Capítulo 5

Redutibilidade

No Capítulo 4 estabelecemos a máquina de Turing como nosso modelo de um computador de propósito geral. Apresentamos vários exemplos de problemas que são solúveis sobre uma máquina de Turing e demos um exemplo de um problema, A_{MT}, que é computacionalmente insolúvel. Neste capítulo examinamos vários problemas insolúveis adicionais. Ao fazer isso introduzimos o método principal para provar que problemas são computacionalmente insolúveis. Ele é chamado ***redutibilidade***.

Uma ***redução*** é uma maneira de converter um problema em um outro problema de tal maneira que uma solução para o segundo problema pode ser usada para resolver o primeiro problema. Tais redutibilidades surgem frequentemente no cotidiano, mesmo se não nos refiramos a eles usualmente dessa maneira. Por exemplo, suponha que você queira saber andar em uma nova cidade. Você sabe que isso seria fácil se você tivesse um mapa. Por conseguinte você pode reduzir o problema de saber andar na cidade ao problema de obter um mapa da cidade.

Redutibilidade sempre envolve dois problemas, que chamamos A e B. Se A se reduz para B, podemos usar uma solução para B para resolver A. Portanto em nosso exemplo, A é o problema de saber andar na cidade e B é o problema de obter um mapa. Note que redutibilidade não diz nada sobre resolver A ou B sozinhos, mas somente a solubilidade de A na presença de uma solução para B.

A seguir estão mais exemplos de redutibilidades. O problema de viajar de Boston a Paris se reduz ao problema de comprar uma passagem aérea entre as duas cidades. Esse problema por sua vez se reduz ao problema de ganhar dinheiro para a passagem. E esse problema se reduz ao problema de encontrar um emprego. Redutibilidade também ocorre em problemas matemáticos. Por exemplo, o problema de medir a área de um retângulo se reduz ao problema de medir sua altura e sua largura. O problema de resolver um sistema de equações lineares se reduz ao problema de inverter uma matriz.

Redutibilidade desempenha um papel importante na classificação de problemas por decidibilidade e mais tarde em teoria da complexidade também. Quando A é redutível a B, resolver A não pode ser mais difícil que resolver B porque uma solução para B dá uma solução para A. Em termos de teoria da computabilidade, se A é redutível a B e B é decidível, A também é decidível. Equivalentemente, se A é indecidível e redutível a B, B é indecidível. Essa última versão é a chave para se provar que vários problemas são indecidíveis.

Em resumo, nosso método para provar que um problema é indecidível será: Mostre que algum problema já conhecido como indecidível se reduz a ele.

5.1 Problemas indecidíveis da teoria de linguagens....

Já estabelecemos a indecidibilidade de A_{MT}, o problema de se determinar se uma máquina de Turing aceita uma dada entrada. Vamos considerar um problema relacionado, $PARA_{\mathsf{MT}}$, o problema de se determinar se uma máquina de Turing pára (aceitando ou rejeitando) sobre uma dada entrada.[1] Usamos a indecidibilidade de A_{MT} para provar a indecidibilidade de $PARA_{\mathsf{MT}}$ reduzindo A_{MT} a $PARA_{\mathsf{MT}}$. Seja

$$PARA_{\mathsf{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT e } M \text{ pára sobre a entrada } w\}.$$

Teorema 5.1 .....
$PARA_{\mathsf{MT}}$ é indecidível.

.....

Idéia da Prova. Esta prova é por contradição. Assumimos que $PARA_{\mathsf{MT}}$ é decidível e usamos a suposição para mostrar que A_{MT} é decidível, contradizendo o Teorema 4.9. A idéia chave é mostrar que A_{MT} é redutível a $PARA_{\mathsf{MT}}$.

Vamos assumir que temos uma MT R que decide $PARA_{\mathsf{MT}}$. Então usamos R para construir S, uma MT que decide A_{MT}. Para ter uma idéia de como construir S, faça de conta que você é S. Sua tarefa é decidir A_{MT}. Você recebe uma entrada da forma $\langle M, w\rangle$. Você tem que dar como saída *aceita* se M aceita w, e você tem que dar como saída *rejeita* se M entra em loop ou rejeita sobre w. Tente simular M sobre w. Se ela aceita ou rejeita, faça o mesmo. Mas você pode não ser capaz de determinar se M está em loop, e nesse caso sua simulação não terminará. Isso é ruim, pois você é um decisor e portanto nunca lhe é permitido entrar em loop. Portanto essa idéia, em si, não funciona.

Ao invés disso, use a suposição de que você tem uma MT R que decide $PARA_{\mathsf{MT}}$. Com R, você pode testar se M pára sobre w. Se R indica que M não pára sobre w, rejeite porque $\langle M, w\rangle$ não está em A_{MT}. Entretanto, se R indica que M de fato pára sobre w, você pode fazer a simulação sem qualquer perigo de entrar em loop.

Por conseguinte, se a MT R existe, você pode decidir A_{MT}, mas sabemos que A_{MT} é indecidível. Em virtude dessa contradição podemos concluir que R não pode existir. Portanto $PARA_{\mathsf{MT}}$ é indecidível.

Prova. Vamos assumir com o propósito de obter uma contradição que a MT R decide $PARA_{\mathsf{MT}}$. Construimos a MT S para decidir A_{MT}, com S operando da seguinte forma.

$S =$ "Sobre a entrada $\langle M, w\rangle$, uma codificação de uma MT M e uma entrada w:

1. Rode a MT R sobre a entrada $\langle M, w\rangle$.
2. Se R rejeita, *rejeite*.
3. Se R aceita, simule M sobre w até que ela páre.
4. Se M aceitou, *aceite*; se M rejeitou, *rejeite*."

[1] Na Seção 4.2, usamos o termo *problema da parada* para a linguagem A_{MT} muito embora $PARA_{\mathsf{MT}}$ seja o real problema da parada. Daqui por diante distinguimos os dois chamando A_{MT} de o *problema da aceitação*.

Claramente, se R decide $PARA_{\mathsf{MT}}$, então S decide A_{MT}. Como A_{MT} é indecidível, $PARA_{\mathsf{MT}}$ também tem que ser indecidível.

.....

O Teorema 5.1 ilustra nossa estratégia para provar que um problema é indecidível. Essa estratégia é comum à maioria das provas de indecidibilidade, exceto para a indecidibilidade de A_{MT} propriamente dita, que é provada diretamente via o método da diagonalização.

Agora apresentamos vários outros teoremas e suas provas como mais exemplos do método da redutibilidade para provar a indecidibilidade. Seja

$$V_{\mathsf{MT}} = \{\langle M, \rangle \mid M \text{ é uma } \mathsf{MT} \text{ e } L(M) = \emptyset\}.$$

Teorema 5.2 .....
V_{MT} é indecidível.

.....

Idéia da prova. Seguimos o padrão adotado no Teorema 5.1. Assumimos para os propósitos de obter uma contradição que V_{MT} é decidível e então mostramos que A_{MT} é decidível—uma contradição. Seja R uma MT que decide V_{MT}. Usamos R para construir uma MT S que decide A_{MT}. Como funcionará S quando ela recebe a entrada $\langle M, w\rangle$?

Uma idéia é para S rodar R sobre a entrada $\langle M\rangle$ e ver se ela aceita. Se sim, sabemos que $L(M)$ é vazia e portanto que M não aceita w. Mas, se R rejeita $\langle M\rangle$, tudo o que sabemos é que $L(M)$ não é vazia e por conseguinte que M aceita alguma cadeia, mas ainda não sabemos se M aceita a cadeia específica w. Portanto precisamos usar uma idéia diferente.

Ao invés de rodar R sobre $\langle M\rangle$ rodamos R sobre uma modificação de $\langle M\rangle$. Modificamos $\langle M\rangle$ para garantir que M rejeita todas as cadeias exceto w, mas sobre a entrada w ela funciona como de costume. Então usamos R para testar se a máquina modificada reconhece a linguagem vazia. A única cadeia que a máquina pode agora aceitar é w, portanto sua linguagem será não-vazia se e somente se ela aceita w. Se R aceita quando ela é alimentada com uma descrição da máquina modificada, sabemos que a máquina modificada não aceita nada e que M não aceita w.

Prova. Vamos escrever a máquina modificada descrita na idéia da prova usando nossa notação padrão. Chamamo-la M_1.

M_1 = "Sobre a entrada x:

1. Se $x \neq w$, *rejeite*.
2. Se $x = w$, rode M sobre a entrada w e *aceite* se M aceita."

Essa máquina tem a cadeia w como parte de sua descrição. Ela conduz o teste de se $x = w$ da maneira óbvia, fazendo uma varredura na entrada e comparando-a caracter a caracter com w para determinar se eles são o mesmo.

Colocando tudo isso junto, assumimos que a MT R decide V_{MT} e construimos a MT S que decide A_{MT} da seguinte maneira.

S = "Sobre a entrada $\langle M, w\rangle$, uma codificação de uma MT M e uma cadeia w:

1. Use a descrição de M e w e construa a MT M_1 descrita acima.

2. Rode R sobre a entrada $\langle M_1 \rangle$.

3. Se R aceita, *rejeite*; se R rejeita, *aceite*."

Note que S tem na realidade que ser capaz de computar uma descrição de M_1 a partir de uma descrição de M e w. Ela é capaz de fazer isso porque ela precisa apenas adicionar estados extra a M que realizem o teste $x = w$.

Se R fosse um decisor para V_{MT}, S seria um decisor para A_{MT}. Um decisor para A_{MT} não pode existir, portanto sabemos que V_{MT} tem que ser indecidível.

Um outro problema computacional interessante acerca de máquinas de Turing concerne testar se uma dada máquina de Turing reconhece uma linguagem que também pode ser reconhecida por um modelo computacional mais simples. Por exemplo suponhamos que $REGULAR_{\mathsf{MT}}$ seja o problema de se testar se uma dada máquina de Turing tem um autômato finito que lhe é equivalente. Esse problema é o mesmo que testar se a máquina de Turing reconhece uma linguagem regular. Seja

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

Teorema 5.3
$REGULAR_{\mathsf{MT}}$ é indecidível.

Idéia da prova. Como de costume para teoremas de indecidibilidade, esta prova é por redução a partir de A_{MT}. Assumimos que $REGULAR_{\mathsf{MT}}$ é decidível por uma MT R e usamos essa suposição para construir uma MT S que decide A_{MT}. Menos óbvio agora é como usar a capacidade de R ajudar S nessa tarefa. Não obstante, podemos fazê-lo.

A idéia é que S tome sua entrada $\langle M, w \rangle$ e modifique M de tal modo que a MT resultante reconheça uma linguagem regular se e somente se M aceita w. Chamamos a máquina modificada M_2. Projetamos M_2 para reconhecer a linguagem não-regular $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$ se M não aceita w e a linguagem regular Σ^* se M aceita w. Temos que especificar como S pode construir tal M_2 a partir de M e w. Aqui, M_2 funciona automaticamente aceitando todas as cadeias em $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$. Adicionalmente, se M aceita w, M_2 aceita todas as outras cadeias.

Prova. Supomos que R seja uma MT que decide $REGULAR_{\mathsf{MT}}$ e construimos uma MT S para decidir A_{MT}. Então S funciona da seguinte maneira.

S = "Sobre a entrada $\langle M, w \rangle$ onde M é uma MT e w é uma cadeia:

1. Construaa seguinte MT M_2.

   M_2 = "Sobre a entrada x:

   1. Se x tem a forma $\mathtt{0}^n\mathtt{1}^n$, *aceite*.
   2. Se x não tem essa forma, rode M sobre a entrada w e *aceite* se M aceita w."

2. Rode R sobre a entrada $\langle M_2 \rangle$.

3. Se R aceita, *aceite*; se R rejeita, *rejeite*.

Similarmente, os problemas de se testar se a linguagem de uma máquina de Turing é uma linguagem livre-do-contexto, uma linguagem decidível, ou mesmo uma linguagem finita, podem ser mostrados serem indecidíveis com provas semelhantes. Na verdade, um resultado geral, chamado Teorema de Rice, enuncia que testar *qualquer propriedade* das linguagens reconhecidas por máquinas de Turing é indecidível. Damos o Teorema de Rice no Problema 5.22.

Até agora, nossa estratégia para provar que linguagens são indecidíveis envolve uma redução a partir de A_{MT}. Às vezes reduzir a partir de alguma outra linguagem indecidível, tal como V_{MT}, é mais conveniente quando estamos mostrando que certas linguagens são indecidíveis. O teorema a seguir mostra que testar a equivalência de duas máquinas de Turing é um problema indecidível. Poderíamos prová-lo por uma redução a partir de A_{MT}, mas usamos essa oportunidade para dar um exemplo de uma prova de indecidibilidade por redução a partir de V_{MT}. Seja

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

Teorema 5.4
EQ_{MT} é indecidível.

Idéia da prova. Mostre que, se EQ_{MT} fosse decidível, V_{MT} também seria decidível, dando uma redução de V_{MT} para EQ_{MT}. A idéia é simples. V_{MT} é o problema de se testar se a linguagem de uma MT é vazia. EQ_{MT} é o problema de se testar se as linguagens de duas MT's são a mesma. Se uma dessas linguagens acontece de ser $\emptyset$, acabamos com o problema de se testar se a linguagem da outra máquina é vazia, ou seja, o problema V_{MT}. Portanto, em um certo sentido, o problema V_{MT} é um caso especial do problema EQ_{MT} no qual uma das máquinas está fixada para reconhecer a linguagem vazia. Essa idéia faz com que dar a redução fique fácil.

Prova. Supomos que a MT R decida EQ_{MT} e construimos a MT S para decidir V_{MT} da seguinte maneira.

$S =$ "Sobre a entrada $\langle M\rangle$, onde M é uma MT:

1. Roder R sobre a entrada $\langle M, M_1\rangle$, onde M_1 é uma MT que rejeita todas as entradas.

2. Se R aceita, *aceite*; se R rejeita, *rejeite*."

Se R decide EQ_{MT}, S decide V_{MT}. Mas V_{MT} é indecidível pelo Teorema 5.2, portanto EQ_{MT} também tem que ser indecidível.

Reduções via histórias de computação

O método da história de computação é uma técnica importante para provar que A_{MT} é redutível a certas linguagens. Esse método é frequentemente útil quando o problema a ser mostrado indecidível envolve testar pela existência de algo. Por exemplo, esse método é usado para mostrar a indecidibilidade do décimo problema de Hilbert, testar pela existência de raízes inteiras em um polinômio.

A história de computação para uma máquina de Turing sobre uma entrada é simplesmente a seqüência de configurações pelas quais que a máquina passa quando ela processa a entrada. É um registro completo da computação dessa máquina.

Definição 5.5

Seja M uma máquina de Turing e w uma cadeia de entrada. Uma ***história de computação de aceitação*** para M sobre w é uma seqüência de configurações $C_1, C_2, \ldots, C_l$, onde C_1 é a configuração inicial de M sobre w, C_l é uma configuração de aceitação de M, e cada C_i legitimamente segue de C_{i-1} conforme as regras de M. Uma ***história de computação de rejeição*** para M sobre w é definida similarmente, exceto que C_l é uma configuração de rejeição.

Histórias de computação são seqüências finitas. Se M não pára sobre w, nenhuma história de computação de aceitação ou de rejeição existe para M sobre w. Máquinas determinísticas têm no máximo uma história de computação sobre qualquer entrada. Máquinas não-determinísticas podem ter muitas histórias de computação sobre uma única entrada, correspondendo aos vários ramos da computação. Por enquanto, continuamos a nos concentrar em máquinas determinísticas. Nossa primeira prova de indecidibilidade usando o método da história de computação concerne um tipo de máquina chamada autômato linearmente limitado.

Definição 5.6

Um ***autômato linearmente limitado*** é um tipo restrito de máquina de Turing no qual a cabeça da fita não pode mover para fora da parte da fita contendo a entrada. Se a ḿaquina tentar mover sua cabeça para fora de qualquer uma das extremidades da entrada, a cabeça permanece onde está, da mesma maneira que a cabeça não moverá para fora da extremidade esquerda da fita de uma máquina de Turing comum.

Um autômato linearmente limitado é uma máquina de Turing com uma quantidade limitada de memória, como mostrado esquematicamente na Figura 5.1. Ele pode resolver somente problemas que requerem memória que pode caber dentro da fita usada para a entrada. Usando um alfabeto de fita maior que o alfabeto de entrada permite a memória disponível a ser aumentada de até um fator constante. Daí dizemos que para uma entrada de comprimento n, a quantidade de memória é linear em n—daí o nome desse modelo.

Figura 5.1: Esquema de um autômato linearmente limitado

A despeito de sua restrição de memória, autômatos linearmente limitados são bastante poderosos. Por exemplo, os decisores para A_{AFD}, A_{GLC}, V_{AFD}, e V_{GLC} todos são ALL's. Toda LLC pode ser decidida por um ALL. Na verdade, chegar a uma linguagem decidível que não pode ser decidida por um ALL dá algum trabalho. Desenvolvemos as técnicas para fazer isso no Capítulo 9.

Aqui, A_{ALL} é o problema de se testar se um ALL aceita sua entrada. Muito embora A_{ALL} seja o mesmo que o problema indecidível A_{MT} onde a máquina de Turing é restrita a ser um ALLm podemos mostrar que A_{ALL} é decidível. Seja

$$A_{\mathsf{ALL}} = \{\langle M, w\rangle \mid M \text{ é um ALL que aceita a cadeia } w\}.$$

Antes de provar a decidibilidade de A_{ALL}, achamos que o lema a seguir é útil. Ele diz que um ALL pode ter somente um número limitado de configurações quando uma cadeia de comprimento n é a entrada.

Lema 5.7
Seja M um ALL com q estados e g símbolos no alfabeto de símbolo. Existem exatamente qng^n configurações distintas de M para uma fita de comprimento n.

Prova. Lembremo-nos de que uma configuração de M é como um retrato instantâneo no meio de sua computação. Uma configuração consiste do estado do controle, posição da cabeça, e conteúdo da fita. Aqui, M tem q estados. O comprimento de sua fita é n, portanto a cabeça pode estar em uma das n posições, e g^n possíveis cadeias de símbolos de fita aparecem na fita. O produto dessas três quantidades é o número total de diferentes configurações de M com uma fita de comprimento n.

Teorema 5.8
A_{ALL} é decidível.

Idéia da prova. De modo a decidir se o ALL M aceita a entrada w, simulamos M sobre w. No decorrer da simulação, se M pára e aceita ou rejeita, aceitamos ou rejeitamos conformemente. A dificuldade ocorre se M entra em loop sobre w. Precisamos de ser capazes de detectar a entrada em loop de modo que possamos parar e rejeitar.

A idéia para detectar quando M está em loop é que, à medida M que computa sobre w, ela continua a repetir essa configuração muitas e muitas vezes e por conseguinte está em loop. Devido ao fato de M é um ALL, a quantidade de fita que lhe é disponível é limitada. Pelo Lema 5,7, M pode estar em somente um número limitado de configurações sobre essa quantidade de fita. Portanto somente uma quantidade limitada de tempo está disponível a M antes que ela entre em alguma configuração em que ela tenha entrado anteriormente. Detectar que M está em loop é possível simulando M para o número de passos dado pelo Lema 5.7. se M não parou até aí, ela tem que estar em loop.

Prova. O algoritmo que decide A_{ALL} é o seguinte.

$L =$ "Sobre a entrada $\langle M, w\rangle$, onde M é um ALL e w é uma cadeia:

1. Simule M sobre w por qng^n passos ou até que ela páre.
2. Se M parou, *aceite* se ela aceitou e *rejeite* se ela rejeitou. Se ela não parou, *rejeite*."

Se M sobre w não parou dentro de qng^n passos, ela tem que estar repetindo uma configuração de acordo com o Lema 5.7 e portanto em loop. É por isso que nosso algoritmo rejeita nessa instância.

O Teorema 5.8 mostra que ALL's e MT's diferem de uma maneira essencial: Para ALL's o problema da aceitação é decidível, mas para MT's não o é. Entretanto, alguns outros problemas envolvendo ALL's permanecem indecidíveis. Uma é o problema da vacuidade $V_{\mathsf{ALL}} = \{\langle M\rangle \mid M \text{ é um ALL onde } L(M) = \emptyset\}$. Para provar que V_{ALL} é indecidível, damos uma redução que usa o método da história de computação.

Teorema 5.9 .....
V_{ALL} é indecidível.

.....

Idéia da prova. Esta prova é por redução a partir de A_{MT}. Mostramos que, se V_{ALL} fosse decidível, A_{MT} também seria. Suponha que V_{ALL} seja decidível. Como podemos usar essa suposição para decidir A_{MT}?

Para uma MT M e uma entrada w podemos determinar se M aceita w construindo um certo ALL B e aí testar se $L(B)$ é vazia. A linguagem que B reconhece compreende todas as histórias de computação para M sobre w. Se M aceita w, essa linguagem contém uma cadeia e portanto é não-vazia. Se M não aceita w, essa linguagem é vazia. Se podemos determinar se a linguagem de B é vazia, claramente podemos determinar se M aceita w.

Agora descrevemos como construir B partir de M e w. Note que precisamos mostrar mais que a mera existência de B. Temos que mostrar como uma máquina de Turing pode obter uma descrição de B dadas as descrições de M e w.

Construimos B para aceitar sua entrada x se x é uma história de computação de aceitação para M sobre w. Relembremo-nos que uma história de computação de aceitação é a seqüência de configurações, $C_1, C_2, \ldots, C_j$ pela qual M passa quando ela aceita alguma cadeia w. Para os propósitos desta prova assumimos que a história de computação de aceitação é apresentada como uma única cadeia, com as configurações separadas uma da outra pelo símbolo #, como mostrado na Figura 5.2.

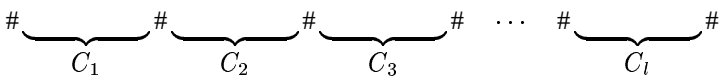

Figura 5.2: Uma possível entrada para B

O ALL B funciona da seguinte maneira. Quando ele recebe uma entrada x, B supostamente aceita x se x é uma computação de aceitação para M sobre w. Primeiro, B quebra x conforme os delimitadores em cadeias $C_1, C_2, \ldots, C_l$. Então B verifica se a C_i satisfaz as três condições de uma história de computação:

1. C_1 é uma configuração inicial para M sobre w.
2. Cada C_{i+1} legimitamente segue de C_i.
3. C_l é uma configuração de aceitação para M.

A configuração inicial C_1 para M sobre w é a cadeia $q_0 w_1 w_2 \cdots w_n$, onde q_0 é o estado inicial para M sobre w. Aqui, B tem essa cadeia diretamente embutida, de modo que ela seja capaz de verificar a primeira condição. Uma configuração de aceitação é uma que contém o estado q_{aceita}, portanto B pode verificar a terceira condição fazendo uma varredura em C_l por q_{aceita}. A segunda condição é a mais difícil de verificar. Para cada par de configurações adjacentes, B verifica se C_{i+1} legitimamente segue de C_i. Esse passo envolve verificar que C_i e C_{i+1} são idênticas exceto pelas posições sob e adjacente à cabeça en C_i. Essas posições têm que ser atualizadas conforme a função

de transição de M. Então, B verifica que a atualização foi feita corretamente fazendo um zigue-zague entre posições correspondentes de C_i e C_{i+1}. Para manter o registro das posições correntes enquanto faz o zigue-zague, B marca a posição corrente com pontos sobre a fita. Finalmente, se as condições 1, 2, e 3 são satisfeitas, B aceita sua entrada.

Note que o ALL B *não* é construído para os propósitos de verdadeiramente rodá-lo sobre alguma entrada—uma confusão comum. Construimos B somente para o propósito de alimentar uma descrição de B num decisor para V_{ALL} que assumimos existir. Uma vez que esse decisor retorna sua resposta podemos invertê-la para obter a resposta se M aceita w. Por conseguinte podemos decidie A_{MT}, uma contradição.

Prova. Agora estamos prontos para enunciar a redução de A_{MT} para V_{ALL}. Suponha que a MT R decida V_{ALL}. Construa a MT S que decide A_{MT} da seguinte forma.

$S =$ "Sobre a entrada $\langle M, w\rangle$, onde M é uma MT e w é uma cadeia:

1. Construa o ALL B a partir de M e w conforme descrito na idéia da prova.
2. Rode R sobre a entrada $\langle B\rangle$.
3. Se R rejeita, *aceite*; se R aceita, *rejeite*."

Figura 5.3: O ALL B verificando uma história de computação de uma MT

Se R aceita $\langle B\rangle$, então $L(B) = \emptyset$. Por conseguinte M não tem história de computação de aceitação sobre w e M não aceita w. Consequentemente S rejeita $\langle M, w\rangle$. Similarmente se R rejeita $\langle B\rangle$, a linguagem de B é não-vazia. A única cadeia que B pode aceitar é uma história de computação de aceitação para M sobre w. Portanto M tem que aceitar w. Consequentemente, S aceita $\langle M, w\rangle$. A Figura 5.3 mostra tal verificação de uma história de computação de uma MT.

.....

Podemos também usar a técnica de redução via histórias de computação para estabelecer a indecidibilidade de certos problemas relacionados a gramáticas livres-do-contexto e autômatos a pilha. Relembremo-nos de que no Teorema 4.7 apresentamos um algoritmo para decidir se uma gramática livre-do-contexto gera alguma cadeia, ou seja, se $L(G) \neq \emptyset$. Agora mostramos que um problema relacionado é indecidível. É o problema de se testar se uma gramática livre-do-contexto gera todas as cadeias possíveis. Provar que esse problema é indecidível é o passo principal na demostração de que o problema da equivalência para gramáticas livres-do-contexto é indecidível. Seja

$$TODAS_{\mathsf{GLC}} = \{\langle G\rangle \mid G \text{ é uma GLC e } L(G) = \Sigma^*\}.$$

Teorema 5.10 .....
$TODAS_{\mathsf{GLC}}$ é indecidível.

Prova. Esta prova é por contradição. Para obter a contradição assumimos que $TODAS_{\mathsf{GLC}}$ é decidível e usamos essa suposição para mostrar que A_{MT} é decidível. Esta prova é semelhante àquela do Teorema 5.9 mas com um pequeno truque extra: Ela é uma redução de A_{MT} via histórias de computação, mas temos que modificar a representação das histórias de computação levemente por uma razão técnica que explicamos mais adiante.

Agora descrevemos como usar um procedimento de decisão para $TODAS_{\mathsf{GLC}}$ para decidir A_{MT}. Para uma MT M e uma entrada w construimos uma GLC G que gera todas as cadeias se e somente se M não aceita w. Portanto, se M não aceita w, G *não* gera alguma cadeia específica. Essa cadeia é—adivinhe—a história de computação de aceitação para M sobre w. Ou seja, G é projetada para gerar todas as cadeias que *não* são histórias de computação de aceitação para M sobre w.

Para fazer a GLC G gerar todas as cadeias que falhem em ser uma história de computação de aceitação para M sobre w, utilizamos a seguinte estratégia. Uma cadeia pode falhar em ser uma história de computação de aceitação por várias razões. Uma história de computação de aceitação para M sobre w aparece como $\#C_1\#C_2\#\ldots\#C_l\#$, onde C_i é a história de computação para M no i-ésimo passo da computação sobre w. Então G gera todas as cadeias que

1. *não* começam com C_1,
2. *não* terminam com uma configuração de aceitação, ou
3. onde algum C_i *não* leva corretamente a C_{i+1} sob as regras de M.

Se M não aceita w, nehuma história de computação de aceitação existe, portanto *todas* as cadeias falham de uma maneira ou de outra. Por conseguinte G geraria todas as cadeias, como desejado.

Agora vamos à verdadeira construção de G. Ao invés de construir G, construimos um AP D. Sabemos que podemos usar a construção dada no Teorema 2.12 na página 106 para converter D para uma GLC. Fazemos isso porque, para nossos propósitos, projetar um AP é mais fácil que projetar uma GLC. Nesta instância, D começará ramificando não-deterministicamente para adivinhar qual das três condições anteriores verificar. Um ramo verifica se o começo da cadeia de entrada é C_1 e aceita se não o é. Um outro ramo verifica se a cadeia de entrada termina com uma configuração contendo o estado de aceitação, q_{aceita}, aceita se não o é.

O terceiro ramo deverá aceitar se alguma C_i não leva corretamente a C_{i+1}. Ele funciona fazendo uma varredura na entrada até que ele não-deterministicamente decide que ela veio para C_i. A seguir, ele empilha C_i até que ele chega no final marcado pelo símbolo #. Então, D desempilha para comparar com C_{i+1}. Elas deveriam casar exceto em torno da posição da cabeça onde a diferença é ditada pela função de transição de M . Finalmente, D aceita se é descasamento ou uma atualização incorreta.

O problema com essa idéia é que, quando D desempilha C_i, ela está na ordem reversa e não está adequada para comparação com C_{i+1}. Nesse ponto o truque na prova aparece: Escrevemos uma história de computação de aceitação diferentemente. Alternadamente, uma em cada configuração aparece na ordem reversa. As posições ímpares permanecem escritas na ordem de-trás-para-a-frente, mas as posições pares são escritas de-frente-para-trás. Por conseguinte uma história de computação de aceitação apareceria como mostrado na Figura 5.4.

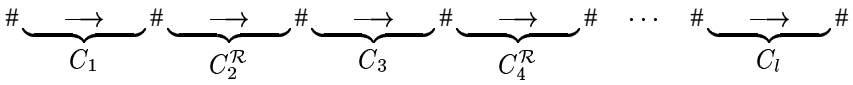

Figura 5.4: Alternadamente, uma em cada duas configurações escritas em ordem revers

Nessa forma modificada o AP é capaz de empilhar uma configuração de modo que quando ela é desempilhada a ordem está adequada para comparação com a próxima. Projetamos D para aceitar qualquer cadeia que não é uma história de computação de aceitação na forma modificada.

5.2 Um problema indecidível simples . . . . . .

Nesta seção mostramos que o fenômeno da indecidibilidade não está confinado a problemas concernentes a autômatos. Damos um exemplo de um problema indecidível concernente a simples manipulações de cadeias. É chamado o ***problema da correspondência de Post***, ou ***PCP***.

Podemos descrever esse problema facilmente como um tipo de charada. Começamos com uma coleção de dominós, cada um combinando duas cadeias, uma em cada lado. Um dominó individual tem a seguinte aparência

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]$$

e uma coleção de dominós é da forma

$$\left\{\left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right]\right\}$$

O trabalho é fazer uma lista desses dominós (repetições permitidas) de modo que a cadeia que obtemos lendo os símbolos em cima é a mesma que a cadeia de símbolos embaixo. Essa lista é chamada um ***emparelhamento***. Por exemplo, a seguinte lista é um emparelhamento para essa charada.

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{b}}{\mathtt{ca}}\right]\left[\frac{\mathtt{ca}}{\mathtt{a}}\right]\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{abc}}{\mathtt{c}}\right].$$

Lendo a cadeia de cima obtemos abcaaabc, que é a mesma ao ler a de baixo. Podemos ilustrar esse emparelhamento deformando os dominós de modo que os símbolos correspondentes de cima e de baixo se alinham.

Para algumas coleções de dominós encontrar um emparelhamento pode não ser possível. Por exemplo, a coleção

$$\left\{\left[\frac{\mathtt{abc}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{acc}}{\mathtt{ba}}\right]\right\}$$

não pode conter um emparelhamento porque toda cadeia de cima é maior que a cadeia de baixo correspondente.

O problema da correspondência de Post é determinar se uma coleção de dominós tem um emparelhamento. Esse problema é insolúvel por algoritmos.

Antes de chegar ao enunciado formal desse teorema e sua prova, vamos enunciar o problema precisamente e então expressá-lo como uma linguagem. Uma instância do PCP é uma coleção P de dominós

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\},$$

e um emparelhamento é uma seqüência $i_1, i_2, \ldots, i_l$, onde $t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$. O problema é determinar se P tem um emparelhamento. Seja

$PCP = \{\langle P \rangle \mid P$ é uma instância do problema da correspondência de Post com um emparelhamento$\}$.

Teorema 5.11 .....
PCP é indecidível.

.....

Idéia da prova. Conceitualmente esta prova é simples, embora tenha muitos detalhes técnicos. A técnica principal é redução a partir de A_{MT} via histórias de computação de aceitação. Mostramos que de qualquer MT M e entrada w podemos construir uma instância P onde um emparelhamento é uma história de computação de aceitação para M sobre w. Se pudéssemos determinar se a instância tem um emparelhamento, seríamos capazes de determinar se M aceita w.

Como podemos construir P de modo que o emparelhamento seja uma história de computação de aceitação para M sobre w? Escolhemos os dominós em P de modo que fazer um emparelhamento força que uma simulação de M ocorra. No emparelhamento, cada dominó liga uma posição ou posições em uma configuração com a(s) correspondente(s) na próxima configuração.

Antes de chegar à construção tratamos dois pequenos detalhes técnicos. (Não se preocupe demais com eles na sua primeira leitura dessa construção.) Primeiro, por conveniência em construir P, assumimos que M sobre w nunca tenta mover sua cabeça para fora da extremidade esquerda da fita. Isso requer primeiro alterar M para evitar esse comportamento. Segundo, modificamos o PCP para requerer que um emparelhamento comece com o primeiro dominó

$$\left[\frac{t_1}{b_1}\right].$$

Mais adiante mostramos como eliminar esse requisito. Chamamos esse problema o problema da correspondência de Post modificado, PCPM. Seja

$PCPM = \{\langle P \rangle \mid P$ é uma instância do problema da correspondência de Post com um emparelhamento que começa com o primeiro dominó$\}$.

Agora vamos aos detalhes da prova e projetar P para simular M sobre w.

Prova. Supomos que a MT R decide o PCP e construimos S que decide A_{MT}. Seja

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{aceita}}, q_{\text{rejeita}}),$$

onde Q, Σ, Γ, e δ, são o conjunto de estados, o alfabeto de entrada, o alfabeto de fita, e a função de transição de M, respectivamente.

Nesse caso S constrói uma instância do PCP P que tem um emparelhamento se e somente se M aceita w. Para fazer isso S primeiro constrói uma instância P' do PCPM. Descrevemos a construção em sete partes, cada uma das quais realiza um aspecto específico de se simular M sobre w. Para explicar o que estamos fazendo intercalamos a construção com um exemplo da construção em ação.

Parte 1. A construção começa da seguinte maneira

$$\text{Ponha } \left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right] \text{ em } P' \text{ como o primeiro dominó } \left[\frac{t_1}{b_1}\right].$$

Devido ao fato de que P' é uma instância do PCPM o emparelhamento tem que começar com esse dominó. Por conseguinte a cadeia de baixo começa corretamente com $C_1 = q_0 w_1 w_2 \cdots w_n$, a primeira configuração na história de computação de aceitação para M sobre w, como mostrado na Figura 5.5.

Figura 5.5: Início do emparelhamento do PCPM

Nessa ilustração do emparelhamento parcial obtido até então, a cadeia de baixo consiste de $\#q_0 w_1 w_2 \cdots w_n \#$ e a cadeia de cima consiste apenas de #. Para obter um emparelhamento precisamos estender a cadeia de cima para emparelhar com a cadeia de baixo. Fornecemos dominós adicionais para permitir essa extensão. Os dominós adicionais fazem com que a próxima configuração de M apareça na extensão da cadeia de baixo forçando uma simulação de um único-passo de M.

Nas partes 2, 3, e 4, adicionamos a P' dominós que realizam a parte principal da simulação. A parte 2 lida com os movimentos da cabeça para a direita, a parte 3 lida com os movimentos da cabeça para a esquerda, e a parte 4 lida com as células da fita não adjacentes à cabeça.

Parte 2. Para todos $a, b \in \Gamma$ e todos $q, r \in Q$ onde $q \neq q_{\text{rejeita}}$,

$$\text{se } \delta(q, a) = (r, b, \text{D}), \text{ ponha } \left[\frac{qa}{br}\right] \text{ em } P'.$$

Parte 3. Para todos $a, b, c \in \Gamma$ e todos $q, r \in Q$ onde $q \neq q_{\text{rejeita}}$,

$$\text{se } \delta(q, a) = (r, b, \text{E}), \text{ ponha } \left[\frac{cqa}{rcb}\right] \text{ em } P'.$$

Parte 4. Para todo $a \in \Gamma$,

$$\text{ponha } \left[\frac{a}{a}\right] \text{ em } P'.$$

Agora criamos um exemplo hipotético para ilustrar o que construimos até agora. Seja $\Gamma = \{\texttt{0}, \texttt{1}, \texttt{2}, \sqcup\}$. Digamos que w é a cadeia `0100` e que o estado inicial de M é q_0. No estado q_0, ao ler um `0`, digamos que a função de transição ordena que M

entre no estado q_7, escreva um 2 na fita, e mova sua cabeça para a direita. Em outras palavras, $\delta(q_0, 0) = (q_7, 2, \mathrm{D})$.

A parte 1 coloca o dominó

$$\left[\frac{\#}{\#q_0 0100}\right] = \left[\frac{t_1}{b_1}\right]$$

em P', e o emparelhamento começa:

(figura 1 da pagina 186)

Adicionalmente, a parte 2 coloca o dominó

$$\left[\frac{q_0 0}{2 q_7 r}\right]$$

pois $\delta(q_0, 0) = (q_7, 2, \mathrm{D})$ e a parte 4 coloca o dominó

$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right], \text{ e } \left[\frac{\sqcup}{\sqcup}\right]$$

em P', pois 0, 1, 2, e $\sqcup$ são os membros de Γ. Isso, juntamente com a parte 5, nos permite estender o emparelhamento da seguinte forma:

(figura 2 da pagina 186)

Por conseguinte os dominós das partes 2,3, e 4 nos deixam estender o emparelhamento adicionando a segunda configuração após a primeira. Queremos que esse processo continue, adicionando a terceira configuração, e aí a quarta, e assim por diante. Para isso acontecer precisamos adicionar um dominó a mais para copiar o símbolo #.

Parte 5.

$$\text{Ponha } \left[\frac{\#}{\#}\right] \text{ e } \left[\frac{\#}{\sqcup \#}\right] \text{ em } P'.$$

O primeiro desses dominós nos permite copiar o símbolo # que marca a separação das configurações. Além disso, o segundo dominó nos permite adicionar um símbolo branco $\sqcup$ no final da configuração para simular a quantidade infinita de brancos à direita que são suprimidos quando escrevemos a configuração.

Continuando com o exemplo, vamos dizer que no estado q_7, ao ler um 1 M vai para o estado q_5, escreve um 0, e move a cabeça para a direita. Ou seja, $\delta(q_7, 1) = (q_5, 0, \mathrm{D})$. Então temos o dominó

$$\left[\frac{q_7 1}{0 q_5}\right] \text{ em } P'.$$

Portanto o último emparelhamento parcial se estende para

(figura 1 da pagina 187)

Então, suponha que no estado q_5, ao ler um 0, M vai para o estado q_9, escreve um 2, e move sua cabeça para a esquerda. Portanto $\delta(q_5, 0) = (q_9, 2, \mathrm{E})$. Então temos os dominós

$$\left[\frac{0 q_5 0}{q_9 02}\right], \left[\frac{1 q_5 0}{q_9 12}\right], \left[\frac{2 q_5 0}{q_9 22}\right], \text{ e } \left[\frac{\sqcup q_5 0}{q_9 \sqcup 2}\right].$$

O primeiro é relevante porque o símbolo à esquerda da cabeça é um 0. O emparelhamento parcial precedente se estende para

(figura 2 da pagina 187)

Note que, à medida que construímos um emparelhamento, somos forçados a simular M sobre a entrada w. Esse processo continua até que M atinja um estado de parada. Se um estado de aceitação ocorre, queremos fazer com que a parte de cima do emparelhamento parcial "acompanhe" a parte de baixo de modo que o emparelhamento seja completo. Podemos arranjar para que isso aconteça adicionando mais dominós.

Parte 6. Para todo $a \in \Gamma$,

$$\text{ponha } \left[\frac{aq_{\text{aceita}}}{q_{\text{aceita}}}\right] \text{ e } \left[\frac{q_{\text{aceita}}a}{q_{\text{aceita}}}\right] \text{ em } P'.$$

Esse passo tem o efeito de adicionar "pseudo-passos" da máquina de Turing depois que ela tenha parado, onde a cabeça "come" símbolos adjacentes até que nenhum mais permaneça. Continuando com o exemplo anterior, se o emparelhamento parcial até o ponto quando a máquina pára em um estado de aceitação é

(figura 1 da pagina 188)

Os dominós que acabamos de acrescentar permitem que o emparelhamento continue:

(figura 2 da pagina 188)

Parte 7. Finalmente adicionamos o dominó

$$\left[\frac{q_{\text{aceita}}\#\#}{\#}\right]$$

e completamos o emparelhamento:

(figura 3 da pagina 188)

Isso conclui a construção de P'. Lembre-se que P' é uma instância do PCPM na qual o emparelhamento simula a computação de M sobre w. Para terminar a prova, lembremo-nos que o PCPM difere do PCP no sentido de que o emparelhamento tem que começar com o primeiro dominó na lista. Se olharmos para P' como uma instância do PCP ao invés do PCPM, ele obviamente contém um emparelhamento, independentemente de se M pára sobre w. Você pode vê-lo? (Dica: É muito curto.)

Agora mostramos como converter P' em P, uma instância do PCP que ainda simula M sobre w. Fazemos isso com um truque um tanto técnico. A idéia é construir o requisito de começar com o primeiro dominó direto no problema do modo que enunciar o requisito fica desnecessário. Precisamos introduzir um pouco de notação para esse propósito.

Seja $u = u_1u_2\cdots u_n$ uma cadeia qualquer de comprimento n. Defina $\star u$, $u\star$, e $\star u\star$ como sendo as cadeias

$$\begin{aligned}
\star u &= *u_1 * u_2 * u_3 * \cdots * u_n \\
u\star &= u_1 * u_2 * u_3 * \cdots * u_n* \\
\star u\star &= *u_1 * u_2 * u_3 * \cdots * u_n * .
\end{aligned}$$

Aqui, $\star u$ adiciona o símbolo $*$ antes de cada caracter em u, $u\star$ adiciona um após cada caracter em u, e $\star u\star$ adiciona um tanto antes quanto depois de cada caracter em u.

Para converter P' em P, uma instância do PCP, fazemos o seguinte. Se P' fosse a coleção

$$\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \left[\frac{t_3}{b_3}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\},$$

fazemos P ser a coleção

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star}\right], \left[\frac{\star t_1}{b_1 \star}\right], \left[\frac{\star t_2}{b_2 \star}\right], \left[\frac{\star t_3}{b_3 \star}\right], \cdots, \left[\frac{\star t_k}{b_k \star}\right], \left[\frac{* \Diamond}{\Diamond}\right] \right\}.$$

porque ela é a única na qual ambas as partes de cima e de baixo começam com o mesmo símbolo, a saber $*$. Além de forçar que o emparelhamento comece com o primeiro dominó, a presença dos $*$'s não afeta possíveis emparelhamentos porque eles simplesmente se intercalam com os símbolos originais. Os símbolos originais agora ocorrem nas posições pares do emparelhamento. O dominó

$$\left[\frac{* \Diamond}{\Diamond}\right]$$

está aí para permitir que a parte de cima acrescente o $*$ a mais no final do emparelhamento.

. . . . . .

5.3 Redutibilidade por mapeamento . . . . . .

Mostramos como usar a técnica da redutibilidade para provar que vários problemas são indecidíveis. Nesta seção formalizamos a noção de redutibilidade. Fazer isso nos permite usar redutibilidade de maneiras mais refinadas, tais como provar que certas linguagens não são Turing-reconhecíveis e para aplicações em teoria da complexidade.

A noção de reduzir um problema a outro pode ser definida formalmente de uma dentre diversas maneiras. A escolha de qual delas usar depende da aplicação. Nossa escolha é um tipo simples de redutibilidade chamado ***redutibilidade por mapeamento***.[2]

A grosso modo, ser capaz de reduzir o problema A ao problema B usando uma redutibilidade por mapeamento significa que uma função computável existe que converte instâncias do problema A para instâncias do problema B. Se tivermos tal função de conversão, chamado de uma *redução*, podemos resolver A com um resolvedor para B. A razão é que qualquer instância de A pode ser resolvida primeiro usando uma redução para convertê-la numa instância de B e aí aplicar o resolvedor para B. Uma definição precisa de redutibilidade por mapeamento segue em breve.

Funções computáveis

Uma máquina de Turing computa uma função iniciando com a entrada para a função sobre a fita e parando com a saída da função na fita.

Definição 5.12 . . . . . .
Uma função $f : \Sigma^* \longrightarrow \Sigma^*$ é uma ***função computável*** se alguma máquina de Turing M, sobre toda entrada w, pára com exatamente $f(w)$ na sua fita.

[2] Ele é chamado ***redutibilidade muitos-para-um*** em alguns outros livros-texto.

Exemplo 5.13 .....
Todas as operações aritméticas usuais sobre inteiros são funções computáveis. Por exemplo, podemos fazer uma máquina que toma como entrada $\langle m, n \rangle$ e retorna $m + n$, a soma de m e n. Não damos quaisquer detalhes aqui, deixando-os como exercícios.

Exemplo 5.14 .....
Funções computáveis podem ser transformações de descrições de máquinas. Por exemplo, uma função computável f toma como entrada w e retorna a descrição de uma máquina de Turing $\langle M' \rangle$ se $w = \langle M \rangle$ é uma codificação de uma máquina de Turing M. A máquina M' é uma máquina que reconhece a mesma linguagem que M, mas nunca tenta mover sua cabeça para fora da extremidade esquerda de sua fita. A função f realiza essa tarefa adicionando vários estados à descrição de M. A função retorna ε se w não é uma codificação legítima de uma máquina de Turing.

Definição formal de redutibilidade por mapeamento

Agora definimos redutibilidade por mapeamento. Como de costume representamos problemas computacionais por linguagens.

Definição 5.15 .....
A linguagem A é ***redutível por mapeamento*** à linguagem B, escrito $A \leq_{\mathrm{m}} B$, se existe uma função computável $f : \Sigma^* \longrightarrow \Sigma^*$, onde para toda w,

$$w \in A \Longleftrightarrow Bf(w) \in B.$$

A função f é chamada a ***redução*** de A para B.

A Figura 5.6 ilustra a redutibilidade por mapeamento.

Figura 5.6: A função f reduzindo A para B

Uma redução por mapeamento de A para B provê uma maneira de converter questões sobre teste de pertinência em A para teste de pertinência em B. Para testar se $w \in A$, usamos a redução f para mapear w para $f(w)$ e testar se $f(w) \in B$. O termo *redução por mapeamento* vem da função ou mapeamento que provê os meios de se fazer a redução.

Se um problema é redutível por mapeamento a um segundo, que já tenha sido previamente resolvido, podemos portanto obter uma solução para o problema original. Capturamos essa idéia no teorema seguinte.

Teorema 5.16 .....
Se $A \leq_{\mathrm{m}} B$ e B é decidível, então A é decidível.

Prova. Fazemos M ser o decisor para B e f a redução de A para B. Descrevemos um decisor N para A da seguinte forma.

$N =$ "Sobre a entrada w:

1. Compute $f(w)$.

2. Rode M sobre a entrada $f(w)$ e dê como saída o que quer que M dê como saída."

Claramente, se $w \in A$, então $f(w) \in B$ porque f é uma redução de A para B. Por conseguinte M aceita $f(w)$ sempre que $w \in A$. Daí N funciona como desejado.

O corolário seguinte do Teorema 5.16 tem sido nossa principal ferramenta para provar indecidibilidade.

Corolário 5.17
Se $A \leq_{\mathrm{m}} B$ e A é indecidível, então B é indecidível.

Agora revisitamos algumas de nossas provas anteriores que usaram o método da redutibilidade para obter exemplos de redutibilidades por mapeamento.

Exemplo 5.18
No Teorema 5.1 usamos uma redução de A_{MT} para provar que $PARA_{\mathsf{MT}}$ é indecidível. Essa redução mostrou como um decisor para $PARA_{\mathsf{MT}}$ poderia ser usado para dar um decisor para A_{MT}. Podemos demonstrar uma redutibilidade por mapeamento de A_{MT} para $PARA_{\mathsf{MT}}$ como segue. Para fazer isso temos que apresentar uma função f que toma uma entrada da forma $\langle M, w\rangle$ e retorna uma saída da forma $\langle M', w'\rangle$, onde

$$\langle M, w\rangle \in A_{\mathsf{MT}} \text{ se e somente se } \langle M', w'\rangle \in PARA_{\mathsf{MT}}.$$

A seguinte máquina F computa uma redução f.

$F =$ "Sobre a entrada $\langle M, w\rangle$:

1. Construa a seguinte máquina M'.

   $M' =$ "Sobre a entrada x:

   1. Rode M sobre x.
   2. Se M aceita, *aceite*.
   3. Se M rejeita, entre em loop."

2. Dê como saída $\langle M', w'\rangle$."

Exemplo 5.19
A prova da indecidibilidade do problema da correspondência de Post no Teorema 5.11 contém duas reduções por mapeamento. Primeiro, ela mostra que $A_{\mathsf{MT}} \leq_{\mathrm{m}} PCPM$ e então ela mostra que $PCPM \leq_{\mathrm{m}} PCP$. Em ambos os casos podemos facilmente obter a verdadeira função de redução e mostrar que ela é uma redução por mapeamento. Como o Exercício 5.6 mostra, redutibilidade por mapeamento é transitiva, de modo que essas duas reduções juntas implicam que $A_{\mathsf{MT}} \leq_{\mathrm{m}} PCP$.

Exemplo 5.20 .....
Uma redução por mapeamento de V_{MT} para EQ_{MT} está por trás da prova do Teorema 5.4. Nesse caso a redução f mapeia a entrada $\langle M \rangle$ para a saída $\langle M, M_1 \rangle$, onde M_1 é a máquina que rejeita todas as entradas.

Exemplo 5.21 dotfill
A prova do Teorema 5.2 mostrando que V_{MT} é indecidível ilustra a diferença entre a noção formal de redutibilidade por mapeamento que definimos nesta seção e a noção informal de redutibilidade que usamos anteriormente neste capítulo. A prova mostra que V_{MT} é indecidível reduzindo A_{MT} para ele. Vamos ver se podemos converter essa redução em uma redutibilidade por mapeamento.

Da redução original podemos facilmente construir uma função f que toma como entrada $\langle M, w \rangle$ e produz a saída $\langle M_1 \rangle$, onde M_1 é a máquina de Turing descrita naquela prova. Mas M aceita w se e somente se $L(M_1)$ *não* é vazia portanto f é uma redução por mapeamento de A_{MT} para $\overline{V_{\mathsf{MT}}}$. Ela ainda mostra que V_{MT} é indecidível porque decidibilidade não é afetada por complementação, mas ela não dá uma redução por mapeamento de A_{MT} para V_{MT}. Na verdade, nenhuma tal redução existe, como lhe pede para mostrar o Exercício 5.5.

A sensitividade da redutibilidade por mapeamento à complementação é importante no uso da redutibilidade para provar não-reconhecibilidade de certas linguagens. Podemos também usar redutibilidade por mapeamento para mostrar que problemas não são Turing-reconhecíveis. O teorema seguinte é análogo ao Teorema 5.16.

Teorema 5.22 .....
Se $A \leq_{\mathrm{m}} B$ e B é Turing-reconhecível, então A é Turing-reconhecível.

A prova é a mesma que a do Teorema 5.16, exceto que M e N são reconhecedores ao invés de decisores.

Corolário 5.23 .....
Se $A \leq_{\mathrm{m}} B$ e A não é Turing-reconhecível, então B não é Turing-reconhecível.

Em uma aplicação típica desse corolário, fazemos A ser $\overline{A_{\mathsf{MT}}}$, o complemento de A_{MT}. Sabemos que $\overline{A_{\mathsf{MT}}}$ não é Turing-reconhecível do Corolário 4.17. A definição de redutibilidade por mapeamento implica que $A \leq_{\mathrm{m}} B$ significa o mesmo que $\overline{A} \leq_{\mathrm{m}} \overline{B}$. Para provar que B não é reconhecível podemos mostrar que $A_{\mathsf{MT}} \leq_{\mathrm{m}} \overline{B}$. Podemos também usar redutibilidade por mapeamento para mostrar que certos problemas não são nem Turing-reconhecíveis nem co-Turing-reconhecíveis como no teorema seguinte.

Teorema 5.24 .....
EQ_{MT} nem é Turing-reconhecível nem co-Turing-reconhecível.

Prova. Primeiro mostramos que EQ_{MT} não é Turing-reconhecível. Fazemos isso mostrando que A_{MT} é redutível a $\overline{EQ_{\mathsf{MT}}}$. A função de redução f funciona da seguinte forma.

$F =$ "Sobre a entrada $\langle M, w \rangle$ onde M é uma MT e w uma cadeia:

1. Construa as seguintes máquinas de Turing M_1 e M_2.

   M_1 = "Sobre qualquer entrada:

   1. *Rejeite*."

   M_2 = "Sobre qualquer entrada:

   1. Rode M sobre w. Se ela aceita, *aceite*."

2. Dê como saída $\langle M_1, M_2 \rangle$."

Aqui, M_1 não aceita nada. Se M aceita w, M_2 aceita tudo, e portanto as duas máquinas não são equivalentes. Reciprocamente, se M não aceita w, M_2 não aceita nada, e elas são equivalentes. Por conseguinte f reduz A_{MT} a $\overline{EQ_{\mathsf{MT}}}$, como desejado.

Para mostrar que $\overline{EQ_{\mathsf{MT}}}$ não é Turing-reconhecível damos uma redução de A_{MT} para o complemento de $\overline{EQ_{\mathsf{MT}}}$, a saber, EQ_{MT}. Logo, mostramos que $A_{\mathsf{MT}} \leq_{\mathrm{m}} EQ_{\mathsf{MT}}$. A seguinte MT G computa a função redutora g.

G = "A entrada é $\langle M, w \rangle$ onde M é uma MT e w uma cadeia.

1. Construa as seguintes máquinas de Turing M_1 e M_2: M_1 = "Sobre qualquer entrada:

   1. *Aceite*."

   M_2 = "Sobre qualquer entrada:

   1. Rode M sobre w.
   2. Se ela aceita, *aceite*."

2. Dê como saída $\langle M_1, M_2 \rangle$."

A única diferença entre f e g é na máquina M_1. Em f, a máquina M_1 sempre rejeita, enquanto que em g ela sempre aceita. Em ambas f e g, M aceita w se e somente se M_2 sempre aceita. Em g, M aceita w se e somente se M_1 e M_2 são equivalentes. É por isso que g é uma redução de A_{MT} para EQ_{MT}.

. . . . . .

. . . . . .

Exercícios

5.1 Mostre que EQ_{GLC} é indecidível.

5.2 Mostre que EQ_{GLC} é co-Turing-reconhecível.

5.3 Encontre um emparelhamento na seguinte instância do PCP.

$$\left\{ \left[\frac{\mathtt{ab}}{\mathtt{abab}}\right], \left[\frac{\mathtt{b}}{\mathtt{a}}\right], \left[\frac{\mathtt{aba}}{\mathtt{b}}\right], \left[\frac{\mathtt{aa}}{\mathtt{a}}\right] \right\}$$

5.4 Se $A \leq_{\mathrm{m}} B$ e B é uma linguagem regular, isso implica que A seja uma linguagem regular? Por que ou por que não?

5.5 Mostre que A_{MT} não é redutível por mapeamento a V_{MT}.