログラミング手法の記述性における利点を定性的に述べる。

次に第３節において、ＳＩＬＱと他のシミュレーション言語との比較を行い、本言語の記述性を評価する。比較対象としては、第２章で述べた時刻概念導入方法の(1)(2)の両者をかね添えた論理型シミュレーション言語Ｔ－Ｐｒｏｌｏｇとシミュレーション言語の中で現在もっとも広く使用されているＧＰＳＳを取り上げる。言語の定量的評価については未だ統一的解釈は存在しないが、本節ではステップ数を一応の目安として評価する。

最後に言語としてのＳＩＬＱの総合的評価をまとめる。

４．２　モデル記述

４．２．１　ＳＩＬＱによるモデル記述

２．２に述べたシミュレーション過程の内ｂ）ｃ）についてＳＩＬＱでは、シミュレーションの仮定をプログラミングプリミティブによって宣言的に記述する

ことによって行う。すなわち、

b'）対象とすべきシステムの特徴を考慮し、システムがどのようなタイムオブジェクトに相当するプロセスで構成されているかを考え、プロセス毎のシミュレーション仮定を設ける。

c'）モデルの構造であるプロセスの接続関係、プロセス毎のシミュレーション仮定並びに統計出力等をプログラミングプリミティブによって記述する。

以下この手順に従ってソフトウェア作成を行う。

４．２．２　公衆電話ボックスモデルの記述

まず、１つの公衆電話ボックスの電話を不特定多数の人が使用するモデルを記述する。

客が、電話ボックスに到着した時に他の客が電話を使用している場合は、先着順で待ち行列に並ぶ。電話の使用を終了すると客は電話ボックスから退去する。そして、待ち行列の先頭の客が電話ボックスに入る。

b'）電話ボックスへの客の到着は、ARRIVALプロセスによって表される.電話ボックスの前の待ち行列は、単一と考え１つのQUEUEプロセスで表す.公衆電話はSERVERプロセス.電話が終ってボックスから出ることはDEPARTUREプロセスで表す。これを図式化したものを図４－１に示す。

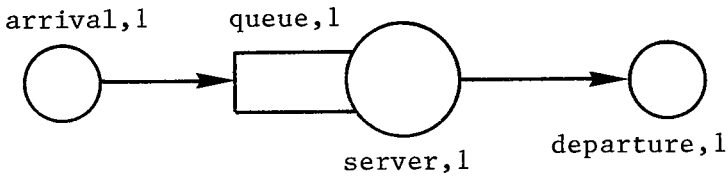


図４－１　公衆電話ボックスモデル

また、各プロセス毎のシミュレーション仮定並びに統計量収集について以下のように設定する。

（1）客の到着間隔は平均５分の指数分布に従う。

（2）サーバで客が費やす時間、すなわち各客ごとの電話をかける時間は平均３分の指数分布に従う。

（3）待行列の規律はＦＩＦＯであり、離脱客はないとする。

（4）統計量は、定常状態に達したシミュレーション時刻１０００分から３０００分までの電話の利用率、待ち行列長、各客ごとの待ち時間を求める。

c'）シミュレーション仮定をプログラミングプリミティブによって記述し、プログラミングしたものを図４－２に示す。

```
/*      Queueing Model     */
/*          M/M/1          */

/*      ARRIVAL            */
interval(1,T) :- exp_dis(5,T).

/*      QUEUE              */
queue_out_condition(1) :-
                sys_customers_in_server(1,[]).

/*      SERVER             */
service_time(1,T) :- exp_dis(3,T).

/*      CONNECT            */
connect(1,[arrival ,1],[queue    ,1]).
connect(2,[queue   ,1],[server   ,1]).
connect(3,[server  ,1],[departure,1]).

/*      START & END TIME          */
start_time(0).
end_time(3000).

/*      STATISTICS  DEFINITION     */
data_collect_section(1000,3000) :-
        sys_customers_number([departure,1],_),
        sys_mean_delay_time([arrival,1],
                                [departure,1]),
        sys_mean_queue_length(1),
        sys_mean_queueing_time(1),
        sys_utilization(1).
```

図４－２　モデル記述例１（公衆電話ボックスモデル）

図４－２には以下のような内容がホーン節によって記述されていることが明白である。

（１）客の到着間隔は平均５分の指数分布である。

（２）(server,1)に客がいない時に先頭の客は、待ち行列から出られる。

（３）サービス時間は平均３分の指数分布である。

（４）(arrival,1)と(queue,1)、(queue,1)と(server,1)および(server,1)と(departure,1)がそれぞれ接続されている。

（５）シミュレーション開始時刻は０分、終了時刻は３０００分である。

（６）シミュレーション時刻１０００分から３０００分における(departure,1)に到着した客の数、(arrival,1)から(departure,1)への客の平均遅延時間、平均待ち行列長、平均待ち時間、(server,1)の利用率をシミュレーション結果として要求する。

尚、待ち行列規律は記述されていないが、デフォルトとしてＳＩＬＱが規定しているＦＩＦＯである。

この様にＳＩＬＱによるプログラミングでは、シミュレーション仮定そのものをホーン節によって記述しているため、シミュレーション仮定を立てた段階でプログラミングのほとんどを終了したと言える。

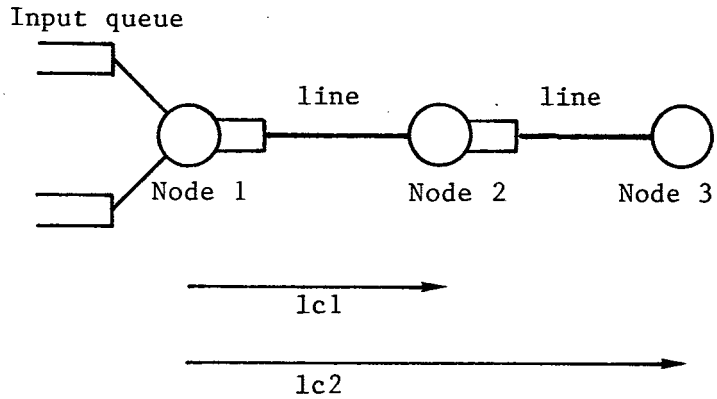
また、プログラミングの全てをプログラムの実行状態を考慮することなしにすなわち宣言的な記述によって行える。

４．２．３　ウインドウフロー制御モデルの記述

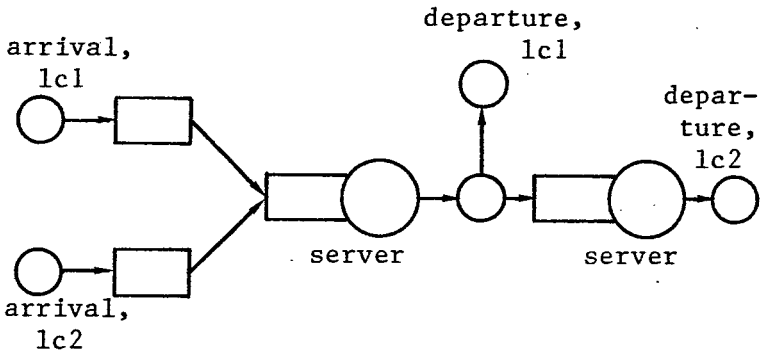
ネットワークモデルの記述例としてウインドウフロー制御を用いるパケット交換網を挙げる。ウインドウフロー制御方式[32]は、パケット交換網等におけるフロー制御の一手法であり、網内に入れる客（ここでは、パケット）の総数に予め一定の上限値（ウインドウサイズ）を決めておく。ある客がこのシステムに到着した場合に、その時点において網に入っている客数がウィンドウサイズ以下の時にはこの客は直ちに網内に入れるが、上限値に達していた時には客は入力待ち行

列に並ぶ。入力待ち行列内の客は、到達確認によってウィンドウに空きができたときに網内に入れる。

ここで取り上げるモデルは、図４－３（ａ）に示す様な発信局・目的局の対（論理チャネルＬＣ）毎にウィンドウサイズを設定するエンドツーエンドウインドウフロー制御を実装した３局直列網である。



(a) Three-node network simulation model



(b) Two-server model in Time-object

図４－３　ウィンドウフロー制御シミュレーションモデル

まず、この図４－３（ａ）の３局直列網モデルは局内処理時間が回線の伝搬遅延時間に比べて小さく無視できるとすれば、回線をサーバと見なすことで図４－３（ｂ）に示されるような２個のサーバが直列に接続された待ち行列網モデルと考えることができる。

この待ち行列網モデルでは以下の条件を設けることによってウインドウフロー制御を行うことができる。

【１】　論理チャネルの設定

ＬＣ１上のパケットは、(arrival,lc1)で発生し、(server,1)を通り(departure,lc1)で退去する。

ＬＣ２上のパケットは、(arrival,lc2)で発生し、(server,1)、(server,2)を通り(departure,lc2)で退去する。

論理チャネルは、(branch,1)に到達したパケットの情報を用いて分岐先を決定することによって設定する。これはシステムフィールドの履歴を参照しても行うことが可能であるが、arrivalにおいて、発生したパケットのモデルフィールドにそのプロセスの識別子を書き込むことによって、パケットのＬＣを容易に判定可能である。

【２】　ウインドウサイズの決定

各論理チャネルの入力待ち行列からパケットが出ることのできる条件は、その論理チャネルのdepartureから退去したパケットの数とqueueを通り過ぎたパケットの数の差がウインドウサイズよりも小さいことである。

ＳＩＬＱによるウインドウフロー制御モデルの記述を図４－４に示す。また、各種数値条件を次のように設定した。

・パケットの到着間隔　(arrival,lc1)…平均３秒の指数分布
　　(arrival,lc2)…平均２秒の指数分布
・伝送遅延時間　(server,1),(server,2)共に平均１秒の指数分布
・ウインドウサイズ　ＬＣ１…サイズ２
　　ＬＣ２…サイズ３

```
/*              WINDOW FLOW                    */
/*  Arrival  */
interval(lc1,T):-exp_dis(3,T).
interval(lc2,T):-exp_dis(2,T).
append_model_field(Id,[SF],[SF,[Id]]).
/*  Queue  */
queue_out_condition(Id):-
        member_of(Id,[lc1,lc2]),
        sys_customers_record([queue,Id],X),
        sys_customers_record([departure,Id],Y),
        C is X - Y ,
        window_size(Id,MP),
        C < MP.
window_size(lc1,2).
window_size(lc2,3).
queue_out_condition(Id,Tc):-
        member_of(Id,[1,2]),
        sys_customers_in_server(Id,[]).
queue_out_selection(Id,[Packet|Rest],Packet).
buffer_capacity(Id,3):-
        member_of(Id,[lc1,lc2]).
/*  Server */
server_capacity(Id,1).
service_time(Id,T):-
        member_of(Id,[1,2]),exp_dis(1,T).
/* Branch  */
branch_selection(1,1):-
        sys_customer([X,[1]]).
branch_selection(1,2):-
        sys_customer([X,[2]]).
/*    Connect   */
connect(_,[arrival,lc1],[queue    ,lc1]).
connect(_,[queue  ,lc1],[queue    ,1]).
connect(_,[arrival,lc2],[queue    ,lc2]).
connect(_,[queue  ,lc2],[queue    ,1]).
connect(_,[queue  ,1],  [server   ,1]).
connect(_,[server ,1],  [branch   ,1]).
connect(1,[branch ,1],  [departure,lc1]).
connect(2,[branch ,1],  [queue    ,2]).
connect(_,[queue  ,2],  [server   ,2]).
connect(_,[server ,2],  [departure,lc2]).
```

図４－４　モデル記述例２（ウィンドウフロー制御モデル）

この結果、入力待ち行列からパケットが出られる条件等に対するプログラミングプリミティブによる記述は極めて自然であり、ウィンドウフロー制御のような複雑なモデルに対してもＳＩＬＱの宣言的記述が可能な利点が記述性向上に寄与していることがわかった。

４．３　他の言語との比較

４．３．１　Ｔ－Ｐｒｏｌｏｇとの比較

記述例は、図４－５に示すような２局が存在し、閉路を持つような待ち行列網である。以下にシミュレーション仮定を示す。

（１）客の到着時間間隔は０から２０単位時間までの一様分布

（２）各サーバにおけるサービス時間は０から９単位時間までの一様分布

（３）Server２を出た客が網から退去する確率は０．５である。

（４）シミュレーション開始時刻は、０、終了時刻は１００００単位時間である。

この例に対するシミュレーションプログラムをＴ－ＰｒｏｌｏｇとＳＩＬＱによって記述した[49]。図４－６にＴ－ＰｒｏｌｏｇとＳＩＬＱによるサーバの部分の記述を示す。この部分についてのみ比較を行う。まず、Ｔ－Ｐｒｏｌｏｇでは、以下の４ステップでサーバの動作定義が成される。

（１）Queue１に対してサーバが空きであることを知らせる。

（２）Queue1からのパケットを待つ。

（３）サービス時間だけパケットを保持する。

（４）パケットをQueue2に送信する。

これらの記述順序は決定されており、例えば、述語wait_forとholdを入れ換えたものは全く異なった意味を成す.また、Ｔ－Ｐｒｏｌｏｇでは待ち行列やSERVERの記述の他にプロセス間のメッセージ送受信を記述しなければならない。このことは、Ｔ－Ｐｒｏｌｏｇでの時刻概念がホーン節内部に処理順序を表現することによって実現されているためであり、これによってＰｒｏｌｏｇの宣言的記述性が損なわれている。

これに対し、ＳＩＬＱでは、パケットがServer1で費やす時間、Server1で同時にサービスを受けることのできるパケット数の２つの定義から成り立っている。

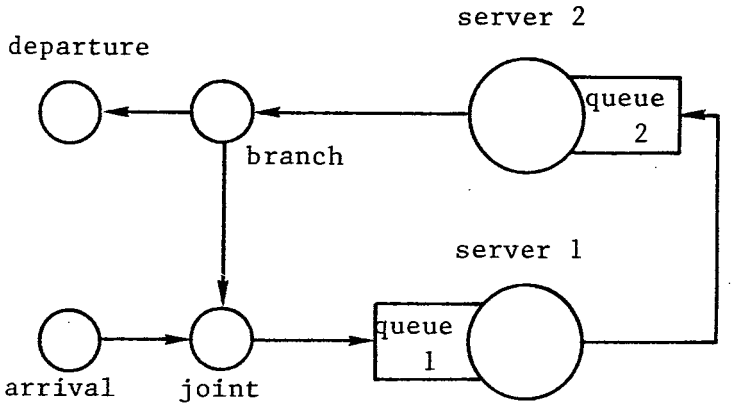


図４－５　２局網シミュレーションモデル

```
        /*  Server 1  */
server1:-send(queue1,[qout,s1]),
        wait_for(Packet),
        intserv(s1,T),hold(T),
        send(queue2,[qin|Rest]),
        server1.
intserv(s1,T):-s_random(9,T).
```

(a) Program list in T-Prolog

```
        /*  server  */
service_time(Id,T):-s_random(9,T).
server_capacity(Id,1).
```

(b) Program list in SILQ

図４－６　モデル記述例３

（ａ）Ｔ－Ｐｒｏｌｏｇによる記述

（ｂ）ＳＩＬＱによる記述

これらの定義は、ソフトウェアの実行状態すなわち節の評価順序を考慮せずに行える。この性質は、Serverに限らずQueue等でも顕著な差が生じる。

４．３．２　ＧＰＳＳとの比較

次に、シミュレーション専用言語ＧＰＳＳとの比較を行う。

ＧＰＳＳでは、トランザクションが通る道筋すなわちトランザクションの受ける処理過程をブロックによって構成する。各ブロックには、数個のパラメータが用意され、ユーザは、このパラメータを指定することによって、各ブロックの機能を設定する。

ＳＩＬＱでは、客の通る道筋をタイムオブジェクトの組合せによって構成する点、サービス施設において費やす時間等のパラメータを設定する点については、ＧＰＳＳと同様であるが、例えば、待ち行列から出られる条件、分岐条件等が複雑になった場合には、ＧＰＳＳはあくまでパラメータの操作によって行わざるを得ないため、工夫が必要なのに対し、ＳＩＬＱではホーン節という条件の表現に適した記述形式を採用しているためにこれらの記述が容易である。

この意味において従来の手続き型言語と比べ論理型言語の利点を十分に生かしていると言える。

さらに、ＧＰＳＳでは、２つのトランザクションの同期を取る必要があるモデルに対しては、内部構造に関する理解が必要となる。そこで、次にタクシー乗り場モデルの記述を考える。以下の仮定を設ける。

［１］　タクシーと客はそれぞれ独立に０～１０分の一様分布に従って到着する。

［２］　タクシーと客はそれぞれ別の待ち合わせ施設に先着順に並ぶ。客が待ち合わせ施設から出るのは、自分が待ち行列の先頭であり、かつタクシー待ち合わせ施設にタクシーがいるときである。タクシーの場合も同様である。

このモデルの記述を図４－８に示す。

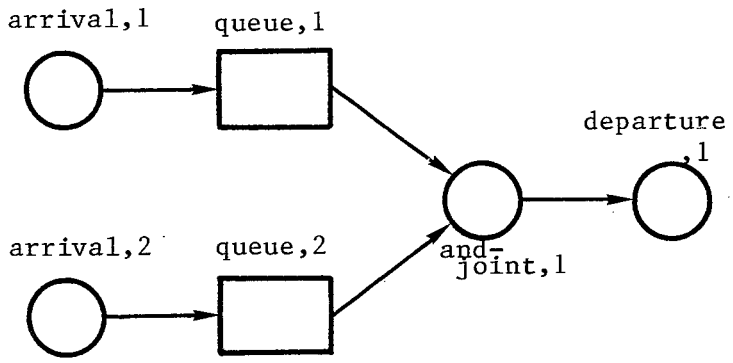


図４－７　タクシー乗り場モデル

```
/*      Taxi Stand Simulation in GPSS          */

GENERATE        0,10      |    GENERATE        0,10
QUEUE           1         |    QUEUE           2
SEIZE           1         |    SEIZE           2
GATE U          2         |    GATE U          1
DEPART          1         |    DEPART          2
PRIORITY        1         |    PRIORITY        1
RELEASE         1         |    RELEASE         2
TERMINATE                 |    TERMINATE

GENERATE        ,,1000,1
TERMINATE       1
```

```
/*      Taxi Stand Simulation in SILQ          */

interval(Id,Ti)      :-   s_random(10,Ti).
queue_out_condition(Id):-
          sys_customers_in_queue(1,[Cust1|R1]),
          sys_customers_in_queue(2,[Cust2|R2]).
buffer_capacity(Id,10000).
connect( 1,[arrival    ,1],[queue      ,1]).
connect( 2,[arrival    ,2],[queue      ,2]).
connect( 3,[queue      ,1],[and_joint  ,1]).
connect( 4,[queue      ,2],[and_joint  ,1]).
connect( 5,[and_joint  ,1],[departure  ,1]).
start_time(0).
end_time(1000).
```

図４－８　モデル記述例４

ＳＩＬＱではタクシーの待ち行列と客の待ち行列をそれぞれ別のプロセスと考え、２つの待ち行列から出てきた客とタクシーをJOINTプロセスによって結合すると考えると、図４－７のようにモデル化できる。各待ち行列からタクシーまたは、客が出られるのは、他方の待ち行列が空でないという条件が満たされたときである。これらがプログラミングプリミティブによって記述されている。

この例では、ＳＩＬＱでの記述１０行に対し、ＧＰＳＳでの記述は２０行程度であり、記述行数の点では２倍の差が現れる。この差は、ＳＩＬＱのデフォルト機能によるところが大きい。さらに、ＳＩＬＱでは、モデルの論理的性質を条件によって記述しているのに対し、ＧＰＳＳではタイミングを取るための本来モデルとは無関係なブロックの挿入が余儀なくされる。しかもこのブロックの挿入はＧＰＳＳの内部機構に関して熟知していなければ不可能である[4]。

４．４　本言語の記述性の評価

上記以外にもＬＡＮ等の記述を行いＳＩＬＱの記述性を検討した[33]。その結果次のことが分かった。

（１）Ｐｒｏｌｏｇの利点を損なうことなくモデルの定義をプログラミングプリミティブによって記述することが可能である。このことは、ＳＩＬＱによるモデル記述例からだけでなく、Ｔ－Ｐｒｏｌｏｇのようなプロログに時刻概念を導入した他の論理型言語との比較によっても明らかになった。これによりタイムオブジェクトによるプロセス間の時刻関係の内包がＰｒｏｌｏｇの宣言的記述性を保つ効果があることも明らかになった。

（２）タイムオブジェクトはユーザに理解しやすい具体的な概念を与えるためソフトウェア作成の際のみならず、モデル構築に対しても有効である。

（３）デフォルト機能によって記述行数が軽減される。

（４）同種プロセスの一括定義が可能であること並びに階層的構造が扱い易いこと等からモデルが複雑化した場合であっても、記述行数はあまり増加しない。

（５）従来のシミュレーション言語の中で最も使用されているＧＰＳＳとの比較

によって

a）客の受ける処理過程をタイムオブジェクトによって構成する点等の概念が類似しているため、ＧＰＳＳからの移行が容易であること

b）ＧＰＳＳでの煩雑なパラメータによるプログラミングに比べ、プログラミングプリミティブを用いたホーン節によるプログラミングはユーザの要求に柔軟に対処可能である。特に複雑なモデルを対象とする場合に有効であること

c）ＧＰＳＳで記述しにくい複数の客の流れの同期を取らねばならないモデルに対して、ＳＩＬＱでは、論理の組合せによって容易に記述が可能であること

が明かになった。

４．５　結言

本章では、ＳＩＬＱによる待ち行列網モデルの記述例を通してＳＩＬＱのプログラミング法を示した。

まず、待ち行列網モデルの簡単な例として公衆電話ボックスモデルをＳＩＬＱのプログラミングプリミティブで記述した。この例においては、ユーザが実際のシステムをモデル化し、その記述を行う過程を示しながら述べた。

また、実用的な例として、ウィンドウフロー制御を用いるパケット交換網の記述を行い、パケットが入力待ち行列から出られる条件等の記述にＳＩＬＱは向いていることを示した。

さらに、他のシミュレーション言語（Ｔ－Ｐｒｏｌｏｇ、ＧＰＳＳ）とそれぞれ１つの例を用いて比較し、Ｔ－Ｐｒｏｌｏｇに対しては、Ｐｒｏｌｏｇの宣言的記述性をＳＩＬＱが保っている点、ＧＰＳＳに対しては、複数のトランザクションの同期を取るような問題に対してＳＩＬＱが有効であることを示した。

第５章

ＳＩＬＱ逐次処理系

５．１　緒言

本章では、ＳＩＬＱを用いて実際にシミュレーションを実行することを目的にして、パーソナルコンピュータ上の逐次処理Ｐｒｏｌｏｇ[34]を用いたＳＩＬＱ逐次処理系を構成する。まず、本処理系のモジュール構成及びシミュレーション実行の手順を概説する。処理系の作成に当たっては、各プロセスの実行順序の制御及びシミュレーション時刻の更新といったシミュレーション制御と、第３章で述べたタイムオブジェクトの状態遷移のインプリメントのためにシステム述語とシステムリストを用意した。次に、各モジュールの機能及び各モジュール間における処理の流れについて詳説する。

最後に本処理系の評価とその改善案を提示する。

５．２　逐次処理系の概要

処理系の構成は図５－１に示されるように、プロセス生成モジュール、タイムオブジェクト実行モジュール、シミュレーション制御モジュール、統計データ収集モジュール、シミュレーション結果出力モジュールの５個のモジュールから成る。これらのモジュールはすべてＰｒｏｌｏｇによって記述されている。

本処理系では、まずプロセス生成モジュールがユーザプログラムによって示されたモデルからプロセスを生成する。シミュレーションは、現在の時刻に事象の発生するプロセスに対応するタイムオブジェクト実行モジュールをシミュレーション制御モジュールがコルーチン制御によって呼び出すことによって実行する。コルーチン制御の際には、各プロセスの状態、システム時刻や各プロセスの現在の状態の管理を行うためのテーブルであるシステムリストを用いる。また、タイム

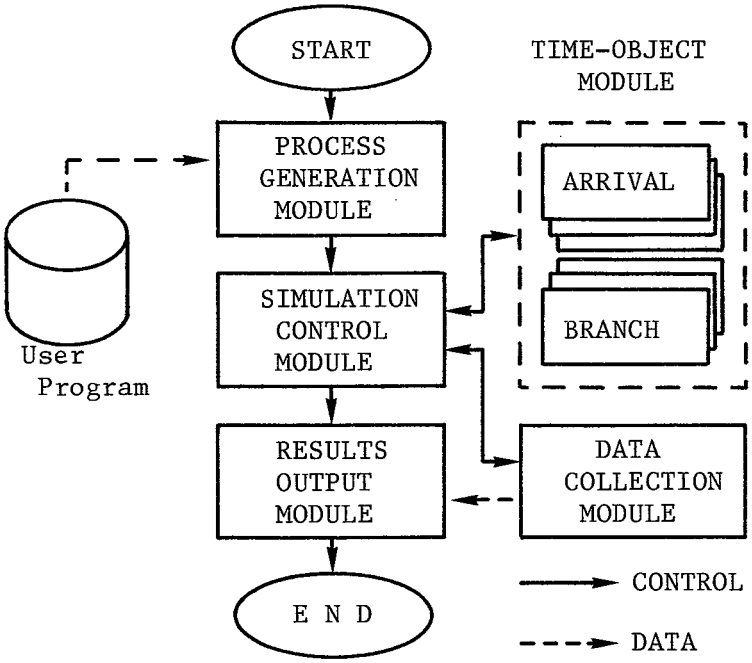


図５－１　ＳＩＬＱ逐次処理系の構成

オブジェクトの状態遷移についてはＴ－Ｐｒｏｌｏｇとほぼ同等の機能を持つ述語（システム述語）を作成し実現している。シミュレーション実行中に得られた統計データは、データ収集モジュールによって一時的に蓄えられ、シミュレーション終了時刻が来ると結果出力モジュールがデータを編集して出力する。

５．２．１　システム述語

それぞれのタイムオブジェクトの状態遷移を処理系内で記述するために、５種類の専用のシステム述語を用意した。システム述語はＴ－Ｐｒｏｌｏｇの組み込み述語に準じて作成した。

以下にこれらの機能を示す。

send(P,C) …… 目的プロセスPにメッセージCを送信する。

hold(T,P) …… シミュレーション時間Tの間、プロセスPは停止する。

wait_for(P) …… メッセージの到着まで、プロセスPは待機する。

time_wait(P)…… メッセージの到着または論理時刻が更新される直前まで、プロセスPは停止する。

wait(P) …… 決められた条件が満たされるまで、プロセスPは待機する。

5．2．2 システムリスト

モデル上のプロセスの実行制御のためにプロセス管理テーブルであるシステムリストを用意する。システムリストは、以下の項およびリストから成るリスト構造をしており、現在実行中のプロセス名や、停止しているプロセス名とそれに関する情報が記入されている。

システムリストの内容

1．	Active Process	現在実行中のプロセス名
2．	Next Process	優先的に次に実行するプロセス名のリスト
3．	Message Waiting List	wait_for によって、客の到着まで待機しているプロセス名のリスト
4．	Message List	sendによって各プロセスから送信された客と、その受信プロセス名のリストから成るリスト
5．	Holding List	holdによって、決められた時刻まで停止しているプロセス名と停止状態を解除される時刻を要素とするリストからなるリスト
6．	Time Waiting List	time_wait によって、客の到着または論理時

		刻更新直前まで待機しているプロセス名のリスト
7．	Condition Waiting List	waitによって決められた条件が満たされるまで待機しているプロセス名のリスト
8．	Simulation Time	現在のシミュレーション時刻

5．2．3　ステップ管理表

ステップ管理表は、現在各プロセスが対応するタイムオブジェクトのどの状態にあるかを示すためのものである。タイムオブジェクトの状態とステップの関係は、5．3．3に述べる。

尚、この表はシステムリストからも作成可能であるが、本処理系では処理効率向上と管理の容易さのために作成する。

ステップ管理表は、次のような形をしている。

step(P,S)　：　P……プロセス名
　　S……Pの現在のステップ番号

5．3　各モジュールの機能

以下に各モジュールの機能を述べる。

5．3．1　プロセス生成モジュール

プロセス生成モジュールは、ＳＩＬＱで記述されたユーザプログラムを読み込み、モデルに含まれるプロセスを生成する。そして、シミュレーションの開始時刻、および終了時刻の設定と共に以下に示すようなシステムリストの初期化を行う。

1．ユーザプログラムのモデルの構造の定義である、connect(各プロセス間の接続関係を記述する述語)の引数を参照しながらシステムリストのHolding Listに各プロセス名をプロセスが停止状態を解除される時刻と共に書き込む。尚この時刻は、初期化の段階ではユーザプログラムのシミュレーション開始時刻に設定する。

2．シミュレーション終了時刻にシミュレーション結果出力モジュールに制御を渡すための終了プロセスを Holding Listに加える。

3．シミュレーション時刻をシミュレーション開始時刻に設定する。そして、Holding List から先頭のプロセスを取り出し、Active Processに書き込む。

4．タイムオブジェクト実行モジュールで使用するステップ管理表を作成する。

尚、ＳＩＬＱでは定常状態のみを取り扱う。本処理系では、モデルの初期状態は全てのサーバ、バッファに客のいない状態とする。

本モジュールの実行終了後、制御はシミュレーション制御モジュールに渡される。

５．３．２　シミュレーション制御モジュール

現在のシミュレーション時刻に発生する事象を持つプロセスに対応するタイムオブジェクト実行モジュールに制御を渡す。そして、その時刻に発生する事象をすべて処理したときには次の事象が発生する時刻までシミュレーション時刻を更新する。

また、必要時には統計データ収集モジュールに制御を渡し、データを一時記憶する。

具体的には次の３つの動作が繰り返し行われる。

Ｓｔｅｐ１　現在の Active Process に対応するタイムオブジェクト実行モジュールにプロセス識別子と共に制御を渡す。

Ｓｔｅｐ２　タイムオブジェクト実行モジュールが、システム述語で実行を停止し、制御が返されるとそれぞれのシステム述語に対応して、以下のような内容

をシステムリストに書き込む。

send …… 送出される客 と 送出先のプロセス名を Message Listに書き込む。

hold …… 停止したプロセス名を停止状態を解除される時刻と共にHolding Listに書き込む。

wait_for …… 待機したプロセス名をMessage Waiting Listに書き込む。

time_wait …… 待機したプロセス名をTime Waiting List に書き込む。

wait …… 待機したプロセス名をCondition Waiting Listに書き込む。

Ｓｔｅｐ３　システムリストから次に実行すべきプロセス (Active Process) を次に述べるアルゴリズムによって選び出す。

５．３．２．１　プロセスの選択

プロセスの選択は、次のように行われる。まず、システム述語sendでプロセスが一旦中断し本モジュールに制御が渡された場合には、同じプロセスを選択する。

他のシステム述語 hold, wait_for, time_wait, waitでプロセスが停止または待機し、システムリストへの書き込みが終了すると、以下に示す優先順位のもとで新たなActive Processを選択する。

[１] Next Process にあるプロセスを選択する。

[２] Holding Listに現在のシミュレーション時刻に停止状態を解除されるプロセスがあれば、そのプロセスを選択する。

[３] Message Waiting List で客を待機中のプロセスに向けて送信された客が Message List にあれば、客を受信すると共にそのプロセスを選択する。

[４] Time Waiting Listで客を待機中のプロセスに向けて送信された客が Message List にあれば、客を受信すると共にそのプロセスを選択する。

[５] Condition Waiting List中のプロセスで指定された条件が満たされたものがあれば、そのプロセスを選択する。ただし、このリストに入るプロセスはQUEUEプロセスのみであるから、指定された条件とはqueue_out_conditionに相当する。

[６] Time Waiting List中にあるプロセスを選択する。つまり、時刻更新の直前まで条件を待つ必要のあるプロセスを選ぶ。これには、XOR－JOINTプロセスがある。

[７] 以上によって選択されるプロセスがない場合には、Holding Listから停止状態を解除される時刻が一番早いプロセスを選択し、シミュレーション時刻をその時刻に進める。

尚、選択されたプロセスがシミュレーションを終了させる終了プロセスの場合には、シミュレーション結果出力モジュールに制御を渡す。

５．３．２．２　強制バックトラック

シミュレーションでは、多くのサンプルを採集することによって初めて意味のある統計量が求められる。しかし、サンプルを多く採集しようとすると、通常のＰｒｏｌｏｇ処理系では解探索木が深くなりずぎ記憶領域が容易に溢れてしまう。

そこで本処理系ではいくつかのプロセスを処理した後、強制的にバックトラックを発生させ記憶領域を開放する方法を用いている。強制バックトラックの直前には、システムリスト等の情報を待避させ、バックトラック終了時にこの情報を基にシミュレーションを再開する。

５．３．３　タイムオブジェクト実行モジュール

タイムオブジェクト実行モジュールには、各タイムオブジェクトごとにある状態から次の状態へ遷移する際に処理する内容が、Ｐｒｏｌｏｇによって記述されている。この記述は、オブジェクトの待機状態または停止状態から次に待機ある

いは停止するまでに実行すべき処理を一つのステップとして書かれている。従って、各ステップの最後の述語はhold、time_wait、wait_for、waitの４つのシステム述語のいずれかであり、ステップ数はタイムオブジェクトによって異なる。また、タイムオブジェクト実行モジュールは、対応するプロセスごとにどのステップまで実行したかを記憶しておくステップ管理表を持っている。

シミュレーション制御モジュールから、客の到着、時間の経過等の条件が満たされたプロセスの識別子を受け取ったタイムオブジェクト実行モジュールは、ステップ管理表から検索したステップの処理を行う。各ステップにおいては、プロセス識別子と、ユーザプログラムにおけるモデルの構造及び動作に関する記述を参照しながら処理される。

処理が完了すると、次に実行するステップをステップ管理表に書き込み、シミュレーション制御モジュールに制御が返される。

５．３．４　統計データ収集モジュール

ユーザが指定した統計量の算出に必要な統計データを、客及びプロセスの履歴から抽出し一時的に保存を行うモジュールである。

シミュレーションにおける統計量は、各プロセスに関する統計と各客に関する統計の二種類に大別することができる。本モジュールは（１）各プロセスに関する統計についてはプロセスを通過した客の数、 QUEUEプロセスにおける待ち行列長、SERVERの利用率に関するデータを、そして（２）各客に関する統計についてはQUEUEプロセスでの待ち時間、客がARRIVALからDEPARTUREまでに費やした時間を保存する。

本モジュールは、（１）についてはシミュレーション時刻更新時に、（２）については客が退去する際、つまりDEPARTUREプロセスにおいて客の消去の処理を終えた時点に、シミュレーション制御モジュールから制御が渡されデータを抽出保存する。

実際のシミュレーションでは、各統計データの平均、分散、最大値等を求める

場合が多いが、本処理系ではパーソナルコンピュータ上のＰｒｏｌｏｇの逐次処理系の制約により各統計の平均値に関するデータについてのみデータ収集を行っている。

５．３．５　シミュレーション結果出力モジュール

シミュレーション終了時刻に到達するとシミュレーション制御モジュールから制御を受け継ぐ。そして、統計データ収集モジュールにおいて一時的に保存されていたデータから予めユーザが指定した統計量を算出し編集する。そしてこれをシミュレーション結果として出力し、シミュレーションを終了する。

本処理系では、Ｐｒｏｌｏｇ処理系における数値計算の処理能力が不十分であることからデータの出力のみにとどめ、最終的な統計量の計算は手作業に依っている。

５．４　シミュレーション実行例と処理システムの評価

ここでは、４．２．３で述べたウィンドウフロー制御を用いるパケット交換網のシミュレーションを本システム上で実行して得られた結果を示す。

表５－１　シミュレーション結果

	ＬＣ１	ＬＣ２
網内平均遅延時間(秒)	２．７９３	４．１９３
Ｉｎｐｕｔ Ｑｕｅｕｅでの 平均遅延時間(秒)	０．９４６	１．２８４
システム 全体での 平均遅延時間(秒)	３．７３４	５．４７６
スループット	０．３２５	０．４５６

このデータ自身には特に意味はないが、この値と他の数値計算例とを比較したところ、結果はほぼ一致しシミュレーションは正しく実行されていることが確認できた[35]。

しかしながら、本処理系の処理速度は非常に低速であり、上記例の場合１時間に採集できるサンプル数は約５００程度である。通常のシミュレーションが数十万のサンプルを採集していることを考えるとこの処理系は試験システムの域を出ない。そこで、処理速度の改善のためには、以下の２点が考えられる。

（１）強制バックトラックには多くの時間を要する。しかも、強制バックトラックの際に消去される情報は後のシミュレーション実行には不要であり、この情報の保存等は無駄である。

そこでＳＩＬＱ処理系のうち、バックトラックする可能性のある部分とない部分を明確にし、保存しなくてもよい情報は何かを把握する。そして、本処理系の

ような汎用Ｐｒｏｌｏｇ処理系ではなくＳＩＬＱ専用の処理系をＣ言語等の高速なコンパイラ言語を用いて記述し、逐次処理系の高速化を図る。

（２）本処理系の処理時間はモデルに含まれるプロセス数をｎとすればＯ（ｎ）になる。モデルに含まれている並列性を利用すればより処理時間の短縮が図れると考えられる。

Ｐｒｏｌｏｇの並行処理についてはホーン節単位、ゴール単位等種々の方法が開発されている。また、待ち行列網シミュレーションには極めて高いもかかわらず利用することが難しい並列性も存在する。この並列性の利用に関しては、第６章にて詳説する。

５．５　結言

本章では、第３章において提案した待ち行列シミュレーション専用言語ＳＩＬＱの処理系を、パーソナルコンピュータ上の逐次処理Ｐｒｏｌｏｇを用いて構成した。

本処理系は、プロセス生成モジュール、シミュレーション制御モジュール、タイムオブジェクト実行モジュール、統計データ収集モジュール、シミュレーション結果出力モジュールの５個のモジュールから構成されている。

まずタイムオブジェクトの状態遷移を停止・待機状態から次の停止・待機状態までの一連の処理を１ステップと考え、各オブジェクトの動作を幾つかのステップに分けてＰｒｏｌｏｇで記述することで構成した。この構成には、処理系専用のシステム述語、システムリスト及びステップ管理表を作成し用いている。

最後に本処理系の評価として非常に低速であるため試験システムの域を出ないことを示し、その対策として、ＳＩＬＱで記述されたソフトウェアの実行に必要な情報のみの保存を行う専用処理系、並びにモデルの並列性を利用した並行処理系が必要であることを示した。

第６章

ＳＩＬＱ並行処理系

６．１　緒言

待ち行列網シミュレーションの効率化を図るためには、ユーザにとって待ち行列網モデルを記述し易い言語とその処理系が必要である。

前章までに、待ち行列網シミュレーション専用論理型言語ＳＩＬＱを提案し、ソフトウェア記述に対して効率化を図った。また逐次処理系によるＳＩＬＱ処理系を構成した。しかしながら、ＳＩＬＱ逐次処理系ではモデルに含まれるプロセス数ｎに対してＯ（ｎ）の処理時間が必要とされ、実用に供するためには高速化を行わねばならない。

本章では、並行処理によってＳＩＬＱ処理系の高速化を図る。

一般にＰｒｏｌｏｇ形式で記述されたソフトウェアは、同じ頭部を持つホーン節毎並びに１つのホーン節の本体部の述語毎に並列性を有している。前者をOR並列、後者をAND並列と呼ぶ。近年、これら等の並列性に注目して並列Ｐｒｏｌｏｇマシンの設計が各所で行われている[17)-(20)(36)-(38)]。その中には４００ＫＬＩＰＳを越す超高速マシンもある。

本研究では、個々の節に関する並列性の利用はこれらの技術に委ねることにし、待ち行列網モデルに限定した場合のモデルの有する並列性を利用することを考察する。ＳＩＬＱで記述されたモデルの並列性とは、プロセス毎つまりプロセスを定義しているホーン節集合ごとの並列性を指す。

まず、ｕｎｋｎｏｗｎ型メッセージパッシングの概念を提案し、待ち行列網シミュレーションソフトウェアを並行処理する際に生じる問題点について述べる。

次に、ｕｎｋｎｏｗｎ型メッセージパッシングの一制御方式である先行制御方式を取り上げる。従来、本方式ではプロセッサ数に対して線形に処理能力（逐次処理系に対する並行処理系の速度改善比）が向上することが実験的に示されてき

たが、取り扱う時刻が単調に増加する実時刻と細かい増減を繰り返しつつ全体としては増加する論理時刻の２つを考慮しなければならないため、解析されていなかった。そこで、本方式を解析し、実験結果を裏付けると共に処理能力を左右する根元を検討する。

最後にＳＩＬＱで記述されたモデルを先行制御方式によって実行した場合にどの程度処理能力が改善されるかを示す。

尚、一般に並行処理において用いられるプロセス間通信（ＩＰＣ）[39]では”プロセス”を用いるが、本研究ではＳＩＬＱにおけるプロセスとの混同を避けるために、”プロセッサ”を用いることにする。

６．２　待ち行列網シミュレーションの並列性とその利用

プロセス単位の並列性を利用する場合、プロセスを割り当てられたプロセッサの同期を取り、結果の正しさを保証しなければならない。しかし、この同期は画像処理等処理の実行以前に処理計画が立てやすいジョブに対しては比較的容易であるが、待ち行列網シミュレーションのように高い並列性を有してはいるが各処理単位内に確率的不確定要素を含むジョブに対しては非常に困難である。これは、不確定要素を含むジョブを分割し並行処理する場合、次に示すｕｎｋｎｏｗｎ型メッセージパッシングを含む通信形態が存在するためである[40]。

まずいくつかの基本概念を示す。

プロセッサは無限にあるものとし１つのプロセッサにはいくつかのプロセスを割り当てられている。個々のプロセッサは固有の論理時刻を持っている。プロセッサは、メッセージを受信するとメッセージに処理を施し、その処理によって自分の論理時刻を更新する。他のプロセッサにメッセージを送信するときには、その時点でのプロセッサの論理時刻をメッセージに書き込んで送る。並行処理系がなんらかのジョブを実行し正しい結果を出力すると言うことは”プロセッサの論理時刻は単調に増加する”ことと同意である。そして、これを実現することがプロセッサの同期制御である。

６．２．１　プロセッサ間通信とメッセージパッシング

プロセッサ間通信を説明するためにまず、簡単な例を提示する。

図６－１のように論理時刻T_0を持つプロセッサP_1に論理的な入力経路が２本あったとする。

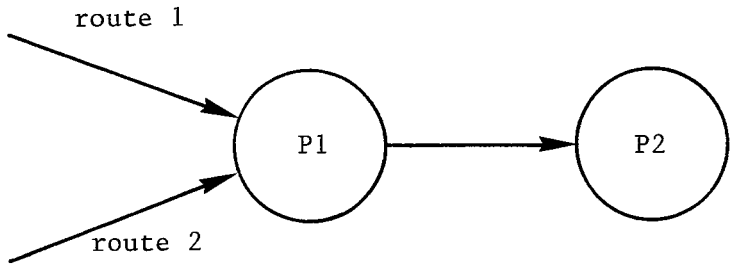


図６－１　プロセッサ間通信

経路１から論理時刻T_1（$T_1 > T_0$）を持ったメッセージM_1が実時刻t_1に到着したとする。

t_1におけるP_1の知り得る情報に関して次の分類が可能である。

（１）経路２からのメッセージM_2を”将来”（$t = (t_1, \infty)$なるｔにおいて)受信しない。

（２）M_2を”将来”受信し、かつM_2の持っている論理時刻T_2がわかっている。

（３）M_2を”将来”受信するが、T_2はわからない。

このいずれの場合でも時刻$t_1 + \varepsilon$（$\varepsilon > 0$）でM_1に対する処理を実行するか、M_2を待つべきかを決定できる。

例えば、（２）の場合、以下のようにして決定される。

$T_1 \leqq T_2$　：　M_1に対する処理を行う。

$T_1 > T_2$　：　M_2に対する処理は、M_1に対する処理よりも論理時刻上で早く行わなければならない。したがってP_1はM_2の受信を待つ。

この様に次ぎに実行する処理を決定できるジョブは、ジョブを分割し、プロセッ

サを割り当てる段階において、処理順序並びに同時に実行できる処理（Partial Ordering）が決定されていなければならない。この為、この種のジョブはジョブの実行以前に処理のスケジュールを行うことが可能である。この例としては、画素単位にプロセッサを割り当てる画像処理、配列計算等がある。

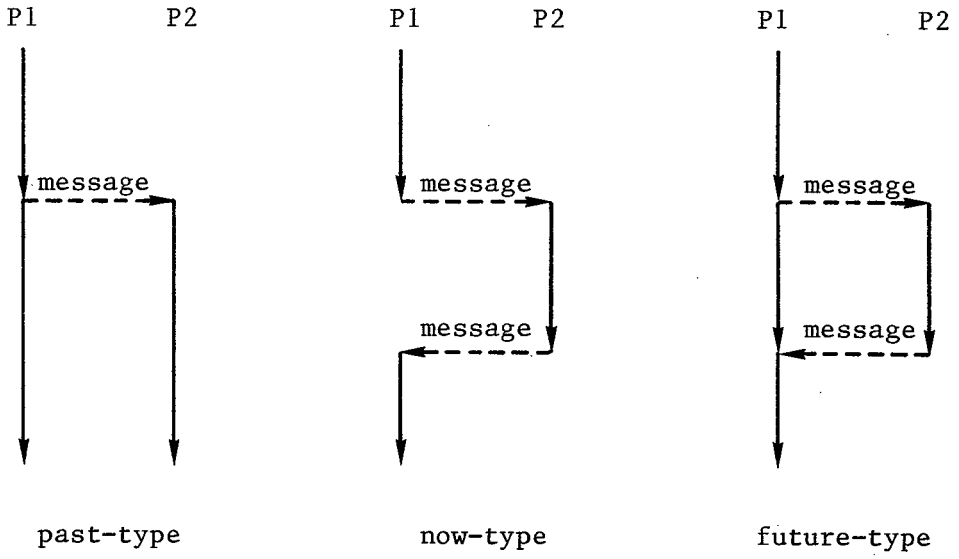
しかしながら、待ち行列網シミュレーションや分散データベースの更新といった確率的要素を含むジョブはジョブ分割並びにプロセッサ割当の段階では処理のスケジュールを立てることが不可能である。そして、本来持っているはずの並列性を利用しようとして分散化したとすると、例えば、経路２からのM_2の受信の有無とT_2の値が確率的になるため時刻$t_1+\varepsilon$においてM_1の処理を行うべきかM_2を待つべきか決定できない。

この様な状況は、自ら送信したメッセージが他に影響を及ぼし、その結果がひいては自らの処理にも影響するすなわち論理時刻上での順序関係が閉路を成すような場合に生じる。

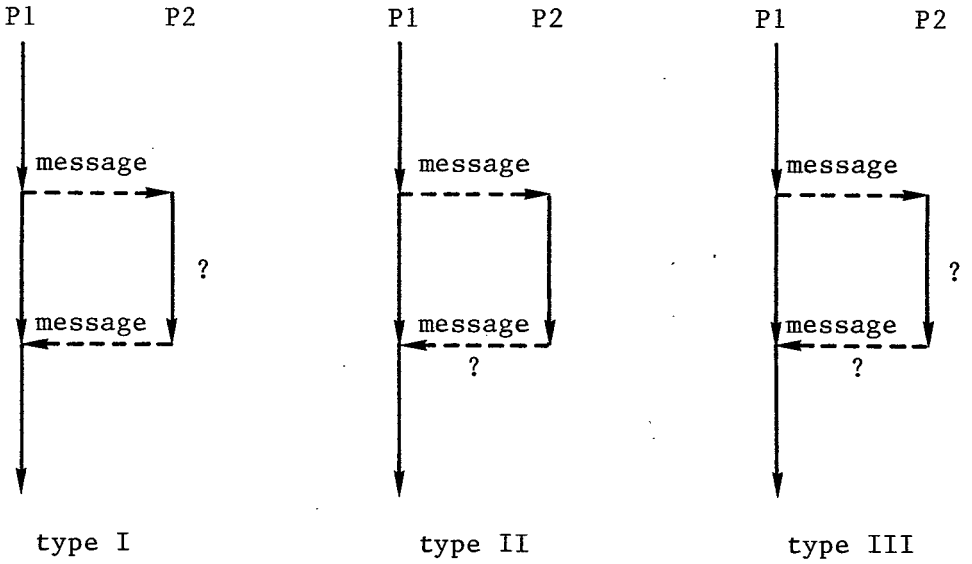
以上のことをｕｎｋｎｏｗｎ型メッセージパッシングとして一般化する。メッセージパッシングとは複数の通信し合うプロセッサがあった場合それらの間に生じる通信の形をＣ．Ｈｅｗｉｔｔらがｐａｓｔ型、ｎｏｗ型、ｆｕｔｕｒｅ型の３つに分類したものである。これらを、図６－２（ａ）に示す。

同図において、縦軸は実時刻をそして実線が論理時刻の更新を表している。各プロセッサは、実時刻と論理時刻からなる平面を持っているため正確さを期すためには３次元表現を用いなければならないが、ここでは、実時刻上でプロセッサが処理を行っているときにはそれに比例して論理時刻は増加するものとして考え、２次元で表す。（但し、比例係数はプロセッサの処理速度並びにメッセージの処理内容によって異なる。）

ｐａｓｔ型メッセージパッシングはプロセッサ１はメッセージ送信後プロセッサ２からメッセージを待つことなく処理を進める場合、ｎｏｗ型メッセージパッシングはプロセッサ１がメッセージを送信した後プロセッサ２からのメッセージを待つ場合、そしてｆｕｔｕｒｅ型メッセージパッシングはプロセッサ１はメッ



(a) known-type message passings



(b) unknown-type message passings

図６－２　メッセージパッシング
（ａ）ｋｎｏｗｎ型メッセージパッシング
（ｂ）ｕｎｋｎｏｗｎ型メッセージパッシング

セージ送信後いくつかの処理を実行した後プロセッサ２からのメッセージを受信する場合を指している。しかし、これらはプロセス１はプロセス２からのメッセージ受信があるかどうか、ある場合はいつ受信するのかを予め知っていなければならない。従ってこれらは、上記の事前にスケジュールが可能なジョブを分散化した場合に相当し、これをｋｎｏｗｎ型メッセージパッシングと呼ぶことにする。

一方分散化された待ち行列網シミュレーションではメッセージ受信の有無及びメッセージ受信の時刻はプロセッサ２で確率的に決定される。　これをｕｎ－ｋｎｏｗｎ型メッセージパッシングと呼び、図６－２（ｂ）に示す３種類の型が考えられる。

ここで、ｕｎｋｎｏｗｎ型メッセージパッシングの例として、図６－３に示すようなプロセッサ２からプロセッサ１に送られるメッセージは確率的に分岐するモデルがあったとする。プロセッサ１は、自分がメッセージを送ってから（プロセッサ１の論理的な時刻で）いつメッセージが帰ってくるかあるいはメッセージが帰ってくるかどうかが分からないため中断せざるを得ない。プロセッサ２からメッセージがリンク１を通りプロセッサ１に送られた場合にはその時点でリンク０からのメッセージ又はリンク１からのメッセージに対する処理の何れを行うべきかを決定できるが、プロセッサ２がメッセージをリンク２を通してモデル外に退去させたときにはプロセッサ１は中断したままである。この様な場合プロセッサ１を再開させるためには、何等かの制御を行わなければならない。この制御には集中制御方式と分散制御方式がある。

尚、以上の概念を表６－１にまとめる。

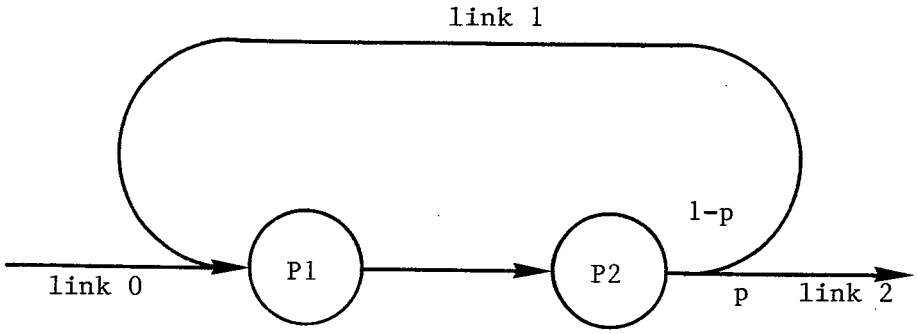
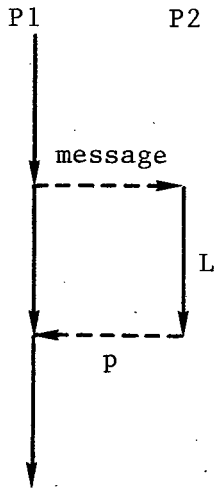


図６－３　２局フィードバックモデル

表６－１　メッセージパッシングの分類

p	L	メッセージパッシングの型
0	－－	ｐａｓｔ型
1	確定	ｎｏｗ型、ｆｕｔｕｒｅ型
	?	ｕｎｋｎｏｗｎ型Ｉ
0～1	確定	ｕｎｋｎｏｗｎ型II
	?	ｕｎｋｎｏｗｎ型III



６．２．２　集中制御方式

この方式は中央に時刻を管理する時刻管理装置を設ける方法であり、以下の２つが考えられる。

（１）事象駆動方式

この方式は以下のようなアルゴリズムによって論理時刻を制御する。

ａ）ある論理時刻にいかなる事象が発生するかを時刻管理装置が知っておりその事象の発生するプロセスを割り当てられているプロセッサを起動する。

ｂ）プロセッサは、起動されると処理を行い、その処理結果を返答として時刻管理装置に送る。

ｃ）時刻管理装置は、その時刻に生じる全ての事象の処理終了を起動した全てのプロセッサからの返答によって知ることができる。また、返答の内容によって新たな事象のスケジュールを行う。

この方法では、同一の論理時刻に発生する事象が多ければ、プロセッサが稼働している確率は高いが、そうでない場合にはどれか１台のプロセッサが動作していることになり効率面で問題が残る。そのため、この方式はあまり採用されていない。

（２）時刻駆動方式

事象駆動方式の欠点を克服するために、ある時刻からの一定幅（クロック）に含まれる事象を持つプロセスを割り当てられているプロセッサを起動する方式である。

この方式は、実装並びに制御が比較的容易なことからＮＴＴのＮＥＷＴＳ、阪大基礎工学部のＨＡＳＳ－ＱＮ等で利用されている。

また、この方式ではクロックを大きく取ればそれだけ同時に稼働するプロセッサが増加するが、１クロックに含まれる事象は同一論理時刻に生じるものとされるため、モデル上で事象がクロックよりも小さな間隔で生じる場合にはそれらの事象間の時刻経過は無視される。よって、結果の精度が保証されない可能性がある。

６．２．３　分散制御方式

これには、制御メッセージを送受信する方法と、一時的な矛盾を許容し、矛盾発生時にはこれを解消する方法がある。

（１）制御メッセージを用いる方法

これは、慶応大学のＫＤＳＳ等で採用されている方法である。前掲の６．２．１で述べた例を用いると中断したプロセッサ１を再開させるためにプロセッサ２がメッセージをモデル外に送り出すと同時にこの旨を知らせる制御メッセージをリンク１に送る。プロセッサ１は、制御メッセージが帰ってきた時点でリンク０からのメッセージの処理を行える。

処理を中断しているプロセッサ１がリンク１を通して受信するメッセージが制御メッセージの場合並びにリンク０から受信したメッセージの持つ論理時刻よりも大きな論理時刻を持ったメッセージをリンク１を通して受信した場合にはプロセス１は無駄な時間を費やしたことになる。

つまり、退去するメッセージが少ない場合には制御メッセージによって待たされる無駄な時間はさほど問題ではないが、制御メッセージが多い場合にはプロセス１とプロセス２はほとんど何れか一方が動作していることになる。すなわち逐次処理とあまり変わりがない。

（２）一時的な矛盾の発生を許す方法

これには、先行制御方式・Time-Warp Mechanism[41]等がある。これらはほぼ同等の制御方式であり、第２章において先行制御方式について述べたのでここでは詳しい説明は省く。この方式においては、いかにプロセッサの稼働率を高く保ちつつ矛盾の発生率を抑えるかが問題となる。

６．３　先行制御方式によるシミュレーション速度の改善

ｕｎｋｎｏｗｎ型メッセージパッシングを含む非同期ジョブを並行処理する方法としては、ＮＥＷＴＳ、ＨＡＳＳ－ＱＮ、ＫＤＳＳ等があるが、これらでは、

待ち行列網モデルの持つ並列性を十分に利用しているとは言えない。ここでは、一時的な矛盾の発生を許容するＤ－ＳＳＱの先行制御方式を採用した場合について考察する。

６．３．１　並行処理待ち行列網シミュレータＤ－ＳＳＱ

先行制御方式の実験システムであるＤ－ＳＳＱを図６－４に示す。同図においてＣＰはシミュレーションの実行、割り当てられたプロセスに関する統計収集、ＭＰはマンマシンインターフェース、通信制御等の機能を有する。

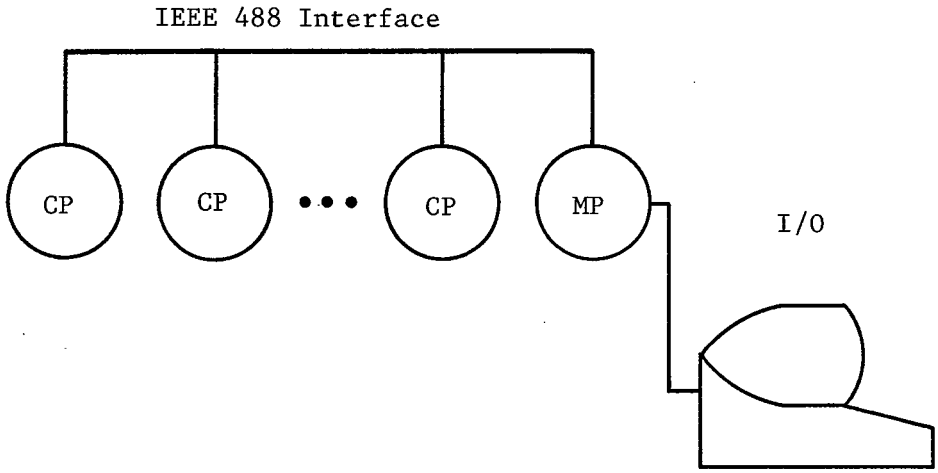


図６－４　実験システムＤ－ＳＳＱの構成

先行制御方式は、モデルの複雑度に対して線形に処理速度改善が得られること、しかし矛盾発生率が大きいこと等が問題となりその大きさはさほど大きくないことがＤ－ＳＳＱでの実験によって示されている。また、プロセッサの進みすぎを抑え矛盾発生を低減させる規制先行制御方式の有効性も実験によって示されている[21]。

規制先行制御方式では、各プロセッサは、ある時刻（これを規制時刻と呼ぶ）以上に時刻を進めようとしたときに進み過ぎと判断し、動作を中断し、他のプロセッサの時刻が進むのを待つ。この待っている状態を遊休状態と呼ぶ。

規制時刻の設定は、確定時刻に固定値（規制値と呼ぶ）を加えたものとする。この設定並びに確定時刻検出はＭＰが行う。

６．３．２　先行制御方式の処理能力解析[42]

先行制御方式並びに規制先行制御方式の処理能力は、実験的に示されているが、論理時刻の増減を伴う複雑な制御であるため従来解析されていない。

そこで、先行制御方式が規制値が無限大である規制先行制御方式に帰着できるためここでは規制先行制御方式を用いたＤ－ＳＳＱの処理能力を近似的に解析する。解析では、実時刻とシミュレーション時刻の２つを考慮しなければならない。

以下実時刻に関する記号は小文字ｔで、シミュレーション時刻に関する記号は大文字Ｔで表す。また、対比用の１プロセッサシミュレータをＳ－ｓｙｓｔｅｍと呼ぶ。各記号の添え字１はＳ－ｓｙｓｔｅｍを、ＭはＤ－ＳＳＱを表す。

Ｓ－ｓｙｓｔｅｍを用いて待ち行列網シミュレーションを行った場合にシミュレーション時刻がどの様に更新されて行くかを示したものが図６－５である。同図の小文字はノード１の内部で生じた事象、大文字は他ノードからノード１への通信によって生じた事象を表す.この様に実時刻が増加するに従ってシミュレーション時刻は階段状に増加する。一方、Ｄ－ＳＳＱのプロセッサ１（ノード１が割り当てられているプロセッサ）におけるシミュレーション時刻の更新状況は図６－５（ｂ）のように凹凸のある時刻更新過程となる。これはＤ－ＳＳＱでは一度実行した処理を取り消してやり直すことがあるからである。しかし、十分に長い実時間観測すれば全体としては、実時刻に対しシミュレーション時刻は増加する。図６－５（ｂ）を用いてＤ－ＳＳＱの時刻更新過程を説明する。

時刻０からシミュレーションを開始し、a, ｂの事象処理を実行した後他のプロセッサから事象Ａの処理要求が送信されたとする。このときこのプロセッサの現在時刻はT_bになっている。Ａの処理を実行すべき時刻T_Aは、T_bよりも小さい、つまり、過去であるため時刻矛盾が生じキャンセル処理を行う。キャンセル処理終了後事象Ａの処理から実行を再開し、ｂ'，ｃ，ｄと進んだとする.このとき、

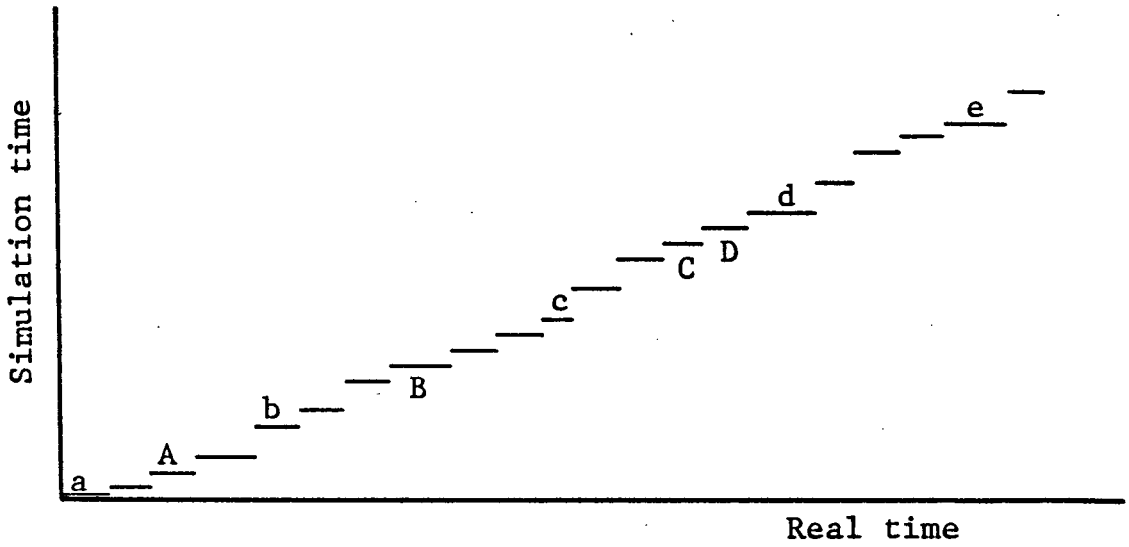
b'はAの影響を受けることがあるため必ずしもbとは処理内容が一致しないが、ここでは、処理内容は等しく、処理に要する実時間及び処理によって更新されるシミュレーション時間は同じであると考える。dの処理実行後、遊休状態となっているのは次の事象eの処理を行うと規制時刻を越えるためである。この遊休状態の間、プロセッサ1は他のプロセッサのシミュレーション時刻が進むのを待っている。

上記のようにして実時刻tまで進んだときには、シミュレーション時刻はT_Mまで進んでいる。tまでに実行された処理の内有効な処理はa，b'，c''，d'''，A，B，C，Dに対する処理である。e，f，g等は無駄な処理となったわけであり、これらを無効処理と呼ぶ。また、b，cの処理は2回、dは4回繰り返して実行されている。

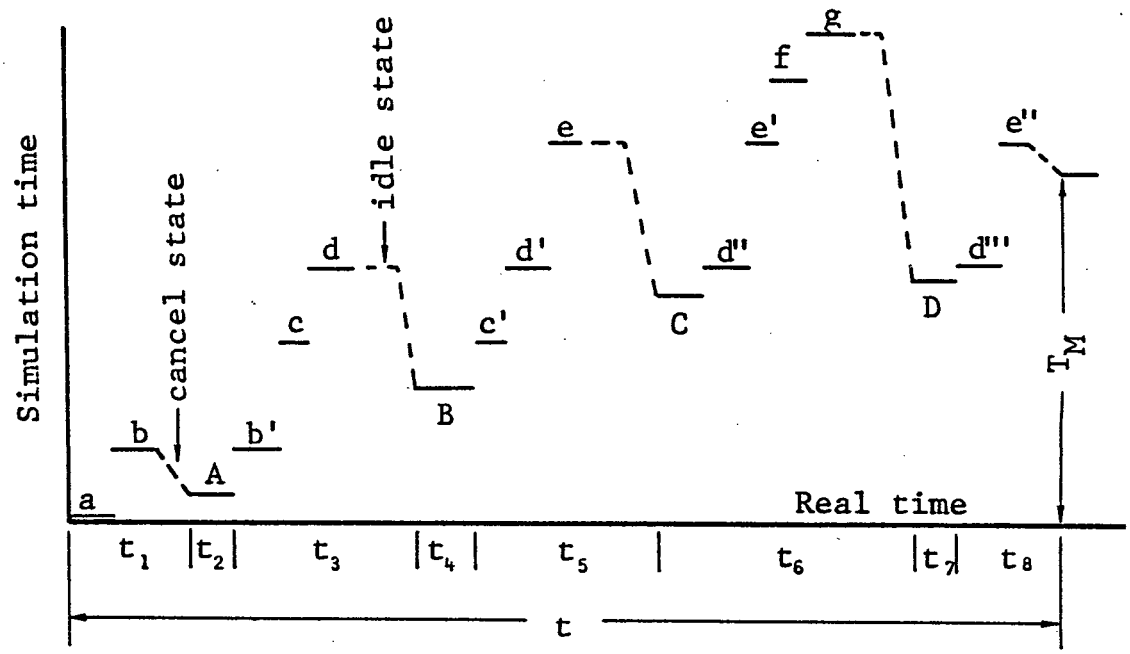
表6－2　記号の定義

	S－system	D－SSQ
シミュレーション実行実時間	t	t
時間tに進むシミュレーション時間	T_1	T_M
1処理（※）に要する平均実時間	t_1	t_M
1処理（※）によって更新されるシミュレーション時間	T_{a1}	T_{aM}

※　D－SSQにおいては1有効処理を意味する。



(a)



(b)

図６－５　シミュレーション時刻更新過程
(a) １プロセッサシミュレータ
(b) Ｄ－ＳＳＱ

もっとも上記の”有効処理”が本当に有効であるためには、時刻ｔにおいて確定時刻がT_dを越えている必要があるが、ここではこの条件は満たされているとする。

処理能力を解析するに当たってＳ－ｓｙｓｔｅｍとＤ－ＳＳＱにおいて同じＮノードのモデルを実時間ｔだけ実行した場合の記号を表６－２に定義する。Ｄ－ＳＳＱ中のプロセッサ数はＮとし、１プロセッサに１ノードが割り当てられているとする。

実時間ｔ内に実行される処理数はＳ－ｓｙｓｔｅｍでは

$$\frac{T_1}{T_{a1}} = \frac{t}{t_1} \qquad (1)$$

Ｄ－ＳＳＱでは

$$\frac{T_M}{T_{aM}} = \frac{t}{t_M} \qquad (2)$$

Ｄ－ＳＳＱの処理能力Ａは、（０，ｔ）に更新されるＳ－ｓｙｓｔｅｍのシミュレーション時間に対するＤ－ＳＳＱのシミュレーション時間の比と考えられるから、（１）（２）を用いて、

$$A = \frac{T_M}{T_1} = \frac{t_1 \cdot T_{aM}}{t_M \cdot T_{a1}} \qquad (3)$$

となる。

Ｄ－ＳＳＱ上に$T_{aM}=N \cdot T_{a1}$とジョブを展開できる、すなわち、Ｄ－ＳＳＱ

の各プロセッサにおける平均シミュレーション時刻更新間隔は同一で負荷はN分の1になるとすれば

$$A = \frac{N \cdot t_1}{t_M} \qquad (4)$$

図6－5（b）に戻るとtは、

$$t = \sum_{j=1}^{8} t_j = (t_a + 2t_b + 2t_c + 4t_d + 3t_e + t_f + t_g) + (t_A + t_B + t_C + t_D) + (\text{キャンセル時間}) + (\text{遊休時間})$$

t_j ： j番目の処理に要した実時間 (5)

と表される。

図6－5（b）をシミュレーション続行中の任意の区間（τ，$\tau + t$）であると考えると、e，f，gに対する処理に要した時間は時刻τ以前のa，b等に対する処理（その後無効となった処理）に要した時間に相当すると考えられる。従って、一般化すれば次式となる。

$$t = \sum_{j=1}^{k} n_j \cdot r_j + \sum_{j=1}^{n_c} r_{cj} + \sum_{j=1}^{n_i} r_{ij} \qquad (6)$$

ただし、k　：　有効処理数

n_j ：　j番目の事象の処理の繰り返し回数

r_j ：　j番目の事象の処理に用した実時間

n_c ：　キャンセル回数

r_{cj}：　j番目のキャンセル処理に要した実時間

n_i ：　遊休状態になった回数

r_{ij}：　j番目の遊休状態で費やした実時間

上式より、１有効処理に要する平均実時間t_Mは、

$$t_M = \frac{t}{k} = n \cdot t_{M0} + n_c \cdot \frac{t_c}{k} + n_i \cdot \frac{t_i}{k} \qquad (7)$$

ただし、$n = \overline{n_j}$, $t_{M0} = \overline{r_j}$, $t_c = \overline{r_{cj}}$, $t_i = \overline{r_{ij}}$

$\overline{\ }$は、jについての平均を表す

これを（4）に代入して、

$$A = \frac{N \cdot t_1}{n \cdot t_{M0} + n_c \cdot \frac{t_c}{k} + n_i \cdot \frac{t_i}{k}} \qquad (8)$$

式（8）の分子はプロセッサがN倍になったために得られる最大処理能力を意味している。Ｄ－ＳＳＱでは、同じ処理を繰り返すために１つの有効処理に$n \cdot t_{M0}$かかる。この影響が分母第１項に現れている。分母第２項、第３項はそれぞれキャンセルと遊休状態の影響を示している。

Ｄ－ＳＳＱでは、事象処理に履歴保存等の付加処理が含まれているため、Ｓ－

ｓｙｓｔｅｍより事象処理実時間は多くかかる。そこで、αをＳ－ｓｙｓｔｅｍにおける１事象処理時間に対するＤ－ＳＳＱにおける１事象処理時間比（$t_{m0}=\alpha \cdot t_1$）とする。またn_tを有効処理数と無効処理数の和（$n_t=k \cdot n$）、p_cを１処理を行った後に時刻矛盾が生じる確率（$p_c=n_c/n_t$）、p_iを総処理数（n_t，n_c，n_iの和）に対する遊休状態回数（n_i）の比、すなわち、

$$p_i = \frac{n_i}{n_t + n_c + n_i} \qquad (9)$$

とすれば、式（８）は次のように変形できる。

$$A = \frac{N}{n \cdot \left\{ \alpha + p_c \cdot \frac{t_c}{t_1} + p_i \cdot \frac{1+p_c}{1-p_i} \cdot \frac{t_i}{t_1} \right\}} \qquad (10)$$

となる。

上式のｎ，α，p_c，t_c/t_1，p_i，t_i/t_1について以下の仮定を設け、解析した。詳しい導出は付録２に示す。

（１）確定時刻は全てのプロセッサのシミュレーション時刻の最小値に等しく瞬時に求められる。

（２）事象を処理して更新されるシミュレーション時刻の幅は、平均１／μシミュレーション秒の指数分布にしたがう。

すなわち、ｘをシミュレーション時刻幅、p_eをｘの確率密度関数とすれば、

$$p_e=\mu \cdot \exp(-\mu \cdot x)$$

（３）規制値をＴとする。

ここで式（９）のＤ－ＳＳＱの処理能力Ａは、

$$A = A(T, \mu; \alpha, \beta, \eta_0, \eta_1, P_r, b_0) \quad (11)$$

ただし、β　：無規制時におけるの２つのキャンセルの間に実行される処理数

η_0, η_1　：キャンセル実時間に関する係数

P_r　：無規制時における時刻矛盾発生確率

b_0　：時刻矛盾発生確率に関する係数

と表された。

ここで、規制値Ｔは、シミュレーションモデルのμに依存するためＴの代わりにＫ＝μ・Ｔなる規制処理数を用いる。Ｋは規制値Ｔ内に平均して存在する処理の数を意味する。Ｋを用いると

$$A = A(K; \alpha, \beta, \eta_0, \eta_1, P_r, b_0) \quad (12)$$

となる。

尚、式（１２）において（$\delta A / \delta K$）＝０となるＫが最適規制処理数である。

各パラメータは次のようにして測定できる。

Ｓｔｅｐ　１：Ｓ－ｓｙｓｔｅｍ とＤ－ＳＳＱのプログラムステップ数よりαを求める。

Ｓｔｅｐ　２：無規制（Ｋ＝∞）においてβ，P_r，η_0，η_1を測定する。

Ｓｔｅｐ　３：適当な規制処理数を設定しb_0を求める。

式（１２）中のパラメータをＤ－ＳＳＱの実験システムによって測定した。その結果、プロセッサ数に対してこれらは安定していることが実験的に判明し、α＝3.3、β=5.0、P_r=0.1、η_0=0.42、η_1=8.1、b_0=0.13であった。図６－６にパラメータのうちη_0、η_1の測定例を示す。・は実験システムによって、そして実線は最小２乗法によって求めた１回のキャンセルによって戻った処理の数とそのキャンセル処理に要した実時間の関係である。

図６－７に最適規制処理数を与えたときのプロセッサ数に対する処理能力を表

す。同図から、プロセッサ数に対して線形に処理能力が向上することを解析的に示すことができた。また式（8）からも分かるように時刻矛盾の発生率P_cとαを小さく抑えることがこの処理能力を左右することが分かる。

尚、本解析の結果は、D－SSQでの実験結果とほぼ一致しその妥当性が確かめられた。

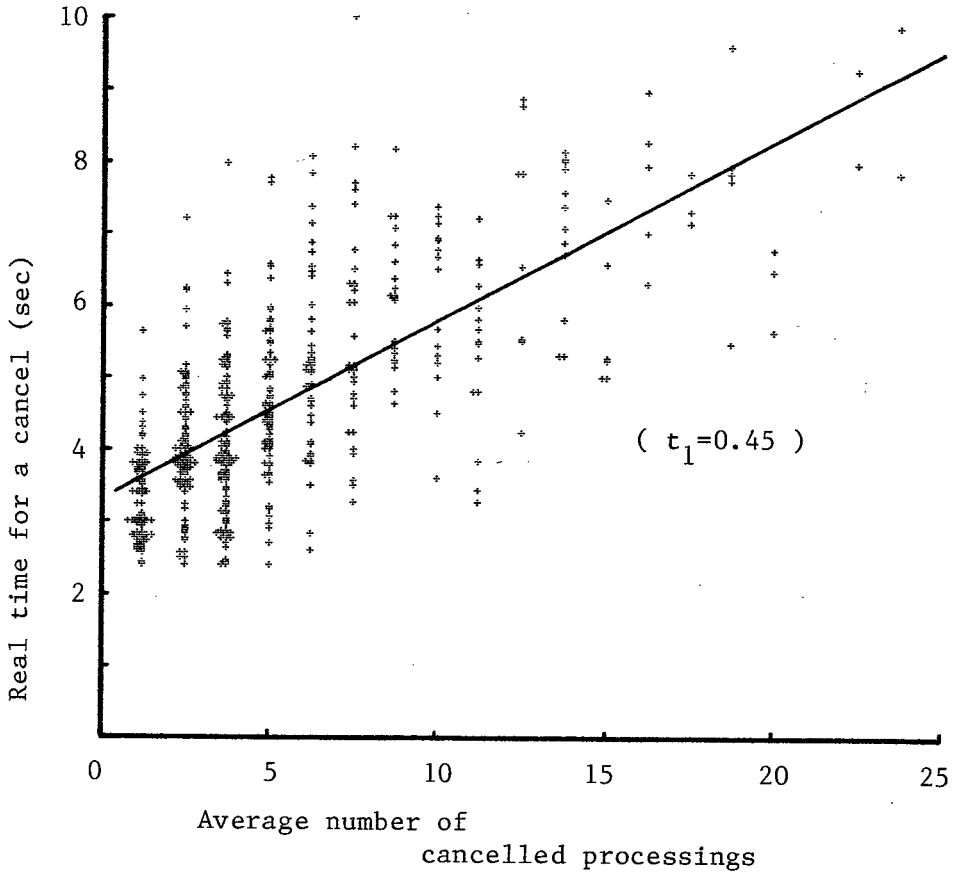


図６－６　パラメータ測定例（η_0、η_1）

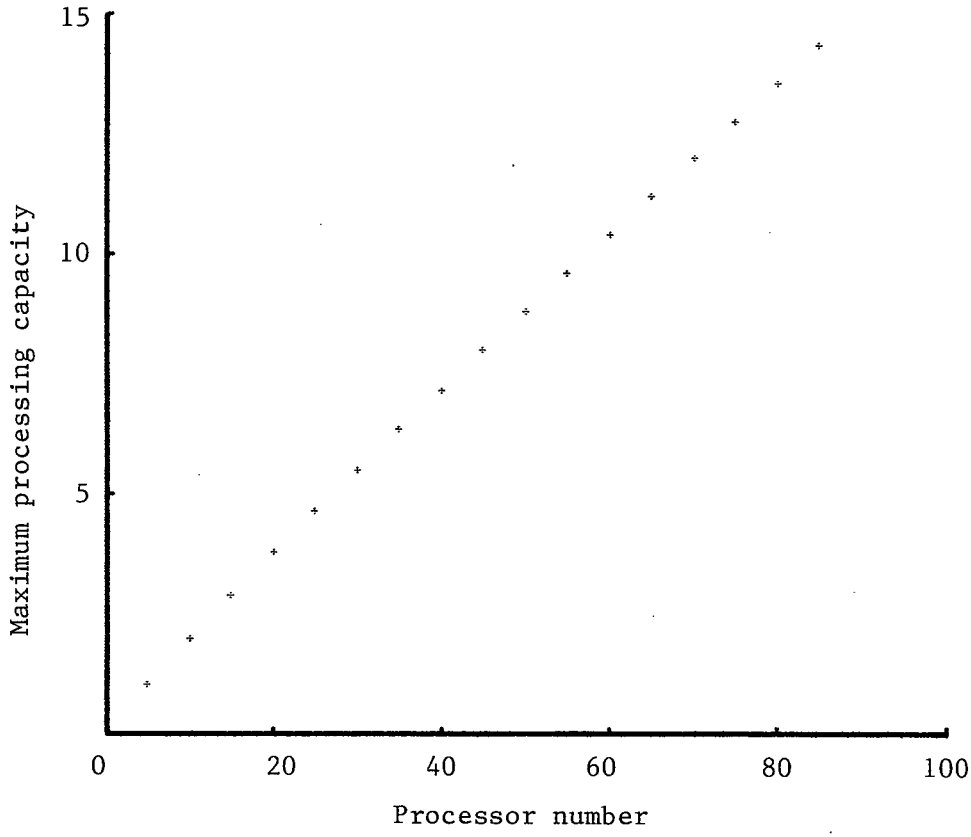


図６－７　最適規制処理数を与えたときの処理能力

６．３．３　ＳＩＬＱ並行処理への応用

前項までで解析した結果を用いてここでは、ＳＩＬＱで記述されたモデルを先行制御方式によって実行した場合どの程度処理改善が得られるかについて検討する。

先行制御方式の実験システムであるＤ－ＳＳＱでは均一網のみを扱える。そこで、ＳＩＬＱで次のようなモデルを記述した。

（１）Ｎ局が環状に接続されている。

（２）各局への客の到着、各客のサービス時間は全て同一の確率分布である。

Ｎ＝３の場合のモデルをタイムオブジェクトで構成したものを図６-８に示す。

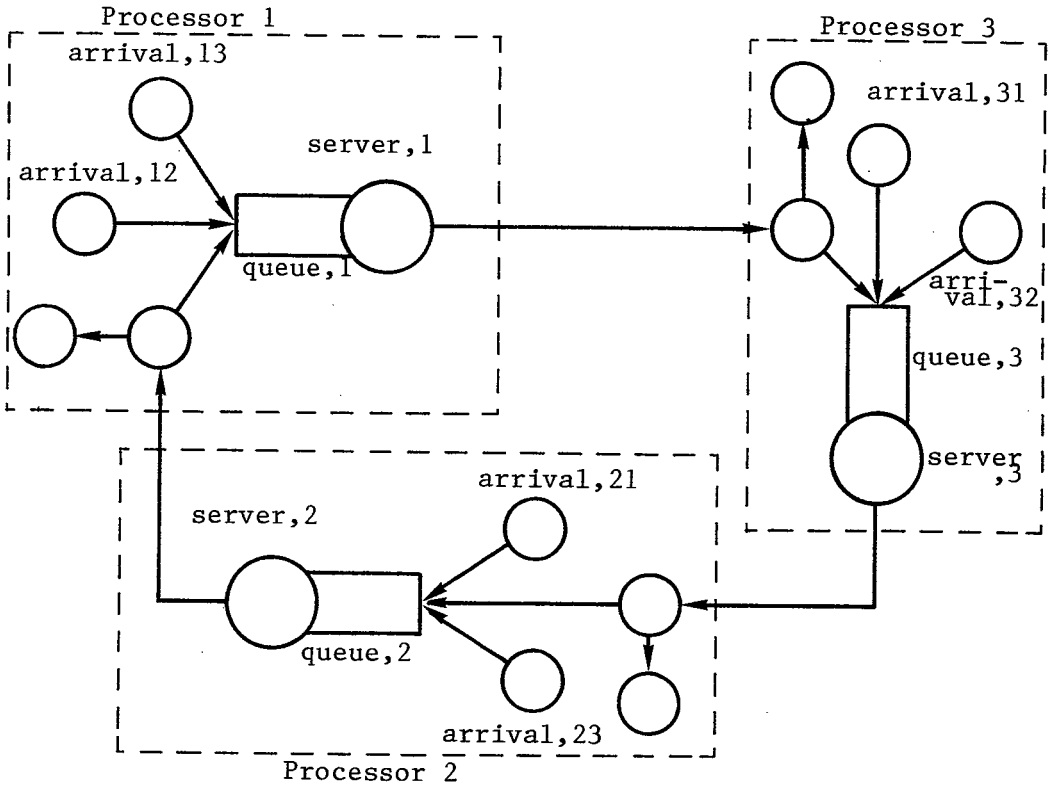


図６－８　Ｎ局環状網モデル　（Ｎ＝３）

同図において破線で示した部分を３台のプロセッサに割り当てた場合の処理速度改善率を図６－９に示す。

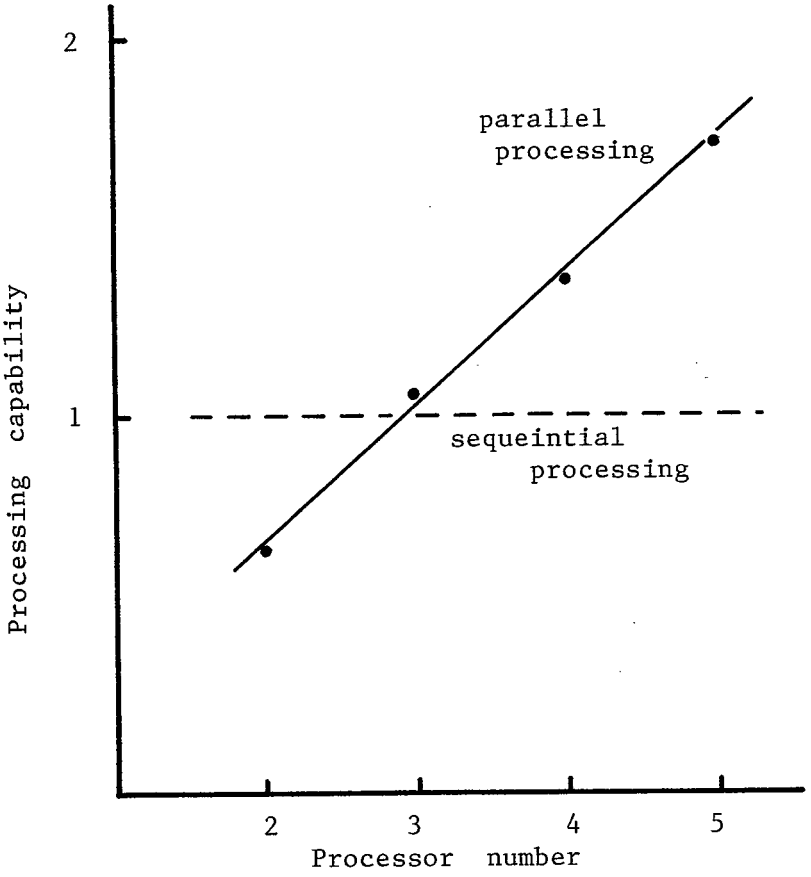


図６－９　並行処理による処理能力改善率

同図から、均一なモデルに限れば局数Ｎに対してＯ（Ｎ）の処理能力が得られることが分かった。ここで、Ｎ局モデルには発信局目的局対ごとにARRIVAL、DEPARTUREプロセス等が含まれているためプロセス数ｎはＯ（N^2）となる。このプロセス数をモデルの複雑度と考える。複雑度ｎのモデルのシミュレーションを行ったとき逐次処理系ではＯ（ｎ）の処理時間が必要である。これに対し、同図に座

標変換 $n = N^2$ と

$$\text{並行処理系の処理時間} = \frac{\text{逐次処理系の処理時間}}{\text{処理能力}}$$

の変換を施すことによって先行制御方式を用いた並行処理系では$O(\sqrt{n})$で処理が完了することが判明した。

但し、この結果には以下の条件が必要である。

a）プロセッサ間通信に要する時間は、プロセッサからのメッセージ送信間隔に比べて十分小さいこと

b）確定時刻を求めるのに要する時間は、プロセッサの処理時間に対して十分小さいこと

c）モデルが均一であること

これらの条件のうちa）及びb）については、十分実現可能なものであると考えられる。また、ここでは図６－８の破線部分を１つのプロセッサに割り当てたが、プロセスとプロセッサを１対１に割り当てればより処理時間の低減が期待できる。この場合、プロセッサの負荷が均等でなくなるためモデルが不均一な場合についてのさらに詳しい考察を必要とする。

６．４　結言

本章では、ＳＩＬＱの並行処理系について考察した。Ｐｒｏｌｏｇの並行処理としては、種々の方法が考えられているが、ここでは、プロセス単位の並列性に限って議論した。

並行処理に先立って、ジョブ分割、プロセッサ割当の段階では同期を取ることが困難であることをｕｎｋｎｏｗｎ型メッセージパッシングを提案し説明した。さらに、ｕｎｋｎｏｗｎ型メッセージパッシングを含むジョブの時刻制御方式である先行制御方式、並びに先行し過ぎを抑える規制先行制御方式を示し、これによってＳＩＬＱのモデルを並行処理した場合の逐次処理系に対する速度改善率を規制先行制御方式の解析並びにＤ－ＳＳＱの実験によって示した。その結果、均

一なモデルに限定し、プロセッサ間通信が十分高速であるという仮定の基においては、モデルの複雑度nに対して逐次処理では$O(n)$の処理時間を要し、一方並行処理系では、$O(\sqrt{n})$で表される時間によって実行できる可能性が有ることを示唆した。

第７章

結　論

今日の高度情報化社会の構築を根底から支えてきた分散処理型システムは、今後も益々発展して行くことは想像に難くない。分散型システムの性能評価は一般社会からはあまり目だたない存在であるかも知れないが、このことを考慮すれば将来においてより重んじられて行くと思われる。

また、ＶＬＳＩ等のハードウェア技術の発展に伴い、計算機能力が飛躍的に向上しつつあること及び理論解析手法がその対象とできる範囲を広げる速度に比して分散処理型システムの発展が速いことが予想されることから計算機を用いたシミュレーションに対する要求は高まって行くであろう。

現在のシミュレーションは主にシミュレーション専用言語によって行われているが、そのソフトウェア記述性並びに実行効率は必ずしもシステム解析者の要求を満足しているとは言い難い。

そこで本研究では、効率的なシミュレーションの実現を待ち行列網シミュレーション専用論理型言語ＳＩＬＱの開発とその処理系の観点から考察した。

ＳＩＬＱは、待ち行列網モデルを定義するシミュレーション仮定が条件によって記述されることが多いことと、論理型言語Ｐｒｏｌｏｇが高い記述性を持っていることに注目し待ち行列網モデルをＰｒｏｌｏｇ形式で定義する言語である。

Ｐｒｏｌｏｇは、第一階述語論理中のホーン節をソフトウェアとみなす言語であり、従来の言語がアルゴリズムすなわち、処理手順を記述していたのに対し、解くべき問題の論理的部分に注目し、事物間の論理的関係を記述することによってソフトウェアを作成できる。その反面、Ｐｒｏｌｏｇでは、”時刻”という共通媒体を持った複数のプロセスの表現が不可能である。

本研究では、Ｐｒｏｌｏｇの利点を損なわずに時刻概念を導入するためにプロセスの状態遷移をパターン化したARRIVAL、DEPARTURE、QUEUE、SERVER、JOINT、BRANCHの６種類のタイムオブジェクトを提案した。

そして、各プロセスの接続関係、各プロセスの動作内容、シミュレーション開始時刻及び終了時刻の設定、統計出力をＰｒｏｌｏｇ形式で記述するためにソフトウェアの最小構成要素であるプログラミングプリミティブを用意した。

次にＳＩＬＱの記述性の検討を、ＳＩＬＱによる記述例並びに他のシミュレーション言語との比較によって行った。その結果をまとめると次のようになる。

（１）タイムオブジェクトによる視覚的かつ具体的なモデル構築概念を提供しているため、ユーザが容易に対象システムをモデル化できる。

（２）待ち行列網モデルの定義を３０個のプログラミングプリミティブによって宣言的に記述することが可能である。このことは、ＳＩＬＱによるモデル記述例からだけでなく、Ｔ－ＰｒｏｌｏｇのようなＰｒｏｌｏｇに時刻概念を導入した他の論理型言語との比較によっても明らかになった。

（３）デフォルト機能によって記述行数が軽減される。

（４）同種プロセスの一括定義が可能なこと及び階層構造が扱えることからモデルが複雑になっても記述行数はほとんど増加しない。

（５）従来のシミュレーション言語の中で最も広く使用されているＧＰＳＳとの比較すると以下の利点を持つ。

ａ）客の受ける処理過程をタイムオブジェクトによって構成する点等の概念が類似している。そのため、ＧＰＳＳからの移行が容易である。

ｂ）ＧＰＳＳでの煩雑なパラメータによるプログラミングに比べ、プログラミングプリミティブを用いたホーン節によるプログラミングはユーザの要求に柔軟に対処可能である。特に複雑なモデルを対象とする場合に有効である。

ｃ）ＧＰＳＳで記述しにくい複数の客の流れの同期を取らねばならないモデルに対して、ＳＩＬＱでは、論理の組合せによって記述が可能である。

次に、ＳＩＬＱ逐次処理系を作成しＳＩＬＱ処理環境の実現を図った。逐次処理系の構成においては、まずタイムオブジェクトの状態遷移を記述するシステム述語を用意し、モデルを構成する複数のプロセスのそれぞれに対応するタイムオ

ブジェクト実行モジュールをシステムリストとステップ管理表で制御しながら逐次動作させる方法によってシミュレーションを実行する。

しかし、本逐次処理系はパーソナルコンピュータ上の1kLIPSのＰｒｏｌｏｇ処理系を用いているため実行速度が遅いという欠点を持つ。そこで、逐次処理系の処理速度改善案として、処理速度の早いＣ言語等を用いたＳＩＬＱ専用のＰrolog処理系の構成の可能性を示した。しかし、高速シミュレーションへのユーザの要求に答えるためにはモデルの並列性を利用する並行処理系の実現が不可欠である。

Ｐｒｏｌｏｇの並行処理に関しては、種々の面から検討されているが、本研究では、プロセス単位の並列性を利用する方式を用いた。待ち行列網シミュレーションが容易に分散化できない理由をｕｎｋｎｏｗｎ型メッセージパッシングの概念を提案し示した。そして、ｕｎｋｎｏｗｎ型メッセージパッシングを含むジョブを並行処理する際に問題となるプロセッサの時刻同期方式を先行制御方式によって行った場合どの程度の処理能力改善が行えるかについて本方式を解析し検討した。その結果、均一なモデルの場合に限って言えば、モデルの複雑度に対して線形に処理能力が改善されることを示し、高速シミュレーションの可能性を示唆した。従来のＰｒｏｌｏｇ並行処理技術と本研究で述べたプロセス単位の並列性の利用する技術を併用すればさらに一層の高速化が期待できる。

近年、待ち行列網解析ツールが登場しつつある[43]-[45]。これらのツールでのモデル記述は、ツールが用意した表をユーザが埋めるパラメータ指定型である。この方式は、パラメータによって定義できるモデルに対しては有効である。しかし、ツールが用意していないものに対しては、扱えないか扱えても手続き型言語を用いて行わねばならない。本研究で示したＳＩＬＱは、プログラミングプリミティブの中の引き数というパラメータ指定型の特長と、複雑なものに対してユーザがプログラムを作成するプログラミング型の両方の利点を持っていると考えられる。そのため、これらのツールのモデル記述言語としても有効であると期待できる[46]。

今後の課題としてはより人間の感性に訴えるソフトウェア作成環境を提供するための視覚的プログラミングまたは視覚的シミュレーション[47]等の導入、ＳＩ

ＬＱ並行処理系のインプリメントとホーン節単位の並列性の利用並びに不均一網モデル・有限プロセッサ数等の条件付きジョブ分割、プロセッサ割当等の問題が残されている。

謝　　辞

本研究の全過程を通じ、直接懇切丁寧なる御指導、御鞭撻を賜った大阪大学工学部通信工学教室手塚慶一教授に心から御礼申し上げる。

学部及び大学院において御指導御教授賜った同教室の熊谷信昭教授（現総長）、中西義郎教授、 滑川敏彦教授 （ 現名誉教授 ）、倉薗貞夫教授、本学産業科学研究所の角所 収教授、北橋忠宏教授に厚く御礼申し上げる。

筆者の属する手塚研究室の 真田英彦助教授、 打浪清一助教授、中西 暉講師、井上 健助手、後藤嘉代子技官を始め研究室の諸氏には種々の面で御世話になった。

また、本学産業科学研究所山口高平助手並びに手塚研究室大学院生小林真也氏、荒木 大氏、学部学生の巽 秀宜氏には、本研究に欠かすことのできなかった記述例並びに処理系の細部に至るまで熱心な御討論を頂いた。

本学卒業生佐藤 圭氏（現野村コンピュータシステム）、肥塚貞男氏（現松下電器産業株式会社）をはじめ、 神戸商科大学秋吉一郎講師、 三田工業株式会社生田善久氏、 柴田哲也氏、 本学大学院生山本 幹氏、杜 榕平氏、高取正浩氏、松浦聡夫氏、Chanintorn Jittawiriyanukoon氏、Suvepon Saecol氏、芳竹宣裕氏、学部学生川嶋哲司氏、北尾 充氏、大阪電気通信大学内山孝彦氏には熱心な御協力を頂いた。

本研究の実験システムを構成するに当たり御支援を頂いた株式会社東芝の関係各位に心から感謝する。

ここに記して、以上の皆様に深く感謝の意を表する。

文　　献

(1) 例えば、Schoemaker,S. Editor : “Computer Networks and Simulation 3,” North-Holland (1986).

(2) 中西: “シミュレーションの基礎,” 日新出版 (1978).

(3) Kobayashi,H: “Modeling and Analysis,” The System Programing Series, Addison Wesley (1978).

(4) 中西: “コンピュータシミュレーション,” 近代科学社 (1977).

(5) 山本,浦: “離散型シミュレーション言語の現状と将来の展望(1)～(2),” 情処学会誌 vol.22 , No.9 , No.11 (1981).

(6) Clocksin,W.L. and Mellish,C.S.: “Programming in Prolog,” Springer-Verlag , Berlin (1981).

(7) 例えば、“特集 : プログラミング言語Prolog,” 情処学会誌 Vol.25, No.12 (1984).

(8) 古川: “第5世代計算機のソフトウェア技術の展望,” 信学会誌 Vol.66,No.4 (1983).

(9) Shapiro,E.Y.: “A Subset of Concurrent Prolog and its Interpreter,” ICOT Technical Report TR-003 (1983)

(10) Futo,I. and Seredi,J.: “A Discrete Simulation based on Artificial Intelligence method,” Discrete simulation and related fields IMACS North-Holland (1982).

(11) Futo,I. and Seredi,J.: “T-Prologユーザマニュアル,” (1983).

(12) Futo,I. Gergely,I.: “A Logical Approach to Simulation (TS-Prolog),” Adequate System Modelling , Wedde.H ,Springer-Varlag Berlin (1983).

(13) Clark,K.L. and Gergory,S.: “ Parlog : A Parallel Logic Programming Language,” Research Report of Computing Imperial college of Science of Technology (1983).

(14) 藤田,田中,元岡: “時相論理によるハードウェア仕様記述とPrologを用いた

ゲート回路の検証," 情報処理論文誌 25,2 (1984).

〔15〕 長尾,淵:"論理と意味," 岩波情報科学7 (1983).

〔16〕 Campbell,J.A.:"Implemetation of PROLOG," ELLIS HORWOOD (1984).

〔17〕 "人工知能プログラムを高速に実行する マルチプロセッサ方式の並列処理マシン," 日経エレクトロニクス NO.387 (1986).

〔18〕 横田他:"PSIにおけるオブジェクトサポート," 情報処理第30回全大 1C-1 (1985).

〔19〕 西川他:"PSIのアーキテクチャ評価 (1)," 情報処理第30回全大 1C-2 (1985).

〔20〕 中島他:"PSIの性能評価 (2)," 情報処理第30回全大 1C-3 (1985).

〔21〕 佐藤,中西,真田,手塚:"並行処理待ち行列網シミュレータD-SSQ," 信学論(D) Vol.J69-D , No.3 (1986).

〔22〕 渡辺,佐藤,中西,真田,手塚:"並行処理事象型待ち行列網シミュレータD-SSQについて," 信学技法 CS83-102 (1983).

〔23〕 肥塚,渡辺,佐藤,中西,真田,手塚 :"時刻事象混合駆動型待ち行列網シミュレータについて," 昭59信学会通信、光・電波部門全大 275 (1984).

〔24〕 竹内:"Smalltalk(1)～(3)," bit Vol.15,No.12～13 Vol.16,No.1 (1983).

〔25〕 上田:"並列プログラミング言語," 情報処理学会誌 Vol.27,No.9 (1986).

〔26〕 稲守,戸田 :"複数マイクロプロセッサを用いた並列形通信網トラヒックシミュレータの評価," 信学論(B) J68-B,1,pp.22-29 (1985).

〔27〕 佐竹,上月,西田,宮原,高島:"分散形待行列網シミュレータ," 信学技報 EC83-40 (1983).

〔28〕 中川,小林,相磯:"データ駆動型離散系シミュレータKDSS-I," 信学論(D) J65-D,3,pp.386-393 (1982).

〔29〕 Peacock,J.K. Wong,J.W. and Manning,E.:"Distributed simulation using a network of microcomputers," Computer Networks,3 North-Holland (1979).

〔30〕 淵,黒川:"新世代プログラミング," 共立出版 (1986).

(31) 米澤:“オブジェクト指向プログラミングについて,” コンピュータソフトウェア Vol.1,No.1 (1984).

(32) M.Reiser, “A Qeueuing Network Analysis with Window Flow Control,” IEEE Tras. Vol.COM-27,No.8 (1979).

(33) 小林,渡辺,山口,中西,真田,角所,手塚 :“待ち行列網シミュレーション専用論理型言語の記述性に関する考察,” 昭和60年情報理論とその応用研究会 (1985).

(34) 荻野,桜川,柴山:“Prolog-KABA リファレンスマニュアル,” 岩崎技研工業 (1984).

(35) 荒木,渡辺,中西,真田,手塚:“論理型言語による待ち行列網シミュレーションについて,” 信学会情報ネットワーク研究会 IN86-49 (1986).

(36) 山本他:“高性能PROLOGマシン:HPM,” 情報処理第30回全大 1C4-9 (1985).

(37) 松田,小畑,金田,前川:“並列PrologマシンPARKの全体構成,” 情報処理第30回全大 2C-1 (1985).

(38) 浜中,田中,元岡:“PIEのハードウェアシミュレータ,” 情報処理第30回全大 2C-2 (1985).

(39) Tokuda,H. and Manning,E.G. :“An Interprocess Communication Model for a Distributed Software Testbed,” ACM pp.205-212(1983).

(40) 渡辺,中西,真田,手塚:“分散型待ち行列網シミュレータD-SSQの処理能力解析,” 待ち行列理論とその応用研究会 京都大学数理解析研 講究録 564 (1985).

(41) Jefferson,D and Sowizral,H :“Fast Concurrent Simulation Using the Time Warp Mechanism,” Proc. of the SCS Distributed Simulation Conference,SanDiego (1985).

(42) 渡辺,中西,真田,手塚 :“規制先行制御方式を用いた非同期ジョブ並行処理システムの処理能力解析,” 信学論(D) Vol.J69-D No.10 (1986).

(43) Sauer,C.H. MacNair,E.A. and Salza,S. :“ A Language for Extended Queueing Network Models,” IBM J. RES. DEVELOP Vol.24 No.6 (1980).

(44) Potier,D. and Verani,M. : " The Markovian Solver of QNAP2 and Examples," International Seminar on Computer Networking and Performance Evaluation (1985).

(45) 小島,米田,田中,春木 :"蟻塚とその利用環境," 情報処理研報 Vol.86 , OS研究会 No.12 (1986).

(46) 渡辺,中西,真田,手塚 :"待ち行列網解析ツールにおけるモデル記述言語の一提案," 待ち行列理論とその応用研究会 京都大学数理解析研 講究録 596 (1986).

(47) "Visual Programming," Computer Vol.18 , No.8 , IEEE (1985).

(48) Nance,R.E.: "The Time and State Relationships," Information Processing 83 IFIP (1983).

(49) 渡辺,小林,山口,中西,真田,角所,手塚: "待ち行列網シミュレーション専用論理型言語について," 情報処理研報 Vol.85 , MDP 26-3 (1985).

付録1　ＢＮＦによるＳＩＬＱシンタクス

```
<statement>::=
  <unit cls>|<non-unit cls>|<query>
<unit cls>::=<head>.
<non-unit cls>::=<head>:-<goal sqnc>.
<head>::=<User primitive>|<Prolog term>
<goal sqnc>::=<goal>|<goal>,<goal sqnc>
<goal>::=<System primitive>|<Prolog term>
<query>::=sim(<file name>).
<User primitive>::=
  connect(<C_Id>,<P_Id>,<P_Id>)
 |append_model_field(<Ar_Id>,<MDL_F>)
 |interval(<Ar_Id>,<Time>)
 |queue_out_condition(<Q_Id>)
 |buffer_capacity(<Q_Id>,<Number>)
 |queue_priority(<Q_Id>,<Cstm>,<Cstm>)
 |server_capacity(<S_Id>,<Number>)
 |service_time(<S_Id>,<Time>)
 |customer_merge(<J_Id>,<MDL_F>,<MDL_F>)
 |branch_selection(<B_Id>,<Cstm>,<List>)
 |customer_change(<P_Id>,<MDL_F>,<MDL_F>)
 |start_time(<Time>)
 |end_time(<Time>)
 |data_collect_section(<Time>,<Time>)
<System primitive>::=
  sys_customers_in_queue(<Q_Id>,<Cstms>)
 |sys_customers_in_server(<S_Id>,<Cstms>)
 |sys_time(<Time>)
 |sys_customers_record(<P_Id>,<Cstms>)
 |sys_customers(<Cstms> )
 |sys_change(<Prolog term>,<Prolog term>)
 |sys_customers_number(<P_Id>,<List>)
 |sys_mean_delay_time(<P_Id>,<P_Id>)
 |sys_div_delay_time(<P_Id>,<P_Id>)
 |sys_mean_queue_length(<Q_Id>)
 |sys_div_queue_length(<Q_Id>)
```

```
  |sys_max_queue_length(<Q_Id>)
  |sys_mean_queueing_time(<Q_Id>)
  |sys_div_queueing_time(<Q_Id>)
  |sys_max_queueing_time(<Q_Id>)
  |sys_utilization(<S_Id>)
<C_Id>::=<atom>|<C_Id>
<P_Id>::=
  <Ar_Id>|<Q_Id>|<S_Id>|<J_Id>|<B_Id>|<D_Id>
<Ar_Id>::=[arrival,<Id>]
<Q_Id>::=[queue,<Id>]
<S_Id>::=[server,<Id>]
<J_Id>::=[joint,<Id>]
<B_Id>::=[branch,<Id>]
<D_Id>::=[departure,<Id>]
<MDL_F>::=<atom>
<Cstm>::=[<Sys_F>|<MDL_F>]
<Sys_F>::=[]|[[<Time>|<P_Id>],<Sys_F>]
<Cstms>::=[]|[<Cstm>,<Cstms>]
```

付録２　P_c, t_i, t_c, n, p_iの導出

（a）p_c

規制値Tが０に近づくと時刻矛盾を生じる確率p_cは０に近づく。一方、Tが∞に近づく、すなわち無規制の状態においてはp_cはある一定確率に近づく。よって、

$$p_c = P_r \cdot \{1 - \exp(-b_0 \cdot \mu \cdot T)\} \qquad (A.1)$$

ただし、P_r：無規制時の時刻矛盾発生率

b_0：定数

（b）t_i

あるプロセッサにおける現在のシミュレーション時刻から次の時刻までの更新幅をｘとすれば、ｘ＞Ｔの時、このプロセッサは遊休状態となる。解析仮定の（２）の指数分布の無記憶性より、ｘ＞Ｔの条件のもとでの遊休シミュレーション時間（ｘ－Ｔ）の期待値は１／μである。

また、稼働中のプロセッサ数N_wは、全プロセッサ数Ｎのうち注目しているプロセッサ以外の半分とすれば

$$N_w = \frac{N-1}{2} \qquad (A.2)$$

確定時刻はN_wのプロセッサによって、$\frac{1}{N_w} \cdot \frac{1}{\mu}$ずつ更新されるため

$$\frac{\frac{1}{\mu}}{\frac{1}{N_w} \cdot \frac{1}{\mu}} = N_w \qquad (A.3)$$

だけの処理が遊休状態の間に実行される。

また、１つの処理にはt_{m0}ずつかかるため注目したプロセッサが遊休状態を抜け出すまでの実時間は次式となる。

$$t_i = N_w \cdot t_{m0} = N_w \cdot \alpha \cdot t_1 \qquad (A.4)$$

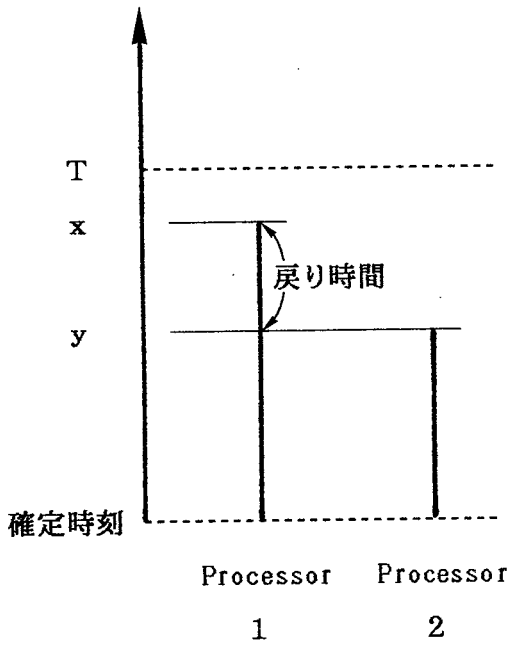
（c）t_c

まず、一回のキャンセルによって戻る平均シミュレーション時間を計算する。

キャンセルが生じるまでのプロセッサ１およびプロセッサ２の時刻更新間隔分布密度関数ｐ（ｘ）、ｐ（ｙ）は、１／μの指数分布がいくつか重なったものであるがここでは簡単のため、以下のようにそれぞれm_1、m_2の平均を持った指数分布と仮定する。

$$p(x) = m_2 \cdot \exp(-m_2 \cdot x)$$
$$p(y) = m_1 \cdot \exp(-m_1 \cdot y) \qquad (A.5)$$

また、ｘとｙはメッセージ送受信の間は独立であるので、ｐ（ｘ，ｙ）＝ｐ（ｘ）・ｐ（ｙ）となる。



x：Processor１の確定時刻からの時間幅

y：Processor２の確定時刻からの時間幅

図Ａ－１　時刻矛盾発生時の状況

図Ａ－１に示すようにプロセッサ２からプロセッサ１へメッセージの送信が起こり時刻矛盾が生じた時の戻りシミュレーション時間の期待値は次式となる。

$$E(T) = \frac{\int_{x=0}^{T} \int_{y=0}^{x} (x-y) \cdot p(x, y)\, dy\, dx}{\int_{x=0}^{T} \int_{y=0}^{x} p(x, y)\, dy\, dx}$$

$$= \frac{E_1(T)}{E_2(T)} \qquad (A.6)$$

ただし、

$$E_1(T) = -T \cdot \exp(-m_2 \cdot T) + \frac{m_1 - m_2}{m_1 \cdot m_2} \cdot \{1 - \exp(-m_2 \cdot T)\} + \frac{m_2}{m_1 \cdot (m_1 + m_2)} \cdot \{1 - \exp(-m_1 \cdot T - m_2 \cdot T)\}$$

$$E_2(T) = \frac{m_1}{m_1 + m_2} + \frac{m_2}{m_1 + m_2} \cdot \exp(-m_1 \cdot T - m_2 \cdot T)$$

待ち行列網シミュレーションにおいては、発生、伝送、到着が主な処理内容であるからプロセッサ２からプロセッサ１へメッセージを送信する時間にプロセッサが実行する処理数は約３倍と考えられる。すなわち$m_2 = 3 \cdot m_1$が成立する。

また、$T \to \infty$において$E(T) \to 1/m_2$すなわち無規制状態では$1/m_2$だけ戻ることになる。$1/m_2$の間実行される処理数はμ/m_2である。これをβとすれば$m_1 = \mu/(3 \cdot \beta)$，$m_2 = \mu/\beta$となる。

キャンセル処理に必要な実時間t_cは、戻るシミュレーション時間内に存在する事象の数$\{\mu \cdot E(T)\}$に線形に増加すると考えられる。従って、t_1で正規化した係数η_0、η_1を用いると、

$$t_c = \{\eta_0 \cdot \mu \cdot E(T) + \eta_1\} \cdot t_1 \qquad (A.7)$$

と考えられる。

（d）n

図６－５（b）により０～ｔに進むシミュレーション時間T_Mは、

$$T_M = \sum_{j=1}^{k} n_j \cdot T_j - \sum_{j=1}^{n_c} T_{cj} \qquad (A.8)$$

ただし、T_j　：ｊ番目の事象によって更新されるシミュレーション時間

T_{cj}　：ｊ番目のキャンセルで戻るシミュレーション時間

ｊについての平均を考えれば、

$$T_M = k \cdot n \cdot T_0 - n_c \cdot T_c \qquad (A.9)$$

ただし、$n = \overline{n_j}$、$T_0 = \overline{T_j} = 1/\mu$、$T_c = \overline{\overline{T_{cj}}} = E(T)$

また、$T_M = k \cdot T_0$より

$$k \cdot T_0 = k \cdot n \cdot T_0 - n_c \cdot T_c$$
$$= k \cdot n \cdot T_0 - k \cdot n \cdot p_c \cdot T_c \qquad (A.10)$$

よって

$$n = \frac{T_0}{T_0 - p_c \cdot T_c} = \frac{1}{1 - p_c \cdot \mu \cdot E(T)} \qquad (A.11)$$

（e）p_i

プロセッサが遊休状態となるのは、シミュレーション時刻更新幅が規制値Tを越えるときであるから、

$$p_i = \int_{x=T}^{\infty} p_e(x)\,dx = \exp(-\mu \cdot T) \qquad (A.12)$$

以上の式（A．1）（A．4）（A．7）（A．11）（A．12）を第6章式（10）に代入すれば、式（11）を得る。