Teoria da Computação **FCT-UNL, 2º semestre 2023-2024**

Notas 11: Turing-computabilidade

Autor: João Ribeiro

Introdução

Nestas notas estudamos a teoria da computabilidade para Máquinas de Turing. Veremos o que é talvez os resultados filosoficamente mais importantes nesta cadeira – a existência (e exemplos concretos simples) de problemas de decisão que não podem ser resolvidos por nenhum algoritmo.

11.1 Linguagens semi-decidíveis e decidíveis

Seja $L \subseteq \Sigma^*$ uma linguagem qualquer. Estamos interessados em perceber quando é que (não) existe uma MT que resolve o problema de decisão associado a L. Temos de ter algum cuidado em relação ao significado de "resolver [o problema de decisão]". No contexto dos AFDs, tínhamos de descrever um AFD que aceita um input $w \in \Sigma^*$ se $w \in L$ e rejeita w se $w \notin L$. Em contraste, como mencionado anteriormente, uma MT pode parar e aceitar, parar e rejeitar, *ou não parar*.

Vamos estudar duas interpretações do que significa resolver um problema de decisão.

Definição 11.1 (Linguagem semi-decidível) *Dizemos que uma linguagem $L \subseteq \Sigma^*$ é* semi-decidível *se existe uma MT M com estados de aceitação e rejeição q_{acc} e q_{rej}, respectivamente, tal que M com input w entra em q_{acc} se e só se $w \in L$.*

Portanto, uma linguagem L é semi-decidível quando existe uma MT M que satisfaz o seguinte:

- Quando recebe $w \in L$ como input, entra em q_{acc};
- Quando recebe $w \notin L$, pode ou entrar em q_{rej} (e portanto pára e rejeita), ou *não parar*.

Por outras palavras, a MT M nunca se engana e devolve sempre "sim" quando $w \in L$, mas não tem de parar e responder quando $w \notin L$.

Definimos também o conceito mais forte de linguagens decidíveis.

Definição 11.2 (Linguagem decidível) *Dizemos que uma linguagem $L \subseteq \Sigma^*$ é* decidível *se existe uma MT M com estados de aceitação e rejeição q_{acc} e q_{rej}, respectivamente, tal que M com input w entra em q_{acc} se $w \in L$ e entra em q_{rej} se $w \notin L$.*

Ao contrário das MTs associadas a linguagens semi-decidíveis, as MTs associadas a linguagens decidíveis têm de parar eventualmente em qualquer input. Parece-nos razoável afirmar que estas MTs são muito mais úteis do que aquelas que apenas semi-decidem uma linguagem. De facto, certos teóricos dizem que MTs que não param em certos inputs não merecem ser chamadas de "algoritmos" [LP97, Section 5.1]. O seguinte teorema é uma consequência directa deste facto.

Teorema 11.1 *Se L é decidível, então L também é semi-decidível.*

Concluímos que o conceito de decidibilidade é potencialmente mais estrito que semi-decidibilidade.

Codificação de objectos. No resto destas notas teremos de nos referir à "descrição", ou "codificação", de vários tipos de objectos. Por exemplo, vamos considerar linguagens que contêm como elementos descrições de MTs que satisfazem uma certa propriedade. No geral, se M for um objecto (tal como, por exemplo, uma MT, um AFD, ou uma GLC), então denotamos por $\langle M \rangle$ a descrição de M num alfabeto standard fixo, tal como o alfabeto binário ou ASCII. Quando nos queremos referir a descrições de vários objectos $M_1, \ldots, M_k$, podemos também escrever $\langle M_1, M_2, \ldots, M_k \rangle$.

11.1.1 Exemplos de linguagens decidíveis e semi-decidíveis

Exemplo 11.1 Seja $L_1 = \{w \# w \mid w \in \{a, b\}^*\}$. Argumentamos que L_1 é decidível descrevendo por alto uma MT M que decide esta linguagem.

Não percebi porque tenho que ver os a e b de cada lado

1. M e verifica se existe um único # no input. Caso contrário, rejeita. M volta a percorrer a fita e verifica se antes do # não existem b's e que depois do # não existem a's. Caso contrário, rejeita. Se o input for apenas #, aceita.
2. M volta ao início da fita e marca o primeiro símbolo do input antes do #. De seguida, move-se para a direita até encontrar o primeiro símbolo depois do #, e verifica que este é igual ao símbolo marcado. Se não forem iguais ou se tal símbolo não existir (i.e., se a cabeça encontrar $\sqcup$), rejeita. Se forem iguais, marca este símbolo também.
3. M volta ao início da fita e repete o Passo 2, ignorando os símbolos já marcados. Quando já não existirem símbolos por marcar à esquerda do #, a MT M move-se para a direita e verifica se falta marcar algum símbolo após o #. Em caso afirmativo, rejeita. Caso contrário, aceita.

Relembramos que L_1 não é regular!

Exemplo 11.2 Seja $L_2 = \{a^n \# b^n \# c^n \mid n \in \mathbb{N}\}$. Argumentamos que L_2 é decidível descrevendo por alto uma MT M que decide L_2. Esta MT comporta-se de forma muito semelhante à MT que decide $\{a^n \# b^n \mid n \in \mathbb{N}\}$ (que discutimos nas notas anteriores), excepto que quando marca um a move-se para a direita e procura o primeiro b por marcar após o #, e de seguida move-se outra vez para a direita à procura do primeiro c por marcar após o #. Se em algum ponto deste procedimento não existir um b e c por marcar, ou se todos os a's já estiverem marcados mas existir um b ou um c por marcar, M rejeita. Antes de começar este processo, a MT verifica se existem exactamente dois

#'s no input (caso contrário rejeita, e se existem b's e c's antes do primeiro #, a's e c's entre os dois #'s, e a's e b's depois do segundo # (casos em que rejeita).

Relembramos que L_2 não é livre de contexto!

Exemplo 11.3 Seja $ACC_{\mathsf{AFD}} = \{\langle M, w\rangle \mid M \text{ é um AFD e aceita } w\}$, onde usamos $\langle M, w\rangle$ para denotar a descrição do AFD M e do input w codificada segundo algum alfabeto standard (como, por exemplo, o alfabeto binário ou ASCII). Argumentamos que ACC_{AFD} é decidível. Para isto basta considerar a MT M' que simula o AFD M no input w a partir da descrição $\langle M\rangle$ e aceita (resp. rejeita) se a computação de M com input w termina num estado de aceitação de M (resp. não termina num estado de aceitação de M).

Este exemplo mostra que MTs conseguem simular qualquer AFD.

Exemplo 11.4 Seja $ACC_{\mathsf{GLC}} = \{\langle G, w\rangle \mid G \text{ é uma GLC e } w \in L(G)\}$. Argumentamos que ACC_{GLC} é semi-decidível. Consideramos a MT M' que vai aplicando várias combinações de regras de substituição à variável inicial de G. Como $w \in L(G)$, existe pelo menos uma derivação que leva a w, e portanto M' vai, eventualmente, parar e aceitar. No entanto, como pode existir um conjunto infinito de derivações, M' poderá não parar quando $w \notin L(G)$.

Usando resultados não discutidos em aula, é possível mostrar que, na realidade, ACC_{GLC} é decidível. Este exemplo mostra que MTs conseguem simular GLCs.

Tendo em conta os exemplos acima, obtemos a relação entre as noções de linguagem regular, linguagem livre de contexto, linguagem decidível, e linguagem semi-decidível ilustrada na Figura 11.1. Sabemos também que o conjunto das linguagens regulares é um subconjunto próprio das linguagens livres de contexto e que o conjunto das linguagens livres de contexto é um subconjunto próprio das linguagens decidíveis.

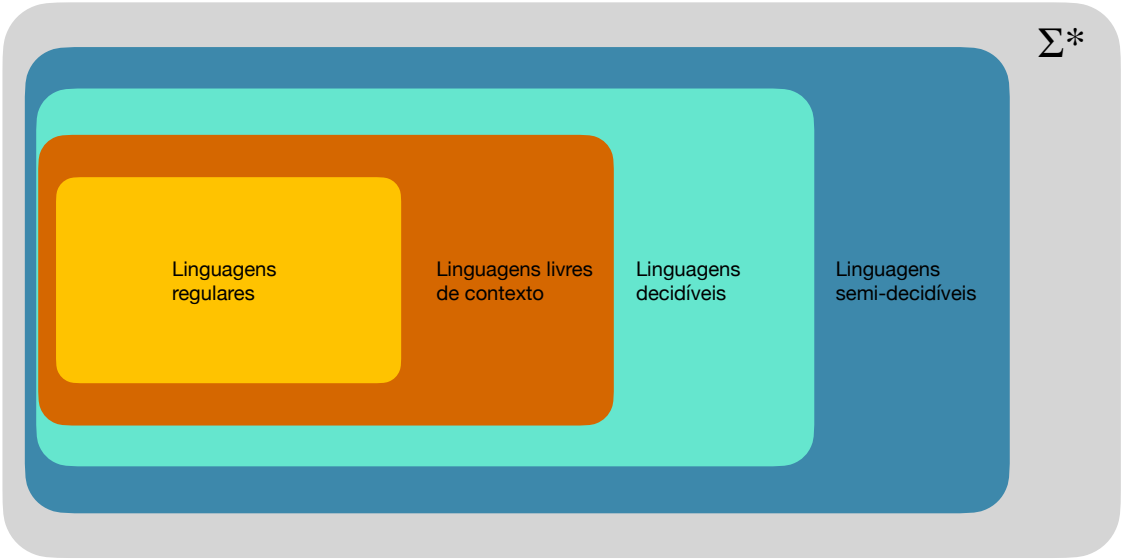

Figure 11.1: Relações entre várias tipos de linguagens estudadas nesta cadeira.

Consideramos mais um exemplo.

Exemplo 11.5 Seja $ACC_{\mathsf{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT e aceita } w\}$. Argumentamos que ACC_{MT} é semi-decidível descrevendo a alto nível uma MT M' que pára e aceita somente quando o input $(M, w) \in L_{acc}$. A MT M' procede da seguinte forma:

1. Simular a MT M no input w. Sejam q_{acc} e q_{rej} os estados de aceitação e rejeição de M, respectivamente.
2. Se M entrar em q_{acc} no input w, então M' entra no seu estado de aceitação e pára. Se M entrar em q_{rej} no input w, então M' entra no seu estado de rejeição e pára.

Se $\langle M, w\rangle \in ACC_{\mathsf{MT}}$, então sabemos que M entra no estado q_{acc} após um número finito de passos com input w. Isto significa que, neste caso, a nossa MT M' vai entrar no seu estado de aceitação e parar após um número finito de passos. Se $\langle M, w\rangle \notin L_{acc}$, então M nunca entra no estado q_{acc} com input w. Segue que M' nunca entra no seu estado de aceitação.

Não é, no entanto, claro se ACC_{MT} é decidível...

11.2 Linguagens não decidíveis e não semi-decidíveis

Já vimos exemplos de linguagens decidíveis e semi-decidíveis. Surgem algumas questões naturais:

1. Será que existem linguagens que não são semi-decidíveis (e, portanto, também não são decidíveis)?
2. Será que existem linguagens que são semi-decidíveis, mas não decidíveis? Por exemplo, não é claro se a linguagem do Exemplo 11.5 é decidível, pois teríamos de desenhar uma MT M' que pára e rejeita mesmo quando a MT M não pára no input w.

Uma resposta puramente existencial à primeira questão é fácil. O conjunto de todas as linguagens é $\mathcal{P}(\Sigma^*)$, que não é contável pelo teorema de Cantor. Por outro lado, o conjunto de todas as MTs é contável (pois podemos codificar cada MT como uma sequência em, por exemplo, $\{0,1\}^*$). Concluímos que existem muitas linguagens que não são semi-decidíveis!

11.2.1 Uma linguagem semi-decidível mas indecidível

Para respondermos à segunda questão, vamos considerar a linguagem ACC_{MT} do Exemplo 11.5, que sabemos ser semi-decidível.

Teorema 11.2 *A linguagem ACC_{MT} não é decidível.*

Demonstração: Vamos aplicar o princípio da diagonalização a MTs. Suponhamos que existe uma MT H que decide ACC_{MT}. Isto quer dizer que para qualquer par $\langle M, w\rangle$ onde M é uma MT e w um input qualquer temos

$$H(\langle M, w\rangle) = \begin{cases} accept, & \text{se } M \text{ aceita } w, \\ reject, & \text{caso contrário.} \end{cases}$$

Vamos agora criar uma nova MT D que, intuitivamente, faz sempre o contrário de H em qualquer input. Mais precisamente, D procede da seguinte maneira num input a descrição de uma MT M:

1. Simula H no input $\langle M, w = \langle M\rangle\rangle$.
2. Se H aceita, então D rejeita. Se H rejeita, então D aceita.

Se dermos $\langle D, w = \langle D\rangle\rangle$ como input a H, obtemos

$$H(\langle D, w = \langle D\rangle\rangle) = \begin{cases} accept, & \text{se } D \text{ aceita input } \langle D\rangle, \\ reject, & \text{caso contrário.} \end{cases}$$

Como D faz sempre o contrário de H, concluímos que

$$D(\langle D\rangle) = \begin{cases} reject, & \text{se } D(\langle D\rangle) = accept, \\ accept, & \text{se } D(\langle D\rangle) = reject, \end{cases}$$

uma contradição. Concluímos que a MT H não pode existir, e portanto ACC_{MT} não é decidível. ■

Onde está a diagonalização? Vamos discutir o uso do princípio da diagonalização na demonstração do Teorema 11.2 com mais cuidado.

Consideramos uma tabela cujas linhas são indexadas por descrições de MTs, $\langle M\rangle$, e cujas colunas são indexadas por possíveis inputs, w, conforme exemplificado abaixo. A entrada $(\langle M\rangle, w)$ desta tabela contém *acc* se M aceita o input w, e $\perp$ caso contrário (se M com input w pára e rejeita ou simplesmente não pára).

	w_1	w_2	w_3	$\ldots$
$\langle M_1\rangle$	*acc*	*acc*	$\perp$	$\ldots$
$\langle M_2\rangle$	$\perp$	$\perp$	*acc*	$\ldots$
$\langle M_3\rangle$	$\perp$	*acc*	$\perp$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Como assumimos (com vista a uma contradição) que existe uma MT H que decide ACC_{MT}, conseguimos calcular qualquer entrada desta tabela. A ideia é, então, construir uma MT D que não aparece nesta enumeração. Mais concretamente, vamos construir D de forma a que ela se comporte de maneira diferente de qualquer outra MT M da enumeração nalgum input w. Isto é claramente absurdo, pois todas as MTs aparecem na enumeração.

A diagonal da tabela acima corresponde às entradas indexadas por $(\langle M\rangle, \langle M\rangle)$. A estratégia para construir D é natural, dado o que já vimos nesta cadeira: A MT D que construímos vai ter um comportamento diferente da MT M no input $\langle M\rangle$. Como podemos garantir isto? Primeiro, a MT D usa a MT H para determinar o comportamento de M no input $\langle M\rangle$ (aceita/não aceita). Depois, faz exactamente o contrário de M: Se M aceita $\langle M\rangle$, então D rejeita esse input. Caso contrário, D aceita.

11.2.2 Exemplo concreto de uma linguagem não semi-decidível

Vimos acima que existem linguagens não semi-decidíveis (mas sem dar exemplos concretos), e também vimos que a linguagem ACC_{MT} é semi-decidível mas não decidível. No entanto, também é natural procurarmos exemplos concretos de linguagens que não são semi-decidíveis. Com isto em mente, introduzimos a seguinte noção: Dizemos que uma linguagem L é *co-semi-decidível* se $\overline{L}$ é semi-decidível.

Teorema 11.3 *Se L é semi-decidível e co-semi-decidível, então L é decidível.*

Demonstração: Sejam M e $\overline{M}$ MTs que semi-decidem L e $\overline{L}$, respectivamente. Criamos uma MT D que decide L da seguinte forma: Dado input w, a MT D simula M e $\overline{M}$ em paralelo no input w. Se a simulação de M aceitar, então D aceita. Se a simulação de $\overline{M}$ aceitar, então D rejeita.

Argumentamos que D decide L correctamente. Se $w \in L$, a simulação de M irá aceitar eventualmente, caso em que D aceita w. Notamos que neste caso a simulação de $\overline{M}$ ou rejeita ou não pára. Se $w \notin L$, a simulação de $\overline{M}$ (que semi-decide $\overline{L}$) irá aceitar eventualmente, caso em que D rejeita w. ■

O Teorema 11.3 permite-nos concluir que se L é uma linguagem semi-decidível mas indecidível, então o complemento $\overline{L}$ não é semi-decidível (pois caso contrário L seria co-semi-decidível, e portanto decidível). Temos, então, o seguinte corolário.

Corolário 11.1 *A linguagem $\overline{ACC_{\mathsf{MT}}}$ não é semi-decidível.*

Demonstração: Pelo Teorema 11.2, a linguagem ACC_{MT} é semi-decidível mas indecidível. Logo, o Teorema 11.3 implica que o seu complemento não é semi-decidível. ■

11.3 Reduções entre problemas computacionais

Quando é que podemos afirmar que um dado problema computacional A é "mais difícil" do que outro problema B? Uma opção (bastante natural) consiste em mostrar que qualquer programa que resolva A pode ser usado para resolver B. A isto chamamos uma *redução* de B para A. Reduções entre problemas computacionais são um dos conceitos mais fundamentais na teoria da computação.

No caso especial em que A e B são problemas de decisão associados a linguagens L_A e L_B, a existência de uma redução de B para A permite-nos concluir que se L_B é indecidível, então L_A também é indecidível.

Nesta secção veremos alguns exemplos de como reduções entre problemas nos permitem estabelecer a indecidibilidade de muitas linguagens, culminando no *teorema de Rice*.

11.3.1 O problema da paragem e outros exemplos

O problema da paragem é um clássico da teoria da computação. Este problema pede para desenharmos um algoritmo que, dada a descrição de um qualquer programa e um qualquer input, decide se o programa pára nesse input. Mais formalmente, o problema da paragem é o problema de decisão associado à linguagem

$$HALT_{\mathsf{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT e pára no input } w\}.$$

Temos o seguinte teorema.

Teorema 11.4 *A linguagem $HALT_{\mathsf{MT}}$ é indecidível.*

Demonstração: Demonstramos este teorema mostrando que qualquer MT D que decide $HALT_{\mathsf{MT}}$ pode ser transformada numa MT D' que decide ACC_{MT}. Como ACC_{MT} é indecidível pelo Teorema 11.2, concluímos que $HALT_{\mathsf{MT}}$ também tem de ser indecidível.

Seja D uma MT que decide $HALT_{\mathsf{MT}}$. Construímos D' que decide ACC_{MT} da seguinte maneira: Dado input $\langle M, w\rangle$, a MT D' simula D no input $\langle M, w\rangle$. Se D rejeitar (o que significa que M não pára com input w), então D' rejeita. Se D aceitar (caso em que sabemos que M pára com input w), então D' simula M no input w e aceita se M aceitar e rejeita se M rejeitar. ■

Consideremos agora o seguinte problema: Dada uma MT M, decidir se M aceita pelo menos um input. Mais formalmente, este é o problema de decisão associado à linguagem

$$E_{\mathsf{MT}} = \{\langle M\rangle \mid \ M \text{ é uma MT e } L(M) = \varnothing\}.$$

Teorema 11.5 *A linguagem E_{MT} é indecidível.*

Demonstração: Vamos mostrar que uma MT D que decide E_{MT} pode ser transformada numa MT D' que decide ACC_{MT}. Como ACC_{MT} não é decidível, o teorema segue.

Relembramos o problema de decisão associado a ACC_{MT} consiste em decidir, dado $\langle M, w\rangle$, se M é uma MT que aceita w. A MT D' procede da seguinte forma dado input $\langle M, w\rangle$:

1. Transforma M numa MT M' que é exactamente igual a M, mas que rejeita todos os inputs que são diferentes de w. Isto pode ser assegurado adicionando, antes da computação de M começar, um procedimento que verifica se o input é igual a w, e que rejeita caso contrário.
2. Simula a MT D no input $\langle M'\rangle$. Se D aceitar, então D' rejeita. Se D rejeitar, então D' aceita.

Justificamos que D' decide ACC_{MT}.

Se $\langle M, w\rangle \in ACC_{\mathsf{MT}}$, então M aceita w. Isto quer dizer que M' também aceita w, e portanto $L(M') \neq \varnothing$. Logo, D rejeita $\langle M'\rangle$, e portanto D' vai aceitar $\langle M, w\rangle$.

Se $\langle M, w\rangle \in ACC_{\mathsf{MT}}$, então M rejeita w. Isto quer dizer que M' também rejeita w, e portanto $L(M') = \varnothing$, pois construímos M de forma a esta rejeitar todos os inputs $w' \neq w$. Logo, D aceita $\langle M'\rangle$, e portanto D' vai rejeitar $\langle M, w\rangle$. ■

Consideramos mais um problema: Dada uma MT M, decidir se a sua linguagem $L(M)$ é regular. Formalmente, este é o problema de decisão associado à linguagem

$$REG_{\mathsf{MT}} = \{\langle M\rangle \mid \ M \text{ é uma MT e } L(M) \text{ é regular}\}.$$

Teorema 11.6 *A linguagem REG_{MT} é indecidível.*

Demonstração: Seja D uma MT que decide REG_{MT}. Construímos a partir de D uma MT D' que decide ACC_{MT}.

A MT D' procede da seguinte forma com input $\langle M, w\rangle$:

1. Constrói uma MT M' a partir de $\langle M, w\rangle$ com as seguintes propriedades: M' aceita todas as sequências da forma $a^n b^n$ para algum $n \in \mathbb{N}$. Mais ainda, se M aceita w, então M' aceita todos os inputs em $(\Sigma \cup \{a, b\})^*$, onde Σ é o alfabeto de M.

   Para implementar M', primeiro esta corre um procedimento que verifica se o seu input u é da forma $a^n b^n$. Se este for o caso, aceita. Caso contrário, M' simula M no input w. Se M parar e entrar no seu estado de aceitação, então M' aceita u. Caso contrário, M' não pára.

2. Simula D no input $\langle M'\rangle$, e dá a mesma resposta que D.

Justificamos que D' decide ACC_{MT}.

Se $\langle M, w\rangle \in ACC_{\mathsf{MT}}$, então $L(M') = (\Sigma \cup \{a, b\})^*$, que é regular. Logo, D aceita $\langle M'\rangle$, e portanto D' aceita $\langle M, w\rangle$.

Se $\langle M, w\rangle \notin ACC_{\mathsf{MT}}$, então $L(M') = \{a^n b^n \mid n \in \mathbb{N}\}$, que não é regular. Logo, D rejeita $\langle M'\rangle$, e portanto D' rejeita $\langle M, w\rangle$. ■

11.3.2 Reduções por mapeamento

Vamos agora formalizar um conceito especial de redução entre linguagens. Começamos por definir funções *computáveis*.

Definição 11.3 (Função computável) *Uma função $f : \Sigma^* \to \Sigma^*$ é* computável *se existe uma MT M tal que dado qualquer input $w \in \Sigma^*$ a computação de M pára e aceita com $f(w)$ escrito na fita e com a cabeça de M a apontar para a posição mais à esquerda da fita.*

Intuitivamente, uma linguagem L_1 reduz-se a uma linguagem L_2 se existe um algoritmo que transforma palavras $w \in L_1$ em palavras $f(w) \in L_2$, e palavras $w \notin L_1$ em palavras $f(w) \notin L_2$.

Definição 11.4 (Redução por mapeamento) *Dadas duas linguagens $L_1, L_2 \subseteq \Sigma^*$, dizemos que L_1* se reduz por mapeamento a *L_2, o que denotamos por $L_1 \leq_m L_2$, se existe uma função computável $f : \Sigma^* \to \Sigma^*$ tal que*

$$w \in L_1 \iff f(w) \in L_2.$$

Uma redução de L_1 para L_2 permite transferir propriedades de L_1 para L_2 e vice-versa, conforme descrito nos seguintes teoremas e corolários.

Teorema 11.7 *Sejam $L_1, L_2 \subseteq \Sigma^*$ duas linguagens. Se L_2 é decidível e $L_1 \leq_m L_2$, então L_1 também é decidível.*

Demonstração: Seja D uma MT que decide L_2. Como $L_1 \leq_m L_2$, existe função computável f tal que $w \in L_1 \iff f(w) \in L_2$. Consideramos a seguinte MT D' que decide L_1. Dado input $w \in \Sigma^*$, a MT D' procede da seguinte forma:

1. Usa a MT que computa f para escrever $f(w)$ na fita;
2. Simula a MT D com input $f(w)$ e aceita se e só se D aceita.

Suponhamos que $w \in L_1$. Então $f(w) \in L_2$, o que significa que D aceita com input $f(w)$, e portanto D' aceita com input w. Caso contrário, se $w \notin L_1$ então $f(w) \notin L_2$. Neste caso, D rejeita com input $f(w)$, e portanto D' rejeita com input w. Concluímos que D' decide L_1. ■

O seguinte corolário do Teorema 11.7 é imediato.

Corolário 11.2 *Sejam $L_1, L_2 \subseteq \Sigma^*$ duas linguagens. Se L_1 é indecidível e $L_1 \leq_m L_2$, então L_2 também é indecidível.*

Usando um raciocínio análogo, obtemos os seguintes resultados.

Teorema 11.8 *Sejam $L_1, L_2 \subseteq \Sigma^*$ duas linguagens. Se L_2 é semi-decidível e $L_1 \leq_m L_2$, então L_1 também é semi-decidível.*

Corolário 11.3 *Sejam $L_1, L_2 \subseteq \Sigma^*$ duas linguagens. Se L_1 não é semi-decidível e $L_1 \leq_m L_2$, então L_2 também não é semi-indecidível.*

11.3.3 Uma linguagem que não é semi-decidível nem co-semi-decidível

Usamos uma redução por mapeamento para mostrar a existência de uma linguagem que não é semi-decidível nem co-semi-decidível. Consideramos o seguinte problema: Dadas duas MTs M_1 e M_2, será que estas aceitam o mesmo conjunto de inputs? Mais formalmente, este é o problema de decisão associado à linguagem

$$EQ_{\mathsf{MT}} = \{\langle M_1, M_2\rangle \mid \ M_1 \text{ e } M_2 \text{ são MTs e } L(M_1) = L(M_2)\}.$$

Teorema 11.9 EQ_{MT} *não é semi-decidível nem co-semi-decidível.*

Demonstração: Começamos por mostrar que $\overline{ACC_{\mathsf{MT}}} \leq_m EQ_{\mathsf{MT}}$. Como $\overline{ACC_{\mathsf{MT}}}$ não é semi-decidível, segue pelo Teorema 11.8 que EQ_{MT} também não é semi-decidível. Construímos uma MT R que computa uma função f tal que $z \in \overline{ACC_{\mathsf{MT}}} \iff f(z) \in EQ_{\mathsf{MT}}$. Tal como em casos anteriores, podemos assumir que $z = \langle M, w\rangle$ para alguma MT M e $w \in \Sigma^*$ (caso contrário forçamos $f(z) = \langle M_1, M_2\rangle$, onde M_1 aceita todos os inputs e M_2 rejeita todos os inputs). A MT R procede da seguinte forma com input $\langle M, w\rangle$:

1. Cria uma MT M_w que se comporta como M, excepto que rejeita todos os inputs diferentes de w;
2. Escreve $\langle M_w, M'\rangle$ na fita, onde M' é uma MT tal que $L(M') = \varnothing$.

Suponhamos que $\langle M, w\rangle \in \overline{ACC_{\mathsf{MT}}}$. Então M com input w ou rejeita ou não pára. Em qualquer dos casos temos $L(M_w) = \varnothing = L(M')$, e portanto $f(\langle M, w\rangle) = \langle M_w, M'\rangle \in EQ_{\mathsf{MT}}$. Se $\langle M, w\rangle \notin \overline{ACC_{\mathsf{MT}}}$, então M aceita o input w. Segue que $L(M_w) = \{w\} \neq L(M')$, e portanto $f(\langle M, w\rangle) = \langle M_w, M'\rangle \notin EQ_{\mathsf{MT}}$. Concluímos que $\overline{ACC_{\mathsf{MT}}} \leq_m EQ_{\mathsf{MT}}$.

De forma semelhante podemos mostrar que $\overline{ACC_{\mathsf{MT}}} \leq_m \overline{EQ_{\mathsf{MT}}}$, o que implica que EQ_{MT} não é co-semi-decidível. Esta redução é deixada como exercício para o leitor. ■

11.3.4 O teorema de Rice

Vimos acima vários exemplos de linguagens indecidíveis. Algumas destas linguagens, como E_{MT} e REG_{MT} capturam o problema de decidir propriedades de programas/MTs. Nesta secção discutimos um resultado importante, devido a Rice [Ric53], que generaliza estes exemplos de indecidibilidade. Resumidamente, o teorema de Rice diz que todas as propriedades semânticas (no sentido em que apenas dependem da linguagem da MT) não-triviais de programas são indecidíveis.

Teorema 11.10 (Teorema de Rice) *Seja* $P = \{\langle M\rangle \mid M \text{ é uma MT tal que } L(M) \in \mathcal{C}\}$, *em que* $\mathcal{C}$ *é uma classe de linguagens não-trivial no sentido em que existe uma MT* M *tal que* $L(M) \in \mathcal{C}$ *e existe uma MT* M' *tal que* $L(M') \notin \mathcal{C}$. *Então* P *não é decidível.*

Demonstração: Demonstramos este teorema através de uma redução por mapeamento. Vamos mostrar que $ACC_{\mathsf{MT}} \leq_m P$. Como ACC_{MT} é indecidível pelo Teorema 11.2, concluímos que P também é indecidível pelo Corolário 11.2.

Seja $T_\varnothing$ uma MT que rejeita todos os inputs. Sem perda de generalidade, podemos assumir que $\langle T_\varnothing\rangle \notin P$ (caso contrário analisamos $\overline{P}$ em vez de P), ou seja, $\varnothing \notin \mathcal{C}$. Como $\mathcal{C}$ não é trivial, também sabemos que existe uma MT T tal que $L(T) \in \mathcal{C}$, logo $\langle T\rangle \in P$. Intuitivamente, a redução usa o facto que P consegue distinguir entre $\langle T_\varnothing\rangle$ e $\langle T\rangle$, mas não consegue distinguir duas MTs com a mesma linguagem.

A MT R que computa a redução f de ACC_{MT} para P procede da seguinte forma com input z:

1. Verifica se $z = \langle M, w\rangle$ para alguma MT M e $w \in \Sigma^*$. Se este não for o caso, R escreve $\langle T_\varnothing\rangle$ na fita e pára (ou seja, $f(z) = \langle T_\varnothing\rangle$.
2. Usa $\langle M, w\rangle$ para construir a MT M_w que se comporta da seguinte forma com input u:
   (a) Simula M com input w. Se M pára e rejeita, então M_w pára e rejeita. Se M pára e aceita, avançamos para o segundo passo.
   (b) Simula T com input u. Se T pára e aceita, então M_w aceita.

   No final R escreve $\langle M_w\rangle$ na fita, e portanto $f(z) = \langle M_w\rangle$.

Resta argumentar que $z \in ACC_{\mathsf{MT}} \iff f(z) \in P$. Primeiro, se z não for da forma $\langle M, w\rangle$ para alguma MT M e $w \in \Sigma^*$, então $z \notin ACC_{\mathsf{MT}}$. Neste caso temos $f(z) = \langle T_\varnothing\rangle \notin P$. Podemos agora assumir que $z = \langle M, w\rangle$ para alguma MT M e $w \in \Sigma^*$, caso em que $f(z) = \langle M_w\rangle$.

Se $\langle M, w\rangle \in ACC_{\mathsf{MT}}$, então a MT M_w satisfaz $L(M_w) = L(T)$. Como $\langle T\rangle \in P$, temos que $L(M_w) = L(T) \in \mathcal{C}$, logo $\langle M_w\rangle \in P$. Se $\langle M, w\rangle \notin ACC_{\mathsf{MT}}$, então a MT M_w satisfaz $L(M_w) = \varnothing$. Como $L(T_\varnothing) = \varnothing \notin \mathcal{C}$, temos que $L(M_w) \notin \mathcal{C}$, logo $\langle M_w\rangle \notin P$. ■

Exemplo 11.6 Vamos re-derivar a indecidibilidade de E_{MT} usando o teorema de Rice.

Seja $\mathcal{C} = \{\varnothing\} \subseteq \mathcal{P}(\Sigma^*)$. Então, podemos escrever

$$E_{\mathsf{MT}} = \{\langle M\rangle \mid \ M \text{ é uma MT e } L(M) \in \mathcal{C}\}.$$

Resta argumentar que $\mathcal{C}$ não é trivial. Observamos que a MT M com apenas um estado inicial sem transições definidas rejeita todos os inputs, e portanto $L(M) = \varnothing \in \mathcal{C}$. Por outro lado, a MT M' com transições $\sqcup, \Sigma \to q_{acc}, R$ a partir do estado inicial aceita todos os inputs, e portanto $L(M') = \Sigma^* \notin \mathcal{C}$.

Exemplo 11.7 Discutimos mais um exemplo de aplicação do teorema de Rice. Seja

$$L = \{\langle M\rangle \mid \ M \text{ é uma MT e } 0011 \in L(M)\}.$$

Vamos mostrar que L é indecidível.

Seja $\mathcal{C} = \{L \in \mathcal{P}(\{0,1\}^*) \mid 0011 \in L\}$. Podemos escrever

$$L = \{\langle M\rangle \mid \ M \text{ é uma MT e } L(M) \in \mathcal{C}\}.$$

Resta verificar que L não é trivial. Basta observar que existem MTs M e M' tal que M aceita 0011 e M' rejeita 0011 (por exemplo, M pode ser a MT que aceita todos os inputs, e M' a MT que rejeita todos os inputs).

11.4 Para explorar

Recomendamos a leitura de [Sip13, Chapters 4 and 5]. A noção de reduções por mapeamento (também conhecidas por "many-one reductions") foi introduzida por Post [Pos44]. No contexto da teoria da complexidade, em que limitamos os recursos usados por MTs, o conceito de reduções por mapeamento leva a reduções à Karp.

References

[LP97] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, USA, 2nd edition, 1997.

[Pos44] Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50(5):284 – 316, 1944.

[Ric53] Henry G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[Sip13] Michael Sipser. *Introduction to the Theory of Computation*. CEngage Learning, 3rd edition, 2013.