

数式処理システム Risa/Asir のインプリメンテーション

野呂 正行 竹島 卓

(株) 富士通研究所 国際情報社会科学研究所

Risa/Asir は、他のプログラムに組み込み可能なサブルーチンライブラリと、対話的使用のためのアプリケーションプログラム Asir からなる数式処理システムである。ライブラリは、基本演算部 (エンジン)、メモリ管理部、パーザ、インタプリタをもち、様々なレベルからの使用が可能である。四則演算、因数分解等を実行する基本演算部は、他のシステムと同等またはそれ以上の性能を持つ。Asir は、C 言語風のユーザ言語、dbx 風のユーザ言語デバッガを持つ。本稿では、Risa/Asir の構成、データ構造、使用アルゴリズムなどについて述べ、タイミングデータおよび他のシステムとの比較を示す。

Implementation of a Computer Algebra System Risa/Asir

Masayuki Noro and Taku Takeshima

IIAS-SIS, FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-shi, Shizuoka, 410-03 JAPAN

(e-mail:noro@iias.flab.fujitsu.co.jp)

A computer algebra system Risa/Asir consists of an application program Asir for interactive use and several subroutine libraries which can be used as parts of other programs. The libraries include basic arithmetic subroutines (engine), storage manager, parser and interpreter, which can be used from various levels. The engine executes arithmetic on bignum's and polynomials, factorization, etc. Its performance is sufficiently high compared with other systems. Asir has a C-like user language and a dbx-like debugger for the user language. This paper describes the structure of Risa/Asir, the data structures and the algorithms used in the system. We also show some timing data and comparisons with other systems.

1 Risa システムの概要

Risa [9] は、富士通国際研で開発中の数式処理システム全体の名称である。その実体は、数式処理を行なうサブルーチンを核とするライブラリであり、大部分が C 言語で記述されている。また、Asir は、Risa の各ライブラリをひとまとめにして一つのアプリケーションプログラムとしたもので、UNIX、Macintosh、MS-DOS 上で動作する。ここでは、UNIX 版における Risa ライブラリの構成について述べる。

Risa ライブラリは、機能別に次のように分割されている。

- 基本演算部 (エンジン)
- メモリ管理部
- パーザおよびインタプリタ、組み込み関数インタフェース
- プロセス間通信インタフェース

エンジン (libca.a) およびメモリ管理部 (libgc.a) は、Risa の核である。エンジンは、既に内部形式に変換されたオブジェクトに対する演算を行なう。エンジンの主な機能は、任意多倍長数 (bignum)、多項式、有理式、ベクトル、行列などの四則演算、GCD、因数分解、終結式、行列式などである。また、メモリ管理部は、エンジンに限らず、システム全体のメモリ管理を行なう。この部分は、Boehm-Weiser [3] による conservative GC を用いている。

Risa を他のプログラムのサブルーチンとして用いる場合、エンジンおよびメモリ管理部を結合することにより最低限の機能を果たすことができるが、その場合、そのプログラムにおける数式の表現を Risa の内部表現に変換する操作を、呼び出しプログラム側で行なう必要がある。特に、入力が、 $(x+y)^2$ などの文字列による場合、数式のパーザが必要となる。このようなパーザは、libparse.a に用意してある。また、対話的な環境 (Asir など) におけるユーザおよびプログラムファイルからの入力に対するパーザ、インタプリタもこのライブラリに入っている。

Risa では、内部形式を他のプロセスとやりとりするための基本機能をプロセス間通信インタフェース (libtcp.a) により提供している。

Asir は一つのコマンドとして存在する。使用感、ユーザインタフェースは、UNIX の bc に近い。特に、bc と同様ユーザ言語 (これも Asir と称する) は C 言語に近い文法を持つ。

現状では、Asir の組み込み関数は、REDUCE、Macsyma、Mathematica などに比較して多くはない。特に、グラフィックス、積分、微分方程式、パターンマッチング機能などはほとんど実装されていない。しかし、基本的な四則演算、因数分解などに関しては、他のシステムと同等あるいはそれ以上の性能を持つ。また、Asir で書かれたものとして、Gröbner 基底計算パッケージがある。これは、ごく最近の改良までを採り入れているため、少なくとも現在一般に入手可能などのシステムよりも高速であると言える。

Risa ライブラリに欠落している部分として、任意精度浮動小数演算 (bigfloat) がある。Asir では、この部分を、やはりサブルーチンライブラリとして配布されている PARI ライブラリ [1] を結合することによって補っている。PARI は、主に数の演算に関するサブルーチンの集合体で、bigfloat も非常に高速に実行し、また、超越函数の評価なども bigfloat で高速に行なうことができる。これを結合することにより、Asir にも自動的にこれらの函数の評価機能が加わることになる。

Asir のもう一つの特徴として、ユーザ言語 Asir で書かれたプログラムをデバッグするためのデバッガが組み込まれていることが挙げられる。このデバッガは、Asir が C 言語に近いということを考慮して、UNIX の標準的なデバッガ dbx 風のものとした。

2 Risa エンジン

エンジンで扱われる主なオブジェクトは以下の通りである。

- | | | |
|--------|---------|------------|
| 1. 数 | 2. 多項式 | 3. 有理式 |
| 4. リスト | 5. ベクトル | 6. 行列 |
| 7. 文字列 | 8. 構造体 | 9. 分散表現多項式 |

インデックスは、各型の識別子を表す。各型の先頭 1 ワードは、

```
typedef struct oObj { short id, pad; } *Obj;
```

なる共通の構造を持つ。メンバ pad は、識別子 id に応じて幾つかに分割されて、固有の意味を持つ。特に、数の場合、pad 中に更に数の識別子を持ち、数における階層構造を表現する。数は、更に、有理数、倍精度浮動小数、bigfloat、複素数、代数的数に分けられる。型情報に 1 ワードを用いるのは過分であるが、RISC CPU の場合、構造体のメンバのアラインメントが、最も制限の厳しいメンバによって決定される場合が多いため、このような扱いとなっている。

各型に対して四則、冪などの演算が定義されているが、原則として、各関数は、自分の識別子以下のオブジェクトを、その演算が意味を持つ場合には受け付ける。例えば、多項式と数の加算は、多項式に対する演算 addp() を呼び出すことによりなされる。この原則は、型の異なる数どうしにも適用される。

エンジンに対する新しい型の追加は、その型の識別子を決め、四則、冪、比較演算を記述することにより行なう。これは、数に対しても同様であり、実際に区間数の追加 [7] が試みられている。

2.1 有理数

エンジンの中で、最も重要な役割を果たすのは、有理数係数多項式である。よって、その係数である有理数の表現は、システム全体の性能に大きな影響を及ぼす。Risa における有理数は次のように定義されている。

```
typedef struct oQ { short id; char nid, sgn;
                  N nm, dn; } *Q;
typedef struct oN { int p; int b[1]; } *N;
```

oQ が有理数、oN は自然数を表す。すなわち、自然数は、分母分子を表す自然数と、符号からなる。自然数 oN は実際には不定長の配列であり、メンバ p が、実際の配列の長さを表す。自然数は、適当な整数 N により 2^N 進表現される。この N の値はシステムの機械依存性に影響を与える。すなわち、 $N \leq 16$ ならば、多倍長乗除算に現れる整数乗除算が 32 ビット整数の範囲内で収まるため、全て C で記述できる。 N がこれより大きい場合、アセンブリ言語による記述が必要となる。さらに、 $N = 32$ の場合加減算もアセンブリ言語なしでは円滑に記述できない。このため、通常は $N = 31$ などを用いるが、現在の版では半端な $N = 27$ を用いている。これは、Sparc 版の倍長除算ルーチンの簡略化のため、

暫定的にこうせざるを得なかったためであり、今後可能ならば改良する予定である。(Sparc 以外では、 $N = 31$ で動作可能である。)

有理数、自然数の表現方法としては、リスト表現による方法、短整数を即値表現する方法などが考えられる。これらについては比較は行なっていない。特に後者については、メモリ効率、計算効率共に向上することが期待されるが、これに基づいて書換えを行なうことは、エンジン全体に波及するため残念ながら変更の予定はない。

整数の演算において、除算の効率はシステム全体の性能を左右する。Risa では Knuth [6] に記載のアルゴリズムを採用している。ただし、その最小構成要素である倍長整数を単整数で割る命令を CPU が提供しているか否かで、除算の効率は大きく影響される。例として、同一クロックで、総合性能もほぼ同一とみなされる、Sparc と MC68040 において、整数除算のみを比較すると、2 倍程度 MC68040 が高速である。特に、Sparc においては乗算もソフトウェアにより実行されるので、整数演算の効率は著しく劣っている。倍長除算命令が省かれるのは、最近の RISC プロセッサに共通の傾向であるが、数式処理にとっては不利な環境である。

2.2 多項式

多項式は、ある主変数について降冪の順に整理し、各項の指数、係数をリストにした再帰表現を用いている。

```
typedef struct oP { short id, pad; V v;
                  struct oDCP *dc; } *P;
typedef struct oDCP { Q d; P c;
                   struct oDCP *next; } *DCP;
```

多項式も、配列により表現することが可能である。配列表現の長所は、表現のコンパクトさと、各項のアクセスのしやすさであるが、加算などで変化しない末尾項の扱いと、ポインタを含む不定長の領域を頻繁に用いることの GC への影響を考慮して、上記のようなリストによる表現とした。

多項式の四則演算に関しては特に特殊なアルゴリズムは用いていないが、冪乗の計算は、二項展開を再帰的に用いることにより高速化を図っている。

2.3 分散表現多項式

因数分解や多項式の変数への代入を行なう場合は、多項式を再帰表現するほうが扱いやすいが、Gröbner 基底を計算する場合、多項式を単項式の和として表現するのが便利である。これを分散表現と呼ぶ。この表現は、エンジンの他の部分から呼び出されることはなく、Gröbner 基底計算パッケージと直接に結びついている。しかし、その四則演算効率、数式処理において重要な位置を占める Gröbner 基底の計算に大きく影響するため、基本オブジェクトとして採用した。

通常の整数における乗算は高速アルゴリズムを使用していないが、Gröbner 基底計算の中間結果の係数はしばしば十進数千桁にも達するため、簡単な高速アルゴリズム (Karatsuba 法 [5]) により整数乗算を高速化している。

Gröbner 基底を計算するアルゴリズムは一般に Buchberger アルゴリズムと呼ばれ、その基本は極めて単純かつ明解であるが、実用的なプログラムにするためには、無駄な計算を省く criterion と、計算順序を制御する strategy をよいものにすることが不可欠となる。これらについては数多くの研究があり、到底ここで紹介はできないが、これらの成果を利用して、Asir 上に実現された Gröbner 基底計算パッケージ [10] は十分実用的なレベルまで高速化されている。

2.4 因数分解、GCD、無平方分解

四則演算以外でエンジンの最も大きな部分を占めるのは、多項式の因数分解および GCD に関する部分である。一変数多項式の因数分解は、古典的な Berlekamp-Hensel アルゴリズムを採用しているが、随所に高速化のための工夫を入れることにより、REDUCE、Maple などと比較して十分に高速であるといえる。多変数の因数分解、GCD は、一変数に落して因子のタネを作り、EZ 法 [8] により Hensel 構成することにより行なっている。これは、順次変数の個数を上げていく EEZ 法 [11] より一般に効率が劣るとされているが、これも細かな工夫により満足すべき程度まで高速化している。また、無平方分解は、古典的な方法ではなく、Wang-Trager [12] による直接的な方法を用いている。この方法は、重複度が高い因子を持つ場合に特に有効である。

他のシステムとの比較の実際は、タイミングデータ

の節で示す。

3 メモリ管理

Risa/Asir で用いているメモリ管理部 libgc.a は、Boehm-Weiser による conservative GC を、かれら自身がインプリメントしてフリーソフトウェアとして配布しているものを用いている。この GC の長所は、

- free() いらすの malloc() として使えること
- ポインタを含まない領域 (atomic) と含む領域 (composite) が別々に管理されていること

などがある。前者により、ユーザは、自動回収機構付きの malloc() を呼び出していると考えることができ、また後者により、明らかにポインタを含まない多倍長整数用の領域を、atomic な領域として確保することにより無駄な GC を避けることができる。問題は全体の実行時間に占める GC 時間の割合であるが、典型的な多項式演算の例題である因数分解を実行させた場合、十分な量のメモリがある場合には、GC 時間は十数パーセント程度で、許容できる範囲にあると考えられる。

この GC における最大の問題点は、関数内でローカルな不定長作業領域を確保する場合に生ずる。通常このような場合、alloca() を用いるが、コンパイラによってはこれを通常の malloc() を用いたライブラリ関数として実現する場合がある。このような場合、この領域に置かれたポインタは GC から見えなくなるので、不当に GC されてしまう。alloca() の代わりにこの GC の領域割り当て関数である gc_malloc() を用いればこの問題は避けることができるが、コンパイラによっては、最適化のためこの作業領域の先頭ポインタが保存されないことがある。この場合もやはり作業領域上のポインタが GC から見えなくなる。

このように、スタックを用いる alloca() が与えられず、かつコンパイラが領域の先頭ポインタを捨ててしまうことがあり得る場合には、この GC を用いることは危険である。

現在、この問題を避けるため、alloca() が不備な場合、gcc が使用可能ならばそれを用いている。

また、もう一つの問題点として、この GC がコンパクションを行なわないことが挙げられるが、これについては未だ解決案は示されていない。

4 Asir

Asir は、所謂通常の数式処理システムの形態をなす。すなわち、ユーザとの対話およびプログラムの実行およびデバッグを行なうことができる。

Asir の主要部は、パーザ、インタプリタおよびデバッグである。

4.1 パーザ

パーザは、入力から文字列を読み込んで、中間言語に変換する。Asir のユーザ言語は C 言語風で、中間言語も C と同様に 文、式の二つのレベルを持つ。数式処理システムのユーザ言語として、他のシステムと最も異なる点は、数学でいう不定元と、プログラムにおける変数を厳密に区別している点である。すなわち、不定元は値を持つことはできない。不定元がある値を持つ場合の数式の値は、明示的に代入関数を呼び出すことにより得られる。これは、不定元のつもりで使用した変数が値を持っている場合、自動的に代入されてしまうといった既存のシステムにありがちな混乱を避けるためである。Asir においては、不定元は小文字で、プログラム変数は大文字で始まる。

さらに、ファイルからプログラムを読み込む場合、C のプリプロセッサ `cpp` を通してから読み込む。これにより、`#define` によるマクロ定義、`#include` によるヘッダファイルの読み込みなどが、C と同様に可能となっている。

通常、構造を持つデータはリストにより表現可能であるが、Asir では、C における構造体を制限したデータ型をサポートしている。この型を用いることは、次のような利点をもたらす。

- メンバを名前でアクセスできることにより、プログラムの書きやすさ、可読性が向上する。
- リスト表現に比較して、メンバのアクセス速度が向上する。
- デバッグ時に、特定のメンバのみを表示できる。数式処理のプログラムにおいては、中間結果が膨大な式になる場合が多いので、特定の部分だけを表示することができることは重要である。

また、特定の構造体に四則演算の定義を結合することにより、構造体に対する四則演算の多重定義を行なうこと

を現在テスト中である。例えば、特定の法によるモジュラ演算を、ある構造体に対する四則演算の一部とすることにより、法を大域変数とせずに隠蔽することができる。これは、AXIOM などで行なわれているほど対象となるドメインを明確にすることにはならないが、たとえば初期の Macsyma のように、法が冪指数にまで作用するといった障害は避けられる。

4.2 インタプリタ

インタプリタは、パーザにより中間言語に変換された入力を、逐次実行する。中間言語の構造に合わせて、文のインタプリタ `evalstat()` と式のインタプリタ `eval()` を持つ。変換された入力は、文のリストである。主な文の種類は次の通りである。

1. 単文
2. 複文
3. if 文
4. for 文
5. return 文
6. break 文
7. ブレークポイント文

このように、現在の所、C 言語の構成に合わせたものとなっているが、これら自体は言語によらず共通して含まれる成分であると考えられる。よって、言語に応じて文の種類を追加することにより、例えば Maple、REDUCE 風のパーザに対応することも可能である。

4.3 デバッグ

デバッグは、前述のインタプリタに組み込まれた形で動作する。文のインタプリタは、各文の実行前に、ステップ実行フラグをチェックし、もしそれらがオンの場合には自動的にデバッグモードに入る。また、前述のブレークポイント文を実行した場合にもデバッグモードに入る。ステップ実行フラグは、デバッグモードでステップ実行を指示した場合にオンになる。またブレークポイント文は、デバッグモードで、特定の行または関数に対してブレークポイントをセットした場合に、対応する部分に挿入される。また、インタプリタまたはエンジン内で、不正な引数などによってエラー処理ルーチンが呼ばれた場合、および、キーボード割り込みからデバッグが指示された場合にもデバッグモードに入る。

デバッグモードにおけるコマンドは次の通りである。機能、用法は通常の `dbx` と同様である。

- `step`

- next
- print *exprlist*
- stop { at *sourceline* | in *function* } [if *cond*]
- where

デバッグモードにおけるコマンドは、UNIX のデバッガ dbx のコマンドのうち頻繁に使用するごく少数のもののみを採用してあるが、実用上十分であると考えている。現在入手可能な数式処理システムのほとんどがトレーサしか持たないが、トレーサのみでは、実際にユーザ言語でプログラムを書くのは大変困難であると考えている。なぜなら、数式処理におけるプログラムはかなり深いリカーションを生ずるものが多く、また中間結果もしばしば膨大なものとなるため、トレーサの出すメッセージからプログラムの動作状態を判断するのが困難だからである。

デバッガを組み込むことによるオーバーヘッドは、文の実行毎のステップ実行フラグのチェックによって生ずる。しかし、チェックが式ではなく、文の単位で行なわれ、かつ、1 ワードのメモリの読み出しのため、そのオーバーヘッドは無視できる程度である。

4.4 組み込み関数

Asir では、C 言語と同様、四則、冪などの基本演算以外の演算は関数呼び出しにより行なわれる。この際、C における標準ライブラリ関数に相当するのが組み込み関数である。組み込み関数には、最終的にエンジンあるいは PARI の関数を呼び出すもの、あるいは、インタプリタまたは組み込み関数インタフェース自体がデータの操作、演算を行なうものが存在する。前者に属するものとしては、

多項式の因数分解、無平方分解、GCD、終結式、超越関数の値の評価

などがあり、後者では、

内部表現の可読形式による表示、微分、代入、リスト、行列、ベクトルの操作、ファイル I/O

などがある。微分、代入など、一見エンジンに含まれると考えられるものがインタプリタで処理されるのは、中間の演算および結果が、入力型の範囲内に必ずしも収まらない、あるいは、エンジンにおける変数の扱いを越える操作が必要になる場合が生ずるからである。また、

表示も、例えば、関数を含む有理式など、エンジンで扱うには不適当なものがあるため、エンジンからは外してある。

現在実装されている組み込み関数は必要最小限であるため、必要な機能はユーザ言語により記述するか、または、C などにより記述、コンパイルしたオブジェクトファイルを実行時にロードする。後者は、ダイナミックローディング機能をもつコンパイラが使える場合に限られ、例えば、Macintosh、MS-DOS 版ではまだ不可能である。UNIX 版では、この機能は、KCL を参考にして実現されている。すなわち、外部シンボルの参照を解消したオブジェクトファイルをメモリ上に載せ、そのファイルの先頭に置かれた関数登録用のサブルーチンを呼び出すことにより、そのファイル中の関数を組み込み関数として使えるようにする。

4.5 グラフィックス

グラフィックスは、UNIX 版では別プロセスにより X11 のクライアントとして実現され、Asir は、単なる入力インタフェースとして働く。これは、グラフィックスのイベントループと大きな計算に入った時の Asir とを両立させるのが困難なためである。このため、別プロセスへの分離ができない Macintosh 版などでは、入力回りに不自然な変更を加える必要があった。

現在実際に実現されている機能は少なく、中間値の定理を用いた二変数多項式の零点 (陰関数) および等高線の表示のみであるが、マウスによる拡大縮小や、sturm 列を用いてより正確に零点を表示するなどの機能を備えている。

ここで用いられているプロセス間通信は、TCP 上の XDR を用いた Risa の内部表現の通信で、マシンのバイトオーダによらない通信が可能である。

5 移植

Risa/Asir は大部分が C 言語で記述され、ごく一部がアセンブリ言語で記述されている。アセンブリ言語で記述されている部分は当然機種/CPU に依存するがそれ以外にも、OS などに依存する部分はいくつかある。

Risa/Asir は、現在の所、

- UNIX (Mach)

Sun3, Sun4, NEWS(CISC, RISC), VAX, Apollo, DECStation, NeXT, RISC LUNA

- MacOS

Macintosh

- DOS Extender/MS-DOS

IBM-PC Clone, PC98, FM (R, TOWNS)

の三種類の OS 上で、それぞれに挙げたマシン上で動作している。機種/OS/CPU に依存するのは、主に

- メモリ管理部
- ダイナミックオブジェクトローダ
- グラフィックス
- 倍長整数乗除算

である。特に、メモリ管理部は、プロセスのメモリ空間の形状、スタックの上限、ヒープの下限、アラインメント、レジスタの種類、割り当てなどを正しく把握する必要がある。またダイナミックローダに関しては、当該 OS 上で提供されているダイナミックローディングの方法を調べ、最善の方法を探る必要がある。また、OS によっては、UNIX におけるシグナルのような割り込みが使用できないものもある。実際の使用においては、キーボードからの中断ができることが必須のため、MacOS、MS-DOS においては、中程度の頻度で呼び出される領域割り当てルーチンが、キーボード割り込みをチェックすることにより、リアルタイム割り込みを疑似的に実現している。

現在、グラフィックス機能は、UNIX 版では X11、Mac 版では MacOS そのものにより実現されている。X11、MacOS いずれもグラフィックス機能そのものは十分なものを持っているが、その使用に関してはほとんど互換性がないため、ほとんどの部分を個別に書く必要がある。この事情は、MS-DOS 上でグラフィックス機能を実現する場合でも同様であると考えられる。

倍長整数乗除算は CPU に依存する。これは、bignum ルーチンの最も基本となるものであり、

$$a \times b + c \Rightarrow q \times d + r \text{ (商、剰余の計算)}$$

のタイプのサブルーチンを 5 個書くことにより得られる。

6 Timings

SPECmark のように共通に使われるベンチマークは、数式処理システムに対してはまだ存在しない。実際、各機能をくまなく評価できるようなデータを作ることは極めて困難であり、以下に示すタイミングデータは、あくまで一つの側面を示すものである。

使用計算機は、SparcStation2 (40MHz Sparc、64MB) で、Asir、REDUCE3.4 のデータは GC 時間を除いた実行時間を使用し、Maple V R2、Mathematica2.0 (Mma) においては、報告される時間に GC が含まれるか否か不明のため、報告された時間そのものを使用した。単位は秒。横線は、メモリ不足または 10 分以上待っても計算が終了しなかったものを示す。

- 多項式の冪乗

$(a + b + c + d + e)^n$ ($n = 20, 30, 40, 50$) の展開。

n	20	30	40	50
Asir	1.34	6.77	21.1	52.1
REDUCE	8.67	61.7	—	—
Maple	4.23	28.8	—	—
Mma	76.1	528	—	—

- 因数分解

REDUCE3.4 付属の “factor.tst” は、19 の因数分解の問題からなり、1 から 15 は Wang [11] による多変数多項式、16 から 19 は一変数多項式である。

number	1	2	3	4	5
Asir	0.11	0.08	0.1	0.88	0.72
REDUCE	0.27	0.16	0.2	0.54	0.91
Maple	1.07	0.73	1.53	4.7	5.1
Mma	0.12	0.15	0.17	1.78	6.53

6	7	8	9	10	11	12
1.67	0.06	7.36	3.37	1.33	3.51	0.01
1.22	0.31	23.8	2.27	2.01	3.02	0.07
5.4	0.57	32.3	16.3	5.92	8.42	0.07
3	0.27	150.6	66.42	3.68	37.4	0.08

13	14	15	16	17	18	19
0.25	0.24	0.24	1.9	0.23	0.21	0.32
0.24	0.76	0.47	5.08	1.11	1.82	1.97
0.95	2.12	2.42	2.65	1.27	1.07	2.28
0.23	0.57	1.12	12.0	0.75	0.62	0.93

[12] の、重複度の高い多項式の因数分解。

prombem	1	2	3	4
Asir	0.021	0.166	0.092	0.449
REDUCE	0.1	0.37	0.47	3.95
Maple	0.1	0.7	1.76	11.6
Mma	0.017	0.112	0.3	1.8

5	6	7	8	9	10
1.57	2.35	0.097	0.28	0.328	0.925
23.6	48.7	0.96	2.4	3.09	12.2
108	179	0.98	1.51	3.15	9.1
7.03	11.2	0.533	1.2	1.52	4.12

• Gröbner 基底の計算

Gröbner 基底計算プログラムに対してテスト用としてよく用いられる問題を幾つか選んだ。表で、Kn は Katsura-n [2]、G は Gerdt [2]、Cn は Cyclic(n) [4] で末尾の r は revgradlex (全次数+逆辞書式順序)、l は lex (辞書式順序)を示す。Mma は lex のみサポートのため、K3r、K4r は計算できなかった。

problem	K3r	K3l	K4r	K4l	G1	C5l
Asir	0.4	0.63	2.16	7.5	6.62	13.17
REDUCE	0.53	1.56	5.57	150	7.44	—
Maple	3.33	6.12	13.6	653	57.1	—
Mma	—	9.5	—	—	17.4	—

7 まとめ

高速、組み込み可能、デバッグ装備、高い移植性などを目標として Risa/Asir を開発してきた。それらのいくつかは満足すべき状態にあるが、全体的に未だ不備な点は多くある。特に、使いやすさを向上させることが当面の最大の目標である。機能の充実に関しては、特に要望の多い simplifierなどを重点的に実現していく必要はあるが、他の汎用システムと同程度まで単独で機能を揃える必要があるとは考えていない。それより、PARIのようにオープンな他のシステムと結合して機能を充実させるのが賢明であると考えている。

参考文献

- [1] Batut, C., Bernardi, D., Cohen, H., Olivier, M., User's Guide to PARI-GP. February 1991.

- [2] Boege, W., Gebauer, R., Kredel, H., Some Examples for Solving Systems of Algebraic Equations by Calculating Groebner Bases. J. Symb. Comp., 1(1986), 83-98.
- [3] Boehm, H., Weiser, M., Garbage Collection in an Uncooperative Environment. Software Practice & Experience(September 1988), 807-820.
- [4] Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C., "One sugar cube, please" Or Selection strategies in the Buchberger algorithm. Proc. ISSAC '91, ACM Press, New York(1991), 49-54.
- [5] Karatsuba, A., Multiplication of Multidigit Numbers on Automata. Soviet Physics - Doklady 7(1963), 595-596.
- [6] Knuth, D.E., The Art of Computer Programming, Vol. 2: Seminumerical Algorithms (second edition). Addison-Wesley(1981).
- [7] 近藤 祐史, 野田 松太郎, 数式処理と区間演算の結合とその応用. 研究集会 数式処理における理論と応用の研究 (1992).
- [8] Moses, J., Yun, D.Y.Y., The EZGCD Algorithm. Proc. ACM Annual Conf. (1973), 159-166.
- [9] Noro, M., Takeshima, T., Risa/Asir - A Computer Algebra System. Proc. ISSAC '92, ACM Press, New York(1992), 387-396.
- [10] Shimoyama, T., Noro, M., Takeshima, T., — How we implemented the Buchberger algorithm — Buchberger algorithm on the Risa/Asir System. Submitted to DISCO'93.
- [11] Wang, P.S., An Improved Multivariate Polynomial Factoring Algorithm. Math. Comp. 32(1978), 1215-1231.
- [12] Wang, P.S., Trager, B.M., New Algorithms for Polynomial Square-Free Decomposition over the Integers. SIAM J. Comp. 8(1979), 300-305.