Capítulo 8: O problema da parada.

Decidibilidade e computabilidade.

José Lucas Rangel

8.1 - Introdução.

Como observado no capítulo anterior, podemos substituir a definição informal de procedimento pela definição formal de máquina de Turing (mT). Já vimos que, dependendo da forma como a mT é construída, ela pode ser um procedimento gerador ou um procedimento reconhecedor; por outro lado, uma mT que sempre pára, qualquer que seja a sua entrada, é um algoritmo. Portanto, um conjunto recursivamente enumerável (r.e.) é uma linguagem reconhecida por uma mT; e um conjunto recursivo é uma linguagem aceita por uma mT que sempre pára.

Devemos, neste capítulo, apresentar exemplos de linguagens que mostrem que as definições de conjunto recursivamente enumerável e de conjunto recursivo são não triviais. Uma vez que já foi provada a inclusão da segunda classe na primeira, basta apresentar exemplos de conjuntos r.e. que não são recursivos, e de conjuntos enumeráveis que não são r.e. Efetivamente, o reconhecimento de um conjunto não recursivamente enumerável é, então, uma tarefa que não pode ser realizada por nenhuma máquina de Turing, por nenhum procedimento, ou seja, é uma tarefa incomputável.

Para que esses exemplos sejam apresentados, precisamos de alguns conceitos auxiliares.

8.2 - Enumeração de Máquinas de Turing.

Para definir uma enumeração (e uma numeração) de todas as máquinas de Turing (mT) vamos introduzir uma codificação, que vai representar cada mT $M = <K, \Sigma, \Gamma, \delta, i, F>$ através de uma cadeia em um alfabeto adequado Δ; a enumeração das mT, determinísticas e não determinísticas, será baseada em uma enumeração de Δ^*. Para isso é conveniente fazer algumas suposições:

Os estados de K estão ordenados: $q_0, q_1, q_2, \ldots$, de tal forma que $i = q_0$.

De forma semelhante, os símbolos de Γ, $s_0, s_1, s_2, \ldots$ estão ordenados, de forma que s_0 é o símbolo especial branco $\Diamond$.

Assim, para representar M, usaremos um alfabeto $\Delta = \{q, s, |, \#, L, R\}$ de forma que o estado q_i é representado por $q|^i$, e o símbolo s_i é representado por $s|^i$. A representação então usa as cadeias q, q|, q||, q|||, …, s, s|, s||, s|||, … para os estados e símbolos; note, em particular, que q é o estado inicial, e s é o símbolo branco.

Os símbolos #, L, e R de Δ são usados respectivamente como separador e como indicadores de movimento para a esquerda, e para a direita.

Para representar a função δ, devemos representar todos os valores de $\delta(q, s)$. Isso pode ser feito através de uma sequência de quíntuplas (q_i, s_j, q_k, s_l, r), com $r \in \{L, R\}$, que indicam que $(qk, sl, r) \in \delta(qi, sj)$.

A cadeia correspondente à tupla (qi, sj, qk, sl, r) em Δ* é q|i s|j q|k s|l r. Se M é uma mT não determinística, podemos ter várias tuplas com os mesmos valores nos dois primeiros elementos.

Por exemplo, se tivermos a tupla (q_1, s_2, q_3, s_4, L), indicando que (q_3, s_4, L) ∈ δ(q_1, s_2), a cadeia correspondente em Δ* será q|s||q|||s||||L. Para representar a função δ, concatenamos as representações das quíntuplas correspondentes a δ, separadas por #.

Para representar F, basta concatenar, sem separação, as cadeias que representam os estados. Por exemplo, F = {q1, q3, q5 } é representado por q|q|||q|||||. A representação de M se reduz apenas à representação de F e de δ, que podem ser separadas por ##.

Exemplo: Seja M = <K, Σ, Γ, δ, i, F>, onde

K = { p, q, r }
Σ = { a, b }
Γ = { ◊, a, b, X }
i = q
F = { r },

sendo δ dada por

δ(p, a) = {(q, a, R)}
δ(q, a) = {(q, X, R)}
δ(q, b) = {(r, a, L)}
δ(q, ◊) = {(r, X, L)}

Fazendo q_0 = q, q_1 = p, q_2 = r, s_0 = ◊, s_1 = a, s_2 = b, s_3 = X, temos a seguinte representação de M:

q||##q|s|qs|R#qs|qs|||R#qs||q||s|L#qsq||s|||L

Uma entrada aaab de M será codificada em Δ* como s|s|s|s||.

Com essa codificação (ou alguma outra codificação semelhante), é possível representar as mT's e suas entradas através de cadeias em Δ*, onde Δ é um alfabeto apropriado. Sabemos que Δ* é enumerável, e que podemos atribuir um natural a cada cadeia de Δ*, e fazer referência às cadeias de Δ* como x_0, x_1, x_2, … . Assim, podemos atribuir à mT representada por x_i o número i, e fazer a referência a ela como a i-ésima máquina de Turing M_i.

Entretanto, para que a notação M_i sempre represente uma mT, uma convenção adicional é necessária. Observe que nem todas as cadeias x_i de Δ* correspondem a mT's. Por exemplo, a cadeia LLL não descreve nenhuma mT. Para que a cada natural i corresponda uma mT M_i, vamos associar a cada i tal que x_i não descreve nenhuma máquina a máquina vazia ##, que não tem nenhum estado final, e nenhuma transição possível. Incidentalmente, ## aceita a linguagem vazia ∅.

Assim, podemos falar em M_i, para qualquer i: ou M é a máquina representada por x_i, ou x_i não representa nenhuma máquina, e Mi é a máquina representada por ##. Note que é fácil distinguir as cadeias de Δ* que representam mT's das demais, uma vez que elas formam um conjunto regular.

Semelhantemente, podemos enumerar as cadeias que representam entradas:

x_0, x_1, x_2 ...,

de forma que poderemos falar na i-ésima cadeia de entrada x_i.

Exercício: Escolha uma enumeração de Δ^*. Para essa enumeração, descreva algoritmos que permitam

dado um natural i, determinar a cadeia x_i de Δ^* correspondente.
dada uma cadeia x em ?*, determinar o natural i tal que $x = x_i$.
dada uma cadeia x_i em Δ^*, determinar se x_i é ou não a representação de uma mT.

Exercício: Para uma enumeração determinada de Δ^*,

(a) determine o número da sequência ##.
(b) determine qual a máquina M23.

Suporemos daqui para a frente que está fixada uma enumeração de todas as máquinas de Turing, em algum alfabeto Δ fixo.

$M_0, M_1, M_2, M_3, \ldots$

e de suas entradas,

$x_0, x_1, x_2, x_3, \ldots$

Codificação e decodificação. Podemos sempre supor que existem algoritmos com as seguintes finalidades:

dado um natural i, determinar a cadeia xi.
dado um natural i, determinar a representação de Mi.
dada uma cadeia x, determinar i tal que x = xi.
dada uma mT M, determinar i tal que M = Mi.

Faremos referência a esses algoritmos como algoritmos de codificação ou de decodificação. Como se trata de algoritmos, podemos supor que podem ser simulados por uma mT, ou usados por mT's como subrotinas.

8.3 - Máquina de Turing Universal.

Vamos agora mostrar como pode ser construída uma mT universal U, que tem a propriedade de poder simular qualquer outra mT. Por simplicidade, vamos descrever uma mT universal U não determinística. Para simular a computação de uma mT M qualquer, quando recebe como entrada a cadeia x, a idéia é a seguinte:

U recebe uma entrada $\alpha\$\beta$, onde α e β são as representações de M e x em algum alfabeto Δ;
U simula M com entrada x;
U aceita $\alpha\$\beta$ se M aceita x.

Para a simulação, U constrói uma representação de uma configuração de M em sua fita, e faz a simulação alterando essa configuração. A simulação parte da configuração inicial $q\alpha$ Para cada mudança de configuração $C_i \vdash C_{i+1}$, U deve identificar o estado e o símbolo lido em C_i, escolher uma entre as diversas transições previstas na função de transição δ que faz parte de β e construir a nova configuração C_{i+1}.

Como U é não determinística, pode decidir "não-deterministicamente" quando uma configuração final C_f de M foi atingida, e apenas verificar que o estado contido em C_f é um dos estados finais listados em β.

Todas as operações envolvidas são operações de cópia e de comparação de cadeias, e não oferecem nenhum problema maior. Naturalmente, a simulação de um passo da máquina M corresponde a um número considerável de passos de U, mas isto não é um problema.

Assim, a construção de U é possível. Frequentemente, é conveniente supor que a mT U recebe como entrada i$j, para indicar que deve simular Mi com entrada xj, sendo i e j representados em alguma base adequada. Os passos adicionais necessários são de codificação e decodificação, e podem, como já vimos, ser realizados por uma mT.

A mT U é frequentemente usada como uma subrotina, na construção de outras mT's, em geral através de frases como

"então, M simula M_i com entrada x_j e se M parar, ...".

Uma comparação que se faz frequentemente é a da máquina de Turing universal U com o modelo de von Neumann, em que se baseiam os computadores comuns. O modelo de von Neumann se caracteriza justamente por ter o programa armazenado na memória, juntamente com os dados, durante a execução.

Note que U é uma máquina de Turing, e, portanto, faz parte da enumeração M_0, M_1, M_2, M_3, … sendo, para algum natural u, a máquina M_u.

8.4 - Uma linguagem não recursivamente enumerável.

Vamos agora apresentar um exemplo de linguagem não recursivamente enumerável. Considere a linguagem L abaixo:

$L = \{ x_i \mid x_i \text{ não é aceita por } M_i \}$

Fato: L não é recursivamente enumerável.

Dem.: Vamos mostrar, por contradição, que L não é aceita por nenhuma máquina de Turing. Para isso, suponha que L é aceita pela mT M.

Como toda mT, M faz parte da enumeração das mT's, ou seja, para algum i, $M = M_i$, de forma que $L = L(M_i)$. Vamos verificar se $x_i \in L$.

- se tivermos $x_i \in L$, como $L = L(M_i)$, temos que M_i aceita x, e portanto $x_i \notin L$.
- se, alternativamente, tivermos $x_i \notin L$, ou seja, $x_i \notin L(Mi)$, naturalmente, M não aceita x_i, e portanto $x_i \in L$.

Estabelecida a contradição, concluímos que L não é aceita por nenhuma mT M, e que L não é recursivamente enumerável.

Para a linguagem L, portanto, não existem

- um procedimento reconhecedor de L
- um procedimento enumerador de L
- uma gramática, tipo 0 ou não, que gere L.

Fato: , o complemento de L, é uma linguagem recursivamente enumerável.

Dem.: Temos = { x_i | x_i é aceita por M_i }. Podemos descrever uma mT M que aceita L, da seguinte forma:

a partir de xi, M obtém a representação de Mi.

como foi descrito para a máquina universal U, M simula Mi com entrada xi.

se Mi parar e aceitar xi, M também pára e aceita xi.

Note que se M_i não pára, com entrada x_i, então M também não pára, com a mesma entrada

Fato: A linguagem $\overline{L}$ não é recursiva.

Dem.: Já vimos anteriormente que o complemento de uma linguagem recursiva também é uma linguagem recursiva. Como o complemento de $\overline{L}$ é L, que não é r.e., e portanto, também não é recursiva, $\overline{L}$ não pode ser recursiva.

A linguagem $\overline{L}$, portanto, tem uma gramática tipo 0, mas não sendo recursiva, também não pode ser sensível ao contexto, e não pode ter nenhuma gramática tipo 1.

8.5 - Problemas decidíveis e indecidíveis.

Para qualquer conjunto X não recursivo, a pergunta "$x \in X$?" não admite solução através de um algoritmo que aceite x como entrada e responda SIM ou NÃO corretamente à pergunta. Mas, do ponto de vista prático, a diferença entre um conjunto não recursivamente enumerável, e um conjunto recursivamente enumerável que não é recursivo pode ser considerada pequena. Com efeito, suponha que um conjunto L é recursivamente enumerável mas não recursivo, e que dispomos de uma mT M que reconhece L, mas, não pára quando sua entrada não pertence a L. Suponha que, com entrada x, M foi executada por, digamos, mil passos, e que não parou. Nada podemos responder à pergunta "$x \in L$?". Será que M vai parar nos próximos mil passos?

Uma das maneiras de dizer que uma linguagem L não é recursiva é dizer que o problema "$x \in L$?" não é decidível. De uma forma geral, um problema indecidível é um conjunto de questões que pode ser reduzido por codificação ao problema da pertinência em uma linguagem não recursiva, e, assim, não pode ser respondido por uma mT que sempre pára; se um problema pode ser reduzido por codificação ao problema de pertinência em uma linguagem recursiva, dizemos que é decidível.

Um ponto importante é que um problema indecidível sempre envolve uma família de questões, ou questões em que aparece um parâmetro. O problema P(i) - "M_i aceita x_i?" pode ser indecidível, mas o problema P(23) - "M_{23} aceita x_{23}?" é decidível. Como prova disso, oferecemos duas mTs: M_{sim} e $M_{não}$. M_{sim} sempre pára e aceita qualquer entrada (sempre responde SIM); $M_{não}$ sempre pára, mas nunca aceita qualquer de suas entradas (sempre responde NÃO). É irrelevante o fato de ser difícil ou trabalhoso descobrir qual das duas máquinas corresponde à resposta correta da pergunta "M_{23} aceita x_{23}?", mas certamente uma das duas resolve corretamente o problema. De fato, o que é indecidível em "M_i aceita x_i ?" é exatamente qual das duas respostas corresponde a um valor de i arbitrário, ou seja, qual das duas máquinas consideradas deve ser usada em cada caso.

Redução. Uma das técnicas mais comuns de estabelecer se um problema é decidível ou indecidível é através de redução desse problema a outro problema cuja resposta é conhecida.

> Dizemos que P_1 *se reduz a* P_2 se a resposta a qualquer consulta a P_1 pode ser deduzida da resposta a uma consulta apropriada a P_2.
>
> Se P_1 se reduz a P_2, podemos dizer que, de certa forma, P_2 é mais geral que P_1.

Supondo que P_2 é indecidível, e que P_2 se reduz a P_1, podemos concluir que P_1 também é indecidível; por outro lado, se P_1 é decidível, e P_2 se reduz a P_1, podemos concluir que P_2 também é decidível.

O problema da parada. Como exemplo da técnica de redução, vamos mostrar que o problema da parada das máquinas de Turing (*halting problem*) é indecidível. Este problema pode ser formulado como "M_i pára com entrada x_j?", embora algumas vezes seja confundido o problema da aceitação "M_i aceita x_j?".

Como é fácil reduzir qualquer um dos problemas ao outro, os dois problemas podem ser considerados equivalentes, e a confusão de certa forma é justificada. Para verificar a equivalência, basta verificar que é sempre possível alterar uma mT M, construindo outra mT M' que aceita a mesma linguagem, de maneira tal que M' tem as propriedades de parar sempre que atinge um estado final, e de nunca parar em um estado que não seja final. Isso quer dizer que M' pára se e somente se aceita sua entrada. Vamos representar o problema da parada por Q(i,j) = "M_i aceita x_j?"

Como sabemos que P(i) = "Mi aceita xi?" é indecidível (veja a linguagem L acima), e sabemos que podemos reduzir P a Q, porque P(i) é equivalente a Q(i, i), concluímos que o problema da parada Q é indecidível.

Usualmente, a demonstração de que um problema é indecidível é, feita através de redução, a partir do problema da parada, diretamente, ou então, de forma indireta, a partir de problemas cuja indecidibilidade já foi provada anteriormente.

(junho 99)