実験的仕様の関数

Risa/Asir 実験的仕様関数説明書
1.0 版
2010 年 9 月

by Risa/Asir committers

1 実験的仕様の関数説明書について

この説明書では **Asir** に導入された実験的仕様の関数について説明する．正式な関数として導入されたものの記述は Risa/Asir マニュアルに移動される． ChangeLog の項目は www.openxm.org の cvsweb でソースコードを読む時の助けになる情報が書かれている．

2 実験的仕様の関数

2.1 クオート

2.1.1 quotetotex, quotetotex_env

`quotetotex(`*q*`)`
:: *q* を latex 形式で表現した文字列に変換する.

`quotetotex_env(`*key*`,`*value*`)`
:: quotetotex の動作を制御するパラメータを変更する.

`quotetotex_env()`
:: quotetotex の動作を制御するパラメータの現在値を戻す.

`quotetotex_env(0)`
:: quotetotex の動作を制御するパラメータをデフォールト値に戻す.

return 文字列 (quotetotex) または リストまたはオブジェクト (quotetotex_env)

q quote

key 文字列

value オブジェクト

- quotetotex は *q* を latex 形式で表現した文字列に変換する.
- 以下 quotetotex_env のパラメータの意味を説明する.
- conv_rule: 3 ビットを用いて変換ルールを指定する. 0 ビット目は symbol_table による変換を行うか, 1 ビット目は添字変換を行うか, 2 ビット目は d から始まる変数名を微分作用素とみなして処理するか, を意味する. たとえば conv_rule として 3 を指定すると, 0 ビット目, 1 ビット目が 1 となるので symbol_table による変換を行い, 添字変換をおこなう. 添字変換は数字と英字の境目および _ 記号を区切りとする. symbol_table による変換が最初に適用される. alpha, beta, 等は自動的をギリシャ文字に変換するテーブルは内蔵ずみ.
- dp_vars_prefix: 分散表現多項式は $x_0, x_1, \cdots$ の多項式として latex 形式に変換されるがこの x の部分を変更する.
- dp_vars_origin: インデックスの始まりの値を指定する. デフォールトは 0.
- dp_vars_hweyl: 分散表現多項式をワイル代数の元とみなして latex 形式に変換する. 偶数個変数があるときは 最初の半分を $x_0, x_1, \cdots$ に後半の半分を $\partial_0, \partial_1, \cdots$ に変換する. 奇数個の場合は最後の変数が同時化変数として h で表示される.
- dp_dvars_prefix: dp_vars_hweyl が 1 の時に後半部分の prefix を指定する. デフォールトは ∂
- dp_dvars_origin: dp_vars_hweyl が 1 の時のインデックスの始まりの値.
- conv_func: ユーザ定義の変換関数をよぶ.

```
[3] quotetotex(quote(1/(x+1)));
\frac{ 1} { (  {x}+ 1)}
[4]  quotetotex(objtoquote(diff(x^x,x)));
```

```
   {x}^{  {x}- 1}   {x}+  \log( {x})   {x}^{ {x}}
[5] quotetotex_env("conv_rule",3);
[6] quotetotex(objtoquote( (alpha2beta+x_i_j)^2));
  {\alpha}_{2,\beta}^{ 2} +   2  {x}_{i,j}  {\alpha}_{2,\beta}+  {x}_{i,j}^{ 2}
```

参照 Section 2.1.2 [objtoquote], page 3 print_tex_form(contrib)

ChangeLog

- この関数は 2004 年 2 月末から 3 月にかけて asir を knoppix 版 texmacs に対応させるために書かれた. Asir-contrib の print_tex_form がその原型であり, それを効率化しまた出力形式を改善した. OpenXM/src/kxx/ox_texmacs.c, OpenXM/src/texmacs も参照.
- OpenXM/src/asir-contrib/packages/src/noro_print.rr 1.1–1.8, noro_print_default.rr 1.1–1.3 も参照.
- 変更を受けたファイルは OpenXM_contrib2/asir2000 の下の次のファイル. builtin/strobj.c 1.14–1.43, include/ca.h 1.46, io/cexpr.c 1.18, io/pexpr.c 1.32, io.sexpr.c 1.29, parse/arith.c 1.12, parse/parse.h 1.28–1.29, parse/quote.c 1.7–1.8, 1.12.
- knoppix/math は 福岡大学の濱田さんが中心となり開発されている.
- dp_dvars_prefix, *_origin は builtin/strobj.c 1.46 で導入された.
- Todo: quotetoterminalform (分散表現多項式の見易い出力).

2.1.2 objtoquote

`objtoquote(`*ob*`)`
:: オブジェクトと quote 型のデータに変換する.

return quote

ob オブジェクト

- `objtoquote(ob)` は, *ob* を quote 型のデータに変換する.

```
[1150] quotetolist(quote(1+2));
[b_op,+,[internal,1],[internal,2]]
[1151] quotetolist(objtoquote(1+x));
[b_op,+,[internal,x],[internal,1]]3
```

参照 ⟨undefined⟩ [quotetotex], page ⟨undefined⟩ ⟨undefined⟩ [quotetolist], page ⟨undefined⟩

ChangeLog

- この関数は quotetotex の前処理をするために書かれた.
- asir-contrib の関数 quote_to_quote も参照.
- OpenXM_contrib2/asir2000/builtin/print.c 1.16.

2.1.3 flatten_quote

`flatten_quote(`*q*`,`*op*`)`
:: quote の括弧をとりさる.

return Quote

q Quote

op 演算子を表す文字列.

- Quote 型のデータは木構造をしている (quotetolist 参照). `quote_flatten()` は, *q* の中にあられる演算子 *op* の子供ノードを平等にする. つまり演算子 *op* に関する括弧づけがあった場合それをすべてとりさる. たとえば (1+2)+(3+4) という表現を 1+2+3+4 に変換する.
- 現在の実装では n-ary の演算子は定義されていないので, 1+2+3 は実は 1+(2+3) と表現されている. つまり + 演算子は右結合的である.
- R=0; for (I=0; I<N; I++) R = R+ P[I]; なる足し算を繰り返すと, + は左結合的になる. 右結合的に変換するには flatten_quote を呼ぶ.
- 名前は quote_flatten でなく flatten_quote である.

```
[1288] flatten_quote(quote((1+2)+(3+4*(x+3))),"+");
quote(1+2+3+4*(x+3))
[1289] flatten_quote(quote( (x*y)*(p*3)-(x*y)*z),"*");
quote(x*y*p*3-x*y*z)
[1290] quotetolist(quote(1+2+3));
[b_op,+,[b_op,+,[internal,1],[internal,2]],[internal,3]]
```

参照 ⟨undefined⟩ [quotetolist], page ⟨undefined⟩, ⟨undefined⟩ [print_tex_form], page ⟨undefined⟩(contrib)

ChangeLog

- この関数は 2004-7-7 から 2004-7-8 にかけて quote に関する操作を研究するために実験的に書かれた. OpenXM/fb で蓄積された公式の不要な括弧をとりはずし, tex 形式に変換するのに応用.
- 変更をうけたソースコードは builtin/strobj.c 1.47, parse/eval.c 1.35, parse/parse.h 1.31, parse/quote.c 1.14–1.16.

2.1.4 `quote_to_funargs`, `funargs_to_quote`, `remove_paren`

`quote_to_funargs(`*q*`)`
:: quote を funarg 形式 (リスト) へ.

`funargs_to_quote(`*f*`)`
:: funarg 形式を quote へ.

`get_function_name(`*f*`)`
:: funarg 形式の op を文字列へ.

`remove_paren(`*q*`)`
:: 上の関数を用いて書かれた余分な括弧を取り去る simplifier (asir-contrib マニュアルへ: todo)

return quote(funargs_to_quote, remove_paren) か リスト (quote_to_funargs)

q quote

f リスト

- quote_to_funargs は quote 型のデータ (内部的には FNODE) を quote への復元可能な形でリストへ変換する. quotetolist は quote をリストへ変換するが, 一部の情報を捨てるためもとの quote の復元はできない.
- quote_to_funargs の戻り値は [fid, op, arg1, arg2, ...] なる形式をしている. ここで op は node の名前であり, 関数 get_function_name を用いて人間が読める形式で取りだせる. たとえば get_function_name(quote_to_funargs(quote(1+2))[1]) は "+" を戻す.
- 名前 get_function_name はそのうち変更されるだろう.
- 下の例で quote_to_funargs(FA[2]); [34,[b_op,+,[internal,x],[internal,1]]] となる. 34 は I_PAREN を意味する. 数と意味の対応表は OpenXM/src/asir-contrib/packages/src/noro_simplify.rr または OpenXM_contrib2/asir2000/parse/parse.h を見よ. 以下の fid が 0, 1, 2, ... に対応づけられている. I_BOP, I_COP, I_AND, I_OR, I_NOT, I_CE, I_PRESELF, I_POSTSELF, I_FUNC, I_FUNC_OPT, I_IFUNC, I_MAP, I_RECMAP, I_PFDERIV, I_ANS, I_PVAR, I_ASSPVAR, I_FORMULA, I_LIST, I_STR, I_NEWCOMP, I_CAR, I_CDR, I_CAST, I_INDEX, I_EV, I_TIMER, I_GF2NGEN, I_GFPNGEN, I_GFSNGEN, I_LOP, I_OPT, I_GETOPT, I_POINT, I_PAREN, I_MINUS, I_NARYOP

次の例では (x+1)+(x+2) の括弧をはずして x+1+x+2 に変換している.

```
[0] ctrl("print_quote",1) $

[1] Q=quote((x+1)+(x+2));
[b_op,+,[u_op,(),[b_op,+,[internal,x],[internal,1]]],
        [u_op,(),[b_op,+,[internal,x],[internal,2]]]]

[2] FA=quote_to_funargs(Q);
[0,<...quoted...>,
  [u_op,(),[b_op,+,[internal,x],[internal,1]]],
  [u_op,(),[b_op,+,[internal,x],[internal,2]]]]

[3] FA2=quote_to_funargs(FA[2])[1];
[b_op,+,[internal,x],[internal,1]]

[4] FA3=quote_to_funargs(FA[3])[1];
[b_op,+,[internal,x],[internal,2]]

[5] funargs_to_quote([FA[0],FA[1],FA2,FA3]);
[b_op,+,[b_op,+,[internal,x],[internal,1]],
        [b_op,+,[internal,x],[internal,2]]]
```

次の例は OpenXM/asir-contrib 版の asir で実行.

```
[1287] load("noro_simplify.rr");
1
[1293] noro_simplify.remove_paren(quote( f(1-(x))));
quote(f(1-x))
```

funargs_to_quote を用いて既存の quote の子供を置き換えて新しい quote をつくり出せる.

```
[1184] R=quote_to_funargs(quote(a+(b+c)));
[0,<...quoted...>,<...quoted...>,<...quoted...>]
```

```
[1185] T=quote_to_funargs(quote(1+2));
[0,<...quoted...>,<...quoted...>,<...quoted...>]
[1186] funargs_to_quote([0,R[1],R[2],T[2]]);
quote(a+1)
```

参照 ⟨undefined⟩ [quotetolist], page ⟨undefined⟩

ChangeLog

- これらの関数は 2004-7-8 から開発のはじまっている quote の simplification 関連の実験的関数である. 変更をうけたソースコードは多岐にわたるのでまだ書かない.
- 括弧を取り去る問題は OpenXM/fb が蓄えている公式を tex で綺麗に表示するのが動機の一つ.
- 2004-6-26 の計算代数セミナーにおいて, 中川さんが simplifier についていろいろ問題提起をした (計算代数セミナービデオ参照).
- parse/quote.c の \tt strcut fid_spec fid_spec_tab[] の部分に書いてある形式に `funargs_to_quote` は変換する.

2.1.5 `eval_quote`

`eval_quote(`*Q*`);`
:: quote 型データ *Q* を asir のオブジェクトに変換する.

return オブジェクト

Q quote 型

- quote 型データ *Q* を asir のオブジェクトに変換する.
- 逆関数は `objtoquote`

```
ctrl("print_quote",2);
A=quote((x-1)^2+(x-1)+3);
     出力: ((((x)-(1))^(2))+((x)-(1)))+(3)
eval_quote(A);
     出力: x^2-x+3
print_input_form(A);     /* asir-contrib */
     出力: quote((x-1)^2+(x-1)+3)
```

参照 Section 2.1.2 [objtoquote], page 3, ⟨undefined⟩ [quotetolist], page ⟨undefined⟩, ⟨undefined⟩ [eval_string], page ⟨undefined⟩, Section 2.1.4 [quote_to_funargs], page 4, ⟨undefined⟩ [funargs_to_quote], page ⟨undefined⟩

ChangeLog

- ——まだ書いてない.

2.1.6 `nqt_match`

`nqt_match(`*Expr*`,`*Pattern*`[,`*Mode*`])`
:: *Expr* が *Pattern* にマッチ (適合) すると 1 を戻す. しないと 0 を戻す.

return 整数

Expr quote 型

Pattern quote 型

Mode 整数

- *Expr* が *Pattern* にマッチ (適合) すると 1 を戻す. しないと 0 を戻す.
- 適合した場合, 副作用として, *Pattern* に含まれるプログラム変数 (大文字ではじまる変数) に適合した値が代入される.
- nqt は normalized quote の略であり fnode 標準形に変換してから適合検査をする. fnode 標準形については Section 2.1.8 [qt_normalize], page 8 を見よ.
- *Mode* により展開方法を指定し, その展開方法により得られた *Expr* の fnode 標準形と *Pattern* を比較する.

```
ctrl("print_quote",2);
A=quote((x-y)*(x+y));
nqt_match(A,quote(P*Q));
[P,Q]
    出力:  [x-y, x+y]
nqt_match(A,quote(P*Q),1);
    マッチしない.
nqt_match(A,quote(P*Q),2);
    マッチしない.
qt_normalize(A,1);
    出力:  ((x)^(2))+((x)*(y))+((-1)*((y)^(2)))+((-1)*(y)*(x))
qt_normalize(A,2);
    出力:  ((x)*(x))+((x)*(y))+((-1)*(y)*(x))+((-1)*(y)*(y))
```

参照 Section 2.1.7 [nqt_match_rewrite], page 7, Section 2.1.14 [qt_rewrite], page 12

ChangeLog

- —— まだ書いてない.

2.1.7 nqt_match_rewrite

`nqt_match_rewrite(`*Expr*`,`*Rule*`,`*Mode*`)`
:: *Expr* を *Rule* に従い書き換える.

return quote 型

Expr quote 型

Rule [*Pattern*,*Action*] かまたは [*Pattern*,*Condition*,*Action*]. これらの要素はすべて quote 型.

Mode 整数

- *Expr* を *Rule* に従い書き換える. *Pattern* に適合しない場合は *Exprt* 自体を戻す.
- nqt は normalized quote の略であり fnode 標準形に変換してから適合検査をする. fnode 標準形については Section 2.1.8 [qt_normalize], page 8 を見よ.

```
ctrl("print_quote",2);
nqt_match_rewrite(‘x*y*z,[‘X*Y,‘X+Y],1);
   出力: (x)+((y)*(z))
A=‘x*x;
nqt_match_rewrite(A,[‘X*Y,‘X+Y],1);
```

```
   出力: x^2   (マッチしていない)
nqt_match_rewrite(A,[‘X*Y,‘X+Y],2);
   出力: 2*x
```

適合についてのモードの違いを理解するために次の例および fnode 標準形 (qt_normalize) を参照.

```
quotetolist(qt_normalize(‘x*x,0));
    出力:  [b_op,^,[internal,x],[internal,2]]
quotetolist(qt_normalize(‘x*x,1));
    出力:  [b_op,^,[internal,x],[internal,2]]
quotetolist(qt_normalize(‘x*x,2));
    出力:  [n_op,*,[internal,x],[internal,x]]
```

参照 Section 2.1.6 [nqt_match], page 6, Section 2.1.14 [qt_rewrite], page 12, Section 2.1.8 [qt_normalize], page 8

ChangeLog

- ——まだ書いてない.

2.1.8 qt_normalize

qt_normalize(*Expr*[,*Mode*])
:: *Expr* を fnode 標準形に変換する. *Mode* により標準形への展開アルゴリズムを指定できる.

return quote 型

Expr quote 型

Mode 整数

fnode は quote 型の実体である. fnode は木であり, 型 id および子供からなる. 型および子供を取り出す関数が funargs_to_quote である. また fnode をリストに変換する関数が quotetolist である.

fnode の標準形はパターンマッチング, 書き換えを容易におこなうために導入された. fnode の標準形を fn と書くとき, 標準形の BNF 風表現での定義は以下のとおり.

```
fn           = formula | functor(nf [,...]) | sum_of_monom
    fnode の標準形.  functor は関数よびだしみたいなもの.
sum_of_monom = monom [+ ...]
    モノミアルの和
monom        = [formula *] nfpow [* ...]
    モノミアル
nfpow        = nf | nf^(nf)
    冪乗部分の標準形
formula      = Risa object
```

- *Expr* を fnode 標準形に変換する. *Mode* により標準形への展開アルゴリズムを指定できる.
- 展開は再帰的である.
- 入力が fnode に変換された初期状態では + や * は子供が 2 人の binary operator (b_op) であるが, qt_normalize を作用させることにより, + や * は任意人数の子供を持てる n-ary operator に変換される.

- n-ary operator を基礎とした fnode 標準形を用いることにより, パターンマッチ用のパターンの数を減らせることが経験的にわかっている.
- *Mode*=0. 展開しない. これが既定の動作.
- *Mode*=1. 展開する. ただし x*x 等を x^2 等に変換
- *Mode*=2. 展開する. ただし x*x 等を x^2 等に変換しない.

Mode の違いについては以下の例も参考に.

```
ctrl("print_quote",2);
A=quote((x-y)*(x+y));
   出力:  ((x)-(y))*((x)+(y))
B=qt_normalize(A,0);
   出力: ((x)+((-1)*(y)))*((x)+(y))  Mode=0. 展開はされない. +, * は n_op (nary-op) へ
quotetolist(B);
   出力: [n_op,*,[n_op,+,[internal,x],[n_op,*,[internal,-1],[internal,y]]],[n_op,+,[

B=qt_normalize(A,1);
   出力: ((x)^(2))+((x)*(y))+((-1)*((y)^(2)))+((-1)*(y)*(x))
            Mode=1. 展開する. +, * は n_op (nary-op) へ. 巾をまとめる.
quotetolist(B);
   出力: [n_op,+,[b_op,^,[internal,x],[internal,2]],[n_op,*,[internal,x],[internal,y

qt_normalize(A,2);
   出力: ((x)*(x))+((x)*(y))+((-1)*(y)*(x))+((-1)*(y)*(y))
            Mode=2. 展開する. +, * は n_op (nary-op) へ. 巾は使わない.
quotetolist(B);
   出力: [n_op,+,[b_op,^,[internal,x],[internal,2]],[n_op,*,[internal,x],[internal,y

qt_normalize(‘x^2,2);
   出力: (x)*(x)
       Mode=2. 巾は使わない. n-ary の * へ.
```

参照 Section 2.1.6 [nqt_match], page 6, Section 2.1.7 [nqt_match_rewrite], page 7, ⟨undefined⟩ [quotetolist], page ⟨undefined⟩, Section 2.1.4 [quote_to_funargs], page 4

ChangeLog

- ——まだ書いてない.

2.1.9 qt_set_coef

`qt_set_coef`(*ParamList*)
:: 以下 *ParamList* に現れる多項式変数を変数とする有理関数体を係数とする非可換多項式を扱う.

return リスト

ParamList リスト

- 以下 *ParamList* に現れる多項式変数を変数とする有理関数体を係数とする非可換多項式を扱う.
- この宣言をしないと係数体を数とする非可換多項式として計算する.

- qt_normalize およびその機能を用いる関数がこの機能の影響を受ける.
- qt_comp 関数がこの機能の影響を受ける.

```
ctrl("print_quote",2);
qt_set_coef([a]);
B=qt_normalize(quote((a*x+a)^2),2);
     出力: ((a^2)*(x)*(x))+((2*a^2)*(x))+(a^2)
qt_normalize(B+B,2);
     出力: ((2*a^2)*(x)*(x))+((4*a^2)*(x))+(2*a^2)
```

参照 Section 2.1.8 [qt_normalize], page 8

ChangeLog

- ——まだ書いてない.

2.1.10 qt_set_ord

`qt_set_ord(`*VarList*`)`
:: *VarList* を変数順序とする.

return リスト

VarList リスト

- *VarList* を辞書式に用いた変数順序を以下使用する.
- この宣言をしないとある不定元についての既定の辞書式順序——まだ書いてない—を用いて項を比較する. *VarList* に現れない変数についてはこの順序が適用される.
- qt_normalize およびその機能を用いる関数がこの機能の影響を受ける.
- qt_comp 関数がこの機能の影響を受ける.

```
ctrl("print_quote",2);
qt_normalize(quote(x+y),2);
     出力: (x)+(y)
qt_set_ord([y,x]);
     出力: [y,x,z,u,v,w,p,q,r,s,t,a,b,c,d,e,f,g, 以下省略 ]
qt_normalize(quote(x+y),2);
     出力: (y)+(x)
```

参照 Section 2.1.8 [qt_normalize], page 8, Section 2.1.12 [nqt_comp], page 11

ChangeLog

- ——まだ書いてない.

2.1.11 qt_set_weight

`qt_set_weight(`*WeightVector*`)`
:: 変数について weight ベクトルを設定する.

return リスト

WeightVector
リスト

fnode f の weight w(f) は次の式で計算する.

`f` が葉の場合は原則 `0`. `qt_weight_vector` で `weight` が与えられている不定元についてはその値.

`f` が `node` の場合は次の規則で再帰的にきめる.
`w(f+g) = max(w(f),w(g))`
`w(f g) = w(f) + w(g)`
`w(f^n) = n w(f)`
関数については? -----まだ書いてない.

- *WeightVector* でまず順序の比較をして, それから qt_set_order による順序, 最後に既定の順序で比較する. *WeightVector* に現れない変数についての weight は 0 となる.
- qt_normalize およびその機能を用いる関数がこの機能の影響を受ける. qt_normalize での展開では, この順序を用いて項がソートされる.
- qt_comp およびその機能を用いる関数がこの機能の影響を受ける.
- weight ベクトルによる順序比較についてはグレブナ基底の節 Section 2.3.1 [dp_gr_main], page 16 も参照.

```
ctrl("print_quote",2);
qt_set_weight([[x,-1],[y,-1]]);
     結果: [[x,-1],[y,-1]]
qt_normalize(quote( 1+(x+y)+(x+y)^2),1);
     結果: (1)+(y)+(x)+((y)^(2))+((y)*(x))+((x)^(2))+((x)*(y))
```

参照 Section 2.1.8 [qt_normalize], page 8, Section 2.1.10 [qt_set_ord], page 10, Section 2.1.11 [qt_set_weight], page 11, Section 2.3.1 [dp_gr_main], page 16

ChangeLog

- ——まだ書いてない.

2.1.12 nqt_comp

`nqt_cmp(`*Expr1*`,`*Exprt2*`)`
:: *Expr1* と *Expr2* の順序を比較する.

return 整数

Expr1, Expr2
quote 型

- *Expr1* と *Expr2* の順序を比較する.
- *Expr1* > *Exprt2* なら 1.
- *Expr1* < *Exprt2* なら -1.
- *Expr1* = *Exprt2* (おなじ順序) なら 0.

```
ctrl("print_quote",2);
qt_set_ord([y,x]); qt_set_weight([[x,-1],[y,-1]]);
[nqt_comp(‘x,‘y), nqt_comp(‘y,‘x), nqt_comp(‘x,‘x)];
     出力: [-1,1,0]
```

参照 Section 2.1.8 [qt_normalize], page 8, Section 2.1.10 [qt_set_ord], page 10, Section 2.1.11 [qt_set_weight], page 11

ChangeLog

- まだ書いてない.

2.1.13 qt_is_var, qt_is_coef

`qt_is_var(`*Expr*`)`
:: *Expr* が不定元に対応する quote なら 1 を戻す.

`qt_is_coef(`*Expr*`)`
:: *Expr* が係数の有理関数体に属するとき 1 を戻す.

return 整数

Expr quote 型

- *Expr* が不定元に対応する quote なら 1 を戻す. そうでないとき 0 を戻す.

```
[qt_is_var(quote(x)), qt_is_var(quote(3/2))];
```
出力: `[1,0]`

参照 Section 2.1.14 [qt_rewrite], page 12, Section 2.1.7 [nqt_match_rewrite], page 7

ChangeLog

- ——まだ書いてない.

2.1.14 qt_rewrite

`qt_rewrite(`*Expr*`,`*Rules*`,`*Mode*`)`
:: *Expr* を規則集合 *Rules* を用いて書き換える.

return quote 型

Expr quote 型

Rules リスト

Mode 整数

- ユーザ言語を用いて定義された関数. `import("noro_rewrite.rr")` しておくこと. (noro_rewrite.rr が OpenXM/lib/asir-contrib に存在しない場合ソースの OpenXM/src/asir-contrib/testing/noro/new_rewrite.rr をコピー)
- *Expr* を規則集合 *Rules* を用いて書き換える.
- 規則の適用は fnode 木に対して再帰的である. 一方 `nqt_match_rewrite` ではトップレベルのみに規則が適用される.
- 規則集合 *Rules* の各要素の書き方は `nqt_match()` の *Pattern* と同じ書き方. つまり [パターン, 書き換え結果] または [パターン, 条件, 書き換え結果].
- *Mode* の意味は Section 2.1.8 [qt_normalize], page 8 の *Mode* と同様. パターンマッチ, 書き換えは *Mode* で `qt_normalize()` されてから遂行される.

注意: 数学的には X*Y=Y*X が可換性を与える規則だが, これをそのまま規則として与えると書き換えが停止しない. 次の例では, 上の例のように順序比較し, たとえば, 順序が大きくなる場合のみに書き換えるべきである.

```
import("noro_rewrite.rr");
R=[[‘X*Y,‘nqt_comp(Y*X,X*Y)>0, ‘Y*X]];
qt_rewrite(‘(x-y)^2,R,2);
```
出力: `quote(x*x+-2*x*y+y*y)`

外積代数の計算 (asir-contrib をロードした状態).

```
import("noro_rewrite.rr");
Rext0=[quote(X*Y),quote(qt_is_var(X) && qt_is_var(Y) && nqt_comp(Y,X)>0),
                  quote(-Y*X)];
Rext1=[quote(X^N),quote(eval_quote(N)>=2),quote(0)];
Rext2=[quote(X*X),quote(0)];
Rext=[Rext0,Rext1,Rext2];
qt_rewrite(quote( (x+2*y)*(x+4*y) ), Rext,1);
     出力: 2*x*y

qt_set_coef([a,b,c,d]);
qt_rewrite(quote((a*x+b*y)*(c*x+d*y)), Rext,1);
     出力: (d*a-c*b)*x*y
```

微分の計算 (asir-contrib をロードした状態).

```
import("noro_rewrite.rr");
qt_set_coef([a,b]);
Rd1=[‘d(X+Y), ‘d(X)+d(Y)];
Rd2=[‘d(X*Y),‘d(X)*Y+X*d(Y)];
Rd3=[‘d(N), ‘qt_is_coef(N), ‘0];
Rd4=[‘d(x),‘1];
Rd=[Rd1,Rd2,Rd3,Rd4];
B=qt_rewrite( ‘d( (a*x+b)^3),Rd,2);
     出力: quote(3*a^3*x*x+6*b*a^2*x+3*b^2*a)
fctr(eval_quote(B));
     出力: [[3,1],[a,1],[a*x+b,2]]
```

参照 Section 2.1.6 [nqt_match], page 6, Section 2.1.7 [nqt_match_rewrite], page 7, Section 2.1.8 [qt_normalize], page 8

ChangeLog

- qt 系の関数の原型は OpenXM/src/asir-contrib/testing/tr.rr である. このユーザ言語による開発が 2005 年の春まで行われ, そのあと組み込み関数主体の qt 系の関数が開発された.
- qt 系の関数についてのその他の参考文献: OpenXM/doc/Papers/2005-rims-noro.tex および OpenXM/doc/Papers/2005-rims-noro.tm (TeXmacs の記事).
- Todo: qt 系の関数を用いたおもしろい計算を Risa/Asir ジャーナルの記事として書く.

2.2 文字列処理

2.2.1 copyright

`copyright()`
:: Risa/Asir の copyright 表示を文字列として戻す.

return 文字列

- Risa/Asir の copyright 表示を文字列として戻す.

```
[1150] copyright();
This is Risa/Asir, Version 20040312 (Kobe Distribution).
Copyright (C) 1994-2000, all rights reserved, FUJITSU LABORATORIES LIMITED.
Copyright 2000-2003, Risa/Asir committers, http://www.openxm.org/.
GC 6.2(alpha6) copyright 1988-2003, H-J. Boehm, A. J. Demers, Xerox, SGI, HP.
PARI 2.0.17, copyright 1989-1999, C. Batut, K. Belabas, D. Bernardi,
   H. Cohen and M. Olivier.
```

ChangeLog

- この関数は texmacs 用に書かれた (2004-03).
- OpenXM_contrib2/asir2000 の下の以下のファイルをみよ. builtin/miscf.c 1.21, parse/glob.c 1.47.

2.2.2 string_to_tb, tb_to_string, write_to_tb

`string_to_tb`(*s*)
`tb_to_string`(*tb*)
`write_to_tb`(*s*,*tb*)
:: 文字列可変長配列型 (text buffer) のデータの処理

return 文字列可変長配列型 (string_to_tb), 文字列型 (tb_to_string)

s 文字列

tb 文字列可変長配列型

- `string_to_tb`(*s*) は, 文字列 *s* をはじめの要素とする文字列可変長配列型オブジェクトを生成する.
- `tb_to_string_`(*tb*) は, 文字列可変長配列型オブジェクト *tb* から通常の文字列オブジェクトを生成する.
- `write_to_tb`(*s*,*tb*) は, 文字列 *s* を文字列可変長配列型オブジェクト *tb* へ書き出す.
- *SS* を文字列変数とするとき, *SS* += "文字列" で *SS* へ文字列を書き足していくことができるが, 無駄なメモリを大量に消費する. 代りに関数 `write_to_tb` を用いるべきである. 文字列可変長配列型オブジェクトは文字列の可変長の配列でありメモリ管理に優しいデータ構造である.

```
[219] T=string_to_tb("");

[220] write_to_tb("Hello",T);
0
[221] write_to_tb(" world!",T);
0
[222] tb_to_string(T);
Hello world!
```

ChangeLog

- この関数は 2004-3 に print_tex_form を効率化するために書かれた.
- OpenXM_contrib2/asir2000 の下の以下のファイルを見よ. io/ox_asir.c 1.52, builtin/strobj.c 1.12–1.13, 1.16, engine/str.c 1.5, parse/quote.c 1.9.
- rtostr が text buffer 型のデータに関しておそかった. 速度の改善は asir2000/io/pexpr_body.c 1.2, asir2000/parse/lex.c 1.32.

2.2.3 set_print_function

`set_print_function([`*fname*`])`
:: 画面表示用の関数を登録

return 整数

fname 文字列

- `set_print_function` は `fname(F)` を通常の画面表示関数の代わりによぶ. 引数がない場合は画面表示関数をデフォールトへ戻す. Asir-contrib はこの関数を用いて出力関数を Asir-contrib 用に変更している.

```
[219] def my_output(F) {
           print("Out: ",0); print(rtostr(F));
      }
[220] set_print_function("my_output");
Out: 0
[221]  1+2;
Out: 3
```

参照 ⟨undefined⟩ [rtostr], page ⟨undefined⟩

ChangeLog

- この関数は 2001-9-4 に asir-contrib のために導入された. 変更をうけたソースコードは builtin/print.c 1.11 である.

2.2.4 printf, fprintf, sprintf

`printf(`*format*`[,`*args*`])`
`fprintf(`*fd*`,`*format*`[,`*args*`])`
`sprintf(`*format*`[,`*args*`])`
:: C に似たプリント関数

return 整数 (printf,fprintf), 文字列 (sprintf)

format 文字列

fd 非負整数 (ファイル記述子)

args オブジェクト

- `printf` は書式文字列 *format* にしたがい, オブジェクト *args* を標準出力に書き出す.
- `fprintf` は結果を, ファイル記述子 *fd* の指すファイルに書き出す.
- `sprintf` は結果を文字列で返し, 標準出力には書き出さない.
- 書式文字列の中で `%a` (any) が利用可能. *args* の個数は書式文字列の中の `%a` の個数に等しくすること.
- ファイル記述子は, `open_file` 関数を用いて得ること.

```
[0] printf("%a: rat = %a\n",10,x^2-1)$
10: rat = x^2-1
[1] S=sprintf("%a: rat = %a",20,x^2-1)$
[2] S;
```

```
20: rat = x^2-1
[3] Fd=open_file("hoge.txt","w");
0
[4] fprintf(Fd,"Poly=%a\n",(x-1)^3)$
[5] close_file(Fd)$
[6] quit;

$ cat hoge.txt
Poly=x^3-3*x^2+3*x-1
```

参照 ⟨undefined⟩ [rtostr], page ⟨undefined⟩,⟨undefined⟩ [open_file], page ⟨undefined⟩,⟨undefined⟩ [close_file], page ⟨undefined⟩

ChangeLog

- 関数 sprintf は 2004-7-13 にコミットされた. 変更をうけたソースコードは builtin/strobj (1.50) である.
- 関数 printf は 2007-11-8 にコミットされた. 変更をうけたソースコードは builtin/print.c (1.23) である.
- 関数 fprintf は 2008-11-18 にコミットされた. 変更をうけたソースコードは builtin/file.c (1.25) である.
- %a は Maple の sprintf の真似か.

2.3 グレブナー基底

2.3.1 dp_gr_main

dp_gr_main(*f* | v=*vv*, order=*oo*, homo=*n*, matrix=*m*, block=*b*, sugarweight=*sw*)
:: dp_gr_main の新しいインタフェース.

return リスト (グレブナ基底. 再帰表現多項式か分散表現多項式のリスト)

f リスト (入力多項式系. 再帰表現多項式か分散表現多項式のリスト)

vv リスト (変数のリスト)

oo リスト (順序をあらわすリスト)

n 0 か 1 (homogenization をするか)

m 順序を matrix で表現する場合 (cf. dp_ord).

b ???

sw Sugar strategy を適用するときの weight vector. 全ての要素は非負.

- dp_gr_main(*f*) は, *f* のグレブナ基底を計算する. グレブナ基底は順序を変えるとその形が変わる. asir ではいままで順序の指定方法が系統だっていなかった. dp_gr_main の新しいインタフェースでは順序をある文法に従い指定する.
- 順序 order は次の文法で定義する. {, } は 0 回以上の繰り返しを意味する.

```
order          : '[' orderElement { ',' orderElement } ']'
orderElement   :  weightVec | builtinOrder
```

```
weightVec     : '[' weightElement { ',' weightElement } ']'
builtiniOrder : '[' orderName  ',' setOfVariables ']'
weightElement :   NUMBER | setOfVariables ',' NUMBER
setOfVariables:  V | range(V,V)
orderName     : @grlex | @glex | @lex
```

ここで V は 変数名, NUMBER は整数をあわらす. 例 1: v=[x,y,z,u,v], order=[[x,10,y,5,z,1],[@grlex,range(x,v)]] は x,y,z がそれぞれ weight 10, 5, 1 をもつ順序で比較したあと, [x,y,z,u,v] についての graded reverse lexicographic order を tie-breaker として用いることを意味する. 参考書: B.Sturmfels: Gr\"obner Bases and Convex Polytopes (1995). M.Saito, B.Sturmfels, N.Takayama: Gr\"obner Deformations of Hypergeometric Differential Equations (2000).

- 順序要素 (orderElement) の指定方法は (1) 変数名または range で指定された変数の集合と重みの値の繰り返し (2) 重みの値を変数リストの順番に並べる方法 (3) 変数名または range で指定された変数の集合と順序名の組の三通りの基礎的方法がある. 似た指定方法が Macaulay, Singular, CoCoA, Kan/sm1 等の環論システムで使用されていた. Risa/Asir の指定方法はこれらのシステムの指定方法を参考にさらに改良を加えたもので柔軟性が高い.
- order の tie-breaker は grlex がデフォールト.
- 分散表現多項式を引数としたときは結果も分散表現多項式として戻る. order 指定にもちいるデフォールトの変数名はこのとき x0, x1, x2, ... となる.
- オプションの値は option_list キーワードを用いてリストで与えてもよい. 下の例を参照.

```
[218] load("cyclic");
[219] V=vars(cyclic(4));
[c0,c1,c2,c3]
[220]dp_gr_main(cyclic(4) | v=V, order=[[c0,10,c1,1],[c2,5],[@grlex,range(c0,c3)]])
[ 10 1 0 0 ]
[ 0 0 5 0 ]
[ R R R R ]
[(-c3^6+c3^2)*c2^2+c3^4-1,c3^2*c2^3+c3^3*c2^2-c2-c3,
 (c3^4-1)*c1+c3^5-c3,(c2-c3)*c1+c3^4*c2^2+c3*c2-2*c3^2,-c1^2-2*c3*c1-c3^2,
 c0+c1+c2+c3]

[1151]  F=map(dp_ptod,katsura(4), vars(katsura(4)));
[(1)*<<1,0,0,0,0>>+(2)*<<0,1,0,0,0>>+ ... ]
[1152] dp_gr_main(F | order=[[range(x0,x3),1]]);
[ 1 1 1 1 0 ]
[ R R R R R ]
[(47774098944)*<<0,0,0,0,13>>+ ... ]

[1153] Opt=[["v",[x,y]], ["order",[[x,5,y,1]]]];
[[v,[x,y]],[order,[[x,5,y,1]]]]
[1154]  dp_gr_main([x^2+y^2-1,x*y-1] | option_list=Opt);
[ 5 1 ]
[ R R ]
[-y^4+y^2-1,x+y^3-y]
```

ChangeLog

- この関数は 2003-12 から 2004-2 の始めに大きな修正が行われた.

- `setOfVariables`の表現のために range オブジェクトが導入された.
- グレブナ基底は順序を変えるとその形が変わる. asir ではいままで順序の指定方法が系統だっていなかった. dp_gr_main の新しいインタフェースでは順序をある文法に従い指定する.
- OpenXM_contrib2/asir2000 の下の次の各ファイルが修正をうけた. builtin/gr.c 1.56–1.57, builtin/dp-supp.c 1.27–1.31 (create_composite_order_spec), builtin/dp.c 1.46–1.48 (parse_gr_option), engine/Fgfs.c 1.20, engine/dist.c 1.27–1.28 engine/nd.c 1.89, include/ca.h 1.42–1.43, io/pexpr.c 1.28, io/sexpr.c 1.26, parse/arith.c 1.11, parse/glob.c 1.44-1.45, parse/lex.c 1.29, parse/parse.h 1.23–1.26
- Todo: return キーワードで戻り値のデータを quote のリストにできるように. attribute, ring 構造体.

2.3.2 dp_weyl_gr_main

`dp_weyl_gr_main(`*f* `| v=`*vv*`, order=`*oo*`, homo=`*n*`, matrix=`*m*`, block=`*b*`, sugarweight=`*sw*`)`

:: dp_weyl_gr_main の新しいインタフェース. dp_gr_main と同じ形式である.

return リスト (グレブナ基底. 再帰表現多項式か分散表現多項式のリスト)

f リスト (入力多項式系. 再帰表現多項式か分散表現多項式のリスト)

vv リスト (変数のリスト)

oo リスト (順序をあらわすリスト)

n 0 か 1 (homogenization をするか). [テストまだ]

m 順序を matrix で表現する場合 (cf. dp_ord). [テストまだ]

b ???

sw Sugar strategy を適用するときの weight vector. 全ての要素は非負. [テストまだ]

- `dp_weyl_gr_main(`*f*`)` は, *f* のグレブナ基底を計算する. グレブナ基底は順序を変えるとその形が変わる. asir ではいままで順序の指定方法が系統だっていなかった. dp_weyl_gr_main の新しいインタフェースでは順序をある文法に従い指定する. 指定方法については dp_gr_main のマニュアルを参照.
- 分散表現多項式の各モノミアルの長さが偶数のときはワイル代数 K[x_1, ..., x_n, d_1, ..., d_n] で計算がおこなわれる. ワイル代数では x_i と d_i は非可換な掛け算規則 d_i x_i = x_i d_i +1 をみたし, x_i と x_j や d_i と d_j は可換である. また i と j が異なる場合は x_i と d_j も可換である.
- 分散表現多項式の各モノミアルの長さが奇数のときは同次化ワイル代数 K[x_1, ..., x_n, d_1, ..., d_n, h] で計算がおこなわれる. 同次化ワイル代数では x_i と d_i は非可換な掛け算規則 d_i x_i = x_i d_i + h^2 をみたし, h は任意の元と可換, その他の変数もワイル代数と同様な可換性の規則をみたす. 詳しくは dp_gr_main で参照した Saito, Sturmfels, Takayama の教科書をみよ.

```
[1220] F=sm1.gkz([ [[1,1,1,1],[0,1,3,4]], [0,0]]);  /* Command in asir-contrib*/
[[x4*dx4+x3*dx3+x2*dx2+x1*dx1,4*x4*dx4+3*x3*dx3+x2*dx2,-dx1*dx4+dx2*dx3,-dx2^2*dx4+
[1221] V=[x1,x2,x3,x4,dx1,dx2,dx3,dx4]$
[1222] dp_weyl_gr_main(F[0] | v=V, order=[[dx1,1,dx2,1,dx3,1,dx4,1]]);
```

```
...
[1238] FF=map(dp_ptod,F[0],V);
[(1)*<<1,0,0,0,1,0,0,0>>+(1)*<<0,1,0,0,0,1,0,0>>+(1)*<<0,0,1,0,0,0,1,0>>+(1)*<<0,0,

[1244] FF=map(dp_ptod,F[0],V);
[(1)*<<1,0,0,0,1,0,0,0>>+(1)*<<0,1,0,0,0,1,0,0>>+(1)*<<0,0,1,0,0,0,1,0>>+(1)*<<0,0,

dp_weyl_gr_main(FF | v=V, order=[[0,0,0,0,1,1,1,1]]);

[1246] dp_weyl_gr_main(FF | v=V, order=[[dx1,1,dx2,1,dx3,1,dx4,1]]);
[ 0 0 0 0 1 1 1 1 ]
[ R R R R R R R R ]
 ...
```

参照 Section 2.3.1 [dp_gr_main], page 16

ChangeLog

- dp_gr_main のインタフェースが dp_weyl_gr_main へも導入された.
- OpenXM_contrib2/asir2000 の下の次の各ファイルが修正をうけた. builtin/dp-supp.c 1.32–1.33 builtin/dp.c 1.49–1.50

2.3.3 dp_initial_term

`dp_initial_term(`*f* `| v=`*vv*`, order=`*oo*`)`
:: dp_initial_term は与えられた weight に対する先頭項の和を戻す.

return 分散表現多項式または分散表現多項式のリスト.

f 分散表現多項式か分散表現多項式のリスト.

vv リスト (変数のリスト)

oo リスト (順序をあらわすリスト)

- dp_initial_term は与えられた weight w に対する先頭項の和を戻す. これは多くの教科書で $\mathrm{in}_w(f)$ と書かれている.
- 順序を表すリストは dp_gr_main で定義した文法に従う. このリストの先頭が weight vector で無い場合はエラーとなる. たとえば order=[[@lex,...]] はエラーとなる.
- 結果は与えられた順序に関してソートされてるわけではない.

```
[1220] F=<<2,0,0>>+<<1,1,0>>+<<0,0,1>>;
(1)*<<2,0,0>>+(1)*<<1,1,0>>+(1)*<<0,0,1>>
[1220] dp_initial_term(F | order=[[1,1,1]]);
[ 1 1 1 ]
[ R R R ]
(1)*<<2,0,0>>+(1)*<<1,1,0>>
[1221] dp_initial_term(F | v=[x,y,z], order=[[x,1]]);
[ 1 0 0 ]
[ R R R ]
(1)*<<2,0,0>>
```

参照 Section 2.3.1 [dp_gr_main], page 16, Section 2.3.2 [dp_weyl_gr_main], page 18, Section 2.3.4 [dp_order], page 20, ⟨undefined⟩ [dp_hm], page ⟨undefined⟩

ChangeLog

- OpenXM_contrib2/asir2000 の下の次の各ファイルが修正をうけた. builtin/dp-supp.c 1.32 builtin/dp.c 1.49

2.3.4 dp_order

dp_order(*f* | v=*vv*, order=*oo*)
:: dp_order は与えられた weight に対する次数の最大値を戻す.

return 数か数のリスト

f 分散表現多項式か分散表現多項式のリスト.

vv リスト (変数のリスト)

oo リスト (順序をあらわすリスト)

- 順序を表すリストは dp_gr_main で定義した文法に従う. このリストの先頭が weight vector で無い場合はエラーとなる. たとえば order=[[@lex,...]] はエラーとなる.
- dp_order は与えられた weight w に対する次数の最大値を戻す. これを $\mathrm{ord}_w(f)$ と書く論文や教科書もある.
- 引数がリストの場合各要素の次数が計算される.

```
[1220] F=<<2,0,0>>+<<1,1,0>>+<<0,0,1>>;
(1)*<<2,0,0>>+(1)*<<1,1,0>>+(1)*<<0,0,1>>
[1222] dp_order(F | order=[[1,1,1]]);
[ 1 1 1 ]
[ R R R ]
2
[1223] dp_order(F | v=[x,y,z], order=[[x,1]]);
[ 1 0 0 ]
[ R R R ]
```

参照 Section 2.3.1 [dp_gr_main], page 16, Section 2.3.2 [dp_weyl_gr_main], page 18, Section 2.3.3 [dp_initial_term], page 19, ⟨undefined⟩ [dp_hm], page ⟨undefined⟩

ChangeLog

- OpenXM_contrib2/asir2000 の下の次の各ファイルが修正をうけた. builtin/dp-supp.c 1.32 builtin/dp.c 1.49

2.3.5 nd_gr, nd_gr_trace (加群)

nd_gr(*gen*,*vars*,*char*,*ord*)
nd_gr_trace(*gen*,*vars*,*homo*,*char*,*ord*)
nd_weyl_gr(*gen*,*vars*,*char*,*ord*)
nd_weyl_gr_trace(*gen*,*vars*,*homo*,*char*,*ord*)
:: 部分加群のグレブナー基底の計算

gen リストのリスト

ord *[IsPOT,Ord]* なるリスト

return リストのリスト

- 多項式環あるいはワイル代数上の自由加群の部分加群のグレブナー基底を計算する. 結果はリストのリストである. 各要素リストは, 自由加群の元であるベクトルとみなす.
- *ord* として *[IsPOT,Ord]* という 2 要素リストが指定された場合, 加群のグレブナー基底計算を実行する. この場合, *gen* は, 多項式のリストのリストとして与える必要がある.
- *IsPOT* が 1 の場合, POT (position over term), 0 の場合 TOP (term over position) で比較する. 基礎環での項比較は *Ord* で行う.
- 説明されていない引数は, イデアルの場合の解説を参照のこと.

```
[0] Gen=[[x,y,z],[y^2+x,x^2,z],[y^2,z^3+x,x+z]];
[[x,y,z],[x+y^2,x^2,z],[y^2,x+z^3,x+z]]
[1]  nd_gr(Gen,[x,y,z],0,[0,0]);
[[x,y,z],[y^2,x^2-y,0],[y^2,x+z^3,x+z],[y^3+z^3*y^2,y^3*x-y^3,
-x^3-z*x^2+(z*y^2+y)*x-z*y^2+z*y],[0,0,x^4+z*x^3+(-z*y^2-y)*x^2
+(-y^3+z*y^2-z*y)*x+z^4*y^2]]
```

参照 ⟨undefined⟩ [nd_gr], page ⟨undefined⟩, ⟨undefined⟩ [nd_gr_trace], page ⟨undefined⟩

2.3.6 `nd_gr`, `nd_gr_trace` (option)

`nd_gr(`*...*`[|`*opt*`,`*opt*`,...])`
`nd_gr_trace(`*...*`[|`*opt*`,`*opt*`,...])`
`nd_weyl_gr(`*...*`[|`*opt*`,`*opt*`,...])`
`nd_weyl_gr_trace(`*...*`[|`*opt*`,`*opt*`,...])`
:: グレブナー基底計算に関する種々のオプションの説明

opt *key=value* なるオプション設定

return オプションにより異なる

- グレブナー基底計算をオプションにより制御する.
- 現状では次の 3 つのオプションを受け付ける.

  gentrace *value* が 0 でないとき, グレブナー基底の計算経過情報を出力する.

  gentsyz *value* が 0 でないとき, 計算されたグレブナー基底に対する syzygy の生成系を出力する.

  nora *value* が 0 でないとき, 最終ステップで相互簡約を行わない.
- *gentrace* が指定された場合, 出力は, *[GB,Homo,Trace,IntRed,Ind,InputRed,SpairTrace]* なるリストである. 各要素の意味は次の通りである.

  GB グレブナー基底

  Homo 中間基底が斉次化されている場合 1, そうでない場合 0.

  Trace 全中間基底に対する計算経過情報

  IntRed 相互簡約に対する計算経過情報

  Ind 簡約グレブナー基底の各要素の, 全中間基底のにおけるインデックス

InputRed 各入力多項式をグレブナー基底で簡約して剰余 0 を得るまでの計算経過情報 (*gensyz* が指定された場合)

SpairTrace
簡約グレブナー基底に対する S 多項式を簡約して剰余 0 を得るまでの計算経過情報 (syzygy 加群の生成系の要素のみ; *gensyz* が指定された場合)

- 詳細は, 入力多項式集合とグレブナー基底の相互変換行列, および syzygy 計算関数の項で説明する予定.

```
[0] C=[c3*c2*c1*c0-1,((c2+c3)*c1+c3*c2)*c0+c3*c2*c1,...]
[1] D=nd_gr_trace(C,[c0,c1,c2,c3,c4],0,1,0|gentrace=1,gensyz=1)$
[2] D[0];
[c0+c1+c2+c3,-c1^2-2*c3*c1-c3^2,...]
[3] D[2];
[[[0,0,1],[1,1,1],[2,2,1],[3,3,1]],[4,[[1,2,(1)*<<0,0,0,0>>,1],...]
[4] D[6];
[[-1,[[1,0,(1)*<<0,0,2,4>>,1],[1,6,(-1)*<<1,0,0,0>>,1],...]
```

参照 ⟨undefined⟩ [nd_gr], page ⟨undefined⟩, ⟨undefined⟩ [nd_gr_trace], page ⟨undefined⟩

2.4 システム

2.4.1 asir-port.sh, asir-install.sh

`asir-install.sh`
`asir-port.sh`
:: これは asir の内部コマンドではない. asir をネットワークからダウンロードかつ実行するシェルスクリプト

- asir-port.sh は knoppix 専用である. このコマンドは asir のバイナリおよび FLL で配布できない部分を ftp.math.kobe-u.ac.jp よりダウンロードして/home/knoppix/.asir-tmp へセーブして, 実行する. .asirrc および .TeXmacs/plugins/ox/progs/init-ox.scm もダウンロードする.
- asir-install.sh は Debian GNU Linux / openxm-binary*.deb 専用である. asir-install.sh は asir をダウンロードして /usr/local/OpenXM/bin および/usr/local/OpenXM/lib/asir へインストールする.

ChangeLog

- これらのシェルスクリプトは knoppix/math のために 2004/2, 3 月に書かれた.
- knoppix/math は福岡大学の濱田さんが中心となり開発されている.
- OpenXM/misc/packages/Linux/Debian の下の全てのファイル (2004-2-22 から 2004-3 の末まで). (~taka/this03/misc-2003/A3/knoppix-03-05 (プライベートファイル) も見よ.) OpenXM/src/asir-port の下の次の各ファイル. Makefile 1.1–1.8, asir-install.sh 1.1–1.2, asir-port.sh 1.1–1.6.

2.4.2 asirgui.hnd

asirguid.hnd
:: asirgui の main window のハンドル番号を保持するファイル

- asirgui (Windows 版) を起動すると asirgui.exe のあるフォルダおよび 環境変数 TEMP が定義されていればこのファイルが作成される.
- 中身は１０進整数で, asirgui の main window のハンドルである. このハンドルあてに PostMessage をすれば, asuirgui にキーボードから入力したのと同様な効果が得られる.
- text editor で作成, 保存したファイルを text editor 側から asirgui に読み込ませたりするために利用可能.
- http://www.math.kobe-u.ac.jp/Asir/Add-ons にて winfep.exe を配布している. winfep ではあらかじめファイルに入力スクリプトを書いておいて, asirgui で一行づつ実行させることができる. winfep はプレゼンテーション用のソフトウエアである. これは asirgui.hnd を利用している.

```
// cl test.c user32.lib

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <process.h>

int main()
{

HWND hnd;
FILE *fp = fopen("c:/Program Files/asir/bin/asirgui.hnd","r");
fscanf(fp,"%d",&hnd);
fclose(fp);
while (1) {
int c;
c = getchar();
if ( c == '#' ) break;
PostMessage(hnd,WM_CHAR,c,1);
}
return 0;
}
```

参照 ⟨undefined⟩ [xyz_abc], page ⟨undefined⟩

ChangeLog

- この機能は 2006-12-5, 2007-02-13 に加えられた.
- OpenXM_contrib2/windows/asir32gui/asir32gui.clw 1.11
- OpenXM_contrib2/windows/asir32gui/asir32guiview.cpp 1.15, 1.1.6
- winfep は 2010-01-20 頃に第一版が commit された.
- OpenXM_contrib2/windows/winfep 1.1

2.4.3 chdir, pwd

chdir(*directory*)

pwd() :: シェルコマンド cd と pwd に対応する操作.

return 文字列 (pwd), 整数 (chdir)

dirctory 文字列

- pwd はカレントディレクトリを文字列で返す.
- chdir はカレントディレクトリを *directory* に変更する. 成功すれば 0 を失敗すれば -1 を返す.
- これらの関数は UNIX 版にのみ実装されている.

```
[0] S=pwd();
/home/ohara
[1] chdir(".../taka");
-1
[2] chdir("/usr/bin");
0
```

ChangeLog

- これらの関数は 2008-8-27 にコミットされた. 変更をうけたソースコードは builtin/miscf.c (1.27) である.

2.4.4 dcurrenttime

dcurrenttime()
:: 現在時刻を取得.

return 浮動小数点数

- 返り値は 1970 年 1 月 1 日 0 時 0 分 0 秒からの経過秒数である.

```
[0] ctrl("real_digit", 16);
16
[1] dcurrenttime();
1226390851.34476
[2] currenttime();
1226390854
```

参照 ⟨undefined⟩ [currenttime], page ⟨undefined⟩

ChangeLog

- この関数は 2008-9-12 にコミットされた. 変更をうけたソースコードは builtin/time.c (1.6) である.

2.5 言語

2.5.1 get_struct_name, get_element_names, get_element_at, put_element_at

`get_struct_name(`*s*`)`
`get_element_names(`*s*`)`
`get_element_at(`*s*`,`*key*`)`
`put_element_at(`*s*`,`*key*`,`*obj*`)`
:: 構造体 *s* に対する操作

return 文字列 (get_struct_name), 文字列のリスト (get_element_names), オブジェクト (get_element_at), オブジェクト (put_element_at)

s 構造体

key 文字列

obj オブジェクト

- `get_struct_name(s)` は, 構造体 *s* の名前を戻す.
- `get_element_names(s)` は, 構造体のメンバーの名前のリストを戻す.
- `get_element_at(s,key)` は構造体 s のメンバー key の値を戻す.
- `put_element_at(s,key,obj)` は構造体 s のメンバー key の値を obj に設定する.

```
[219]  struct point { x, y, color};
[220]  P = newstruct(point);
{0,0,0}
[221]  P->x = 10$ P->y=5$ P->color="red"$
[222]  get_element_names(P);
[x,y,color]
[223] put_element_at(P,"color","blue");
blue
[224] P->color;
bule
```

参照 ⟨undefined⟩ [newstruct], page ⟨undefined⟩, ⟨undefined⟩ [struct], page ⟨undefined⟩

ChangeLog

- 構造体の定義を知らずに構造体を扱うユーザ関数を書くときに便利. asir-contrib の noro_print.rr を見よ.
- OpenXM_contrib2/asir2000/builtin/compobj.c 1.8.

2.5.2 mapat

`mapat(`*fname*`,`*pos*`[,`*arg0*`, `*arg1*`, ...])`
:: *pos* に対する map 関数

return オブジェクト

pos 整数

arg0, arg1, arg2, ...
オブジェクト

- `map` 関数は 0 番目の引数に対してしか動作しないが, `mapat` 関数は指定した番号の引数に対して `map` 関数を実行する.
- `mapat(fname,0,A0,A1,...)` は `map(fname,A0,A1,...)` に等価である.
- 次の副作用がある. まだ書いてない.

```
[219] mapat(deg,1,x^2+y^3+x+y,[x,y]);
[2,3]
[220] mapat(subst,1,x+y+z,[x,y,z],2);
[y+z+2,x+z+2,x+y+2]
```

参照 ⟨undefined⟩ [map], page ⟨undefined⟩

ChangeLog

- この関数は 2004-6-22 にコミットされた. 変更をうけたソースコードは builtin/pf.c, subst.c である.

2.5.3 list

`list([`*arg0*`, `*arg1*`, ...])`
:: list を生成する.

return リスト

arg0, arg1, arg2, ...
オブジェクト

- *arg0*, *arg1*, ... を要素とするリストを生成する.

```
[219] list(1,2,3);
[1,2,3]
[220] list(1,2,[3,4]);
[1,2,[3,4]]
```

参照 ⟨undefined⟩ [cons], page ⟨undefined⟩

ChangeLog

- この関数は 2004-6-22 にコミットされた. 変更をうけたソースコードは builtin/list.c である.

2.5.4 assoc

`assoc(`*a*`,`*b*`)`
:: 連想リストをつくる

return List

a List

b List

- リスト *a*, *b* より [[a[0],b[0]], [a[1],b[1]], ...] なる新しいリストを生成する.

下の例では `A` に動物の名前が, `B` に足の本数が入っている. `assoc(A,B)` で動物と足の本数をペアにしたリストを生成する.

```
[1192]  A=["dog","cat","snake"];
[dog,cat,snake]
[1193] B=[4,4,0];
[4,4,0]
[1194] assoc(A,B);
[[dog,4],[cat,4],[snake,0]]
```

参照 ⟨undefined⟩ [cons], page ⟨undefined⟩, ⟨undefined⟩ [append], page ⟨undefined⟩

ChangeLog

- この関数は 2004-6-28 に書かれた. 変更をうけたソースコードは builtin/list.c 1.9 parse/eval.c 1.35, parse/parse.h 1.31, parse/quote.c 1.14–1.16.

2.5.5 set_secure_flag, set_secure_mode

`set_secure_flag`(*fname*,*m*)
`set_secure_mode`(*m*)
:: 関数の実行権限を設定する. (web サービス用)

return 整数

fname 文字列

m 整数

- `set_secure_flag`, `set_secure_mode` は asir を web サーバ等で公開するために加えられた関数. `set_secure_flag` で公開する関数を指定する. `secure_mode` が 1 の場合は `set_secure_flag` で指定された関数しか実行できない. 関数の実行途中では `secure_mode` が 0 となっているので, 任意の関数を実行できる. またエラーの時等は, `secure_mode` は 1 に自動的に復帰する. ただし `def` は実行できない. 公開する関数では, その処理中は任意の関数が実行できるので, security に十分注意した実装をする必要がある.
- `set_secure_flag` は, *fname* の secure flag を *m* にする. 公開する命令は 1 に設定する.
- *set_secure_mode(1)* で `secure_mode` が 1 となり, 公開された関数しか実行できなくなる. quit 等も実行できないので注意.
- `timer` の引数として secure_flag を設定していない関数を指定して実行してもエラーが表示されない. このときは, `ctrl("error_in_timer",1)` を実行しておく.

```
[1194] set_secure_flag("print_input_form_",1);
1
[1195] set_secure_flag("fctr",1);
1
[220] set_secure_mode(1);
1
[1197] fctr((x-1)^3);
[[1,1],[x-1,3]]
[1198] fctr(shell("ls"));
evalf : shell not permitted
return to toplevel
```

参照 ⟨undefined⟩ [timer], page ⟨undefined⟩

ChangeLog

- set_secure_flag, set_secure_mode は asir を web サーバ等で公開するために加えられた関数．sm1 の同様な関数 RestrictedMode で採用された方法を用いている．つまり, set_secure_flag で公開する関数を指定する．secure_mode が 1 の場合は set_secure_flag で指定された関数しか実行できない．v 関数の実行途中では secure_mode が 0 となっているので, 任意の関数を実行できる．
- この関数は 2004-10-27 から 2004-11-22 にかけて開発された．
- cgiasir.sm1, cgi-asir.sh と組み合わせて cgi サービスを提供するために利用する．cgi-asir.sh では CGI_ASIR_ALLOW 環境変数で公開するコマンドを指定する．
- 1.24–1.25 OpenXM_contrib2/asir2000/builtin/miscf.c
- 1.36–1.38 OpenXM_contrib2/asir2000/parse/eval.c
- 1.6–1.7 OpenXM_contrib2/asir2000/parse/function.c
- 1.33 OpenXM_contrib2/asir2000/parse/parse.h

2.5.6 initialize_static_variable

static 変数の初期化の問題点．初期化の時に segmentation fault がおきる．

- static 変数の取扱．下の例を参照のこと．

```
if (1) {
 module abc;
 static A;
 A=1;
 endmodule;
} else { };

end$
```

を t.rr とするとき，

```
[6] load("./t.rr");
1
internal error (SEGV)
```

となる．

t.rr を

```
if (1) {
 module abc;
 static A;
 localf initA;
 localf foo;
 def initA() {
   A=1;
 }
 initA();
 def foo() {
   return A;
 }
 endmodule;
} else { };
```

```
end$
```
とすると正しく初期化される.

ChangeLog

- oxasir-win.rr の取扱で問題点として浮上. 2005.07.25.
- oxasir-win.rr の取扱で double quote の取り扱いに問題があったが, これは asir2000/io/ox_asir.c, 1.58, で問題点解決.
- ox_asir に計算を依頼する時は if (1) { ... }で囲む.

2.6 数論・代数

2.6.1 small_jacobi

`small_jacobi`(*a*,*m*)
:: Jacobi 記号の計算

return 整数

arg1, arg2 整数

- *m* が素数のときは Legendre 記号とよばれ, x^2 = *a* mod *m* に解があるとき 1, 解がないとき -1 をもどす.
- Jacobi 記号は Legendre 記号の積で定義される (初等整数論の本参照).
- この関数は machine int の範囲で jacobi 記号を計算する.

```
[1286] small_jacobi(2,3);
-1
[1287] small_jacobi(2,7);
1
```

参照 http://members.jcom.home.ne.jp/yokolabo/asirlib/ も見てね.

ChangeLog

- この関数の由来は不明.

2.6.2 noro_matrix.rr

`linalg.unit_mat`(*arg1*)

`linalg.random_rmat`(*arg1*, *arg2*, *arg3*)

`linalg.minipoly_mat`(*arg1*)

`linalg.compute_kernel`(*arg1*)

`linalg.compute_image`(*arg1*)

`linalg.jordan_canonical_form`(*arg1*)

- 簡単な解説および実例は http://www.math.kobe-u.ac.jp/HOME/taka/2007/knx/noro_matrix-ja.txt を参照.

```
load("noro_matrix.rr");
A=newmat(4,4,[[2,0,0,0],[3,5,1,0],[-9,-9,-1,0],[-5,0,0,1]]);
B=linalg.jordan_canonical_form(A);
```

参照 ⟨undefined⟩ [invmat], page ⟨undefined⟩

ChangeLog

- この関数は 2004-04 頃から線形代数 III の講義をしながら書かれた.
- ソース: OpenXM/src/asir-contrib/packages/src/noro_matrix.rr

2.6.3 f_res

f_res

- f_res は各種の終結式を計算するモジュールである. ox_grep("f_res"); で online manual を閲覧可能である.

参照 ⟨undefined⟩ [], page ⟨undefined⟩

ChangeLog

- このモジュールは Fujiwara 君の修士論文が元になり, それを改造したものである.
- OpenXM/src/ox_cdd, OpenXM/src/asir-contrib/packages/src/f_res.rr

2.7 D 加群の制限に関する関数

2.7.1 nk_restriction.restriction

nk_restriction.restriction(*Id*, *VL*, *DVL*, *W*)
:: ホロノミック D イデアル *Id* を重みベクトル *W* についての制限加群を返す。

Id イデアルの生成元のリスト

VL 変数のリスト

DVL 変数のリスト (*VL* に対応する微分作用素の方の変数)

W 重みベクトルを表すリスト

- *W* の要素は非負整数で、0 番目の要素から連続して正の整数が入らなければならない。(すなわち、[1,1,0,0,0] は OK だが、 [1,0,1,0,0] はダメ)
- 正の重みを持つ変数についての制限を行う。例えば、*VL* =[x,y,z], *W* =[1,1,0] であれば x,y について制限を行う。

以下は、イデアル $I = D\cdot\{x\partial_x - 1, y\partial_y - 1\}$ の x についての制限加群を計算した例である。

```
[1432] nk_restriction.restriction([x*dx-1,y*dy-1],[x,y],[dx,dy],[1,0]);
-- generic_bfct_and_gr :0.001sec(0.001629sec)
generic bfct : [[1,1],[s-1,1]]
S0 : 1
B_{S0} length : 2
-- fctr(BF) + base :0.000999sec(0.0005109sec)
[[y*dy-1,(y*dy-1)*dx,-1],[[1],[0]]]
```

2.7.2 nk_restriction.restriction_ideal

nk_restriction.restriction_ideal(*Id*, *VL*, *DVL*, *W*)
:: ホロノミック D イデアル *Id* を重みベクトル *W* についての制限イデアルを返す。

Id イデアルの生成元のリスト

VL 変数のリスト

DVL 変数のリスト (*VL* に対応する微分作用素の方の変数)

W 重みベクトルを表すリスト

- *W* の要素は非負整数で、0 番目の要素から連続して正の整数が入らなければならない。(すなわち、[1,1,0,0,0] は OK だが、 [1,0,1,0,0] はダメ)
- 正の重みを持つ変数についての制限を行う。例えば、*VL* =[x,y,z], *W* =[1,1,0] であれば x,y について制限を行う。

以下は、イデアル $I = D \cdot \{x\partial_x - 1, y\partial_y - 1\}$ の x についての制限イデアルを計算した例である。

```
[1346] nk_restriction.restriction_ideal([x*dx-1,y*dy-1],[x,y],[dx,dy],[1,0]);
-- generic_bfct_and_gr :0.002sec(0.001652sec)
generic bfct : [[1,1],[s-1,1]]
S0 : 1
B_{S0} length : 2
-- fctr(BF) + base :0sec(0.000566sec)
-- restriction_ideal_internal :0.001sec(0.0007441sec)
[-1]
```

2.7.3 nk_restriction.integration

nk_restriction.integration(*Id*, *VL*, *DVL*, *W*)
:: ホロノミック D イデアル *Id* を重みベクトル *W* についての積分加群を返す。

Id イデアルの生成元のリスト

VL 変数のリスト

DVL 変数のリスト (*VL* に対応する微分作用素の方の変数)

W 重みベクトルを表すリスト

- *W* の要素は非負整数で、0 番目の要素から連続して正の整数が入らなければならない。(すなわち、[1,1,0,0,0] は OK だが、 [1,0,1,0,0] はダメ)
- 正の重みを持つ変数についての積分を行う。例えば、*VL* =[x,y,z], *W* =[1,1,0] であれば x,y について積分を行う。

以下は、イデアル $I = D \cdot \{2t\partial_x + \partial_t, t\partial_t + 2x\partial_x + 2\}$ の t についての積分イデアルを計算した例である。([SST, Ex5.5.2, Ex5.5.6])

```
[1351] nk_restriction.integration([2*t*dx+dt,2*x*dx+t*dt+2],[t,x],
[dt,dx],[1,0]);
-- generic_bfct_and_gr :0.001sec(0.001796sec)
generic bfct : [[1,1],[s,1],[s-1,1]]
```

```
S0 : 1
B_{S0} length : 2
-- fctr(BF) + base :0.001sec(0.0006731sec)
[[4*x*dx^2+6*dx,-4*t*x*dx^2-6*t*dx,2*x*dx+1,-2*t*x*dx,2*t*dx],[[1],[0]]]
```

2.7.4 nk_restriction.integration_ideal

`nk_restriction.integration_ideal`(Id, VL, DVL, W)
:: ホロノミック D イデアル Id を重みベクトル W についての積分イデアルを返す。

Id イデアルの生成元のリスト

VL 変数のリスト

DVL 変数のリスト (VL に対応する微分作用素の方の変数)

W 重みベクトルを表すリスト

- W の要素は非負整数で、0 番目の要素から連続して正の整数が入らなければならない。(すなわち、`[1,1,0,0,0]` は OK だが、 `[1,0,1,0,0]` はダメ)
- 正の重みを持つ変数についての積分を行う。例えば、VL =`[x,y,z]`, W =`[1,1,0]` であれば x,y について積分を行う。

以下は、イデアル $I = D \cdot \{2t\partial_x + \partial_t, t\partial_t + 2x\partial_x + 2\}$ の t についての積分イデアルを計算した例である。([SST, Ex5.5.2, Ex5.5.6])

```
[1431]  nk_restriction.integration_ideal([2*t*dx+dt,t*dt+2*x*dx+2],[t,x],
[dt,dx],[1,0]);
-- generic_bfct_and_gr :0.002999sec(0.002623sec)
generic bfct : [[1,1],[s,1],[s-1,1]]
S0 : 1
B_{S0} length : 2
-- fctr(BF) + base :0.001sec(0.001091sec)
-- integration_ideal_internal :0.002sec(0.001879sec)
[2*x*dx+1]
[1432]
```

2.7.5 nk_restriction (option)

`nk_restriction.restriction(... | inhomo=`n`, param=`p`, s0=`m`)`

`nk_restriction.restriction_ideal(... | inhomo=`n`, param=`p`, s0=`m`, ht=`b`, ord=`ord`)`

`nk_restriction.integration(... | inhomo=`n`, param=`p`, s0=`m`)`

`nk_restriction.integration_ideal(... | inhomo=`n`, param=`p`, s0=`m`, ht=`b`, ord=`ord`)`
:: D 加群の制限, 積分に関する関数のオプションの説明

n 0 または 1

p リスト (係数体に属する変数のリスト)

m 整数

b 0, 1, 2, 3 のいずれか

ord 重み 0 の変数に対する項順序

- n が 0 でないとき, 非斉次部分の計算を行う.

  restriction_ideal (integration_ideal) に関しては, イデアル `I` の `M` 変数についての制限 (積分) イデアル `J` と

  `J[K]-(1/IH[K][1])(IH[K][0][0][0]IH[K][0][0][1]+...+IH[K][0][M][0]IH[K][0][M][1])` $\in$ `I`

  を満たす非斉次部分を構成する情報 `IH` とのペア `[J,IH]` を出力する. 詳しい出力の見方については, 下の例やソースの `inhomo_part` のコメントを参照.

  restriction, integration に対する `inhomo` オプションは restriction_ideal, integration_ideal のサブルーチンとしての実行用なので, ユーザが明示的に使用することはない.

- *param* に指定された変数は係数体に属するものとみなされて計算が行われる. また, "generic" であることが仮定される. つまり, これらの変数に依存するような generic b-関数の根は, 最大整数根でないということである.

- *param* が指定されると, generic b-関数の計算は noro による高速アルゴリズムではなく, 消去法が用いられる. *param* に空リストを指定することで, b-関数の計算方法のフラグとしても利用できる.

- m が負でないとき, 計算を行わずに s-m を generic b-関数として制限, 積分等の計算を行う.

- *b* により, 加群のグレブナ基底計算に斉次化, trace アルゴリズムを用いるかどうか指定できる. ただし, 斉次化ありで計算できるのは Risa/Asir バージョン 20100415 以降である.

  0: 斉次化なし, trace なし

  1: 斉次化なし, trace あり

  2: 斉次化あり, trace なし (デフォルト)

  3: 斉次化あり, trace あり

- *ord* が指定されると, 出力の積分, 制限イデアルは, その項順序に関するグレブナ基底となる. 加群のグレブナ基底計算の POT 順序の tie breaker として使用されるので計算効率に大きな影響を与える可能性がある. デフォルトは 0, つまり全次数逆次書式順序である.

  このオプションは *param* と同時利用できない. (後に対応予定.)

以下は, $t^{b-1}(1-t)^{c-b-1}(1-xt)^{-a}$ の annihilator $I = D\cdot\{x(1-x)\partial_x^2+((1-t)\partial_t-(a+b+1)x+c-1)\partial_x-ab, (1-t)x\partial_x+t(1-t)\partial_t+(2-c)t+b-1, (xt-1)\partial_x+at\}$ の t についての積分イデアル J を計算し, Gauss の超幾何微分方程式を導出した例である. ([SST, Chap 1.3])

```
[1555] A=ndbf.ann_n([t,1-t,1-x*t])$
[1556] I=map(subst,A,s0,b-1,s1,c-b-1,s2,-a);
[(x^2-x)*dx^2+((t-1)*dt+(a+b+1)*x-c+1)*dx+b*a,(-t+1)*x*dx+(t^2-t)*dt+(-c+2)*t+b-1,(
[1557] J=nk_restriction.integration_ideal(I,[t,x],[dt,dx],[1,0]|inhomo=1, param=[a,
-- nd_weyl_gr :0sec(0.001875sec)
-- weyl_minipoly_by_elim :0.008001sec(0.006133sec)
-- generic_bfct_and_gr :0.008001sec(0.006181sec)
generic bfct : [[-1,1],[s,1],[s-a+c-1,1]]
```

```
S0 : 0
B_S0 length : 1
-- fctr(BF) + base :0sec(0.003848sec)
-- integration_ideal_internal :0sec(0.07707sec)
[[(x^2-x)*dx^2+((a+b+1)*x-c)*dx+b*a],[[[[dt,(-t+1)*dx]],1]]]
```

この出力は $\{(x^2-x)\partial_x^2+((a+b+1)x-c)\partial_x+ab\}-1/1\{\partial_t(-t+1)\partial_x\}\in I$ であることを意味する.

2.7.6 nk_restriction.trans_inhomo

`nk_restriction.trans_inhomo`(*P*, *INT*, *VL*, *DVL*, *W*)
:: D 加群の制限イデアル, 積分イデアルの生成元に対する非斉次部分の情報から, 任意の元に対する非斉次部分を計算する関数

P 積分イデアルの元, または制限イデアルの元

INT nk_restriction.integration_ideal(... |inhomo=1); または
nk_restriction.restriction_ideal(... |inhomo=1); の出力

VL 変数のリスト

DVL 変数のリスト (*VL* に対応する微分作用素の方の変数)

W 重みベクトルを表すリスト

- *VL*, *DVL*, *W* は *INT* の計算に用いたものをそのまま使用しなければならない.
- もし, *P* が *INT[0]* で生成される積分, 制限イデアルの元でない場合はエラーメッセージが表示される.

ChangeLog

- これらの関数は OpenXM/src/asir-contrib/packages/src/nk_restriction.rr で定義されている. nk_restriction.rr, 1.1–1.6 を見よ.
- 2010-02-05 に 3 つの option (`inhomo`, `param`, `s0`) が追加された. 1.7–1.9 を見よ.
- 2010-05 から 2010-07 にかけて 2 つの option (`ht`, `ord`) と新たな関数 trans_inhomo() が追加された. 1.10–1.13 を見よ.

2.8 その他 (未分類)

2.8.1 tk_pfn.rkn

`tk_pfn.rkn`(*F*,*X*,*Y*,*Xs*,*Ys*,*Ht*,*H*)
:: Pfaffian 方程式に対する Runge-Kutta 法

return リスト 独立変数と解の組

F, X, Y, Xs, Ys, Ht, H
F は Pfaffian 方程式の係数行列リスト. *X* は独立変数リスト. *Y* は従属変数リスト. *Xs* 独立変数の出発値リスト. *Ys* は出発時の従属変数の値リスト. *Xt* は停止する独立変数の値リスト. *H* は微少数.

- この関数は連立 Pfaffian 方程式 dY/d X[i] = F[i] Y を数値的に解く.

- 任意の holonomic system は Pfaffian 方程式に変換できる ([SST, Chap 1]). 変換には yang.rr パッケージを用いる.
- d F[i]/d X[j] + F[i] F[j] = d F[j]/d X[i] + F[j] F[i] = 0 が任意の i, j に対して成立していることが解が存在する必要十分条件である. この条件が成立しないときにこの関数を用いて解を計算してもその解は偽物である.
- X[i] が動く範囲は実数でないといけない.
- Xs[i] <= X[i] <= Xt[i] または Xt[i] <= X[i] <= Xs[i] である.
- 引数の与え方の例はソースコード (`OpenXM/lib/asir-contrib/tk_pfn.rr`) の `tk_pfn.test1`, `tk_pfn.test2` を参照.
- 下の例の出力は X=(1,3) での値が Y=(-8,2,-6) であることを意味する.
- 参考. taka_runge_kutta.rr, yang.rr

```
[1355] import("tk_pfn.rr");
[1590]  tk_pfn.test1();
Value at (3,0.1)[8.99,6,-0.2]
Value at (1,3)[-8,2,-6]
[[[1,3],-8,2,-6],
 [[1,2.9],-7.41,2,-5.8],
 --- snip ---
 [[3,0.1],8.99,6,-0.2]]
```

ChangeLog

- この関数は 2009-12 から 2010-01 にかけて最初の版が書かれた.
- `OpenXM/src/asir-contrib/packages/src/tk_pfn.rr` 1.1, 1.2
- このモジュールの前身は `tk_pf2.rr` である. これは独立変数が 2 個の場合である.

2.8.2 `tk_pfn.graph`

`tk_pfn.graph`(*Pf*,*Dom*,*Iv*,*Step*)
:: 2 変数 Pfaffian 方程式を Runge-Kutta 法で解いてグラフ表示する.

return リスト リストの要素は以下の形式 [[x の値,y の値],Y_1 の値,Y_2 の値, ...]. [x の値, y の値] は [0,0],[0.2,0],[0.4,0], ... , [0,0.2],[0.2,0.2], ... のように y の値が外側ループ, x の値が内側ループの形式で増える.

Pf, Dom, Iv, Step
Pf は Pfaffian 方程式の係数行列リスト. 独立変数は x, y で固定. *Dom* リスト. 解くべき領域. *Iv* リスト. 領域の左端での初期値. *Step* 刻み幅.

- tk_pf2.rr, mt_graph.rr を import しておく必要がある.
- この関数は連立 Pfaffian 方程式 dY/dx = Pf[0] Y, dY/dy = Pf[1] Y を数値的に解いてグラフ表示する.
- *Dom* は [[xmin,xmax],[ymin,ymax]] の形式.
- 例はソースコード (`OpenXM/lib/asir-contrib/tk_pfn.rr`) の `tk_pfn.testgraph1()`, `tk_pfn.testgraph2()` を参照.
- option としては fit=1 がある. Z 軸を適宜調整する.
- *Dom* の端はグラフ表示の時に一部カットされるので注意.

```
[1355] import("tk_pf2.rr"); import("mt_graph.rr"); import("tk_pfn.rr");
[1590] tk_pfn.testgraph1();
```

ここで testgraph1() は以下のとおり.

```
def testgraph1()
  /* tk_bess2.bess2pf(1/2); */
  Pf=  [[[ 0, (1)/(x), 0 ],
         [ -x, (2*x^2+1)/(x), -2*x ],
         [ -y, 0, 0 ]],
        [[ 0, 0, (1)/(y) ],
         [ -x, 0, 0 ],
         [ -x, (1/2)/(x), (-1/2)/(y) ]]];
  /* tk_bess2.bess2Iv(1/2,[0.5,1.5]); */
  Iv = [0.105994,-0.651603,-0.760628];
  Dom=[[0.5,1.5],[1.5,9]];
  Step = 0.5;
  return tk_pfn.graph(Pf,Dom,Iv,Step | fit=1);
```

ChangeLog

- この関数は 2010-08 に最初の版が書かれた.
- OpenXM/src/asir-contrib/packages/src/tk_pfn.rr 1.8

2.8.3 tk_rk.runge_kutta_4

tk_rk.runge_kutta_4(*Eq*,*X*,*Y*,*X0*,*Y0*,*Terminal*,*Step*)
:: 4 次の Runge-Kutta 法による微分方程式の数値近似解

return リスト リストの要素は以下の形式 [*X* の値,Y_1 の値,Y_2 の値, ...]. *X* の値は減っていく. よってリストの先頭が *Terminal* 付近での *Y* の値.

Eq, X, Y, Step
Eq は 方程式の右辺. リスト. Y[0]'=Eq[0], Y[1]'=Eq[1], ... である. *X* 独立変数名. *Y* リスト. 従属変数のリスト. *Step* 刻み幅.

X0, Y0, Terminal
X0 出発点の *X* の値. *Y0* 出発点での *Y* の初期値. *Terminal X* の終着点.

- taka_runge_kutta.rr を import しておく必要がある.
- この関数は連立常微分方程式 Y[0]'=Eq[0], Y[1]'=Eq[1], ... を数値的に解く.
- 例はソースコード (OpenXM/lib/asir-contrib/src/taka_runge_kutta.rr) の tk_rk.test4() を参照.

```
[1355] import("taka_runge_kutta.rr");
[1590] tk_rk.test4();
```

ここで test4() は以下のとおり. 振動の方程式, y0'=y1, y1'=-y0 (y0''+y0=0). 答は y0=cos(x)
taka_plot_auto は下方向で y が正.

```
def test4()
  A=runge_kutta_4([y1,-y0],x,[y0,y1],0,[1,0],3.14*2,0.1);
  taka_plot_auto(A);
  return(A);
```

ChangeLog

- この関数は 2000 年代の前半に最初の版が書かれた. 2010 年 Pfaffian の数値解析の為に再度整備
- `OpenXM/src/asir-contrib/packages/src/taka_runge_kutta.rr` 1.17

2.8.4 tk_rk.runge_kutta_4_linear

`tk_rk.runge_kutta_4_linear`(*P*,*X*,*Y*,*X0*,*Y0*,*Terminal*,*Step*)
:: 4 次の Runge-Kutta 法による微分方程式の数値近似解. 線形方程式専用.

return リスト リストの要素は以下の形式 [*X* の値,Y_1 の値,Y_2 の値, ...]. *X* の値は減っていく. よってリストの先頭が *Terminal* 付近での *Y* の値.

P, X, Y, Step
P は 方程式の右辺. リスト. Y'=P Y である. 従属変数 *Y* は不要. *X* 独立変数名. *Y* リスト. 従属変数のリスト. 従属変数は自動生成される. 使われていないので空リストでよい. *Step* 刻み幅.

X0, Y0, Terminal
X0 出発点の *X* の値. *Y0* 出発点での *Y* の初期値. *Terminal X* の終着点.

- taka_runge_kutta.rr を import しておく必要がある.

```
[1355] import("taka_runge_kutta.rr");
[1590] A=tk_rk.runge_kutta_4_linear([[0,1],[-1,0]],x,[ ], 0, [1,0], 3.14*2, 0.1);
[1591] taka_plot_auto(A);
```

振動の方程式, `y0'=y1, y1'=-y0 (y0''+y0=0)`. 答は `y0=cos(x)` を解いている. `taka_plot_auto` は下方向で `y` が正.

ChangeLog

- 2010 年 Pfaffian の数値解析の為に再整備.
- `OpenXM/src/asir-contrib/packages/src/taka_runge_kutta.rr` 1.17

2.8.5 fj_simp.simplify

`fj_simp.simplify`(*arg1*)
:: arg1 を簡単化する.

return 多項式, 有理式 または quote

arg1 多項式 または 有理式

- 現在のバージョンでは fj_simp.simplify でなく, 単に simplify とよぶ.

- この関数は Joel S. Cohen, Computer Algebra and Symbolic Computation, http://web.cs.du.edu/~jscohen/MathematicalMethods/index.htm に記述されている Automatic simplification algorithm と B.F.Caviness, R.J.Fateman, Simplification of Radical Expressions (1976) に記述されている radcan アルゴリズムの実装である.
- 複素多値関数としては (x*y)^a = x^a*x^b は一般には成立しないので, 結果を複素関数に使うときは注意が必要である. (x^a = exp(a*log(x)) なので)

```
load("fj_simplify.rr");
[1434] simplify((x^(1/2))^3);
((x)^(3/2))
[1435] simplify((2^(1/2))^2);
2
[1436] simplify((2+2^(1/2))^3);
14*((2)^(1/2))+20
[1437] simplify(exp(x)*exp(-x+y));
((@e)^(y))
```

参照 ⟨undefined⟩ [quote], page ⟨undefined⟩

ChangeLog

- 将来的には module 化して fj_simp module に含める予定. poly_simplify から fj_simp.simplify を呼ぶ.
- Todo: exp 以外の特殊関数についての simplification の機能.
- この関数は 2010.01 に M.Fujimoto により最初の版が書かれた. OpenXM/src/asir-contrib/packages/src/fj_simplify.rr

2.8.6 xyz_pqr, syz_stu

`xyz_pqr(`*arg1*`,`*arg2*`[,`*flag*`])`
`xyz_stu(`*arg1*`,`*arg2*`)`
:: xyz に関する操作.

return 整数

arg1, arg2 整数

flag 0 または 1

- この項目は新しい関数の説明を書くためのテンプレートである. 消すな.
- `xyz_pqr()` は, *arg1*, *arg2* を pqr する.
- *flag* が 0 でないとき, モジュラ計算を行う.
- `xyz_stu()` は stu アルゴリズムを用いる.

```
[219] xyz_pqr(1,2);
3
[220] xyz_pqr(1,2,1);
3
0
[221] xyz_stu(1,2);
3
```

参照 ⟨undefined⟩ [xyz_abc], page ⟨undefined⟩

ChangeLog

- この関数は 2004-3-1 から 2004-3-14 にかけて アルゴリズム xyz (論文 http://www.afo.org/xyz.pdf) を用いて書き直された. 変更をうけたソースコードは xxxyy.rr, ppp.c である.
- この関数は 2000 頃にはじめてのバージョンが書かれた. ソースは ppp.c である.

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents