数理解析研究所講究録
第1568巻 2007年 40-47

斉次化と inter-reduction によるグレブナー基底計算の効率化について

野呂 正行
MASAYUKI NORO
神戸大学理学部*

1 Risa/Asir における Buchberger アルゴリズムの実装

Risa/Asir の dp_gr_main および nd_gr_trace においては, 計算効率を上げるために次のような工夫がなされている.

- 斉次化 trace アルゴリズム

  Buchberger アルゴリズムは, 途中で選ばれる S 多項式によらずに停止し, グレブナー基底を与えるが, 実際の計算効率は, 選ばれる S 多項式に大きく依存する. このため, いわゆる selection strategy が重要となる. 非効率性は, 標数 0 の体上では係数膨張として現れることが多い. 代表的なものに sugar strategy があり, これは斉次化を simulate していると考えられるが, 失敗する場合も多い. すなわち非斉次な入力多項式集合に対し sugar strategy を適用した場合に係数膨張が起きる場合にも, 実際に斉次化することにより, 係数膨張なしで計算できる場合がある. 実際に斉次化すると, 0 reduction が増える場合があるが, 有限体上で 0 に正規化される S 多項式を無視することで, グレブナー基底の候補が得られる. この候補を非斉次化し, 生じた冗長元を除いた後, inter-reduction したものをチェックすることにより, グレブナー基底を得る方法が, Risa/Asir に実装されている斉次化 trace アルゴリズムである.

- Content 除去

  得られた正規形の content を取り除くことが効率向上にとって重要である. さらに, 各正規形計算の途中に現れる中間剰余の段階で既に 1 でない content を持つ場合があり, これを適宜除くことで, 更に効率が上がる. Risa/Asir においては, 中間剰余の先頭係数のサイズ (bit 長) が, ある時点での大きさの指定された大きさ (デフォルトは 2) 倍以上になったとき, 強制的に content を外している. 整数 GCD 計算は高コストなので, 頻度を上げすぎると overhead の方が大きくなるが, 発見的に決めた 2 という値を使って content 除去をすることで, 多くの入力に対して効率向上が認められる.

*noro@math.kobe-u.ac.jp

- Inter-reduction

  斉次な多項式集合に対しては, S 多項式の集合を, 同じ全次数ごとにまとめて, 最低全次数のものから処理するのが自然である. この場合, ある時点での中間基底から作られる最低全次数 (d とする) の S 多項式を全部処理したあとに生成される S 多項式の全次数は全て d より真に大きい. よって, この時点でグレブナー基底中の全次数 d 以下の元は全て生成されている. 後で詳しく述べるが, 生成された d 次の基底間で inter-reduction を行って, 各基底を正規化されたものに置き換えてもアルゴリズムは正しい.

以上のような工夫により, 種々の問題に対してグレブナー基底の計算がある程度満足すべき効率で計算できたが, 係数膨張に関しては不満が残る.

例 1
McKay において, $d = 16$ における各正規形計算が, 全体計算時間の大部分を占めるが, それは, $d = 16$ における大量の S 多項式を処理するうちに, そこで生ずる中間基底の係数が大変大きくなり, その後の正規形計算に時間がかかるようになるせいである. しかし, $d = 16$ の処理が終ったあと, inter-reduction を行うと, 各基底から大きな content がとれる. 最終的に得られる基底の係数は大変小さい.

この例から, 全ての中間基底が得られた後の inter-reduction が, 基底の係数を小さくすると思い込んでいたが, これは間違いであった. 中間基底がいくつか生成された後で, 例に inter-reduction してみると, それなりの大きさの content が得られることが分かったのである.

2 Inter-reduction

本節では, 入力多項式集合 F が斉次多項式からなる場合を考える. この場合, 全ての S 多項式は斉次であり, またそれらから生成される基底もやはり斉次である. 以下, $d-1$ 次までの S 多項式の処理が終った時点で残っている d 次の S 多項式の集合を S_d, $d-1$ 次までに得られた中間基底を $G_{<d}$, S_d から得られる基底を G_d とする (G_d は inter-reduction する前は, S_d の元の処理の順序に依存する). G_d の各元は, それまでに得られた中間基底に関して正規形になっており, また, G_d の頭項は, G_d の他の元の頭項では割れない. よって, G_d と, 既に得られた中間基底から生成される S 多項式の全次数は全て d より真に大きい.

さて, 今考えている場合に, Buchberger アルゴリズムを次のように変更する:

S_d を処理中に, 得られた基底で, 既に得られた基底を正規化し, もとの基底を, 正規化された基底と置き換える. (常にするとは限らない.)

このような変更のもとで, 次が成り立つ.

命題 1
S_d の処理を全て行い, 得られた d 次の基底を inter-reduction したものを G'_d とおく. このとき, $G'_d \subset \langle F \rangle$, $\mathbf{HT}(G'_d) = \mathbf{HT}(G_d)$ で, G'_d は, $\langle F \rangle$ のグレブナー基底の元のうち, 全次数が d であるものの全体と一致する.

証明 G_d の各元は, S_d の元から, $G_{<d}$ の元の単項式倍である R_d の元 (M とする), あらかじめ G に入っている d 次の元および既に生成された d 次の基底 (B とする), の定数倍を引いて作られ

る. 詳しく言えば, B の元は, S_d のどれかの項を消去するようなものに限られる. $S_d \cup M \cup B$ で K 上生成される線形空間の基底が $G \cap R_d$ であるが, 途中で上で述べたような基底の取り換えを行っても, $G'_d \cup M \cup B$ で生成される線形空間は $G'_d \cup M \cup B$ で生成される線形空間と変わらない. よって, 命題が成り立つ.

この命題により, いくつか d 次の基底が生成されるごとに, 既に生成された d 次の基底の間で inter-reduction を行い, 各基底をその結果に置き換えて構わないことになる. この方法は, 行列のはき出しにおいて, 途中で上三角部分のはき出しを行うことに相当する. よって, 一般には手間が増えるだけのように見えるが, 少なくとも有理数体上の場合, この中間はき出しを行うと, 得られた基底から content がとれる場合があることが分かった. 問題は頻度であるが, もともと発見的な手法であるから, 実験により最適と思える値を探した. 現状では基底が 6 つ生成されるごとに, 中間はき出しを行っている.

注意 1
最近, CoCoA のソースコードを見る機会があり, やはり inter-reduction が頻繁に行われているように見えた. 開発者である J. Abbott, M. Caboara に確認したところ, そのとおりであるとのことであった. ただし, 頻度については確認していない. 頻繁な inter-reduction については, Risa/Asir は CoCoA と独立に導入したものの, 開発履歴によれば, 先に行ったのは CoCoA のようである.

3 代数体, 有理関数体上でのグレブナー基底計算

3.1 代数体係数の場合

前節の inter-reduction は, 係数膨張を生じるような他の体についても有効である. 例えば, K を $\mathbf{Q}$ の逐次代数拡大とし, K 上でグレブナー基底計算を行うことを考える. この場合, 拡大の生成元の定義多項式を入力イデアルに追加して, $\mathbf{Q}$ 上で計算する方法と, K 上で直接計算する方法があるが, [2] では, 正規形を常にモニック化することで, 正規形計算自体は前者の方法で行うものの, 生成される正規形は後者で生成されるものと同じになるような方法を提案した. この方法により効率が向上する例もあるが, 同じ sugar をもつ S 多項式が多い場合, かえって効率が下がる場合があった. この場合にも inter-reduction を適用してみると, 効果が見られる場合がある.

3.2 有理関数体係数の場合

3.2.1 dp_gr_main の実装

有理関数体上でのグレブナー基底計算は 旧実装である dp_gr_main でのみグレブナー基底計算が可能であったが, 最近 nd_gr, nd_gr_trace でも可能となった. dp_gr_main の実装は以下の通りである.

- 斉次化 trace アルゴリズム

  計算中の係数は全て $\mathbf{Z}[x_1, \ldots, x_m]$ の元として計算される. trace アルゴリズムのための準同型写像 $\phi : \mathbf{Z}[t_1, \ldots, t_m] \to \mathbf{F}_p$ は, 素数 p およびランダムに選んだ代入点 $(a_1, \ldots, a_m) \in \mathbf{Z}^m$ を選び

  $$\phi(f(x_1, \ldots, x_n, t_1, \ldots, t_m)) = f(x_1, \ldots, x_n, a_1, \ldots, a_m) \bmod p$$

を用いる. $\phi(S(f,g))$ の $\phi(G)$ による剰余が 0 なら, $S(f,g)$ の K 上での計算をスキップするのである. もちろん, 代入点が基底の頭係数を 0 にする場合には, それらをとり直す.

- content 除去

  効率上最も問題になるのはこれである. これは, 複数の多変数多項式の GCD 計算であるが, dp_gr_main においては, 現れる係数多項式を 2 組に分け, それぞれ乱数倍して足して 2 つの多項式を作り, その GCD で係数多項式を割って見て, 失敗したら乱数をとり直す, という方法をとっている. これにより GCD は実質 1 回ですむが, GCD (EZ-GCD) が重い演算のため, 正規形に対してのみ content 除去を行っている.

- 途中での inter-reduction

  もちろん実装していない.

3.2.2 content 除去

nd_gr 形の関数を有理関数係数対応にするにあたり, content 除去の効率化を何通りか試みた. 正規形計算で生じる content の大部分が正規形計算で用いた基底の頭係数から来るであろうと推測される. これを, 部分終結式のようにシステマティックにできないかという試みが Mandache により行われたが, うまくいかなかったと報告されているので, ここでは割れるだけ割り, 残りを GCD にかけるという単純な方法を採った. 前処理に使う因子としては, 次のようなものを試した.

1. 基底の頭係数そのものを使う

   これは手間いらずだが, content が取りきれない場合が多い.

2. 基底の頭係数を既約分解して, 因子を個別に使う

   これが最も完全と思われるが, 因数分解のコストが大きすぎる.

3. これまで得られた因子で割り切れる場合には割って得た商を使う

   これはかなりいい加減に見える方法だが, この方法で多くの場合にほとんどの content が取れる.

以上により, 現状では 3. の方法を用いている. 正規形計算の途中で content 除去を行うタイミングとしては, 有理数係数と同様, 先頭係数の項数が所定の大きさ倍を越えた場合としている. デフォルトは 2 倍としているが, GCD の計算が重いので, 少し頻度を下げるべきかもしれない.

3.2.3 Inter-reduction

nd_gr 系の実装において初めて有理関数体上でも頻繁に inter-reductrion を行うことを試みた. 頻度については, Q 上の場合と同様に, デフォルトでは基底が 6 つ生成されたら inter-reduction するが, これも最適な値を決めるには多くの実験が必要である.

3.2.4 有理関数体上のグレブナー基底の, 消去順序による計算

よく知られているように, 有理関数体上のグレブナー基底は, 有理関数体の変数を本来の変数の後ろにおいた消去順序でのグレブナー基底として計算できる.

定理 2
$f_i \in K[X,T]$ (i=1,...,l, $X = x_1,\ldots,x_n$, $T = t_1,\ldots,t_m$) とする. $<_X$, $<_T$ をそれぞれ $K[X]$, $K[T]$ における項順序とし, G, $\langle f_1,\ldots,f_l\rangle \subset K[X,T]$ の, 積順序 $<_X \times <_T$ に関するグレブナー基底を G とするとき, G は $K(T)[X]$ の部分集合として, $\langle f_1,\ldots,f_l\rangle$ の $<_X$ に関するグレブナー基底となる.

この方法と, 有理関数体上で直接計算する方法のどちらが効率よくグレブナー基底を計算できるかは, 一般には判定がむずかしい. 後に実行例でみるように入力に依存する. ただし, 消去順序を trace アルゴリズムと合わせて用いる場合, 候補チェックまで行ってしまうのは得策ではない. すなわち, 得られた基底 G を有理関数体上の基底とみなすとき, 冗長な元が現れ, また, K 上で reduced でも $K(T)$ 上では reduced でない可能性がある. よって, 冗長性を省き, $K(T)$ 上での inter-reduction を行ってから, チェックする方がよい場合がある.

4 実験

前節までに述べた方法を実装し, 種々の例で計算時間を計測した結果を示す. 例は, 主として [4] および [3] から取った. 実行環境は, Linux/Intel Xeon 3.4GHz, 計算時間は GC 時間を含み, 単位は秒である.

4.1 Q 上の場合

表 1 において, tl は単なる trace アルゴリズム, tl+int は inter-reduction つきの trace アルゴリズムで, 候補生成時間を表す. check は候補チェックの時間で, 両方に共通である.

問題によってはかえって遅くなっているものもあるが, C_8, *mckay* のように 3 倍程度高速化したものもある.

4.2 代数体上の場合

[2] と重複するので例を一つだけ挙げる. 表 2 に, C_7 で, c_7 に $t^2+5t+1=0$ の根 α を代入した場合の比較を示す. 上段は, 消去順序での計算, 下段は代数体上での計算時間の比較である. この例は, 前者の方が高速である例だが, inter-reduction の効果は双方に現れていることが分かる.

4.3 Q 上の有理関数体上の場合

4.3.1 消去順序による計算との比較

表 3 は, [4] に入っている Chou [1] にある問題 (Geometry.Chou.N) についてグレブナー基底計算を行った結果である. 各問題は x_i, u_j という変数を含むが, u_i を係数体の不定元とみなして,

	tl+int	tl	check		tl+int	tl	check
*assur*44	3	5	8	*ilias_k_2*	49	121	88
*cohn*3	24	30	41	*ilias_k_3*	89	234	247
C_7	22	24	12	*jcf*26	7	14	40
C_8	1602	4768	317	*kin*1	4	7	63
dl	29	15	45	*kotsireas*	3	3	3
*eco*9	2	2	4	*noon*8	363	149	185
*eco*10	15	13	50	*pinchon*1	3	3	4
*eco*11	171	143	909	*rbpl*	7	7	2397
*eco*12	2435	2112	7401	*rbpl*24	3	4	19
*extcyc*6	112	166	54	*redcyc*7	18	19	2
*f*855	51	39	23	*redeco*10	4	3	12
*fabrice*24	3	4	19	*redeco*11	34	23	121
*filter*9	25	12	1	*redeco*12	354	197	1386
*hairer*2	79	81	39	*reimer*6	76	48	12
*i*1	9	11	203	*virasoro*	2	2	11
*ilias*13	438	937	357	*mckay*	3817	11987	1

表 1: 有理数体上の斉次化 trace アルゴリズムの比較

	tl+int	tl	check
$C_{7,c_7=\omega}/\mathbf{Q}(\omega)$	198	1476	0
$C_{7,c_7=\omega}/\mathbf{Q}$	119	221	0

表 2: 代数体上の斉次化 trace アルゴリズムの比較

grlex のグレブナー基底計算を行った. rat(i) は有理関数体上での直接計算 (i は, i 個基底が生成される毎に inter-reduction, rat(−) は S_d 処理後のみ inter-reduction), elim は u_i を後ろに回した, grlex の積順序でのグレブナー基底計算による計算である. 後者では, 候補生成までを $\mathbf{Q}$ 上で行い, inter-reduction およびチェックを $\mathbf{Q}(u)$ 上で行った時間である. x_i の順序は i が大きい程上とした. 表以外の例については, 計算時間の合計のみ示した. これらの例では, 有理関数体上で計算した方が明らかによい. 消去順序で時間がかかるのは, 結果的には不要な基底が大量に生成されるからである.

しかし, 逆の場合も存在する. C_7 から, 最高次の式を取り除いたものを, c_1, c_2 をパラメタとしてグレブナー基底計算すると, 消去順序によりグレブナー基底候補を求めるのに, 26 sec, 有理関数体に移って冗長元を除き, inter-reduce するのに 2850 sec, チェックは 0.01 秒であるが, 有理関数体上で計算すると 12 時間たっても計算が終らない [1]. この例の場合, GCD 計算の困難さによるもので, 有理関数体上で効率が落ちる典型的な例となっている.

N	rat(1)	rat(6)	rat(−)	elim
106_1	4.7	4.5	4.5	11
310_1	866	1035	1357	$> 1h$
329_1	2.5	4.5	8.0	1520
337_1	2.0	1.2	1.1	73
353_1	1.3	0.7	0.7	560
393_1	81	72	71	200
401_1	1.8	1.8	1.7	282
459_1	0.3	0.2	0.2	91
460_1	0.1	0.1	0.1	26
その他	12.1	11.4	10.4	55

表 3: 有理関数体上のグレブナー基底計算

4.3.2 heuristic な content 除去の効果

Chou の 310_1 について, いくつかの場合における content 除去の様子を調べてみた. 表 4 で, rat(i) は前述のもの, rat(-)-plain は, 正規形に対してのみ content 除去を行うとしたものである. #cont は content 除去の回数, #nontrivial は content が 1 でない回数, #hc は, 頭係数による 1 でない content 除去の回数, #GCD は, GCD による content 除去の回数を示す. inter-reduction が多い程無駄に GCD を計算しているように見えるが, 少なくともこの例では, inter-reduction がこまめに係数を小さくし, 全体の計算時間を下げる効果を与えている. ただし, GCD が本当に必要となる場合はどの計算でも少なく, 頭係数による heuristic な 方法で大部分の content が除去できている. よって, GCD 計算の頻度を下げることで, より高速化できる可能性がある.

[1] 別のマシンで $1.2 \times 10^5 sec$ で計算できたが, これは, 次節の考察により, 正規形計算の途中の content 除去を heuristic のみにした方法であった

	#cont	#nontrivial	#hc	#GCD
rat(1)	549	73	69	7
rat(6)	267	71	67	7
rat(−)-plain	93	43	39	10

表 4: content 除去の内訳

参 考 文 献

[1] Chou, S.-C., Mechanical Geometry Theorem Proving, D. Reidel Publishing Company, 1988.

[2] Noro, M., An efficient implementation for computing groebner bases over algebraic number fields, Proceedings of Second International Congress on Mathematical Software - ICMS2006,LNCS4151, 99-109,2006.

[3] `http://invo.jinr.ru/`.

[4] `http://www.symbolicdata.org`