わかったつもりになる Gröbner基底

石井 大海

自己紹介

- 石井大海（a.k.a. @mr_konn）
  - Twitter: @mr_konn
  - Blog: http://blog.konn-san.com/
  - Web: http://konn-san.com/
- Haskell lover
- 早稲田大学数学科四年
  - 本当はロジックの人間の筈が今年は縁あって代数幾何（特に Gröbner 基底） の研究室に
  - computational-algebra の作者

Table of Contents

- Gröbner 基底の簡単な導入
- Gröbner 基底の計算法
  - Buchberger アルゴリズム
  - 最適化手法：syzygy, sugar strategy
- Haskell の型レベルプログラミングの現在

Table of Contents

- Gröbner 基底の簡単な導入
- Gröbner 基底の計算法
  - Buchberger アルゴリズム
  - 最適化手法：syzygy, sugar strategy
- Haskell の型レベルプログラミングの現在

Gröbner 基底 の簡単な導入

Gröbner 基底の応用

- 高次連立方程式の文字消去
- 初等幾何の自動証明
- 統計にも何か応用があるらしい
- ロボティクス
- 計算機代数、代数幾何学

Gröbner 基底

Gröbner 基底

- “グレブナーきてい“ と読む

Gröbner 基底

- “グレブナーきてい” と読む
- 1960s: 廣中平祐、B. Buchberger らが独立に発見

Gröbner 基底

- “グレブナーきてい” と読む
- 1960s: 廣中平祐、B. Buchberger らが独立に発見
- Gröbner は Buchberger の師匠の名前

Gröbner 基底

- “グレブナーきてい” と読む
- 1960s: 廣中平祐、B. Buchberger らが独立に発見
- Gröbner は Buchberger の師匠の名前
- 廣中は特異点解消のために考えた

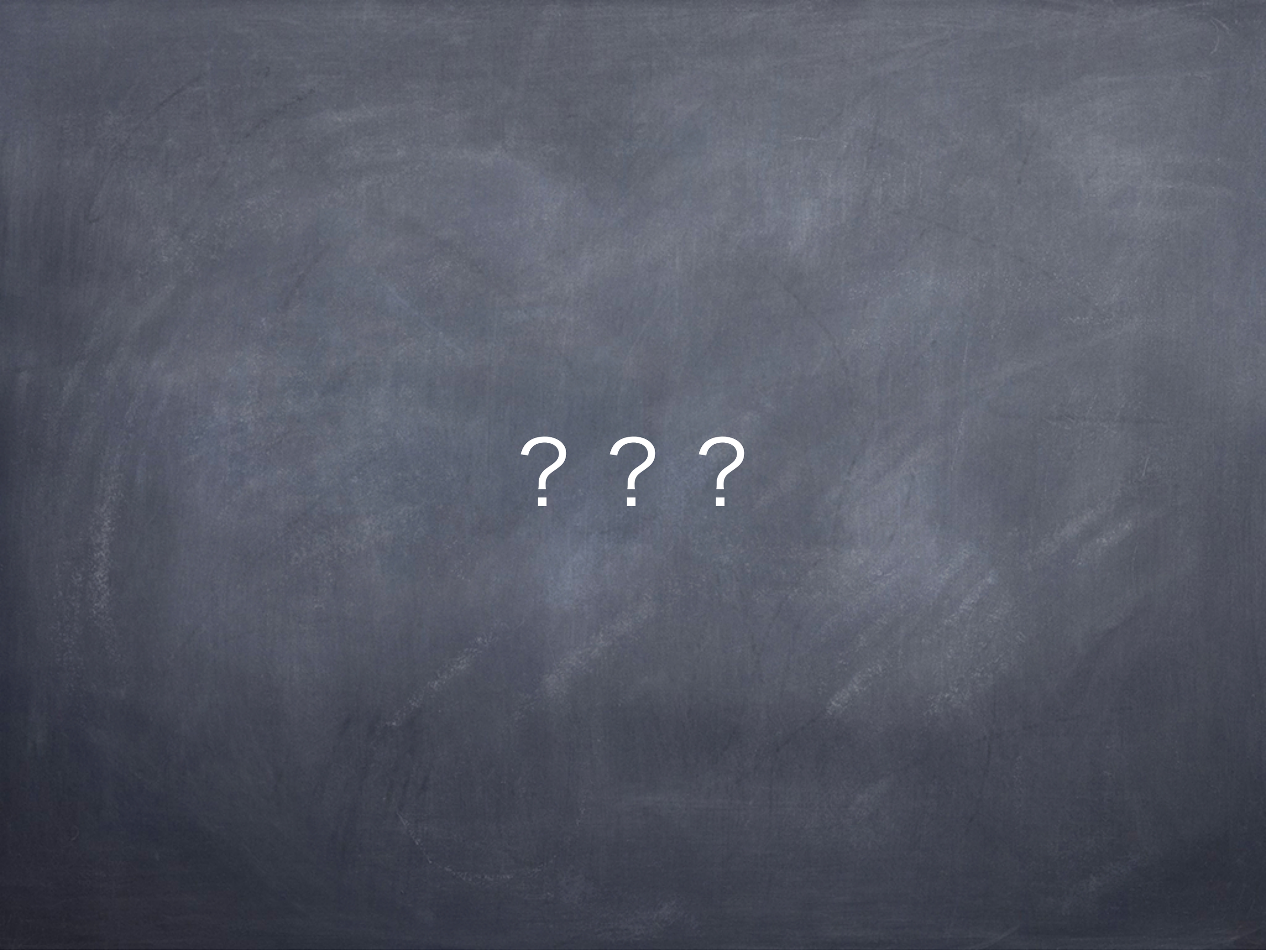
Gröbner 基底

- “グレブナーきてい” と読む
- 1960s: 廣中平祐、B. Buchberger らが独立に発見
  - Gröbner は Buchberger の師匠の名前
  - 廣中は特異点解消のために考えた
- 技術の進歩で計算出来るようになり応用が出る

Gröbner 基底

- “グレブナーきてい” と読む
- 1960s: 廣中平祐、B. Buchberger らが独立に発見
  - Gröbner は Buchberger の師匠の名前
  - 廣中は特異点解消のために考えた
- 技術の進歩で計算出来るようになり応用が出る
- 体上多項式環のイデアルの性質のよい生成元

体上多項式環のイデアル
の性質のよい生成元



体上多項式環のイデアル
の性質のよい生成元

体上多項式環のイデアル
の性質のよい生成元

体

- たい（≠からだ）と読む
- 足し算、引き算、掛け算、割り算（0以外）が出きるような構造
- 例：有理数体 $\mathbb{Q}$ とか複素数体 $\mathbb{C}$ とか
- 例：整数全体 $\mathbb{Z}$ は体でない（1/2は整数じゃない）
- 以下では小文字の k で体を表す

体上多項式環のイデアル
の性質のよい生成元

体上の多項式環

- $k[X_1, \ldots, X_n] = \{k$ を係数とする n-変数多項式全体$\}$
  - 変数の数が自明な時は $k[\mathbb{X}] = k[X_1, \ldots, X_n]$ と書く
    - 自然に足し算、引き算、掛け算が定義できる（割り算は出来るとは限らない）
- 例：$X_1^2 + 2\,X_1X_2 + X_2^2 \in k[X_1, X_2]$, $y^2 - x \in k[x, y]$

体上多項式環のイデアル
の性質のよい生成元

イデアル

定義

- 環 R の部分集合 I が イデアル であるとは
  1. 任意の $x, y \in I$ について $x - y \in I$
  2. 任意の $c \in R, x \in I$ について $cx \in I$
- 倍数概念の或る種の一般化になっている
  - イデアル I に属する $\Leftrightarrow$ I の元で割れる
  - a が b の倍数 $\Leftrightarrow \langle a \rangle \subseteq \langle b \rangle$

なぜイデアル？

なぜイデアル？

- イデアルは幾何的にはその零点の成す図形と対応する
  （正確には根基イデアルが対応）

なぜイデアル？

- イデアルは幾何的にはその零点の成す図形と対応する
  （正確には根基イデアルが対応）
  - $\langle y - x^2 \rangle$：放物線

なぜイデアル？

- イデアルは幾何的にはその零点の成す図形と対応する
  （正確には根基イデアルが対応）
  - $\langle y - x^2 \rangle$：放物線
  - $\langle y^2 + x^2 - 1, y + 2x - 1 \rangle$：円と直線の交点

なぜイデアル？

- イデアルは幾何的にはその零点の成す図形と対応する
  （正確には根基イデアルが対応）
  - $\langle y - x^2 \rangle$：放物線
  - $\langle y^2 + x^2 - 1, y + 2x - 1 \rangle$：円と直線の交点
- 交点 ⇔ 連立方程式の解

なぜイデアル？

- イデアルは幾何的にはその零点の成す図形と対応する
  （正確には根基イデアルが対応）
  - $\langle y - x^2 \rangle$：放物線
  - $\langle y^2 + x^2 - 1, y + 2x - 1 \rangle$：円と直線の交点
- 交点 ⇔ 連立方程式の解
  - イデアルは連立された関係式とも見ることが出来る

なぜイデアル？

- イデアルは幾何的にはその零点の成す図形と対応する
  （正確には根基イデアルが対応）
  - $\langle y - x^2 \rangle$：放物線
  - $\langle y^2 + x^2 - 1, y + 2x - 1 \rangle$：円と直線の交点
- 交点 ⇔ 連立方程式の解
  - イデアルは連立された関係式とも見ることが出来る
  - Gröbner 基底を使うと色んな図形・連立方程式に関する計算が簡単に出来るようになる

体上多項式環のイデアル
の性質のよい生成元

生成元

定義

$f_1, \ldots, f_r \in k[\mathbb{X}]$ について、

$\langle f_1,\ldots,f_r\rangle := \{a_1 f_1 + \ldots + a_r f_r \mid a_i \in k[\mathbb{X}]\ (i = 1, \ldots r)\}$

を $f_1,\ldots,f_r$ により生成されるイデアルという。

$\mathrm{I} = \langle f_1,\ldots,f_r\rangle$ の時、$f_1,\ldots,f_r$ を I の基底、または生成元と呼ぶ。

- 上の $\langle f_1,\ldots,f_r\rangle$ はさっきのイデアルの定義を満たす。
- 逆に、体上多項式環のイデアルは、適当な $f_1,\ldots,f_r$ により、すべてこの形で書ける（Hilbert の基底定理）

Hilbert の基底定理

定理

Noether 環上の多項式環は Noether 環である

Hilbert の基底定理

定理

Noether 環上の多項式環は Noether 環である

- Noether 環とは「任意のイデアルが有限個の生成元で書ける（＝有限生成）」という条件を満たす環

Hilbert の基底定理

定理

Noether 環上の多項式環は Noether 環である

- Noether 環とは「任意のイデアルが有限個の生成元で書ける（＝有限生成）」という条件を満たす環
- 計算機で扱うには、多項式環のイデアルは有限個の多項式の組として定義すれば十分

Hilbert の基底定理

定理

Noether 環上の多項式環は Noether 環である

- Noether 環とは「任意のイデアルが有限個の生成元で書ける（＝有限生成）」という条件を満たす環
- 計算機で扱うには、多項式環のイデアルは有限個の多項式の組として定義すれば十分
- 証明はむずかしいが、体の場合については Gröbner 基底を使うと簡単に示せる。

  （証明：Gröbner 基底を取ればよい■）

体上多項式環のイデアル
の性質のよい生成元

よい性質？

問題 1

$$f_1 = x^3 + x^2 + x + 1$$
$$f_2 = x^2 - x - 2$$
$$I = \langle f_1, f_2 \rangle \subseteq k[x]$$

とすると、$g = x^3 - 1$, $h = x^2 + 1$ はそれぞれ I に属すか？

- 左の問題を考える
- f_1 と f_2 の“最大公約式”$f = \mathrm{GCD}(f_1, f_2)$ がわかればよい
- $I = \langle f \rangle$ となるので、あとは g, h を f で割り切れるか見る

最大公約式の求め方

- Euclid の互除法を使う
  - 整数だけでなく、割り算原理が成立する任意の環で使える
- 整数の場合：ある元が $\langle a, b \rangle$ に属する
  $\Leftrightarrow$ a, b で割れる
  $\Leftrightarrow$ a, b の最大公約数で割れる
- 同様のことが一変数多項式環でも成立

Euclid の互除法

- 入力：f, g

  1. f を g で割り算した余りを r_0とする
  2. g を r_0 で割り算した余りを r_1 とする
  3. r_0 を r_1 で割り算した余りを r_2 とする
  4. 以下ゼロになるまで繰り返し

- 出力：ゼロでない最後の r_{n}

一変数多項式の割り算

- 先程の問題で Euclidの互除法を使うときの最初のステップ
- 降冪の順に並べる
- 大きい項との帳尻を合わせていく

$$
\begin{array}{r}
x\ \ +2 \\
\hline
x^2-x-2\ \big)\ x^3+x^2\ \ +x+1 \\
x^3-x^2-2x \\
\hline
2x^2+3x+1 \\
2x^2-2x-4 \\
\hline
5x+5
\end{array}
$$

これを多変数に

一般化できないか？

多変数の場合

多変数の場合

問題 2

$$f_1 = X^2Y - 1$$
$$f_2 = X^3 - Y^2 - X$$
$$I = \langle f_1, f_2 \rangle \subseteq k[X, Y]$$

とすると、

$$g = X^2Y^2 - X^3Y - XY - 1$$

は I に属すか？

多変数の場合

問題 2

$$f_1 = X^2Y - 1$$
$$f_2 = X^3 - Y^2 - X$$
$$I = \langle f_1, f_2 \rangle \subseteq k[X, Y]$$

とすると、

$$g = X^2Y^2 - X^3Y - XY - 1$$

は I に属すか？

- 二変数の場合を考える

多変数の場合

問題 2

$f_1 = X^2Y - 1$
$f_2 = X^3 - Y^2 - X$
$I = \langle f_1, f_2 \rangle \subseteq k[X, Y]$

とすると、

$g = X^2Y^2 - X^3Y - XY - 1$

は I に属すか？

- 二変数の場合を考える
- f_1 と f_2 の“最大公約式”のような物を考えられるか？

多変数の場合

問題 2

$f_1 = X^2Y - 1$
$f_2 = X^3 - Y^2 - X$
$I = \langle f_1, f_2 \rangle \subseteq k[X, Y]$

とすると、

$g = X^2Y^2 - X^3Y - XY - 1$

は I に属すか？

- 二変数の場合を考える
- f_1 と f_2 の“最大公約式”のような物を考えられるか？

➡ 出来ない！

最大公約式は取れない

最大公約式は取れない

- 最大公約式が必ず取れるなら、任意のイデアルは一つの多項式により$\langle f \rangle$の形で書ける

最大公約式は取れない

- 最大公約式が必ず取れるなら、任意のイデアルは一つの多項式により$\langle f \rangle$の形で書ける
- しかし、例えば$\langle x, y \rangle$について上のようなfは存在しない！

最大公約式は取れない

- 最大公約式が必ず取れるなら、任意のイデアルは一つの多項式により$\langle f \rangle$の形で書ける
  - しかし、例えば$\langle x, y \rangle$について上のようなfは存在しない！
- 別の手段を考えるべし

最大公約式は取れない

- 最大公約式が必ず取れるなら、任意のイデアルは一つの多項式により$\langle f \rangle$の形で書ける
  - しかし、例えば$\langle x, y \rangle$について上のようなfは存在しない！
- 別の手段を考えるべし
  - 割り算の余りが重要だった

最大公約式は取れない

- 最大公約式が必ず取れるなら、任意のイデアルは一つの多項式により$\langle f \rangle$の形で書ける
  - しかし、例えば$\langle x, y \rangle$について上のようなfは存在しない！
- 別の手段を考えるべし
  - 割り算の余りが重要だった

➡ 割り算を一般化してみよう！

割り算の一般化

割り算の一般化

- 複数の多項式で一つの多項式を割ろう！
  - 多項式を並べておいて順に割れないか試す
- 一変数多項式の場合の降冪の順のように、何か都合のよい並べ方を考える必要がある

割り算の一般化

- 複数の多項式で一つの多項式を割ろう！
  - 多項式を並べておいて順に割れないか試す
- 一変数多項式の場合の降冪の順のように、何か都合のよい並べ方を考える必要がある

➡ 単項式順序！

多項式の表現

多項式の表現

- 単項式順序の前に、多項式の表現について

多項式の表現

- 単項式順序の前に、多項式の表現について
- 文字と指数の列（$1,\ x^2,\ xy,\ y^2$ など）を単項式といい、それぞれに係数を書けて加えたものを多項式と呼ぶのだった

多項式の表現

- 単項式順序の前に、多項式の表現について
- 文字と指数の列（$1,\ x^2,\ xy,\ y^2$ など）を単項式といい、それぞれに係数を書けて加えたものを多項式と呼ぶのだった
- 文字の間に順番を決めれば、単項式は整数のベクトルと同一視出来る

多項式の表現

- 単項式順序の前に、多項式の表現について
- 文字と指数の列（$1,\ x^2,\ xy,\ y^2$ など）を単項式といい、それぞれに係数を書けて加えたものを多項式と呼ぶのだった
- 文字の間に順番を決めれば、単項式は整数のベクトルと同一視出来る
  - $x > y$ とすれば、$x^2 \leftrightarrow (2,0), xy \leftrightarrow (1,1),$ $1 \leftrightarrow (0,0)$ と言う具合に対応する

多項式の表現

- 単項式順序の前に、多項式の表現について
- 文字と指数の列（$1,\ x^2,\ xy,\ y^2$ など）を単項式といい、それぞれに係数を書けて加えたものを多項式と呼ぶのだった
- 文字の間に順番を決めれば、単項式は整数のベクトルと同一視出来る
  - $x > y$ とすれば、$x^2 \leftrightarrow (2,0), xy \leftrightarrow (1,1),$ $1 \leftrightarrow (0,0)$ と言う具合に対応する
  - そこで n-変数単項式全体を $\mathbb{Z}^n{}_{\geq 0}$ と同一視する

単項式順序

定義

単項式順序

定義

$\mathbb{Z}^n_{\geq 0}$ 上の順序関係 $>$ は、次の三つの条件を満たすとき単項式順序であると言う：

単項式順序

定義

$\mathbb{Z}^n_{\geq 0}$ 上の順序関係 $>$ は、次の三つの条件を満たすとき単項式順序であると言う：

1. 全順序性： $(\alpha < \beta)$ or $(\alpha = \beta)$ or $(\alpha > \beta)$

単項式順序

定義

$\mathbb{Z}^n_{\geq 0}$ 上の順序関係 $>$ は、次の三つの条件を満たすとき単項式順序であると言う：

1. 全順序性： $(\alpha < \beta) \text{ or } (\alpha = \beta) \text{ or } (\alpha > \beta)$
2. 加法性：$\alpha \leq \beta \Rightarrow \alpha + \gamma \leq \beta + \gamma$

単項式順序

定義

$\mathbb{Z}^n{}_{\geq 0}$ 上の順序関係 $>$ は、次の三つの条件を満たすとき単項式順序であると言う：

1. 全順序性： $(\alpha < \beta) \text{ or } (\alpha = \beta) \text{ or } (\alpha > \beta)$
2. 加法性：$\alpha \leq \beta \Rightarrow \alpha + \gamma \leq \beta + \gamma$
3. 非負性：$\alpha \geq 0$ (任意の $\alpha \in \mathbb{Z}^n{}_{\geq 0}$ について)

単項式順序の補足

単項式順序の補足

- 3 の非負性の条件は、全順序性と加法性の下で以下の二つと同値

単項式順序の補足

- 3 の非負性の条件は、全順序性と加法性の下で以下の二つと同値
  - 整列性：任意の空でない $\mathbb{Z}^n_{\geq 0}$ の部分集合が $<$ に関して最小値を持つ

単項式順序の補足

- 3 の非負性の条件は、全順序性と加法性の下で以下の二つと同値
  - 整列性：任意の空でない $\mathbb{Z}^n_{\geq 0}$ の部分集合が $<$ に関して最小値を持つ
  - 降鎖条件：$>$ に関する無限下降列

$$\alpha_0 > \alpha_1 > \alpha_2 > \cdots > \alpha_n > \cdots$$

は存在しない

単項式順序の補足

- 3 の非負性の条件は、全順序性と加法性の下で以下の二つと同値
  - 整列性：任意の空でない $\mathbb{Z}^n_{\geq 0}$ の部分集合が $<$ に関して最小値を持つ
  - 降鎖条件：$>$ に関する無限下降列

$$\alpha_0 > \alpha_1 > \alpha_2 > \cdots > \alpha_n > \cdots$$

  は存在しない

- これらは、アルゴリズムの停止性の証明で使う

単項式順序の例

- 以下 $\alpha = (a_1, \ldots, a_n), \beta = (b_1, \ldots, b_n)$ とする
- 辞書式順序 $>_{lex}$ は単項式順序

$$\alpha >_{lex} \beta :\Longleftrightarrow a_1 = b_1, \ldots, a_{i-1} = b_{i-1},\ a_i > b_i$$

- 逆辞書式順序 $>_{revlex}$ は単項式順序でない

$$\alpha >_{revlex} \beta :\Longleftrightarrow a_{i+1} = b_{i+1}, \ldots, a_n = b_n,\ a_i < b_i$$

（※最後の不等号の向きに注意！）

単項式順序の例

- 次数付き辞書式順序 $>_{grlex}$ は単項式順序

$$\alpha >_{grlex} \beta :\Longleftrightarrow |\alpha| > |\beta| \text{ or } (|\alpha|=|\beta| \text{ and } \alpha >_{lex} \beta)$$

ただし $|\alpha| = a_1 + \cdots + a_n$（全次数）

- 次数付き逆辞書式順序 $>_{grevlex}$ は単項式順序

$$\alpha >_{grevlex} \beta :\Longleftrightarrow |\alpha|>|\beta| \text{ or } (|\alpha|=|\beta| \text{ and } \alpha >_{revlex} \beta)$$

約束

- 単項式順序 $>$ を固定する。多項式 f に現れる（係数抜きの）単項式の中で、$>$ について最大となるものを先頭単項式と云い、$\mathrm{LM}(f)$ で表す。$\mathrm{LM}(f)$ の係数を先頭係数といい、$\mathrm{LC}(f)$ で表す。$\mathrm{LT}(f) = \mathrm{LC}(f)\,\mathrm{LM}(f)$ を先頭項と呼ぶ。

例

- $f = x + 2y^2w + 3z^3 \quad (x > y > z > w)$
- $>_{lex}$: $\mathrm{LM}(f) = x, \mathrm{LC}(f) = 1, \mathrm{LT}(f) = x$
- $>_{grlex}$: $\mathrm{LM}(f) = y^2w, \mathrm{LC}(f) = 2, \mathrm{LT}(f) = 2y^2w$
- $>_{grevlex}$: $\mathrm{LM}(f) = z^3, \mathrm{LC}(f) = 3, \mathrm{LT}(f) = 3z^3$

多変数の割り算

- 以上を用いて多項式版の割り算を定義する
- 単項式順序を一つ決め、割る式、割られる式ともに大きい単項式順に整列しておく
- どの順番で割るかは固定する！

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & \\
a_2 = & \\
\begin{array}{r} XY-1 \\ Y^2-1 \end{array} & \overline{\left)\, X^2Y + XY^2 + Y^2 \right.}
\end{array}
$$

$XY-1, Y^2-1$ で $X^2Y + XY^2 + Y^2$ を割る。辞書式順序を使う。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & \\
a_2 = & \\
XY - 1 \\
Y^2 - 1 &
\end{array}
\overline{\left)\, X^2Y + XY^2 + Y^2 \right.}
$$

先頭項に注目。最初の式で割れる

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X \\
a_2 = & \\
XY - 1 \;\overline{)} & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\overline{)} & X^2Y - X \\
\hline
\end{array}
$$

X を掛ければ先頭項が消える。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X \\
a_2 = & \\
XY - 1 \;\big)\!\!\big. & \overline{X^2Y + XY^2 + Y^2} \\
Y^2 - 1 \;\big)\!\!\big. & \overline{X^2Y - X} \\
\hline
 & XY^2 + X
\end{array}
$$

引く。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X \\
a_2 = & \\
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\hline
 & XY^2 + X + Y^2
\end{array}
$$

上から一つ降ろして、降冪の順に並べる

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X \ + \\
a_2 = & \\
XY - 1 \overline{)} & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \overline{)} & X^2Y - X \\
\hline
 & XY^2 + X + Y^2
\end{array}
$$

先頭項に注目。 $XY - 1$ で割れる。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & \\
\multicolumn{2}{r}{\begin{array}{r|l} XY - 1 & X^2Y + XY^2 + Y^2 \\ Y^2 - 1 & X^2Y - X \end{array}} \\
\multicolumn{2}{r}{\begin{array}{r} XY^2 + X + Y^2 \\ XY^2 - Y \end{array}} \\
\multicolumn{2}{r}{\begin{array}{r} X + Y^2 + Y \end{array}}
\end{array}
$$

先程と同様に計算。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & \\
XY - 1 \overline{)} & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \overline{)} & X^2Y - X \\
\hline
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\hline
 & X + Y^2 - Y
\end{array}
$$

先頭項に注目。XY, Y^2 どちらでも割れない。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & \qquad\qquad 1 \\
\hline
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\hline
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\hline
 & X + Y^2 - Y \longrightarrow X \\
\hline
 & Y^2 - Y
\end{array}
$$

割れないので X は余りとして横によける

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & 1 \\
\hline
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\hline
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\hline
 & X + Y^2 - Y \longrightarrow X \\
\hline
 & Y^2 - Y
\end{array}
$$

先頭項に注目。XY は無理だ

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & \quad 1 \\
\cline{2-2}
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\cline{2-2}
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\cline{2-2}
 & X + Y^2 - Y \longrightarrow X \\
\cline{2-2}
 & Y^2 - Y \\
 & Y^2 - 1 \\
\cline{2-2}
 & Y + 1
\end{array}
$$

Y^2 で割れるので、割る。

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & 1 \\
\hline
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\hline
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\hline
 & X + Y^2 - Y \longrightarrow X \\
\hline
 & Y^2 - Y \\
 & Y^2 - 1 \\
\hline
 & Y + 1 \longrightarrow Y \\
\hline
 & 1 \longrightarrow 1 \\
\hline
 & 0
\end{array}
$$

$Y, 1$ 共にどの項でも割れないので横によける

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & 1 \\
\hline
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\hline
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\hline
 & X + Y^2 - Y \longrightarrow X \\
\hline
 & Y^2 - Y \\
 & Y^2 - 1 \\
\hline
 & Y + 1 \longrightarrow Y \\
\hline
 & 1 \longrightarrow 1 \\
\hline
 & 0 \qquad r = X + Y + 1
\end{array}
$$

よけたのを足して、余り r が出る

実例：多変数の割り算

$$
\begin{array}{rl}
a_1 = & X + Y \\
a_2 = & 1 \\
\hline
XY - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
Y^2 - 1 \;\big)\; & X^2Y - X \\
\hline
 & XY^2 + X + Y^2 \\
 & XY^2 - Y \\
\hline
 & X + Y^2 - Y \longrightarrow X \\
\hline
 & Y^2 - Y \\
 & Y^2 - 1 \\
\hline
 & Y + 1 \longrightarrow Y \\
\hline
 & 1 \longrightarrow 1 \\
\hline
 & 0 \qquad r = X + Y + 1
\end{array}
$$

$$X^2Y + XY^2 + Y^2 = (XY - 1)\,(X+Y) + (Y^2 - 1)\cdot 1 + X + Y + 1$$

多変数割り算

- これでいいのでは……！？
- ところが、 あまり 嬉しくないことが……

あまり の問題

$$
\begin{array}{r}
a_1 = \\
a_2 = \\
Y^2 - 1 \overline{)\ X^2Y + XY^2 + Y^2} \\
XY - 1
\end{array}
$$

割る順番を変え、$Y^2 - 1, XY - 1$で$X^2Y + XY^2 + Y^2$を割る。

余りの問題

$$
\begin{array}{rl}
a_1 = & X + 1 \\
a_2 = & X \\
\hline
Y^2 - 1 \;\big)\; & X^2Y + XY^2 + Y^2 \\
XY - 1 \;\big)\; & X^2Y - X \\
\hline
& XY^2 + X + Y^2 \\
& XY^2 - X \\
\hline
& 2X + Y^2 \longrightarrow 2X \\
\hline
& Y^2 \\
& Y^2 - 1 \\
\hline
& 1 \longrightarrow 1 \\
\hline
& 0 \qquad\quad r = 2X + 1
\end{array}
$$

$$X^2Y + XY^2 + Y^2 = (XY - 1)\,X + (Y^2 - 1)(X + 1) + 2X + 1$$

余りが違う！

余りが違う！

- 多項式の割り算は 割る順番で余りが変わる！

余りが違う！

- 多項式の割り算は 割る順番で余りが変わる！
- 単に「割って余りゼロ⇒イデアルに入る！」と

は出来ない！

余りが違う！

- 多項式の割り算は 割る順番で余りが変わる！
- 単に「割って余りゼロ⇒イデアルに入る！」と
  は出来ない！
- 与えられたイデアルに対して、割り算の余りが
  順番に依らないような生成元を取れないか？

余りが違う！

- 多項式の割り算は 割る順番で余りが変わる！
- 単に「割って余りゼロ⇒イデアルに入る！」と
  は出来ない！
- 与えられたイデアルに対して、割り算の余りが
  順番に依らないような生成元を取れないか？

  ➡ Gröbner 基底！

Gröbner 基底

定義

単項式順序 $>$ を一つ固定する。多項式環のイデアル $I \subseteq k[\mathbb{X}]$ について、I の有限部分集合 $G = \{g_1, \ldots, g_n\} \subseteq I$ が次の条件を満たすとき、G は $>$ に関する I の Gröbner 基底 であると言う。

$$\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_n) \rangle$$

$$(\text{i.e. } \langle \mathrm{LT}(f) \mid f \in I \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_n) \rangle)$$

Gröbner 基底の性質

定理

Gröbner 基底の性質

定理

- 任意の多項式環イデアルに対し Gröbner 基底が存在する

Gröbner 基底の性質

定理

- 任意の多項式環イデアルに対し Gröbner 基底が存在する
- イデアル I の Gröbner 基底 $G = \{g_1, \ldots, g_k\}$ は I を生成する。即ち、$I = \langle g_1, \ldots, g_k \rangle$

Gröbner 基底の性質

定理

- 任意の多項式環イデアルに対し Gröbner 基底が存在する
- イデアル I の Gröbner 基底 $G = \{g_1, \ldots, g_k\}$ は I を生成する。即ち、$I = \langle g_1, \ldots, g_k \rangle$
- Gröbner 基底による割り算の余りは順番に依らない。

Gröbner 基底の性質

定理

- 任意の多項式環イデアルに対し Gröbner 基底が存在する
- イデアル I の Gröbner 基底 $G = \{g_1, \ldots, g_k\}$ は I を生成する。即ち、$I = \langle g_1, \ldots, g_k \rangle$
- Gröbner 基底による割り算の余りは順番に依らない。
- Gröbner 基底で割った余りがゼロである事と I に属することは同値： $r = \overline{f}^G = 0 \Longleftrightarrow f \in I$

まとめ

- Gröbner 基底は体上多項式環のイデアルの良い性質を持った生成元
- 単項式順序を決めると計算出来る
- 割り算の余りが順番に依らない
- イデアル所属問題に使える
- 他にも応用多数

ここまでで質問？

Table of Contents

- Gröbner 基底の簡単な導入
- Gröbner 基底の計算法
  - Buchberger アルゴリズム
  - 最適化手法：syzygy, sugar strategy
- Haskell の型レベルプログラミングの現在

Gröbner 基底の計算法

Buchberger アルゴリズム

Buchberger アルゴリズム

- Gröbner 基底が便利そうなのはわかった

Buchberger アルゴリズム

- Gröbner 基底が便利そうなのはわかった
- 具体的な計算法：Buchberger アルゴリズム

Buchberger アルゴリズム

- Gröbner 基底が便利そうなのはわかった
- 具体的な計算法：Buchberger アルゴリズム
- 準備的な定義：$f, g \in k[\mathbb{X}]$ の S-多項式

Buchberger アルゴリズム

- Gröbner 基底が便利そうなのはわかった
- 具体的な計算法：Buchberger アルゴリズム
- 準備的な定義：$f, g \in k[\mathbb{X}]$ の S-多項式

$$S(f, g) := \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(f)} f - \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(g)} g$$

Buchberger アルゴリズム

- Gröbner 基底が便利そうなのはわかった
- 具体的な計算法：Buchberger アルゴリズム
- 準備的な定義：$f, g \in k[\mathbb{X}]$ の S-多項式

- $S(f, g) := \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(f)} f - \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(g)} g$
- f, g の先頭項同士を打消し合わせた物

Buchberger アルゴリズム

- Gröbner 基底が便利そうなのはわかった
- 具体的な計算法：Buchberger アルゴリズム
- 準備的な定義：$f, g \in k[\mathbb{X}]$ の S-多項式

- $S(f, g) := \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(f)} f - \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(g)} g$
- f, g の先頭項同士を打消し合わせた物
- ただし $LCM((a_1, \ldots, a_n), (b_1, \ldots, b_n))$

$$:= (\max\{a_1, b_1\}, \ldots, \max\{a_n, b_n\})$$

Buchberger 判定法

定理

$I \subseteq k[\mathbb{X}]$：イデアル　　$>$：単項式順序

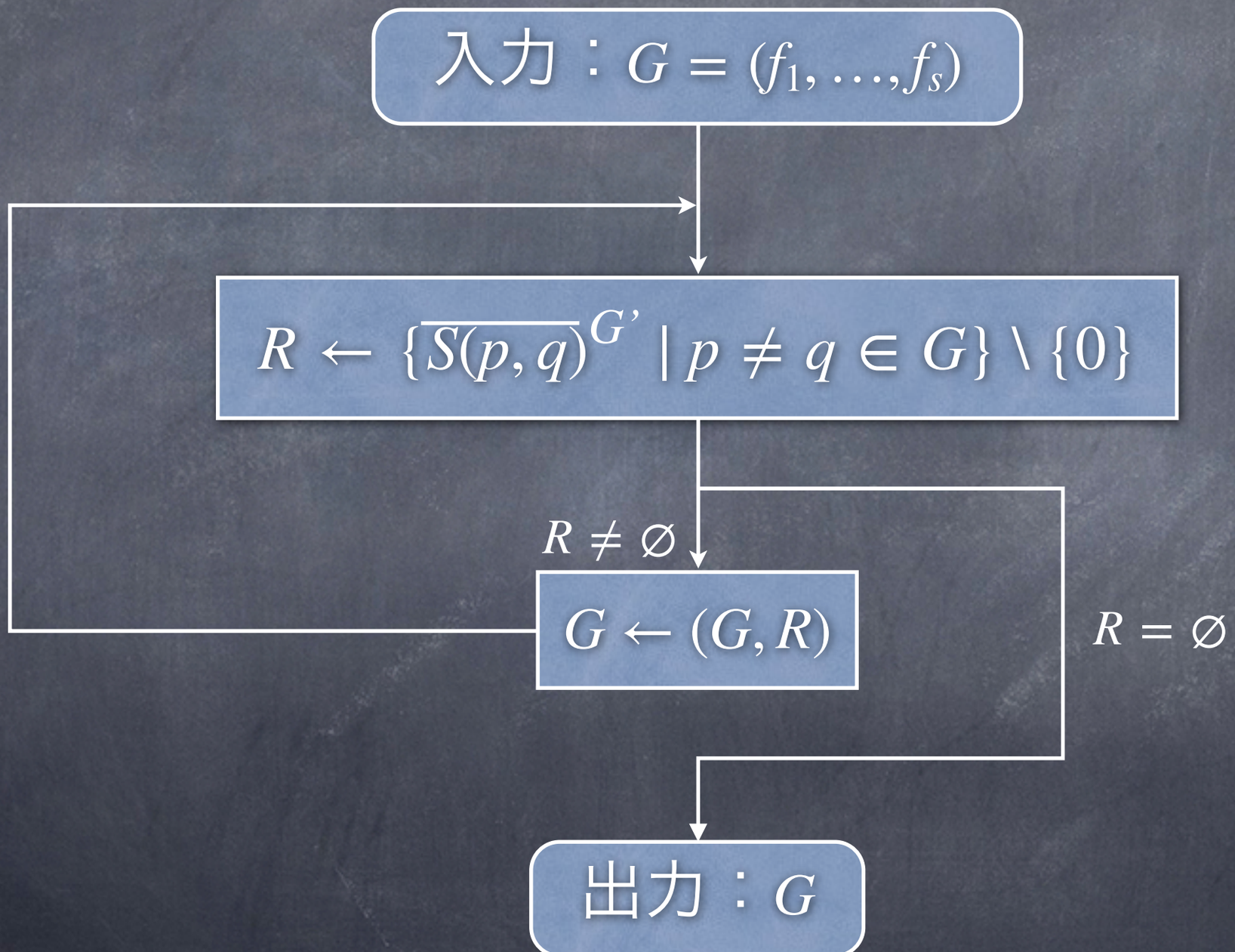
$G = \{g_1, \ldots, g_k\} \subseteq I$：$I$ の生成元

$G' = (g_1, \ldots, g_k)$：G の元を適当に並べたもの

このとき以下の二つは同値：

1. G は $>$ に関する I の Gröbner 基底
2. $\overline{S(g_i, g_k)}^{G'} = 0 \quad (\forall\, i < j)$

Buchberger アルゴリズム



簡単な実装

```haskell
buchberger :: [Polynomial k ord n] → [Polynomial k ord n]
buchberger ideal =
  fst $ until (null ∘ snd) loop (ideal, calc ideal)
  where
    loop (ggs, acc) =
      let cur = nub $ ggs ++ acc
      in (cur, calc cur)
    calc acc =
      [ q | f ← acc, g ← acc, f ≠ g
          , let q = sPolynomial f g `mod` acc
          , q ≠ zero
      ]
```

計算例

計算例

- $I = \langle x^2y - 1, x^3 - y^2 - x \rangle$ の Gröbner 基底

計算例

- $I = \langle x^2y - 1, x^3 - y^2 - x \rangle$ の Gröbner 基底
- Grevlex で計算

$$G = \{x^2y - 1, x^3 - y^2 - x, y^3 + xy - x, \ -y^3 - xy + x\}$$

計算例

- $I = \langle x^2y - 1, x^3 - y^2 - x \rangle$ の Gröbner 基底
- Grevlex で計算

  $G = \{x^2y - 1, x^3 - y^2 - x, y^3 + xy - x, \ -y^3 - xy + x\}$

- 割る順番で答えが変わらないことは、まあ、確かめといてください。

lex で計算すると？

lex で計算すると？

$$G = \{x^2y - 1, x^3 - y^2 - x, y^3 + xy - x, -y^3 - xy + x, y^5 + y^4 + y^3 + x^2 - x - 1, -y^5 - y^4 - y^3 - x^2 + x + 1, -y^6 - y^5 - y^4 - y^3 + x + y - 1, y^7 + 2y^6 + 2y^5 + 2y^4 + 2y^3 - y^2 - 2x + 1, y^6 + y^5 + y^4 + y^3 - x - y + 1, -y^7 - 2y^6 - 2y^5 - 2y^4 - 2y^3 + y^2 + 2x - 1, -y^9 - 2y^8 - 3y^7 + y^4 - 4y + 3, -1/2y^{10} - 3/2y^9 - 5/2y^8 - 7/2y^7 + 1/2y^5 + 1/2y^4 - 9/2y + 7/2, y^7 - y^2 + 2y - 1, 1/2y^8 + y^7 - 1/2y^3 + 3/2y - 1, -y^7 + y^2 - 2y + 1, -y^8 - 2y^7 + y^3 - 3y + 2, -1/2y^9 - 3/2y^8 - 5/2y^7 + 1/2y^4 + 1/2y^3 - 7/2y + 5/2, y^9 + 2y^8 + 3y^7 - y^4 + 4y - 3, y^8 + 2y^7 - y^3 + 3y - 2, 1/2y^7 - 1/2y^2 + y - 1/2, 1/2y^{10} + 3/2y^9 + 5/2y^8 + 7/2y^7 - 1/2y^5 - 1/2y^4 + 9/2y - 7/2, -1/2y^8 - y^7 + 1/2y^3 - 3/2y + 1, 1/2y^9 + 3/2y^8 + 5/2y^7 - 1/2y^4 - 1/2y^3 + 7/2y - 5/2, -1/2y^7 + 1/2y^2 - y + 1/2\}$$

でかっ！

効率の問題

- Grevlex が一番速くて基底も綺麗なものが求まりやすいとされる
  - lex は基底も複雑になるし時間がかかる
  - 物によっては Grevlex の方が遅いこともあるが、ほぼ大抵は Grevlex が圧倒的に速い
- 応用分野によっては grevlex ではなく lex を使う

極小・被約Gröbner基底

極小・被約Gröbner基底

- Buchberger アルゴリズムで求めた基底には無駄がある

極小・被約Gröbner基底

- Buchberger アルゴリズムで求めた基底には無駄がある
  - 極小・被約 Gröbner 基底！

極小・被約Gröbner基底

- Buchberger アルゴリズムで求めた基底には無駄がある
  - 極小・被約 Gröbner 基底！

- Gröbner基底 G が極小 $\Leftrightarrow$ 任意の i について

  $\mathrm{LC}(g_i) = 1, \mathrm{LT}(g_i)$ はどの $\mathrm{LT}(g_j)$ $(j \neq i)$ でも割れない

極小・被約Gröbner基底

- Buchberger アルゴリズムで求めた基底には無駄がある
  - 極小・被約 Gröbner 基底！

- Gröbner基底 G が極小 $\Leftrightarrow$ 任意の i について

  $\mathrm{LC}(g_i) = 1, \mathrm{LT}(g_i)$ はどの $\mathrm{LT}(g_j)$ $(j \neq i)$ でも割れない

- G が被約 $\Leftrightarrow$ 任意の i について $\mathrm{LC}(g_i) = 1$ かつ

  g_i のどの項も $\mathrm{LT}(g_j)$ $(j \neq i)$ でも割れない

被約 Gröbner 基底の一意性

被約 Gröbner 基底の一意性

- Gröbner 基底は複数存在しうる

被約 Gröbner 基底の一意性

- Gröbner 基底は複数存在しうる
  - buchberger で求めた物、そ
の極小化……etc

被約 Gröbner 基底の一意性

- Gröbner 基底は複数存在しうる
  - buchberger で求めた物、その極小化……etc
- それらは被約化すれば一致する！（単項式順序が同じなら）

定理

$I \subseteq k[\mathbb{X}]$：イデアル

$>$：単項式順序

I の $>$ についての被約 Gröbner 基底は一意に定まる

被約 Gröbner 基底の一意性

- Gröbner 基底は複数存在しうる
  - buchberger で求めた物、その極小化……etc
- それらは被約化すれば一致する！（単項式順序が同じなら）
- イデアルが等しいかどうかを、被約 Gröbner 基底の生成元が等しいかで判定出来る！

定理

$I \subseteq k[\mathbb{X}]$：イデアル

$>$：単項式順序

I の $>$ についての被約 Gröbner 基底は一意に定まる

被約化の手続

被約化の手続

- まず極小化を行う

1. G の各元を最高次係数で割って LC(p) = 1
2. (2) の条件を満たさない物を取り除く

被約化の手続

- まず極小化を行う

  1. G の各元を最高次係数で割って LC(p) = 1
  2. (2) の条件を満たさない物を取り除く

- それを被約化する

  1. $p \in G$ を取り、$p' = \overline{p}^{G \setminus \{p\}}, G' = G \setminus \{p\} \cup \{p'\}$ とする
  2. $q \in G' \setminus \{p'\}$ について同様の事をする。以下同文。

計算例

計算例

- 先程のイデアルについて

計算例

- 先程のイデアルについて
- Grevlex に関する被約基底

$$G = \{y^3 + xy - x, x^2y - 1, x^3 - y^2 - x\}$$

計算例

- 先程のイデアルについて
- Grevlex に関する被約基底

  $G = \{y^3 + xy - x, x^2y - 1, x^3 - y^2 - x\}$
- Lex に関する被約基底

  $G = \{y^7 - y^2 + 2y - 1, -y^6 - y^5 - y^4 - y^3 + x + y - 1\}$

計算例

- 先程のイデアルについて
- Grevlex に関する被約基底

  $G = \{y^3 + xy - x, x^2y - 1, x^3 - y^2 - x\}$
- Lex に関する被約基底

  $G = \{y^7 - y^2 + 2y - 1, -y^6 - y^5 - y^4 - y^3 + x + y - 1\}$
  - うわっ…ナイーブな基底、無駄多すぎ…？

計算例

- 先程のイデアルについて
- Grevlex に関する被約基底

  $G = \{y^3 + xy - x, x^2y - 1, x^3 - y^2 - x\}$
- Lex に関する被約基底

  $G = \{y^7 - y^2 + 2y - 1, -y^6 - y^5 - y^4 - y^3 + x + y - 1\}$
  - うわっ…ナイーブな基底、無駄多すぎ…？
- 最初から被約グレブナー基底を産むアルゴリズムもある

実装例

```
minimizeGroebnerBasis :: (Field k, IsPolynomial k n, IsMonomialOrder order)
                      ⇒ [OrderedPolynomial k ord n] → [OrderedPolynomial k ord n]
minimizeGroebnerBasis bs = runST $ do
  left  ← newSTRef bs
  right ← newSTRef []
  whileM_ (not . null <$> readSTRef left) $ do
    f : xs ← readSTRef left
    writeSTRef left xs
    ys     ← readSTRef right
    if any (λg → leadingMonomial g `divs` leadingMonomial f) xs ∨
       any (λg → leadingMonomial g `divs` leadingMonomial f) ys
      then writeSTRef right ys
      else writeSTRef right (monoize f : ys)
  readSTRef right
```

- 適宜アップデートする必要があるので ST で楽して書いた
- 被約化の実装も同様

Table of Contents

- Gröbner 基底の簡単な導入
- Gröbner 基底の計算法
  - Buchberger アルゴリズム
  - 最適化手法：syzygy, sugar strategy
- Haskell の型レベルプログラミングの現在

最適化手法

- ナイーブな実装だと遅い！
- 代表的な実装である SINGULAR の千倍くらい遅い（推定）
- 何とか百倍くらいにする方法
  - 指数オーダーなので〜倍っていう表現はおかしいのだけど・・・

紹介する手法

- ゼロ簡約関係
- syzygy の基底
- sugar strategy

紹介する手法

- ゼロ簡約関係
- syzygy の基底
- sugar strategy

ゼロ簡約関係

ゼロ簡約関係

- Buchberger は何故遅い？

ゼロ簡約関係

- Buchberger は何故遅い？
  - Buchberger は割り算のかたまり

ゼロ簡約関係

- Buchberger は何故遅い？
  - Buchberger は割り算のかたまり
  - lex より grevlex が速いのは、そっちのほうが割り算のステップが小さいから

ゼロ簡約関係

- Buchberger は何故遅い？
  - Buchberger は割り算のかたまり
  - lex より grevlex が速いのは、そっちのほうが割り算のステップが小さいから
  - 割り算の回数をもっと減らせない？

ゼロ簡約関係

「割って余りゼロ」を一般化したのが次

ゼロ簡約関係

「割って余りゼロ」を一般化したのが次

定義

多項式 $f \in k[\mathbb{X}]$ が $G = \{g_1, \ldots, g_k\} \subseteq k[\mathbb{X}]$ を

法としてゼロに簡約（記号：$f \to_G 0$）される

$\Leftrightarrow f = a_1 g_1 + \cdots + a_k g_k \ (a_i \in k[\mathbb{X}])$ かつ

$[a_i g_i = 0$ または $\deg(f) \geq \deg(a_i g_i)]$

但し、$\deg(f)$ は f の多重次数 $\deg(f) = \mathrm{LM}(f)$

改良 Buchberger 判定法

実は Buchberger 判定法はゼロ簡約で十分！

改良 Buchberger 判定法

実は Buchberger 判定法はゼロ簡約で十分！

定理

$I \subseteq k[\mathbb{X}]$：イデアル　　$>$：単項式順序

$G = \{g_1, \ldots, g_k\} \subseteq I$：$I$ の生成元

このとき以下の二つは同値：

1. G は $>$ に関する I の Gröbner 基底
2. $S(g_i, g_j) \ \rightarrow_G \ 0 \quad (\forall\, i < j)$

具体的にいつゼロ簡約？

命題

$I \subseteq k[\mathbb{X}]$：イデアル　　$>$：単項式順序

$G = \{g_1, \ldots, g_k\} \subseteq I$: I の生成元

具体的にいつゼロ簡約？

命題

$I \subseteq k[\mathbb{X}]$：イデアル　　$>$：単項式順序

$G = \{g_1, \ldots, g_k\} \subseteq I$: I の生成元

- $\overline{S(f,g)}^{G'} = 0 \Rightarrow S(f,g) \to_G 0$

具体的にいつゼロ簡約？

命題

$I \subseteq k[\mathbb{X}]$：イデアル　　$>$：単項式順序

$G = \{g_1, \ldots, g_k\} \subseteq I$：$I$ の生成元

- $\overline{S(f,g)}^{G,} = 0 \Rightarrow S(f,g) \to_G 0$
- $\mathrm{LCM}(\mathrm{LM}(f), \mathrm{LM}(g)) = \mathrm{LM}(f)\,\mathrm{LM}(g) \Rightarrow S(f,g) \to_G 0$

  （f, g が 互いに素 なら S-多項式はゼロに簡約される）

互いに素判定

- 二番目の条件が使える！
  - x^3 と z^4 は互いに素なので $S(x^3 + y, z^4) \to_G 0$ となる

$$S(x^3 + y, z^4) = yz^4$$

$$\therefore \overline{S(x^3 + y, z^4)}^G = y \neq 0$$

今までの判定法だと無駄に割り算をする！

- どれくらい改善？

Benchmark

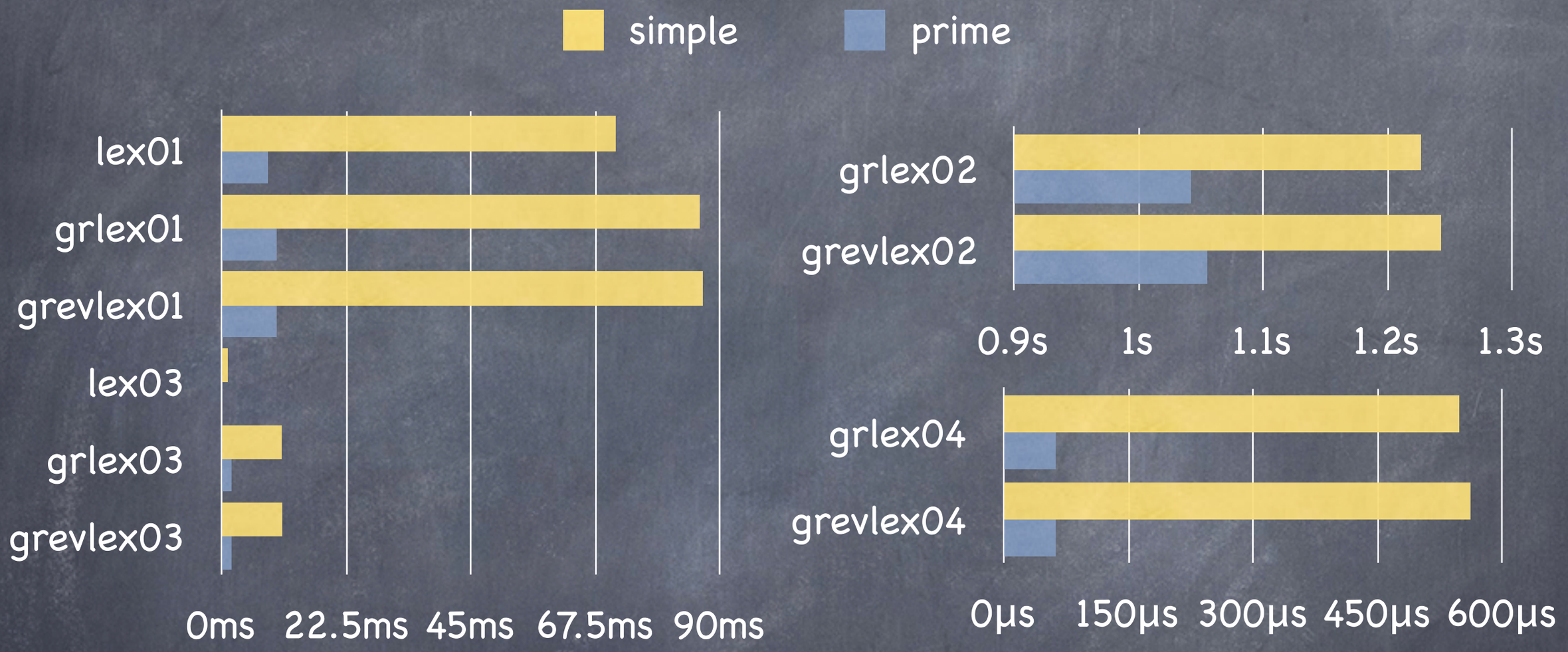
simple prime

$$I_1 = \langle x^2 + y^2 + z^2 - 1, x^2 + y^2 + z^2 - 2x, 2x - 3y - z \rangle$$

$$I_2 = \langle x^2y - 2xy - 4z - 1, z - y^2, x^3 - 4zy \rangle$$

$$I_3 = \langle 2S - ay, b^2 - (x^2 + y^2), c^2 - ((a - x)^2 + y^2) \rangle \quad I_4 = \langle z^5 + y^4 + x^3 - 1, z^3 + y^3 + x^2 - 1 \rangle$$

Benchmark



$$I_1 = \langle x^2 + y^2 + z^2 - 1, x^2 + y^2 + z^2 - 2x, 2x - 3y - z \rangle$$

$$I_2 = \langle x^2y - 2xy - 4z - 1, z - y^2, x^3 - 4zy \rangle$$

$$I_3 = \langle 2S - ay, b^2 - (x^2 + y^2), c^2 - ((a - x)^2 + y^2) \rangle \quad I_4 = \langle z^5 + y^4 + x^3 - 1, z^3 + y^3 + x^2 - 1 \rangle$$

紹介する手法

- ゼロ簡約関係
- syzygy の基底
- sugar strategy

syzygy の基底

- そもそも S-多項式とは何なのか？
- 実は syzygy と呼ばれるものの基底！
- Gröbner 基底の基底とは意味が若干違う
- “syzygy のもと” と思えばよい

syzygy

定義

syzygy

定義

- $(h_1, \ldots, h_s) \in k[\mathbb{X}]^s$ が $F = (f_1, \ldots, f_s) \subseteq k[\mathbb{X}]^s$ の 先頭項の

  syzygy $:\Leftrightarrow h_1 \mathrm{LT}(f_1) + \cdots + h_s \mathrm{LT}(f_s) = 0$

  F の先頭項 syzygy の全体を $S(F)$ と書く。

syzygy

定義

- $(h_1, \ldots, h_s) \in k[\mathbb{X}]^s$ が $F = (f_1, \ldots, f_s) \subseteq k[\mathbb{X}]^s$ の 先頭項の

  syzygy $:\Leftrightarrow h_1 \mathrm{LT}(f_1) + \cdots + h_s \mathrm{LT}(f_s) = 0$

  F の先頭項 syzygy の全体を $S(F)$ と書く。

- $T \subseteq S(F)$ が $S(F)$ の基底 $\Leftrightarrow S(F)$ のどの元も T の元の有限個の和で書ける。

syzygy

定義

- $(h_1, \ldots, h_s) \in k[\mathbb{X}]^s$ が $F = (f_1, \ldots, f_s) \subseteq k[\mathbb{X}]^s$ の 先頭項の

  syzygy $:\Leftrightarrow\ h_1 \mathrm{LT}(f_1) + \cdots + h_s \mathrm{LT}(f_s) = 0$

  F の先頭項 syzygy の全体を $S(F)$ と書く。

- $T \subseteq S(F)$ が $S(F)$ の基底 $\Leftrightarrow S(F)$ のどの元も T の元の有限個の和で書ける。

- $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ (i 番目だけ 1) と書くことにする。このとき

$$S_{ij} := \frac{LCM(LT(f_i), LT(f_j))}{LT(f_i)}\,\mathbf{e}_i - \frac{LCM(LT(f_i), LT(f_j))}{LT(f_j)}\,\mathbf{e}_j$$

と置くと、 $S(f_i, f_j) = S_{ij} \cdot F$ （内積）となる。

syzygy の基底とゼロ簡約

syzygy の基底とゼロ簡約

定理

$I \subseteq k[\mathbb{X}]$：イデアル　　$>$：単項式順序

$G = \{g_1, \ldots, g_k\} \subseteq I$：$I$ の生成元

$S \subseteq S(G)$：G の先頭項 syzygy の基底

このとき以下の二つは同値：

1. G は $>$ に関する I の Gröbner 基底
2. 任意の $T \in S$ に対し、

$$T \cdot G = \sum_{i=1}^{k} t_i g_i \to_G 0 \quad (\forall i < j)$$

S-多項式と syzygy

S-多項式と syzygy

- 実は S_{ij} の全体は $S(G)$ の基底

S-多項式と syzygy

- 実は S_{ij} の全体は $S(G)$ の基底
  - 余分な基底も混じっている

$$F = (x^2y^2 + z, xy^2 - y, x^2y + yz)$$

$$S_{12} = (1, \ -x, 0) \quad S_{13} = (1, 0, \ -y) \quad S_{23} = (0, x, \ -y)$$

$$S_{13} - S_{12} = (0, x, \ -y) = S_{23}$$

$\therefore S_{23}$ は要らない子

S-多項式と syzygy

- 実は S_{ij} の全体は $S(G)$ の基底
  - 余分な基底も混じっている

$$F = (x^2y^2 + z, xy^2 - y, x^2y + yz)$$

$$S_{12} = (1, -x, 0) \quad S_{13} = (1, 0, -y) \quad S_{23} = (0, x, -y)$$

$$S_{13} - S_{12} = (0, x, -y) = S_{23}$$

$\therefore S_{23}$ は要らない子

- 余分な基底を予め判定出来ないか？

無駄な syzygy 基底の判定

無駄な syzygy 基底の判定

命題

$G = (g_1, \ldots, g_s) \quad g_i, g_j, g_k \in G$
$\mathcal{S} \subseteq \{S_{ij} \mid 1 \leq j < k \leq s\}$: $S(G)$ の基底

$\mathrm{LT}(g_k) \mid \mathrm{LCM}(\mathrm{LT}(g_i), \mathrm{LT}(g_j))$ かつ $S_{ik}, S_{jk} \in \mathcal{S}$

$\Rightarrow \quad S \setminus \{S_{ij}\}$ も $S(G)$ の基底

無駄な syzygy 基底の判定

命題

$G = (g_1, \ldots, g_s) \quad g_i, g_j, g_k \in G$
$\mathcal{S} \subseteq \{S_{ij} \mid 1 \leq j < k \leq s\}$: $S(G)$ の基底

$\mathrm{LT}(g_k) \mid \mathrm{LCM}(\mathrm{LT}(g_i), \mathrm{LT}(g_j))$ かつ $S_{ik}, S_{jk} \in \mathcal{S}$

$\Rightarrow \quad S \setminus \{S_{ij}\}$ も $S(G)$ の基底

これで割り算の回数を大幅に減らせる！

改良版 Buchberger・改

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i, j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）
  $G := F \quad t := s$

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）
  $G := F \quad t := s$
- 手順： $B = \varnothing$ となるまで次を繰り返す：

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）
  $G := F \quad t := s$
- 手順： $B = \varnothing$ となるまで次を繰り返す：
  1. $(i,j) \in B$ を任意に選ぶ

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）
  $G := F \quad t := s$
- 手順： $B = \varnothing$ となるまで次を繰り返す：
  1. $(i,j) \in B$ を任意に選ぶ
  2. $\mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j)) \neq \mathrm{LT}(f_i)\,\mathrm{LT}(f_j)$ かつ $\mathrm{Test}(f_i, f_j, B)$ が偽なら

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）

  $G := F \quad t := s$
- 手順： $B = \varnothing$ となるまで次を繰り返す：

1. $(i,j) \in B$ を任意に選ぶ
2. $\mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j)) \neq \mathrm{LT}(f_i)\,\mathrm{LT}(f_j)$ かつ $\mathrm{Test}(f_i, f_j, B)$ が偽なら
   1. $S := \overline{S(f_i,f_j)}^{G}$

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）

  $G := F \quad t := s$
- 手順： $B = \varnothing$ となるまで次を繰り返す：

1. $(i,j) \in B$ を任意に選ぶ
2. $\mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j)) \neq \mathrm{LT}(f_i)\,\mathrm{LT}(f_j)$ かつ $\mathrm{Test}(f_i, f_j, B)$ が偽なら
   1. $S := \overline{S(f_i,f_j)}^{G}$
   2. $S \neq 0$ ならば

      $t = t + 1,\ f_t = S, G = G \cup \{f_t\}, B = B \cup \{(i,t) | 1 \leq i \leq t-1\}$

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）

  $G := F \quad t := s$
- 手順： $B = \emptyset$ となるまで次を繰り返す：

1. $(i,j) \in B$ を任意に選ぶ
2. $\mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j)) \neq \mathrm{LT}(f_i)\,\mathrm{LT}(f_j)$ かつ $\mathrm{Test}(f_i, f_j, B)$ が偽なら
   1. $S := \overline{S(f_i,f_j)}^{G}$
   2. $S \neq 0$ ならば

      $t = t + 1,\ f_t = S, G = G \cup \{f_t\}, B = B \cup \{(i,t) | 1 \leq i \leq t-1\}$
3. $B = B \setminus \{(i,j)\}$

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）

  $G := F \quad t := s$
- 手順： $B = \emptyset$ となるまで次を繰り返す：
  1. $(i,j) \in B$ を任意に選ぶ
  2. $\mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j)) \neq \mathrm{LT}(f_i)\,\mathrm{LT}(f_j)$ かつ $\mathrm{Test}(f_i, f_j, B)$ が偽なら
     1. $S := \overline{S(f_i,f_j)}^{G}$
     2. $S \neq 0$ ならば

        $t = t + 1,\ f_t = S, G = G \cup \{f_t\}, B = B \cup \{(i,t) | 1 \leq i \leq t-1\}$
  3. $B = B \setminus \{(i,j)\}$
- 出力：G

改良版 Buchberger・改

- 入力： $F = (f_1, \ldots, f_s)$
- 初期化：$B := \{(i,j) | 1 \leq i < j \leq s\}$（syzygy基底の候補）

  $G := F \quad t := s$
- 手順： $B = \emptyset$ となるまで次を繰り返す：
  1. $(i,j) \in B$ を任意に選ぶ
  2. $\mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j)) \neq \mathrm{LT}(f_i)\,\mathrm{LT}(f_j)$ かつ $\mathrm{Test}(f_i, f_j, B)$ が偽なら
     1. $S := \overline{S(f_i,f_j)}^{G}$
     2. $S \neq 0$ ならば

        $t = t + 1,\ f_t = S, G = G \cup \{f_t\}, B = B \cup \{(i,t) | 1 \leq i \leq t-1\}$
  3. $B = B \setminus \{(i,j)\}$
- 出力：G
- 但し$\mathrm{Test}(f_i, f_j, B) \Leftrightarrow \exists k \neq i,j\ [(i,k), (i,j) \notin B,\ \mathrm{LT}(f_k) \mid \mathrm{LCM}(\mathrm{LT}(f_i), \mathrm{LT}(f_j))]$

実装

```
syzygyBuchberger ideal = runST $ do
  bases ← newSTRef { Entry LM(f) f | f ← ideal }
  pairs ← newSTRef {(f, g) | (f, g) ← combinations ideal }
  len ← newSTRef $ length ideal
  whileM_ (not . null <$> readSTRef pairs) $ do
    Just ((f, g), rest) ← viewMin <$> readSTRef pairs
    bases0 ← readSTRef bases
    b .= rest
    let redundant =
          any (λ(Entry _ h) → h ∉ {f, g} ∧ (all (∉ rest) [(f, h), (g, h)])
                                ∧ LM(h) | LCM(LM(f), LM(g))) bases
    when (LCM(LM(f), LM(g)) ≠ LM(f) LM(g) ∧ not redundant) $ do
      len0 ← readSTRef len
      let s = S(f, g) `mod` map payload (toList bases0)
      when (s ≠ zero) $ do
        pairs += { (q, s) | Entry _ q ← qs }
        bases %= insert (Entry LM(s) s)
        len *= 2
  map payload . toList <$> readSTRef bases
```

実装の解説

実装の解説

- 効率のため ST モナドにした

実装の解説

- 効率のため ST モナドにした
- {} で囲まれているのは優先度キューのつもり

実装の解説

- 効率のため ST モナドにした
- {} で囲まれているのは優先度キューのつもり
- 経験的に、割り算をする際は先頭項を昇順に並べておくとよい

実装の解説

- 効率のため ST モナドにした
- {} で囲まれているのは優先度キューのつもり
- 経験的に、割り算をする際は先頭項を昇順に並べておくとよい
- どんどん割れて回数が少なくなる

実装の解説

- 効率のため ST モナドにした
- {} で囲まれているのは優先度キューのつもり
- 経験的に、割り算をする際は先頭項を昇順に並べておくとよい
- どんどん割れて回数が少なくなる
- 基底候補を先頭項を優先度とするキューとして持っておいて、割るときに toList で昇順に並べている

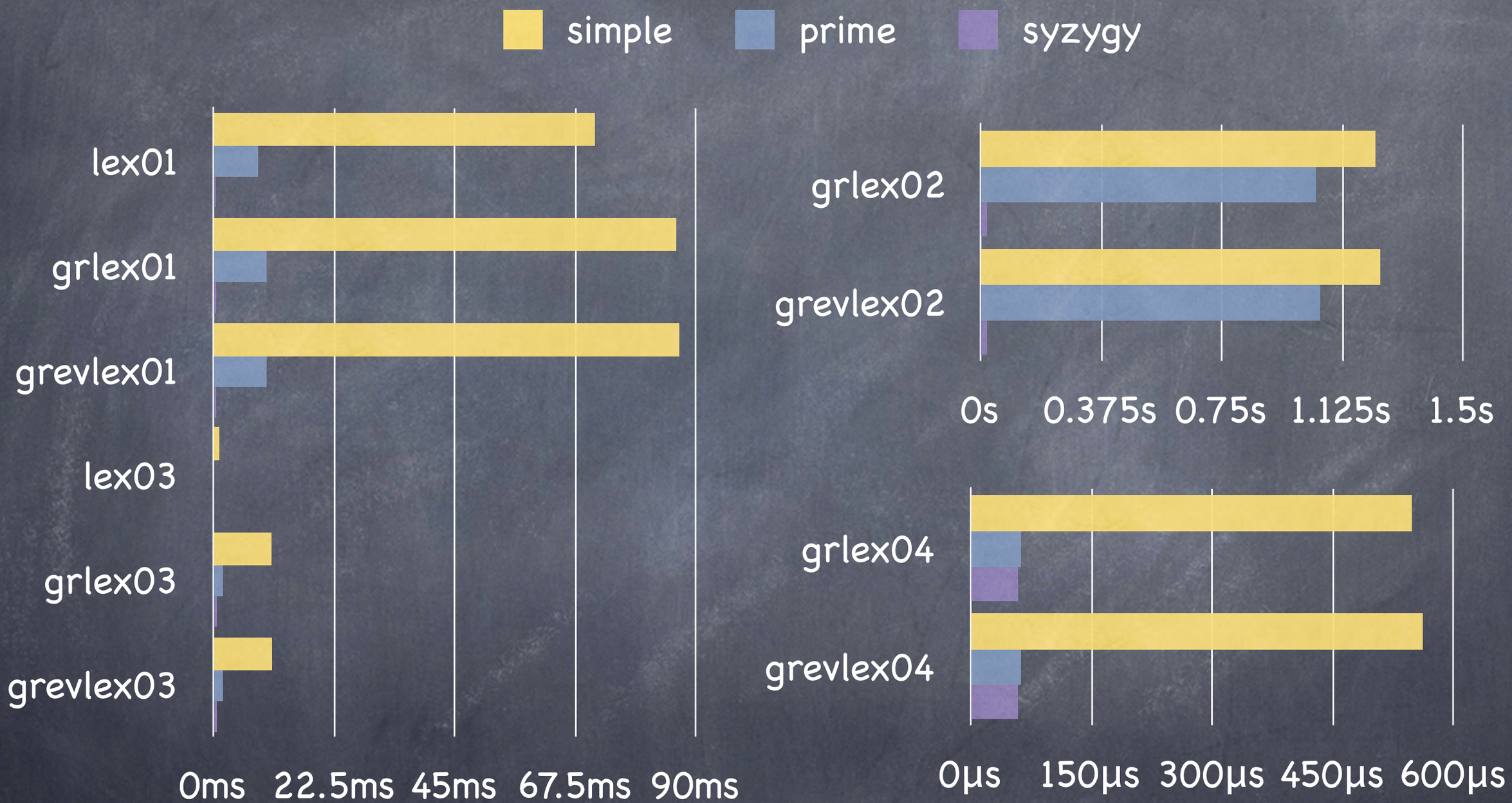
Benchmark

simple prime syzygy

Benchmark

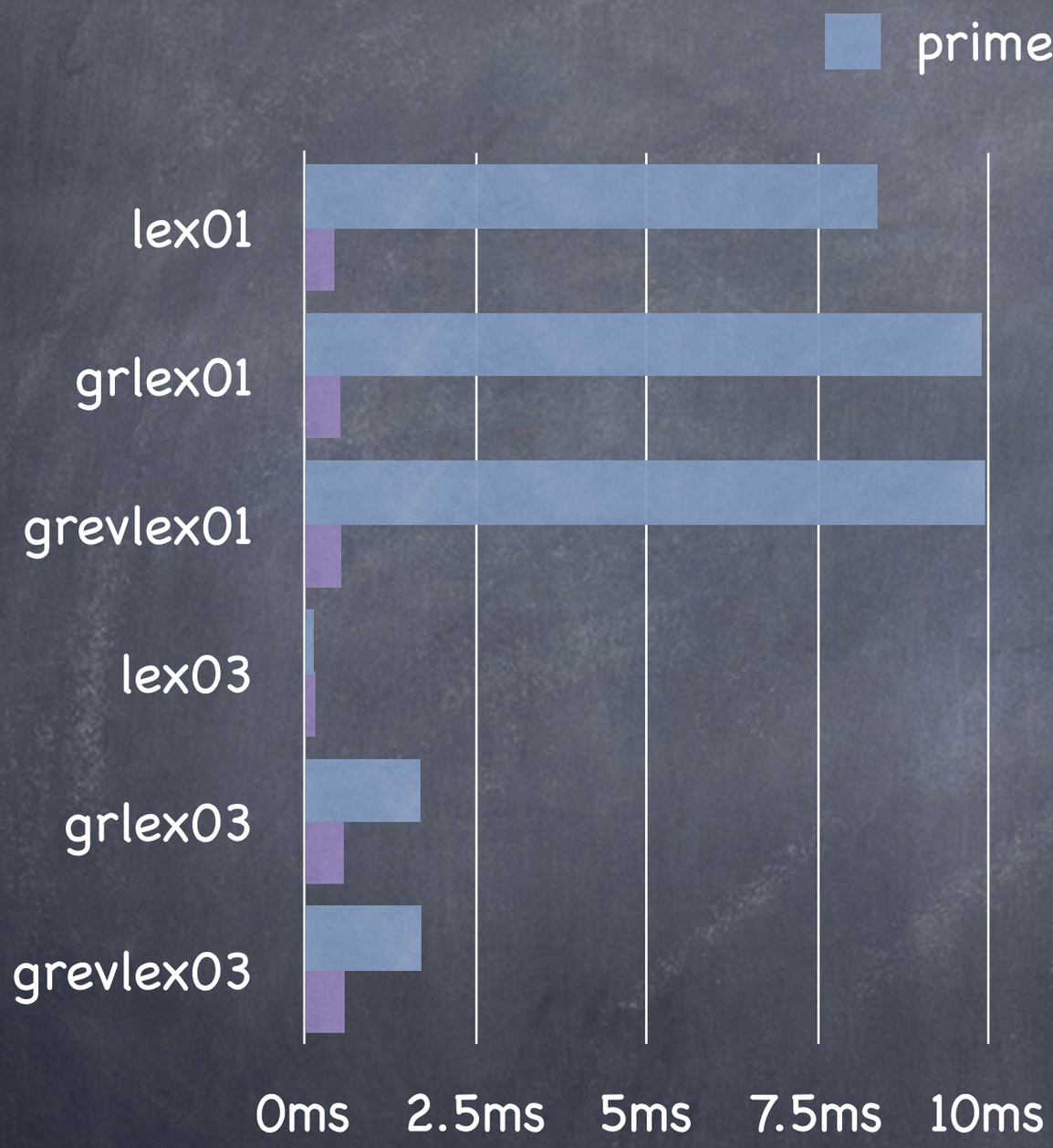

simple
prime
syzygy
lex01
grlex01
grevlex01
lex03
grlex03
grevlex03
0ms
22.5ms
45ms
67.5ms
90ms
grlex02
grevlex02
0s
0.375s
0.75s
1.125s
1.5s
grlex04
grevlex04
0µs
150µs
300µs
450µs
600µs

Benchmark



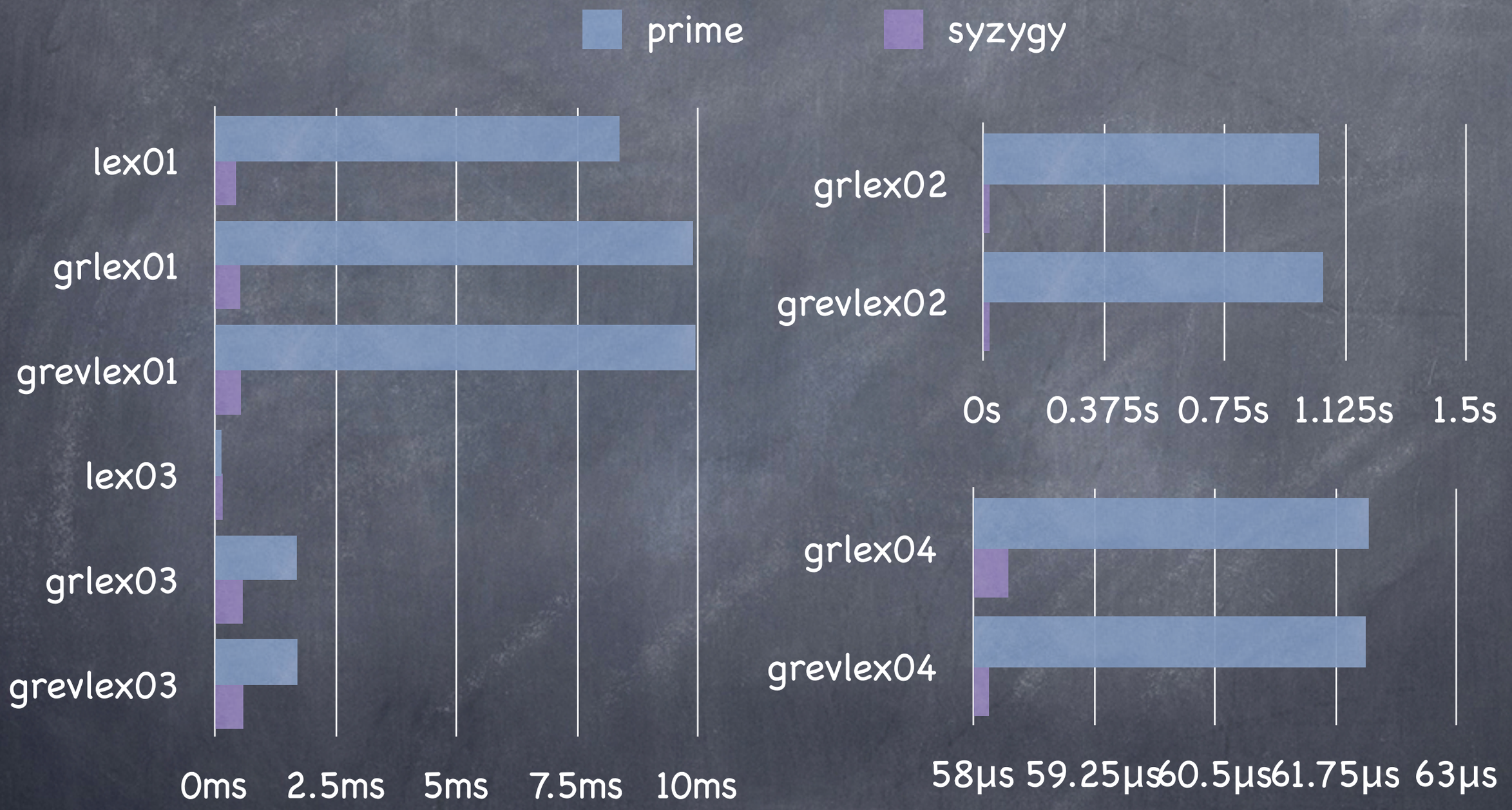
速すぎて見えない……

互いに素判定との比較


prime
syzygy
lex01
grlex01
grevlex01
lex03
grlex03
grevlex03
0ms
2.5ms
5ms
7.5ms
10ms


grlex02
grevlex02
0s
0.375s
0.75s
1.125s
1.5s
grlex04
grevlex04
58µs
59.25µs
60.5µs
61.75µs
63µs

互いに素判定との比較



それでも速い！

紹介する手法

- ゼロ簡約関係
- syzygy の基底
- sugar strategy

Buchberger の自由度

Buchberger の自由度

- 基底の候補 (i,j) を検討していく順番は自由

Buchberger の自由度

- 基底の候補 (i,j) を検討していく順番は自由
- この順番の選び方をどうするか？ = selection strategy

Buchberger の自由度

- 基底の候補 (i,j) を検討していく順番は自由
- この順番の選び方をどうするか？ = selection strategy
  - ヒューリスティックな戦略が知られている

selection strategy

selection strategy

- normal strategy

selection strategy

- normal strategy
  - 先頭項の LCM が常に最小になるもの

selection strategy

- normal strategy
  - 先頭項の LCM が常に最小になるもの
  - lex 順序など次数を保たない順序では逆効果

selection strategy

- normal strategy
  - 先頭項の LCM が常に最小になるもの
  - lex 順序など次数を保たない順序では逆効果
- sugar strategy

selection strategy

- normal strategy
  - 先頭項の LCM が常に最小になるもの
  - lex 順序など次数を保たない順序では逆効果
- sugar strategy
  - 仮想的な斉次化次数が最小の物を選ぶ

selection strategy

- normal strategy
  - 先頭項の LCM が常に最小になるもの
  - lex 順序など次数を保たない順序では逆効果
- sugar strategy
  - 仮想的な斉次化次数が最小の物を選ぶ
- いずれも同率一位が出たら早く出て来た方を選ぶ

selection strategy

- normal strategy
  - 先頭項の LCM が常に最小になるもの
  - lex 順序など次数を保たない順序では逆効果
- sugar strategy
  - 仮想的な斉次化次数が最小の物を選ぶ
- いずれも同率一位が出たら早く出て来た方を選ぶ
  - この条件を抜かすと恐しく遅くなる……

斉次化次数

斉次化次数

- 斉次式（各単項式の全次数が等しい多項式）を生成元とするイデアルの計算は経験的に速い

斉次化次数

- 斉次式（各単項式の全次数が等しい多項式）を生成元とするイデアルの計算は経験的に速い
- 余分な変数で生成元を斉次化してから計算するというヒューリスティクスが知られている

斉次化次数

- 斉次式（各単項式の全次数が等しい多項式）を生成元とするイデアルの計算は経験的に速い
- 余分な変数で生成元を斉次化してから計算するというヒューリスティクスが知られている
- 斉次化の例：$xy + 2x + y^3 + 3 \quad \Rightarrow \quad xyz + 2xz^2 + y^3 + 3z^3$

斉次化次数

- 斉次式（各単項式の全次数が等しい多項式）を生成元とするイデアルの計算は経験的に速い
- 余分な変数で生成元を斉次化してから計算するというヒューリスティクスが知られている
  - 斉次化の例：$xy + 2x + y^3 + 3 \quad \Rightarrow \quad xyz + 2xz^2 + y^3 + 3z^3$
- 問題点：得られた基底から余分な変数を消去しても Gröbner 基底になるとは限らない

斉次化次数

- 斉次式（各単項式の全次数が等しい多項式）を生成元とするイデアルの計算は経験的に速い
- 余分な変数で生成元を斉次化してから計算するというヒューリスティクスが知られている
  - 斉次化の例：$xy + 2x + y^3 + 3 \quad \Rightarrow \quad xyz + 2xz^2 + y^3 + 3z^3$
- 問題点：得られた基底から余分な変数を消去しても Gröbner 基底になるとは限らない
  - 斉次化して得られた元に対してもう一度 Buchberger を使うことになる（それでも多少は速くなる場合があるらしい）

sugar strategy

sugar strategy

- 普通に Buchberger を計算するが、(i, j) を選択するときだけ仮想的な斉次化を行い、その次数最小のものを選ぶ

sugar strategy

- 普通に Buchberger を計算するが、(i, j) を選択するときだけ仮想的な斉次化を行い、その次数最小のものを選ぶ
- この仮想的な次数のことを sugar という

sugar

多項式 (f_i, f_j) の sugar を次のように定める。

sugar

多項式 (f_i, f_j) の sugar を次のように定める。

- $S_{f_i} = \deg(f_i)$, $S_{f_j} = \deg(f_j)$

sugar

多項式 (f_i, f_j) の sugar を次のように定める。

- $S_{f_i} = \deg(f_i)$, $S_{f_j} = \deg(f_j)$
- S_f：given で t が単項式なら $S_{tf} = \deg(t) + S_f$

sugar

多項式 (f_i, f_j) の sugar を次のように定める。

- $S_{f_i} = \deg(f_i)$, $S_{f_j} = \deg(f_j)$
- S_f：given で t が単項式なら $S_{tf} = \deg(t) + S_f$
- $f = g + h$ の時、$S_f = \max(S_g, S_f)$

sugar

多項式 (f_i, f_j) の sugar を次のように定める。

- $S_{f_i} = \deg(f_i)$, $S_{f_j} = \deg(f_j)$
- S_f：given で t が単項式なら $S_{tf} = \deg(t) + S_f$
- $f = g + h$ の時、$S_f = \max(S_g, S_f)$
- (f_i, f_j) の sugar は、上に従って計算した S-多項式の sugar とする。

実際の実装

実際の実装

- おあつらえ向きに優先度キューを使っていた

実際の実装

- おあつらえ向きに優先度キューを使っていた
  - "selection strategy = 優先度の計算方法"
として抽象化出来る！

実際の実装

- おあつらえ向きに優先度キューを使っていた
  - "selection strategy = 優先度の計算方法"
として抽象化出来る！
- double sugar という手法はこれだけでは無理
  （でもそこまで効果がないらしいので無視）

実際の実装

- おあつらえ向きに優先度キューを使っていた
  - "selection strategy = 優先度の計算方法"
    として抽象化出来る！
    - double sugar という手法はこれだけでは無理
      （でもそこまで効果がないらしいので無視）
- (i, j) を保存する時に重みも保存しておく

実装

```
syzygyBuchbergerWithStrategy strategy ideal = runST $ do
  let gens = zip [1..] ideal
  bases ← newSTRef { Entry LM(f) f | (_, f) ← gens }
  pairs ← newSTRef { Entry (calcWeight strategy f g, j) (f, g)
                   | ((i, f), (j, g)) ← combinations ideal }
  len ← newSTRef $ length ideal
  whileM_ (not . null <$> readSTRef pairs) $ do
    Just (Entry _ (f, g), rest) ← viewMin <$> readSTRef pairs
    -- 中略

      when (s ≠ zero) $ do
        pairs += { Entry (calcWeight strategy q s, j) (q, s)
                 | Entry _ q ← qs | j ← [len0 + 1 ..] }
        bases %= insert (Entry LM(s) s)
        len *= 2
  map payload . toList <$> readSTRef bases
```

重みを追加するだけで実装出来てしまった……

Benchmark

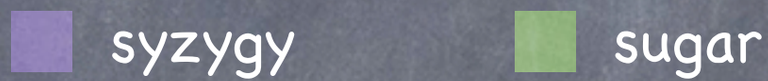
syzygy

sugar

Benchmark


syzygy
sugar
lex01
grlex01
grevlex01
lex03
grlex03
grevlex03
0ms
0.15ms
0.3ms
0.45ms
0.6ms
grlex02
grevlex02
16ms
17ms
18ms
19ms
20ms
grlex04
grevlex04
58µs
58.1µs
58.2µs
58.3µs
58.4µs

Benchmark（論文版）

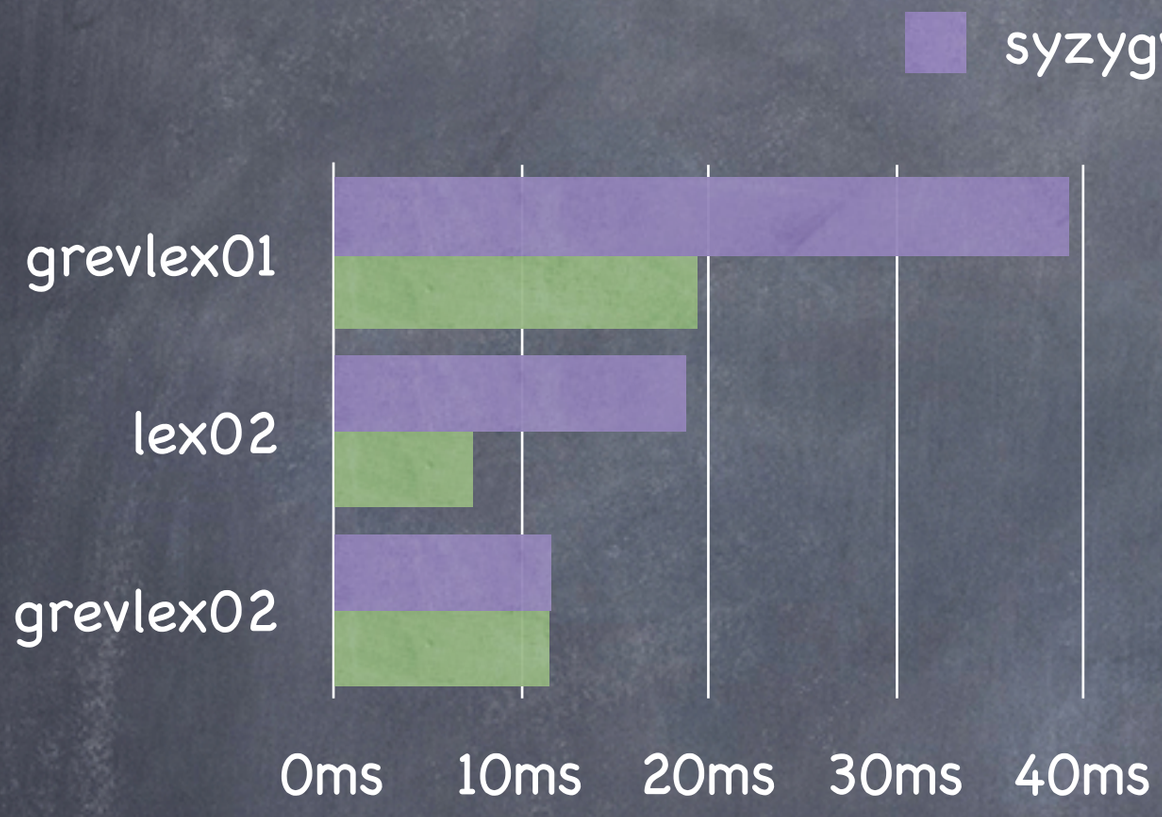


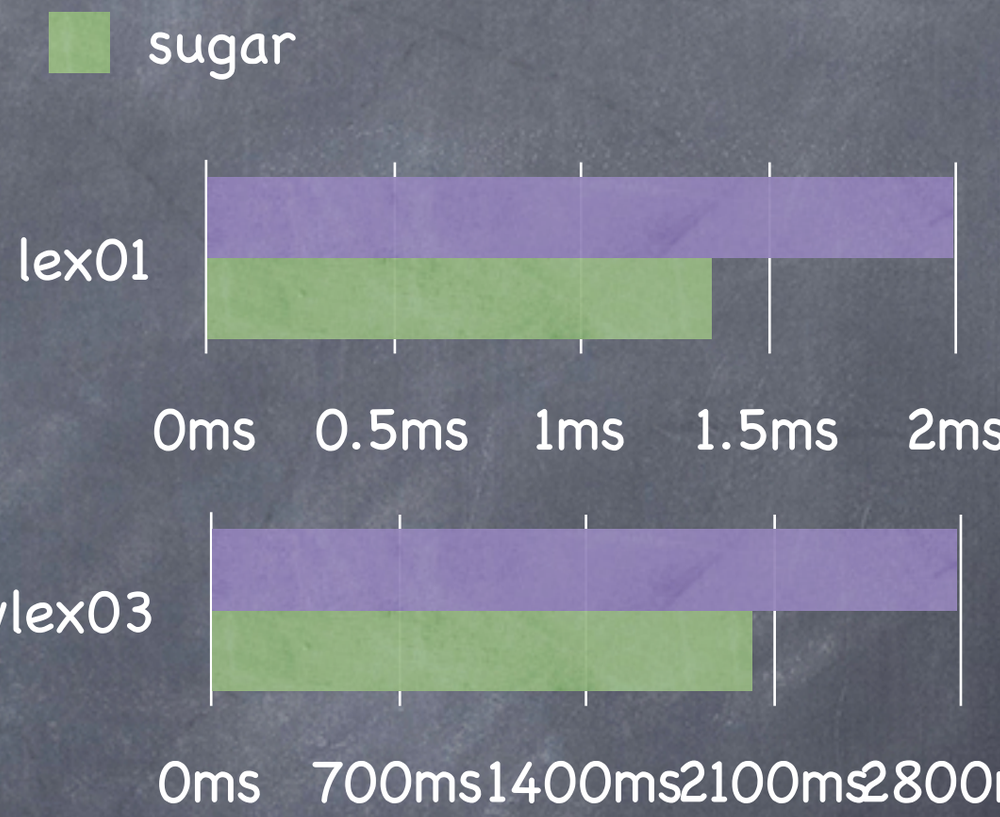
$$I_1 = \langle 35y^4 - 10xy^2 - 210y^2z + 3x^2 + 30xz - 105z^2 + 140yt - 21u,$$
$$5xy^3 - 140y^3z - 3x^2y + 45xyz - 420yz^2 + 210y^2t - 25xt + 70zt + 126yu\rangle$$

$$I_2 = \langle x + y + z + w, xy + yz + zw + wx, xyz + yzw + zwx + wxy, xyzw - 1\rangle$$

$$I_3 = \langle x^{31} - x^6 - x - y, x^8 - z, x^{10} - t\rangle$$

Benchmark（論文版）





$$I_1 = \langle 35y^4 - 10xy^2 - 210y^2z + 3x^2 + 30xz - 105z^2 + 140yt - 21u,$$
$$5xy^3 - 140y^3z - 3x^2y + 45xyz - 420yz^2 + 210y^2t - 25xt + 70zt + 126yu\rangle$$

$$I_2 = \langle x + y + z + w, xy + yz + zw + wx, xyz + yzw + zwx + wxy, xyzw - 1\rangle$$

$$I_3 = \langle x^{31} - x^6 - x - y, x^8 - z, x^{10} - t\rangle$$

考察

考察

- 論文に出てくる例については sugar が速い

考察

- 論文に出てくる例については sugar が速い
  - それでも驚くほど速いようには見えない

考察

- 論文に出てくる例については sugar が速い
  - それでも驚くほど速いようには見えない
- こちらで選んだデータセットに関しては sugar のほうが遅い場合もある？

考察

- 論文に出てくる例については sugar が速い
  - それでも驚くほど速いようには見えない
- こちらで選んだデータセットに関しては sugar のほうが遅い場合もある？
- 追跡調査予定

ここまでで質問？

Table of Contents

- Gröbner 基底の簡単な導入
- Gröbner 基底の計算法
- Buchberger アルゴリズム
- 最適化手法：syzygy, sugar strategy
- Haskell の型レベルプログラミングの現在

Haskell の
型レベルプログラミングの現在

Haskell の
型レベルプログラミングの現在

- computational-algebra では型機能をふんだんに利用している

Haskell の
型レベルプログラミングの現在

- computational-algebra では型機能をふんだんに利用している
  - 変数の数、体の種類、単項式順序、選択戦略...

Haskell の
型レベルプログラミングの現在

- computational-algebra では型機能をふんだんに利用している
  - 変数の数、体の種類、単項式順序、選択戦略...
  - 全て型にパラメトライズ

Haskell の
型レベルプログラミングの現在

- computational-algebra では型機能をふんだんに利用している
  - 変数の数、体の種類、単項式順序、選択戦略...
  - 全て型にパラメトライズ
  - 依存型と証明のオンパレード

依存型と証明のオンパレード

依存型と証明のオンパレード

- DataKinds や GADTs といった拡張機能を多用

依存型と証明のオンパレード

- DataKinds や GADTs といった拡張機能を多用
- 単項式は指数ベクトルで表現しているので、変数の数の区別をしっかりつけたい

依存型と証明のオンパレード

- DataKinds や GADTs といった拡張機能を多用
- 単項式は指数ベクトルで表現しているので、変数の数の区別をしっかりつけたい
- 長さつきベクトル！

依存型と証明のオンパレード

- DataKinds や GADTs といった拡張機能を多用
- 単項式は指数ベクトルで表現しているので、変数の数の区別をしっかりつけたい
- 長さつきベクトル！
- 境界条件も証明しよう！

依存型と証明のオンパレード

- DataKinds や GADTs といった拡張機能を多用
- 単項式は指数ベクトルで表現しているので、変数の数の区別をしっかりつけたい
- 長さつきベクトル！
- 境界条件も証明しよう！
  - これだけでもかなり大変なので、「アルゴリズムの妥当性」とかは示していないです（死ぬ）

詳しくはコード参照

```haskell
-- | N-ary Monomial. IntMap contains degrees for each x_i.
type Monomial (n :: Nat) = Vector Int n

-- | Monomial with monomial order.
newtype OrderedMonomial (ord :: ★) n =
    OrderedMonomial { getMonomial :: Monomial n }

deriving instance (Eq (Monomial n)) ⇒ Eq (OrderedMonomial ordering n)


type MonomialOrder = ∀n. Monomial n → Monomial n → Ordering

-- | Class to lookup ordering from its (type-level) name.
class IsOrder (ordering :: ★) where
  cmpMonomial :: Proxy ordering → MonomialOrder

-- | Class for Monomial orders.
class IsOrder name ⇒ IsMonomialOrder name where
```

詳しくはコード参照

```haskell
data Lex = Lex
data Revlex = Revlex
data Grevlex = Grevlex

lex :: MonomialOrder
lex Nil Nil = EQ
lex (x :- xs) (y :- ys) = x `compare` y <> xs `lex` ys
lex _ _ = error "cannot happen"

revlex :: Monomial n → Monomial n → Ordering
revlex (x :- xs) (y :- ys) = xs `revlex` ys <> y `compare` x
revlex Nil       Nil       = EQ
revlex _ _ = error "cannot happen!"

graded :: (Monomial n → Monomial n → Ordering) → (Monomial n → Monomial n →
Ordering)
graded cmp xs ys = comparing totalDegree xs ys <> cmp xs ys

instance IsOrder Grevlex where
  cmpMonomial _ = graded revlex

instance IsOrder Revlex where
  cmpMonomial _ = revlex
```

cannot happen!

cannot happen!

- 型レベルで禁止されるパターン

cannot happen!

- 型レベルで禁止されるパターン
- しかし、 -fwarn-incomplete-pattern を指定

すると「このパターン抜けてるよ！」

cannot happen!

- 型レベルで禁止されるパターン
- しかし、 -fwarn-incomplete-pattern を指定すると「このパターン抜けてるよ！」
- そのパターンを書くと型エラーで弾かれる

cannot happen!

- 型レベルで禁止されるパターン
- しかし、 -fwarn-incomplete-pattern を指定すると「このパターン抜けてるよ！」
- そのパターンを書くと型エラーで弾かれる
- だから wildcard pattern

イデアル hack

```haskell
data Ideal r = ∀n. Ideal (Vector r n)

instance Eq r ⇒ Eq (Ideal r) where
  (≡) = (≡) `on` generators

generators :: Ideal r → [r]
generators (Ideal is) = toList is

addToIdeal :: r → Ideal r → Ideal r
addToIdeal i (Ideal is) = Ideal (i :- is)

toIdeal :: NoetherianRing r ⇒ [r] → Ideal r
toIdeal = foldr addToIdeal (Ideal Nil)
```

イデアル hack

```haskell
data Ideal r = ∀n. Ideal (Vector r n)

instance Eq r ⇒ Eq (Ideal r) where
  (≡) = (≡) `on` generators

generators :: Ideal r → [r]
generators (Ideal is) = toList is

addToIdeal :: r → Ideal r → Ideal r
addToIdeal i (Ideal is) = Ideal (i :- is)

toIdeal :: NoetherianRing r ⇒ [r] → Ideal r
toIdeal = foldr addToIdeal (Ideal Nil)
```

- イデアルはリスト……ではない！

イデアル hack

```
data Ideal r = ∀n. Ideal (Vector r n)

instance Eq r ⇒ Eq (Ideal r) where
  (≡) = (≡) `on` generators

generators :: Ideal r → [r]
generators (Ideal is) = toList is

addToIdeal :: r → Ideal r → Ideal r
addToIdeal i (Ideal is) = Ideal (i :- is)

toIdeal :: NoetherianRing r ⇒ [r] → Ideal r
toIdeal = foldr addToIdeal (Ideal Nil)
```

- イデアルはリスト……ではない！
  - 非常に深遠なる理由により存在量化された sized-vector

イデアル hack

```haskell
data Ideal r = ∀n. Ideal (Vector r n)

instance Eq r ⇒ Eq (Ideal r) where
  (≡) = (≡) `on` generators

generators :: Ideal r → [r]
generators (Ideal is) = toList is

addToIdeal :: r → Ideal r → Ideal r
addToIdeal i (Ideal is) = Ideal (i :- is)

toIdeal :: NoetherianRing r ⇒ [r] → Ideal r
toIdeal = foldr addToIdeal (Ideal Nil)
```

- イデアルはリスト……ではない！
  - 非常に深遠なる理由により存在量化された sized-vector
  - 消去イデアルの計算の時に暗黙裏に型を合わせる為

辻褄を合わせに用意された

```haskell
data Monomorphic k = ∀a. Monomorphic (k a)

-- | A types which have the monomorphic representation.
class Monomorphicable k where
  -- | Monomorphic representation
  type MonomorphicRep k :: ★
  -- | Promote the monomorphic value to the polymophic one.
  promote :: MonomorphicRep k → Monomorphic k
  -- | Demote the polymorphic value to the monomorphic representation.
  demote  :: Monomorphic k → MonomorphicRep k

-- | Convinience function to demote polymorphic types
--   into monomorphic one directly.
demote' :: Monomorphicable k ⇒ k a → MonomorphicRep k
demote' = demote ∘ Monomorphic

-- | Demote polymorphic nested types directly into
--   monomorphic representation.
demoteComposed :: Monomorphicable (f :∘: g)
               ⇒ f (g a) → MonomorphicRep (f :∘: g)
demoteComposed = demote ∘ Monomorphic ∘ Comp
```

制約付けてる感じ

```
thEliminationIdeal :: ( IsMonomialOrder ord, Field k, IsPolynomial k m
                      , IsPolynomial k (m :-: n)
                      , (n ⩽ m) ~ True)
                   ⇒ SNat n
                   → Ideal (OrderedPolynomial k ord m)
                   → Ideal (OrderedPolynomial k ord (m ⊖ n))
thEliminationIdeal n =
  case singInstance n of
    SingInstance →
          mapIdeal (changeOrderProxy Proxy) ∘
            thEliminationIdealWith (weightedEliminationOrder n) n
```

制約付けてる感じ

```
thEliminationIdeal :: ( IsMonomialOrder ord, Field k, IsPolynomial k m
                      , IsPolynomial k (m :-: n)
                      , (n ⩽ m) ~ True)
                   ⇒ SNat n
                   → Ideal (OrderedPolynomial k ord m)
                   → Ideal (OrderedPolynomial k ord (m ⊖ n))
thEliminationIdeal n =
  case singInstance n of
    SingInstance →
          mapIdeal (changeOrderProxy Proxy) ∘
            thEliminationIdealWith (weightedEliminationOrder n) n
```

- Gröbner 基底を使うと、連立方程式から文字を消去出来る（すごい！）

制約付けてる感じ

```
thEliminationIdeal :: ( IsMonomialOrder ord, Field k, IsPolynomial k m
                      , IsPolynomial k (m :-: n)
                      , (n ⩽ m) ~ True)
                   ⇒ SNat n
                   → Ideal (OrderedPolynomial k ord m)
                   → Ideal (OrderedPolynomial k ord (m ⊖ n))
thEliminationIdeal n =
  case singInstance n of
    SingInstance →
          mapIdeal (changeOrderProxy Proxy) ∘
            thEliminationIdealWith (weightedEliminationOrder n) n
```

- Gröbner 基底を使うと、連立方程式から文字を消去出来る（すごい！）
- n 文字目までを消去したい⇒ 変数は最低でも n 個

証明してる感じ

```
intersection :: ∀ r k n ord.
                ( IsMonomialOrder ord, Field r, IsPolynomial r k, IsPolynomial r n
                , IsPolynomial r (k ⊕ n)
                )
             ⇒ Vector (Ideal (OrderedPolynomial r ord n)) k
             → Ideal (OrderedPolynomial r ord n)
intersection Nil = Ideal $ singletonV one
intersection idsv@(_ :- _) =
    let sk = sLengthV idsv
        sn = sing :: SNat n
        ts  = genVars (sk %+ sn)
        tis = zipWith (λ ideal t → mapIdeal ((t *) . shiftR sk) ideal)
                      (toList idsv) ts
        j = foldr appendIdeal (principalIdeal (one - foldr (+) zero ts)) tis
    in case plusMinusEqR sn sk of
         Eql → case propToBoolLeq (plusLeqL sk sn) of
                 LeqTrueInstance → thEliminationIdeal sk j
```

今気付いたけど

全部説明するの無理だ

もうだめだ

詳しくは直接捕まえて

訊いてください

最後にいいたいこと

最後にいいたいこと

- 帰納法はコストがかかる

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない
- シングルトンもコストがかかる

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない
- シングルトンもコストがかかる
- Haskell での依存型で実用的なアプリ書くのはむずい

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない
- シングルトンもコストがかかる
  - Haskell での依存型で実用的なアプリ書くのはむずい
- HP 標準添付の GHC 7.4.* はクソ

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない
- シングルトンもコストがかかる
  - Haskell での依存型で実用的なアプリ書くのはむずい
- HP 標準添付の GHC 7.4.* はクソ
  - コンパイルする度に rm *.hi *.o しないと型機能が使えない

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない
- シングルトンもコストがかかる
  - Haskell での依存型で実用的なアプリ書くのはむずい
- HP 標準添付の GHC 7.4.* はクソ
  - コンパイルする度に rm *.hi *.o しないと型機能が使えない
  - 種多相な外部ライブラリとリンク出来ない

最後にいいたいこと

- 帰納法はコストがかかる
  - 証明は実行時に遅延されるからタダじゃない
- シングルトンもコストがかかる
  - Haskell での依存型で実用的なアプリ書くのはむずい
- HP 標準添付の GHC 7.4.* はクソ
  - コンパイルする度に rm *.hi *.o しないと型機能が使えない
  - 種多相な外部ライブラリとリンク出来ない
- 案外型の辻褄はあう

まとめ

まとめ

- Gröbner基底を使うと代数の計算が出来るように

まとめ

- Gröbner基底を使うと代数の計算が出来るように
  - イデアル商とか共通部分とか

まとめ

- Gröbner基底を使うと代数の計算が出来るように
  - イデアル商とか共通部分とか
- 文字消去、連立方程式の解

まとめ

- Gröbner基底を使うと代数の計算が出来るように
  - イデアル商とか共通部分とか
  - 文字消去、連立方程式の解
- 高速化手法が幾つかあって、切磋琢磨

まとめ

- Gröbner基底を使うと代数の計算が出来るように
  - イデアル商とか共通部分とか
    - 文字消去、連立方程式の解
- 高速化手法が幾つかあって、切磋琢磨
- Haskell 型システムの今後に期待

参考文献

- D. Cox, J. Little and D. O'Shea. Ideals, Varieties and Algorithms. UTM. Springer.
- A. Giovini, T. Moran, G. Niesi, L. Robbiano and C. Traverse. "One sugar cube, please" OR Selection strategies in the Buchberger algorithm.
- 楫元. グレブナー基底は面白い！. 講義録.
- 2012年度早稲田大学基幹理工学部数学科代数学C1 講義資料

Any Questions?

御清聴

ありがとうございました