

---

**Universidade Federal de Uberlândia**

Faculdade de Matemática

---

Bruno Félix Rezende Ribeiro

**Estudo e Implementação do Algoritmo de  
Buchberger na linguagem UserRPL para a  
Calculadora Gráfica HP 50g**

Uberlândia - MG

2018

Bruno Félix Rezende Ribeiro

**Estudo e Implementação do Algoritmo de  
Buchberger na linguagem UserRPL para a  
Calculadora Gráfica HP 50g**

Monografia apresentada à Faculdade de Matemática da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de Bacharel em Matemática, sob a orientação do Prof. Dr. Victor Gonzalo Lopez Neumann.

Uberlândia - MG

2018

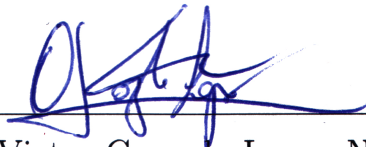
Bruno Félix Rezende Ribeiro

# Estudo e Implementação do Algoritmo de Buchberger na linguagem UserRPL para a Calculadora Gráfica HP 50g

Monografia apresentada à Faculdade de Matemática da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de Bacharel em Matemática, sob a orientação do Prof. Dr. Victor Gonzalo Lopez Neumann.

Aprovada em 13/12/2018

## BANCA EXAMINADORA



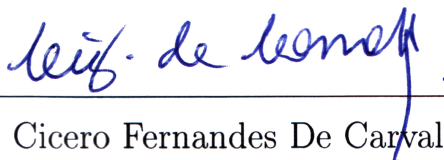
---

Prof. Dr. Victor Gonzalo Lopez Neumann (orientador)



---

Prof. Dr. Neiton Pereira da Silva



---

Prof. Dr. Cicero Fernandes De Carvalho

# Agradecimentos

Agradeço a todos os matemáticos que fundamentaram as teorias sobre as quais o presente trabalho se baseia, em particular Bruno Buchberger pelo brilhantismo de sua inovação matemática teórica e computacional.

Agradeço aos engenheiros da Hewlett Packard responsáveis pelo desenvolvimento da série de calculadoras baseadas na linguagem UserRPL, cuja destreza técnica possibilitou tanto divertimento na programação do modelo 50g.

Agradeço a todos os professores que me ensinaram as habilidades necessárias para a produção do presente trabalho, em particular meu orientador Victor Neumann por sua grande competência em álgebra comutativa e generoso apoio frente às dificuldades.

Agradeço a todos os entes queridos que me deram suporte emocional para a conclusão deste trabalho, em particular minha futura esposa Maiane Ribeiro pelo seu amor imensurável.

Por último, mas não menos importante, agradeço ao  $\emptyset$  por tudo existir.

# Resumo

Neste trabalho estuda-se os conceitos e resultados algébricos que fundamentam a teoria das bases de Gröbner e o algoritmo de Buchberger que possibilita o cálculo efetivo destas. A teoria é estabelecida com respeito a um anel polinomial em um número arbitrário de variáveis com coeficientes sobre um corpo qualquer. Dá-se ênfase aos assuntos relativos à ordem monomial, algoritmo da divisão, ideais monomiais, lema de Dickson e teorema da base de Hilbert. A título de curiosidade, desenvolve-se uma implementação do algoritmo de Buchberger como descrito originalmente em [Buchberger, 1985] na linguagem UserRPL para a Calculadora Gráfica HP 50g e explora-se suas propriedades técnicas e computacionais. Esta implementação provê suporte para polinômios com coeficientes complexos em um número arbitrário de variáveis com ordenações monomiais canônicas pré-definidas e também programáveis pelo usuário.

**Palavras-Chave:** Álgebra Computacional. Bases de Gröbner. Algoritmo de Buchberger. UserRPL. HP 50g.

# Sumário

<b>Introdução</b>	<b>8</b>
<b>1 Anéis, Corpos e Polinômios</b>	<b>9</b>
<b>2 Ordem Monomial e Algoritmo da Divisão</b>	<b>12</b>
<b>3 Lema de Dickson e Teorema da Base de Hilbert</b>	<b>19</b>
<b>4 Bases de Gröbner</b>	<b>24</b>
<b>5 Algoritmo de Buchberger</b>	<b>30</b>
<b>6 UserRPL</b>	<b>32</b>
6.1 Objetos . . . . .	33
6.2 Estruturas . . . . .	34
6.3 Comandos . . . . .	37
6.3.1 Pilha . . . . .	37
6.3.2 Lógico . . . . .	38
6.3.3 Comparação . . . . .	39
6.3.4 Aritmética . . . . .	40
6.3.5 Sinalizadores . . . . .	41
6.3.6 Sequências . . . . .	41
6.3.7 Nomes e Algébricos . . . . .	43
6.3.8 Diversos . . . . .	44
6.4 Núcleo UserRPL . . . . .	45
6.4.1 Biblioteca Base . . . . .	45

6.4.2	Biblioteca Polinomial . . . . .	51
<b>7</b>	<b>Implementação do Algoritmo de Buchberger</b>	<b>61</b>
7.1	Exemplos . . . . .	64
7.1.1	O problema da pertinência a um ideal . . . . .	64
7.1.2	O problema de resolver equações polinomiais . . . . .	65
7.1.3	O problema da implicitação . . . . .	68
<b>8</b>	<b>Referências Bibliográficas</b>	<b>70</b>

# Introdução

Dentro do ramo da álgebra comutativa e geometria algébrica computacional uma *base de Gröbner* é um tipo particular de conjunto gerador de um ideal dentro de um anel polinomial sobre um corpo qualquer. O conceito de bases de Gröbner é fundamental para a dedução de várias propriedades importantes de um ideal e sua variedade algébrica associada, como dimensão e número de zeros. O *algoritmo de Buchberger* permite o cálculo efetivo das bases de Gröbner e é uma das principais ferramentas em sistemas algébricos computacionais para a resolução de sistemas de equações polinomiais e a computação de imagens de variedades algébricas sob projeções e mapeamento racional. Este algoritmo pode ser visto como uma generalização multivariável não linear do algoritmo euclidiano para o cálculo do máximo divisor comum de polinômios e da eliminação gaussiana para sistemas lineares. A teoria das bases de Gröbner, junto com o algoritmo de Buchberger, foram introduzidas por Bruno Buchberger em sua tese de doutorado, sendo o nome dado às bases homenagem ao seu orientador Wolfgang Gröbner [Buchberger, 1985].

Na primeira parte deste trabalho estuda-se os conceitos e resultados algébricos que fundamentam a teoria das bases de Gröbner e o algoritmo de Buchberger que possibilita o cálculo efetivo destas. A teoria é estabelecida com respeito a um anel polinomial em um número arbitrário de variáveis com coeficientes sobre um corpo qualquer. Dá-se ênfase aos assuntos relativos à ordem monomial, algoritmo da divisão, ideais monomiais, lema de Dickson e teorema da base de Hilbert. Na segunda parte do presente trabalho desenvolve-se uma implementação do algoritmo de Buchberger como descrito originalmente em [Buchberger, 1985] na linguagem UserRPL para a Calculadora Gráfica HP 50g e explora-se suas propriedades técnicas e computacionais.



# Capítulo 1

## Anéis, Corpos e Polinômios

Neste capítulo revisaremos os conceitos pertinentes aos anéis, corpos e polinômios que fundamentam a teoria estudada.

**Definição 1.1** (Anel comutativo). *Sejam  $A$  um conjunto não-vazio, e as operações  $+$  e  $\cdot$  definidas sobre  $A$ . A terna  $(A, +, \cdot)$  é um **anel comutativo** se satisfaz as seguintes propriedades:*

**(Associatividade)** *Para todos  $a, b$  e  $c$  em  $A$ :*

- $(a + b) + c = a + (b + c)$
- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

**(Comutatividade)** *Para todos  $a$  e  $b$  em  $A$ :*

- $a + b = b + a$
- $a \cdot b = b \cdot a$

**(Distributividade)** *Para todos  $a, b$  e  $c$  em  $A$ :*

- $a \cdot (b + c) = a \cdot b + a \cdot c$

**(Identidade)** *Para todo  $a$  em  $A$ , existem  $0$  e  $1$  em  $A$  tais que:*

- $a + 0 = a$
- $a \cdot 1 = a$

**(Inverso aditivo)** Para todo  $a$  em  $A$ , existe  $b$  em  $A$  tal que:

- $a + b = 0$

Quando as operações aditiva e multiplicativa estão subentendidas, não ocorrendo risco de ambiguidade, identifica-se a terna  $(A, +, \cdot)$  com o próprio conjunto  $A$ , chamando-se a este de “anel”.

**Definição 1.2** (Corpo). Um anel comutativo  $(A, +, \cdot)$  é um **corpo** se satisfaz a seguinte propriedade:

**(Inverso multiplicativo)** Para todo  $a \neq 0$  em  $A$ , existe  $b$  em  $A$  tal que:

- $a \cdot b = 1$

Quando as operações aditiva e multiplicativa estão subentendidas, não ocorrendo risco de ambiguidade, identifica-se a terna  $(F, +, \cdot)$  com o próprio conjunto  $F$ , chamando-se a este de “corpo”.

**Definição 1.3** (Monômio). Um **monômio** nas variáveis  $x_1, \dots, x_n$  é uma expressão algébrica da forma  $\prod_{i=1}^n x_i^{\alpha_i}$ , onde  $\alpha_i \in \mathbb{N}$  para  $1 \leq i \leq n$ .

**Notação.** Denota-se  $\prod_{i=1}^n x_i^{\alpha_i}$  por  $x^\alpha$ , onde  $x = (x_1, \dots, x_n)$  e  $\alpha = (\alpha_1, \dots, \alpha_n)$ .

**Definição 1.4** (Grau total de monômio). O **grau total de um monômio**  $f = x^\alpha$  é definido como  $\text{grau}(f) := \sum_{i=1}^n \alpha_i$ .

**Definição 1.5** (Polinômio). Um **polinômio** nas variáveis  $x_1, \dots, x_n$  com coeficientes sobre um corpo  $F$  é uma expressão algébrica da forma  $\sum_{\alpha \in A} a_\alpha x^\alpha$ , onde  $A$  é um subconjunto finito de  $\mathbb{N}^n$ , e  $a_\alpha \in F$  para todo  $\alpha \in A$ .

**Notação.**

- Quando não há risco de ambiguidade nem necessidade de explicitação, costuma-se omitir o conjunto  $A$  na notação da expressão algébrica geral de um polinômio (como acima) por:

$$\sum_{\alpha} a_{\alpha} x^{\alpha};$$

- Denota-se o conjunto de todos os polinômios nas variáveis  $x_1, \dots, x_n$  com coeficientes sobre um corpo  $F$  por  $F[x_1, \dots, x_n]$ ;

**Definição 1.6** (Grau total de polinômio). O *grau total de um polinômio*  $f = \sum_{\alpha} a_{\alpha} x^{\alpha} \neq 0$  é definido como  $\text{grau}(f) := \max \{ \text{grau}(x^{\alpha}) \mid a_{\alpha} \neq 0 \}$ .

**Definição 1.7** (Ideal). Um subconjunto  $I \subset F[x_1, \dots, x_n]$  é um **ideal** se satisfaz:

(i)  $0 \in I$ ;

(ii) Se  $f, g \in I$ , então  $f + g \in I$ ;

(iii) Se  $f \in I$  e  $h \in F[x_1, \dots, x_n]$ , então  $h \cdot f \in I$ ;

Note que o menor ideal possível, chamado de “**ideal trivial**”, é  $\{0\}$ .

**Notação.** Sejam  $f_1, \dots, f_s$  polinômios em  $F[x_1, \dots, x_n]$ . Denota-se

$$\langle f_1, \dots, f_s \rangle := \left\{ \sum_{i=1}^s h_i f_i \mid h_1, \dots, h_s \in F[x_1, \dots, x_n] \right\}.$$

**Lema 1.8** (Ideal gerado). Se  $f_1, \dots, f_s \in F[x_1, \dots, x_n]$ , então  $\langle f_1, \dots, f_s \rangle$  é um ideal de  $F[x_1, \dots, x_n]$ , que chamamos de **ideal gerado** por  $f_1, \dots, f_s$ .

*Demonstração.* Primeiramente,  $0 \in \langle f_1, \dots, f_s \rangle$  visto que  $0 = \sum_{i=1}^s 0 \cdot f_i$ . Agora, suponha que  $f = \sum_{i=1}^s p_i f_i$  e  $g = \sum_{i=1}^s q_i f_i$ , e seja  $h \in F[x_1, \dots, x_n]$ . Então as equações,

$$f + g = \sum_{i=1}^s (p_i + q_i) f_i$$

$$hf = \sum_{i=1}^s (hp_i) f_i$$

completam a prova de que  $\langle f_1, \dots, f_s \rangle$  é um ideal. □

# Capítulo 2

## Ordem Monomial e Algoritmo da Divisão

**Definição 2.1** (Relação). Uma **relação** sobre um conjunto  $M$  é um subconjunto de  $M^2 = M \times M$ .

**Notação.** Denota-se  $(f, g) \in \geq$  por  $f \geq g$ .

**Definição 2.2** (Ordem). **Ordem** sobre um conjunto  $M$  é uma relação  $\geq$  tal que para todo  $f, g$  e  $h$  em  $M$  satisfaz:

(Reflexiva)  $f \geq f$ ;

(Anti-simétrica) Se  $f \geq g$  e  $g \geq f$ , então  $f = g$ ;

(Transitiva) Se  $f \geq g$  e  $g \geq h$ , então  $f \geq h$ ;

**Notação.** Denota-se:

- $f \geq g$  por  $g \leq f$ ;
- $f \geq g$  e  $f \neq g$  por  $f > g$ ;
- $f > g$  por  $g < f$ ;

**Definição 2.3** (Ordem total). **Ordem total** é uma ordem  $\geq$  sobre um conjunto  $M$  tal que para todo  $f$  e  $g$  em  $M$  verifica-se exclusivamente um dos seguintes casos:

- $f > g$ ;
- $f = g$ ;

- $f < g$ ;

A ordem usual dos números reais é uma ordem total, pois para quaisquer dois elementos vale a tricotomia.

**Definição 2.4** (Boa ordem). *Boa ordem* é uma ordem  $\geq$  sobre um conjunto  $M$  em que para todo  $X \subset M$  não vazio, existe  $f \in X$  tal que para todo  $g \in X$ , tem-se  $f \leq g$ .

**Proposição 2.5.** *Uma ordem  $\geq$  sobre um conjunto  $M$  é boa ordem se, e só se, toda sequência estritamente decrescente em  $M$  é finita.*

*Demonstração.* Demonstremos a contrapositiva: uma ordem  $\geq$  sobre um conjunto  $M$  não é boa ordem se, e só se, existe uma sequência infinita estritamente decrescente em  $M$ .

$\Rightarrow$  Seja  $S$  um subconjunto de  $M$  sem menor elemento. Tome  $f_0$  em  $S$ . Como  $f_0$  não é o menor elemento, existe  $f_1$  em  $S$  tal que  $f_1 < f_0$ . Para cada  $i \in \mathbb{N}$  maior que 0, como  $f_i \in M$  não é o menor elemento, tome  $f_{i+1}$  tal que  $f_{i+1} < f_i$ . Portanto, temos a sequência infinita estritamente decrescente  $\dots < f_{i+1} < f_i < \dots < f_1 < f_0$ .

$\Leftarrow$  Seja  $\dots < f_{i+1} < f_i < \dots < f_1 < f_0$  uma sequência infinita estritamente decrescente de elementos de  $M$ . Considere o conjunto  $S$  formado exatamente por estes elementos. Como  $S \subset M$  é não vazio e não possui menor elemento,  $\geq$  não é boa ordem.

□

**Definição 2.6** (Ordem monomial). *Ordem monomial* é uma ordem  $\geq$  sobre o conjunto  $M$  dos monômios de  $F[x_1, \dots, x_n]$ , que satisfaz:

- (i)  $\geq$  é ordem total;
- (ii)  $\geq$  é boa ordem;
- (iii) Dados  $f$  e  $g$  em  $M$ , se  $f \geq g$  e  $h \in M$ , então  $f \cdot h \geq g \cdot h$ ;

Observe que toda ordem monomial  $\geq$  induz uma ordem  $\geq'$  sobre o conjunto  $\mathbb{N}^n$  (e vice-versa) tal que dados  $\alpha, \beta \in \mathbb{N}^n$ , vale  $\alpha \geq' \beta \iff x^\alpha \geq x^\beta$ . Portanto  $\geq'$  goza das mesmas propriedades que definem  $\geq$  acima, e logo, quando não há risco de ambiguidade usa-se ambas intercambiavelmente.

**Definição 2.7** (Ordem lexicográfica). *Ordem lexicográfica* é uma ordem  $\geq_{\text{lex}}$  sobre o conjunto dos monômios de  $F[x_1, \dots, x_n]$ , que para todo  $f = x^\alpha$  e  $g = x^\beta$  satisfaz  $f \geq_{\text{lex}} g$  se, e só se, verifica-se uma das seguintes condições:

- $f = g$ ;
- Senão, sendo  $m = \min \{i \mid 1 \leq i \leq n \text{ e } \alpha_i \neq \beta_i\}$ , tem-se  $\alpha_m > \beta_m$ ;

Por exemplo,  $(7, 9, 1) \geq_{\text{lex}} (7, 1, 9)$  e  $(8, 0, 5) \geq_{\text{lex}} (8, 0, 5)$ .

**Proposição 2.8.** *A ordem lexicográfica  $\geq_{\text{lex}}$  é uma ordem monomial sobre o conjunto dos monômios de  $F[x_1, \dots, x_n]$ .*

*Demonstração.* (i) Sejam  $f = x^\alpha$  e  $g = x^\beta$ . Se  $f = g$  o resultado é imediato. Do contrário, seja  $m = \min \{i \mid 1 \leq i \leq n \text{ e } \alpha_i \neq \beta_i\}$ . Caso  $\alpha_m > \beta_m$ , segue da definição de ordem lexicográfica que  $f \geq_{\text{lex}} g$ . Senão teremos  $\beta_m > \alpha_m$  e logo, por definição,  $g \geq_{\text{lex}} f$ .

(ii) Suponha por absurdo que  $\geq_{\text{lex}}$  não seja boa ordem. Pela proposição 2.5 temos que então existe uma sequência infinita estritamente decrescente de monômios em  $F[x_1, \dots, x_n]$ . Tome  $f_0 = x^\alpha$  e  $f_1 = x^\beta$  em tal sequência de forma que  $f_1 <_{\text{lex}} f_0$ . Seja  $m_0 = \min \{i \mid 1 \leq i \leq n \text{ e } \alpha_i \neq \beta_i\}$  que satisfaz  $\alpha_{m_0} > \beta_{m_0}$ . Tome  $f_2 = x^\gamma$  tal que  $f_2 <_{\text{lex}} f_1$  e seja  $m_1 = \min \{i \mid 1 \leq i \leq n \text{ e } \beta_i \neq \gamma_i\}$ . Logo,  $m_1 < m_0$  ou  $\gamma_{m_1} < \beta_{m_1}$ . Vemos então que repetindo o processo e tomando um monômio menor a cada passo, tem-se que ou o índice ordinal  $m$  do expoente decresce ou o próprio expoente decresce. Como ambos são números naturais a sequência eventualmente termina, o que é absurdo.

(iii) Sejam  $f = x^\alpha$ ,  $g = x^\beta$  e  $h = x^\gamma$  monômios de  $F[x_1, \dots, x_n]$  tais que  $f \geq_{\text{lex}} g$ . Seja  $m = \min \{i \mid 1 \leq i \leq n \text{ e } \alpha_i \neq \beta_i\}$ . Observe que  $f \cdot h = x^\alpha x^\gamma = x^{\alpha+\gamma}$  e  $g \cdot h = x^\beta x^\gamma = x^{\beta+\gamma}$ . Visto que  $\alpha_m > \beta_m$ , então  $\alpha_m + \gamma_m > \beta_m + \gamma_m$ , donde  $f \cdot h \geq_{\text{lex}} g \cdot h$ .

□

**Definição 2.9** (Ordem lexicográfica graduada). *Ordem lexicográfica graduada* é uma ordem  $\geq_{\text{grlex}}$  sobre o conjunto dos monômios de  $F[x_1, \dots, x_n]$ , que para todo  $f$  e  $g$  satisfaz  $f \geq_{\text{grlex}} g$  se, e só se, verifica-se uma das seguintes condições:

- $\text{grau}(f) > \text{grau}(g)$ ;

- Senão, se  $\text{grau}(f) = \text{grau}(g)$  então  $f \geq_{\text{lex}} g$ ;

Por exemplo,  $(1, 2, 3) \geq_{\text{grlex}} (3, 2, 0)$  e  $(1, 2, 4) \geq_{\text{grlex}} (1, 1, 5)$ .

**Definição 2.10** (Ordem lexicográfica reversa graduada). *Ordem lexicográfica reversa graduada* é uma ordem  $\geq_{\text{grevlex}}$  sobre o conjunto dos monômios de  $F[x_1, \dots, x_n]$ , que para todo  $f = x^\alpha$  e  $g = x^\beta$  satisfaz  $f \geq_{\text{grevlex}} g$  se, e só se, verifica-se uma das seguintes condições:

- $\text{grau}(f) > \text{grau}(g)$ ;
- Senão, se  $\text{grau}(f) = \text{grau}(g)$  então sendo  $m = \max\{i \mid 1 \leq i \leq n \text{ e } \alpha_i \neq \beta_i\}$ , tem-se  $\alpha_m < \beta_m$ ;

Por exemplo,  $(4, 7, 1) \geq_{\text{grevlex}} (4, 2, 3)$  e  $(1, 5, 2) \geq_{\text{grevlex}} (4, 1, 3)$ . Considere o polinômio  $f = 4xy^2z + 4z^2 - 5x^3 + 7x^2z^2 \in \mathbb{C}[x, y, z]$ . Ordenando seus termos em ordem decrescente com respeito às três ordens definidas anteriormente, temos:

$$(\geq_{\text{lex}}) f = -5x^3 + 7x^2z^2 + 4xy^2z + 4z^2;$$

$$(\geq_{\text{grlex}}) f = 7x^2z^2 + 4xy^2z - 5x^3 + 4z^2;$$

$$(\geq_{\text{grevlex}}) f = 4xy^2z + 7x^2z^2 - 5x^3 + 4z^2.$$

**Definição 2.11.** Sejam  $f = \sum_{\alpha} a_{\alpha}x^{\alpha} \neq 0$  um polinômio de  $F[x_1, \dots, x_n]$  e  $\geq$  uma ordem monomial. Define-se sobre  $f$ :

$$\text{(Multigráu)} \quad \text{multigráu}(f) := \max_{\geq} \{\alpha \in \mathbb{N}^n \mid a_{\alpha} \neq 0\}$$

$$\text{(Coeficiente líder)} \quad \text{CL}(f) := a_{\text{multigráu}(f)}$$

$$\text{(Monômio líder)} \quad \text{ML}(f) := x^{\text{multigráu}(f)}$$

$$\text{(Termo líder)} \quad \text{TL}(f) = \text{CL}(f) \cdot \text{ML}(f)$$

Para exemplificar, sejam  $f = 4xy^2z + 4z^2 - 5x^3 + 7x^2z^2$  e  $\geq$  a ordem lexicográfica. Temos então:

$$\text{multigrau}(f) = (3, 0, 0),$$

$$\text{CL}(f) = -5,$$

$$\text{ML}(f) = x^3,$$

$$\text{TL}(f) = -5x^3.$$

**Proposição 2.12.** *Sejam  $f$  e  $g$  polinômios não nulos de  $F[x_1, \dots, x_n]$ . Então:*

(i)  $\text{multigrau}(f \cdot g) = \text{multigrau}(f) + \text{multigrau}(g)$ ;

(ii) *Se  $f + g \neq 0$ , então  $\text{multigrau}(f + g) \leq \max\{\text{multigrau}(f), \text{multigrau}(g)\}$ . Além disso, se  $\text{multigrau}(f) \neq \text{multigrau}(g)$  a igualdade ocorre.*

A demonstração desta proposição é deixada como exercício para o leitor.

**Notação.** *Dados  $f, g \in F[x_1, \dots, x_n]$ ,*

- *Denota-se a existência de  $h \in F[x_1, \dots, x_n]$  tal que  $g = f \cdot h$ , por “ $f$  divide  $g$ ” ou  $f \parallel g$ ;*
- *Do contrário escreve-se “ $f$  não divide  $g$ ” ou  $f \nparallel g$ .*

**Teorema 2.13.** *Seja  $\geq$  uma ordem monomial e  $D = (f_1, \dots, f_s)$  uma  $s$ -upla de polinômios em  $F[x_1, \dots, x_n]$ . Então, qualquer  $f \in F[x_1, \dots, x_n]$  pode ser escrito como  $f = \sum_{i=1}^s q_i f_i + r$ , onde  $q_i, r \in F[x_1, \dots, x_n]$ , e  $r = 0$  ou  $\text{TL}(f_1), \dots, \text{TL}(f_s)$  não dividem os termos de  $r$ . Além disso, se  $q_i f_i \neq 0$ , então  $\text{multigrau}(f) \geq \text{multigrau}(q_i f_i)$ , para  $1 \leq i \leq s$ . Nessas condições, o polinômio  $r$  é chamado “**resto** da divisão de  $f$  por  $D$ ”.*

*Demonstração.* Considere o seguinte algoritmo:

```

1  ALGORÍTMO Divisão( $f, \{f_1, \dots, f_s\}$ )
2  |   PARA  $i \in \{1, \dots, s\}$ 
3  |   |    $q_i := 0$ 
4  |   |    $r := 0$ 
5  |   |    $p := f$ 
6  |   |   ENQUANTO  $p \neq 0$ 
7  |   |   |    $i := 1$ 
8  |   |   |    $d := 0$ 
9  |   |   |   ENQUANTO  $i \leq s$  E  $d = 0$ 
10 |   |   |   |   SE  $\text{TL}(f_i) \parallel \text{TL}(p)$ 
11 |   |   |   |   |    $q_i := q_i + \frac{\text{TL}(p)}{\text{TL}(f_i)}$ 
12 |   |   |   |   |    $p := p - \frac{\text{TL}(p)}{\text{TL}(f_i)} \cdot f_i$ 
13 |   |   |   |    $d = 1$ 

```



```

14 | | | SENÃO
15 | | |    $i := i + 1$ 
16 | | | SE  $d = 0$ 
17 | | |    $r := r + \text{TL}(p)$ 
18 | | |    $p := p - \text{TL}(p)$ 
19 | | | RETORNE  $\{q_1, \dots, q_s\}, r$ 

```

Afirmamos que  $q_1, \dots, q_s$  e  $r$  são construídos pelo algoritmo acima, que opera corretamente para qualquer entrada. Para provar que o algoritmo funciona, primeiramente mostremos que vale

$$f = \sum_{i=1}^s q_i f_i + r + p \quad (2.1)$$

para cada passo do algoritmo. Isso é claramente verdade para os valores iniciais de  $q_1, \dots, q_s, r, p$ . Agora, suponha que (2.1) valha para um dado passo do algoritmo. Destacam-se as duas situações mutuamente exclusivas que podem ocorrer quando o laço ENQUANTO mais externo é executado:

**(Passo de divisão)** Se para algum  $i \in \{1, \dots, s\}$  tem-se  $\text{TL}(f_i) \parallel \text{TL}(p)$ ;

**(Passo de resto)** Do contrário;

Se o próximo passo é de *divisão*, a igualdade  $q_i f_i + p = \left(q_i + \frac{\text{TL}(p)}{\text{TL}(f_i)}\right) f_i + \left(p - \frac{\text{TL}(p)}{\text{TL}(f_i)} f_i\right)$  mostra que  $q_i f_i + p$  não muda. Visto que todas as outras variáveis mantêm seus respectivos valores, a igualdade (2.1) permanece verdadeira. Por outro lado, se o próximo passo é de *resto*, então as variáveis  $p$  e  $r$  mudarão, mas sua soma  $p + r$  não, dado que  $p + r = (p - \text{TL}(p)) + (r + \text{TL}(p))$ , e logo a igualdade (2.1) é preservada. Agora, observe que o algoritmo pára quando  $p = 0$ . Neste caso, a igualdade (2.1) se torna  $f = \sum_{i=1}^s q_i f_i + r$ . Visto que termos são adicionados a  $r$  apenas quando não são divisíveis por nenhum dos  $\text{TL}(f_i)$ , segue que  $q_1, \dots, q_s$  e  $r$  tem as propriedades desejadas assim que o algoritmo termina.

Finalmente, precisamos provar que o algoritmo eventualmente termina. A principal observação é que cada vez que um valor é atribuído à variável  $p$ , ou seu multigráu diminui ou se torna 0. Para elucidar este fato, suponha que durante um passo de divisão,  $p$  tenha o valor  $p' = p - \frac{\text{TL}(p)}{\text{TL}(f_i)} f_i$  atribuído a si. Pela proposição 2.12, temos  $\text{TL}\left(\frac{\text{TL}(p)}{\text{TL}(f_i)} f_i\right) = \frac{\text{TL}(p)}{\text{TL}(f_i)} \text{TL}(f_i) = \text{TL}(p)$ , o que significa que  $p$  e  $\frac{\text{TL}(p)}{\text{TL}(f_i)} f_i$  tem o mesmo termo líder. Portanto, caso  $p' \neq 0$ , a diferença  $p'$  de ambos deve ter multigráu estritamente menor. Agora, suponha que durante um passo de resto,  $p$  assumo o valor  $p' = p - \text{TL}(p)$ . Neste caso é óbvio que  $\text{multigráu}(p') < \text{multigráu}(p)$ ,

se  $p' \neq 0$ . Portanto, em qualquer caso, o multigrau decresce. Se o algoritmo nunca terminasse, então obteríamos uma sequência decrescente infinita de monômios associados aos multigraus de  $p$ , o que não pode ocorrer, pela proposição 2.5, pois  $\geq$  é boa ordem.

Resta estudar a relação entre  $\text{multigrau}(f)$  e  $\text{multigrau}(q_i f_i)$ . Todo termo em  $q_i$  é da forma  $\frac{\text{TL}(p)}{\text{TL}(f_i)}$  para algum valor da variável  $p$ . O algoritmo começa com  $p = f$ , e acabamos de mostrar que o multigrau de  $p$  decresce. Isso mostra que  $\text{TL}(p) \leq \text{TL}(f)$ , e então segue facilmente, usando a condição (iii) da definição 2.6, que  $\text{multigrau}(q_i f_i) \leq \text{multigrau}(f)$ , quando  $q_i f_i \neq 0$ .  $\square$

# Capítulo 3

## Lema de Dickson e Teorema da Base de Hilbert

**Definição 3.1** (Ideal monomial). *Um ideal  $I \subset F[x_1, \dots, x_n]$  é um **ideal monomial** se existe um subconjunto  $A \subset \mathbb{N}^n$  tal que  $I$  consiste de todos os polinômios que são somas finitas da forma  $\sum_{\alpha \in A} h_\alpha x^\alpha$ , onde  $h_\alpha \in F[x_1, \dots, x_n]$ . Neste caso escrevemos  $I = \langle x^\alpha \mid \alpha \in A \rangle$ .*

Um exemplo de ideal monomial é  $\langle x^4y^2, x^3y^4, x^2y^5 \rangle \in F[x, y]$ . Um exemplo de um ideal não monomial sobre o mesmo anel polinomial é  $\langle x, y^2 - y \rangle$ .

**Lema 3.2.** *Seja  $I = \langle x^\alpha \mid \alpha \in A \rangle$  um ideal monomial e  $\beta \in \mathbb{N}^n$ . Então,*

$$x^\beta \in I \iff \exists \alpha \in A, x^\alpha \parallel x^\beta.$$

*Demonstração.*

⊆ Se  $x^\beta$  é um múltiplo de  $x^\alpha$  para algum  $\alpha \in A$ , então  $x^\beta \in I$  pela definição de ideal.

⊇ Se  $x^\beta \in I$ , então  $x^\beta = \sum_{i=1}^s h_i x^{\alpha_i}$ , onde  $h_i \in F[x_1, \dots, x_n]$  e  $\alpha_i \in A$ . Se expandirmos cada  $h_i$  como uma soma de termos, obtemos

$$x^\beta = \sum_{i=1}^s h_i x^{\alpha_i} = \sum_{i=1}^s \left( \sum_j c_{i,j} x^{\beta_{i,j}} \right) x^{\alpha_i} = \sum_{i,j} c_{i,j} x^{\beta_{i,j} + \alpha_i}.$$

Depois de juntar os termos de mesmo multigrado, todo termo no lado direito da equação é divisível por algum  $x^{\alpha_i}$  e logo o mesmo vale para  $x^\beta$ .

□

**Lema 3.3.** *Sejam  $I$  um ideal monomial e  $f \in F[x_1, \dots, x_n]$ . Então as seguintes proposições são equivalentes:*

(i)  $f \in I$ ;

(ii) *Todo termo de  $f$  pertence a  $I$ ;*

(iii)  *$f$  é uma combinação linear sobre  $F$  dos monômios de  $I$ ;*

*Demonstração.* As cadeias de implicação (iii)  $\Rightarrow$  (ii)  $\Rightarrow$  (i) e (ii)  $\Rightarrow$  (iii) são triviais. A prova de (i)  $\Rightarrow$  (ii) é similar ao que fizemos no lema 3.2.  $\square$

**Corolário 3.4.** *Dois ideais monomiais são o mesmo se, e só se, contém os mesmos monômios.*

**Teorema 3.5** (Lema de Dickson). *Seja  $I = \langle x^\alpha \mid \alpha \in A \rangle \subset F[x_1, \dots, x_n]$  um ideal monomial. Então existe  $B \subset A$  finito tal que  $I = \langle x^\beta \mid \beta \in B \rangle$ .*

*Demonstração.* A prova é por indução sobre o número de variáveis  $n$ . Se  $n = 1$ , então  $I$  é gerado pelo monômio  $x_1^\alpha$ , onde  $\alpha \in A \subset \mathbb{N}$ . Seja  $\beta$  o menor elemento de  $A \subset \mathbb{N}$ . Então  $\beta \leq \alpha$ , para todo  $\alpha \in A$ , de forma que  $x_1^\beta$  divide todos os outros geradores  $x_1^\alpha$  e logo  $I = \langle x_1^\beta \rangle$ . Agora suponha que  $n > 1$  e que o teorema é verdadeiro para  $n - 1$  variáveis. Escreveremos as variáveis como  $x_1, \dots, x_{n-1}, y$ , de forma que os monômios em  $F[x_1, \dots, x_{n-1}, y]$  possam ser escritos como  $x^\alpha y^m$ , onde  $\alpha = (\alpha_1, \dots, \alpha_{n-1}) \in \mathbb{N}^{n-1}$  e  $m \in \mathbb{N}$ .

Suponha que  $I \subset F[x_1, \dots, x_{n-1}, y]$  é um ideal monomial. Para encontrar geradores para  $I$ , seja  $J$  o ideal em  $F[x_1, \dots, x_{n-1}]$  gerado pelos monômios  $x^\alpha$  para o qual  $x^\alpha y^m \in I$  para algum  $m \geq 0$ . Visto que  $J$  é um ideal monomial em  $F[x_1, \dots, x_{n-1}]$ , nossa hipótese de indução implica que um número finito de monômios do tipo  $x^\alpha$  geram  $J$ , digamos  $J = \langle x^{\alpha_1}, \dots, x^{\alpha_s} \rangle$ . O ideal  $J$  pode ser visto como uma “projeção” de  $I$  em  $F[x_1, \dots, x_{n-1}]$ .

Para cada  $i$  entre 1 e  $s$ , a definição de  $J$  nos diz que  $x^{\alpha_i} y^{m_i} \in I$ , para alguma  $m_i \geq 0$ . Seja  $m$  o maior dos  $m_i$ . Então, para cada  $l$  entre 0 e  $m - 1$ , considere o ideal  $J_l \subset F[x_1, \dots, x_{n-1}]$  gerado pelos monômios  $x^\beta$  tais que  $x^\beta y^l \in I$ . Pode-se pensar em  $J_l$  como uma “fatia” de  $I$  gerada pelos monômios contendo exatamente  $y$  elevado à  $l$ -ésima potência. Usando nossa hipótese de indução novamente,  $J_l$  tem um número finito de monômios geradores, digamos

$J_l = \langle x^{\alpha_l(1)}, \dots, x^{\alpha_l(s_l)} \rangle$ . Afirmamos que  $I$  é gerado pelos monômios da seguinte lista:

$$\begin{aligned} J &: x^{\alpha(1)}y^m, \dots, x^{\alpha(s)}y^m, \\ J_0 &: x^{\alpha_0(1)}y^0, \dots, x^{\alpha_0(s_0)}y^0, \\ J_1 &: x^{\alpha_1(1)}y^1, \dots, x^{\alpha_1(s_1)}y^1, \\ &\vdots \\ J_{m-1} &: x^{\alpha_{m-1}(1)}y^{m-1}, \dots, x^{\alpha_{m-1}(s_{m-1})}y^{m-1} \end{aligned}$$

Primeiramente note que todo monômio em  $I$  é divisível por algum na lista. Para ver o porquê, seja  $x^\alpha y^p \in I$ . Se  $p \geq m$ , então  $x^\alpha y^p$  é divisível por algum  $x^{\alpha(i)}y^m$  pela construção de  $J$ . Por outro lado, se  $p \leq m-1$ , então  $x^\alpha y^p$  é divisível por algum  $x^{\alpha_p(j)}y^p$  pela construção de  $J_p$ . Segue do lema 3.2 que os monômios acima geram um ideal com os mesmos monômios que  $I$ . Pelo corolário 3.4, isso garante que os ideais sejam o mesmo, e então nossa afirmação fica demonstrada.

Para completar a prova, precisamos mostrar que um conjunto finito de geradores pode ser escolhido de um conjunto dado de geradores para o ideal. Se voltarmos a escrever as variáveis como  $x_1, \dots, x_n$ , então o nosso ideal monomial passa a ser  $I = \langle x^\alpha \mid \alpha \in A \rangle \subset F[x_1, \dots, x_n]$ . Precisamos mostrar que  $I$  é gerado por um subconjunto finito  $B \subset A$ . Sabemos que  $I = \langle x^{\beta_1}, \dots, x^{\beta_s} \rangle$  para monômios  $x^{\beta_i} \in I$ . Dado que  $x^{\beta_i} \in I = \langle x^\alpha \mid \alpha \in A \rangle$ , pelo lema 3.2, cada  $x^{\beta_i}$  é divisível por  $x^{\alpha_i}$ , para algum  $\alpha_i \in A$ . Não é difícil então mostrar que  $I = \langle x^{\alpha_1}, \dots, x^{\alpha_s} \rangle$ .  $\square$

**Corolário 3.6.** *Seja  $>$  uma relação em  $\mathbb{N}^n$  satisfazendo:*

(i)  $>$  é uma ordem total em  $\mathbb{N}^n$ ;

(ii) Se  $\alpha > \beta$  e  $\gamma \in \mathbb{N}^n$ , então  $\alpha + \gamma > \beta + \gamma$ ;

Então,  $>$  é uma boa ordem se, e só se,  $\alpha \geq 0$  para todo  $\alpha \in \mathbb{N}^n$ .

*Demonstração.*

$\Rightarrow$  Presumindo que  $>$  é uma boa ordem, seja  $\alpha_0$  o menor elemento de  $\mathbb{N}^n$ . Basta mostrar que  $\alpha_0 \geq 0$ . Isto é claramente verdade, pois se  $0 > \alpha_0$ , então pela hipótese (ii), somando  $\alpha_0$  a ambos aos lados da inequação obtemos  $\alpha_0 > 2\alpha_0$ , o que é impossível dado que  $\alpha_0$  foi tomado como o menor elemento de  $\mathbb{N}^n$ .

◁ Presumindo que  $\alpha \geq 0$  para todo  $\alpha \in \mathbb{N}^n$ , seja  $A \subset \mathbb{N}^n$  não vazio. Precisamos mostrar que  $A$  tem um menor elemento. Já que  $I = \langle x^\alpha \mid \alpha \in A \rangle$  é um ideal monomial, o teorema 3.5 nos dá que  $\alpha_1, \dots, \alpha_s \in A$ , de tal forma que  $I = \langle x^{\alpha_1}, \dots, x^{\alpha_s} \rangle$ . Podemos supor sem perda de generalidade que  $\alpha_1 < \alpha_2 < \dots < \alpha_s$ . Afirmamos que  $\alpha_1$  é o menor elemento de  $A$ . Com efeito, seja  $\alpha \in A$ . Então  $x^\alpha \in I = \langle x^{\alpha_1}, \dots, x^{\alpha_s} \rangle$ , e pelo lema 3.2,  $x^\alpha$  é divisível por algum  $x^{\alpha_i}$ . Isso significa que  $\alpha = \alpha_i + \gamma$ , para algum  $\gamma \in \mathbb{N}^n$ . Então  $\gamma \geq 0$  e a hipótese (ii) implica que  $\alpha = \alpha_i + \gamma \geq \alpha_i + 0 = \alpha_i \geq \alpha_1$ . Portanto,  $\alpha_1$  é o menor elemento de  $A$ . □

**Proposição 3.7** (Base minimal). *Um ideal monomial  $I \subset F[x_1, \dots, x_n]$  tem um conjunto gerador  $x^{\alpha_1}, \dots, x^{\alpha_s}$  com a propriedade de que  $x^{\alpha_i} \nmid x^{\alpha_j}$  para  $i \neq j$ . Além disso, este conjunto é único e é chamado de **base minimal** de  $I$ .*

*Demonstração.* Pelo teorema 3.5,  $I$  tem uma base finita consistindo apenas de monômios. Se um dos monômios nesta base divide outro, então podemos descartar este e ainda termos uma base. Iterando sobre este procedimento provamos a existência da base minimal  $x^{\alpha_1}, \dots, x^{\alpha_s}$ . Para demonstrar a unicidade, suponha que  $x^{\beta_1}, \dots, x^{\beta_t}$  é uma segunda base minimal de  $I$ . Então  $x^{\alpha_1} \in I$  e o lema 3.2 implica que  $x^{\beta_i} \parallel x^{\alpha_1}$  para algum  $i$ . Logo,  $x^{\alpha_j} \parallel x^{\alpha_1}$ , o que pela minimalidade implica  $j = 1$  e por conseguinte  $x^{\alpha_1} = x^{\beta_i}$ . Continuando dessa forma vemos que a primeira base está contida na segunda. Portanto, alternando as bases, a igualdade segue pelo mesmo raciocínio. □

**Notação.** *Seja  $I \subset F[x_1, \dots, x_n]$  um ideal não trivial com uma ordem monomial subentendida em  $F[x_1, \dots, x_n]$ . Então:*

(i) *Denota-se por  $\text{TL}(I)$  o conjunto de termos líderes de elementos não nulos de  $I$ , isto é,*

$$\text{TL}(I) = \{cx^\alpha \mid \text{TL}(f) = cx^\alpha, \exists f \in I \setminus \{0\}\}$$

(ii) *Denota-se por  $\langle \text{TL}(I) \rangle$  o ideal gerado pelos elementos de  $\text{TL}(I)$ .*

**Proposição 3.8.** *Seja  $I \subset F[x_1, \dots, x_n]$  um ideal não trivial. Então:*

(i)  *$\langle \text{TL}(I) \rangle$  é um ideal monomial;*

(ii) Existem  $g_1, \dots, g_t \in I$  tais que  $\langle \text{TL}(I) \rangle = \langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle$ .

*Demonstração.*

(i) Os monômios líderes  $\text{ML}(g)$  de elementos  $g \in I \setminus \{0\}$  geram o ideal monomial

$$\langle \text{ML}(g) \mid g \in I \setminus \{0\} \rangle.$$

Dado que  $\text{ML}(g)$  e  $\text{TL}(g)$  diferem por uma constante não nula, este ideal equivale a  $\langle \text{TL}(g) \mid g \in I \setminus \{0\} \rangle = \langle \text{TL}(I) \rangle$ . Portanto,  $\langle \text{TL}(I) \rangle$  é um ideal monomial.

(ii) Já que  $\langle \text{TL}(I) \rangle$  é gerado pelos monômios  $\text{ML}(g)$  para  $g \in I \setminus \{0\}$ , o teorema 3.5 garante que  $\langle \text{TL}(I) \rangle = \langle \text{ML}(g_1), \dots, \text{ML}(g_t) \rangle$  para uma quantidade finita de  $g_1, \dots, g_t \in I$ . Visto que  $\text{ML}(g_i)$  difere de  $\text{TL}(g_i)$  por uma constante não nula, segue que  $\langle \text{TL}(I) \rangle = \langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle$ .

□

**Teorema 3.9** (Teorema da Base de Hilbert). *Todo ideal  $I \subset F[x_1, \dots, x_n]$  tem um conjunto gerador finito. Em outras palavras,  $I = \langle g_1, \dots, g_t \rangle$  para alguns  $g_1, \dots, g_t \in I$ .*

*Demonstração.* Se  $I$  for um ideal trivial, tomamos nosso conjunto gerador como  $\{0\}$ , que é obviamente finito. Se  $I$  tem como elemento algum polinômio não nulo, então um conjunto gerador  $g_1, \dots, g_t$  para  $I$  pode ser construído da seguinte forma. Primeiramente selecionamos uma ordem monomial particular para usar com o algoritmo da divisão e na computação de termos líderes. Então  $I$  terá um ideal de termos líderes  $\langle \text{TL}(I) \rangle$ . Pela proposição 3.8, existem  $g_1, \dots, g_t \in I$  tais que  $\langle \text{TL}(I) \rangle = \langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle$ . Afirmamos que  $I = \langle g_1, \dots, g_t \rangle$ . Fica claro que  $\langle g_1, \dots, g_t \rangle \subset I$ , visto que cada  $g_i \in I$ . Reciprocamente, seja  $f \in I$  um polinômio qualquer. Se aplicarmos o algoritmo da divisão usado na prova do teorema 2.13 para dividir  $f$  por  $(g_1, \dots, g_t)$ , chegamos a uma expressão da forma  $f = q_1g_1 + \dots + q_tg_t + r$ , onde nenhum termo de  $r$  é divisível por nenhum dos  $\text{TL}(g_1), \dots, \text{TL}(g_t)$ . Afirmamos que  $r = 0$ . De fato, note que  $r = f - q_1g_1 - \dots - q_tg_t \in I$ . Se  $r \neq 0$ , então  $\text{TL}(r) \in \langle \text{TL}(I) \rangle = \langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle$ , e pelo lema 3.2 segue que  $\text{TL}(r)$  é divisível por algum  $\text{TL}(g_i)$ . Isso contradiz a definição de resto e logo só pode ser o caso que  $r$  é nulo. Portanto,  $f = q_1g_1 + \dots + q_tg_t + 0 \in \langle g_1, \dots, g_t \rangle$ , o que mostra que  $I \subset \langle g_1, \dots, g_t \rangle$ .

□

# Capítulo 4

## Bases de Gröbner

**Definição 4.1** (Base de Gröbner). *Fixe uma ordem monomial no anel polinomial  $F[x_1, \dots, x_n]$ . Um subconjunto finito  $G = \{g_1, \dots, g_t\}$  de um ideal não trivial  $I \subset F[x_1, \dots, x_n]$ , é uma **base de Gröbner** (ou **base padrão**) se  $\langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle = \langle \text{TL}(I) \rangle$ . Convencionando que  $\langle \emptyset \rangle = \{0\}$ , definimos o conjunto vazio como a base de Gröbner do ideal  $\{0\}$ .*

Por exemplo, uma base de Gröbner para o ideal  $\langle xz - y^2, x^3 - z^2 \rangle$  é  $\{y^6 - z^5, y^4x - z^4, y^2x^2 - z^3, zx - y^2, x^3 - z^2\}$ .

**Corolário 4.2.** *Fixada uma ordem monomial, todo ideal  $I \subset F[x_1, \dots, x_n]$  tem uma base de Gröbner. Ainda mais, qualquer base de Gröbner para um ideal  $I$  é uma base de  $I$ .*

*Demonstração.* Dado um ideal não trivial, o conjunto  $G = \{g_1, \dots, g_t\}$  construído na prova do teorema 3.9 é uma base de Gröbner por definição. Com relação à segunda afirmação, observe que se  $\langle \text{TL}(I) \rangle = \langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle$ , então o argumento dado para a prova do teorema 3.9 mostra que  $I = \langle g_1, \dots, g_t \rangle$ , e portanto  $G$  é uma base para  $I$ .  $\square$

**Definição 4.3** (Cadeia ascendente). *Uma **cadeia ascendente** de ideais é uma sequência aninhada crescente  $I_1 \subset I_2 \subset I_3 \subset \dots \subset I_n \subset \dots$ , onde  $I_i$  é um ideal para todo  $i \in \mathbb{N}$ .*

**Teorema 4.4** (Condição da cadeia ascendente). *Seja  $I_1 \subset I_2 \subset I_3 \subset \dots \subset I_n \subset \dots$  uma cadeia ascendente de ideais em  $F[x_1, \dots, x_n]$ . Então existe  $N \geq 1$  tal que  $I_N = I_{N+1} = I_{N+2} = \dots$ .*

*Demonstração.* Dada a cadeia ascendente  $I_1 \subset I_2 \subset I_3 \subset \dots \subset I_n \subset \dots$ , considere o conjunto  $I = \bigcup_{i=1}^{\infty} I_i$ . Começemos por mostrar que  $I$  é também um ideal em  $F[x_1, \dots, x_n]$ . Primeiramente,  $0 \in I$  dado que  $0 \in I_i$  para cada  $i$ . Agora, se  $f, g \in I$ , então por definição  $f \in I_i$  e  $g \in I_j$ ,



para algum  $i$  e  $j$  não necessariamente iguais. No entanto, como os ideais  $I_i$  formam uma cadeia ascendente, supondo sem perda de generalidade que  $i \leq j$ , temos que ambos  $f$  e  $g$  estão em  $I_j$ . Visto que  $I_j$  é um ideal, a soma  $f + g \in I_j$ , e logo  $f + g \in I$ . Similarmente, se  $f \in I$  e  $r \in F[x_1, \dots, x_n]$ , então  $f \in I_i$  para algum  $i$  e  $r \cdot f \in I_i \subset I$ . Portanto,  $I$  é um ideal. Pelo teorema 3.9,  $I$  possui um conjunto gerador finito:  $I = \langle f_1, \dots, f_s \rangle$ . Mas cada um dos geradores é um elemento em algum dos  $I_j$ , digamos  $f_i \in I_{j_i}$  para algum  $j_i$ ,  $1 \leq i \leq s$ . Definindo  $N$  como o máximo dos  $j_i$ , temos que, pela definição de cadeia ascendente,  $f_i \in I_N$  para todo  $i$ . Portanto,  $I = \langle f_1, \dots, f_s \rangle \subset I_N \subset I_{N+1} \subset \dots \subset I$ .  $\square$

**Proposição 4.5.** *Seja  $I \subset F[x_1, \dots, x_n]$  um ideal e seja  $G = \{g_1, \dots, g_t\}$  uma base de Gröbner para  $I$ . Então dado  $f \in F[x_1, \dots, x_n]$ , existe um único  $r \in F[x_1, \dots, x_n]$  com as seguintes propriedades:*

(i) *Nenhum termo de  $r$  é divisível por nenhum dos  $\text{TL}(g_1), \dots, \text{TL}(g_t)$ ;*

(ii) *Existe  $g \in I$  tal que  $f = g + r$ ;*

*Em particular,  $r$  é o resto da divisão de  $f$  por  $G$  independentemente de como os elementos de  $G$  são listados quando se usa o algoritmo da divisão.*

*Demonstração.* O algoritmo da divisão nos dá  $f = q_1g_1 + \dots + q_tg_t + r$ , onde  $r$  satisfaz (i). Pode-se também satisfazer a (ii) fazendo  $g = q_1g_1 + \dots + q_tg_t \in I$ . Isso prova a existência de  $r$ . Para provar sua unicidade, suponha que  $f = g + r = g' + r'$  satisfazendo (i) e (ii). Então  $r - r' = g' - g \in I$ , de forma que se  $r \neq r'$ , então  $\text{TL}(r - r') \in \langle \text{TL}(I) \rangle = \langle \text{TL}(g_1), \dots, \text{TL}(g_t) \rangle$ . Pelo lema 3.2, segue que  $\text{TL}(r - r')$  é divisível por algum  $\text{TL}(g_i)$ . Isso é impossível, dado que nenhum termo de  $r$  e  $r'$  é divisível por um dos  $\text{TL}(g_1), \dots, \text{TL}(g_t)$ , e logo a diferença  $r - r'$  é nula. A parte final da proposição segue diretamente da unicidade de  $r$ .  $\square$

**Corolário 4.6.** *Seja  $G = \{g_1, \dots, g_t\}$  uma base de Gröbner para um ideal  $I \subset F[x_1, \dots, x_n]$  e seja  $f \in F[x_1, \dots, x_n]$ . Então  $f \in I$  se, e só se, o resto da divisão de  $f$  por  $G$  é zero.*

*Demonstração.* Se o resto é zero, então já sabemos que  $f \in I$ . Reciprocamente, dado  $f \in I$ , então  $f = f + 0$  satisfaz as duas condições da proposição 4.5. Logo tem-se que 0 é o resto da divisão de  $f$  por  $G$ .  $\square$

**Notação.** *Denota-se o resto da divisão de  $f$  pela  $s$ -upla ordenada  $D = (f_1, \dots, f_s)$  como  $\bar{f}^D$ . Neste caso se  $D$  é uma base de Gröbner para  $\langle f_1, \dots, f_s \rangle$ , então pela proposição 4.5 podemos considerar  $D$  como um conjunto (sem nenhuma ordem particular).*

**Definição 4.7.** *Sejam  $f, g \in F[x_1, \dots, x_n]$  polinômios não nulos.*

(i) *Se  $\text{multigrau}(f) = \alpha$  e  $\text{multigrau}(g) = \beta$ , então seja  $\gamma = (\gamma_1, \dots, \gamma_n)$ , onde  $\gamma_i = \max\{\alpha_i, \beta_i\}$  para cada  $i$ . Defina-se o **mínimo múltiplo comum** de  $\text{ML}(f)$  e  $\text{ML}(g)$  como*

$$\text{mmc}\{\text{ML}(f), \text{ML}(g)\} := x^\gamma$$

(ii) *Defina-se o **S-polinômio** de  $f$  e  $g$  como*

$$S(f, g) = \frac{x^\gamma}{\text{TL}(f)} \cdot f - \frac{x^\gamma}{\text{TL}(g)} \cdot g$$

**Lema 4.8.** *Suponha que para a soma  $\sum_{i=1}^s p_i$ , vale que  $\text{multigrau}(p_i) = \delta \in \mathbb{N}^n$  para todo  $i$ . Se  $\text{multigrau}(\sum_{i=1}^s p_i) < \delta$ , então  $\sum_{i=1}^s p_i$  é uma combinação linear com coeficientes em  $F$ , de polinômios-S  $S(p_j, p_l)$  para  $1 \leq j, l \leq s$ . Além disso, cada  $S(p_j, p_l)$  tem multigrau menor que  $\delta$ .*

*Demonstração.* Seja  $d_i = \text{CL}(p_i)$ , de forma que  $d_i x^\delta$  é o termo líder de  $p_i$ . Já que a soma  $\sum_{i=1}^s p_i$  tem multigrau estritamente menor, segue diretamente que  $\sum_{i=1}^s d_i = 0$ . Agora, observe que como  $p_i$  e  $p_j$  tem o mesmo monômio líder, o S-polinômio de ambos se reduz a

$$S(p_i, p_j) = \frac{1}{d_i} p_i - \frac{1}{d_j} p_j. \quad (4.1)$$

Portanto,

$$\begin{aligned} \sum_{i=1}^{s-1} d_i S(p_i, p_s) &= d_1 \left( \frac{1}{d_1} p_1 - \frac{1}{d_s} p_s \right) + d_2 \left( \frac{1}{d_2} p_2 - \frac{1}{d_s} p_s \right) + \dots \\ &= p_1 + p_2 + \dots + p_{s-1} - \frac{1}{d_s} (d_1 + \dots + d_{s-1}) p_s. \end{aligned} \quad (4.2)$$

No entanto, a igualdade  $\sum_{i=1}^s d_i = 0$  implica que  $d_1 + \dots + d_{s-1} = -d_s$ , e logo a equação (4.2) se reduz a

$$\sum_{i=1}^{s-1} d_i S(p_i, p_s) = p_1 + \dots + p_{s-1} + p_s.$$

Portanto,  $\sum_{i=1}^s p_i$  é uma soma de polinômios-S da forma desejada e a equação (4.1) mostra facilmente que  $S(p_i, p_j)$  tem multigrau menor que  $\delta$ .  $\square$

**Teorema 4.9** (Critério de Buchberger). *Seja  $I$  um ideal polinomial. Então uma base  $G = \{g_1, \dots, g_t\}$  de  $I$  é uma base de Gröbner de  $I$  se, e só se, para todos os pares  $i \neq j$ , o resto da divisão de  $S(g_i, g_j)$  por  $G$  (listado em alguma ordem) é zero.*

*Demonstração.*

$\Rightarrow$  Se  $G$  é uma base de Gröbner, então pelo corolário 4.6, dado  $S(g_i, g_j) \in I$ , o resto da divisão por  $G$  é zero.

$\Leftarrow$  Seja  $f \in I$  não nulo. Provaremos que  $TL(f) \in \langle TL(g_1), \dots, TL(g_t) \rangle$ . Fazendo

$$f = \sum_{i=1}^t h_i g_i, \quad h_i \in F[x_1, \dots, x_n],$$

do lema 2.12, tem-se

$$\text{multigrau}(f) \leq \max \{ \text{multigrau}(h_i g_i) \mid h_i g_i \neq 0 \} \quad (4.3)$$

A estratégia da prova é selecionar a representação mais eficiente de  $f$ , isto é, dentre todas as expressões  $f = \sum_{i=1}^t h_i g_i$ , selecionamos uma para a qual

$$\delta = \max \{ \text{multigrau}(h_i g_i) \mid h_i g_i \neq 0 \}$$

é mínimo. O  $\delta$  mínimo existe pela propriedade de boa ordem da ordem monomial estabelecida. Pela equação (4.3), segue que  $\text{multigrau}(f) \leq \delta$ . Se a igualdade ocorre, então  $\text{multigrau}(f) = \text{multigrau}(h_i g_i)$  para algum  $i$ . É fácil ver que isso implica que  $TL(g_i) \parallel TL(f)$ . Portanto,  $TL(f) \in \langle TL(g_1), \dots, TL(g_t) \rangle$ . Consideremos agora o caso em que o  $\delta$  mínimo satisfaz  $\text{multigrau}(f) < \delta$ . Usaremos  $\overline{S(g_i, g_j)}^G = 0$ , para  $i \neq j$ , de forma a encontrar uma nova expressão para  $f$  que diminui  $\delta$ , gerando uma contradição com sua minimalidade e completando a prova. Dada uma expressão  $f = \sum_{i=1}^t h_i g_i$  com  $\delta$  mínimo, comecemos por isolar a parte da soma onde o multigrau  $\delta$  ocorre:

$$\begin{aligned} f &= \sum_{\text{multigrau}(h_i g_i) = \delta} h_i g_i + \sum_{\text{multigrau}(h_i g_i) < \delta} h_i g_i \\ &= \sum_{\text{multigrau}(h_i g_i) = \delta} TL(h_i) g_i + \sum_{\text{multigrau}(h_i g_i) = \delta} (h_i - TL(h_i)) g_i + \sum_{\text{multigrau}(h_i g_i) < \delta} h_i g_i. \end{aligned} \quad (4.4)$$

Os monômios que aparecem na segunda e terceira soma da segunda linha têm multigrau menor que  $\delta$ . Portanto,  $\text{multigrau}(f) < \delta$  significa que a primeira soma da segunda linha também tem multigrau menor que  $\delta$ . O segredo para reduzir  $\delta$  é reescrever a primeira soma em dois estágios: use o lema 4.8 para reescrever a primeira soma em termos do S-polinômio, e então use  $\overline{S(g_i, g_j)}^G = 0$  para reescrever o S-polinômio sem cancelamentos. Para expressar a primeira soma na segunda linha de (4.4) usando polinômios-S note que

$$\sum_{\text{multigrau}(h_i g_i) = \delta} TL(h_i) g_i \quad (4.5)$$

satisfaz a hipótese do lema 4.8 já que cada  $p_i = \text{TL}(h_i)g_i$  tem multigrav  $\delta$  e a soma tem multigrav menor que  $\delta$ . Portanto, a primeira soma é uma combinação linear com coeficientes em  $F$  do S-polinômio  $S(p_i, p_j)$ . Visto que  $S(p_i, p_j) = x^{\delta-\gamma_{ij}} S(g_i, g_j)$ , onde  $x^{\gamma_{ij}} = \text{mmc}\{\text{ML}(g_i), \text{ML}(g_j)\}$ . Segue-se que a primeira soma (4.5) é uma combinação linear de  $x^{\delta-\gamma_{ij}} S(g_i, g_j)$  para certos pares  $(i, j)$ . Considere um destes polinômios-S  $S(g_i, g_j)$ . Visto que  $\overline{S(g_i, g_j)}^G = 0$ , o algoritmo da divisão (dado na prova do teorema 2.13) nos dá a expressão

$$S(g_i, g_j) = \sum_{l=1}^t A_l g_l, \quad (4.6)$$

onde  $A_l \in F[x_1, \dots, x_n]$  e

$$\text{multigrav}(A_l g_l) \leq \text{multigrav}(S(g_i, g_j)), \quad (4.7)$$

quando  $A_l g_l \neq 0$ . Agora, multiplicando cada membro da equação (4.6) por  $x^{\delta-\gamma_{ij}}$  para obter

$$x^{\delta-\gamma_{ij}} S(g_i, g_j) = \sum_{l=1}^t B_l g_l, \quad (4.8)$$

onde  $B_l = x^{\delta-\gamma_{ij}} A_l$ . Logo, a equação (4.7) implica que quando  $B_l g_l \neq 0$ , temos

$$\text{multigrav}(B_l g_l) \leq \text{multigrav}(x^{\delta-\gamma_{ij}} S(g_i, g_j)) < \delta \quad (4.9)$$

visto que  $\text{TL}(S(g_i, g_j)) < \text{mmc}\{\text{ML}(g_i), \text{ML}(g_j)\} = x^{\gamma_{ij}}$ . Segue-se que a primeira soma (4.5) é uma combinação linear de certos  $x^{\delta-\gamma_{ij}} S(g_i, g_j)$ , cada um satisfazendo (4.8) e (4.9). Logo, podemos escrever a primeira soma como

$$\sum_{\text{multigrav}(h_i g_i) = \delta} \text{TL}(h_i) g_i = \sum_{l=1}^t \tilde{B}_l g_l, \quad (4.10)$$

com a propriedade de que quando  $\tilde{B}_l g_l \neq 0$ , temos  $\text{multigrav}(\tilde{B}_l g_l) < \delta$ . Substituindo (4.10) na segunda linha da equação (4.4), chegamos a uma expressão para  $f$  como uma combinação polinomial dos  $g_i$  onde todos os termos tem multigrav menor que  $\delta$ , o que é uma contradição com a minimalidade de  $\delta$ .  $\square$

Para um exemplo de como usar o critério de Buchberger, considere o ideal  $I = \langle y - x^3, z - x^3 \rangle$ . Afirmamos que  $G = \{y - x^2, z - x^3\}$  é uma base de Gröbner segundo a ordem lexicográfica com  $y > z > x$ . Para provar, considere o S-polinômio

$$S(y - x^2, z - x^3) = \frac{yz}{y}(y - x^2) - \frac{yz}{z}(z - x^3) = -zx^2 + yx^3.$$

Usando o algoritmo da divisão, encontramos

$$-zx^2 + yx^3 = x^3(y - x^2) + (-x^2)(z - x^3) + 0$$

de forma que  $\overline{S(y - x^2, z - x^3)}^G = 0$ . Portanto, pelo critério de Buchberger,  $G$  é uma base de Gröbner para  $I$ .

# Capítulo 5

## Algoritmo de Buchberger

Dado um ideal polinomial  $I = \langle f_1, \dots, f_s \rangle \neq \{0\}$ , o algoritmo de Buchberger constrói uma base de Gröbner para  $I$  em um número finito de passos. Abaixo listamos sua versão otimizada conforme dada em [2, p. 196, 197]. A implementação do mesmo em UserRPL é dada na seção 7.

```
1  ALGORÍTMO Buchberger( $R$ )
2   $P := \emptyset$ ;  $G := \emptyset$ ;  $B := \emptyset$ ;
3  Reduza( $R, P, G, B$ ); NovaBase( $P, G, B$ );
4  ENQUANTO  $B \neq \emptyset$ 
5  |    $\{f_1, f_2\} :=$  MenorMMC( $B$ )
6  |    $B := B - \{\{f_1, f_2\}\}$ 
7  |   SE NÃO Critério1( $f_1, f_2, G, B$ ) E NÃO Critério2( $f_1, f_2$ )
8  |   |    $h :=$  FormaNormal( $G, S(f_1, f_2)$ )
9  |   |   SE  $h \neq 0$ 
10 |   |   |    $G_0 := \{g \in G \mid \text{ML}(h) \parallel \text{ML}(g)\}$ 
11 |   |   |    $R := G_0$ ;  $P := \{h\}$ ;  $G := G - G_0$ ;
12 |   |   |    $B := B - \{\{f_1, f_2\} \mid f_1 \in G_0 \vee f_2 \in G_0\}$ 
13 |   |   |   Reduza( $R, P, G, B$ ); NovaBase( $P, G, B$ );
14 |   RETORNE  $G$ 
```

```
1  ALGORÍTMO Reduza(referência :  $R, P, G, B$ )
2  ENQUANTO  $R \neq 0$ 
3  |    $h :=$  Escolha( $R$ );  $R := R - \{h\}$ ;
4  |    $h :=$  FormaNormal( $G \cup P, h$ )
5  |   SE  $h \neq 0$ 
6  |   |    $G_0 := \{g \in G \mid \text{ML}(h) \parallel \text{ML}(g)\}$ 
7  |   |    $P_0 := \{p \in P \mid \text{ML}(h) \parallel \text{ML}(p)\}$ 
8  |   |    $G := G - G_0$ 
9  |   |    $P := P - P_0$ 
10 |   |    $R := R \cup G_0 \cup P_0$ 
11 |   |    $B := B - \{\{f_1, f_2\} \in B \mid f_1 \in G_0 \vee f_2 \in G_0\}$ 
12 |   |    $P := P \cup \{h\}$ 
```

```

1  ALGORÍTMO NovaBase(referência : P, G, B)
2  | G := G ∪ P
3  | B := B ∪ {{g, p} | g ∈ G ∧ p ∈ P ∧ g ≠ p}
4  | H := G; K := ∅;
5  | ENQUANTO H ≠ ∅
6  | | h := Escolha(H); H := H - {h}
7  | | k := FormaNormal(G - {h}, h); K := K ∪ {k}
8  | G := K

```

```

1  ALGORÍTMO FormaNormal(G, f)
2  | Q, r := Divisão(f, G)
3  | RETORNE r

```

```

1  PREDICADO Critério1(f1, f2, G, B) ⇔ ∃p ∈ G, f1 ≠ p ∧ p ≠ f2
2  | ∧ ML(p) || mmc{ML(f1), ML(f2)}
3  | ∧ {f1, p} ∉ B ∧ {p, f2} ∉ B

```

```

1  PREDICADO Critério2(f1, f2) ⇔ mmc{ML(f1), ML(f2)} = ML(f1) · ML(f2)

```

```

1  FUNÇÃO MenorMMC(B) ↦ {f1, f2} ⇔
2  | mmc{ML(f1), ML(f2)} = min≥ {mmc{ML(g1), ML(g2)} | {g1, g2} ∈ B}

```

```

1  FUNÇÃO Escolha(X) ↦ x ∈ X

```

# Capítulo 6

## UserRPL

A calculadora gráfica HP 50g possui um poderoso sistema de álgebra computacional (CAS) programável em UserRPL (User's Reverse Polish Lisp), uma linguagem de programação de alto nível desenvolvida pela Hewlett Packard para suas calculadoras científicas. UserRPL é uma linguagem de programação estruturada baseada em RPN (Reverse Polish Notation) que também é capaz de processar expressões algébricas e fórmulas em notação infixa usando um interpretador intermediário. UserRPL tem atributos comuns à linguagem Forth (orientada à pilha) e à linguagem LISP (orientada à lista). Como tal, a programação em UserRPL tem como princípio a existência de uma pilha arbitrariamente grande (limitada apenas pela memória disponível na calculadora), onde objetos são empilhados, operados e avaliados.

Aqui apresentamos a definição de um pequeno subconjunto estrito da linguagem sob uma formulação simplificada satisfatória ao propósito didático de exposição e compreensão da implementação dos algoritmos propostos. Para uma descrição compreensiva a respeito dos recursos de programação consulte [5]. Para uma descrição abrangente dos recursos de utilização veja [4]. Para uma introdução ao uso do calculadora leia [3]. Caso não possua a calculadora gráfica HP 50g, um emulador completo e software livre encontra-se publicamente disponível:

**(Emu48)** <https://www.hpcalc.org/details/3644>

**(Emu48+ Service Pack)** <https://www.hpcalc.org/details/6523>

Todas as alusões ao desempilhamento de objetos e possível subsequente nomeação metalinguística devem ser consideradas em ordem decrescente de profundidade e tendo como limite o topo da pilha, a menos que feita ressalva explícita do contrário.



## 6.1 Objetos

Em UserRPL os objetos são as entidades lógicas que descrevem dados e procedimentos. Dois tipos de ações são definidas para todo objeto: execução e avaliação. A primeira diz respeito ao comportamento deste dentro de um programa quando executado, enquanto a segunda corresponde ao comportamento do mesmo no topo da pilha sob execução do comando **EVAL**.

**(Número)** Número racional. A execução o empilha. A avaliação não faz nada.

**(Cadeia de caracteres)** Sequência de caracteres. Denotada literalmente entre `""`. A execução a empilha. A avaliação não faz nada.

**(Nome)** Referência a um objeto. Denotada literalmente entre `''`. Pode ser global, local ou compilada. A referência global é visível no escopo de qualquer programa, a menos que ofuscada por uma referência local do programa corrente ou referência compilada do programa corrente ou de algum chamador dele. A referência local tem escopo léxico, isto é, sua visibilidade é restrita sintaticamente ao programa relativo à sua estrutura de declaração. A referência compilada tem escopo dinâmico, isto é, é visível enquanto durar a execução do programa relativo à sua estrutura de declaração. A execução empilha o nome quando delimitado entre `''`, senão (sendo o nome global) o objeto referenciado é avaliado caso seja um programa ou feito corrente caso seja um diretório, do contrário é executado. A avaliação é a execução sem delimitação entre `''`.

**(Algébrico)** Expressão algébrica simbólica infixa entre `''`. A execução o empilha. Se possível, a avaliação o desempilha e então empilha um objeto de mesmo valor, mas simplificado, por exemplo um número se o valor for determinado.

**(Lista)** Lista de objetos. Denotada literalmente entre `{}`. A execução a empilha. A avaliação a desempilha e empilha seus objetos componentes, em ordem posicional crescente.

**(Vetor)** Vetor de nomes, números ou algébricos. Denotada literalmente entre `[]`. A execução o empilha. A avaliação não faz nada.

**(Diretório)** Coleção de referências globais a objetos, inclusive outros diretórios, possivelmente caracterizando uma estrutura de árvore. Em qualquer momento existe um e apenas um diretório de trabalho atual, onde todas (e apenas) as referências globais dele e de qualquer

um de seus diretórios superiores (recursivamente), são visíveis. O diretório de mais alto nível é **HOME**. A execução o empilha. A avaliação não faz nada.

**(Comando)** Rotina primitiva da calculadora. A execução pode consumir zero ou mais argumentos da pilha, devolver zero ou mais valores de retorno para a pilha, manipular variáveis locais, globais e compiladas, dependendo da definição da rotina em questão. A avaliação não faz nada. Veja a seção 6.3 para uma lista e definição dos comandos utilizados na implementação dos algoritmos deste trabalho.

**(Programa)** Um programa é uma sequência de objetos e estruturas. Denotado literalmente por  $\ll \gg$ . A execução o empilha. A avaliação é a execução sequencial de seus objetos constituintes.

Qualquer objeto pode ser rotulado, (i.e., ter um objeto de metadados associado) por uma cadeia de caracteres, nome ou número. Denota-se literalmente o rótulo entre  $::$ . A execução de um objeto rotulado o empilha. A avaliação de um objeto rotulado o desempilha e empilha a avaliação do objeto associado (consequentemente descartando o rótulo).

## 6.2 Estruturas

Além de objetos programas podem também conter estruturas. Estruturas podem utilizar e mudar o comportamento padrão de execução e avaliação dos objetos que as compõem e alterar o fluxo sequencial de execução do programa.

**(Estrutura algébrica de variável local)** Desempilha  $n$  objetos e os fazem referenciáveis pelos nomes locais  $N_1, \dots, N_n$  resp., e o objeto 'a' é avaliado. Programas compostos por essa estrutura em seu nível mais externo são chamados de *funções do usuário* e podem ser diretamente usadas pelo CAS para efeito de manipulação algébrica, e.g., derivação e integração.

$$\begin{array}{l} \ll \\ \left| \begin{array}{l} \dots \\ \rightarrow N_1 \dots N_n \text{ 'a'} \\ \dots \end{array} \right. \\ \gg \end{array}$$

**(Estrutura procedural de variável local e compilada)** Desempilha  $n$  objetos e os fazem referenciáveis pelos nomes locais ou compilados (caso prefixados por  $\leftarrow$ )  $[\leftarrow]N_1 \cdots [\leftarrow]N_n$  resp., e os objetos  $\ll o_1 \cdots o_i \gg$  são avaliados.

```

<<
  ...
  → [←]N1 ⋯ [←]Nn
  <<
    | o1 ⋯ oi
  >>
  ...
>>

```

**(Estrutura condicional composta)** Executa objetos  $c_{1,1} \cdots c_{1,i}$ , depois disso desempilha um objeto. Se este for diferente de  $\emptyset.\emptyset$ , executa  $v_{1,1} \cdots v_{1,j}$  e depois continua a execução após o fim da estrutura, do contrário — caso existam mais cláusulas **THEN** — repete o mesmo processo sobre seus objetos de condição e consequência. Caso nenhuma condição avalie para um número diferente de  $\emptyset.\emptyset$ , executa  $d_1 \cdots d_n$  (caso existam) e depois continua a execução após o fim da estrutura.

```

<<
  ...
  CASE
    | c1,1 ⋯ c1,i THEN v1,1 ⋯ v1,j END
    | [c2,1 ⋯ c2,i THEN v2,1 ⋯ v2,k END] ⋯
    | [d1 ⋯ dn]
  END
  ...
>>

```

**(Estrutura condicional simples)** Caso particular da estrutura condicional composta, com apenas uma condição, uma consequência e possivelmente uma alternativa. O código à esquerda é equivalente ao código à direita.

```

<<
  ...
  IF c1 ⋯ ci THEN v1 ⋯ vj
  [ELSE d1 ⋯ dn]
  END
  ...
>>
  ⇒
  <<
  ...
  CASE
    | c1 ⋯ ci THEN v1 ⋯ vj END
    | [d1 ⋯ dn]
  END
  ...
>>

```

**(Estrutura de captura de erro)** Tenta executar objetos  $c_1 \cdots c_i$ . Caso haja um erro executa

$v_1 \cdots v_j$ , do contrário — caso exista cláusula **ELSE** — executa  $d_1 \cdots d_n$ . Qualquer que seja o caso, continua a execução após o fim da estrutura.

```

<<
| ...
| IFERR  $c_1 \cdots c_i$  THEN  $v_1 \cdots v_j$ 
| [ELSE  $d_1 \cdots d_n$ ]
| END
| ...
>>

```

**(Estrutura de laço determinado com contador)** Desempilha  $c$  e  $1$ , faz do nome  $n$  uma referência local ao  $c$  (com escopo limitado à estrutura) e depois executa os objetos  $o_1 \cdots o_i$ . Caso a cláusula final seja **STEP** desempilha objeto  $p$ , então o avalia (caso seja um nome ou um algébrico), e o adiciona ao  $c$ . Caso  $p \neq c$   $1 \leq$  **AND**  $p \neq c$   $1 \geq$  **AND OR** seja diferente de  $\emptyset$ , repete o processo a partir da execução de  $o_1 \cdots o_i$ , do contrário continua a execução após o fim da estrutura. A cláusula **NEXT** é equivalente a  $1$  **STEP**.

```

<<
| ...
| FOR  $n$ 
| |  $o_1 \cdots o_i$ 
| [STEP | NEXT]
| ...
>>

```

**(Estrutura de laço determinado sem contador)** Funciona exatamente como a estrutura de laço determinado com contador, exceto que não cria a referência  $n$  para  $c$ .

```

<<
| ...
| START
| |  $o_1 \cdots o_i$ 
| [STEP | NEXT]
| ...
>>

```

**(Estrutura de laço indeterminado de cauda)** Executa objetos  $o_1 \cdots o_i$  e  $c_1 \cdots c_n$ . Depois, desempilha um objeto e caso este seja  $\emptyset$  repete o processo, senão continua a execução após o fim da estrutura.

```

<<
| ...

```

```

DO
|  o1 ··· oi
UNTIL c1 ··· cn END
...

```

**(Estrutura de laço indeterminado de cabeça)** Executa objetos  $c_1 \cdots c_n$ . Depois, desempilha um objeto e caso este seja diferente de  $\emptyset$  executa os objetos  $o_1 \cdots o_i$  e repete o processo, senão continua a execução após o fim da estrutura.

```

<<
...
WHILE c1 ··· cn
REPEAT o1 ··· oi END
...
>>

```

## 6.3 Comandos

Nesta seção apresentamos uma lista compreensiva de todos os comandos usados para a implementação dos algoritmos propostos por este trabalho. Os comandos encontram-se separados pela categoria que definem o principal significado de seu comportamento.

### 6.3.1 Pilha

Como já elucidado, todo o processamento de dados na linguagem UserRPL acontece sobre uma pilha de tamanho virtualmente ilimitado. Os comandos de manipulação de objetos sobre a pilha são, portanto, de fundamental importância para a construção de processos computacionais mais complexos.

**DROPN** Desempilha número  $n$ , então desempilha  $n$  objetos.

**DROP** Equivalente a 1. **DROPN**.

**DROP2** Equivalente a 2. **DROPN**.

**PICK** Desempilha número  $n$ , então empilha cópia do objeto que está no nível de profundidade  $n$ .

**DUP** Equivalente a 1. **PICK**.

**OVER** Equivalente a 2. **PICK**.

**PICK3** Equivalente a 3. **PICK**.

**DUPDUP** Equivalente a **DUP DUP**.

**DUPN** Desempilha número  $n$ , então empilha uma cópia de cada um dos  $n$  objetos do topo da pilha, do mais profundo ao mais raso.

**DUP2** Equivalente a 2. **DUPN** ou **OVER OVER**.

**UNPICK** Desempilha o objeto  $o$  e o número  $n$ , então substitui objeto que está no nível de profundidade  $n$  por  $o$ .

**ROLL** Desempilha número  $n$ , então rotaciona os  $n$  objetos restantes do topo da pilha, de forma que o mais profundo se torna o mais raso.

**ROT** Equivalente a 3. **ROLL**.

**ROLLD** Desempilha número  $n$ , então rotaciona os  $n$  objetos restantes do topo da pilha, de forma que o mais raso se torna o mais profundo.

**UNROT** Equivalente a 3. **ROLLD**.

**SWAP** Equivalente a 2. **ROLL** ou 2 **ROLLD**.

**NIP** Equivalente a **SWAP DROP**.

**DEPTH** Empilha o número de objetos existentes na pilha.

**NDUPN** Desempilha número  $n$ , então empilha  $n$  cópias do objeto do topo da pilha, e por fim empilha  $n$ .

### 6.3.2 Lógico

Em UserRPL um valor lógico, também conhecido como *booleano*, é um número real de ponto flutuante, interpretado como falso caso seja idêntico a  $0.0$ , e verdadeiro caso contrário. A menos que feita ressalva particular contrária, qualquer comando empilha  $1.0$  para representar

o valor verdadeiro, mesmo que desempilhando qualquer valor diferente de  $0.0$  para significar o mesmo.

**NOT** Desempilha objeto  $e$ , se não é um nome ou um algébrico, empilha  $1.0$  caso este seja  $0.0$ , senão empilha  $0.0$ . Se é um nome ou algébrico, empilha uma expressão algébrica cuja avaliação resolve para o resultado da negação.

**AND** Desempilha dois objetos  $e$ , se nenhum deles é um nome ou um algébrico, empilha  $1.0$  caso ambos sejam diferentes de  $0.0$ , senão empilha  $0.0$ . Se algum dos objetos é um nome ou algébrico, empilha uma expressão algébrica cuja avaliação resolve para o resultado da conjunção.

**OR** Equivalente a **NOT SWAP NOT AND NOT**.

**XOR** Equivalente a **DUP2 NOT AND UNROT SWAP NOT AND OR**.

### 6.3.3 Comparação

O significado de comparação, seja com a finalidade de identificar ou ordenar é dependente do tipo dos objetos em questão. Objetos de tipos diferentes são normalmente tidos como diferentes, exceto em condições específicas envolvendo nomes ou algébricos. Sequências (veja seção 6.3.6) são tidas como iguais se forem do mesmo tipo, tiverem o mesmo número de objetos componentes, e seus respectivos objetos componentes forem iguais. A ordenação de números procede segundo a ordem usual dos números reais, já a ordenação de algébricos só tem valor lógico definido se os algébricos comparados avaliarem para números, enquanto que a ordenação de cadeias de caracteres segue a ordem lexicográfica convencional. Por outro lado, programas não podem ser ordenados e o resultado da ordenação de listas são listas em vez de valores lógicos.

**==** Desempilha dois objetos  $e$ , se nenhum deles é um nome ou um algébrico, empilha  $1.0$  caso sejam iguais, senão empilha  $0.0$ . Se algum dos objetos é um nome ou algébrico, empilha uma expressão algébrica cuja avaliação resolve para o resultado da igualdade.

**SAME** Desempilha dois objetos  $e$ , se nenhum deles é um nome ou um algébrico, seu comportamento é equivalente a **==**. Se algum dos objetos é um nome ou algébrico, empilha  $1.0$  caso sejam iguais, senão empilha  $0.0$ .

$\neq$  Equivalente a `== NOT`.

`<` Desempilha dois objetos e, se nenhum deles é um nome ou um algébrico, empilha `1.0` caso o mais profundo seja menor que o mais raso, senão empilha `0.0`. Se algum dos objetos é um nome ou algébrico, empilha uma expressão algébrica cuja avaliação resolve para o resultado da comparação.

`≤` Equivalente a `DUP2 < UNROT == OR`.

`>` Equivalente a `DUP2 < NOT UNROT ≠ AND`.

`≥` Equivalente a `DUP2 > UNROT == OR`.

### 6.3.4 Aritmética

Os comandos aritméticos se aplicam tanto a números quanto a nomes e algébricos, produzindo como resultado também um número, nome ou um algébrico, conforme apropriado.

**NEG** Desempilha objeto e empilha objeto de mesmo valor e sinal oposto.

**+** Desempilha dois objetos e empilha sua soma.

**-** Desempilha dois objetos e empilha a diferença entre o mais profundo e o mais raso.

**\*** Desempilha dois objetos e empilha seu produto.

**/** Desempilha dois objetos e empilha a razão entre o mais profundo e o mais raso.

**^** Desempilha dois objetos e empilha o mais profundo elevado à potência do mais raso.

**RAND** Empilha um número racional pseudo-aleatório tal que `DUP 0 ≥ SWAP 1 < AND` é `1.0`.

**RND** Desempilha números `z` e `n`, e empilha o arredondamento de `z` limitado a `n` casa decimais.

**MAX** Desempilha dois objetos e empilha o maior deles.

**LCM** Desempilha dois objetos e empilha seu menor múltiplo comum.

**GCD** Desempilha dois objetos e empilha seu maior divisor comum.

**ABS** Desempilha objeto e empilha seu valor absoluto.

**SIGN** Desempilha número e empilha `1.0` se este for positivo ou `-1.0` se este for negativo.



### 6.3.5 Sinalizadores

Um sinalizador é um bit de informação, que em qualquer dado momento pode-se encontrar exclusivamente em um de dois estados distintos: ligado ou desligado. Na calculadora, existem 256 sinalizadores que são identificados pelos números inteiros do conjunto  $[-128, -1] \cup [1, 128]$ . Os sinalizadores negativos são reservados para uso pelo sistema e tem em sua maior parte significados pré-definidos. Eles são usados por programas de usuário para verificar condições e configurar o processamento de dados feito pelas funções primitivas da calculadora. Os sinalizadores positivos são reservados aos programas de usuário e em sua maior parte e podem ser usados livremente pelos mesmos, tendo seu significado definido pelo programa em execução.

**SF** Desempilha número, e liga sinalizador de mesmo número.

**CF** Desempilha número, e desliga sinalizador de mesmo número.

**FS?** Desempilha número, e empilha 1.0 caso o sinalizador de mesmo número esteja ligado, senão empilha 0.0.

**FC?** Desempilha número, e empilha 1.0 caso o sinalizador de mesmo número esteja desligado, senão empilha 0.0.

### 6.3.6 Sequências

Sequências são objetos compostos de outros objetos. Cadeias de caracteres, listas, vetores e programas se enquadram nessa categoria. Alguns comandos desta seção se aplicam a um ou mais tipos de sequência.

**→STR** Desempilha objeto e empilha sua conversão para uma cadeia de caracteres.

**STR→** Desempilha cadeia de caracteres e empilha a sua conversão para um objeto.

**SREPL** Desempilha cadeias de caracteres *s*, *f* e *r*, e empilha uma cadeia de caracteres que é o resultado de substituir todas as ocorrências de *f* em *s* por *r*, e por fim empilha o número de substituições efetuadas.

**→LIST** Desempilha número *n*, e então desempilha *n* objetos, empilhando uma lista formada por todos eles, em ordem decrescente de profundidade.

**AXL** Desempilha objeto e empilha um vetor equivalente, caso seja uma lista, ou empilha uma lista equivalente, caso seja um vetor.

**TAIL** Desempilha objeto e empilha um objeto do mesmo tipo contendo todos os componentes do objeto original, com exceção do primeiro.

Desempilha objeto e empilha o primeiro objeto que o compõe.

**SIZE** Desempilha objeto e empilha o número de objetos que o compõe.

**GET** Desempilha  $l$  e  $n$ , e empilha o  $n$ -ésimo objeto componente de  $l$ .

**GETI** Desempilha  $l$  e  $n$ , caso  $n \leq \text{SIZE } l$  seja  $l.\theta$ , executa  $l \ n \ l \ + \ l \ n \ \text{GET } -64 \ . \ \text{CF}$ , do contrário executa  $l \ l \ . \ l \ n \ \text{GET } -64 \ . \ \text{SF}$ .

**PUT** Desempilha lista ou nome que referencia lista  $l$ , número  $n$  e objeto  $o$ . Se  $l$  é uma lista, empilha uma lista idêntica à  $l$ , com exceção de que nesta  $o$  ocupa a posição  $n$ . Se  $l$  é um nome, executa  $l \ \text{EVAL } n \ o \ \text{PUT } l \ \text{STO}$ .

**POS** Desempilha  $l$  e  $o$ , e empilha o índice posicional do primeiro objeto componente de  $l$  igual a  $o$ , ou  $\theta.\theta$  caso não haja.

**REVLIST** Desempilha objeto e empilha objeto de mesmo tipo com os mesmos objetos componentes, mas com a ordem invertida.

**$\Sigma$ LIST** Desempilha objeto e empilha a soma de todos os objetos que o compõe.

**DOLIST** Caso o objeto do topo da pilha seja um comando, um programa contendo exatamente um comando ou uma função do usuário e o objeto do segundo nível de profundidade não seja um número, executa  $l \ . \ \text{SWAP}$ . Então, desempilha número  $n$  e o programa  $f$ , e então desempilha as  $n$  listas de mesmo tamanho  $t$ ,  $l_1, \dots, l_n$ . Empilha lista vazia  $r$ . Em ordem crescente, para cada  $i$  natural no intervalo  $[1, t]$ , empilha o  $i$ -ésimo elemento de cada uma das  $l_1, \dots, l_n$ , nesta ordem, executa  $f$ , e por fim insere todos os objetos da pilha acima de  $r$  ao final da mesma.

**STREAM** Desempilha  $l$  e o programa  $p$ , empilha os dois primeiros objetos componentes de  $l$ , e executa  $p$ . Então, para cada objeto componente restante de  $l$ , a partir do terceiro e em ordem crescente, o empilha e executa  $p$ .

### 6.3.7 Nomes e Algébricos

Nomes e algébricos estão intimamente relacionados. Um nome que não referencia nenhum objeto é o algébrico que tem aquele nome como uma incógnita. Quando algébricos que representam expressões mais complexas são avaliados, nomes que aparecem neles são simplificados ou expandidos pelo CAS (conforme apropriado). Nomes podem referenciar quaisquer outros objetos, inclusive outros nomes e algébricos.

**STO** Desempilha objeto *o* e o nome *n*, e faz de *n* uma referência para o objeto *o*.

**STO+** Equivalente a **DUP ROT + EVAL SWAP STO**.

**RCL** Desempilha objeto então empilha o objeto referenciado por este. A referência pode ser um nome relativo ao diretório de trabalho atual ou uma lista contendo um caminho de diretório e o nome final (onde cada elemento é um nome que referencia um diretório — exceto possivelmente o último — seguindo a estrutura hierárquica de árvore). Pode-se rotular a referência para indicar à qual porta esta se aplica:

**H** Diretório HOME;

**0** ou **R** Memória RAM;

**1** ou **E** Memória RAM Extendida;

**2** ou **F** Memória Flash;

**3** ou **SD** Cartão SD;

**RCLVX** Equivalente a `:H:{CASDIR VX} RCL`. O nome `/HOME/CASDIR/VX` diz ao CAS quais nomes são considerados como variáveis em expressões polinomiais. Isso é significativo para o funcionamento de comandos de processamento algébrico.

**STOVX** Equivalente a **PUSH HOME CASDIR 'VX' STO POP**.

**DEGREE** Desempilha polinômio (na variável empilhada por **RCLVX**), e empilha seu grau polinomial.

**FDISTRIB** Desempilha algébrico, e empilha algébrico de valor equivalente com a distribuição da multiplicação sobre a soma completamente efetuada.

**LNAME** Empilha um vetor com todos os nomes que compõem o algébrico do topo da pilha.

| Desempilha algébrico **s** e lista **l**. Presumivelmente, em **l** os elementos de índice posicional ímpar  $n$  são nomes e os respectivos objetos de índice posicional par  $n + 1$  são objetos associados. Empilha a o algébrico que resulta de substituir em **s** todos os nomes em **l** por seus respectivos objetos associados.

**FXND** Desempilha algébrico e empilha lista contendo seu numerador e denominador, nesta ordem.

**SOLVE** Desempilha lista com equações e vetor de incógnitas. Empilha lista de soluções.

### 6.3.8 Diversos

Aqui estão alguns outros comandos usados que não se enquadram nas categorias das seções anteriores.

**PUSH** Empilha o estado atual dos sinalizadores e o caminho do diretório de trabalho atual na variável `/HOME/CASDIR/ENVSTACK` (uma lista de listas), para que sejam eventualmente recuperados por **POP**.

**POP** Desempilha e restaura o estado dos sinalizadores e o caminho do diretório de trabalho guardados pela execução mais recente de **PUSH** ainda não recuperada.

**HOME** Muda diretório de trabalho atual para `/HOME`.

**EVAL** Desempilha objeto e empilha a avaliação do mesmo.

**TEVAL** Desempilha objeto e empilha a avaliação do mesmo e o tempo de processamento em segundos gasto para fazê-lo.

**NOVAL** Empilha a si mesmo. Usado para representar a ausência de valor.

**TYPE** Desempilha objeto e empilha número que indica o seu tipo segundo a seguinte relação:

- 0 Número real de ponto flutuante;
- 1 Número complexo de ponto flutuante;
- 2 Cadeia de caracteres;

- 3 Vetor de números reais de ponto flutuante;
- 4 Vetor de números complexos de ponto flutuante;
- 5 Lista;
- 6 Nome global;
- 7 Nome local;
- 8 Programa;
- 9 Algébrico;
- 10 Inteiro binário;
- 11 Gráfico;
- 12 Objeto rotulado;
- 28 Inteiro;
- 29 Vetor de nomes e/ou algébricos;

## 6.4 Núcleo UserRPL

O Núcleo UserRPL é um conjunto de bibliotecas software livre, escrito pelo autor deste trabalho, que estendem a calculadora HP 50g. Seu objetivo é fornecer um conjunto versátil e poderoso de funções abrangendo uma ampla variedade de domínios de programação, a fim de facilitar e capacitar o desenvolvimento de software em UserRPL. O próprio núcleo é escrito na linguagem UserRPL e compilado usando o comando **CRLIB**. O código completo da biblioteca encontra-se publicamente disponível em um repositório do GitHub que pode ser acessado a partir de <https://oitofelix.github.io/8f-userrpl-kernel/>. Aqui reproduzimos e documentamos apenas as partes relevantes para o presente trabalho, relativos às bibliotecas base e polinomial. Programas cujos nomes começam com o caractere \$ são internos à biblioteca, isto é, não são visíveis nem podem ser usados por código externo à mesma.

### 6.4.1 Biblioteca Base

A biblioteca base lida com aspectos gerais de programação, como ordenação e processamento de listas.

## **\$INSERTSORTSTACK**

Desempilha números **a** e **b**, e então ordena os elementos restantes da pilha do nível de profundidade **a** até o nível de profundidade **b** usando um algoritmo de inserção simples. Esse algoritmo tem uma eficiência média de ordem quadrática, mas é frequentemente mais rápido que o Quicksort para um número suficientemente pequeno de elementos. Este comando existe para permitir que **\$QUICKSORTSTACK** forneça uma implementação híbrida do Quicksort. Veja também: **\$QUICKSORTSTACK**.

```
1  <<
2  → low high
3  <<
4  IF low high < THEN
5  high 1. - low
6  FOR I high I 1. +
7  FOR J
8  IF I PICK J 1. + PICK ←cmp EVAL 0. < THEN
9  I ROLL J ROLLD
10 END
11 -1. STEP
12 -1. STEP
13 END
14 >>
15 >>
```

## **\$QUICKSORTSTACK**

Desempilha números **a** e **b**, e então ordena os elementos restantes da pilha do nível de profundidade **a** até o nível de profundidade **b** usando um algoritmo Quicksort híbrido cuidadosamente implementado com as seguintes boas propriedades:

- Iterativo;
- Baseado em pilha;
- Modificação direta;
- Seleção de pivô baseada em média de três;
- Simplificação de intervalo de pivôs;
- Otimização de recursão sobre partição mínima;
- Ordenação por inserção para partições pequenas;

Veja também: [\\$INSERTSORTSTACK](#).

```
1  <<
2  NOVAL NOVAL NOVAL NOVAL NOVAL NOVAL NOVAL
3  NOVAL NOVAL NOVAL NOVAL NOVAL NOVAL NOVAL NOVAL
4  → low high highest nparts plow phigh top p c I
5  | p1i p2i p3i p1 p2 p3 dh1
6  <<
7  0. 'nparts' STO
8  high 1. + 'highest' STO
9
10 DO
11  CASE
12  @fall back to insertion sort for small partitions@
13  low high < high low - 10. < AND THEN
14  | low high $INSERTSORTSTACK
15  | 0. DUP 'low' STO 'high' STO
16  END
17
18  @quick sort algorithm@
19  low high < THEN
20  | @Median of three pivot selection@
21  | high low - 'dh1' STO
22  | dh1 RAND * 0. RND low + DUP 'p1i' STO PICK 'p1' STO
23  | dh1 RAND * 0. RND low + DUP 'p2i' STO PICK 'p2' STO
24  | dh1 RAND * 0. RND low + DUP 'p3i' STO PICK 'p3' STO
25
26  CASE
27  | p2 p1 ←cmp EVAL 0. ≤ p1 p3 ←cmp EVAL 0. ≤ AND
28  | p3 p1 ←cmp EVAL 0. ≤ p1 p2 ←cmp EVAL 0. ≤ AND OR
29  | THEN p1i END
30  | p1 p2 ←cmp EVAL 0. ≤ p2 p3 ←cmp EVAL 0. ≤ AND
31  | p3 p2 ←cmp EVAL 0. ≤ p2 p1 ←cmp EVAL 0. ≤ AND OR
32  | THEN p2i END
33  | p3i
34  END
35
36  @initialize pivot ranges and top comparison limit@
37  DUPDUP 'plow' STO 'phigh' STO PICK 'p' STO
38  high 'top' STO
39
40  @partition the lower section@
41  low 'I' STO
42  WHILE I plow < REPEAT
43  | I PICK p ←cmp EVAL 'c' STO
44  | CASE
45  | | c 0. < THEN
46  | | I ROLL high ROLLD
47  | | 'plow' 1. STO-
48  | | 'phigh' 1. STO-
49  | | 'top' 1. STO-
50  | | END
51  | | c 0. SAME THEN
52  | | I ROLL plow ROLLD
53  | | 'plow' 1. STO-
54  | | END
55  | | 'I' 1. STO+
56  | END
```

```

57      END
58
59      @partition the upper section@
60      IF phigh 1. + top ≤ THEN
61          phigh 1. + top
62          FOR I
63              I PICK p ←cmp EVAL 'c' STO
64              CASE
65                  c 0. > THEN
66                      I ROLL low ROLL D
67                      'plow' 1. STO+
68                      'phigh' 1. STO+
69                  END
70                  c 0. SAME THEN
71                      I ROLL plow ROLL D
72                      'phigh' 1. STO+
73                  END
74              END
75          NEXT
76      END
77
78      @recurse into the smaller section and delay
79      @recursion into the other one
80      IF plow 1. - low - high phigh 1. + - ≤ THEN
81          @delay recursion into upper section@
82          phigh 1. + highest ROLL D
83          high highest ROLL D
84          'nparts' 1. STO+
85          @recurse into lower section@
86          plow 1. - 'high' STO
87      ELSE
88          @delay recursion into lower section@
89          low highest ROLL D
90          plow 1. - highest ROLL D
91          'nparts' 1. STO+
92          @recurse into upper section@
93          phigh 1. + 'low' STO
94      END
95  END
96
97  @recurse into most recently delayed larger section@
98  nparts 0. > THEN
99      highest ROLL 'high' STO
100     highest ROLL 'low' STO
101     'nparts' 1. STO-
102  END
103  END
104  UNTIL nparts 0. SAME high low ≤ AND END
105  >>
106  >>

```



## QUICKSORT

Desempilha a lista `l` e o programa de comparação `c`, e então ordena `l` em ordem crescente segundo `c`. O programa de comparação deve desempilhar dois objetos e empilhar um número negativo, zero ou positivo, caso o objeto mais profundo seja menor, igual ou maior que o objeto mais raso, respectivamente. Para ordenar em ordem decrescente basta executar `REVLIST` subsequentemente. Veja também: `REVLIST`.

```
1  <<
2  NOVAL → list ←cmp size
3  <<
4  list 'size' STO
5  1. size $QUICKSORTSTACK
6  size →LIST
7  >>
8  >>
```

## SUMLIST

Desempilha lista `l`, e se esta tiver dois ou mais elementos, executa `l ΣLIST`. Caso contrário, se tiver um único elemento executa `l`, senão empilha `0`.

```
1  <<
2  DUP SIZE
3  CASE
4  DUP 1. > THEN DROP ΣLIST END
5  DUP 1. SAME THEN DROP HEAD END
6  DROP 0
7  END
8  >>
```

## PRODLIST

Desempilha lista `l`, e se esta tiver dois ou mais elementos, executa `l ΠLIST`. Caso contrário, se tiver um único elemento executa `l`, senão empilha `1`.

```
1  <<
2  DUP SIZE
3  CASE
4  DUP 1. > THEN DROP ΠLIST END
5  DUP 1. SAME THEN DROP HEAD END
6  DROP 1
7  END
8  >>
```

## DOLISTX

Comporta-se como **DOLIST**, a menos que as listas sejam vazias. Neste caso desempilha todos os objetos destinados ao processamento do **DOLIST** e empilha uma lista vazia.

```
1  <<
2  | 1. → depth
3  | <<
4  | | IF PICK3 {} SAME
5  | | THEN OVER 2. + DROPN {}
6  | | ELSE
7  | | | DEPTH PICK3 2. + - 'depth' STO+ DOLIST
8  | | | IF DEPTH depth < THEN {} END
9  | | END
10 | >>
11 >>
```

## STREAMLIST

Desempilha lista **l** e programa **p**, e se **l** tiver dois ou mais elementos, executa **l p STREAM**. Caso contrário, **l .**

```
1  <<
2  | IF OVER SIZE 1. >
3  | THEN STREAM
4  | ELSE DROP HEAD
5  | END
6  >>
```

## PAIRLIST

Desempilha duas listas de mesmo tamanho e empilha uma lista cujos elementos de posição ímpar são sequencialmente os elementos do objeto mais profundo, e os elementos de posição par são sequencialmente os elementos do objeto mais raso.

```
1  <<
2  | → 11 12
3  | <<
4  | | 11 12 2. <<>> DOLIST
5  | >>
6  >>
```

## UNPAIRLIST

Inverso de **PAIRLIST**. Isso significa que **PAIRLIST UNPAIRLIST** e **UNPAIRLIST PAIRLIST** não fazem nada.

```

1  <<
2  | {} {} → 1 11 12
3  <<
4  | WHILE 1 SIZE 0. > REPEAT
5  |   11 1 HEAD SWAP 1. + →LIST '11' STO
6  |   1 TAIL '1' STO
7  |   12 1 HEAD SWAP 1. + →LIST '12' STO
8  |   1 TAIL '1' STO
9  | END
10 | 11 12
11 >>
12 >>

```

## 6.4.2 Biblioteca Polinomial

A biblioteca polinomial estende o CAS da calculadora para efetuar processamento polinomial multivariado.

### STOPOLYVX

Desempilha lista de nomes de variáveis e a armazena em `/HOME/CASDIR/POLYVX`. Por padrão, a lista de variáveis é `{X}`. Todos os nomes de variáveis (e somente esses) são considerados pelo CAS como variáveis polinomiais simbólicas e a ordem em que estão listadas determina a ordenação de variáveis usada. Os nomes de variáveis devem ser listados em ordem decrescente. Veja também: `RCLPOLYVX`, `STOPOLYORD`.

```

1  <<
2  | PUSH HOME CASDIR 'POLYVX' STO POP
3  >>

```

### RCLPOLYVX

Empilha a lista de nomes armazenados em `/HOME/CASDIR/POLYVX`, ou `{X}`, caso não exista. Veja também: `STOPOLYVX`.

```

1  <<
2  | IFERR :H:{CASDIR POLYVX} RCL THEN DROP {X} END
3  >>

```

## STOPPOLYORD

Desempilha função de comparação de representação monomial interna e a armazena em /HOME/CASDIR/POLYORD. Por padrão, a função de comparação monomial é `<< PLEX >>`. As seguintes funções de comparação já estão implementadas para uso: `PLEX`, `PREVLEX`, `PGRLEX` e `PGREVLEX`. O usuário pode também implementar a sua própria. Veja também: `RCLPOLYORD`, `STOPPOLYVX`, `PLEX`, `PREVLEX`, `PGRLEX`, `PGREVLEX`.

```
1 <<
2 | PUSH HOME CASDIR 'POLYORD' STO POP
3 >>
```

## RCLPOLYORD

Empilha a função de comparação de representação monomial interna armazenada em /HOME/CASDIR/POLYORD, ou `<< PLEX >>`, caso não exista. Veja também: `STOPPOLYORD`.

```
1 <<
2 | IFERR :H:{CASDIR POLYORD} RCL THEN DROP <<PLEX>> END
3 >>
```

## TDEG

Desempilha polinômio e empilha seu grau total, que é definido como o máximo das somas dos expoentes de cada um de seus monômio. Se o polinômio for nulo, retorna um número negativo indefinido.

```
1 <<
2 | $POLY→LIST 1. <<TAIL SUMLIST>> DOLIST <<MAX>> STREAMLIST
3 >>
```

## MDEG

Desempilha polinômio e empilha seu multigrau, que é definido como a ênupla de expoentes de seu monômio máximo.

```
1 <<
2 | $POLY→LIST $PORDER DUP SIZE GET TAIL
3 >>
```

## LCOEF

Desempilha polinômio e empilha seu coeficiente líder, que é definido como o coeficiente de seu monômio máximo.

```
1 <<
2 | $POLY→LIST $PORDER DUP SIZE GET HEAD
3 >>
```

## LMONO

Desempilha polinômio e empilha seu monômio líder, que é definido como o produto de seu monômio máximo pelo inverso do coeficiente deste.

```
1 <<
2 | $POLY→LIST $PORDER DUP SIZE GET TAIL 1 SWAP + $LIST→MONO
3 >>
```

## LTERM

Desempilha polinômio e empilha seu termo principal, que é definido como seu monômio máximo.

```
1 <<
2 | $POLY→LIST $PORDER DUP SIZE GET $LIST→MONO
3 >>
```

## \$PORDER

Desempilha representação polinomial interna e empilha representação polinomial interna equivalente ordenada crescentemente em relação à função de comparação de representação monomial interna dada por /HOME/CASDIR/POLYORD. Veja também: **PORDER**.

```
1 <<
2 | RCLPOLYORD QUICKSORT
3 >>
```

## PORDER

Desempilha um polinômio e empilha o polinômio equivalente ordenado crescentemente em relação à função de comparação de representação monomial interna dada por /HOME/CASDIR/POLYORD. Veja também: **STOPOLYORD**, **\$PORDER**.

```

1  <<
2  | $POLY→LIST $PORDER $LIST→POLY
3  >>

```

## PLEX

Desempilha representações monomiais internas  $A$  e  $B$ , e empilha o resultado da comparação de ambas segundo a ordem lexicográfica. Sejam

$A = \{c_1 e_{1,1} \cdots e_{1,n}\}$  e  $B = \{c_2 e_{2,1} \cdots e_{2,n}\}$ . Nesta ordem,  $A > B$  se a entrada não nula mais à esquerda da lista  $\{e_{1,1} - e_{2,1} \cdots e_{1,n} - e_{2,n}\}$  for positiva,  $A < B$  se for negativa e  $A = B$ , caso não exista tal entrada. Veja também: **QUICKSORT**, **\$MONO→LIST**.

```

1  <<
2  | TAIL SWAP TAIL SWAP - 1.
3  | WHILE GETI DUP NOT -64. FC? AND REPEAT DROP END
4  | UNROT DROP2
5  >>

```

## PREVLEX

Desempilha representações monomiais internas  $A$  e  $B$ , e empilha o resultado da comparação de ambas segundo a ordem lexicográfica reversa. Sejam  $A = \{c_1 e_{1,1} \cdots e_{1,n}\}$  e  $B = \{c_2 e_{2,1} \cdots e_{2,n}\}$ . Nesta ordem,  $A > B$  se a entrada não nula mais à direita da lista  $\{e_{1,1} - e_{2,1} \cdots e_{1,n} - e_{2,n}\}$  for negativa,  $A < B$  se for positiva e  $A = B$ , caso não exista tal entrada.

Veja também: **QUICKSORT**, **\$MONO→LIST**.

```

1  <<
2  | TAIL SWAP TAIL SWAP - REVLIST 1.
3  | WHILE GETI DUP NOT -64. FC? AND REPEAT DROP END
4  | UNROT DROP2 NEG
5  >>

```

## PGRLEX

Desempilha representações monomiais internas  $A$  e  $B$ , e empilha o resultado da comparação de ambas segundo a ordem lexicográfica graduada. Sejam  $A = \{c_1 e_{1,1} \cdots e_{1,n}\}$  e  $B = \{c_2 e_{2,1} \cdots e_{2,n}\}$ . Nesta ordem,  $A > B$  se  $e_{1,1} + \cdots + e_{1,n} > e_{2,1} + \cdots + e_{2,n}$  ou  $e_{1,1} + \cdots + e_{1,n} = e_{2,1} + \cdots + e_{2,n}$  e  $A \text{ B PLEX } 0 >$  for  $1.0$ . Veja também: **QUICKSORT**, **\$MONO→LIST**, **PLEX**.

```

1  <<
2  | IF DUP2 TAIL SUMLIST SWAP TAIL SUMLIST SWAP - DUP
3  | THEN UNROT DROP2
4  | ELSE DROP PLEX
5  | END
6  >>

```

### PGREVLEX

Desempilha duas representações monomiais internas  $A$  e  $B$ , e empilha o resultado da comparação de ambas segundo a ordem lexicográfica graduada reversa. Sejam  $A = \{c_1 e_{1,1} \cdots e_{1,n}\}$  e  $B = \{c_2 e_{2,1} \cdots e_{2,n}\}$ . Nesta ordem,  $A > B$  se  $e_{1,1} + \cdots + e_{1,n} > e_{2,1} + \cdots + e_{2,n}$  ou  $e_{1,1} + \cdots + e_{1,n} = e_{2,1} + \cdots + e_{2,n}$  e  $A \text{ B } \text{PREVLEX } 0 >$  for  $1.0$ . Veja também: [QUICKSORT](#), [\\$MONO→LIST](#), [PLEX](#).

```

1  <<
2  | IF DUP2 TAIL SUMLIST SWAP TAIL SUMLIST SWAP - DUP
3  | THEN UNROT DROP2
4  | ELSE DROP PREVLEX
5  | END
6  >>

```

### PDIVORDER

Desempilha dois polinômios e empilha o resultado da comparação do monômio líder de ambos segundo a função de comparação empilhada por [RCLPOLYORD](#). Veja também: [RCLPOLYORD](#), [QUICKSORT](#).

```

1  <<
2  | → p1 p2
3  | <<
4  | | p1 $POLY→LIST $PORDER DUP SIZE GET
5  | | p2 $POLY→LIST $PORDER DUP SIZE GET
6  | | RCLPOLYORD EVAL
7  | >>
8  >>

```

### \$COEF

Desempilha um monômio e empilha seu coeficiente.

```

1  <<
2  | RCLPOLYVX 1 OVER SIZE NDUPN →LIST PAIRLIST | EVAL
3  >>

```

### \$MONO→LIST

Desempilha monômio e empilha a representação monomial interna correspondente. Nesta o monômio é representado por uma lista onde o primeiro elemento é o coeficiente monomial e os elementos subsequentes são as potências das variáveis monomiais declaradas em /HOME/CASDIR/POLYVX, na ordem definida. Veja: [STOPOLYVX](#), [\\$LIST→MONO](#).

```
1  <<
2  RCLVX NOVAL → M VXB M.LNAME
3  <<
4  M $COEF
5
6  M LNAME NIP
7  IF DUP TYPE 5. ≠ THEN AXL END
8  'M.LNAME' STO
9
10 RCLPOLYVX 1.
11 DO GETI
12 IF DUP M.LNAME SWAP POS
13 THEN STOVX M DEGREE
14 ELSE DROP 0
15 END UNROT
16 UNTIL -64. FS? END
17
18 DROP SIZE 1. + →LIST
19
20 VXB STOVX
21 >>
22 >>
```

### \$LIST→MONO

Desempilha representação monomial interna e empilha monômio correspondente. Veja também:

[\\$LIST→MONO](#).

```
1  <<
2  | DUP HEAD SWAP RCLPOLYVX SWAP TAIL ^ PRODLIST *
3  >>
```

### \$POLY→LIST

Desempilha polinômio e empilha a representação polinomial interna correspondente. Tal representação é uma lista de monômios na representação monomial interna, conforme descrito em [\\$MONO→LIST](#). Veja também: [\\$LIST→POLY](#), [\\$MONO→LIST](#).

```
1  <<
2  | NOVAL 1. NOVAL NOVAL → P L I A B
```



```

3  <<
4  | P FDISTRIB →STR "+" "' ' " SREPL DROP "(" "' SREPL DROP ")" "' SREPL DROP
5  | "{ " SWAP + "}" + STR→
6  | 1. '$MONO→LIST' DOLIST
7  >>
8  >>

```

### \$LIST→POLY

Desempilha representação polinomial interna e empilha o polinômio correspondente. Veja também: \$POLY→LIST.

```

1  <<
2  | 1. << $LIST→MONO >> DOLIST SUMLIST
3  >>

```

### \$MONODIV

Desempilha duas representações monomiais internas e empilha 1.0 caso a mais profunda seja divisível pela mais rasa, senão empilha 0.0.

```

1  <<
2  | $MONO→LIST TAIL SWAP $MONO→LIST TAIL - 1.
3  | WHILE GETI DUP 0. ≥ -64. FC? AND REPEAT DROP END
4  | 0. ≥ UNROT DROP2
5  >>

```

### MONOMIALS

Desempilha polinômio e empilha lista de seus monômios.

```

1  <<
2  | → p
3  | <<
4  | | p $POLY→LIST 1. << $LIST→MONO >> DOLIST
5  | >>
6  >>

```

### PDIV

Desempilha polinômio e lista de divisores e empilha lista quociente e lista dos restos da divisão.

```

1  <<
2  | NOVAL 0 NOVAL NOVAL NOVAL NOVAL NOVAL
3  | → P D Q R I M D.SIZE I.D.LTERM P.LTERM
4  | <<

```

```

5  D SIZE 'D.SIZE' STO
6  0 D.SIZE NDUPN →LIST 'Q' STO
7  WHILE P 0 SAME NOT REPEAT
8    1. 'I' STO
9    0. 'M' STO
10  WHILE I D.SIZE ≤ M NOT AND REPEAT
11    'D' I GET LTERM 'I.D.LTERM' STO
12    P LTERM 'P.LTERM' STO
13    IF I.D.LTERM 0 SAME NOT
14      I.D.LTERM P.LTERM $MONODIV AND THEN
15      'Q' I DUP2 GET P.LTERM I.D.LTERM / + EVAL PUT
16      P P.LTERM I.D.LTERM / 'D' I GET * - EVAL 'P' STO
17      1. 'M' STO
18    ELSE 1. 'I' STO+
19  END
20 END
21 IF M NOT THEN
22   R P.LTERM + EVAL 'R' STO
23   P P.LTERM - EVAL 'P' STO
24 END
25 END
26 Q R
27 >>
28 >>

```

### \$MONOLCM

Desempilha dois monômios e empilha seu menor múltiplo comum.

```

1  <<
2  → m1 m2
3  <<
4  m1 $MONO→LIST TAIL
5  m2 $MONO→LIST TAIL
6  << MAX >> DOLIST 1 SWAP + $LIST→MONO
7  >>
8  >>

```

### SPOLY

Desempilha dois polinômios e empilha seu polinômio S.

```

1  <<
2  NOVAL NOVAL NOVAL → p1 p2 lt1 lt2 lcm
3  <<
4  @ compute leading term (list form)
5  p1 LTERM 'lt1' STO
6  p2 LTERM 'lt2' STO
7
8  @ compute least common multiple of the leading terms
9  lt1 lt2 $MONOLCM 'lcm' STO
10
11 @ compute S-polynomial

```

```

12 | | lcm lt1 / p1 *
13 | | lcm lt2 / p2 *
14 | | - EVAL
15 | >>
16 | >>

```

## \$POLYINT

Desempilha polinômio  $p$  e empilha número  $n$ , tal que  $p \cdot n$  tem apenas coeficientes inteiros.

```

1 | <<
2 | | → P
3 | | <<
4 | | | P MONOMIALS 1. << $COEF FXND >> DOLIST UNPAIRLIST
5 | | | << LCM >> STREAMLIST SWAP
6 | | | << GCD >> STREAMLIST /
7 | | | ABS P LCOEF SIGN *
8 | | >>
9 | >>

```

## POLYEQ

Desempilha dois polinômios e empilha 1.0 caso ambos sejam iguais, senão empilha 0.0.

```

1 | <<
2 | | - EVAL 0 SAME
3 | >>

```

## POSPOLY

Desempilha lista  $l$  e polinômio  $p$ , e empilha o índice posicional de  $p$  em  $l$  caso aquele esteja neste, senão empilha 0.0.

```

1 | <<
2 | | 0. 0. → l p r l.SIZE
3 | | <<
4 | | | l SIZE 'l.SIZE' STO
5 | | | IF l.SIZE 0. > THEN
6 | | | | l l.SIZE
7 | | | | FOR I
8 | | | | | IF 'l' I GET p POLYEQ THEN
9 | | | | | | I 'r' STO
10 | | | | | | l.SIZE 'I' STO+
11 | | | | | END
12 | | | | NEXT
13 | | | | END
14 | | | r
15 | | >>
16 | >>

```

## POSPOLYPAIR

Desempilha lista l e par polinomial {p1, p2} e empilha o índice posicional do par em l caso aquele esteja neste (desconsiderando ordem), senão empilha 0.0.

```
1  <<
2  0. 0. 0 0 0 0 → l p r l.SIZE p1 p2 lp1 lp2
3  <<
4  p {'p1' 'p2'} STO
5  l SIZE 'l.SIZE' STO
6  IF l.SIZE 0. > THEN
7  | l.SIZE
8  | FOR I
9  | | 'l' I GET {'lp1' 'lp2'} STO
10 | | IF p1 lp1 POLYEQ p2 lp2 POLYEQ AND
11 | | | p1 lp2 POLYEQ p2 lp1 POLYEQ AND OR
12 | | THEN
13 | | | I 'r' STO
14 | | | l.SIZE 'I' STO+
15 | | END
16 | NEXT
17 END
18 r
19 >>
20 >>
```

# Capítulo 7

## Implementação do Algoritmo de Buchberger

Aqui implementamos o algoritmo de Buchberger dado no capítulo 5 usando as funcionalidades documentadas e definidas nas seções anteriores — providas pelo ambiente nativo UserRPL da calculadora e pelas bibliotecas do Núcleo UserRPL. Procura-se espelhar a especificação do algoritmo original tanto quanto possível, para que ambos possam ser comparados lado a lado.

### BUCHBERGER

```
1  <<
2  {} {} {} 0 0 0 → ←R ←P ←G ←B f1 f2 h
3  <<
4  $ReduceAll $NewBasis
5  WHILE ←B {} ≠ REPEAT
6  | ←B HEAD {'f1' 'f2'} STO
7  | ←B TAIL '←B' STO
8  | IF
9  | | f1 f2 $Criterion1 NOT
10 | | f1 f2 $Criterion2 NOT AND
11 | THEN
12 | | ←G f1 f2 SPOLY $NormalForm 'h' STO
13 | | IF h 0 SAME NOT THEN
14 | | | ←G 1. << → g << IF h LTERM g LTERM $MONODIV THEN g END >> >> DOLISTX
15 | | | '←R' STO
16 | | | h 1. →LIST '←P' STO
17 | | | ←G 1. << → g << IF ←R g POSPOLY NOT THEN g END >> >> DOLISTX
18 | | | '←G' STO
19 | | | ←B 1. << → b << IF ←R b HEAD POSPOLY ←R b TAIL POSPOLY OR NOT
20 | | | THEN b END >> >> DOLISTX '←B' STO
21 | | $ReduceAll $NewBasis
22 | END
23 END
24 END
```

```

25 | IF ←G {} SAME THEN {0}
26 | ELSE ←G 1. << DUP $POLYINT * EVAL >> DOLIST
27 | END
28 | >>
29 | >>

```

## \$ReduceAll

```

1 <<
2 0 {} {} → h G0 P0
3 <<
4 WHILE ←R {} ≠ REPEAT
5   ←R HEAD 'h' STO
6   ←R TAIL '←R' STO
7   ←G ←P 1. << → p << IF ←G p POSPOLY NOT THEN p END >> >>
8   | DOLISTX + h $NormalForm 'h' STO
9   IF h 0 SAME NOT THEN
10    ←G 1. << → g << IF h LTERM g LTERM $MONODIV
11    | THEN g END >> >> DOLISTX 'G0' STO
12    ←P 1. << → p << IF h LTERM p LTERM $MONODIV
13    | THEN p END >> >> DOLISTX 'P0' STO
14    ←G 1. << → g << IF G0 g POSPOLY NOT THEN g END >> >>
15    | DOLISTX '←G' STO
16    ←P 1. << → p << IF P0 p POSPOLY NOT THEN p END >> >>
17    | DOLISTX '←P' STO
18    G0 1. << → g0 << IF ←R g0 POSPOLY NOT P0 g0 POSPOLY NOT
19    | AND THEN g0 END >> >> DOLISTX
20    P0 1. << → p0 << IF ←R p0 POSPOLY NOT G0 p0 POSPOLY NOT
21    | AND THEN p0 END >> >> DOLISTX
22    + '←R' STO+
23    ←B 1. << → b << IF G0 b HEAD POSPOLY G0 b TAIL POSPOLY
24    | OR NOT THEN b END >> >> DOLISTX '←B' STO
25    IF ←P h POSPOLY NOT THEN ←P h + '←P' STO END
26  END
27 END
28 >>
29 >>

```

## \$NewBasis

```

1 <<
2 0 0 0 0 {} {} → p g h k H K
3 <<
4 ←P 1. << → p << IF ←G p POSPOLY NOT THEN p END >> >> DOLISTX '←G' STO+
5 IF ←G SIZE 0 > THEN
6   1. ←G SIZE
7   FOR g.I
8     '←G' g.I GET 'g' STO
9     1. ←P SIZE
10    FOR p.I
11      '←P' p.I GET 'p' STO
12      IF g p POLYEQ NOT

```

```

13     ←B g p 2. →LIST POSPOLYPAIR NOT AND
14     THEN
15         g p 2. →LIST 1. →LIST '←B' STO+
16     END
17     NEXT
18     NEXT
19     END
20     ←B « $SelStrategy » QUICKSORT '←B' STO
21     ←G 'H' STO
22     {} 'K' STO
23     WHILE H {} ≠ REPEAT
24         H HEAD 'h' STO
25         H TAIL 'H' STO
26         ←G 1. « → g « IF g h POLYEQ NOT THEN g END » » DOLISTX
27         h $NormalForm 'k' STO
28         IF K k POSPOLY NOT THEN K k + 'K' STO END
29     END
30     K '←G' STO
31     »
32     »

```

### \$NormalForm

```

1     «
2     → G P
3     «
4     IF G {} ≠ THEN
5         P G « PDIVORDER » QUICKSORT
6         PDIV NIP
7     ELSE P
8     END
9     »
10    »

```

### \$Criterion1

```

1     «
2     0 0. → f1 f2 p c
3     «
4     IF ←G SIZE 0. > THEN
5         1. ←G SIZE
6         FOR p.I
7             '←G' p.I GET 'p' STO
8             IF
9                 f1 p POLYEQ NOT f2 p POLYEQ NOT AND
10                p LTERM f1 LTERM f2 LTERM $MONOLCM $MONODIV AND
11                ←B f1 p 2. →LIST POSPOLYPAIR NOT AND
12                ←B p f2 2. →LIST POSPOLYPAIR NOT AND
13            THEN
14                1. 'c' STO
15            END
16        NEXT

```

```

17 | | END
18 | | c
19 | >>
20 | >>

```

## \$Criterion2

```

1 <<
2 | → f1 f2
3 <<
4 | f1 LTERM f2 LTERM $MONOLCM f1 LMONO f2 LMONO * POLYEQ
5 >>
6 >>

```

## \$SelStrategy

```

1 <<
2 | 0 0 0 0 → p1 p2 p1f1 p1f2 p2f1 p2f2
3 <<
4 | p1 {'p1f1' 'p1f2'} STO
5 | p2 {'p2f1' 'p2f2'} STO
6 | p1f1 LTERM p1f2 LTERM $MONOLCM $MONO→LIST
7 | p2f1 LTERM p2f2 LTERM $MONOLCM $MONO→LIST
8 | RCLPOLYORD EVAL
9 >>
10 >>

```

## 7.1 Exemplos

Aqui listamos alguns exemplos do uso da implementação do algoritmo de Buchberger dada acima, unido aos recursos nativos do CAS da HP 50g, aplicados à resolução de problemas clássicos.

### 7.1.1 O problema da pertinência a um ideal

**Exemplo 7.1.** *Sejam  $I = \langle f_1, f_2 \rangle = \langle xz - y^2, x^3 - z^2 \rangle \in \mathbb{C}[x, y, z]$  e  $f = -4x^2y^2z^2 + y^6 + 3z^5$ . Queremos determinar se  $f \in I$ .*

**Entrada:**

```

1 <<
2 | @ EX1 - The Ideal Membership Problem
3 |

```



```

4   @ Buchberger Algorithm: Find Reduced Gröbner Basis
5   {X Y Z} STOPOLYVX
6   << PGRLEX >> STOPOLYORD
7   {'X*Z-Y^2' 'X^3-Z^2'}
8   BUCHBERGER
9
10  @ Solve Membership Problem
11  '-4*X^2*Y^2*Z^2+Y^6+3*Z^5'
12  SWAP
13  PDIV
14  >>
15  TEVAL

```

### Saída:

```

1   { 1 0 '-(4*2^2)' 0 0}
2   0
3   :s: 50,719

```

Como o resto da divisão de  $f$  pela base de Gröbner do ideal  $I$  é 0, só pode ser o caso que  $f \in I$ .

## 7.1.2 O problema de resolver equações polinomiais

**Exemplo 7.2.** *Queremos resolver o seguinte sistema polinomial:*

$$x^2 + y^2 + z^2 = 1$$

$$x^2 + z^2 = y$$

$$x = z$$

### Entrada:

```

1   <<
2   @ EX2 - The Problem of Solving Polynomial Equations (I)
3
4   @ Buchberger Algorithm: Find Reduced Gröbner Basis
5   {X Y Z} STOPOLYVX
6   << PLEX >> STOPOLYORD
7   {'X^2+Y^2+Z^2-1' 'X^2+Z^2-Y' 'X-Z'}
8   BUCHBERGER
9
10  @ Solve system of equations
11  AXL
12  ['X' 'Y' 'Z']
13  SOLVE
14  >>
15  TEVAL

```

### Saída:

```

1 {
2   ['X=-(\sqrt{-1+\sqrt{5}}/2)',
3    'Y=(-1+\sqrt{5})/2',
4    'Z=-(\sqrt{-1+\sqrt{5}}/2)']
5   ['X=\sqrt{-1+\sqrt{5}}/2',
6    'Y=(-1+\sqrt{5})/2',
7    'Z=\sqrt{-1+\sqrt{5}}/2']
8   ['X=-(i*\sqrt{1+\sqrt{5}}/2)',
9    'Y=-((1+\sqrt{5})/2)',
10    'Z=-(i*\sqrt{1+\sqrt{5}}/2)']
11  ['X=i*\sqrt{1+\sqrt{5}}/2',
12   'Y=-((1+\sqrt{5})/2)',
13   'Z=i*\sqrt{1+\sqrt{5}}/2']
14 }
15 :s: 12,1116

```

Encontramos portanto as 4 soluções complexas:

$$\begin{aligned}
 x &= -\frac{\sqrt{-1+\sqrt{5}}}{2}, & y &= \frac{-1+\sqrt{5}}{2}, & z &= -\frac{\sqrt{-1+\sqrt{5}}}{2} \\
 x &= \frac{\sqrt{-1+\sqrt{5}}}{2}, & y &= \frac{-1+\sqrt{5}}{2}, & z &= \frac{\sqrt{-1+\sqrt{5}}}{2} \\
 x &= -\frac{i\sqrt{1+\sqrt{5}}}{2}, & y &= -\frac{1+\sqrt{5}}{2}, & z &= -\frac{i\sqrt{1+\sqrt{5}}}{2} \\
 x &= \frac{i\sqrt{1+\sqrt{5}}}{2}, & y &= -\frac{1+\sqrt{5}}{2}, & z &= \frac{i\sqrt{1+\sqrt{5}}}{2}.
 \end{aligned}$$

**Exemplo 7.3.** Queremos resolver o seguinte sistema polinomial:

$$3x^2 + 2yz - 2x\lambda = 0$$

$$2xz - 2y\lambda = 0$$

$$2xy - 2z - 2z\lambda = 0$$

$$x^2 + y^2 + z^2 - 1 = 0$$

**Entrada:**

```

1 <<
2 @ EX3 - The Problem of Solving Polynomial Equations (II)
3
4 @ Buchberger Algorithm: Find Reduced Gröbner Basis
5 {\lambda X Y Z} STOPOLYVX
6 << PLEX >> STOPOLYORD
7 {'3*X^2+2*Y*Z-2*X*\lambda', '2*X*Z-2*Y*\lambda', '2*X*Y-2*Z-2*Z*\lambda', 'X^2+Y^2+Z^2-1'}
8 BUCHBERGER
9
10 @ Solve system of equations
11 AXL
12 ['\lambda', 'X', 'Y', 'Z']

```

13 | SOLVE  
 14 >>  
 15 TEVAL

Saída:

```

1 {
2 [ 'λ=3/2' 'X=1' 'Y=0' 'Z=0' ]
3 [ 'λ=-3/2' 'X=-1' 'Y=0' 'Z=0' ]
4 [ 'λ=0' 'X=0' 'Y=1' 'Z=0' ]
5 [ 'λ=0' 'X=0' 'Y=-1' 'Z=0' ]
6 [ 'λ=-4/3' 'X=-2/3' 'Y=1/3' 'Z=2/3' ]
7 [ 'λ=-4/3' 'X=-2/3' 'Y=-1/3' 'Z=-2/3' ]
8 [ 'λ=1/8' 'X=-3/8' 'Y=-(3*√22/16)' 'Z=√22/16' ]
9 [ 'λ=1/8' 'X=-3/8' 'Y=3*√22/16' 'Z=-(√22/16)' ]
10 [ 'λ=-1' 'X=0' 'Y=0' 'Z=1' ]
11 [ 'λ=-1' 'X=0' 'Y=0' 'Z=-1' ]
12 }
13 :s: 21711,6602
```

Encontramos portanto as 10 soluções reais:

$$\lambda = \frac{3}{2}, x = 1, y = 0, z = 0$$

$$\lambda = -\frac{3}{2}, x = -1, y = 0, z = 0$$

$$\lambda = 0, x = 0, y = 1, z = 0$$

$$\lambda = 0, x = 0, y = -1, z = 0$$

$$\lambda = -\frac{4}{3}, x = -\frac{2}{3}, y = \frac{1}{3}, z = \frac{2}{3}$$

$$\lambda = -\frac{4}{3}, x = -\frac{2}{3}, y = -\frac{1}{3}, z = -\frac{2}{3}$$

$$\lambda = \frac{1}{8}, x = -\frac{3}{8}, y = -\frac{3\sqrt{22}}{16}, z = \frac{\sqrt{22}}{16}$$

$$\lambda = \frac{1}{8}, x = -\frac{3}{8}, y = \frac{3\sqrt{22}}{16}, z = -\frac{\sqrt{22}}{16}$$

$$\lambda = -1, x = 0, y = 0, z = 1$$

$$\lambda = -1, x = 0, y = 0, z = -1.$$

### 7.1.3 O problema da implicitação

**Exemplo 7.4.** Queremos encontrar as equações implícitas que definem a variedade cujas equações paramétricas são:

$$x = t^4,$$

$$y = t^3,$$

$$z = t^2$$

**Entrada:**

```
1  <<
2  @ EX4 - The Implicitization Problem (I)
3
4  @ Buchberger Algorithm: Find Reduced Gröbner Basis
5  {T X Y Z} STOPOLYVX
6  << PLEX >> STOPOLYORD
7  {'T^2-Z' 'T*Y-Z^2' 'T*Z-Y' 'X-Z^2' 'Y^2-Z^3'}
8  BUCHBERGER
9  >>
10 TEVAL
```

**Saída:**

```
1  { 'T^2-Z' 'Y*T-Z^2' 'Z*T-Y' 'X-Z^2' 'Y^2-Z^3' }
2  :s: 58,1714
```

Tomando os dois elementos da base de Gröbner encontrada nos quais a variável  $t$  não ocorre, temos as seguintes equações implícitas para a variedade dada:

$$x = z^2$$

$$y^2 = z^3$$

**Exemplo 7.5.** Queremos encontrar as equações implícitas que definem a variedade cujas equações paramétricas são:

$$x = t + u,$$

$$y = t^2 + 2tu,$$

$$z = t^3 + 3t^2u$$

**Entrada:**

```

1  <<
2  @ EX5 - The Implicitization Problem (II)
3
4  @ Buchberger Algorithm: Find Reduced Gröbner Basis
5  {T U X Y Z} STOPOLYVX
6  << PLEX >> STOPOLYORD
7  {'X-T-U' 'Y-T^2-2*T*U' 'Z-T^3-3*T^2*U'}
8  BUCHBERGER
9  >>
10 TEVAL

```

### Saída:

```

1  {
2  '4*Z*X^3-3*Y^2*X^2-6*Z*Y*X+4*Y^3+Z^2'
3  '(2*Y^3-2*Z^2)*U-(4*Z*Y*X^2-(Y^3-2*Z^2)*X-5*Z*Y^2)'
4  '(2*Z*X-2*Y^2)*U+2*Z*X^2-Y^2*X-Z*Y'
5  '(Y*X-Z)*U-(Y*X^2+Z*X-2*Y^2)'
6  'T+U-X'
7  'U^2-(X^2-Y)'
8  '(2*X^2-2*Y)*U-(2*X^3-3*Y*X+Z)'
9  }
10 :s: 1059,2814

```

Tomando o único elemento da base de Gröbner encontrada no qual as variáveis  $t$  ou  $u$  não ocorrem, temos a seguinte equação implícita para a variedade dada:

$$4zx^3 - 3y^2x^2 - 6zyx + 4y^3 + z^2 = 0$$

# Capítulo 8

## Referências Bibliográficas

- [1] Cox, David A. and Little, John and O’Shea, Donal, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, (Undergraduate Texts in Mathematics), Springer-Verlag, 2007.
- [2] Buchberger, *Groebner Bases: An Algorithmic Method in Polynomial Ideal Theory*, in *Multidimensional Systems Theory*, ed. by N.K. Bose (D. Reidel Publishing, Dordrecht, 1985).
- [3] Hewlett-Packard Company, *HP 50G Graphing Calculator: User’s Guide*, (HP Invent, 2006).
- [4] Hewlett-Packard Company, *HP 50G Graphing Calculator: User’s Manual*, (HP Invent, 2006).
- [5] Hewlett-Packard Company, *HP 50g/49g+/48gII Graphing Calculator: Advanced User’s Reference Manual*, (HP Invent, 2009).