構造をもった言語に対する会話的プロセッサ*

牛島和夫** 有田五次郎** 大槻説乎**

1. まえがき

電子計算機は，本来人間のかかえている種々の問題に対する認識や解決の補助手段である．しかるに計算機の演算速度の飛躍的な向上は，経済的に極度に合理化されたモニターによる一括処理方式を現出せしめ，計算機に対する人間の介在を許さなくなった．これは経済性の旗印のもとに人間と機械を分離したことにほかならない．この反省として従来のモニター方式をも内包して，人間と機械の相互作用を十分考慮したシステムが登場したのは歴史的必然である．このかげにランダムアクセスの記憶装置の低廉化および大容量化，実時間処理，多重プログラミングなどの技術の実用化，図形音声入出力装置の近い将来での実現可能性などの裏付けがあったことはいうまでもない．このように，従来のモニターシステムの単なる拡張ということではなく，一次元高いシステムとしてタイムシェアリングシステムを考えなければならないと思われる．したがってこのシステムの運用においては，上述のように人間と機械の相互作用が重要な問題の一つとなるが，本稿においてはタイムシェアリングシステムで使用される共通プログラムの一つとして，会話的なプロセッサについての考察をこころみる．

すでに日本でも FORTRAN 様の言語を機械との会話によってプログラムし，計算実行させるという試みは一部で実用化されているようであるが，ブロック構造をもった言語（たとえば ALGOL, PL/I）と機械との会話によってプログラムし計算実行させるという例は見かけない．本来ブロック構造をもった言語は記述言語としては非常に優れているが，これが会話体でプログラムする場合にも十分使用にたえるかどうかを検討する目的で，材料になるプログラム言語として，ブロック構造をもった言語をとりあげてみた．さらに徹底的な会話というからには，プログラムを作る途中で人間の犯したあやまりを即座に指摘してくれ，修正，一部実行などが随時に，しかも交互にできるような形が望ましい．このような試みをブロック構造をもった言語に対して行なった．

2. 人間と機械との会話について

2.1 会話の意味

人間と機械の対話は，すでに過去において（特に電子計算機の初期の時代においては）機械とにらめっこで，しかも1ステップ，1ステップ機械語で対話しながら，プログラムの虫取りをするということが行なわれてきており，特に目新しいことではない．しかし近年になって，機械語以外の言語による機械との対話が当然のことながら話題にのぼり，そのようなコンパイラなども発表されているようであるが，人間と機械が1対1で会話するというのでは，計算時間の無駄使いというほかない．計算機の会話的利用が，経済性をもって考えられるようになったのは，多重プログラミング，時分割，実時間処理などの技術が開発されてきたからにほかならない．

人間と機械の会話といっても，あくまでも気ままな人間が主であって，機械はその僕であり，人間が困ったとき助け舟を出してくれるにすぎない．気ままな人間の気持の動きを機械が適切に受けとって，反応してくれなければならない．機械と会話しながらプログラムを書き，それを任意の時点で修正し，書き上った部分を任意の時点で一部実行し，これらの操作を任意に組み合わせることができる必要がある．われわれは次のような方法によってこれの実現をはかった．

（1） コンパイラとエクゼキュータの分離

人間は気ままであるし間違いを犯しやすい．したがってプログラムの任意の部分を任意の時に実行させ，出力させる必要がある．このために，処理プログラムを，翻訳を司どるコンパイラと実行を司どるエクゼキュータの二つに分離して，任意の時点で切りかわるようにし，さらにそれぞれが対話的要素をもつように設計した．

（2） 処理の最小単位の導入

---

* A Conversational Processor for a Structuring Language by Kazuo Ushijima, Ituziro Arita and Setsuko Otsuki (Kyushu University, Faculty of Engineering)

** 九州大学工学部

任意の時点でソースプログラムの修正や一部実行などを行なうためには処理の最小単位を導入して，修正の影響を最小限にとどめたり，実行の範囲を明確にする必要がある．さらに処理の最小単位は，言語のブロック構造に伴う諸性質と矛盾することがないように決定されねばならない．

（3） 実行指定プログラムの導入

プログラムを試行錯誤的に実行させるために実行方法をプログラム化して任意の修正，イメージの転記を可能にした．したがってコマンドを使ってその場限りの実行を指定する方法はとらない．

2.2 会話の例

人間と機械との会話の例は，たとえば**第1表**のようになる．この例で機械から出力される番号は，処理番号と呼ばれるもので，処理の最小単位に対応するものである．☆ALGOL，☆EXECUTE などはコマンドであって，これによって人間はコンパイラまたはエクゼキュータのフェーズへの切り換えを行なっているのである．このようにコマンドはその場かぎりのものであって処理の対象にはならない．

第1表 会 話 の 例

```
        Your name please
1       Taro Yamada
        New or experienced
2       New
        Your number 0003
3       ☆ALGOL
3       begin real A,B,C; array D[1:5];
6       procedure P(X,Y,Z);
7       value Y; array X; real Y,Z;
10      X[1]:=Y+Z;
11      A:=B*0.5+C*A;
12      P(D,A+1.0,B);
13      ☆EXECUTE
13      STEP(11,12)11,A; 12,D[1];
14      ☆ASSIGN 11,A:=1.5,B:=2.3,C:=3.8;
14      ☆START
        11 A= .6850000000₁₀1
14      ☆RESTART
        12 D[1]= .1015000000₁₀2
14      ☆ALGOL
14      if D[1]=0 then A:=B*0.5-C*A;
16      ☆EXECUTE
                .
                .
                .
```

（注） 下線を施こしたものは計算機からの出力を示す

さて，われわれが設計したプロセッサはタイムシェアリングシステムの共通プログラムとしての性格を持たねばならない．したがって純粋な手続き部分からなるプログラム領域（X領域と呼ぶ）とそれに接続するデータ領域に分割される．さらに後者は読み書き可能な領域（W 領域と呼ぶ）と読み出しのみ可能な領域（R 領域と呼ぶ）に分かれている．これらの領域の先頭番地を示すベースレジスタとしてそれぞれ X レジスタ，W レジスタ，および R レジスタを対応させている．したがって上述したコマンドを用いた人間によるフェーズの切り換えは，機械の側ではスーパバイザによる X, W および R 領域の切りかえ，すなわち，ベースレジスタの入れかえに対応する．

3. ALGOL コンパイラ

3.1 コンパイラへの要請

タイムシェアリングシステムにおける会話体のコンパイラへの要請をまとめてみると次のようになる．

C·1 多重プログラム，時分割，実時間処理の要請

C·2 会話によるプログラム作成

1) コンパイラフェーズとエクゼキュータフェーズの混在

2) 各フェーズにおいて修正随時

C·3 実行時における計算時間

C·4 コンパイルの速さ

C·5 ブロック構造をもつ言語

これらのうち C·5 は特に ALGOL をとりあげたために生ずる要請で，他の言語では違った要請となるであろう．これらの要請をみたすために，次の諸点が考えられる．

A·1 オブジェクトプログラムは利用者固有のデーター領域にでき上るのであるから，動的再配置が可能（relocatable）であればよく，読み出しのみ可能（read only）である必要はない．

A·2 独立したプログラムとして文法的な完成を待たずに一部実行可能であるためには，入力された点までのプログラムがいつでも実行可能でなければならない．そのためには，逐次的な翻訳方法をとらなければならない．

A·3 ソースプログラムの一部の修正，変更がオブジェクトプログラム全体に及ばないようなオブジェクトプログラムの構成が要求される．

A·4 オブジェクトプログラム実行時における計算時間を短かくするためには，通訳的（interpretive）な部分をできるだけ少なくして，直接機械語の速度で計算可能なように翻訳する．

A·3 の立場は ALGOL が本来句構造言語（phrase

structure language）であって，一部の修正が，構造に関係する部分（begin, end, 宣言部の挿入，削除）に行なわれれば，当然全体に影響を及ぼすので，もし入力プログラム全体として1本のすじにコルパイルされていれば，全体を再コンパイルする必要がある．一方 ALGOL は，文脈に無関係な言語（context free language）でもあるから，そのことを利用して，修正による影響が最小限であるようなオブジェクトプログラムの構成を考える．そこで K. Lock[4] も提案しているように，上のような要請をみたす最小の単位を決めることにした．ここで最小というのは，その単位より小さい単位については修正，実行などの指定が行なえないことを意味する．そしてこれは ALGOL コンパイラの W 領域の最小構成単位でもある．

なお，われわれがとりあげた ALGOL 文法の範囲とそれに対する制限および変更は次のとおりである．

1) 計算機の容量の関係から，JIS 原案 ALGOL 水準 4000+α 程度とする[5]．

2) ブロックと複合文は区別しない．それは一たん完結したブロックまたは複合文も随時の修正によって，いつでも複合文またはブロックにかわりうるからである．

3) 宣言は，その名前が使われる直前までにしてあればよい．

3.2 処理の最小単位

ALGOL は〈文〉を基本にした木構造をもっているのであるが，これを基礎にして，会話的利用のための便宜を考えて，処理の最小単位（これを処理単位と呼ぶ）を次のようにした．各処理単位を他と区別するために与える番号が処理番号である．

プログラムの処理単位の分類

（1）〈名札〉:

（2）begin

（3）宣　言

　単純変数，配列，手続きの頭（値の部と規制部は除く），手続きの値の部，規制部，スイッチ

（4）実行を伴う最小の文または節

　代入文，飛び越し文，手続き文，繰り返えし節，条件節（ただし，算術式，論理式，行先式に含まれる条件節は除く）

（5）end〈end の注釈〉

（6）comment〈; を含まない任意の文字列〉;

3.3 コンパイルの手順のあらまし

コマンド ☆ ALGOL による利用者からのコンパイラの呼び出しによりコンパイラは，処理番号を出力して，ソースプログラムの入力待ちとなる．1行分の入力終了をまってコンパイルが開始される．ALGOL では一般に1行分の入力系列は n 個の処理単位に分解される．これらの n 単位がすべて，上部の文法構造と文法的に矛盾しなければ，すべて処理単位として登録され，それぞれリストとして連結される．n 単位の処理が終るとコンパイラは再び処理番号を出力して，次のソースプログラムの入力待ちとなる．この処理番号は前の処理番号より n だけ多い数となっている．

さて，1行 n 個の処理単位中 m 個（$m<n$）まで処理して，$m+1$ 個目に上部構造とてらしてエラーを検出すると，m 番目までは，処理番号をつけて登録されるが，$m+1$〜n 番目はキャンセルされる．したがって，次の処理番号は $m+1$ だけ多い番号が出力されて入力待ちとなる．結局コンパイルというのは，これらの処理単位を逐次的に実行可能なオブジェクトプログラムに作り上げて，リストとして連結してゆく作業になる．

さて，この処理単位の内部構造は次のようになっている．

＜処理単位＞は大別して＜処理単位の頭＞と＜処理単位の本体＞とからなっている．

＜処理単位の頭＞は次のものからなる．

処理単位の種類
処理番号
下部の処理単位を示すポインタ
次の処理単位を示すポインタ
前の処理単位を示すポインタ
名札の処理単位を示すポインタ
変数の宣言の処理単位を示すポインタ
処理単位の本体を示すポインタ（P0: コンパイラのデータ領域の先頭からの相対番地）
オブジェクトプログラムの先頭番地（P1: P0 からの相対番地）
名前表の先頭番地（P2: P0 からの相対番地）
定数表の先頭番地（P3: P0 からの相対番地）
処理単位の本体の長さ（語数）

＜処理単位の本体＞は次のものからなっている．

P0: ソースプログラムのイメージ
P1: オブジェクトプログラム
P2: ソースプログラム中にあらわれる名前表
P3: ソースプログラム中にあらわれる定数の2進変換された値の表

このように＜処理単位の頭＞は固定長をもっているのでこれをリストの形で連結しておき，＜処理単位の本体＞はデータ領域の別場所にしまってゆく．

3.4 処理単位の本体の作成

3.4.1 単純変数の宣言

第1表処理番号 4 real A, B, C; は次のような本体を作る．

P0:	real A, B, C;
P1:	A r W +
	B r W +
	C r W +

←―― 番地部 ――→

ここで r は実数であることを示す．また W は W レジスタである．単純変数の宣言では，P2, P3 の部分がない．このように P1, P2, P3 の領域は必ず存在するわけではなく，たとえば，begin や value Y; などは，P0 のイメージしかもっていない．

3.4.2 実行を伴う文または節

実行を伴う文または節のうち，代入文について第1表の例で説明する．処理番号 11 の代入文は次のようになる．

P0:	A:=B*0.5+C*A;
P1:	load, 1A/([P2+1]) mult, 1A/[P3+0] load, 2A/([P2+2]) mult, 2A/([P2+0]) add, 1A/2A store, 1A/([P2+0]) jump, /X+＜処理単位のつなぎルーチン＞
P2:	A　BOX4　r (R+BOX4+0) B　BOX4　r (R+BOX4+1) C　BOX4　r (R+BOX4+2)
P3:	0.5

ここで（　）で間接アドレス指定，［　］で SCC 修飾，BOX4 で 3.4.1 で宣言した宣言の処理単位の先頭番地 P0 を示す．また，R は R レジスタ，X は X レジスタである．宣言および実行の際の W または R レジスタ修飾の意味は，オブジェクトプログラムがあるのはコンパイラにとっては W 領域であるが，エクゼキュータにとっては R 領域となるからである．また，この時，変数の割付けはエクゼキュータの W 領域に行なう．

実行を伴う処理単位に共通にいえることであるが，逐次翻訳により P2, P3 を未定義な浮動番地としたまま，一つの処理単位の終りまで，機械語におとす．このとき処理単位内かぎりであらわれる名前を，出現するごとにすでに宣言されている宣言の処理単位を探して，P2 表にのせてゆく．P2 表には，その名前のほか，番地部としてその宣言の処理単位の本体を示すポインタ（上の例で BOX4 にあたる），その名前のある番地（BOX n からの相対番地 j，および R レジスタ修飾）にして，それ全体を間接アドレス指定にしておく．すなわち

P2+i: (R+BOX n+j)

となる．BOX n の j 番目の番地部は

BOX n+j: W+ [　]

となっているので，利用者が☆EXECUTE によって実行を指定するとエクゼキュータは，実行の処理単位の本体の P2 表を見て，そこにかかげてあるすべての名前について，宣言の処理単位の本来の該当場所（BOX n+j 番地）の [　] の中に実際の割付番地 k（実際の割付番地はエクゼキュータの W 領域に作られるので，W からの相対番地）を書きこんでから実行に入る．したがって，たとえば

load, 1A/([2P+i])

とコンパイルされた命令は実行時点では間接アドレス機能により

load, 1A/W+k

と同じになる．

そのほか，くり返し節は，その本体の実行の結果リストの下部の単位へ進むか，次の単位へ進むかの情報を作るようにコンパイルすればよいし，条件節は true か false の値を作るようにコンパイルすればよい．実行時にはこの場合も，処理単位の本体内では通訳的な部分は全くなく，処理単位の本体の実行を終って，次に実行する単位をどれにするかということだけが問題で，通訳的な部分はこれだけで，通訳的といっても，単なるスイッチングにすぎない簡単なものである．

しかし行先文には問題がある．それは，＜名札＞の値というものは，本質的に，プログラムが完結しなければ一意的に決定できない性質のものであるからであ

る．そこでコンパイルの際は，行先式を評価して<名札>の名前をそのまま値とするようなプログラムにコンパイルしておく．エクゼキュータは，処理のリストをたどって，その名前がその実行時点でのみきまる場所を解釈しなければならない．そのほか標準関数の呼び出しなどの際に生ずるデータ領域とプログラム領域とのリンクの問題，オブジェクトプログラムの実行の打切りと，再開の際に生ずる副作用をさけるための配慮など，会話体の故に生ずるコンパイル上の技術的な問題はたくさんあるがここでは省略する[3]．

3.5 ソースプログラムの修正と処理単位の本体への影響

利用者が修正のできる最小単位は前述のように処理単位ごとであって，それより小さい修正は，処理単位全体をそっくり行なわなければならない．

修正の操作は次の三とおりが考えられる．

削除　☆DELETE（<範囲の指定>）

挿入　☆INSERT（<範囲の指定>）

入れ換え　☆EXCHANGE（<範囲の指定>）

ここで範囲の指定方法は（<上限処理番号>，<下限処理番号>）または（<処理番号>）である．

さてプログラムの一部修正の結果，直接修正をうけない処理単位が再コンパイルのために本体の大きさが変化したり，オブジェクトプログラムを変更しなければならなくなっては非常に面倒である．前述のように処理単位の本体を作っておくと，P2 表内の番地部だけを書きかえるだけで，オブジェクトプログラムおよび本体の長さはそのままでよい．

修正は次のような手順で行なわれる．

（1）削　除

指定範囲を処理のリストの連結からはずして，上限，下限を短絡する．次に削除した範囲内に，句構造に影響を与えるような処理単位があれば，下限以下の処理単位の本体の P2 表を書きかえればよい．宣言の単位が削除されて未定義になる場合も生じうるが，この場合は処理単位の本体，リストの連結はそのままにして，処理単位の頭の該当部分にエラービットをつけておく．

（2）挿　入

挿入を指定するとコンパイラは，処理番号のつけ方の規則にしたがって，上限処理番号 +1 を出力する．利用者は，それに対して，挿入したいプログラムを上限番号より以前のプログラムとの文法構造をしらべながら，処理単位ごとに逐次的にコンパイルする．利用者から入力終りの信号をもってコンパイルを終り，挿入プログラムの中に句構造に影響を与える単位が含まれていれば，削除の場合と同様に下限番号以下への影響を調べる．

（3）入れ換え

削除の操作を行ない，下部の影響は調べないで直ちに挿入の手続きを行なう．削除された部分と挿入された部分に句構造に影響を与える単位があれば，下限番号以下への影響をしらべる．

4. エクゼキュータ

4.1 実行に関する原則

すでに述べたように，実行についても人間と機械との間の会話ということを基調とする．したがって，任意の場所で，今までに入力したソースプログラムを実行させたり，変数に値を与えたり，計算結果を出力したりすることができる必要がある．そのため，まず実行指定プログラムなる概念を導入する．実行指定プログラムは，ソースプログラムとは異なった言語体系に属し，エクゼキュータによって解釈実行されるもので，実行のモード，実行の範囲，チェックのため出力する変数などを指定する．利用者のプログラムの実行は，ソースプログラムの上にこの実行指定プログラムがかぶさった形で行なわれる．すなわち，ソースプログラムのうちで実行指定プログラムに指定されているもののみが，指定されたモードで実行され，指定された変数の値が出力される（第1表，13，14参照）．

このようにプログラムの一部を実行させるため変数に値を与えたり，実行の途中で止めて変数の値を調べたり，再実行させたりするため，実行に関するコマンドが用意されている．これらのコマンドや実行指定プログラムは ALGOL に附随したものではなく，たとえば，FORTRAN に対しても，適用できるものである．ただし，解釈の仕方は各エクゼキュータによって異なることがある．

4.2 実行に関するコマンド

実行のフェーズにおいて使用されるコマンドには2種類ある．第1は，実行指定プログラムの記述に関するもので，挿入，削除などであるが，これらはコンパイラのフェーズと共通なので省略する．第2は，実行のフェーズのみにおいて使用できるもので，これらについて次にのべる．

（1）☆EXECUTE

コンパイラのフェーズからエクゼキュータのフェー

ズに切りかえる．このコマンド以後に入力できるのは実行に関するコマンドか，実行指定プログラムのみである．

（2） ☆ASSIGN

変数に値を与えるコマンドで，バッカス記法を用いると次のように書ける．

<代入コマンド>::=☆ASSIGN <入力指定部>

<入力指定部>::=<入力リスト>|<入力リスト>;<入力指定部>

<入力リスト>::=<処理番号>,<入力文のリスト>

<入力文のリスト>::=<入力文>|<入力文>,<入力文のリスト>

<入力文>::=<変数名>:=<値>

（3） ☆TYPE

変数の値の出力を指定するコマンドである．

<出力コマンド>::=☆TYPE<出力指定部>

<出力指定部>::=<出力リスト>|<出力リスト>;<出力指定部>

<出力リスト>::=<処理番号>,<出力変数のリスト>

<出力変数のリスト>::=<変数名>|<変数名>,<出力変数のリスト>

（4） ☆START

実行指定プログラムの終ったことを示し，実行に入る．実行は指定されたもののうちで処理番号の一番小さなものから始まる．

（5） ☆STOP

利用者のプログラムが実行されているときに，リアルタイムインクワイリングのボタンを押すと☆STOPとみなされる．

（6） ☆RESTART

モード指定 STEP（後述）または ☆STOP によって，実行を止められていたプログラムの実行を再開する．次に実行されるものはそのプログラムの論理的流れに従う．

4.3 実行指定プログラム

実行指定プログラムは，会話の原則に従って，デバッグしながらプログラムを構成していくため，未完成のプログラムの実行を可能にするものである．このため記述されたプログラムの任意の場所を指定することが必要になる．またデバックの段階ではプログラムの途中で止めて，各変数の値を調べたり，値を変化させたりすることができる必要がある．このような要求からわれわれは実行のモードに STEP を導入した．もちろん実行の速さを考えて途中で止まらない FULL モードが別にある．

実行指定プログラムはバッカス記法を用いて書くと次のようになる．

<実行指定プログラム>::=<実行指定ステートメントのつながり>

<実行指定ステートメントのつながり>::=<処理番号><実行指定ステートメント><実行指定ステートメントのつながり>

<実行指定ステートメント>::=<モード>(<範囲指定>)<出力指定リストのつながり>

<モード>::=STEP|FULL

<範囲指定>::=<処理番号>|<処理番号>,<処理番号>

<出力指定リストのつながり>::=<空>|<出力指定リスト>|<出力指定リスト>;<出力指定リストのつながり>

<出力指定リスト>::=<処理番号>,<変数名>|<処理番号>,<変数のリスト>

実行指定プログラムは，たとえば次のようになる．

```
10. FULL (3, 5) 3, A, B; 4, C; 5, A,E;
11. STEP (6, 7) 6, X; 7, Y;
12. FULL (8, 9)
```

モード FULL は処理単位の終りで止まらずに指定された範囲全部を実行する．モード STEP は指定された範囲内の最小処理単位の実行が終了するごとに止まって人間の指示を待つ．出力指定リストにあげられた変数は，指定された処理単位の実行が終了するごとにその値が出力される．

4.4 エクゼキュータの概要

4.4.1 実行指定プログラムの解釈

☆EXECUTE によって，制御がわたされると，エクゼキュータは処理番号を出力して，実行指定プログラムの読み込みを行なう．

実行指定ステートメントがタイプインされると，コンパイラのR領域内の処理のリストを調べ，そのステートメントにエラーがないかどうかを調べる．エラーがなければ，指定された処理単位の頭に実行指定ビット，処理単位の本体の P2 表に出力指定ビットを設定する．☆START によって直ちに指定された処理単位の実行にうつる．

☆ASSIGN があれば，W領域内の assign 表にのせておき，実行の際にその変数の番地が割当てられた

ときに値が入れられる．

4.4.2　オブジェクトプログラムの実行

オブジェクトプログラム実行ルーチンは，SCC に相当する処理のリストのポインタを持っていて，そのポインタの示す処理の単位を次々と実行してゆく．

まず，処理単位の頭をみると，実行のモード，処理単位の種類，処理単位の本体の所在，次に実行すべき処理番号が書かれているので，それに従って，処理単位の本体の実行を行なう．このとき，処理単位の本体中の変数で値が使用されるものについては，その値が付与されているかどうかチェックする．処理単位の本体の実行が終ると，変数表を調べ，出力指定のあるものについては値を出力し，STEP モードの場合は，一たんコマンドフェーズに制御をわたす．FULL モードの場合は引き続いて次の処理単位を処理していく．

次に実行すべき処理単位の番号を示すポインタは処理単位の本体の実行によってはじめて決まる場合もある．たとえば，くり返えし節，条件節，飛びこし文などにおいては，処理単位の本体の実行によって，次に実行すべき処理番号が決まり，それをパラメタとして，オブジェクトプログラム実行ルーチンに引き渡すことになる．なお，配列の番地計算のルーチン，標準関数，手続きのパラメタの受け渡しのためのサブルーチンなどは，エクゼキュータ内のサブルーチンとして含まれている（各実行ルーチンについては文献 2）参照）．

したがって，標準関数なども当然リエントラントになっていて，すべての利用者に対して一つのプログラムが存在するのみである．

あとがき

われわれは，以上述べたようなプロセッサを九州大学中央計数施設に設置された OKITAC 5090 H で実験的に実現した．しかし，この計算機はタイムシェアリングシステムが話題になる以前に設計されたものであって，このために必要なハードウェア上の考慮が特に払われているわけではない．したがって，このプロセッサを OKITAC-5090 H で実用化することは考えていない．むしろ最近続々と発表されている新しい計算機システムのライブラリの一つとして，実用の域にまでもってゆきたいと考えている．

参考文献

1) F.J. Corbato 他：Multics 関連論文 6 編，AFIPS Conference Proceedings, Vol, 27, pp. 184～247, 1965. 11
2) 牛島，有田，大槻，久原：OKITAC 5090 H によるタイムシェアリングシステムについて，第 8 回プログラミングシンポジウム報告集 C 2-5 pp. 104～139, 1967. 1
3) 牛島，有田，大槻：九州大学 TSS に関する補足，日本電子協タイムシェアリングシンポジウム資料，1967. 2
4) K. Lock: Structuring Programs For Multiprogran Time-Sharing On-line Applications, AFIPS Conference Proceedings, Vol. 27, pp. 457～472, 1965. 11
5) 日本工業規格原案：電子計算機プログラム用言語 ALGOL, JIS 原案説明会資料，日本電子協，1966. 5
6) J. McCarthy 他：LISP 1.5 Programmer's Manual. MIT Press, 1961
7) 西村：木表現とリスト処理の算法，第 7 回プログラミングシンポジウム報告集 C-1, 1966. 1

（昭和 42 年 4 月 3 日受付）