数式処理システムパーザの論理設計

中川　重和
倉敷芸術科学大学産業科学技術学部

(1998 年 9 月 30 日 受理)

1　はじめに

現在までに数式処理システムは，商用非商用もしくは汎用専用を問わず，多数開発されてきた．それらに搭載されるアルゴリズムも版を重ねる毎に洗練されてきている．また，商用システムの多くは従来に比べ安価に入手可能であり，プラットホームに依存することなく (UNIX, Windows9[5,8], MacOS のいずれの OS でも稼働) 使用可能である．一方，非商用システムのほとんどもインターネット経由で入手できる (`http://www.can.nl/` からは数式処理研究の最新の動向を得ることができる．もちろん，非商用システムの入手情報もこのページから得ることができる)．つまり，現在はユーザが求める数式処理システムはすぐに手に入る状況である．事実，筆者が手元で利用できる数式処理システムを列挙してみると，MapleV [5]，Mathematica [9]，REDUCE [7] (以上は商用)，RISA/Asir [3]，CoCoA [6] (以上は非商用) などがある．

このような状況の下，なぜ新たに数式処理システムを構築する必要があるのかを考えてみる．主な理由は

- 実際に数式処理システムを開発することで，システム開発に関する教科書 (例えば，[2]) からでは得られない知見を獲得したい，
- 既存の数式処理アルゴリズムの実装実験をしたい，
- 新規に開発する数式処理アルゴリズムの実装を自前で実行したい，

などである．

今後筆者の研究室で数式処理システムを開発する予定であるが，本報告はそのための第 1 段階であるパーザ部分の論理設計についての報告である．

第 2 節では，数式処理システムの構成について触れ，要求仕様を述べる．第 3 節では，数式の外部表現と内部表現を規定する．さらに第 4 節では，実装について議論する．

2　数式処理システム

一般に，数式処理システムにおける数式の処理手順は 5 つのフェイズ，1) 入力，2) 数式の評価，3) 数式の (狭義の意味での) 計算，4) 数式の簡単化，5) 出力，からなる．数式処理システムの基本的構成単位はパーザと計算エンジンである．パーザはフェイズ 1 から 2 への橋渡しの役目を担う．一方，計算エンジンが担当するのは，フェイズ 2 から 3 を経由して 4 への橋渡しの役目である (図 1)．

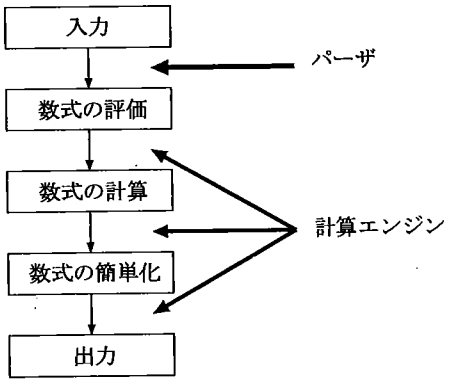

図 1: 数式処理システムの基本的構成

さて，システム設計の第 1 段階では，パーザの仕事，つまり数式をその外部表現 (ユーザからみた入力のための数式表現) から内部表現 (数式処理を効率的に実行するのに都合のよい表現) への翻訳 (構文解析) を明確にすることであり，それは数式の外部表現と内部表現を定義することと同義である．

どのような外部表現や (特に) 内部表現を採用するかという問題はシステムの性格や性能を大きく左右する問題である．逆にいうと，システムが専用か汎用か，あるいはどのようなデータ構造を頻繁に取り扱うかを決定すれば，採用すべき外部表現および内部表現がおのずと決まる．

我々の場合，それは「多変数多項式計算の高速処理」を指向した専用システムを目指している．対称式の基底変換の計算 [8]，Gröbner 基底 [4] の計算などが高速になるようにしたいのである．その機能としては

- 多倍長整数および多倍長有理数の取り扱い
- 多変数多項式の四則演算
- 項順序の設定
- 多変数多項式の操作 (頭項，係数を取り出すなど)
- GCD 計算
- Buchberger's algorithm

等を完備する必要がある．

3 数式の外部表現と内部表現

多項式の表現は通常，ほとんどあらゆるシステムが加減乗演算 +，-，* やベキ乗演算 ^ を内置演算子として入力を線型化している．例えば，$3x^2y^3+2yz^2$ は `3*x^2*y^3+2*y*z^2` である．我々も例にならって，図 2 にあるように (BNF 記法による) 数式の外部表現を定義する．

⟨ 多項式 ⟩ ::= (⟨ 多項式 ⟩) | ⟨ 項 ⟩ ⟨ 二項演算子 ⟩ ⟨ 多項式 ⟩ | [+,-] ⟨ 多項式 ⟩
| ⟨ アトム ⟩
⟨ アトム ⟩ ::= ⟨ 変数 ⟩ | ⟨ 数 ⟩
⟨ 項 ⟩ ::= ⟨ 数 ⟩ ⟨ べき積 ⟩
⟨ べき積 ⟩ ::= ⟨ 変数 ⟩ ^ ⟨ 正整数 ⟩ | ⟨ べき積 ⟩ * ⟨ べき積 ⟩
⟨ 二項演算子 ⟩ ::= + | - | *
⟨ 数 ⟩ ::= ⟨ 多倍長有理数全体 ⟩
⟨ 変数 ⟩ ::= ⟨ 文字列 ⟩

図 2: 多項式の外部表現の定義

一方，多項式の内部表現には再帰的表現，分散表現などがある．例えば，REDUCE では再帰的表現 (図 3 参照) が，Maple では分散表現 (図 4 参照) が，また Risa/Asir ではその両方の表現 (タグによってそれらを明示し，との時々に応じて切替えている) が採用されている．

再帰的表現は，次の例

$$(x+y+z)^2 = 1\cdot x^2 + (2\cdot +(2\cdot z))\cdot x + \Big((2\cdot z)\cdot y\Big(1\cdot z^2\Big)\Big)$$

にあるように特定の変数を主変数 (この場合は x) とする 1 変数多項式で，その他の変数はその変数の 1 変数多項式の係数に主変数を含まない多項式として現れる．この係数が，また，ある変数 (この場合は y) を主変数とする 1 変数多項式となっているような表現をいう．これに対し，多項式を

$$(x+y+z)^2 = 1\cdot x^2 + 2\cdot xy + 2\cdot xz + 1\cdot y^2 + 2\cdot yz + 1\cdot z^2 \qquad (1)$$

のように，変数のべき積と係数の積の和として表現したものを分散表現と呼ぶ．

我々の目的の一つである Gröbner 基底 [4] の計算においてはは，単項式に注目して操作を行うため多項式が分散表現されているほうが効率のよい演算が可能である．そのため，我々も図 5 に示した分散表現を採用する．単項式に関する操作としては

- term order による項の整列
  - term order $>$ とは以下の性質を満たす全順序である
    - $t > 1$ for all term t
    - $s > t \implies su > tu$ for all term s, t, u
  - (1) は $x > y > z$ とする辞書式順序であり，これは term order の例である
- leading monomial の抽出
  - ある term order $>$ が与えられたとき，分散表現多項式の leading monomial とは，$>$ に関して最大のべき積を示す
  - $x > y > z$ とする辞書式順序において，(1) の leading monomial は x^2 である

- leading coefficient の抽出
  - ある term order $>$ が与えられたとき，分散表現多項式の leading coefficient とは，$>$ に関しての leading monomial の係数を示す.
  - $x > y > z$ とする辞書式順序において，(1) の leading coefficient は 1 である

などがある．式 (1) の右辺の図 5 に基づく分散表現多項式は

```
1*<<2,0,0>>+2*<<1,1,0>>+2*<<1,0,1>>+1*<<0,2,0>>+2*<<0,1,1>>+1*<<0,0,2>>
```

となる.

⟨ 標準多項式 ⟩ ::= ⟨ 定数 ⟩ | (⟨ 標準項 ⟩ . ⟨ 標準多項式 ⟩)
⟨ 標準項 ⟩ ::= (⟨ 標準べき ⟩ . ⟨ 標準多項式 ⟩)
⟨ 標準べき ⟩ ::= (⟨ 核 ⟩ . ⟨ べき ⟩)
⟨ べき ⟩ ::= ⟨ 正整数 ⟩
⟨ 定数 ⟩ ::= NIL | ⟨ 整数 ⟩ | (⟨ 整数 ⟩ . ⟨ 整数 ⟩) | ⟨ 浮動小数 ⟩
⟨ 核 ⟩ ::= ⟨ 変数 ⟩ | ⟨ Lisp 前置式 ⟩ | ⟨ 標準多項式 ⟩.

図 3: REDUCE における多項式の内部表現の定義

⟨ 多項式 ⟩ ::= sum ⟨ 多項式リスト ⟩
⟨ 多項式リスト ⟩ ::= ⟨ 係数 ⟩ ⟨ 項 ⟩ | ⟨ 係数 ⟩ ⟨ 項 ⟩ ⟨ 多項式リスト ⟩
⟨ 項 ⟩ ::= product ⟨ 項リスト ⟩
⟨ 項リスト ⟩ ::= ⟨ べき ⟩ ⟨ 項 ⟩ | ⟨ べき ⟩ ⟨ 項 ⟩ ⟨ 項リスト ⟩
⟨ べき ⟩ ::= ⟨ 変数 ⟩ ⟨ 正整数 ⟩
⟨ 係数 ⟩ ::= ⟨ 多倍長有理数 ⟩

図 4: Maple における多項式の内部表現の定義

⟨ 多項式 ⟩ ::= (⟨ 多項式 ⟩) | ⟨ 項 ⟩ ⟨ 二項演算子 ⟩ ⟨ 多項式 ⟩ | [+,-] ⟨ 多項式 ⟩
⟨ 項 ⟩ ::= ⟨ 係数 ⟩ * ⟨ べき積 ⟩
⟨ べき積 ⟩ ::= << ⟨ 式並び ⟩ >>
⟨ 式並び ⟩ ::= ⟨ 空 ⟩ | ⟨ 非負整数 ⟩ | ⟨ 非負整数 ⟩ ⟨ 式並び ⟩
⟨ 二項演算子 ⟩ ::= + | - | *

図 5: 本システムにおける多項式の内部表現の定義

4 おわりに

我々の目指す多項式専用数式処理システム開発において多項式の内部表現についての設計が重要であった．次の段階は，実装である．パーザの実装には lex および yacc を利用するのが妥当であろう．その理由は，パーザの仕事が文脈自由文法 LALR [1] で記述された多項式の内部表現 (図 2) から多項式の内部表現 (図 5) への翻訳だからである．計算エンジンが実装されるまでの期間，パーザ実装のテストには既存の計算エンジンを代用する．例えば，REDUCE のその部分を代用するべく翻訳をおこなえばよい．もちろん，最終的には計算エンジンも C 言語で実装するつもりである．

参考文献

[1] A.V. エイホ，R. セシィ J.D. ウルマン；原田賢一訳 (1990) コンパイラ *I.* サイエンス社.

[2] 佐々木建昭・元吉文男・渡辺隼郎 (1984) 数式処理システム. 昭晃堂.

[3] 野呂正行・竹島卓 (1995) *Risa/Asir User's Manual ed 3.0beta.* 株式会社富士通研究所

[4] W. Adams and P. Lostaunau (194) *An Introdcuction to Gröbner bases.* AMS, Providence RI.

[5] Bruce W. Char *et al.*(1991) *MapleV Language Reference Manual.* Springer-Verlag.

[6] A. Capani and G. Niesi (1993) *CoCoa 3.0b User's Manual.* `http://ideal.dima.unige.it/`.

[7] A. C. Hearn (1994) *REDUCE User's Manual Ver. 3.5. RAND Pub.*

[8] S. Nakagawa (1994) *Symmetric Polynomial Calculus in Distribution Theory of Multivariate Statistics.* Ph. D. Thesis Kyushu University.

[9] S. Wolfram (1991) *Mathematica Second ed..* Addison-Wesley.

A design of a parser in computer algebra systems

Shigekazu NAKAGAWA

Department of Computer Science and Mathematics,
Kurashiki University of Science and the Arts,
Nishinoura, Tsurajima-cho, Kurashiki, Okayama, 712–8505, Japan
(Received September 30, 1998)

Most computer algebra systems consist of a parser and an evaluator. The former is a part of input routines and it interprets input streams (external representations) into appropriate internal representations. The latter is a main part of calculation. At first, we discuss some internal representations such as recursive representation polynomials and distributed representation polynomials. Secondly, we propose here a fundamental concept of a parser which takes advantage of dealing with distributed representation polynomials.