数理解析研究所講究録
第 811 巻 1992 年 165-170

数式処理システム risa/asir の内部構造

富士通研究所　国際研

野呂　正行　　竹島　卓

概要

risa は富士通国際研で開発中の数式処理システムである。risa は、いくつかの UNIX のサブルーチンライブラリおよび、それらを通常の数式処理システムの形にまとめた asir からなる。risa は、他のプログラムに組み込み可能である。すなわち、ライブラリを、他のプログラムにリンクして用いることができる。本稿では、このライブラリの機能について述べる。また、プロセス間通信による分散計算を実現するための機能についても述べる。

エンジン

risa において、内部形式に変換されたオブジェクトを受けとって、さまざまな演算を行なうサブルーチンからなる部分をエンジンと呼ぶ。エンジンは、asir において所謂カーネルとして用いられているが、エンジンの機能は、パーザ、評価器と独立に論ずることができる。エンジンで扱われるオブジェクトには、

1. 数 (有理数、浮動小数)
2. 多項式 (係数は、有理数、浮動小数いずれも可)
3. 有理式 (必ずしも約分はされていない)
4. リスト
5. ベクトル (一次元配列)
6. 行列 (二次元配列)
7. 文字列
8. 構造体 (現在試作中)

がある。これらはそれぞれ構造体として定義されているが、その先頭部分は共通して、

```
struct oObj {
    short id;      /* 識別子 */
    short pad;     /* オブジェクトによっては、いくつかのフィールドに分かれる */
};
```

なる構造をもち、id フィールドにより識別される。これらのオブジェクトに対し、それぞれ加減乗除、冪、比較を行なう函数が用意されている。例えば、多項式の加法は、

```
#include "ca.h"

addp(vl,a,b,rp)       /* *rp = a + b, vl は変数順序リスト */
VL vl;
P a,b,*rp;
```

なる函数により実現されている。実際に演算を行なう場合、一般に型が異なる可能性のあるオブジェクト同士の演算が必要になる。このような場合に不必要な識別子による場合分けを避けるために次のような規約を設けている。

- 各オブジェクトに対する基本演算函数は、対応する識別子 (これは前記リストの番号) 以下のオブジェクトを正しく扱う。すなわち、場合分けを行ない、適切な函数を呼び出す。

これにより、例えば多項式あるいは数のみが現れるプログラムにおいては、多項式に対応する演算函数を呼び出すことにより全て正しく行なわれる。注意すべきことは、有理式に対応する演算函数を用いる場合で、これらは原則として、約分は行なわない。よって、多項式のみが現れるプログラムで、有理式に対応する演算を用いることは望ましくない。インタプリタにおいては、基本演算のトップレベルとして、add()、sub() などが用意されているが、ここでは、識別子の最大値をとり、それに対応する函数を呼び出すという方法をとっている。

上で述べた演算以外の演算も、エンジンに含まれている。主なものは、

- 因数分解 (整数上一変数、多変数、代数体上一変数)
- GCD、無平方分解
- 特殊な除算 (商、剰余、擬剰余など)
- 種々の型変換 (数の radix 変換、有理数-浮動小数変換など)

などである。これらは、一般的な型に対して定義されているわけではなく、特定の型専用となっている。呼び出し方もそれぞれの函数により異なる。

メモリ管理

risa では、メモリ管理機構として、[Boehm,Weiser] によるガーベジコレクタを採用している。これは、プログラム上では、malloc() のかわりに、gc_malloc() を用いることにより、free() を使わずに自動的

にガーベジコレクションが行なわれるいうものである。この方式においては、自ら管理するヒープ領域の範囲外の部分からのマーキングは行なわないが、

- malloc() で得た領域に、gc_malloc() で得た領域のポインタを保存しない。

という規則を守ることにより、通常の malloc() と共存できる。

内部形式への変換

エンジンの各函数は、内部形式に変換されたオブジェクトを対象とするものであった。通常の形式の数式を内部形式に変換するには何らかのパーザを用いる必要がある。asir では、入力された数式、プログラムは、まずパーザにより木に変換され、それが評価器に渡され、内部形式に変換される。asir では、文 (プログラム) をパーズするパーザの他に、デバッガ用に、式のみをパーズするパーザを持っている。これは、文字列をパーズして、評価器の入力となる木を生成するもので、これをそのまま組み込み用として用いることができる。

```
char *debugstrp,*string_to_parse;
NODE debugexprlist;
Obj obj;

debugstrp = string_to_parse;
debugparse();
obj = (Obj)eval(BDY(debugexprlist));
```

により、文字列 string_to_parse で表される数式が、debugexprlistに木のリストとしてパーズされ、eval() により内部形式に変換される。

初期化

以上の各函数を使うためには、いくつかの初期化が必要である。

gc_init()
: メモリ管理機構の初期化

nglob_init()
: 数定数の初期化

glob_init()
: CO などの大域変数の初期化。

その他、パーザ、出力函数を使う場合には、

`output_init()`
: 出力函数関係の初期化。

`arf_init()`
: パーザのトップレベルの四則演算函数の初期化。

内部形式は、ある変数順序に従って構成される。その時点における変数順序は、大域変数 CO で表される。引数として型 VL の値を引数として要求する函数の引数として CO を用いる。glob_init() では、変数順序として、alphabet 小文字 1 文字の変数をあらかじめ変数として登録し、その変数順序を適当に定めるが、それ以外の変数 (変数は、asir のパーザでは、小文字で始まる文字列) に対しては、それが現れる毎に、変数順序リストに登録する必要がある。通常これは、字句解析部で自動的に現在のリストの末尾に追加されるが、それ以外の操作を必要とする場合には、適宜必要な操作を行なう必要がある。

プロセス間通信による分散計算

asir 自身、あるいは risa ライブラリを組み込んだプログラム間でプロセス間通信によりデータを通信しながら分散計算を行なうことも可能である。それは、最も単純には、asir の標準入出力と、通常の形式の数式をやりとりすることで可能となる。しかし、この方法では無駄な変換が数多く起こることと、大規模なデータのやりとりには不十分であることから、risa ライブラリでは、TCP/IP ストリームによる、内部形式の送受信プリミティブを準備した。このための簡単なプロトコルも用意した。XDR を用いることにより、バイトオーダの異なる機種の間でも正しく通信が行なわれる。

現在、asir から、他のコマンドを起動して、そのプロセスと通信をするためのプリミティブが、asir に用意されている。asir においては、`tcpinit("`*host*`","`*command*`")` が、*host* 上で、*command* を起動し、そのプロセスとの通信路を確保するコマンドである。

`tcpinit()` では、

```
s = connect_to_client(rname,epath);
iofp[s].in = fdopen(s,"r"); iofp[s].out = fdopen(s,"w");
xdrstdio_create(&iofp[s].xdro,iofp[s].out,XDR_ENCODE);
xdrstdio_create(&iofp[s].xdri,iofp[s].in,XDR_DECODE);
gensend(&iofp[s].xdro,C_VL,CO); fflush(iofp[s].out);
```

といった操作が行なわれる。クライアント側では、対応して、

```
s = connect_to_server(atoi(argv[2]),argv[1]);
```

```
infp = fdopen(s,"r"); outfp = fdopen(s,"w");
xdrstdio_create(&xdro,outfp,XDR_ENCODE);
xdrstdio_create(&xdri,infp,XDR_DECODE);
gc_init(); nglob_init();
```

という操作が行なわれ、通信が始まる。双方とも、オブジェクト送信用函数、受信用函数はそれぞれ、gensend()、genrecv() で、予約識別子として、

C_ZERO	通常 end of file
C_OBJ	オブジェクト送受信
C_VL	変数リスト送受信

が定められる。それ以外に必要な識別子 (固有のコマンドなど) は、サーバ、クライアント双方の了解のもとに定め、使用する。

実例

ごく簡単な例として、標準入力から式を入力してそれを評価し、標準出力に出力するプログラムを示す。

```
/* main.c */

#include "ca.h"

extern char *debugstrp;
extern NODE debugexprlist;

main() {
    char buf[BUFSIZ];
    Obj obj;

    gc_init(); glob_init(); nglob_init();
    output_init(); arf_init();
    while ( 1 ) {
        gets(buf); debugstrp = buf;
        debugparse(); obj = (Obj)eval(BDY(debugexprlist));
        printexpr(0,obj); putchar('\n');
    }
}
```

これを、

```
% cc -Irisa/include main.c risa/parse/libparse.a risa/gc/libgc.a \
    risa/engine/libca.a risa/asm/libasm.a -lm -o test
```

('risa' は risa のソース、オブジェクトのあるディレクトリ) とすることにより、'test' ができる。

```
% test
2^100
1267650600228229401496703205376
(x+y)^5
x^5+5*y*x^4+10*y^2*x^3+10*y^3*x^2+5*y^4*x+y^5
^C%
```

終りに

以上、risa の、他のプログラムへの組み込みを実現するための機能、内部構造の概要について述べた。詳細については、[野呂] を参照されたい。今後、risa ライブラリの機能拡張としては、bigfloat 、既存の数式処理言語をエミュレートするパーザの実現を考えている。現在、サンプルとして、Maple のユーザ言語をパーズして risa/asir の内部形式に変換するパーザを試作中である。これにより、asir パーザの各部分を部品として使うために、どのように切り分けを行なうべきかがより明確になると考えている。

文献

[Boehm,Weiser]
"Garbage Collection in an Uncooperative Environment", Software Practice & Experience, September 1988, pp. 807-820.

[野呂] "risa 内部仕様 Version 0", 近日公開予定