数式処理言語ALとその処理方式†

池原 悟†† 岡田 博††

数式処理システムのインプリメンテーション方法についてみれば，LISP をホスト言語とするシステムでは数値計算も数式処理と同様のリストを介して処理しているため数値計算の性能が劣る．一方 FORTRAN, PL/I をホスト言語とするシステムでは，数値計算速度が速いという利点があるが，通常プリコンパイル方式がとられるため，会話処理に適さずまた会話処理と一括処理の連続性に欠ける．

これに対し数値計算処理が速く，会話処理と一括処理を統合したクローズドタイプの数式処理言語 AL (Algebraic Language) を開発し，その評価を行った．

本論文では AL における数式処理と数値計算機能の混合形式，数式表現を構成する名詞要素と動詞要素の扱い方，その他の言語処理方式，評価結果等について述べる．AL における言語処理方式により，数値計算速度が向上していること，クローズドタイプの数式処理言語ではインタプリタ方式はコンパイラ方式に比べて悪くないこと，また代表的な多項式処理を例にとり，簡単化機能のきめ細かな使用により処理時間が短縮できること，n-ary トリーデータ構造はバイナリートリーデータ構造に比べて2割近くメモリ節約が可能なこと等が示される．

1. まえがき

計算機を用いて数式処理を行うシステムとして FORMAC[1], ALTRAN[2], MATHLAB[3], REDUCE-2[4], MACSYMA[5], SCRATCHPAD[6] など多くのシステムが発表されており，また特殊目的のプログラムの研究[7]や数式処理アルゴリズムの研究[8]，応用技術の研究[8]なども行われている．

数式処理プログラムのインプリメンテーションの方法についてみれば，パッケージタイプのプログラムから LISP, FORTRAN, PL/I などの言語をホスト言語とする言語処理システムへと移行して来ている．ここで LISP をホスト言語とするシステムでは数値計算も数式処理と同様のリストを介して処理しているため数値計算処理の性能が劣るのに対して，FORTRAN, PL/I をホスト言語とするシステムでは数値計算速度が速いという利点があるもののこれらのシステムではプリコンパイラ方式がとられているため，会話処理に適さず，会話処理と一括処理の連続性に欠けるという欠点があった．

筆者等は数値計算処理が速く，かつ会話処理と一括処理の両方で用いられる数式処理言語 AL[9)~11)] (Algebraic Language) を開発した．AL は従来のパッケージタイプの数式処理プログラムやホスト言語をもつ数式処理システムと比べて以下の特徴をもつ．

（1） ホスト言語を持たない独立した言語であり，数式処理と数値計算の両機能が単一のクローズドタイプの言語として実現されている．

（2） AL はプログラミング言語としての機能を有し，利用者プログラムを作成するために用いられると同時に会話処理のコマンド言語としても使用される．

（3） 会話処理においては一文単位の入力と実行を行うだけでなく一まとまりの文をまとめて実行することもできる．複数の文をまとめて実行させるためには実行単位を DO 文と END 文でかこめば良い．

（4） 一括処理によって作成された利用者プログラムを会話処理の中で随時参照して実行させることができる．

本論文は言語 AL の設計思想と言語処理方式およびその評価結果について述べる．

2. 言語ALの設計

数式処理は本来その機能の一部として数値計算機能をもつ．しかしその実現上の理由から処理速度が遅いのが普通である．そこで AL では数式処理と数値計算の手続きを言語の表現上分離した．

2.1 数式表現と算術表現

AL プログラム構成要素を大きく分類すると，数式処理を行う数式表現と数値計算を行う算術表現になる．各々のシンタックスは**図1**のとおりである．

1）数式表現（数式処理手続き）

数式表現が算術表現と異なる点は〈数式一次子〉の構成である．アトミック定数は自分自身を値とし，関

† AL and It's Interpreter by SATORU IKEHARA and HIROSHI OKADA (Yokosuka Electrical Communication Laboratories, N. T. T.).

†† 日本電信電話公社横須賀電気通信研究所

〈AL の表現〉→〈数式表現〉|〈算術表現〉
〈数式表現〉→〈第一種数式表現〉|〈第二種数式表現〉
〈第一種数式表現〉→〈数式項〉|〈加減演算子〉〈数式項〉
|〈第一種数式表現〉〈加減演算子〉〈数式項〉
〈数式項〉→〈数式因子〉|〈数式項〉〈乗除演算子〉〈数式因子〉
〈数式因子〉→〈数式一次子〉|〈数式因子〉〈ベキ乗演算子〉〈数式一次子〉
〈数式一次子〉→〈アトミック定数〉|〈システム定数〉
|〈第一種アサイン変数〉|〈数式処理関数〉
|〈数学関数〉|〈関数変数〉
|(〈第一種数式表現〉)|〈符号のない定数〉
〈第二種数式表現〉→〈第一種数式表現〉=〈第一種数式表現〉
|〈第二種アサイン変数〉
〈算術表現〉→FORTRAN の算術式と同等

図 1 AL 表現の構文
Fig. 1 Syntax of AL-expressions.

数変数は具体的な関数形式の定義されていない関数名で，やはり自分自身を値とする名標である．これらが数式表現の中から消去されない限り，数式表現の演算結果はやはり数式表現である．第一種アサイン変数は数式処理の結果得られた数式表現を値としてもつ変数であり，数式処理関数は微分・積分等の計算を行う関数である．これらは共にある値(計算結果得られた式)によって置き換えられる要素である．

AL では数式表現の構成要素をその使われ方から，動詞要素（演算の結果消滅もしくは他の値に置き換る部分）と名詞要素（動詞要素のオペランドとならぬ限りそのまま残る）に分類している．

たとえば，

$$y=2x\,; \qquad (1)$$

$$z=2y+\frac{d}{dx}\left\{\sin(x)+\int xe^{x^2}dx\right\}\,; \qquad (2)$$

において(2)式の d/dx, $\int dx$ はそれぞれ微分，積分の実行を指示する動詞要素であり，y は(1)式の結果で置き換えられるので動詞要素である．$2, x, \sin(x)$ およびその他の演算子（加算，ベキ算など）は名詞要素である．動詞要素を名詞化するには引用（'を付与）を用いる．

2）算術表現（数値計算手続き）

算術表現は PL/I のスカラ式または FORTRAN 算術式と同等のシンタックスを有する数値計算手続きであり，数式表現とは表現の属性が異なる．このため，両者が計算結果を相互に参照するときは特に用意されている関数を使ってデータ属性の変換をすることが必要となる．このため言語処理系においてはそれぞれの表現の計算に適した方式をとることができるので，処理速度の向上が図れるという利点がある．

2.2 宣言確立の方法

会話，一括の処理形態を問わず，名標の属性を利用者が決定できるのが便利である．AL では DCL 文による明示宣言と暗黙宣言（FORTRAN に同じ）の他に，文脈宣言と動的宣言の 2 つの宣言方式を導入した．

i）文脈宣言

① 文脈により数式表現と算術表現の区別を可能とするため，数式代入文と算術代入文の代入演算子を分け，それぞれ ==, = を用いることとした．

② 属性を決める手がかりのないとき（入出力文と CALL 文にはじめて現われた名標）はアトミック定数とする．

ここで①は数式処理手続きと数値計算の混在したプログラムを見易くする効果がある．②の約束で実用性をそこなうことはない．

ii）動的宣言

既にアサイン変数として属性の定まった変数のもつ値を実行時に消去し，その名標をアトミック定数化すること，逆にアトミック定数として用いられていた名標をアサイン変数化することを可能とするため，両変数の属性を動的に定め，相互に変化できることとした．属性の変更は実行時の文脈に従って行われる．

2.3 数式の入力とプログラム間数式の授受

AL では GET 文，CALL 文の実行により，数式をプログラムに読み込むことができるので，動的に新しい名標が出現することがある．入力された数式は会話実行（一文実行モード）における代入文投入時と同様に構文解析が行われ，正しい構文の数式のみが数式格納領域に展開格納される．入力された数式がすでにアサイン変数として属性の定まった変数を含むときは，引用されたものと見做して局所的にアトミック化を行う*．

3. 言語処理方式

一般に言語処理方式としてインタプリタ方式とコンパイラ方式がある．従来の数式処理システムのうち，LISP をホスト言語とするシステム[3)~6)]はインタプリタ方式，FORTRAN, PL/I をホスト言語とするシステム[1),2)]はプリコンパイル方式を採用している．

AL では以下の理由でインタプリタ方式を採用した．

1）数式処理言語の言語処理では処理時間におい

* 論理的には入力された数式をプログラム中の数式処理手続きとみて動詞要素の計算を実行することも可能である．このようにすればプログラムをデータとして入力し実行することが可能となるが，言語処理系を簡単化するため，入力される数式はすべて名詞要素からなるものと規定した．

て，インタプリタ方式とコンパイラ方式の間に大きい差異はないと思われる．

コンパイラ方式では数式表現中の動詞要素を機械語化したオブジェクト・プログラムが生成される．各々の動詞要素に対応するオブジェクト・プログラムのほとんどは任意の数式を処理の対象とする計算ルーチンとしてあらかじめ用意されたものとなり，インタプリタ方式の場合に用意されるものと同じである例が多い．多くの問題*ではこれらの計算ルーチンの走行時間が全体の処理時間の大部分を占めると予想される．

2）コンパイラ方式ではオブジェクトの発生が複雑である．

数式表現を構成する名標には動的に名詞要素と動詞要素の間で属性の変更されるものがあり，これに対応する機械語は動的に名標属性を判定して動作を決める（オブジェクト・プログラム自身がインタプリタのようになる）ことが必要である．

ところで，AL 言語では数式表現と算術表現を分けて扱うので，AL の言語処理方式は従来のインタプリタ方式（LISP をホスト言語とする）と下記の点が異なる．

1）従来のインタプリタ方式によるリスト処理言語の言語処理においては一般にプログラムとデータをいずれもトリー構造に展開しているのに対して，AL はデータをトリー構造，プログラムを中間言語に変換してテーブル構造に展開している．このため，プログラム解釈時のトリー操作が不要となる．

2）数式表現と算術表現の中間言語を同じ RPN (Reverse Polish Notation) 形式（**図2参照**）とすることによって，中間言語作成方法の簡素化をはかる一方，実行時には両者を分散して処理の高速化を可能としている．

3.1 数式表現と算術表現の計算

図2に2章で示した（1），(2)式が中間言語に変換された後の解釈実行の途中状態を示す．中間言語に展開された入力文はインタプリタによって解釈実行され数式格納領域に n-ary トリー構造のデータが生成される．この過程で，中間言語内に出現する動詞要素ごとにインタプリタから実行時ルーチンが起動されて計算処理が行われ，数式格納領域には名詞作用素のみで構成された数式が生成される．

次に，算術表現の計算では中間言語内の変数へのポインタがその変数の値を格納する番地を示すため，スタック上で直接計算を行うことができる．

数式表現の計算がトリー上で行われるのに対して算術計算ではトリー操作がなく，処理速度が向上する．

3.2 定義文の実行方式

数式表現の中で変数として扱われる関数変数に対して，その関数の具体的な関数形式や微分された後の関数形式が FUNC 文，DERIV 文によって定義できる．

（1） 関数定義文（FUNC 文）の処理方式

図3に関数 $F(N)$ を定義する例を示す．ここで第一式で定義された関数 $F(N)$ の具体的な関数形式は数式表現の中間言語と同じ形式に展開されており，第二式でこの関数が参照されたとき，引数Nに値3が代入されて EVAL $(2^3\sqrt{3})$ が計算され，その値が変数 Aの値として代入される．

以上のように関数定義文によって定義された関数形式は他の数式表現と同様に中間言語と同一形式で翻訳

$$y = 2x \qquad \cdots\cdots(1)$$

$$z = 2y + \frac{d}{dx}\left\{\sin x + \int x e^{x^2} dx\right\} \cdots(2)$$

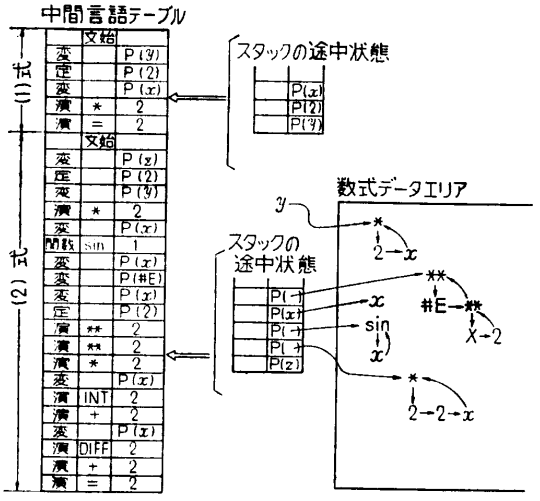

図 2 数式表現の実行法
Fig. 2 Execution of formula expressions.

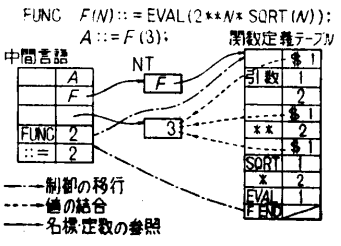

図 3 関数定義文の処理方式
Fig. 3 Execution of defind function.

* ここでは計算手順は明らかであるが手計算により計算を実行するのが複雑で困難な問題を想定している．

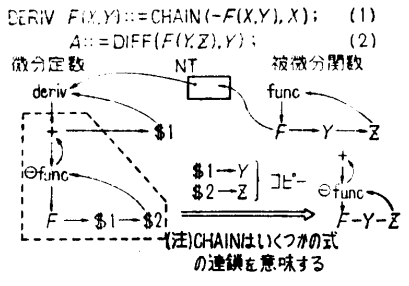

図 4 微分定義文の処理方式
Fig. 4 Execution of derivative defind function.

され参照の都度数式格納領域に展開されるので関数定義文としての特殊な処理は必要としない.

(2) 微分定義文 (DERIV 文) の処理方式

図4に関数 $F(X, Y)$ の微分形

$$\frac{\partial}{\partial X}F(X, Y)=-F(X, Y),\quad \frac{\partial}{\partial Y}F(X, Y)=X$$

を定義し, これを用いて微分を実行する例を示す.

ここで定義された微分形は1つのトリー構造で保存されており, 図中第二式の実行でこれが参照されると, 仮引数 $1, $2 が実引数 Y, Z におきかえられて, 結果として $-F(Y, Z)$ が A の値として代入される.

このように微分定義文では定義された微分形は数式格納領域に木構造データ形式で保存されるので, プログラム実行中に微分形を定義し直すことや, 微分結果に自分自身を含ませる処理が容易に実現できた.

3.3 簡単化実行方式

数式を整理し見易くするために最も良く使用される変形規則を選び簡単化機能とした. 簡単化には3つのレベルがある. 全く簡単化しないレベル 0, 数値0と1の消去を中心としたレベル 1, 項の並べかえ, 約分などを行うレベル2の3レベルである.

簡単化の契機は利用者の指示に基づく方法とシステムの判断に基づく方法である. 前者では, 利用者が指示したレベルに従い, 代入文実行時に常に自動的に簡単化する方法と, 数式の必要な部分にのみ簡単化を行うことのできる関数による方法がある. 後者は数式処理関数の内部で行われるもので数式処理関数の機能の一部をなしている.

レベル1程度の簡単化は数式の上を一通り歩くことで目的を達するが, 高レベルの簡単化はある簡単化が次の簡単化の契機となることが多い. たとえば,

$$A^B * A^{-B} \underset{(*)}{\Rightarrow} A^{B-B} \underset{(+)}{\Rightarrow} A^0 \underset{(**)}{\Rightarrow} 1$$

では順に *, +, ** のオペレータに着目した簡単化が適用される. トリー構造上の演算子セルに AS (Already Simplified) ビットを設ければ簡単化の部分的終了を表示し, 無限ループの防止と効率向上がはかれる.

3.4 誤りの回復処理

誤りには入力文の構文誤りと実行時に発生する誤りがある. 会話処理ではこの両者を救済し, 誤りを発見した場合はその文の入力前の状態に戻る.

構文誤りの回復のためにはあらかじめソース文上で構文チェックをし, 正しい文のみの翻訳を行えば良いが, この方法はソース文の走査回数が増え能率が悪い. そこで新規に現われる名標等のテーブルのエントリを文単位に一時的にチェイニング方式で管理して, 誤り発生時にその文で新規に現われた名標のみを消去できるようにすれば, ソース文の走査回数を減らすことができる.

次に実行時誤りの回復のために, 数式データの多重参照方式をやめ, 必要の都度コピーを作っていく方式としている. この方式は正常処理時には, エリア使用効率, 実行時間共にある程度の損失はあるが, 実行時誤りの救済処理が非常に容易である.

数式処理では処理途中でデータが極度に膨張することが多い. これに対処するため, AL は自動 SAVE, LOAD 機能をもっている. 数式データエリアのオーバフロー発生時には一文廃棄したあと, 残っている数式をすべて二次記憶に退避させる. 退避された数式がその後参照されると, 参照された数式のみを主記憶にロードし, 処理する.

4. 会話処理と一括処理の関係

数式処理は会話処理, 一括処理がそれぞれ別の役割を持ち共に重要であるので, AL では会話系, 一括系の2つの処理系を作成した. 両処理では言語仕様を統一すると共に, 処理系を構成するモジュールのほとんどを共通設計している.

会話処理, 一括処理におけるシステムの状態遷移を図5に示す.

1) 会話処理……一文ごとに図5の破線の状態を遷移し, 実行後入力待ち状態に戻る. DO グループの場合は一文ごと構文チェックのみを行い, グループ全体の入力が完了してから実行される. CALL 文の場合は, コール先の中間言語 (プログラム) を読み込んだあと実行管理部が一括処理状態に移る点に特徴がある. すなわち, 会話処理が一括処理を包含している.

2) 一括処理……プログラム単位にコンパイルさ

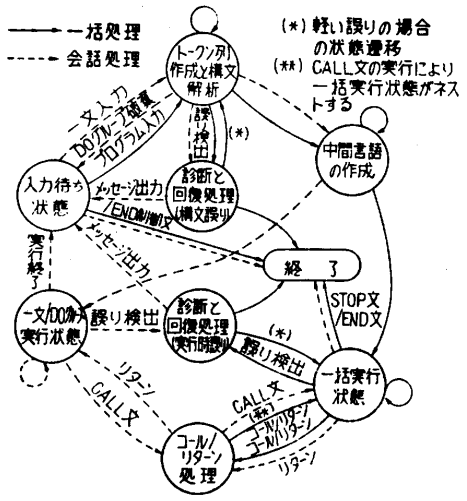

図 5 システム状態遷移図
Fig. 5 System states transition diagram.

れ，プログラム全体の中間言語が実行管理部に渡される．これを一括して実行した後，処理は終了となる．

なお一括処理では誤り検出後は処理は中止されるが，会話では3.4節で述べた方法で一文入力前の状態を回復して，入力待ち状態に復帰する．

5. 言語と処理系の評価

会話系，一括系両システムは各々 40 k ステップのプログラムで共通部分が 23 k ステップとなった．

一般にプログラミング言語の評価ではシステムの使い易さや言語のわかり易さといった利用者習性に依存した評価と目的とする問題がうまく記述できるか否かを判断する機能上の評価，さらに処理方式等処理系の性能に関する評価が必要である．第1の観点では運用後の多くの利用者の意見を集約分析する必要があるため今後の問題として残し，ここでは第 2，第3の観点から評価する．

具体的にはできるだけ現実の実用に即した評価を行うため以下の方法をとった．まず従来から数式処理の研究で扱われて来た問題として SIGSAM Bulletin など多くの文献で発表されている問題 52 題を収集し，その中から類似のものと解の得られていないものを削除して**付表1**に示す 26 題を評価用の問題として選定して，AL でプログラムを作成し解を求めた．

5.1 言語機能の評価

前述のようにしてとり上げた 26 題の問題の中で4題（付表1の例題番号 23～26）は AL の記述能力不足のためプログラム作成を打切った．作成した 22 題のプログラムの総ステップ数は 1462 ステップでプログラム平均 66.5 ステップの記述量であった．

記述能力不足の原因は①多倍長整数演算機能がないこと*（例題番号 23），②指定変数に着目した因数分解が必要（例題番号 24），③三次方程式が解けないこと（例題番号 25），④微分方程式が解けないこと（例題番号 26）などであった．これらのうち②～④は他の機能を用いてプログラム化すれば良いが，AL の単一の文で表現できないということで範囲外とした．

以上の問題の他に研究所内外の利用者から寄せられた問題の一部を**付表2**に示す．これらの問題では連立方程式の解法で AL では10元以下とする制限があるため多少の工夫を必要とした他はいずれも容易にプログラムを書くことができた．但し，一部の問題では TSS で使用可能なメモリの制限内で解を得るためさらに若干の工夫が必要であった．

5.2 文脈宣言導入の効果

付表1の実際に解を得た問題 22 題のプログラムで使用されている変数の数と配列のための明示宣言必要量等を調べた．それによると使用された変数は全体で 432 変数あり，そのうち数式変数 271 個，算術変数 161 個であった．またこれらのうち明示宣言の必要な変数が 70 個である．従って1プログラム当り 19.6 個の変数を使用しており，そのうち 3.2 個の変数を明示宣言していることになる．文脈宣言の機能が無いとすると約6倍に宣言必要変数が増え，文脈宣言導入の効果は明らかである．

5.3 処理時間の考察

会話系，一括系の違いは使用できるメモリ量のみで処理時間は等しい．

処理時間に影響を与える要因ならびに数値計算の処理時間について考察する．

1）簡単化処理時間

図6に $f \cdot g$ 級数計算[13]（例題番号 14）の AL における処理時間と計算次数の関係を示す．**図7**はルジャンドル多項式（例題番号 7）の計算時間を簡単化レベルごとに示したものである．これらの図より全処理時間に占める簡単化（レベル 2）処理時間の比は計算の次数が増大するにつれて大きくなり，簡単化時間の短縮が重要な課題であることが分る．ところできめ細い簡単化の適用を可能とする AL の簡単化レベルの設定機能により，処理時間の短縮をはかることができる．例えばルジャンドル多項式 10 次項を求める場合は簡

* この機能はその後 AL-1 E で追加された．

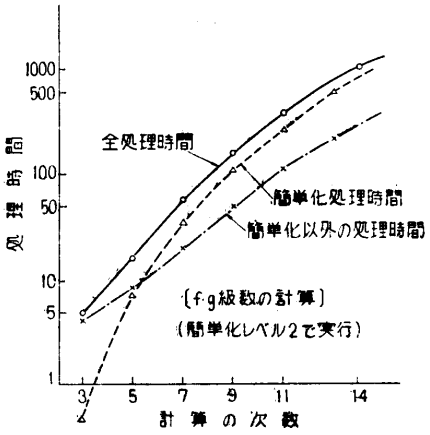

図 6 実行時間比率
Fig. 6 The simplification time and the other execution time.

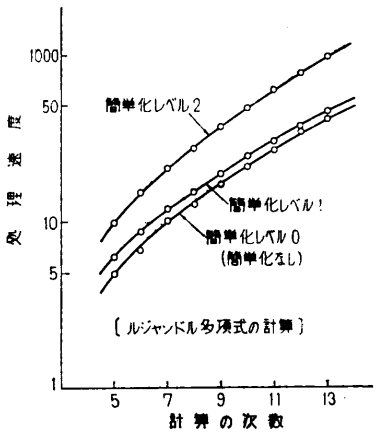

図 7 簡単化レベルと実行時間
Fig. 7 Simplification level and execution time.

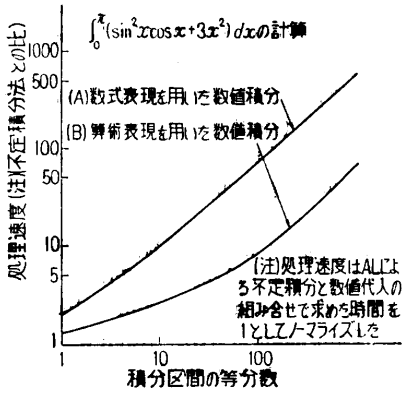

図 8 AL による数値積分時間
Fig. 8 Numerical integration time by AL (Simpson method).

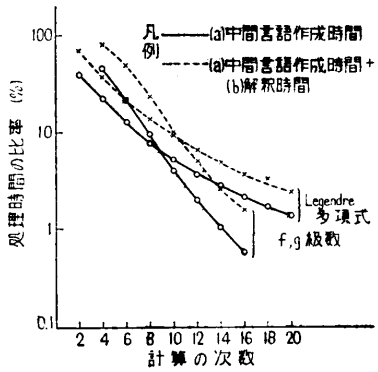

図 9 処理時間比率
Fig. 9 Processing time ratio vs. computation degrees.

3) 数値計算と数式処理の比較

3.1 節で述べたように数式表現と算術表現の処理方式を分けたことにより算術表現の計算速度はリストを介した場合に比べてかなり向上した．例えば付表1の例題番号 22 などでは全体の処理時間が処理方式を分けない場合の 50% に減少している．

処理方式による処理速度の差を見るためにさらに区分求積法による定積分を数式表現を用いて行ったときと算術表現を用いた場合の処理時間の例を図8に示す．これよりシステム初期化の影響の小さくなる積分区間等分数 100 以上の領域でみると，算術表現は数式表現に比べて約1桁計算速度が速く，仕様上両者を分解して実現した効果が現われている．

また，この例題では AL の不定積分機能と数値計算機能を組み合せて答を得ることができる．その方法は数値計算による区分求積に比べて十分速く，原理的に誤差もない．

単化レベル0で 10 次まで計算した後，式を整理するため第2レベルの簡単化を1回施せば良い．このときの処理時間は簡単化なしで 10 次までの処理時間に簡単化1回分の処理時間を加えたもので，毎回簡単化を施すのに比べて2倍近く速くなる．

2) データ構造の影響

図6,7の曲線はいずれも計算次数の増大と共に実行時間がほぼ指数関数的に増大することを示している．これは AL データ構造の汎用性 (n-ary トリー構造) によるもので，数式データ量が指数関数的に増大するに伴い，処理時間も同じ曲線で増大していることを意味する．なおこの種の多項式処理は専用データ構造の導入で高次の計算速度を相当〔1桁程度〕上げられると推定している．

5.4 インタプリタ方式の評価

図6,7の例題の処理時間を(a)中間言語作成時間，(b)中間言語解釈時間，(c)実行時ルーチンに分けて実測した結果を図9に示す．図から手計算の容易な低次の計算次数を除き，全処理時間に占める実行時ルーチンの走行時間の比が最も大きく，計算次数*を増すにつれ，中間言語の作成，中間言語の解釈時間は無視できるようになる．この傾向は5.1節の評価で用いたほとんどの問題で共通している．

これらの問題をコンパイラ方式で処理した場合は(b)の時間が無くなる代りに(a)の処理で機械語の生成まで行われることによって (a) の時間が増大するが，最も大きい(c)の処理時間は影響を受けない．し

* 多項式処理の場合は次数，連立方程式の場合は元の数，マトリックスの場合は行および列の数を示す．

たがって計算量が多く手計算の困難な問題を対象とする数式処理では実行時ルーチンの走行時間の全処理時間に占める比が大きく，コンパイラ方式とインタプリタ方式の間の性能の差は小さい．

5.5 メモリ使用量の考察

内部数式表現として n-ary 型のデータ構造を採用したことにより，実測の結果バイナリトリーに比べて17～20% のメモリ節約ができた．

数値計算ではみられない実行時の使用メモリ量の増加は図6の傾向と等しく，計算次数に対してほぼ指数関数的に増加するが，そのうち 20～30% は数値定数の増加である．従って数値定数は数式データ領域と合わせてプール管理することが適当である．

6. むすび

数式処理言語 AL における数式処理と数値計算機能の混合形式，数式表現を構成する名詞要素と動詞要素の扱い方，その他の言語処理方式を示した．

本論文では(1)数式処理手続きを表わす数式表現と数値計算手続きを表わす算術表現を分けることにより，数値計算速度が向上したこと，(2)AL のようなクローズドタイプの数式処理言語においてはインタプリタ方式はコンパイラ方式に比べて悪くないことが示された．また代表的な多項式処理を例にとり，(1)簡単化機能使用法により処理時間を短縮できること，(2)n-ary トリーデータ構造はバイナリトリー方式に比べて2割近くメモリ節約ができること，(3)処理途上の数式データ膨張において 2～3 割は数値定数の数の膨張であること等が示された．なお，会話処理と一括処理の連続性を実現する方法は TSS における各種言語処理でも応用できると思われる．

今後はここで示した動的宣言の範囲を数式変数と算術変数まで拡大して数式処理と数値計算のよりスムーズな連絡の確保方法および，特殊問題向きデータ構造，トランケーティド・パワーシリーズ処理*，新規アルゴリズムなどの検討を進め，使い易さと処理速度の向上をはかる予定である．

謝辞 おわりに本研究を進めるに当り適切な御指示を頂いた当社東海電気通信局柴山データ通信部長，当研究所戸田統括役，徳山調査役，施設局小泉調査役，関東通信局大野調査役に深謝する．また処理系の設計製造，デバッグツールや，テストデータの作成等に多大の御協力を頂いた池田，神原，小田，角田，今福の各主任ならびに日本電気株式会社の関係各位に深謝する．

* 級数計算においてある一定の次数以上の項を無視して計算を進める処理をいう．

参考文献

1) Sammet, J. E. and Bond, E. R.: Introduction to FORMAC, IEEE Trans., E. C. Aug. pp. 386-394 (1964).
2) Hall, A. D.: The Altran System for Rational Function Manipulation—A Survey, Commun. ACM, Vol. 14, No. 8, pp. 517-521 (1971).
3) Engelman, C.: MATHLAB 68 Information Processing, 68, pp. 462-467 (1969).
4) Hearn, A. C.: Reduce 2 User's Manual, Version 2, UTAH 大学 (1973).
5) MACSYMA Reference Manual, Version 6, MIT (1974).
6) Griesmer, J. H. and Jenks, R. D.: SCRATCHPAD A Capsule View, SIGPLAN Notice Vol. 7, No. 10, pp. 93-102 (1972).
7) 渡辺：数式処理をめぐって，数学，Vol. 24, No. 1, pp. 28-38 (1972).
8) たとえば，Proc. of the 1976 ACM Symposium on Symbolic and Algebraic Computation, (Aug. 1976).
9) 徳山，池原，岡田：数式処理言語，通研実報，Vol. 24, No. 1, pp. 199-218 (1975).
10) Tokuyama, G., Ikehara, S., and Okada, H.: Algebraic Language (AL), Review of the E. C. L. Vol. 23, No. 9 (1975).
11) DEMOS-E AL 説明書，DEMOS-E 4071，日本電信電話公社 (1975).
12) たとえば，Sundblad, Y.: One User's—One Algorithm Comparison of Six Algebraic System on the Y_{2^n}-Problem, SIGSAM Bulletin No. 28, Dec. pp. 14-20.
13) Sconzo, P., Leschack, A. R., and Tobey, R.: Symbolic Computation of f and g Series by Computer, The Astronomical Journal, No. 1329 (May 1965).
14) 戸田：FORMAC, bit, 8月号, p. 849 (1974).
15) SYSTEM/360 PL/I FORMAC, IBM Data Center User's Guide.
16) Tobey, R., et al.: PL/I FORMAC Symbolic Mathematics Interpreter, IBM (1969).
17) 檜山：数式処理 (FORMAC) による因数分解，情報処理，Vol. 12, No. 5, p. 304 (1971).
18) Brown, W. S.: The ALPACK System for Nonnumerical Algebra on a Digital Computer, BSTJ, 42, p. 304 (Sept. 1963).
19) 戸田，鈴木：FORMACを使用してみて，第9回プログラミングシンポジウム報告集，C1，情報処理学会 (1968).
20) 文献 13) に同じ．

21) Rüdinger Loos: Analytic Treatment of Tree Similar Fredholm Integral Equations of the Second Kind with Reduce 2, SIGSAM Bulletin, No. 21, p. 32 (1972).
22) Campbell, J. A.: Problem #2—The Y_{2n} Functions, SIGSAM Bulletin, No. 22, p. 8 (1972).
23) Lew, J. S.: Problem, #3—Reversion of a Double Series, SIGSAM Bulletin, No. 23, p. 6 (1973).
24) D. Barton: Problem #4—The Lie Transform, SIGSAM Bulletin, No. 25, p. 12 (1973).
25) McCarthy, D. P.: Problem #6—The Kiang Problem, SIGSAM Bulletin, No. 28, p. 12 (1973).

付表 1 AL の評価に用いたプログラム一覧
A. Table 1 Problem list for evaluation.

* 出典は参考文献番号を示す.

番号	問題	説明	出典*
1	数学的帰納法	$1^3+2^3+\cdots+n^3=\{n(n+1)\}^2/2^2$ を数学的帰納法によって証明する.	14
2	行列の積の計算	非数値要素をもつ2行2列の行列の積を求める.	1
3	Taylor 展開(一変数)	一変数関数を近傍値と次数を与えて Taylor 展開する.	15
4	Taylor 展開(多変数)	多変数関数を近傍値, 変数の数, 次数を与えて Taylor 展開する.	16
5	$T_n=(1+1/n)^n$ の展開	T_n をベキ級数に展開して $1/n^5$ まで求める.	14
6	Hermite 多項式	微分公式と漸化式の2つの方法で Hermite 多項式を10次まで計算し, 両者が一致することを確かめる.	14
7	Legendre の多項式	項番6と同様の計算を Legendre 多項式について行う.	15
8	Tchebycheff の多項式	漸化式を用いて Tchebycheff の多項式の10次項まで求める.	2
9	Sammet の微分例	$D=1-HZ-H^2$ のとき $Y=(H+\sqrt{D}-1)^3/\sqrt{D}$ について dY/dH, d^2Y/dH^2 を求める.	14
10	二体問題	摂動のある二体問題の微分方程式を解くため, 離心近点距離角を平均近点距離角に変数変換する.	9
11	因数分解	有理整数を係数とするニモニックな n 次多項式を因数分解する.	17
12	プライオリティをもつ待ち行列システム	プライオリティのあるトラヒック解析において $G_m(\alpha)$ を求める.	18
13	不完全 Γ 関数計算用の山内の展開式	$P(x,m)=\int_x^\infty \frac{1}{\Gamma(m)}a^{m-1}e^{-a}da$ を求めるための山内の式を導く.	19
14	f,g 級数	天体力学の軌道計算で使用される f,g 級数の高次項を求める.	20
15	一階非線形連立微分方程式	近似計算のくり返しにより10元までの一階非線形連立微分方程式を解く.	16
16	Fredholm の第2種積分方程式	Fredholm の第2種積分方程式3題をとく. 4×4 の行列式の値の計算を含む.	21
17	微分方程式の級数近似解法	2階の微分方程式を級数展開による近似解法で解く. (Y_{2N} の計算)	22
18	Shrödinger 方程式の固有値問題(2重級数の反転)	一次元の時間独立な Shrödinger 方程式であるポテンシャルが与えられたときの近似解法	23
19	Hamilton の正準方程式の Lie 変換による解法	一組の連立微分方程式を変数変換により近似的に解く.	24
20	Kiang Problem 部分分数展開	$\prod_{j=1}^{i}\frac{1}{\lambda^2-(x+aj)^2}$ を $\sum_{j=1}^{i}\left\{\frac{A_j}{\lambda-(x+aj)}+\frac{B_j}{\lambda+(x+aj)}\right\}$ の形に部分分数にしたときの $\Sigma(A_j-B_j)$ を求める.	25
21	Runge-Kutta 法の打切り誤差	Runge-Kutta 法の打切り誤差を Taylor 展開により求める.	14
22	級数展開と誤差解析	$f=\sum_{i=1}^{50}a^i t^{i+1}x^{i-1}$ を誤差解析を行いながら級数展開する.	1
23	n 乗根の計算	n の i 乗根を求める近似式により無限長演算を用いて n の i 乗根を求める.	2
24	Schrödinger 方程式の例	Schrödinger 方程式の固有値問題の計算途中で必要となる計算.	3
25	微分方程式の1つの解法(1)	$d^3y/dx^3+ady/dx=\sin 2x$ を1つの解法により解く.	3
26	微分方程式の1つの解法(2)	上記問題の別解法	3

付表 2 AL が使用されている問題の例*
A. Table 2 Examples of practical use.

番号	問題	説明	問題提出のあった専門分野
1	連立方程式の解法(1)	非数値係数の 12 元連立方程式の解法	伝送
2	級数近似(1)	音響振動の伝播特性解析	音響
3	級数近似(2)	アバランシェダイオードの温度特性の解析	半導体
4	漸化式の計算	マルチプロセッサメモリコンテンションの解析で必要な5つの漸化式による高次項の導出	データ通信
5	ネットワーク型待ち行列の解析	BCMP 型のネットワーク型待ち行列の積率母関数から微分法および級数展開法により高次項の係数を求める.	データ通信
6	連立方程式の解法(2)	プロセッサ間通信方式の解析で必要な 63 元連立方程式の解法	データ通信
7	連立方程式の解法(3)	構造解析で必要な連立方程式の解法	建築
8	行列式の計算	フィルタ回路解析での行列式の計算	伝送
9	連立方程式の解法	マスストーレッジシステム[13] (MSS) の動作解析	データ通信
10	判別式の導出	4次方程式の判別式を導く	数学

* DEMOS-E のサービスとして一般に解放して以降については用途の確認は困難であり, AL 作成初期において寄せられた問題を示す.

(昭和51年9月8日受付)
(昭和53年9月5日採録)