Capítulo 2: Procedimentos e algoritmos

Para estudar o processo de computação de um ponto de vista teórico, com a finalidade de caracterizar o que é ou não é computável, é necessário introduzir um modelo matemático que represente o que se entende por computação. Entretanto, cada estudioso do assunto tem seu modelo matemático favorito, e por isso há diversas definições essencialmente distintas. (Por exemplo, um programador FORTRAN tenderia certamente a afirmar que computável é exatamente aquilo que pode ser feito por um programa FORTRAN).

Diversos modelos foram apresentados, e podem ser estudados na literatura (veja, por exemplo, [BrLa74][1]). Neste curso veremos apenas dois destes modelos: as funções recursivas parciais e as máquinas de Turing. O modelo das funções recursivas parciais se baseia em idéias semelhantes às da programação funcional, enquanto o modelo das máquinas de Turing procura introduzir um computador elementar, com repertório de instruções e estrutura de memória com a maior simplicidade possível. Outros modelos, que não veremos aqui, são baseados em algoritmos de Markov (com algumas semelhanças com a linguagem SNOBOL), em linguagens de programação mais convencionais (como o FORTRAN citado acima), ou em outras arquiteturas de computadores ideais ou linguagens de programação.

O fato surpreendente a respeito disso é que (até hoje) todos os modelos usados (se não são completamente absurdos) concordam na definição do que quer dizer *"computável"*. Uma conjectura, enunciada por Alonzo Church, diz que *todos os modelos razoáveis do processo de computação, definidos e por definir, são equivalentes.* Essa conjectura é conhecida como a *tese de Church.* Por sua própria natureza, a tese de Church não admite nenhuma prova formal, mas até hoje todos os modelos propostos se mostraram equivalentes.

Neste capítulo, entretanto, não queremos ainda definir modelos matemáticos precisos, mas queremos apenas trabalhar com a idéia intuitiva do que quer dizer computável. Para isso, vamos introduzir, mais abaixo, *informalmente*, dois conceitos relacionados: o conceito de *procedimento*, e o conceito de *algoritmo*.

Codificação: Na discussão que se segue, levaremos em consideração apenas conjuntos enumeráveis. A razão para isto é que nenhum modelo "razoável" proposto para o processo de computação admite o tratamento de conjuntos não enumeráveis. Precisamos ter alguma garantia de que podemos representar de forma finita todos os elementos do conjunto considerado, e isso não seria possível no caso de conjuntos não enumeráveis. Por exemplo, o conjunto dos números reais não é enumerável, mas o conjunto dos números racionais é enumerável, e, por essa razão, o tratamento computacional de números reais é feito através de aproximações racionais.

Adicionalmente, sem perda de generalidade, é conveniente considerar que todos os conjuntos considerados ou são conjuntos de números naturais (subconjuntos de **Nat**) ou são linguagens em algum alfabeto Γ conhecido (subconjuntos de Γ^*). Sabemos que Γ^*, **Nat** e seus subconjuntos são enumeráveis, e que existe uma bijeção entre quaisquer

[1] *Walter S. Brainerd, Lawrence H. Landweber*, Theory of Computation, *John Wiley, 1974*

dois conjuntos enumeráveis infinitos. Isso significa que podemos, usando essa bijeção como codificação, usar valores em um conjunto enumerável qualquer para substituir valores em outro. Em particular, a codificação através de números naturais (*numeração*), e a descrição através de cadeias de símbolos (muitas vezes conhecidos como identificadores) são idéias familiares.

Procedimentos. Vamos definir um *procedimento* como sendo uma sequência *finita* de instruções, e definir *instrução* como uma operação *claramente descrita*, que pode ser executada *mecanicamente*, em *tempo finito*.

- *"mecanicamente"* quer dizer que não há dúvidas sobre o que deve ser feito;

- *"em tempo finito"* quer dizer que não há dúvidas de que a tarefa correspondente à instrução pode, em qualquer caso, ser levada até sua conclusão.

Para descrever um procedimento podemos usar uma linguagem natural, uma linguagem de programação, ou a linguagem normalmente usada em matemática. Frequentemente, usamos uma mistura de todas estas. Sobre a forma de descrever instruções e procedimentos, suporemos apenas que existe uma linguagem, comum a todos os interessados, em que instruções e procedimentos podem ser descritos sem ambiguidades.

Como exemplos, citamos:

- o algoritmo de Euclides para cálculo do máximo divisor comum de dois números naturais;
- um programa em FORTRAN que calcula a soma de dois números;
- a fórmula que calcula as raízes da equação do segundo grau.

Exemplo: O procedimento a seguir pára e diz *"sim"* se o número inteiro i, dado como entrada, for par e não negativo.

1. se i = 0, pare e diga *"sim"*.
2. diminua o valor de i de duas unidades.
3. vá para 1.

Note que se i for originalmente ímpar ou negativo, o procedimento não pára. Na definição de procedimento, dada acima, exigimos que cada instrução possa ser executada em tempo finito, mas não exigimos nada de semelhante para os procedimentos. Isso significa, em particular, que não poderíamos usar como instrução em algum outro procedimento uma chamada do procedimento do exemplo acima, uma vez que não podemos garantir sua parada em tempo finito, para qualquer valor da entrada.

□

Exemplo: O procedimento a seguir pára e diz *"sim"* se o número inteiro i for par e não negativo; pára e diz *"não"* nos demais casos.

1. se i = 0, pare e diga *"sim"*.
2. se i < 0, pare e diga *"não"*.
3. diminua o valor de i de duas unidades
4. vá para 1.

□

Algoritmo. Definimos um algoritmo como sendo um procedimento que sempre pára, quaisquer que sejam os valores de suas entradas.

O segundo exemplo de procedimento dado acima é, portanto, um algoritmo. Programas corretos normalmente são algoritmos; em geral, um programa que não pára para alguns valores de suas entradas é um programa incorreto, um "programa com loop", um "programa com erro de lógica". Em alguns casos, entretanto, isso não é verdade, e o desejado é, exatamente, que a execução do programa se estenda por um tempo ilimitado. O exemplo mais característico desse tipo de procedimento parece ser o de um sistema operacional: uma vez iniciada sua execução, ela continua (em condições normais) até que a máquina seja desligada.

A seguir temos outro exemplo de procedimento que não tem parada prevista:

Exemplo: O procedimento a seguir enumera as sequências de {a, b}*, emitindo (imprimindo, por exemplo) todas essas sequências.

1. Faça X = {ε}.
2. Emita todas as sequências de X.
3. Para cada $y \in$ X, acrescente a X as sequências *y*a e *y*b.
4. Vá para 2.

O procedimento acima emite todas as sequências formadas por a's e b's.

ε, ε, a, b, ε, a, b, aa, ab, ba, bb, ε, a, b, …

Cada sequência é emitida uma infinidade de vezes, como permitido numa enumeração.

□

Exemplo: Seja uma função f: **Nat** → **Nat**. Suporemos que está disponível um algoritmo para calcular f(i), a partir de qualquer i ∈ **Nat**. Considere o procedimento a seguir:

1. faça i = 0.
2. calcule j = f(i), usando o algoritmo dado
3. se j = k, emita i, e pare
4. incremente o valor de i de 1
5. vá para 2.

Note que o passo 2 só pode ser considerado uma instrução por causa da disponibilidade de um *algoritmo* para cálculo dos valores da função. O procedimento do exemplo aceita como entrada um valor k ∈ **Nat**, e só pára se existir um valor de i tal que f(i) = k. Em particular, o valor de i emitido é o menor possível.

□

Conjuntos recursivamente enumeráveis. Dizemos que um conjunto A é recursivamente enumerável (r.e.) se existe um procedimento que enumera os elementos

de A. Isto quer dizer que existe um procedimento que emite todos os elementos de A, possivelmente com repetições.

Exemplo: Em um dos exemplos anteriores vimos que existe um procedimento que enumera as sequências pertencentes ao conjunto A = { a, b }*. Logo, o conjunto A é recursivamente enumerável.

□

Exemplo: Seja f: **Nat** → **Nat**. O procedimento a seguir mostra que o contradomínio de f é recursivamente enumerável, supondo que existe um algoritmo que calcula o valor de f(i), a partir de i.

1. i := 0;
2. emita f(i);
3. i := i + 1;
4. vá para 2.

□

Conjuntos recursivos. Dizemos que um conjunto A é *recursivo* se existe um algoritmo que determina, para um valor arbitrário de sua entrada x, se x ∈ A ou se x ∉ A. Embora isso não seja estritamente necessário, podemos, para fixar as idéias, especificar que o algoritmo deve parar e dizer *"sim"* ou *"não"*, respondendo à pergunta "x ∈ A?".

Exemplo: O conjunto dos naturais pares é recursivo. Basta examinar o algoritmo a seguir, cuja entrada é um natural i:

1. Se i = 0, pare e diga *"Sim"*.
2. Se i = 1, pare e diga *"Não"*.
3. faça i := i - 2.
4. vá para 1.

□

Fato: Todo conjunto recursivo é recursivamente enumerável.

Dem.: Se o conjunto A ⊆ **Nat** é recursivo, existe um algoritmo α que, para cada entrada i determina se i ∈ A. Considere o procedimento a seguir:

1. i := 0.
2. execute α com entrada i.
3. se α respondeu *"Sim"*, emita i.
4. i := i + 1.
5. vá para 2.

Note que:

- i assume todos os valores naturais
- α sempre pára, qualquer que seja sua entrada i;
- α responde "Sim" exatamente para os valores de i que pertencem a A.

Portanto, os valores de i emitidos são exatamente aqueles pertencentes a A.

□

Fato: A classe dos conjuntos recursivos é fechada para as operações de união, interseção e complementação.
Dem.: *(união)* Sejam A e B conjuntos recursivos. Sejam α e β algoritmos que determinam pertinência em A e em B, respectivamente. Podemos construir um algoritmo γ que determina se $x \in A \cup B$, da seguinte forma:

1. execute α com entrada x.
2. se α respondeu *"Sim"*, responda *"Sim"* e pare.
3. execute β com entrada x.
4. pare e responda o que β respondeu.

(interseção) Sejam A, B, α, β como acima. Construa um algoritmo γ que determina se $x \in A \cap B$ da seguinte forma:

1. execute α com entrada x.
2. se α respondeu *"Não"*, responda *"Não"* e pare.
3. execute β com entrada x.
4. pare e responda o que β respondeu.

(complemento) Sejam A e α como acima. Construa um algoritmo γ que determina se $x \in \overline{A}$ = **Nat** - A da seguinte forma:

1. execute α com entrada x.
2. se α respondeu *"Sim"*, responda *"Não"* e pare.
3. se α respondeu *"Não"*, responda *"Sim"* e pare.

□

Fato: Um conjunto A é recursivamente enumerável se e somente se existe um procedimento que, com entrada x, pára e diz *"Sim"*, se $x \in A$, o que não acontece se $x \notin A$. Isto quer dizer que, se $x \notin A$, ou *(1)* o procedimento não pára, ou *(2)* pára, mas não diz *"Sim"*.

Dem.: ($\Rightarrow$) Se A é r.e., existe um procedimento α que enumera seus elementos. Construa um procedimento β que aceita um elemento x qualquer como entrada, modificando α da seguinte maneira: quando α emite um valor y, β testa se y = x. Se isso acontecer, β pára e diz *"Sim"*. Portanto, β dirá *"Sim"* exatamente quando sua entrada for emitida por α, ou seja quando for um elemento de A.

($\Leftarrow$) Seja α um procedimento que pára e diz *"Sim"* quando sua entrada $x \in A$, e que ou não pára, ou não diz *"Sim"*, quando $x \notin A$. Um procedimento β que enumera os elementos de A pode ser construído usando α, da seguinte maneira:

1. faça k = 0;
2. para cada x = 0, …, k execute (2.1) e (2.2).
    2.1. execute um passo (adicional) de α com entrada x
    2.2. se α parou e disse *"Sim"*, emita x.
3. faça k = k + 1.
4. vá para 2.

Note que é necessário executar "em paralelo" α com as várias entradas x porque, se os diversos valores de x fossem tentados sequencialmente, e α não parasse para algum valor de x, α nunca chegaria a ser executado com os valores subsequentes de x.

□

Fato: A classe dos conjuntos recursivamente enumeráveis é fechada para as operações de união e de interseção.

Dem.: *(união)* Sejam A e B conjuntos r.e. e sejam α e β procedimentos que enumeram os elementos de A e de B, respectivamente. Um procedimento *g* que enumera os elementos de $A \cup B$ executa α e β em paralelo é:

1. execute um passo de α.
2. se α emitiu i, emita i.
3. execute um passo de β.
4. se β emitiu i, emita i.
5. vá para 1.

Os elementos emitidos por γ são exatamente os elementos de A e os elementos de B, ou seja, os elementos de $A \cup B$.

(interseção) Sejam A, B, α e β como acima. O procedimento γ que enumera os elementos de $A \cap B$ executa em paralelo α e β, guardando os elementos já emitidos por α e por β nos conjuntos X e Y, respectivamente:

0. Faça $X = \varnothing$ e $Y = \varnothing$.
1. execute um passo de α.
2. se α emitiu i, acrescente i ao conjunto X.
3. execute um passo de β.
4. se β emitiu i, acrescente i ao conjunto Y.
5. emita os elementos comuns a X e a Y.
6. vá para 1.

Os elementos emitidos por γ são aqueles pertencentes aos conjuntos *finitos* X e Y, ou sejam aqueles já emitidos por α e por β.

□

Fato: Se um conjunto A e seu complemento $\overline{A}$ são ambos recursivamente enumeráveis, então A (e o complemento $\overline{A}$) são ambos recursivos.

Dem.: Já vimos que se A é r.e., existe um procedimento α que pára e diz *"Sim"*, exclusivamente quando sua entrada x é um elemento de A; como $\overline{A}$ também é r.e., existe um procedimento β que pára e diz *"Sim"* exclusivamente quando sua entrada x é um elemento de $\overline{A}$, ou seja, exatamente quando x *não é* um elemento de A.

Podemos construir um algoritmo γ que executa os procedimentos α e β em paralelo, com entrada x, para determinar se x pertence ou não a A. O procedimento a seguir é um algoritmo porque eventualmente um dos dois passos (2) ou (4) será executado: ou $x \in A$ ou $x \notin A$, não havendo terceira possibilidade.

1. execute um passo (adicional) de α com entrada x.
2. se α parou e disse *"Sim"*, pare e diga *"Sim"*.
3. execute um passo (adicional) de β com entrada x.
4. se β parou e disse *"Sim"*, pare e diga *"Não"*.
5. vá para 1.

□

(revisão de 27fev97)