Capítulo 10

Limites da Computação Algorítmica: Problemas Indecidíveis

Tendo estudado o que as máquinas de Turing podem fazer, estudaremos, agora, o que elas não podem fazer. Embora a tese de Turing nos leve a acreditar que essas limitações não são muitas, em várias ocasiões observamos que poderia não existir um algoritmo para resolver certos problemas. Agora, explicitaremos o que queremos dizer por isso. Alguns dos resultados apareceram de uma maneira simples. Se uma linguagem não é recursiva, então por definição não existe algoritmo de pertinência para êle. Se essas fossem as questões abordadas, elas não seriam interessantes. As linguagens não recursivas têm pouca utilidade. Mas as questões são mais profundas. Por exemplo, estabelecemos (embora não tenhamos provado ainda) que não existe algoritmo para determinar se uma gramática livre de contexto é ambigua ou não. Esta questão é, claramente, de relevância prática, no estudo das linguagens de programação.

O argumento de que o poder das computações mecânicas é limitado não é surpreendente. Intuitivamente, sabemos que muitas questões vagas e especulativas requerem idéias e raciocínios bem além da capacidade de qualquer computador previsível. O que é mais interessante para o cientista da computação é que existem questões que podem ser clara e simplesmente estabelecidos e tal que aparentemente tem solução algorítmica, mas que se sabe não ser solúvel por qualquer computador.

Antes de tudo, definiremos os conceitos de **decibilidade** e **computabilidade** para precisar o que queremos dizer com a afirmação de que algo não pode ser feito por uma máquina de Turing. Veremos, então, vários problemas clássicos desse tipo, dentre eles o bem conhecido problema da parada para máquinas de Turing. Desse problema segue um número de problemas relacionados para máquinas de Turing e linguagens recursivamente enumeráveis.

10.1 Computabilidade e decibilidade

Na definição 8.1.12, estabelecemos que uma função f, num certo domínio, diz-se computável se existe uma máquina de Turing que computa o valor de f, para todos os argumentos no seu domínio. Como funções mapeam elementos de um conjunto (domínio) em outro conjunto (contradomínio) e maqúinas de Turing, por outro lado, transformam cadeias de um alfabeto (Σ) em cadeias de um alfabeto (Γ), devemos encarar Σ^* e Γ^*, como codificações ou representações do domínio e contradomínio, respectivamente. Uma função é não computável se tal máquina de Turing não existe. Pode existir uma máquina de Turing que computa f em parte de seu domínio, mas, para nós, uma função é computável somente se existe uma máquina de Turing que computa a função sobre todo seu domínio. Disso podemos inferir que quando classificamos uma função como computável ou não computável, devemos ser claros sobre o que é seu domínio.

Por simplicidade, estudaremos uma classe de problemas específica: a dos **problemas de decisão**, que são problemas de determinar se um elemento de algum universo pertence ou não a um determinado conjunto. Se existir um algoritmo que receba um elemento a e dê como resultado um simples "sim" caso $a \in A$ ou "não", caso $a \notin A$, dizemos que o problema de decisão para o conjunto A é **decidível**. Se tal algoritmo não existir, então dizemos que o problema de decisão para o conjunto A é **indecidível**. Observe que todo problema decidível é computável, no sentido de que existe um algoritmo que computa uma solução para o problema, mas o contrário não é correto, uma vez que a decibilidade está atrelada à classe de problemas de decisão. Um problema de decisão pode ser entendido como um conjunto de assertivas que são verdadeiras ou falsas, dependendo do objeto sobre o qual predicam. Por exemplo, consideremos a assertiva "Para uma gramática livre de contexto, G, a linguagem $L(G)$ é inerentemente ambigua". Para algumas gramáticas livres de contexto, G, isso é verdadeiro, mas para outras é falso. Porém, para qualquer gramática livre de contexto, G, devemos ter uma ou outra situação. Como o problema é decidir se a assertiva é verdadeira para alguma gramática livre de contexto, o domínio ou universo subjacente, é o conjunto de todas as gramáticas livres de contexto. Assim, um problema é decidível se existir uma máquina de Turing que dá a resposta correta para cada assertiva no domínio do problema.

Quando estabelecemos resultados de decibilidade ou indecibilidade, deveremos, sempre, saber qual é o domínio, porque pode afetar a conclusão. Um problema pode ser decidível sobre algum domínio mas não sobre outro. Especificamente uma única instância do problema é sempre decidível, desde que a resposta é ou verdadeira ou falsa. No primeiro caso a máquina de Turing que sempre responde "verdadeiro" dá resposta correta, enquanto no segundo caso uma que responde sempre "falso" é apropriada. Isso pode parecer uma resposta falaciosa, mas enfatiza um ponto importante. O fato de que não sabemos qual a resposta correta não faz nenhuma diferença, o que importa é que existe alguma máquina de Turing que dê a resposta correta.

10.2 O Problema da Parada para Máquinas de Turing

Começaremos com alguns problemas com alguma significância histórica e ao mesmo tempo que nos dá um ponto de partida para desenvolver outros resultados. O mais conhecido desses problemas é o **problema da parada**, para as máquinas de Turing. Estabelecido de modo simples, o problema é: *dado a descrição de uma máquina de Turing M e uma entrada w, quando iniciado na configuração inicial q_0w, ela efetua uma computação que pára?* usando uma maneira abreviada de falar do problema, perguntamos se M aplicada a w, ou simplesmente (M, w), pára ou não pára. O domínio desse problema é o conjunto de todas as máquinas de Turing e todo w, isto é, estamos procurando uma única máquina de Turing que, dado a descrição de uma máquina de Turing arbitrária M e w, prevê se a computação de M aplicado a w parará.

Não poderemos achar a resposta simulando a ação de M sôbre w, digamos efetuando numa máquina de Turing universal, porque não existe limite no comprimento da computação. Se M entra num laço infinito, então não importa quanto tempo esperemos, não poderemos, jamais, estar seguros de que M, realmente, está num laço. Pode acontecer, simplesmente, o caso de uma computação muito longa. O que precisamos é um algoritmo que possa determinar a resposta correta para qualquer M e w, ao efetuar uma análise na descrição da máquina e a entrada. Mas, como veremos, tal algoritmo não existe.

Para discussões posteriores, é conveniente ter uma idéia precisa do que queremos dizer com o problema da parada. Por isso, faremos a seguinte definição.

Definição 10.2.1 *Suponha que w_M descreve uma máquina de Turing, $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \Box, F\rangle$, e seja w qualquer elemento de Σ^+. Uma* **solução para o problema da parada** *é uma máquina de Turing H, a qual para qualquer w_M e w, fornece a computação*

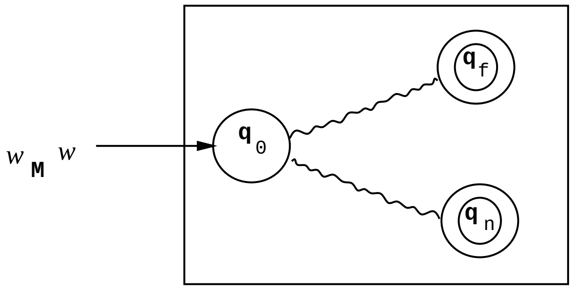

Figura 10.1: Diagrama de blocos para uma suposta máquina de Turing que resolve o problema da parada.

$$q_0 w_M w \vdash^* x_1 q_f x_2,$$

se M aplicado a w pára, e

$$q_0 w_M w \vdash^* y_1 q_n y_2,$$

se M aplicado a w não pára. Aqui q_f e q_n são ambos estados finais de H.

Teorema 10.2.2 *Não existe qualquer máquina de Turing, H, que se comporta como as exigências da definição 10.2.1. O problema da parada é portanto indecidível.*

DEMONSTRAÇÃO: Assumiremos o contrário, isto é, que existe um algoritmo, e consequentemente uma máquina de Turing H, que resolve o problema da parada. A entrada para H será a descrição (codificada de alguma forma) de M, digamos w_M, assim como a entrada w. A exigência é, então, que dado qualquer (w_M, w), a máquina de Turing H parará com um sim ou um não. Conseguiremos isso pedindo que H pare num dos dois estados finais correspondentes, digamos, q_f e q_n. A situação pode ser visualizada por um diagrama de bloco como na figura 10.1.

Esse diagrama tem a intensão de indicar que, se H começar no estado q_0, com entrada (w_M, w) ela parará no estado q_f ou q_n. Pela exigência da definição 10.2.1 devemos pedir que H opere segundo as seguintes regras:

$$q_0 w_M w \vdash^* x_1 q_f x_2,$$

se M aplicado w pára, e

$$q_0 w_M w \vdash^* y_1 q_n y_2,$$

se M aplicado a w não pára.

Em seguida modificamos H para produzir uma máquina de Turing H', como da figura 10.2.

Com o novo estado acrescentados à H queremos assegurar que existem transições entre o estado q_f e o novo estado q_a, não importa que símbolos estão na fita, de tal modo que a fita se mantém inalterada. A maneira como é feita é direta. Comparando H e H' observamos que, na situação onde H atinge q_f e pára, a máquina modificada H' entra num laço infinito. Formalmente, a ação de H' é descrita por

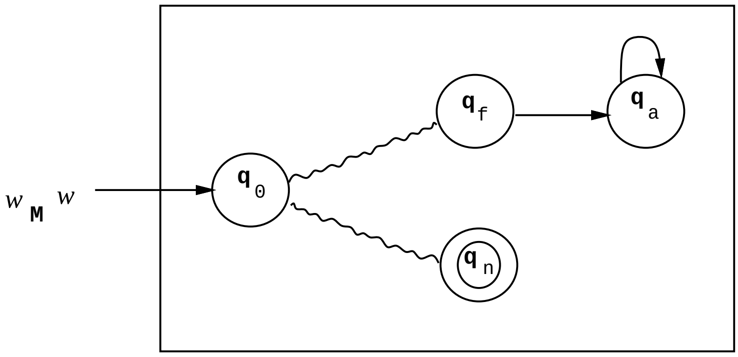

Figura 10.2: Diagrama de blocos para uma variante da suposta máquina de Turing que resolve o problema da parada

$$q_0 w_M w \vdash^*_{H'} \infty,$$

se M aplicada a w pára, e

$$q_0 w_M w \vdash^*_{H'} y_1 q_n y_2,$$

se M aplicada a w não pára.

De H' construimos uma outra máquina de Turing $\widehat{H}$. Esta nova máquina toma a entrada w_M, a cópia, e então se comporta exatamente com H'. Assim, a ação de $\widehat{H}$ é tal que

$$q_0 w_M \vdash^*_{\widehat{H}} q_0 w_M w_M \vdash^*_{\widehat{H}} \infty,$$

se M aplicada a w_M pára, e

$$q_0 w_M \vdash^*_{\widehat{H}} q_0 w_M w_M \vdash^*_{\widehat{H}} y_1 q_n y_2,$$

se M aplicado a w_M não pára.

Agora, $\widehat{H}$ é uma máquina de Turing, que terá alguma descrição em Σ^+, digamos $\widehat{w}$. Essa cadeia além de ser a descrição de $\widehat{H}$ pode, também, ser usada como entrada. Podemos, seguramente, perguntar o que aconteceria se $\widehat{H}$ fosse aplicada a $\widehat{w}$. Assim, identificando M com $\widehat{H}$, obteremos

$$q_0 \widehat{w} \vdash^*_{\widehat{H}} \infty,$$

se $\widehat{H}$ aplicada a $\widehat{w}$ pára, e

$$q_0 \widehat{w} \vdash^*_{\widehat{H}} y_1 q_n y_2,$$

se $\widehat{H}$ aplicada a $\widehat{w}$ não pára. Isto é claramente um absurdo. Essa contradição resultou da suposição de que H existe e portanto a decibilidade do problema da parada, deve ser falsa. ■

Alguém poderia objetar, da definição 10.2.1, que foi exigido da máquina H para resolver o problema da párada que começasse e terminasse numa configuração específica. Não é dificil vêr, no entanto, que as condições escolhidas de modo arbitrário desempenha um papel muito pequeno no argumento. Essencialmente, o mesmo argumento poderia ser usado com quaisquer outras configurações de início e fim. Ligamos o problema a uma configuração específica somente para facilitar o argumento, sem, no entanto, afetar a conclusão.

É importante ter em mente o que o teorema 10.2.2 nos diz. Ele não proibe a solução do problema da parada para casos específicos. Frequentemente podemos, via uma análise de M e w, dizer se a máquina de Turing parará ou não. O que o teorema diz é que isso não pode ser feito sempre. Não existe um algoritmo que pode tomar a decisão correta para todo w_M e w.

O argumento para provar o teorema 10.2.2 foi dado porque êle é clássico e de interesse histórico. A conclusão do teorema é derivada, realmente, de resultados anteriores como o seguinte argumento mostra.

Teorema 10.2.3 *Se o problema da parada fosse decidível, então toda linguagem recursivamente enumerável seria recursiva. Consequentemente, o problema da parada é indecidível.*

DEMONSTRAÇÃO: Para vêr isto, seja $\mathcal{L}$ uma linguagem recursivamente enumerável sobre Σ, e seja M uma máquina de Turing que reconhece $\mathcal{L}$. Seja H a máquina de Turing que resolve o problema da parada. Construimos o seguinte procedimento:

1. Aplique H a $w_M w$. Se H disser "não" (isto é, se parar no estado q_n), então por definição, $w \notin \mathcal{L}$.

2. Se H disser "sim" (isto é, se parar no estado q_f), então aplique M a w. Mas, M deve parar. Portanto ele dirá mais cedo ou mais tarde se $w \in \mathcal{L}$ ou não.

Isto constitui um algoritmo de pertinência, tornando $\mathcal{L}$ recursiva. Mas já sabemos que existem linguagens recursivamente enumeráveis que não são recursivas. A contradição nos diz que o problema da parada é indecidível. ∎

A simplicidade com que o problema pode ser obtido de teorema 9.2.5 é uma consequência do fato de que o problema da parada e o problema de pertinência para linguagens recursivasmente enumeráveis são "quase" idênticos. A única diferença é que no problema da parada não distinguimos entre parar num estado final e não-final, enquanto no problema de pertinência fazemos essa distinção. As provas dos teoremas 9.2.5 e 10.2.2 estão proximamente relacionados, ambos são uma versão da diagonalização.

10.3 Redução de um Problema Indecidível ao Problema da Parada

O argumento usado no teorema 10.2.3 para conectar o problema da parada com o problema da pertinência, ilustra uma técnica de redução muito importante, que pode ser utilizada para demonstrar a indecibilidade de outros problemas de decisão. Dizemos que um problema A é **reduzido** a um problema B se a decibilidade de A acarreta a decibilidade de B. A idéia é expressar o problema B em termos do problema A e de problemas conhecidos como decidíveis. Assim, se soubermos que A é indecidível, podemos concluir que B também é indecidível. Inversamente, se soubermos que B é indecidível então necessariamente A é indecidível. Vamos vêr alguns exemplos para ilustrar essa idéia.

Exemplo 10.3.1 O problema da entrada em um estado *é o seguinte. Dada qualquer máquina de Turing $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \Box, F \rangle$ e qualquer $q \in Q$, $w \in \Sigma^+$, decidir se a máquina entrará ou não no estado q quando M é aplicado a w. Este problema é indecidível.*

Para reduzir o problema da entrada em um estado ao problema da parada, suponha que temos um algoritmo A que resolve o problema da entrada de um estado. Poderíamos, então, usá-lo para resolver o problema da parada. Por exemplo, dado qualquer M e w, primeiro modificamos M para obter $\widehat{M}$ de tal

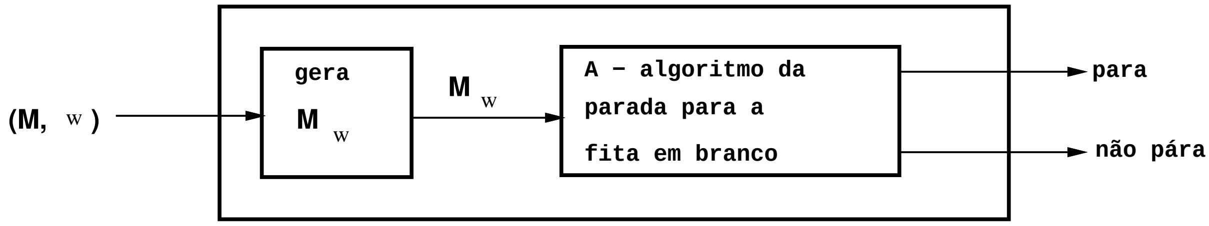

Figura 10.3: Algoritmo para o problema da parada usando um algoritmo de parada para a fita em branco.

maneira que $\widehat{M}$ *pára no estado* q *se e somente se* M *pára. Podemos fazer isso, simplesmente, olhando a função de transição* δ*. Se* M *pára, êla somente faz isso porque algum* $\delta(q_i, a)$ *é indefinido. Para obter* $\widehat{M}$ *trocamos isto por*

$$\delta(q_i, a) = (q, a, D),$$

onde q *é um estado final. Aplicamos o algoritmo da entrada de um estado* A *a* $(\widehat{M}, q, w)$*. Se* A *responde sim, isto é, o estado* q *entrará, então* (M, w) *pára. Se* A *diz não, então* (M, w) *não pára.*

Portanto a hipótese de que o problema da entrada de um estado é decidível nos dá um algoritmo para o problema da parada. Como o problema da parada é indecidível, o problema da entrada de um estado deve ser indecidível.

Exemplo 10.3.2 O problema da parada da fita em branco *é um outro problema que pode ser reduzido ao problema da parada. Dado uma máquina de Turing* M*, determine se* M *pára ou não iniciando com a fita em branco. Este problema é indecidível.*

Para provar isso, tome qualquer máquina de Turing M *e qualquer* w*. Primeiro, construiremos a partir de* M *uma nova máquina,* M_w*, que inicia com a fita em branco, escreve* w *nela, e então se posiciona na configuração* q_0w*. Após isso,* M_w *se comporta exatamente como* M*. É claro que* M_w *parará sobre uma fita em branco se e somente se* M *pára em* w*.*

Suponha, agora, que o problema da parada da fita em branco seja decidível. Dado qualquer (M, w)*, primeiro construimos* M_w*, então aplicamos o algoritmo da parada da fita em branco a ele. A conclusão nos diz se* M *aplicado a* w *parará. Como isso pode ser feito para qualquer* M *e* w*, o algoritmo para o problema da fita em branco pode ser convertido num algoritmo para o problema da parada. Como este último é indecidível o problema da parada para a fita em branco também é.*

A construção dos argumentos desses dois exemplos ilustra uma abordagem comum para estabelecer resultados de indecibilidade. Um diagrama de bloco sempre nos ajuda a visualizar o processo. A construção no exemplo 10.3.2 é resumida na figura 10.3. Naquele diagrama, primeiro usamos um algoritmo que transforma (M, w) em M_w. Tal algoritmo claramente existe. Em seguida usamos o algoritmo para resolver o problema da parada para a fita em branco, A, que assumimos existir. Pondo os dois juntos obteremos um algoritmo para o problema da parada, mas isso é impossível, concluindo que A não existe.

Um problema de decisão é uma função com valores em $\{0, 1\}$, isto é, verdadeiro ou falso. Podemos olhar também para funções mais gerais para vêr se são computáveis. Redução ao problema da parada é, também, adequado aqui. Devido à tese de Turing, esperamos que funções encontradas em cirscunstâncias práticas sejam computáveis. Assim, para achar exemplos de funções não computáveis deveremos ir um pouco além. Muitos exemplos de funções não computáveis estão associadas à tentativa de prevêr o comportamento de uma máquina de Turing.

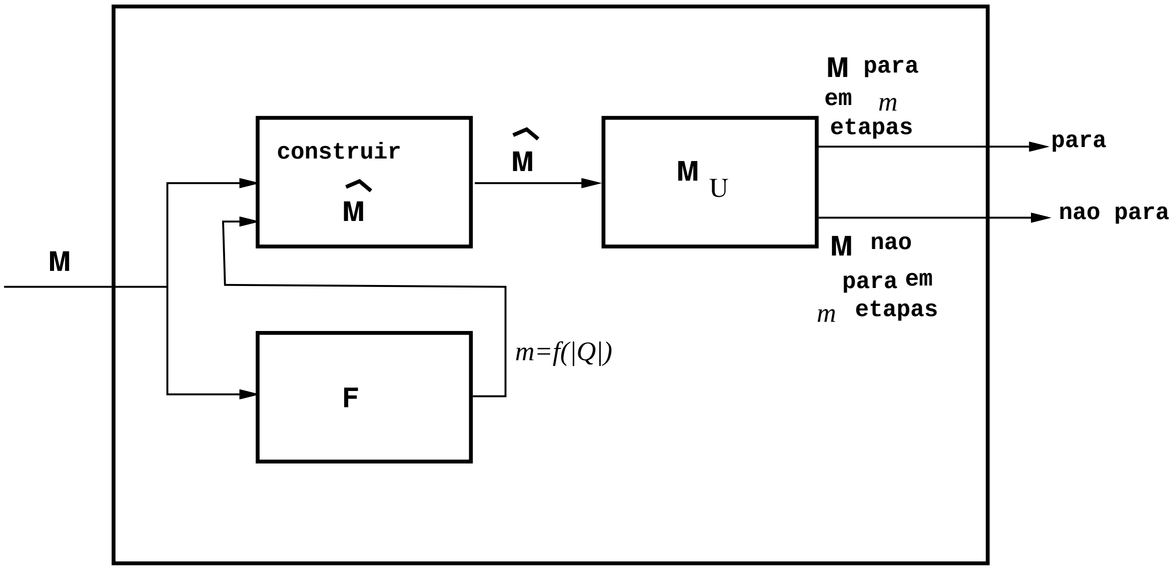

Figura 10.4: Algoritmo para o problema da parada com a fita em branco baseado num algoritmo para computar $f(n)$.

Exemplo 10.3.3 *Seja* $\Gamma = \{0, 1, \Box\}$*. Considere a função* $f(n)$ *cujo valor é o número máximo de movimentos que pode ser feito por uma máquina de Turing, de* n *estados que pára quando inicia com uma fita em branco. Esta função, como não poderia deixar de ser, não é computável.*

Antes de provarmos a afirmativa, vamos nos convencer de que $f(n)$ *é definida para todo* n*. Observe que existe um número finito de máquinas de Turing com* n *estados. Isso acontece porque* Q *e* Γ *são finitos. Portanto,* δ *tem como domínio um conjunto finito de valores. Isso, por outro lado, implica que existe somente um número finito de diferentes* δ*'s e consequentemente um número finito de diferentes máquinas de Turing com* n *estados.*

De todas as máquinas de n *estados, existem algumas que sempre param. Por exemplo, aquelas que tem somente estados finais e não efetua, portanto, nenhum movimento. Algumas das máquinas de* n *estados não param quando iniciam com uma fita em branco, mas essas não entram na definição de* f*. Toda máquina que pára executará um certo número de movimentos. Desses tomamos o maior para fornecer* $f(n)$*.*

Tome qualquer máquina de Turing, M*, e* m *um inteiro positivo. É fácil modificar* M *para produzir* $\widehat{M}$ *de tal maneira que esta última sempre parará com uma das duas respostas:* M *aplicada a uma fita em branco pára em não mais do que* m *movimentos, ou* M *aplicada a uma fita em branco faz mais do que* m *movimentos. Tudo que temos de fazer para isso é tomar* M*, contar seus movimentos e terminar quando este número exceder* m*. Assuma, agora, que* $f(n)$ *é computável por alguma máquina de Turing* F*. Podemos, então, juntar* $\widehat{M}$ *e* F *como é mostrado na figura 10.4. Primeiro computamos* $f(|Q|)$*, onde* Q *é o conjunto de estados de* M*. Isto nos diz o número máximo de movimentos que* M *pode fazer se parar. O valor obtido é, então, usado como* m *para construir* $\widehat{M}$*, como esboçado. Uma descrição de* $\widehat{M}$ *é dado a uma máquina de Turing universal para execução. Isto nos diz se* M *aplicado a uma fita em branco pára ou não em menos de* $f(|Q|)$ *etapas. Se acharmos que* M *aplicada a uma fita em branco faz mais do que* $f(|Q|)$ *movimentos, então, devido à definição de* f*, a implicação é que* M *nunca pára. Portanto, temos uma solução para o problema da parada para a fita em branco. A impossibilidade da conclusão nos leva a aceitar que* f *não é computável.*

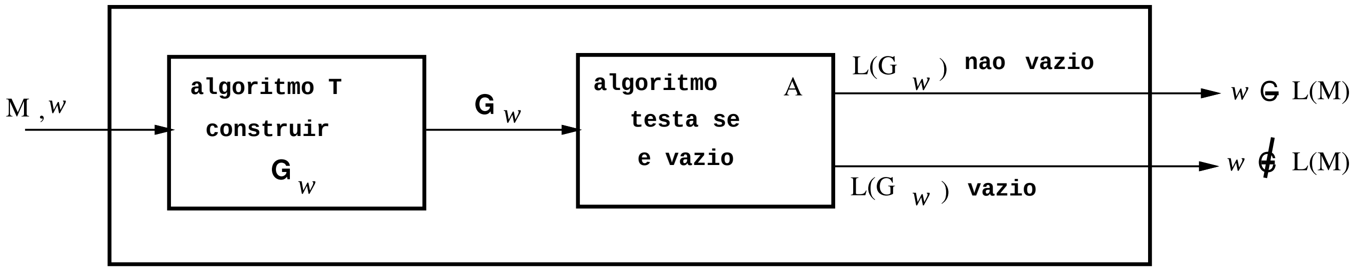

Figura 10.5: Algoritmo de pertinência para linguagens recursivamente enumeráveis.

10.4 Problemas Indecidíveis para Linguagens Recursivamente Enumeráveis

Concluimos que não existe algoritmo de pertinência para linguagens recursivamente enumeráveis. A falta de um algoritmo para decidir sobre alguma propriedade não é exceção para essas linguagens. Como veremos, podemos dizer muito pouco sobre essas linguagens. As linguagens recursivamente enumeráveis são tão gerais que, essencialmente, qualquer questão que fizermos sobre elas é indecidíveis. A demonstração de resultados específicos significa, invariavelmente, numa redução ao problema da parada ou a uma de suas consequências imediatas. Vamos dar aqui alguns exemplos para mostrar como isso é feito e daí derivar uma indicação para uma situação geral.

Teorema 10.4.1 *Seja G uma gramática irrestrita. Então o problema de determinar se*

$$L(G) = \emptyset$$

ou não é indecidível.

Demonstração: Reduziremos esse problema ao problema da pertinência para linguagens recursivamente enumeráveis. Suponha que é dado uma máquina de Turing, M, e uma cadeia w. Podemos construir uma nova máquina de Turing M' a partir de M como segue. M' põe uma cópia de sua entrada em alguma parte especial de sua fita e se comporta como M computando na fita em branco. Se ela entra num estado final, M' verifica a entrada guardada no início da computação e a aceita se e somente se ela é w. Podemos fazer isso modificando δ de um modo simples, criando para cada w uma máquina, M_w, tal que

$$L(M_w) = L(M) \cap \{w\}$$

Usando o teorema 9.3.5, construimos uma gramática G_w correspondente. Claramente, a construção levando de M e w a G_w pode, sempre, ser feito. É claro, também, que $L(G_w)$ é não vazio se e somente se $w \in L(M)$.

Suponha, agora, que existe um algoritmo, A, para decidir se $L(G) = \emptyset$ ou não. Se T denota um algoritmo por meio do qual geramos G_w, então pomos junto T e A como mostrado na figura 10.5. Essa figura é uma máquina de Turing que para qualquer M e w nos diz se $w \in L(M)$ ou não. Se tal máquina de Turing existisse teríamos um algoritmo de pertinência para qualquer linguagem recursivamente enumerável, em contradição com o resultado estabelecido anteriormente. Concluimos, portanto, que o problema estabelecido de se "$L(G) = \emptyset$ ou não" não é decidível. ■

Teorema 10.4.2 *Seja M uma máquina de Turing qualquer. A questão de se $L(M)$ é finita ou não é indecidível.*

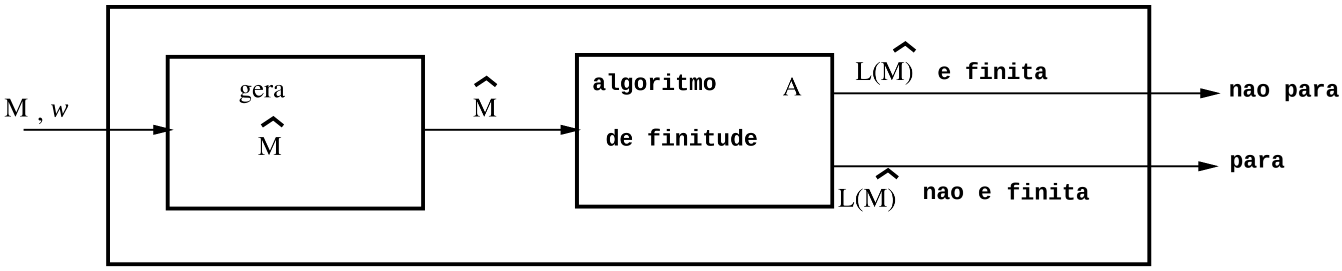

Figura 10.6: Uma solução para o problema da parada baseada num algoritmo que diz se uma linguagem é finita ou não.

Demonstração: Considere o problema da parada, (M, w). De M construiremos uma outra máquina de Turing, $\widehat{M}$, que faz o seguinte. Primeiro, os estados de parada de M são trocados de tal modo que se qualquer um é atingido, toda entrada é aceita por $\widehat{M}$. Isto pode ser conseguido fazendo cada configuração de parada ir para um estado final. Segundo, a máquina original é modificada de modo que a nova máquina, $\widehat{M}$, primeiro gera w na sua fita, então efetua as mesmas computações que M, usando o w recentemente criado e alguns outros espaços não usados. Em outras palavras, os movimentos feitos por $\widehat{M}$ após ela escrever w na fita são os mesmos que teriam sido feitos por M se ela tivesse começado na configuração original q_0w. Se M pára em qualquer configuração, então $\widehat{M}$ parará num estado final.

Portanto, se (M, w) pára, $\widehat{M}$ atingirá um estado final para todas as entradas. Se (M, w) não pára, então $\widehat{M}$, também, não parará e portanto não aceitará nada. Em outras palavras, $\widehat{M}$ ou aceita a linguagem infinita Σ^+ ou a linguagem finita $\emptyset$.

Se, agora, assumimos a existência de um algoritmo, A, que nos diz se $L(M)$ é finita ou não, podemos construir uma solução para o problema da parada como é mostrado na figura 10.6. Logo, não existe algoritmo para decidir se $L(M)$ é finito ou não. ■

Exemplo 10.4.3 *Mostre que para uma máquina de Turing arbitrária M, com $\Sigma = \{a, b\}$, o problema "$L(M)$ contém duas cadeias diferentes com o mesmo tamanho" é indecidível.*

Para mostrar isso, usamos exatamente a mesma abordagem do teorema 10.4.2, exceto que quando $\widehat{M}$ atinge uma configuração de parada, ela será modificada para aceitar as duas cadeias a e b. Para isso, a entrada inicial é guardada e no fim da computação é comparada com a e b, aceitando somente essas duas cadeias. Portanto, se (M, w) pára, $\widehat{M}$ aceitará duas cadeias de comprimentos iguais, caso contrário $\widehat{M}$ não aceitará nada. O resto do argumento procede como no teorema 10.4.2.

Exatamente do mesmo modo, podemos colocar outras questões tais como "$L(M)$ contém qualquer cadeia de comprimento 5?" ou "$L(M)$ é regular?" sem afetar essencialmente o argumento. Essas questões, assim como questões similares, são todas indecidíveis. Um resultado geral formalizando este fato é conhecido como **teorema de Rice**. Seu enunciado e prova pode ser encontrado em [HU79].

10.5 O Problema da Correspondência de Post

A indecibilidade do problema da parada tem diversas consequências de interesse prático, em particular, na área de linguagens livres de contexto. Mas em muitas situações é difícil trabalhar diretamente com o problema da parada. É conveniente estabelecer alguns resultados intermediários que preencha a distância que existe entre o problema da parada e outros problemas. Esses resultados intermediários segue da indecibilidade do problema da parada, mas são mais proximamente relacionados com os problemas que