Fundamentos da Criptografia.
Ferramentas Básicas

(Versão Parcial: 25 de abril de 2007)

Favor não distribuir

Oded Goldreich

Traduzido do original em inglês
Foundations of Cryptography. Basic Tools
(Cambridge University Press, ©2001)
por Ruy J. Guerra B. de Queiroz

Índice

Prefácio

It is possible to build a cabin with no foundations,
but not a lasting building.
Eng. Isidor Goldreich (1906–1995)

Criptografia concerne a construção de esquemas que devem ser capazes de resistir a qualquer abuso. Tais esquemas são construídos de modo a manter uma funcionalidade desejada, mesmo sob tentativas maliciosas com a intenção de fazê-las desviar de sua funcionalidade recomendada.

Capítulo 1

Introdução

Neste capítulo discutiremos brevemente os objetivos da criptografia (Seção 1.1). Em particular, discutiremos os problemas básicos da encriptação segura, assinaturas digitais, e protocolos tolerantes a falha. Esses problemas conduzem às noções de geradores pseudoaleatórios e provas de conhecimento-zero, tópicos que também são discutidos.

Nossa abordagem à criptografia é baseada em complexidade computacional. Portanto, este capítulo introdutório também contém uma seção apresentando os modelos computacionais usados no livro (Seção 1.3). Igualmente, este capítulo contém uma seção apresentando o material básico necessário de teoria da probabilidade que é usado extensivamente no livro (Seção 1.2).

Finalmente, motivamos a abordagem rigorosa empregada em todo o livro e discutimos alguns de seus aspectos (Seção 1.4).

Dica de Ensino. Partes da Seção 1.4 podem ser mais adequadas para a última aula (i.e., como parte das conclusões) do que para a primeira (i.e., como parte da introdução). Isso se refere especificamente às Seções 1.4.2 e 1.4.3.

1.1 Criptografia: Tópicos Principais

Historicamente, o termo "criptografia" tem sido associado ao problema de projetar e analisar *esquemas de encriptação* (i.e., esquemas que provêem comunicação segura sobre meios inseguros de comunicação). Entretanto, desde os anos 1970, problemas tais como construir *assinaturas digitais* infalsificáveis e projetar *protocolos tolerantes a falha* também têm sido considerados como residindo dentro do domínio da criptografia. Na verdade, a criptografia pode ser vista como preocupada com o projeto de qualquer sistema que precise resistir a tentativas maliciosas de abusá-lo. Além disso, criptografia como redefinida aqui faz uso essencial de algumas ferramentas que necessitam ser tratadas em um livro sobre o assunto. Exemplos notáveis incluem funções unidirecionais, geradores pseudoaleatórios, e provas de conhecimento-zero. Nesta seção discutiremos brevemente esses termos.

Começamos mencionando que bastante do conteúdo deste livro repousa sobre a suposição de que funções unidirecionais existem. A definição de funções unidirecionais captura o tipo de dificuldade computacional que é inerente a nossa inteira abordagem à criptografia, uma abordagem que tenta capitalizar nas limitações computacionais de qualquer adversário do mundo real. Por conseguinte, se nada for difícil, então essa

abordagem falha. Entretanto, se, como se acredita largamente, não apenas problemas difíceis realmente existem mas também instâncias deles podem ser eficientemente geradas, então esses problemas difíceis podem ser "postos a funcionar." Por conseguinte, "notícias algoritmicamente ruins" (pelas quais problemas computacionais difíceis existem) implica em boas notícias para a criptografia. O Capítulo 2 é dedicado à definição e manipulação da dificuldade computacional na forma de funções unidirecionais.

1.1.1 Esquemas de Encriptação

O problema de prover *comunicação secreta através de meios inseguros* é o problema mais tradicional e básico da criptografia. O cenário consiste de duas partes se comunicando através de um canal que possivelmente pode ser grampeado por um adversário, chamado de *araponga*. As partes desejam trocar informação entre si, mas manter o araponga tão ignorante quanto possível a respeito do conteúdo dessa informação. Grosso modo, um esquema de encriptação é um protocolo permitindo que essas partes comunicarem *secretamente* uma com a outra. Tipicamente, o esquema de encriptação consiste de um par de algoritmos. Um algoritmo, chamado *encriptação* é aplicado pelo emissor (i.e., a parte enviando uma mensagem), enquanto que o outro algoritmo, chamado *decriptação* é aplicado pelo receptor. Portanto, para enviar uma mensagem, o emissor primeiro aplica o algoritmo de encriptação à mensagem e envia o resultado, chamado *texto-cifrado*, através do canal. Ao receber o texto-cifrado, a outra parte (i.e., o receptor) aplica-lhe o algoritmo de decriptação e recupera a mensagem original (chamada *texto-puro*).

Para que esse esquema possa prover comunicação secreta, as partes comunicantes (pelo menos o receptor) deve saber algo que não é conhecido pelo araponga. (Do contrário, o araponga poderia decifrar o texto-cifrado exatamente como realizado pelo receptor.) Esse conhecimento extra pode tomar a forma do algoritmo de decriptação propriamente dito ou alguns parâmetros e/ou entradas auxiliares usados pelo algoritmo de decriptação. Chamamos esse conhecimento extra *a chave de decriptação*. Note que, sem perda de generalidade, podemos assumir que o algoritmo de decriptação é conhecido ao araponga e que o algoritmo de decriptação necessita de duas entradas: um texto-cifrado e uma chave de decriptação. Enfatizamos que a existência de uma chave secreta, não conhecida pelo araponga, é meramente uma condição necessária para a comunicação secreta.

Avaliar a "segurança" de um esquema de encriptação é um negócio muito complicado. Uma tarefa preliminar é entender o que é "segurança" (i.e., definir apropriadamente o que se quer dizer por esse termo intuitivo). Duas abordagens à definição de segurança são conhecidas. A primeira abordagem ("clássica") é *teórica-de-informação*. Concerne a "informação" sobre o texto-puro que está "presente" no texto-cifrado. Grosso modo, se o texto-cifrado contém informação sobre o texto-puro, então o esquema de encriptação é considerado inseguro. Foi mostrado que tal alto nível (i.e., "perfeito") de segurança pode ser atingido apenas se a chave em uso é pelo menos tão longa quanto o comprimento *total* das mensagens enviadas via o esquema de encriptação. O fato de que a chave tem que ser mais longa que a informação trocada usando tal chave é realmente uma drástica limitação na aplicabilidade de tais esquemas de encriptação. Isso é especialmente verdadeiro quando quantidades *enormes* de informação necessitam ser secretamente comunicadas.

A segunda abordagem ("moderna"), tal qual seguida neste livro, é baseada em *complexidade computacional*. Essa abordagem é baseada no fato de que *não importa se o texto-cifrado contém ou não informação sobre o texto-puro*, porém, ao contrário, *se*

essa informação pode ou não ser eficientemente extraída. Em outras palavras, ao invés de perguntar se é ou não *possível* ao araponga extrair informação específica, perguntamos se é ou não *factível* ao araponga extrair essa informação. Acontece que a nova abordagem (i.e., "complexidade-computacional") oferece segurança mesmo se a chave é muito mais curta que o comprimento total das mensagens enviadas via o esquema de encriptação. Por exemplo, pode-se usar "geradores pseudoaleatórios" (discutidos adiante) que expandem chaves curtas para "pseudo-chaves" muito mais compridas, de modo que as últimas sejam tão seguras quanto "chaves reais" de comprimento comparável.

Adicionalmente, a abordagem da complexidade-computacional permite a introdução de conceitos e primitivas que não podem existir sob a abordagem teórica-da-informação. Um exemplo típico é o conceito de *esquemas de encriptação de chave-pública*. Note que na discussão precedente nos concentramos no algoritmo da decriptação e sua chave. Pode ser mostrado que o algoritmo da encriptação tem que obter, em adição à mensagem, uma entrada auxiliar que depende da chave de decriptação. Essa entrada auxiliar é chamada *chave de encriptação*. Esquemas tradicionais de encriptação, e em particular todos os esquemas de encriptação usados no milênio antes dos anos 1980s, operam com uma chave de encriptação igual à chave de decriptação. Portanto, o araponga nesses esquemas deve ser ignorante da chave de encriptação, e consequentemente o *problema da distribuição da chave* aparece (i.e., como duas partes desejando se comunicar através de um canal inseguro podem concordar a respeito de uma chave secreta de encriptação/decriptação).[1] A abordagem complexidade-computacional permite a introdução de esquemas de encriptação nos quais a chave de encriptação pode ser conhecida pelo araponga sem comprometer a segurança do esquema. Claramente, a chave de decriptação em tais esquemas é diferente da chave de encriptação, e além do mais é impraticável calcular a chave de decriptação a partir da chave de encriptação. Tais esquemas de encriptação, chamados *esquemas de chave-pública*, têm a vantagem de resolver trivialmente o problema da distribuição de chave, porque a chave de encriptação pode ser tornada pública.

No Capítulo 5, que aparecerá no segundo volume deste trabalho e será dedicado a esquemas de encriptação, discutiremos esquemas de chave-pública e chave-privada. Muita atenção é dedicada à definição da segurança de esquemas de encriptação. Finalmente, construções de esquemas seguros de encriptação baseados em várias suposições de intratabilidade são apresentados. Algumas das construções apresentadas são baseadas em geradores pseudoaleatórios, que são discutidos no Capítulo 3. Outras construções usam funções unidirecionais específicas tais como a função RSA e/ou a operação de elevar ao quadrado módulo um número composto.

1.1.2 Geradores Pseudoaleatórios

Acontece que geradores pseudoaleatórios desempenham um papel central na construção de esquemas de encriptação (e esquemas relacionados). Em particular, geradores pseudoaleatórios dão origem a construções simples de esquemas de encriptação de chave-privada, e essa observação é frequentemente usada na prática (usualmente implicitamente).

Embora o termo "geradores pseudoaleatórios" seja comumente usado *na prática*, tanto no contexto da criptografia quanto no contexto mais amplo dos procedimentos

[1] A solução tradicional é trocar a chave através de um canal alternativo que seja seguro, aliás "mais caro de usar," por exemplo, por uma escolta.

probabilísticos, ele é raramente associado a um significado preciso. Acreditamos que usar um termo sem enunciar claramente o que significa é perigoso em geral e mais particularmente em um negócio complicado como criptografia. Portanto, um tratamento preciso de geradores pseudoaleatórios é central para a criptografia.

Grosso modo, um gerador pseudoaleatório é um algoritmo determinístico que expande senhas curtas para seqüências de bits muito mais longas que *parecem* "aleatórias" (embora elas não o sejam). Em outras palavras, embora a saída de um gerador pseudoaleatório não seja realmente aleatória, é *impraticável* notar a diferença. Acontece que pseudoaleatoricidade e dificuldade computacional são ligadas de uma maneira ainda mais fundamental, pois geradores pseudoaleatórios podem ser construídos baseados em várias hipóteses de intratabilidade. Além do mais, o principal resultado nessa área assevera que geradores pseudoaleatórios existem se e somente se funções unidirecionais existem.

O Capítulo 3, dedicado a geradores pseudoaleatórios, começa com um tratamento do conceito de indistinguibilidade computacional. Geradores pseudoaleatórios são definidos a seguir e são construídos usando tipos especiais de funções unidirecionais (definidas no Capítulo 2). *Funções* pseudoaleatórias são também definidas e construídas. Essas últimas oferecem uma plêiade de aplicações adicionais.

1.1.3 Assinaturas Digitais

Uma noção que não existia no mundo pré-computadorizado é aquela da "assinatura digital". A necessidade de se discutir assinaturas digitais surgiu com a introdução de comunicação por computadores no meio dos negócios através da qual as partes precisam se comprometer com propostas e/ou declarações que elas fazem. Discussões sobre "assinaturas infalsificáveis" também tiveram lugar nos séculos precedentes, mas os objetos de discussão foram as assinaturas manuscritas, não as digitais, e a discussão não foi percebida como relacionada à criptografia.

As relações entre encriptação e métodos de assinatura tornaram-se possíveis com a "digitalização" de ambos e a introdução da abordagem baseada em complexidade computacional à segurança. Grosso modo, um *esquema para assinaturas infalsificáveis* requer

- que cada usuário seja capaz de *gerar eficientemente sua própria assinatura* em documentos de sua escolha,
- que cada usuário seja capaz de *verificar eficientemente* se uma dada cadeia é ou não uma assinatura de um outro usuário (específico) num documento específico, e
- que *ninguém seja capaz de produzir eficientemente as assinaturas de outros usuários* para documentos que aqueles usuários não assinaram.

Enfatizamos que a formulação de assinaturas digitais infalsificáveis também provê um enunciado claro dos ingredientes essenciais das assinaturas manuscritas. De fato, os ingredientes são a habilidade de cada um de assinar por si próprio, um procedimento universalmente acordado, e a crença (ou asserção) de que é impraticável (ou pelo menos difícil) falsificar assinaturas de uma maneira que poderia passar no procedimento de verificação. É difícil enunciar até que ponto assinaturas manuscritas cumprem essas exigências. Em contrapartida, nossa discussão sobre assinaturas digitais cumprem as exigências supracitadas. Além do mais, esquemas para assinaturas infalsificáveis

podem ser construídas usando as mesmas hipóteses computacionais que as usadas na construção de esquemas de encriptação (de chave privada).

No Capítulo 6, que aparecerá no segundo volume deste trabalho e será dedicado a esquemas de assinatura, muita atenção será dedicada à definição da segurança (i.e. infalsificabilidade) desses esquemas. A seguir, construções de esquemas infalsificáveis de assinatura baseados em várias hipóteses de intratabilidade serão apresentados. Adicionalmente, trataremos o problema relacionado da autenticação de mensagem.

Autenticação de Mensagem

Autenticação de mensagem é uma tarefa relacionada ao cenário considerado para esquemas de encriptação (i.e., comunicação através de um canal inseguro). Dessa vez, consideramos o caso de um adversário ativo que está monitorando o canal e pode alterar as mensagens enviadas através dele. As partes se comunicando através desse canal inseguro desejam autenticar as mensagens que elas enviam de modo que o receptor desejado possa distinguir uma mensagem original (enviado pelo emissor) de uma mensagem modificada (i.e., modificada pelo adversário). Grosso modo, um *esquema para autenticação de mensagem* requer

- que cada usuário das partes comunicantes seja capaz de *gerar eficientemente uma etiqueta de autenticação* para qualquer mensagem de sua escolha,
- que cada usuário das partes comunicantes seja capaz de *verificar eficientemente* uma etiqueta de autenticação para uma dada mensagem, e
- que *nenhum adversário externo* (i.e., uma entidade que não as partes comunicantes) *seja capaz de produzir eficientemente etiquetas de autenticação* para mensagens não enviadas pelas partes comunicantes.

Em um certo sentido, "autenticação de mensagem" é semelhante a uma assinatura digital. A diferença entre as duas é que no cenário de autenticação de mensagem não se exige que terceiros (que podem ser desonestos) sejam capazes de verificar a validade de etiquetas de autenticação produzidas pelos usuários designados, enquanto que no cenário de esquemas de assinatura é exigido que tais terceiros sejam capazes de verificar a validade de assinaturas produzidas por outros usuários. Portanto, assinaturas digitais provêem uma solução para o problema da autenticação de mensagem. Por outro lado, um esquema de autenticação de mensagem não necessariamente se constitui num esquema de assinatura digital.

Assinaturas Alargam o Escopo da Criptografia

Considerar o problema das assinaturas digitais como pertencente à criptografia alarga o escopo dessa área do problema específico da comunicação secreta para uma variedade de problemas relativos à limitação do "ganho" que pode ser obtido por comportamento "desonesto" de partidos (que são internos ou externos ao sistema). Especificamente:

- No problema da comunicação secreta (resolvido pelo uso de esquemas de encriptação), deseja-se reduzir, tanto quanto possível, a informação que um potencial araponga possa extrair da comunicação entre dois usuários designados. Nesse caso, o sistema designado consiste de duas partes comunicantes, e o araponga é considerado uma parte externa ("desonesta").

- No problema da autenticação de mensagem, busca-se proibir qualquer araponga (externo) de modificar a comunicação entre usuários (designados).
- No problema da assinatura, busca-se prover todos os usuários de um sistema de uma maneira de fazer enunciados auto-amarradores e de assegurar que um usuário não pode fazer enunciados que amarrasse outro usuário. Nesse caso, o sistema designado consiste do conjunto de todos os usuários, e um falsificador potencial é considerado como usuário interno embora desonesto.

Portanto, no sentido amplo, *criptografia concerne qualquer problema no qual se deseja limitar os efeitos de usuários desonestos*. Um tratamento geral de tais problemas é capturado pelo tratamento de protocolos "tolerantes a falha" (ou criptográficos).

1.1.4 Protocolos Tolerantes-a-Falha e Provas de Conhecimento-Zero

Uma discussão sobre esquemas de assinatura leva naturalmente a uma discussão de protocolos criptográficos, porque é uma preocupação natural perguntar sob que circunstâncias uma parte deveria fornecer sua assinatura a uma outra parte. Em particular, problemas como comprometimento mútuo simultâneo (e.g., assinatura de contrato) surgem naturalmente. Um outro tipo de problema, motivado pelo uso de comunicação por computador no meio de negócios, consiste de "implementação segura" de protocolos (e.g., implementar votação secreta e incorruptível).

Problemas de Simultaneidade

Um exemplo típico de um problema de simultaneidade é aquele da troca simultânea de segredos, do qual a assinatura de contrato é um caso especial. O cenário para uma troca simultânea de segredos consiste de duas partes interessadas, cada uma de posse de um "segredo". O objetivo é executar um protocolo tal que se ambas as partes o seguirem corretamente, então ao término cada uma estará de posse do segredo da outra, e em qualquer caso (mesmo se uma das partes engana) a primeira parte estará de posse do segredo da segunda parte se e somente se a segunda parte está de posse do segredo da primeira parte. Trocas de segredos perfeitamente simultâneas podem ser atingidas apenas se assumimos a existência de terceiros nos quais se confia até um certo ponto. Na verdade, troca simultânea de segredos pode ser facilmente atingida usando a participação ativa de uma terceira parte na qual se confia: Cada parte interessada envia seu segredo à terceira parte na qual se confia (usando um canal seguro). A terceira parte, ao receber ambos os segredos, envia o segredo da primeira parte à segunda parte e o segredo da segunda parte à primeira parte. Existem dois problemas com essa solução:

1. A solução requer a participação *ativa* de uma parte "externa" em todos os casos (i.e., também no caso em que ambas as partes interessadas são honestas). Notamos que outras soluções requerendo formas mais brandas de participação de partes externas realmente existem.
2. A solução requer a existência de um terceira entidade *totalmente acreditada*. Em algumas aplicações, tal entidade não existe. Não obstante, discutiremos a seguir o problema de se implementar uma parte acreditada através de um conjunto de usuários com uma maioria honesta (mesmo se a identidade dos usuários honestos não é conhecida).

Implementação Segura de Funcionalidades e Partes Interessadas Acreditadas

Um tipo diferente de problema de protocolo concerne a implementação segura de funcionalidades. Para ser mais específico, discutimos o problema de se avaliar uma função de entradas locais cada uma das quais é mantida por um usuário diferente. Um exemplo ilustrativo e motivante é *votação*, no qual a função é maioria, e a entrada local mantida pelo usuário A é um único bit representando o voto do usuário A (e.g., "pró" ou "contra"). Grosso modo, um protocolo para avaliar seguramente uma função específica deve satisfazer o seguinte:

- *Privacidade*: Nenhuma parte interessada pode "obter informação" da entrada de outras partes interessadas, além do que é deduzido a partir do valor da função.
- *Robustez*: Nenhuma parte interessada pode "influenciar" o valor da função, além da influência exercida pela seleção de sua própria entrada.

Exige-se algumas vezes que essas condições se verifiquem com respeito a "pequenas" (i.e., minoritárias) coalisões de partes interessadas (ao invés de uma só parte).

Claramente, se um dos usuários é conhecido como sendo totalmente digno de confiança então existe uma solução simples para o problema do cálculo seguro de qualquer função. Cada usuário simplesmente envia sua entrada à parte acreditada (usando um canal seguro) que, ao receber todas as entradas, computa a função, envia o resultado a todos os usuários, e apaga todos os cálculos intermediários (incluindo as entradas recebidas) de sua memória. Certamente, é irrealista assumir que uma parte interessada pode ser acreditada a tal ponto (e.g., que ela voluntariamente apagará o que "aprendeu"). Não obstante, o problema de implementar o cálculo seguro de funções se reduz ao problema de implementar uma parte acreditada. Acontece que uma parte acreditada pode ser implementada por um conjunto de usuários com uma maioria honesta (mesmo se a identidade dos usuários honestos não é conhecida). Isso é de fato um grande resultado nessa área, e muito do Capítulo 7, que será publicado no segundo volume, será dedicado a formulá-lo e a demonstrá-lo (bem como suas variantes).

Conhecimento-Zero como um Paradigma

Uma ferramenta importante na construção de protocolos criptográficos é o conceito de sistemas de prova de *conhecimento-zero* e o fato de que sistemas de prova de conhecimento-zero existem para todas as linguagens em $\mathcal{NP}$ (desde que funções unidirecionais existam). Grosso modo, uma prova de conhecimento-zero não produz nada além da validade da asserção. Provas de conhecimento-zero provêem uma ferramenta para "forçar" partes interessadas a seguir apropriadamente um dado protocolo.

Para ilustrar o papel de provas de conhecimento-zero, considere um cenário no qual uma parte interessada, chamada Alice, ao receber uma mensagem cifrada de Bob, deve enviar a Carol o bit menos significantivo da mensagem. Claramente, se Alice envia apenas o bit (menos significativo) (da mensagem), então não há como Carol saber se Alice não enganou. Alice poderia provar que ela não enganou revelando a Carol a mensagem inteira assim como a chave de decriptação, mas isso levaria a uma informação muito além do que tinha sido exigido. Uma idéia muito melhor é permitir que Alice aumente o bit que ela envia a Carol com uma prova de conhecimento-zero de que esse bit é de fato o bit menos significativo da mensagem. Enfatizamos que o

enunciado supracitado é do "tipo $\mathcal{NP}$" (pois a prova especificada anteriormente pode ser eficientemente verificada), e portanto a existência de provas de conhecimento-zero para enunciados $\mathcal{NP}$ implica que o enunciado supra pode ser provado sem revelar nada além de sua validade.

O foco do Capítulo 4, dedicado a provas de conhecimento-zero, está no resultado supracitado (i.e., a construção de provas de conhecimento-zero para qualquer enunciado $\mathcal{NP}$). Adicionalmente, consideraremos diversas variantes e aspectos da noção de provas de conhecimento-zero e seus efeitos sobre a aplicabilidade dessa noção.

1.2 Alguns Conhecimentos Básicos da Teoria da Probabilidade

Probabilidade desempenha um papel central em criptografia. Em particular, probabilidade é essencial de modo a permitir a discussão de informação ou falta da informação (i.e., segredo). Assumimos que o leitor esteja familiarizado com as noções básicas da teoria da probabilidade. Nesta seção, apresentamos meramente as notações probabilísticas que são usadas em todo este livro e três desigualdades probabilísticas úteis.

1.2.1 Convenções de Notação

Em todo este livro nos referiremos apenas a distribuições *discretas* de probabilidade. Tipicamente, o espaço de probabilidade consiste do conjunto de todas as cadeias de um certo comprimento ℓ, tomado com distribuição uniforme de probabilidade. Ou seja, o espaço amostral é o conjunto de todas as cadeias de ℓ-bits de comprimento, e a cada cadeia é associada uma medida de probabilidade $2^{-\ell}$. Tradicionalmente, funções do espaço amostral para os reais são chamadas *variáveis aleatórias*. Abusando da terminologia padrão, nos permitimos usar o termo *variável aleatória* também quando nos referimos a funções mapeando o espaço amostral no conjunto de cadeias binárias. Frequentemente não especificamos o espaço de probabilidade, senão falamos diretamente sobre variáveis aleatórias. Por exemplo, podemos dizer que X é uma variável aleatória tomando valores no conjunto de todas as cadeias, tal que $\mathsf{Pr}[X = 00] = \frac{1}{4}$ e $\mathsf{Pr}[X = 111] = \frac{3}{4}$. (Tal variável aleatória pode ser definida sobre o espaço amostral $\{0,1\}^2$, de modo que $X(11) = 00$ e $X(00) = X(01) = X(10) = 111$.) Na maioria dos casos o espaço de probabilidade consiste de todas as cadeias de um dado comprimento. Tipicamente, essas cadeias representam escolhas aleatórias feitas por algum processo aleatorizado (veja a próxima seção), e a variável aleatória é a saída do processo.

Como Ler Enunciados Probabilísticos. Todos os nossos enunciados probabilísticos se referem a funções de variáveis aleatórias que são previamente definidas. Tipicamente, escreveremos $\mathsf{Pr}[f(X) = 1]$, onde X é uma variável aleatória definida previamente (e f é uma função). Uma convenção importante é que *todas as ocorrências de um dado símbolo em um enunciado probabilístico se refere à mesma (única) variável aleatória*. Portanto, se $B(\cdot,\cdot)$ é uma expressão booleana dependendo de duas variáveis e X é uma variável aleatória, então $\mathsf{Pr}[B(X,X)]$ denota a probabilidade de que $B(x,x)$ se verifique quando x é escolhida com probabilidade $\mathsf{Pr}[X = x]$. A saber,

$$\mathsf{Pr}[B(X,X)] = \sum_x \mathsf{Pr}[X = x] \cdot \chi(B(x,x))$$

onde χ é uma função indicadora, tal que $\chi(B) = 1$ se o evento B se verifica, e igual a zero caso contrário. Por exemplo, para toda variável aleatória X, temos $\mathsf{Pr}[X = X] = 1$. Enfatizamos que se se deseja discutir a probabilidade de que $B(x, y)$ se verifica quando x e y são escolhidas independentemente com a mesma distribuição de probabilidade, então precisa-se definir *duas* variáveis aleatórias independentes, ambas com a mesma distribuição de probabilidade. Portanto, se X e Y são duas variáveis aleatórias independentes, então $\mathsf{Pr}[B(X, Y)]$ denota a probabilidade de que $B(x, y)$ se verifica quando o par (x, y) é escolhido com probabilidade $\mathsf{Pr}[X = x] \cdot \mathsf{Pr}[Y = y]$. A saber,

$$\mathsf{Pr}[B(X, y)] = \sum_{x,y} \mathsf{Pr}[X = x] \cdot \mathsf{Pr}[Y = y] \cdot \chi(B(x, y))$$

Por exemplo, para quaisquer duas variáveis aleatórias independentes, X e Y, temos $\mathsf{Pr}[X = Y] = 1$ somente se ambas X e Y são triviais (i.e., atribuem a massa inteira de probabilidade a uma única cadeia).

Variáveis Aleatórias Típicas. Em todo este livro, U_n denota uma variável aleatória uniformemente distribuída sobre o conjunto de cadeias de comprimento n. A saber, $\mathsf{Pr}[U_n = \alpha]$ é igual a 2^{-n} se $\alpha \in \{0, 1\}^n$, e é igual a zero caso contrário. Adicionalmente, usaremos ocasionalmente variáveis aleatórias (arbitrariamente) distribuídas sobre $\{0, 1\}^n$ ou $\{0, 1\}^{l(n)}$ para alguma função $l : \mathbb{N} \to \mathbb{N}$. Tais variáveis aleatórias são tipicamente representadas por X_n, Y_n, Z_n, etc. Enfatizamos que em alguns casos X_n é distribuída sobre $\{0, 1\}^n$, enquanto que em outros ela é distribuída sobre $\{0, 1\}^{l(n)}$, para alguma função $l(\cdot)$, que é tipicamente um polinômio. Um outro tipo de variável aleatória, a saída de um algoritmo aleatorizado sobre uma entrada fixa, é discutido na Seção 1.3.

1.2.2 Três Desigualdades

As seguintes desigualdades probabilísticas serão muito úteis no curso deste livro. Todas as desigualdades se referem a variáveis aleatórias às quais são atribuídos valores reais. A desigualdade mais básica é a *desigualdade de Markov*, que assevera que para variáveis aleatórias com valores máximos ou mínimos limitados, alguma relação deve existir entre o desvio de um valor da esperança da variável aleatória e a probabilidade de que a variável aleatória receba esse valor. Especificamente, fazendo $\mathsf{E}(X) \stackrel{\text{def}}{=} \sum_{\upsilon} \mathsf{Pr}[X = \upsilon] \cdot \upsilon$ denotar a esperança da variável aleatória X, temos o seguinte:

Desigualdade de Markov: *Seja X uma variável aleatória não-negativa e υ um número real. Então*

$$\mathsf{Pr}[X \geq \upsilon] \leq \frac{\mathsf{E}(X)}{\upsilon}$$

Equivalentemente, $\mathsf{Pr}[X \geq r \cdot \mathsf{E}(X)] \leq \frac{1}{r}$.

Demonstração:

$$\begin{aligned}\mathsf{E}(X) &= \sum_x \mathsf{Pr}[X = x] \cdot x \\ &\geq \sum_{x<v} \mathsf{Pr}[X = x] \cdot 0 + \sum_{x \geq v} \mathsf{Pr}[X = x] \cdot v \\ &= \mathsf{Pr}[X \geq v] \cdot v\end{aligned}$$

A afirmação segue. ■

A desigualdade de Markov é tipicamente usada em casos nos quais se conhece muito pouco sobre a distribuição da variável aleatória; é suficiente conhecer sua esperança e pelo menos um limitante no escopo de seus valores. Veja o Exercício 1.

Usando a desigualdade de Markov, obtém-se um limitante "possivelmente mais forte" para o desvio de uma variável aleatória de sua esperança. Esse limitante, chamado desigualdade de Chebyshev, é útil desde que se tenha conhecimento adicional a respeito da variável aleatória (especificamente, um bom limitante na sua variância). Para uma variável aleatória X de esperança finita, denotamos por $\mathsf{Var}(X) \stackrel{\text{def}}{=} \mathsf{E}[(X - \mathsf{E}(X))^2]$ a variância de X e observamos que $\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$.

Desigualdade de Chebyshev: *Seja X uma variável aleatória, e $\delta > 0$. Então*

$$\mathsf{Pr}[|X - \mathsf{E}(X)| \geq \delta] \leq \frac{\mathsf{Var}(X)}{\delta^2}$$

Demonstração: Definimos uma variável aleatória $Y \stackrel{\text{def}}{=} (X - \mathsf{E}(X))^2$ e aplicamos a desigualdade de Markov. Obtemos

$$\begin{aligned}\mathsf{Pr}[|X - \mathsf{E}(X)| \geq \delta] &= \mathsf{Pr}[(X - \mathsf{E}(X))^2 \geq \delta^2] \\ &\leq \frac{\mathsf{E}[(X - \mathsf{E}(X))^2]}{\delta^2}\end{aligned}$$

e a afirmação segue. ■

A desigualdade de Chebyshev é particularmente útil para a análise da probabilidade de erro de aproximação via amostragem repetida. Basta assumir que as amostragens são escolhidas de uma maneira independente duas-a-duas.

Corolário (Amostragem Independente Duas-a-Duas): *Sejam $X_1, X_2, \ldots, X_n$ variáveis aleatórias independentes duas-a-duas com a mesma esperança, representada por μ e a mesma variância, representada por σ^2. Então, para todo $\varepsilon > 0$,*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right] \leq \frac{\sigma^2}{\varepsilon^2 n}$$

As X_i's são chamadas *independentes duas-a-duas* se para todo $i \neq j$ e todos a e b, verifica-se que $\mathsf{Pr}[X_i = a \wedge X_j = b]$ é igual a $\mathsf{Pr}[X_i = a] \cdot \mathsf{Pr}[X_j = b]$.

Demonstração: Defina as variáveis aleatórias $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$. Note que as $\overline{X}_i$'s são independentes duas-a-duas e cada uma delas tem esperança zero. Aplicando a desigualdade de Chebyshev à variável aleatória definida pela soma $\sum_{i=1}^{n} \frac{X_i}{n}$, e usando a linearidade do operador de esperança, obtemos

$$\mathsf{Pr}\left[\left|\sum_{i=1}^{n}\frac{X_i}{n} - \mu\right| \geq \varepsilon\right] \leq \frac{\mathsf{Var}[\sum_{i=1}^{n}\frac{X_i}{n}]}{\varepsilon^2} \\ \frac{\mathsf{E}[(\sum_{i=1}^{n}\overline{X}_i)^2]}{\varepsilon^2 \cdot n^2}$$

Agora (novamente usando a linearidade de E)

$$\mathsf{E}\left[\left(\sum_{i=1}^{n}\overline{X}_i\right)^2\right] = \sum_{i=1}^{n}\mathsf{E}[\overline{X}_i^2] + \sum_{1\leq i\neq j\leq n}\mathsf{E}[\overline{X}_i\overline{X}_j]$$

Pela independência duas-a-duas das $\overline{X}_i$'s, obtemos $\mathsf{E}[\overline{X}_i\overline{X}_j] = \mathsf{E}[\overline{X}_i]\cdot\mathsf{E}[\overline{X}_j]$, e usando $\mathsf{E}[\overline{X}_i] = 0$, obtemos

$$\mathsf{E}\left[\left(\sum_{i=1}^{n}\overline{X}_i\right)^2\right] = n\cdot\sigma^2$$

O corolário segue. ■

Usando a amostragem independente duas-a-duas, a probabilidade de erro na aproximação é decrescente linearmente com o número de pontos de amostragem. Usando pontos de amostragem totalmente independentes, a probabilidade de erro na aproximação pode ser mostrada como decrescente exponencialmente com o número de pontos de amostragem. (As variáveis aleatórias $X_1, X_2, \ldots, X_n$ são ditas *totalmente independentes* se para toda seqüência $a_1, a_2, \ldots, a_n$ verifica-se que $\mathsf{Pr}[\wedge_{i=1}^{n} X_i = a_1]$ é igual a $\prod_{i=1}^{n}\mathsf{Pr}[X_i = a_i]$.) Limitantes de probabilidade suportando o enunciado acima são dados a seguir. O primeiro limitante, comumente referido como o *limitante de Chernoff*, concerne as variáveis aleatórias (i.e., variáveis aleatórias que recebem valores 0 ou 1).

Limitante de Chernoff: *Seja $p \leq \frac{1}{2}$, e suponha que $X_1, X_2, \ldots, X_n$ sejam variáveis aleatórias independentes 0-1, tal que $\mathsf{Pr}[X_i = 1] = p$ para cada i. Então para todo ε, $0 < \varepsilon \leq p(1-p)$, temos*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^{n}X_i}{n} - p\right| > \varepsilon\right] < 2\cdot e^{-\frac{\varepsilon^2}{2p(1-p)}\cdot n}$$

Usualmente aplicaremos o limitantee com uma constante $p \sim \frac{1}{2}$. Nesse caso, n amostras independentes levam a uma aproximação que desvia de ε da esperança com probabilidade δ que é exponencialmente decrescente com $\varepsilon^2 n$. Tal aproximação é chamada (ε, δ)-*aproximação* e pode ser alcançada usando $n = O(\varepsilon^{-2}\cdot\log(\frac{1}{\delta}))$ pontos de amostragem. É importante lembrar que o número suficiente de pontos de amostragem é polinomialmente relacionado com ε^{-1} e logaritmicamente relacionado com δ^{-1}. Portanto usando $\mathrm{poly}(n)$ amostras, a probabilidade de erro (i.e., δ) pode ser tornada desprezível (como uma função em n), mas a precisão da estimativa (i.e,, ε) pode

ser limitada por cima apenas por por qualquer fração polinomial fixa (mas não pode ser tornada desprezível).[2] Enfatizamos que a dependência do número de amostragens em relação a ε não é melhor que no caso da amostragem independente duas-a-duas; a vantagem das amostragens totalmente independentes reside apenas na dependência do número de amostragens em relação a δ.

Um limitantee mais geral, útil para aproximação da esperança de uma variável aleatória geral (não necessariamente 0-1), é dado como segue:

Desigualdade de Hoefding:[3] *Sejam $X_1, X_2, \ldots, X_n$ n variáveis aleatórias independentes com a mesma distribuição de probabilidade, cada uma variando sobre os intervalos (reais) $[a, b]$, e suponha que μ denote o valor esperado de cada uma dessas variáveis. Então, para todo $\varepsilon > 0$,*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - \mu\right| > \varepsilon\right] < 2 \cdot e^{-\frac{2\varepsilon^2}{(b-a)^2} \cdot n}$$

A desigualdade de Hoefding é útil para estimar o valor médio de uma função definida sobre um conjunto grande de valores, especialmente quando o erro de probabilidade desejado precisa ser desprezível. Pode ser aplicado desde que possamos eficientemente amostrar o conjunto e tenhamos um limitante nos possíveis valores (da função). Veja o Exercício 2.

1.3 O Modelo Computacional

Nossa abordagem à criptografia é fortemente baseada em complexidade computacional. Daí, algum conhecimento básico sobre complexidade computacional é requerido para nossa discussão da criptografia. Nesta seção, recordamos brevemente as definições das classes de complexidade $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$ e "não-uniforme $\mathcal{P}$" (i.e., $\mathcal{P}/\mathrm{poly}$) e o conceito de máquinas-oráculo. Adicionalmente, discutimos os tipos de suposições de intratabilidade usados em todo o restante deste livro.

1.3.1 $\mathcal{P}$, $\mathcal{NP}$, e $\mathcal{NP}$-Completude

Uma abordagem conservadora a dispositivos de computação associa computações eficientes à classe de complexidade $\mathcal{P}$. Saltando adiante, notamos que a abordagem assumida neste livro é uma abordagem mais liberal no sentido de que ela permite que os dispositivos de computação sejam aleatorizados.

Definição 1.3.1 (Classe de Complexidade $\mathcal{P}$): *Uma linguagem L é* **reconhecível em** tempo polinomial (determinístico) *se existe uma máquina de Turing determinística M e um polinômio $p(\cdot)$ tais que*

- *ao receber como entrada uma cadeia x, a máquina M pára após no máximo $p(|x|)$ passos, e*
- *$M(x) = 1$ se e somente se $x \in L$.*

[2] Aqui e no restante deste livro, denotamos por $\mathrm{poly}()$ algum polinômio fixo porém não especificado.

[3] Uma forma mais geral exige que as X_i's sejam independentes, mas não necessariamente idênticas, e usa $\mu \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \mathsf{E}(X_i)$. Veja [6, Apêndice A].

$\mathcal{P}$ é a classe de linguagens que podem ser reconhecidas em tempo polinomial (determinístico).

Igualmente, a classe de complexidade $\mathcal{NP}$ é associada a problemas computacionais tendo soluções que, uma vez dadas, podem ter sua validade eficientemente testadas. É costume se definir $\mathcal{NP}$ como a classe de linguagens que podem ser reconhecidas por uma máquina de Turing não-determinística. Uma formulação mais fundamental de $\mathcal{NP}$ é dada pela seguinte definição equivalente:

Definição 1.3.2 (Classe de Complexidade $\mathcal{NP}$): *Uma linguagem L pertence a $\mathcal{NP}$ se existe uma relação booleana $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ e um polinômio $p(\cdot)$ tais que R_L pode ser reconhecida em tempo polinomial (determinístico), e $x \in L$ se e somente se existe um y tal que $|y| \leq p(|x|)$ e $(x,y) \in R_L$. Tal y é chamado* **testemunha de pertinência** *de $x \in L$.*

Daí, $\mathcal{NP}$ consiste do conjunto de linguagens para as quais existem provas curtas de pertinência que podem ser eficientemente verificadas. Acredita-se largamente que $\mathcal{P} \neq \mathcal{NP}$, e a solução dessa questão é certamente o problema em aberto mais intrigante em ciência da computação. Se de fato $\mathcal{P} \neq \mathcal{NP}$, então existe uma linguagem $L \in \mathcal{NP}$ tal que todo algoritmo reconhecendo L terá um tempo de execução super-polinomial *no pior caso*. Certamente, todas as linguagens $\mathcal{NP}$-completas (definidas a seguir) terão complexidade super-polinomial *no pior caso*.

Definição 1.3.3 ($\mathcal{NP}$-Completude): *Uma linguagem é $\mathcal{NP}$-***completa** *se ela pertence a $\mathcal{NP}$ e toda linguagem em $\mathcal{NP}$ é polinomialmente redutível a ela. Uma linguagem L é* **polinomialmente redutível** *a uma linguagem L' se existe uma função computável em tempo polinomial f tal que $x \in L$ se e somente se $f(x) \in L'$.* Entre as linguagens sabidamente $\mathcal{NP}$-completas estão *Satisfatibilidade* (de fórmulas proposicionais), *Colorabilidade de Grafos*, e *Hamiltonicidade de Grafos*.

1.3.2 Tempo Polinomial Probabilístico

Algoritmos aleatorizados desempenham um papel central em criptografia. Eles são necessários para que se possa permitir que as partes interessadas legítimas gerem segredos e são portanto permitidos aos adversários. Assume-se que o leitor se sente familiarizado e confortável com tais algoritmos.

1.3.2.1 Algoritmos Aleatorizados: Um Exemplo

Para demonstrar a noção de um algoritmo aleatorizado, apresentamos um algoritmo aleatorizado simples para decidir se um dado grafo (não-direcionado) é ou não conexo (i.e., existe um caminho entre cada par de vértices no grafo). Comentamos que o algoritmo seguinte é interessante porque usa significativamente menos espaço que os algoritmos padrão (baseados em busca-em-largura ou busca-em-profundidade). Testar se um grafo é ou não conexo é facilmente reduzido a testar conectividade entre qualquer par de vértices dado.[4] Daí, nos concentramos na tarefa de determinar se dois vértices dados são ou não conexos em um dado grafo.

[4] A complexidade de espaço de tal redução é baixa; simplesmente precisamos de armazenar os nomes de dois vértices (sendo testados naquele momento). Aliás, a complexidade de tempo é de fato relativamente alta; precisamos invocar o teste de dois-vértices $\binom{n}{2}$ vezes, onde n é o número de vértices no grafo.

Algoritmo. Para a entrada um grafo $G = (V, E)$ e dois vértices, s e t, fazemos uma *caminhada aleatória* de comprimento $O(|V| \cdot |E|)$, começando no vértice s, e testamos a cada passo se o vértice t é encontrado ou não. Se o vértice t for encontrado, então o algoritmo aceitará; caso contrário, rejeitará. Por uma caminhada aleatória queremos dizer que a cada passo selecionamos uniformemente uma das arestas incidentes ao vértice atual e percorremos essa aresta até o outro ponto final.

Análise. Claramente, se s não for conectado a t no grafo G, então a probabilidade de que o algoritmo acima aceitará será zero. A parte mais difícil da análise é provar que se s é conectado a t no grafo G, então o algoritmo aceitará com probabilidade no mínimo $\frac{2}{3}$. (A demonstração é fica adiada para o Exercício 3.) Daí, de qualquer forma, o algoritmo errará com probabilidade no máximo $\frac{1}{3}$. A probabilidade de erro pode ser reduzida ainda mais invocando-se o algoritmo várias vezes (usando escolhas aleatórias novas a cada tentativa).

1.3.2.2 Algoritmos Aleatorizados: Dois Pontos de Vista

Algoritmos (máquinas) aleatorizados podem ser vistos de duas formas equivalentes. Uma maneira de se ver algoritmos aleatorizado é permitir que o algoritmo faça movimentos aleatórios (i.e., "cara-ou-coroa"). Formalmente, isso pode ser modelado por uma máquina de Turing na qual a função de transição mapeia pares da forma $(\langle\text{estado}\rangle, \langle\text{símbolo}\rangle)$ a duas possíveis triplas da forma $(\langle\text{estado}\rangle, \langle\text{símbolo}\rangle, \langle\text{direção}\rangle)$. O próximo passo para tal máquina é determinado por uma escolha aleatória de uma dessas triplas. A saber, para dar um passo, a máquina escolhe aleatoriamente (com probabilidade $\frac{1}{2}$ para cada possibilidade) ou a primeira tripla ou a segunda e então age conforme a escolha. Essas escolhas aleatórias são chamadas *cara-ou-coroa internos* da máquina. A saída de uma máquina probabilística M sobre uma entrada x não é uma cadeia mas sim uma variável aleatória que assume cadeias como possíveis valores. Essa variável aleatória, representada por $M(x)$, é induzida pelos cara-ou-coroa internos de M. Por $\mathsf{Pr}[M(x) = y]$ queremos dizer a probabilidade de que a máquina M sobre a entrada x retornará y como saída. O espaço de probabilidade é aquele de todas os possíveis resultados para os cara-ou-coroa internos tomados com distribuição uniforme de probabilidade.[5] Pelo fato de considerarmos apenas máquinas de tempo-polinomial, podemos assumir, sem perda de generalidade, que o número de cara-ou-coroa feitos por M sobre a entrada x é independente de seus resultados e é representado por $t_M(x)$. Representamos por $M_r(x)$ a saída de M sobre a entrada x quando r é o resultado de seus cara-ou-coroa internos. Então $\mathsf{Pr}[M(x) = y]$ é simplesmente a fração de $r \in \{0,1\}^{t_M(x)}$ para a qual $M_r(x) = y$. A saber,

$$\mathsf{Pr}[M(x) = y] = \frac{|\{r \in \{0,1\}^{t_M(x)} : M_r(x) = y\}|}{2^{t_M(x)}}$$

A segunda maneira de se olhar para algoritmos aleatorizados é ver o resultado dos cara-ou-coroa internos da máquina como uma entrada auxiliar. A saber, considera-

[5] Esta sentença é um pouco mais problemática do que parece. O caso simples é quando, sobre a entrada x, a máquina M sempre faz o mesmo número de cara-ou-coroa internos (independente de seus resultados). Em geral, o número de cara-ou-coroa pode depender do resultado dos cara-ou-coroa anteriores. Ainda assim, para todo r, a probabilidade de que o resultado da seqüência de cara-ou-coroa internos será r é igual a $2^{-|r|}$ se a máquina não termina quando a seqüência de resultados é um prefixo estrito de r, e é igual a zero caso contrário. Afortunadamente, pelo fato de considerarmos máquinas de tempo-polinomial, podemos modificar todas as máquinas de modo que elas satisfarão a estrutura do caso simples (e portanto evitaremos a complicação acima).

mos máquinas determinísticas com duas entradas. A primeira entrada faz o papel da "entrada real" (i.e., x) da primeira abordagem, enquanto que a segunda entrada faz o papel de um possível resultado para uma seqüência de cara-ou-coroa internos. Daí, a notação $M(x, r)$ corresponde à notação $M_r(x)$ usada anteriormente. Na segunda abordagem, consideramos a distribuição de probabilidade de $M(x, r)$ para qualquer x *fixo* e um $r \in \{0,1\}^{t_M(x)}$ uniformemente escolhido. Pictorialmente, aqui os cara-ou-coroa não são "internos" mas são fornecidos à máquina pelo dispositivo de cara-ou-coroa "externo".

Antes de seguir adiante, note-se que não devemos confundir o modelo fictício de máquinas "não-determinísticas" com o modelo de máquinas probabilísticas. O primeiro é um modelo irreal que é útil para se falar sobre problemas de busca cujas soluções podem ser eficientemente verificadas (e.g., a definição de $\mathcal{NP}$), enquanto que o último é um modelo realista de computação.

Em todo este livro, a menos que seja enunciado ao contrário, uma *máquina de Turing probabilística de tempo-polinomial* significa uma máquina probabilística que sempre (i.e., independentemente do resultado de seus cara-ou-coroa internos) pára após um número polinomial (no comprimento da entrada) de passos. Segue que o número de cara-ou-coroa para uma máquina de Turing probabilística de tempo-polinomial é limitado por um polinômio, representado por T_M, no comprimento da sua entrada. Finalmente, sem perda de generalidade, assumimos que sobre a entrada x a máquina sempre faz $T_M(|x|)$ cara-ou-coroa.

1.3.2.3 Associando Computações "Eficientes" a $\mathcal{BPP}$

A tese básica subjacente à nossa discussão é a associação de computações "eficientes" a computações probabilísticas de tempo-polinomial. Isto é, consideraremos como eficientes apenas os algoritmos aleatorizados (i.e., máquinas de Turing probabilísticas) para as quais o tempo de execução é limitado no comprimento da entrada.

Tese: *Computações eficientes correspondem a computações que podem ser realizadas por máquinas de Turing probabilísticas de tempo-polinomial.*

Uma classe de complexidade capturando essas computações é a classe, representada por $\mathcal{BPP}$, de linguagens reconhecíveis (com alta probabilidade) por máquinas de Turing probabilísticas de tempo-polinomial. A probabilidade se refere ao evento no qual *a máquina dá o veredito correto sobre a cadeia* x.

Definição 1.3.4 (Tempo Polinomial de Probabilidade-Limitada, $\mathcal{BPP}$): *Dizemos que* L *é* **reconhecida pela máquina de Turing probabilística de tempo-polinomial** M *se*

- *para toda* $x \in L$ *se verifica que* $\mathsf{Pr}[M(x) = 1] \geq \frac{2}{3}$*, e*
- *para toda* $x \notin L$ *se verifica que* $\mathsf{Pr}[M(x) = 0] \geq \frac{2}{3}$.

$\mathcal{BPP}$ é a classe de linguagens que podem ser reconhecidas por uma máquina de Turing probabilística de tempo-polinomial (i.e., um algoritmo aleatorizado).

O termo "probabilidade-limitada" indica que a probabilidade de sucesso é limitada para longe de $\frac{1}{2}$. Na verdade, na Definição 1.3.4, substituindo-se a constante $\frac{2}{3}$ por qualquer outra constante maior ou igual a $\frac{1}{2}$ não modificará a classe definida; veja o Exercício 4. Igualmente, a constante $\frac{2}{3}$ pode ser substituída por $1 - 2^{-|x|}$ e a classe

permanecerá inalterada; veja o Exercício 5. Concluimos que as linguagens em $\mathcal{BPP}$ podem ser reconhecidas por algoritmos de tempo-polinomial com uma probabilidade de erro desprezível. Usamos *desprezível* para descrever qualquer função que decresce mais rápido que o recíproco de qualquer polinômio.

Funções Desprezíveis

Definição 1.3.5 (Desprezível): *Chamamos uma função $\mu : \mathbb{N} \to \mathbb{R}$* **desprezível** *se para todo polinômio positivo $p(\cdot)$ existe um N tal que para todo $n > N$,*

$$\mu(n) > \frac{1}{p(n)}$$

Por exemplo, as funções $2^{-\sqrt{n}}$ e $n^{-\log_2 n}$ são desprezíveis (como funções em n). Funções desprezíveis permanecem assim quando multiplicadas por qualquer polinômio fixo. A saber, para toda função desprezível μ e qualquer polinômio p, a função $\mu'(n) \stackrel{\text{def}}{=} p(n) \cdot \mu(n)$ é desprezível. Segue que um evento que ocorre com probabilidade desprezível seria altamente improvável de ocorrer mesmo se repetíssemos o experimento uma quantidade polinomial de vezes.

Convenção. Na Definição 1.3.5 usamos a frase "existe um N tal que para todo $n > N$." No futuro usaremos a frase mais curta e menos entediante "para todo n suficientemente grande." Isso torna um quantificador $(\exists N)$ implícito, e isso é particularmente benéfico em enunciados que contêm vários quantificadores (mais essenciais).

1.3.3 Tempo Polinomial Não-Uniforme

Um modelo mais forte (e na verdade irreal) de computação eficiente é aquele do tempo polinomial não-uniforme. Esse modelo será usado apenas de maneira negativa, a saber, para dizer que mesmo tais máquinas não podem fazer algo (especificamente, mesmo se o adversário emprega uma tal máquina, não pode causar danos).

Uma *"máquina" de tempo-polinomial não-uniforme* é um par $(M, \overline{a})$, onde M é uma máquina de Turing de tempo-polinomial e $\overline{a} = a_1, a_2, \ldots$ é uma seqüência infinita de cadeias tal que $|a_n| = \text{poly}(n)$.[6] Para todo x, consideramos a computação de máquina M sobre o par de entrada $(x, a_{|x|})$. Intuitivamente, a_n pode ser pensado como um "conselho" extra suprido do exterior (juntamente com a entrada $x \in \{0,1\}^n$). Enfatizamos que a máquina M obtém o mesmo conselho (i.e., a_n) sobre todas as entradas de mesmo comprimento (i.e., n). Intuitivamente, o conselho a_n pode ser útil em alguns casos (i.e., para algumas computações sobre entradas de comprimento n), mas é improvável cifrar informação suficiente para que seja útil a todas as 2^n possíveis entradas.

Uma outra maneira de olhar para "máquinas" de tempo-polinomial não-uniforme é considerar uma seqüência infinita de máquinas de Turing $M_1, M_2, \ldots$ tal que tanto o comprimento da descrição de M_n quanto seu tempo de execução sobre entradas de comprimento n são limitados por polinômios em n (fixos para a seqüência inteira). A máquina M_n é usada apenas sobre entradas de comprimento n. Observe a correspondência entre as duas maneiras de olhar para tempo polinomial não-uniforme. O

[6] Recordemos que $\text{poly}()$ representa algum polinômio fixo (não especificado); isto é, dizemos que existe algum polinômio p tal que $|a_n| = p(n)$ para todo $n \in \mathbb{N}$.

par $(M, (a_1, a_2, \ldots))$ (da primeira definição) dá origem a uma seqüência infinita de máquinas $M_{a_1}, M_{a_2}, \ldots$, onde $M_{a_{|x|}} \stackrel{\text{def}}{=} M(x, a_{|x|})$. Por outro lado, uma seqüência $M_1, M_2, \ldots$ (como na segunda definição) dá origm a um par $(U, (\langle M_1 \rangle, \langle M_2 \rangle, \ldots))$, onde U é a máquina de Turing universal e $\langle M_n \rangle$ é a descrição da máquina M_n (i.e., $U(x, \langle M_{|x|} \rangle) = M_{|x|}(x)$).

Na primeira sentença desta Seção 1.3.3, tempo polinomial não-uniforme foi referido como um modelo mais forte que tempo polinomial probabilístico. Aquele enunciado é válido em muitos contextos (e.g., reconhecimento de linguagens, como visto adiante no Teorema 1.3.7). Em particular, será válido em todos os contextos que discutimos neste livro. Portanto temos o seguinte "meta-teorema" informal:

Meta-teorema: *O que quer que possa ser obtido por máquinas de tempo-polinomial probabilístico pode ser obtido por "máquinas" de tempo-polinomial não-uniforme.*

O meta-teorema obviamente está errado se pensarmos na tarefa de cara-ou-coroa. Portanto o teorema não deve ser entendido literalmente. É meramente uma indicação dos verdadeiros teoremas que podem ser demonstrados em casos razoáveis. Vamos considerar, por exemplo, o contexto de reconhecimento de linguagens.

Definição 1.3.6. *A classe de complexidade* tempo-polinomial não-uniforme *(representada por $\mathcal{P}/\mathrm{poly}$) é a classe de linguagens L que podem ser reconhecidas por uma seqüência não-uniforme de "máquinas" de tempo polinomial. A saber, $L \in \mathcal{P}/\mathrm{poly}$ se existe uma seqüência infinita de máquinas $M_1, M_2, \ldots$ satisfazendo o seguinte:*

1. *Existe um polinômio $p(\cdot)$ tal que para todo n, a descrição da máquina M_n tem comprimento limitado acima por $p(n)$.*
2. *Existe um polinômio $q(\cdot)$ tal que para todo n, o tempo de execução da máquina M_n sobre cada entrada de comprimento n é limitado acima por $q(n)$.*
3. *Para todo n e toda $x \in \{0,1\}^n$, a máquina M_n aceitará x se e somente se $x \in L$.*

Note que a não-uniformidade está implícita na ausência de um requisito relativo à construção das máquinas na seqüência. É requerido apenas que essas máquinas existam. Em contraste, se aumentarmos a Definição 1.3.6 requerendo a existência de um algoritmo de tempo-polinomial que sobre a entrada 1^n (n apresentado em unário) dê como saída a descrição de M_n, então obtemos uma maneira pesada de definir $\mathcal{P}$. Por outro lado, é óbvio que $\mathcal{P} \subseteq \mathcal{P}/\mathrm{poly}$ (na verdade, a inclusão estrita pode ser demonstrada considerando-se linguagens unárias não-recursivas). Além do mais:

Teorema 1.3.7: $\mathcal{BPP} \subseteq \mathcal{P}/\mathrm{poly}$.

Demonstração: Seja M uma máquina de Turing probabilística de tempo-polinomial que reconhece $L \in \mathcal{BPP}$. Seja $\chi_L(x) \stackrel{\text{def}}{=} 1$ se $x \in L$, e $\chi_L \stackrel{\text{def}}{=} 0$ caso contrário. Então, para toda $x \in \{0,1\}^n$,

$$\mathsf{Pr}[M(x) = \chi_L(x)] \geq \frac{2}{3}$$

Assuma, sem perda de generalidade, que sobre cada entrada de comprimento n, a máquina M usa o mesmo número, representado por $m = \mathrm{poly}(n)$, de cara-ou-coroa.

Seja $x \in \{0,1\}^n$. Claramente, podemos encontrar para cada $x \in \{0,1\}^n$ uma seqüência de cara-ou-coroa $r \in \{0,1\}^m$ tal que $M_r(x) = \chi_L(x)$ (na verdade, a maioria das seqüências r têm essa propriedade). Mas será que uma seqüência $r \in \{0,1\}^m$ combina com toda $x \in \{0,1\}^n$? Provavelmente não. (Dê um exemplo!) Não obstante, podemos encontrar uma seqüência $r \in \{0,1\}^n$ que combina com $\frac{2}{3}$ de todas as x's de comprimento n. Isso é feito através de um argumento da média (que assevera que se $\frac{2}{3}$ das r's são boas para cada x, então existe uma r que é boa para no mínimo $\frac{2}{3}$ das x's). Entretanto, isso não nos dá uma r que é boa para toda $x \in \{0,1\}^n$. Para obter tal r, temos que aplicar o argumento precedente sobre uma máquina M' com probabilidade de erro exponencialmente esvaecente. Tal máquina é garantida pelo Exercício 5. A saber, para toda $x \in \{0,1\}^*$,

$$\mathsf{Pr}[M'(x) = \chi_L(x)] > 1 - 2^{-|x|}$$

Aplicando o argumento da média, agora podemos concluir que existe uma $r \in \{0,1\}^m$, representada por r_n, que é boa para *mais que* uma fração $1 - 2^{-n}$ das x's em $\{0,1\}^n$. Segue que r_n é boa para todas as 2^n entradas de comprimento n. A máquina M' (vista como uma máquina determinística de duas-entradas), juntamente com a seqüência infinita $r_1, r_2, \ldots$ construída como dantes, demonstra que L pertence a $\mathcal{P}/\mathrm{poly}$. ■

Famílias Não-Uniformes de Circuitos. Uma maneira mais conveniente de ver tempo-polinomial não-uniforme, que é verdadeiramente a maneira usada neste livro, é através de famílias (não-uniformes) de circuitos booleanos de tamanho-polinomial. Um *circuito booleano* é um grafo direcionado acíclico com nós internos marcados por elementos de $\{\wedge, \vee, \neg\}$. Os nós sem arestas de entrada são chamados *nós de entrada*, e os nós sem arestas de saída são chamados *nós de saída*. Um nó marcado com $\neg$ pode ter apenas um filho. A computação no circuito começa com a colocação de bits de entrada nos nós de entrada (um bit por nó) e procede da seguinte maneira. Se os filhos de um nó (de grau-de-entrada d) marcado com $\wedge$ têm valores $v_1, v_2, \ldots, v_d$, então o nó recebe o valor $\wedge_{i=1}^{d} v_i$. Igualmente para os nós marcados com $\vee$ e $\neg$. A saída do circuito é lida a partir de seus nós de saída. O *tamanho* de um circuito é o número de suas arestas. Uma *família de circuitos de tamanho-polinomial* é uma seqüência infinita de circuitos booleanos $C_1, C_2, \ldots$ tal que para todo n, o circuito C_n tem n nós de entrada e tamanho $p(n)$, onde $p(\cdot)$ é um polinômio (fixo para a família inteira).

A computação de uma máquina de Turing M sobre entradas de comprimento n pode ser simulada por um único circuito (com n nós de entrada) tendo tamanho $O((|\langle M \rangle| + n + t(n))^2)$, onde $t(n)$ é um limitante no tempo de execução de M sobre entradas de comprimento n. Por conseguinte, uma seqüência não-uniforme de máquinas de tempo-polinomial pode ser simulada por uma família não-uniforme de circuitos de tamanho-polinomial. A recíproca também é verdadeira, pois máquinas com comprimentos de descrição polinomiais podem incorporar circuitos de tamanho-polinomial e simular suas computações em tempo polinomial. O bom da formulação através de circuitos é que não há necessidade de repetir o requisito polinomial duas vezes (uma vez para o tamanho e outra vez para o tempo) tal como na primeira formulação.

Convenção. Em nome da simplicidade, frequentemente tomamos a liberdade de considerar famílias de circuitos $\{C_n\}_{n \in \mathbb{N}}$, onde cada C_n tem $\mathrm{poly}(n)$ bits de entrada ao invés de n.

1.3.4 Suposições de Intratabilidade

Consideraremos como *intratável* aquelas tarefas que não podem ser realizadas por máquinas probabilísticas de tempo-polinomial. Entretanto, as tarefas adversárias nas quais estaremos interessados ("quebrar um esquema de encriptação," "falsificar assinaturas," etc.) podem ser realizadas por máquinas não-determinísticas de tempo-polinomial (porque as soluções, uma vez encontradas, podem ser facilmente testadas por validade). Por conseguinte, a abordagem computacional à criptografia (e, em particular, a maior parte do material neste livro) é *interessante* apenas se $\mathcal{NP}$ não está contido em $\mathcal{BPP}$ (que certamente implica em $\mathcal{P} \neq \mathcal{NP}$).[7] Usamos o termo "não-interessante" (ao invés de "não válido") porque todos os nossos enunciados serão da forma "se ⟨`suposição de intratabilidade`⟩ então ⟨`conseqüência útil`⟩." Tal enunciado permanece válido mesmo se $\mathcal{P} = \mathcal{NP}$ (ou se apenas ⟨`suposição de intratabilidade`⟩, que nunca é mais fraco que $\mathcal{P} \neq \mathcal{NP}$, for errado); mas nesse caso a implicação é de pouco interesse (porque qualquer coisa é conseqüência de uma falácia).

Na maioria das vezes em que enunciamos que "se ⟨`suposição de intratabilidade`⟩ então ⟨`conseqüência útil`⟩," será o caso que ⟨`conseqüência útil`⟩ implica ⟨`suposição de intratabilidade`⟩ ou alguma forma mais fraca dessa suposição, que por sua vez implica $\mathcal{NP} \backslash \mathcal{BPP} \neq \emptyset$. Por conseguinte, à luz do estado atual do conhecimento em teoria da complexidade, não podemos esperar asseverar ⟨`conseqüência útil`⟩ sem qualquer suposição de intratabilidade.

Em uns poucos casos, uma suposição relativa à limitação das máquinas probabilísticas polinomiais não bastarão, e usaremos no seu lugar uma suposição relativa às limitações das máquinas de tempo-polinomial não-uniforme. Tal suposição é obviamente mais forte. Mas também as conseqüências nesse caso serão mais fortes, pois elas também permitirão ser enunciadas em termos de complexidade não-uniforme. Entretanto, devido ao fato de que todas as nossas demonstrações são obtidas por reduções, uma implicação enunciada em termos de tempo polinomial probabilístico é mais forte (que uma enunciada em termos de tempo polinomial não-uniforme) e será preferida a menos que não seja conhecida ou seja demasiadamente complicada. Esse é o caso pois uma redução de tempo-polinomial (provando uma implicação na sua formalização probabilística) sempre implica uma redução de tempo-polinomial não-uniforme (provando o enunciado na sua formalização não-uniforme), mas a recíproca nem sempre é verdadeira.[8]

Finalmente, mencionamos que as suposições de intratabilidade relativas à complexidade do-pior-caso (e.g., $\mathcal{P} \neq \mathcal{NP}$) não bastarão. Os esquemas criptográficos que são garantidos como *difíceis de quebrar apenas no pior caso* são inúteis. Um esquema criptográfico deve ser inquebrável na "maioria dos casos" (i.e., "casos típicos"), o que implica que ele será *difícil de quebrar na média*. Segue que devido ao fato de que não temos condições de demonstrar que "intratabilidade no pior-caso" implica o análogo "intratabilidade para o caso médio" (tal resultado seria considerado um avanço em teoria da complexidade), nossa suposição de intratabilidade tem que ser concernente à complexidade do caso-médio.

---

[7] Chamamos a atenção para o fato de que não se sabe se $\mathcal{NP}$ contém $\mathcal{BPP}$. Essa é a razão pela qual enunciamos a conjectura mencionada acima como $\mathcal{NP}$ *não está contida em* $\mathcal{BPP}$, ao invés de $\mathcal{BPP} \neq \mathcal{NP}$. Igualmente, embora funções unidirecionais "suficientemente fortes" impliquem em $\mathcal{BPP} = \mathcal{P}$, não se sabe se essa igualdade se verifica incondicionalmente.

[8] Esse parágrafo pode ser melhor entendido no futuro, após se ver alguns exemplos concretos.

1.3.5 Máquinas-Oráculo

A utilidade original das máquinas-oráculo em teoria da complexidade era capturar as noções de redutibilidade. Neste livro (principalmente nos Capítulos 5 e 6) usamos máquinas-oráculo principalmente com um propósito completamente diferente – modelar um adversário que pode usar um cripto-sistema no curso de sua tentativa de quebrar o sistema. Outros usos de máquinas-oráculo são discutidos nas Seções 3.6 e 4.7.

Informalmente, uma máquina-oráculo é uma máquina que é aumentada de modo que ela pode fazer perguntas ao exterior. Consideramos o caso no qual essas questões (chamadas consultas) são respondidas consistentemente por alguma função $f : \{0,1\}^* \to \{0,1\}^*$, chamada oráculo. Isto é, se a máquina faz uma consulta q, então a resposta que ela obtém é $f(q)$. Nesse caso, dizemos que à máquina-oráculo é dado acesso ao oráculo f.

Definição 1.3.8 (Máquinas-Oráculo): *Uma* **máquina-oráculo** *(determinística/ probabilística) é uma máquina de Turing (determinística/probabilística) com uma fita adicional, chamada* **fita-oráculo***, e dois estados especiais, chamados* **invocação do oráculo** *e* **oráculo apareceu***.* **A computação da máquina-oráculo determinística** M **sobre a entrada** x **e com acesso ao oráculo** $f : \{0,1\}^* \to \{0,1\}^*$ *é definida pela* **relação configuração-sucessiva***. Para configurações com estados diferentes de* invocação do oráculo*, a próxima configuração é definida como de costume. Seja γ uma configuração na qual o estado é* invocação do oráculo *e o conteúdo da fita-oráculo é q. Então a configuração seguinte a γ é idêntica a γ, exceto que o estado é* oráculo apareceu*, e o conteúdo da fita-oráculo é $f(q)$. A cadeia q é chamada* **consulta de** M*, e $f(q)$ ćhamada* **resposta do oráculo***. A computação de uma máquina-oráculo probabilística é definida analogamente. A distribuição de saída da máquina-oráculo M, sobre a entrada x e com acesso ao oráculo f, é representada por $M^f(x)$.*

Enfatizamos que o tempo de execução de uma máquina-oráculo é o número de passos dados durante a sua computação e que a resposta do oráculo a cada consulta é obtida em um único passo.

1.4 Motivação para o Tratamento Rigoroso

Nesta seção tratamos três questões relacionadas:

1. a mera necessidade de um tratamento rigoroso da área,
2. o significado prático e/ou as conseqüências do tratamento rigoroso, e
3. as tendências "conservadoras" do tratamento.

Partes desta seção (correspondentes aos Itens 2 e 3) têm chances de se tornar mais claras após a leitura de quaisquer dos capítulos seguintes.

1.4.1 A Necessidade de um Tratamento Rigoroso

If the truth of a proposition does not follow
from the fact that it is self-evident to us,
then its self-evidence in no way justifies our belief in its truth.
Ludwig Wittgenstein*, Tractatus logico-philosophicus* (1921)

Criptografia concerne a construção de esquemas que serão robustos contra tentativas maliciosas de fazer esses esquemas desviar de sua funcionalidade prevista. Dada uma funcionalidade desejada, um criptógrafo deve desenhar um esquema que não apenas satisfará a funcionalidade desejada sob "operação normal" mas também manterá aquela funcionalidade em face de tentativas adversárias que serão concebidas após o criptógrafo ter completado o desenho. O fato de que um adversário conceberá seu ataque após o esquema ter sido especificado torna o desenho de tais esquemas muito difícil. Em particular, o adversário tentará realizar ações outras que não aquelas que o desenhista tenha visualizado. Por conseguinte, *a avaliação de esquemas criptográficos* tem que levar em conta um conjunto praticamente infinito de estratégias adversárias. É inútil fazer suposições sobre a *estratégia* específica que um adversário pode usar. As únicas suposições que podem ser justificadas dirão respeito às *capacidades* computacionais do adversário. Para resumir, uma avaliação de um esquema criptográfico é um estudo de um conjunto infinito de estratégias potenciais (que não são dadas explicitamente). Tal estudo altamente complexo não pode ser realizado apropriadamente sem um grande cuidado (i.e., rigor).

O desenho de sistemas criptográficos tem que ser baseado em *fundamentos sólidos*, enquanto que abordagens e heurísticas *ad hoc* são um caminho muito perigoso de seguir. Embora sempre inferior a uma solução rigorosamente analisada, uma heurística pode fazer sentido quando o desenhista tem uma idéia muito boa sobre o ambiente no qual um esquema vai operar. Mesmo assim um esquema criptográfico tem que operar em um ambiente maliciosamente selecionado que tipicamente transcenderá à visão do desenhista. Sob tais circunstâncias, heurísticas fazem pouco sentido (se é que fazem algum sentido).

Sobre Confiar em Intuições Infundadas. Acreditamos que *um* dos papéis da ciência é formular, examinar, e refinar nossa intuição sobre a realidade. Uma formulação rigorosa é necessária de modo a permitir um exame cuidadoso que pode levar ou à verificação e à justificação de nossa intuição ou à sua rejeição como falsa (ou como algo que verdadeiro em certos casos ou apenas sob certos refinamentos). Existem muitos casos nos quais nossa intuição inicial vem a ser correta, assim como muitos casos nos quais nossa intuição inicial vem a ser errada. Quanto mais entendemos a disciplina, melhor fica nossa intuição.

Neste estágio na história, seria muito presunçoso afirmar que temos boa intuição sobre a *natureza da computação eficiente*. Em particular, nem sequer sabemos as respostas a tais questões básicas como se $\mathcal{P}$ está ou não estritamente contida em $\mathcal{NP}$, imagine ter um entendimento do que faz um problema computacional ser difícil enquanto um problema aparentemente relacionado é fácil. Consequentemente, deveríamos ser extremamente cuidadosos ao fazer asserções sobre o que pode ou que não pode ser eficientemente computado. Infelizmente, *fazer asserções sobre o que pode ou que não pode ser eficientemente computado é exatamente tudo com o que tem a ver a criptografia*. Pior ainda, muitos dos problemas da criptografia têm descrições muito mais complexas e pesadas que as que são usualmente encontradas em teoria da complexidade. Para resumir, criptografia lida com noções computacionais muito complexas e atualmente tem que fazer isso sem ter um bom entendimento de noções computacionais muito mais simples. Daí, nossas atuais intuições sobre criptografia têm que ser consideradas altamente inseguras até que elas possam ser formalizadas e examinadas cuidadosamente. Em outras palavras, a necessidade geral de se formalizar e se examinar a intuição vem a ser ainda mais aguda em um campo altamente sensível como

criptografia que está intimamente ligada a questões que pouco entendemos.

O Registro da Trilha. Criptografia, como uma disciplina, é bem motivada. Consequentemente, temas criptográficos estão sendo discutidos por muitos pesquisadores, engenheiros, e leigos. Infelizmente, a maior parte dessas discussões são conduzidas sem definições precisas do tópico. Ao invés, é implicitamente assumido que os conceitos básicos de criptografia (e.g., encriptação segura) são auto-evidentes (porque eles são tão naturais) e que não há necessidade de se apresentar definições adequadas. A falácia dessa suposição é demonstrada pelo abandono de artigos (sem falar das discussões privadas) que derivam e/ou saltam a conclusões erradas com respeito a segurança. Na maioria dos casos essas conclusões erradas podem ser rastreadas até concepções errôneas implícitas com respeito a segurança que não poderiam ter escapado aos olhos dos autores se elas tivessem sido tornadas explícitas. Evitamos listar todos casos como esses aqui por várias razões óbvias. Não obstante, mencionaremos um exemplo bem conhecido.

Em torno de 1979, Ron Rivest afirmou que nenhum esquema de assinatura que fosse "demonstrado seguro assumindo a intratabilidade da demonstrado seguro assumindo a intratabilidade da fatoração" poderia resistir a um "ataque de mensagem escolhida." Seu argumento foi baseado numa suposição implícita (e injustificada) com respeito à natureza de uma "prova de segurança (que assume a intratabilidade da fatoração)." Consequentemente, por vários anos acreditava-se que se tinha que escolher entre ter um esquema de assinatura "provado como sendo infalsificável sob a intratabilidade da fatoração" e ter um esquema de assinatura que pudesse resistir a um "ataque de mensagem escolhida." Entretanto, em 1984, Goldwasser, Micali, e Rivest apontaram a falácia sobre a qual o argumento de 1979 de Rivest tinha sido baseado e além disso apresentaram esquemas de assinatura que poderiam resistir a um "ataque de mensagem escolhida," sob suposições gerais. Em particular, a intratabilidade da fatoração basta para provar que existe um esquema de assinatura que pode resistir a "falsificação," mesmo sob um "ataque de mensagem escolhida."

Para resumir, os conceitos básicos da criptografia são de fato muito naturais, mas eles *não* são auto-evidentes nem bem entendidos. Daí, nós ainda não entendemos esses conceitos suficientemente bem para sermos capazes de discutí-los *corretamente* sem usar definições precisas e justificar rigorosamente todos os enunciados realizados.

1.4.2 Conseqüências Práticas do Tratamento Rigoroso

Como costumeiro em teoria da complexidade, nosso tratamento é apresentado em termos de análise assintótica de algoritmos. (Na verdade, seria mais preciso usar o termo "análise funcional de tempo de execução.") Isso faz com que o tratamento seja menos carregado, mas *não* é essencial às idéias subjacentes. Em particular, a abordagem definicional adotada neste livro (e.g., as definições de funções unidirecionais, geradores pseudoaleatórios, provas de zero-conhecimento, esquemas de encriptação seguros, esquemas de assinatura infalsificáveis, e protocolos seguros) é baseada em paradigmas gerais que permanecem válidos em qualquer modelo computacional. Em particular, as definições, embora enunciadas de uma "maneira abstrata," se prestam a interpolações concretas. O mesmo se verifica com respeito aos resultados que tipicamente relacionam diversas tais definições. Para esclarecer o que foi dito acima, consideraremos, como um exemplo, o enunciado de um resultado genérico como apresentado neste livro.

Um resultado típico apresentado neste livro relaciona dois problemas computaci-

onais. O primeiro problema é um problema computacional simples que é suposto ser intratável (e.g., intratabilidade da fatoração), enquanto que o segundo problema consiste da "quebra" de uma implementação específica de uma primitiva criptográfica útil (e.g., um esquema de encriptação específico). O enunciado abstrato pode asseverar que se a fatoração inteira não pode ser realizada em tempo polinomial, então o esquema de encriptação é seguro no sentido de que ele não pode ser "quebrado" em tempo polinomial. Tipicamente, o enunciado é demonstrado através de uma redução em tempo-polinomial fixa da fatoração inteira para o problema de quebrar o esquema de encriptação. Logo, o que está sendo na verdade provado é que se se pode quebrar o esquema em tempo $T(n)$, onde n é o parâmetro de segurança (e.g., comprimento da chave), então se pode fatorar inteiros de comprimento m em tempo $T'(m) = f(m, T(g(m)))$, onde f e g são polinômios fixos que são no mínimo implícitos na prova. De modo a determinar a praticabilidade do resultado, deve-se primeiro determinar esses polinômios (f e g). Para a maioria dos resultados básicos apresentados neste livro, esses polinômios são razoavelmente pequenos, no sentido de que instanciar um esquema com um parâmetro de segurança razoável e fazer suposições de intratabilidade razoáveis (e.g., a respeito da fatoração) levarão a um esquema que é infactível de se quebrar na prática. (Nos casos excepcionais, dizemos isso explicitamente e vemos esses resultados como simplesmente afirmações da plausibilidade de relacionar as duas noções.) Na verdade distinguimos três tipos de resultados:

1. *Resultados de plausibilidade:* Aqui nos referimos aos resultados que são objetivados ao se estabelecer uma conecção entre as duas noções ou ao se prover uma maneira genérica de resolver uma classe de problemas.

   Um resultado do primeiro tipo diz que, em princípio, X (e.g., uma ferramenta específica) pode ser usada de modo a construir Y (e.g., uma utilidade útil), mas a construção específica fornecida na prova pode não ser prática. Mesmo assim, tal resultado pode ser útil na prática porque ele sugere que se pode ser capaz de usar implementações *específicas* de X de modo a prover uma construção prática de Y. No mínimo *minimorum*, tal resultado pode ser visto como um desafio aos pesquisadores para fornecer uma construção prática de Y usando X ou explicar por que uma construção prática não pode ser fornecida.

   Um resultado do segundo tipo diz que qualquer tarefa que pertença a alguma classe $\mathcal{C}$ é solúvel, mas a construção genérica fornecida na prova pode não ser prática. Mesmo assim, essa é uma valiosa peça de informação: Se temos um problema específico que reside na classe acima mencionada, então sabemos que o problema é solúvel em princípio. Entretanto, se precisamos construir um sistema real, então provavelmente deveríamos construir uma solução a partir do zero (ao invés de empregar o resultado genérico precedente).

   Para resumir, em ambos os casos o resultado de plausibilidade provê informação muito útil (mesmo se ele não leva a uma solução prática). Além do mais, é frequentemente o caso que *algumas* ferramentas desenvolvidas para se provar um resultado de plausibilidade podem ser úteis para resolver o problema específico em mãos. Esse é tipicamente o caso para o próximo tipo de resultados.

2. *Introdução de paradigmas e técnicas que podem ser aplicáveis na prática:* Aqui nos referimos aos resultados que são objetivados ao se introduzir uma nova noção, modelo, ferramenta, ou técnica. Tais resultados (e.g., técnicas) tipicamente são aplicáveis na prática, ou como apresentados no trabalho original, ou, após refinamentos adicionais, ou no mínimo como inspiração.

3. *Apresentação de esquemas que são adequados a aplicações práticas.*

Tipicamente, é um tanto fácil determinar a qual das categorias acima um resultado pertence. Infelizmente, a classificação não é sempre enunciada no artigo original; entretanto, tipicamente é evidente a partir da construção. Enfatizamos que todos os resultados dos quais estamos cientes (em particular, todos os resultados mencionados neste livro) vêm com uma construção explícita. Além do mais, a segurança da construção resultante é explicitamente relacionada à complexidade de certas tarefas intratáveis. Contrário a algunas crenças desinformadas, para cada um desses resultados existe uma tradução explícita das suposições de intratabilidade concretas (sobre as quais o esquema é baseado) para limitantes inferiores sobre a quantidade de trabalho necessária para violar a segurança do esquema resultante.[9] Enfatizamos que essa tradução pode ser invocada para qualquer valor do parâmetro de segurança. Fazendo isso determinar-se-á se uma construção específica é adequada para uma aplicação sob específicas suposições de intratabilidade razoáveis. Em muitos casos a resposta é na afirmativa, mas em geral isso de fato depende da construção específica, assim como do valor específico do parâmetro de segurança e do que é razoável assumir para esse valor (do parâmetro de segurança).

1.4.3 A Tendência a Ser Conservador

Quando chegar a capítulos nos quais primitivas criptográficas são definidas, o leitor pode notar que somos irrealisticamente "conservadores" em nossas definições de segurança. Em outras palavras, somos irrealisticamente liberais em nossa definição de insegurança. Tecnicamente falando, essa tendência não dá problemas, porque nossas primitivas que são seguras em um sentido muito forte certamente são também seguras no sentido razoável (mais restrito). Além do mais, somos capazes de implementar tais primitivas (fortemente seguras) usando suposições de intratabilidade razoáveis, e na maioria dos casos podemos mostrar que tais suposições são necessárias até para noções de segurança bem mais (e, na verdade, menos que minimamente) fracas. Ainda assim o leitor pode se perguntar por que escolhemos apresentar definições que parecem mais fortes que o que é exigido na prática.

A razão para nossa tendência a ser conservador quando definindo segurança é que é extremamente difícil capturar o que é *exatamente* exigido em uma aplicação prática específica. Além do mais, cada aplicação prática tem exigências diferentes, e é indesejável redesenhar o sistema para cada nova aplicação. Por conseguinte, precisamos na verdade tratar as preocupações de segurança de todas as aplicações futuras (que são desconhecidas para nós), não simplesmente as preocupações de segurança de algumas aplicações conhecidas. Parece impossível cobrir o que quer que seja exigido em todas as aplicações (ou mesmo em algum largo conjunto de aplicações) sem adotar nossa abordagem conservadora.[10] No que se segue, veremos como nossa abordagem conservadora leva a definições de segurança que podem cobrir todas as possíveis aplicações práticas.

[9] A única exceção ao enunciado anterior é a observação de Levin relativa à existência de uma *função unidirecional universal* (veja Seção 2.4.1).

[10] Pode-se até mesmo argumentar que parece impossível cobrir o que quer que seja exigido em uma aplicação razoável sem adotar nossa abordagem conservadora.

1.5 Miscelânea

Na Figura 1.1 confrontamos a "visão histórica" da criptografia (i.e., a visão da área em meados dos anos 1970) com a abordagem advogada neste texto.

1.5.1 Notas Históricas

O trabalho realizado durante os anos 1980 exerce um papel dominante em nossa exposição. Aquele trabalho, por sua vez, tinha sido tremendamente influenciado por trabalhos anteriores, mas aquelas primeiras influências não são enunciadas explicitamente nas notas históricas a capítulos subseqüentes. Nesta seção tracejaremos algumas daquelas influências. Em geral, aquelas influências tomaram a forma de estabelecer objetivos intuitivos, prover técnicas básicas, e sugerir soluções potenciais que mais tarde serviram como uma base para crítica construtiva (levando a abordagens robustas).

Criptografia Clássica. Respondendo a questão fundamental da criptografia clássica de forma tenebrosa (i.e., é *impossível* desenhar um código que não possa ser quebrado), Shannon [200] também sugeriu uma modificação na questão: Ao invés de perguntar se é ou não *possível* quebrar um código, dever-se-ia perguntar se é ou não *factível* quebrá-lo. Um código deveria ser considerado bom se ele não pode ser quebrado através de um investimento de trabalho que é em razoável proporção com o trabalho exigido das partes legais usando o código. De fato, essa é a aboradagem seguida pela criptografia moderna.

Novas Direções em Criptografia. A perspectiva para aplicações comerciais foi o gatilho para o início das investigações civis de esquemas de encriptação. O *block-cipher* DES [168], desenhado no início dos anos 1970, adotou o novo paradigma: É claramente *possível*, mas supostamente *infactível*, quebrá-lo. A seguir ao desafio de construir e analisar novos esquemas de encriptação (de chave-privada) vieram novas questões, tais quais como trocar chaves sobre um canal inseguro [159]. Novos conceitos foram inventados: *assinaturas digitais* (cf., Diffie & Hellman [63] e Rabin [186]), *criptossistemas de chave-pública* [63], e *funções unidirecionais* [63]. As primeiras implementações desses conceitos foram sugeridas por Merkle & Hellman [163], Rivest, Shamir, e Adleman [191], e Rabin [187].

Criptografia foi explicitamente relacionada à teoria da complexidade no final dos anos 1970 [38, 73, 147]: Ficou entendido que problemas relacionados à quebra de um mapeamento criptográfico 1-1 não poderia ser $\mathcal{NP}$-completo e, mais importante, que $\mathcal{NP}$-dificuldade da tarefa de quebra era evidência pobre para a segurança criptográfica. Técnicas tais como "verificação n-de-$2n$" [186] e compartilhamento secreto [196] foram introduzidas (e de fato foram usadas extensivamente em pesquisas subseqüentes).

Na Aurora de uma Nova Era. Investigações iniciais de protocolos criptográficos revelaram a inadequação de noções imprecisas de segurança, assim como as sutilezas envolvidas em se desenhar protocolos criptográficos. Em particular, problemas tais como *cara-ou-coroa sobre o telefone* [31], *troca de segredos* [30], *transferência oblívia* [188], (cf. [70]) foram formulados. Dúvidas (levantadas por Lipton) relativas à segurança do protocolo do poker-mental de [199] levaram à atual noção de encriptação segura, devida a Goldwasser e Micali [123], e a conceitos tais como indistinguibilidade computacional [123, 210]. Dúvidas (levantadas por Fischer) relativas ao protocolo de

transferência-oblívia de [188] levaram ao conceito de conhecimento-zero (sugerido por Goldwasser, Micali, e Rackoff [124], com versões iniciais datando de Março de 1982).

Uma abordagem formal à segurança de protocolos de criptografia foi sugerida em [65]. Aquela abordagem na verdade identificou uma *subclasse* de protocolos inseguros (i.e., aqueles que poderiam ser quebrados através de um tipo de ataque sintaticamente restrito). Além do mais, verificou-se que era demasiado difícil testar se um dado protocolo era ou não seguro [69]. Recordemos que, em contrapartida, nossa abordagem atual é construir protocolos seguros (juntamente com suas provas de segurança) e que essa abordagem é *completa* (no sentido de que ela nos permite resolver qualquer problema solúvel).

1.5.2 Sugestões para Leitura Adicional

O material básico relativo à teoria da probabilidade e complexidade computacional dado nas Seções 1.2 e 1.3, respectivamente, deveriam bastar para os propósitos deste livro. Ainda assim, um leitor sentindo-se desconfortável com qualquer dessas áreas pode querer consultar alguns livros-texto padrão sobre teoria da probabilidade (e.g., [79]) e complexidade computacional (e.g., [86; 202]). O leitor pode também se beneficiar da familiaridade com computação aleatorizada [167; 97, ap. B].

Problemas computacionais em teoria dos números têm fornecido candidatos populares a funções unidirecionais (e suposições de intratabilidade relacionadas). Entretanto, devido ao fato de que este livro focaliza em conceitos, ao invés de implementação específica de tais conceitos, aqueles candidatos populares não terão um papel principal na exposição. Consequentemente, uma base em teoria computacional dos números não é realmente necessária para este livro. Ainda assim, uma descrição breve de fatos relevantes aparece no Apêndice A anexo, e o leitor interessado é remetido a outro texto (e.g., [10]).

Este livro concentra-se nos conceitos básicos, definições, e resultados em criptografia. Como argumentado anteriormente, esses são de importância chave para a prática segura. Entretanto, prática necessita de muito mais que um arcabouço teórico seguro, enquanto que este livro não faz qualquer tentativa de prover nada além do tal arcabouço teórico. Para sugestões úteis relativas à prática (i.e., criptografia aplicada), o leitor pode consultar *criticamente* outros textos (e.g., [158]).

Nosso tratamento é apresentado em termos de análise assintótica de algoritmos. Além do mais, por simplicidade, enunciamos resultados em termos de robustez contra adversários de tempo-polinomial, ao invés de discutir adversários limitados-por-tempo em geral. Entretanto, como explicado na Seção 1.4, nenhuma dessas convenções é essencial, e os resultados poderiam ser enunciados em termos gerais, como feito na Seção 2.6 e em outros lugares (e.g., [155]). Nossa escolha de apresentar resultados em termos de robustez contra adversários de tempo-polinomial torna o enunciado dos resultados menos pesado, pois evita enunciar a complexidade exata da redução pela qual segurança é provada. Isso basta para enunciar resultados de plausibilidade, que é de fato nosso principal objetivo neste livro, porém quando utilizando esquemas na prática é necessário conhecer a complexidade exata da redução (pois isso determinará a segurança real do esquema concreto). Enfatizamos que tipicamente é fácil determinar a complexidade das reduções apresentadas neste livro, e em alguns casos também incluimos comentários referentes a esse aspecto. Mencionamos que a escolha alternativa, de apresentar resultados em termos de robustez contra adversários limitados-por-tempo em geral, é adotada no livro de Luby [155].

1.5.3 Problemas Em Aberto

Como mencionado anteriormente (e melhor formalizado no próximo capítulo), a maior parte do conteúdo deste livro se baseia na existência de funções unidirecionais. Atualmente, assumimos a existência de tais funções; é de se esperar que o novo século testemunhará uma prova dessa conjectura/suposição largamente acreditada. Mencionamos que a existência de funções unidirecionais implica que $\mathcal{NP}$ não está contida em $\mathcal{BPP}$, e por conseguinte estabeleceria $\mathcal{NP} \neq \mathcal{P}$ (que é o problema em aberto mais famoso em ciência da computação).

Mencionamos que não se sabe se $\mathcal{NP} \neq \mathcal{P}$ implica em quaisquer conseqüências práticas. Para que isso acontecesse, seria requerido que instâncias difíceis não apenas existissem mas também ocorressem um tanto frequentemente (com respeito a alguma distribuição fácil-de-amostrar). Para maiores discussões, veja o Capítulo 2.

1.5.4 Exercícios

Exercício 1: *Aplicações da Desigualdade de Markov:*

1. Seja X uma variável aleatória tal que $\mathsf{E}(X) = \mu$ e $X \leq 2\mu$. Dê um limitante superior para $\mathsf{Pr}[X \leq \frac{\mu}{2}]$.

2. Seja $0 < \varepsilon$ e $\delta < 1$, e suponha que Y seja uma variável aleatória tomando valores no intervalo $[0, 1]$ tal que $\mathsf{E}(Y) = \delta + \varepsilon$. Dê um limitante inferior para $\mathsf{Pr}[Y \geq \delta + \frac{\varepsilon}{2}]$.

   Diretriz: Em ambos os casos, pode-se definir variáveis aleatórias auxiliares e aplicar a desigualdade de Markov. Entretanto, é mais fácil simplesmente aplicar diretamente o raciocínio subjacente à prova da desigualdade de Markov.

Exercício 2: *Aplicações dos limitantes de Chernoff/Hoefding:* Seja $f : \{0,1\}^* \rightarrow [0, 1]$ uma função computável polinomial, e suponha que $F(n)$ represente o valor médio de f sobre $\{0,1\}^n$. A saber,

$$F(n) \stackrel{\text{def}}{=} \frac{\sum_{x \in \{0,1\}^n} f(x)}{2^n}$$

Seja $p(\cdot)$ um polinômio. Apresente um algoritmo probabilístico de tempo-polinomial que a partir da entrada 1^n dê como saída uma estimativa de $F(n)$, escrita $A(n)$, tal que

$$\mathsf{Pr}\left[|F(n) - A(n)| > \frac{1}{p(n)}\right] < 2^{-n}$$

Diretriz: O algoritmo seleciona aleatoriamente uma quantidade polinomial (quantos?) de pontos de amostragem $s_i \in \{0.1\}^n$. Esses pontos são selecionados independentemente e com distribuição uniforme de probabilidade. (Por que?) O algoritmo dá como saída o valor médio tomado sobre essa amostragem. Analise o desempenho do algoritmo usando a desigualdade de Hoefding. [Dica: Defina variáveis aleatórias X_i tais que $X_i = f(s_i)$.)

Exercício 3: *Análise do algoritmo de conectividade de grafos:* Considerando o algoritmo apresentado na Seção 1.3.2.1, mostre que se s é conectado a t no grafo G, então, com probabilidade no mínimo $\frac{2}{3}$, o vértice t será encontrado em uma caminhada aleatória começando em s.

Diretriz: Considere o componente conexo do vértice s, representado por $G' = \{V', E'\}$. Para qualquer aresta $\langle u, v\rangle \in E'$, seja $T_{u,v}$ uma variável aleatória representando o número de passos realizados em uma caminhada aleatória começando em u até que v seja encontrado. Primeiro, prove que $\mathsf{E}[T_{u,v}] \leq 2|E'|$. (Dica: Considere a "freqüência" com a qual essa aresta é percorrida em uma certa direção durante uma caminhada aleatória infinita, e note que essa freqüência é independente da identidade da aresta e da direção.) A seguir, supondo que $cover(G')$ seja o número esperado de passos em uma caminhada aleatória começando em S e terminando quando o último dos vértices de V' é encontrado, prove que $cover(G') \leq 4 \cdot |V'| \cdot |E'|$. (Dica: considere um passeio cíclico através de C passando por todos os vértices de G', e mostre que $cover(G') \leq \sum_{\langle u,v\rangle \in C} \mathsf{E}[T_{u,v}]$.) Conclua aplicando a desigualdade de Markov.

Exercício 4: *Definição equivalente de $\mathcal{BPP}$. Parte 1:* Prove que a Definição 1.3.4 é robusta quando $\frac{2}{3}$ é substituído por $\frac{1}{2} + \frac{1}{p(|x|)}$ para todo polinômio positivo $p(\cdot)$. A saber, mostre que $L \in \mathcal{BPP}$ se existe um polinômio $p(\cdot)$ e uma máquina probabilística de tempo-polinomial M tal que
• para todo $x \in L$ se verifica que $\mathsf{Pr}[M(x) = 1] \geq \frac{1}{2} + \frac{1}{p(|x|)}$, e
• para todo $x \notin L$ se verifica que $\mathsf{Pr}[M(x) = 0] \geq \frac{1}{2} + \frac{1}{p(|x|)}$.

Diretriz: Dada uma máquina probabilística de tempo-polinomial M satisfazendo a condição acima, construa uma máquina probabilística de tempo-polinomial M' da seguinte maneira. Para a entrada x, a máquina M' executa $O(p(|x|)^2)$ cópias de M, sobre a mesma entrada x, e regula pela maioria. Use a desigualdade de Chebyshev (veja Seção 1.2) para mostrar que M' está correta com probabilidade no mínimo $\frac{2}{3}$.

Exercício 5: *Definição equivalente de $\mathcal{BPP}$. Parte 2:* Prove que a Definição 1.3.4 é robusta quando $\frac{2}{3}$ é substituído por $1 - 2^{p(|x|)}$ para todo polinômio positivo $p(\cdot)$. A saber, mostre que para toda $L \in \mathcal{BPP}$ e todo polinômio $p(\cdot)$, existe uma máquina probabilística de tempo-polinomial M tal que
• para todo $x \in L$ se verifica que $\mathsf{Pr}[M(x) = 1] \geq 1 - 2^{p(|x|)}$, e
• para todo $x \notin L$ se verifica que $\mathsf{Pr}[M(x) = 0] \geq 1 - 2^{p(|x|)}$.

Diretriz: Semelhante ao Exercício 4, exceto que você tem que usar uma desigualdade probabilística mais forte (a saber, o limitante de Chernoff; veja Seção 1.2).

Capítulo 2

Dificuldade Computacional

Neste capítulo definimos e estudamos funções unidirecionais. Funções unidirecionais capturam nossa noção de dificuldade computacional "útil" e serve como uma base para a maioria dos resultados apresentados neste livro. Informalmente falando, uma função unidirecional é uma função que é fácil calcular mas é difícil inverter (em um sentido caso-médio). (Veja a ilustração na Figura 2.1.) Em particular, definimos funções unidirecionais fracas e fortes e provamos que a existência de funções unidirecionais fracas implica a existência das fortes. A prova dá um bom exemplo de um *argumento de redutibilidade*, que é um tipo forte de "redução" usado para estabelecer a maioria dos resultados na área. Além do mais, a prova dá um exemplo simples de um caso em que um enunciado computacional é muito mais difícil de provar que seu "análogo informação-teórico."

Adicionalmente, definimos predicados núcleo-duro e provamos que toda função unidirecional tem um predicado núcleo-duro. Predicados núcleos terão um papel importante em quase todos os capítulos subseqüentes (o capítulo sobre esquema de assinatura sendo a exceção).

Organização. Na Seção 2.1 motivamos a definição de funções unidirecionais argumentando informalmente que elas estão implícitas em várias primitivas criptográficas naturais. As definições básicas são dadas na Seção 2.2, e na Seção 2.3 mostramos que funções unidirecionais fracas podem ser usadas para construir as fortes. Uma construção mais eficiente (para certos casos restritos) é adiada para a Seção 2.6. Na Seção 2.4 olhamos para funções unidirecionais como coleções uniformes de funções finitas e consideramos várias propriedades adicionais que tais coleções podem ter. Na Seção 2.5 definimos predicados núcleo-duro e mostramos como construí-los a partir de funções unidirecionais.

Dica de Ensino. Como enunciado anteriormente, a prova de que a existência de funções unidirecionais fracas implica a existência das fortes (veja Seção 2.3) é instrutiva para o restante do material. Por conseguinte, se você escolher pular essa prova, incorpore uma discussão do *argumento de redutibilidade* na primeira ocasião em que você o utilize (e.g., quando mostrar como construir predicados núcleo-duro a partir de funções unidirecionais).

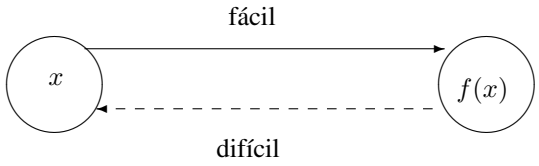

Figura 2.1: Funções unidirecionais: uma ilustração.

2.1 Funções Unidirecionais: Motivação

Como enunciado no capítulo introdutório, a criptografia moderna é baseada numa brecha entre algoritmos eficientes fornecidos para os usuários legítimos e a infactibilidade computacional de se abusar ou se quebrar esses algoritmos (via ações adversárias ilegítimas). Para ilustrar essa brecha, concentramos na tarefa criptográfica da comunicação de dados segura, a saber, esquemas de encriptação.

Nos esquemas de encriptação seguros, os usuários legítimos deveriam ser capazes de decifrar as mensagens usando alguma informação privada a eles disponível, e mesmo assim um adversário (não tendo essa informação privada) não deveria ser capaz de decifrar o texto-cifrado eficientemente (i.e., em tempo-polinomial probabilístico).[1] Por outro lado, uma máquina não-determinística pode rapidamente decifrar o texto-cifrado (e.g., adivinhando a informação privada). Daí, a existência de esquemas de encriptação seguros implica que existem tarefas (e.g., "quebrar" esquemas de encriptação) que podem ser realizadas por máquinas não-determinísticas de tempo-polinomial, e mesmo assim não podem ser realizadas por máquinas determinísticas (ou mesmo aleatorizadas) de tempo-polinomial. Em outras palavras, uma condição necessária para a existência de esquemas de encriptação seguros é que $\mathcal{NP}$ não pode estar contido em $\mathcal{BPP}$ (e por conseguinte $\mathcal{P} \neq \mathcal{NP}$).

Embora $\mathcal{P} \neq \mathcal{NP}$ seja uma condição necessária para a criptografia moderna, ela não é uma condição suficiente. Suponha que a quebra de algum esquema de encriptação seja $\mathcal{NP}$-completo. Então, $\mathcal{P} \neq \mathcal{NP}$ implica que esse esquema de encriptação é difícil de quebrar no pior caso, mas não é descartada a possibilidade de que o esquema de encriptação seja fácil de quebrar quase sempre. Na verdade, pode-se construir "esquemas de encriptação" para os quais o problema da quebra é $\mathcal{NP}$-completo e ainda assim existe um algoritmo de quebra eficiente que é bem sucedido 99no pior-caso é uma medida pobre de segurança. Segurança requer dificuldade na maioria dos casos, ou pelo menos "dificuldade no caso-médio." Uma condição necessária para a existência de esquemas de encriptação seguros é portanto a existência de linguagens em $\mathcal{NP}$ que são difíceis na média. Mencionamos que não se sabe se $\mathcal{P} \neq \mathcal{NP}$ implica ou não a existência de linguagens em $\mathcal{NP}$ que são difíceis na média.

Além do mais, a mera existência de problemas (em $\mathcal{NP}$) que são difíceis na média também não basta. Para que possamos ser capazes de usar tais problemas difíceis-na-média, devemos ser capazes de gerar instâncias difíceis juntamente com informação auxiliar que nos habilitará a resolver essas instâncias rapidamente. Do contrário, essas instâncias difíceis serão difíceis também para os usuários legítimos, e consequentemente os usuários legítimos não terão qualquer vantagem sobre o adversário. Daí, a existência de esquemas de encriptação seguros implica a existência de uma maneira eficiente (i.e., algoritmo probabilístico de tempo-polinomial) de gerar instâncias com entrada auxiliar correspondente tal que

[1] Essa "informação privada" é chamada de chave; veja o Capítulo 5.

1. seja fácil resolver essas instâncias dada a entrada auxiliar, mas

2. seja difícil, na média, resolver essas instâncias quando não se dá a entrada auxiliar.

O requisito acima é refletido na definição de funções unidirecionais (como apresentado na próxima seção). Informalmente falando, uma função unidirecional é uma função que é fácil computar mas difícil (na média) inverter. Por conseguinte, funções unidirecionais capturam a dificuldade de reverter o processo de gerar instâncias (e obter a entrada auxiliar a partir somente da instância), que é responsável pela discrepância entre os dois itens precedentes. (Para maiores discussões sobre esse relacionamento, veja o Exercício 1.)

Ao assumir que funções unidirecionais existem, estamos postulando a existência de processos eficientes (i.e., a computação da função na direção para-frente) que são difíceis de reverter. Note que tais processos do dia-a-dia nos são conhecidos em abundância (e.g., o acender de um fósforo). A suposição de que funções unidirecionais existem é portanto um análogo complexidade-teórico da experiência diária.

2.2 Funções Unidirecionais: Definições

Nesta seção, apresentamos várias definições de funções unidirecionais. A primeira versão, a partir desse ponto chamada de função unidirecional forte (ou apenas função unidirecional), é a mais popular. Apresentamos também funções unidirecionais, funções uniformemente unidirecionais, e candidatos plausíveis para tais funções.

2.2.1 Funções Unidirecionais Fortes

Informalmente falando, uma função unidirecional é uma função que é fácil computar mas difícil inverter. A primeira condição é um tanto clara: Dizer que uma função é fácil computar significa que existe um algoritmo de tempo-polinomial que sobre a entrada x dá como saída $f(x)$. A segunda condição requer mais elaboração. O que queremos dizer ao afirmar que *uma função f é difícil de inverter* é que todo algoritmo probabilístico polinomial tentando, sobre a entrada y, encontrar um inverso de y sob f pode ser bem sucedido apenas com probabilidade desprezível (em $|y|$), onde a probabilidade é tomada sobre as escolhas de y (como discutido mais adiante). Uma seqüência $\{s_n\}_{n\in\mathbb{N}}$ (respectivamente, uma função $\mu : \mathbb{N} \to \mathbb{R}$) é chamada *desprezível* em n se para todo polinômio positivo $p(\cdot)$ e todos os n's suficientemente grandes, se verifica que $s_n < \frac{1}{p(n)}$ (respectivamente, $\mu(n) < \frac{1}{p(n)}$). Uma maior discussão segue a definição.)

Definição 2.2.1 (Funções Unidirecionais Fortes): *Uma função $f : \{0,1\}^* \to \{0,1\}^*$ é chamada* **(fortemente) unidirecional** *se as duas condições seguintes se verificam:*

1. Fácil de computar: *Existe um algoritmo (determinístico) de tempo-polinomial A tal que sobre a entrada x o algoritmo A dá como saída $f(x)$ (i.e., $A(x) = f(x)$).*

2. Difícil de inverter: *Para todo algoritmo probabilístico de tempo-polinomial A', todo polinômio positivo $p(\cdot)$ e todos os n's suficientemente grandes,*

$$\mathsf{Pr}[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

Recordemos que U_n representa uma variável aleatória uniformemente distribuída sobre $\{0,1\}^n$. Daí, a probabilidade na segunda condição é tomada sobre todos os possíveis valores atribuídos a U_n e todos os cara-ou-coroa internos de A', com distribuição uniforme de probabilidade. Note que não se exige que A' dê como saída uma pré-image específica de $f(x)$; qualquer pré-imagem (i.e., elemento no conjunto $f^{-1}(f(x))$ servirá). (De fato, no caso em que f é 1-1, a cadeia x é a única pré-imagem de $f(x)$ sob f; mas em geral pode haver outras pré-imagens.)

A Entrada Auxiliar 1^n. Além de uma entrada no contradomínio de f, o algoritmo de inversão A' também recebe o comprimento da saída desejada (em notação unária). A principal razão para essa convenção é descartar a possibilidade de que uma função será considerada unidirecional meramente porque ela reduz drasticamente o comprimento de sua entrada, e que portanto o algoritmo de inversão simplesmente não tem tempo suficiente para imprimir a saída desejada (i.e., a pré-imagem correspondente). Considere, por exemplo, a função f_{len} definida por $f_{\text{len}}(x) = y$ tal que y é a representação binária do comprimento de x (i.e., $f_{\text{len}}(x) = |x|$). Como $|f_{\text{len}}(x)| = \log_2 |x|$, nenhum algoritmo pode inverter f_{len} sobre y em tempo polinomial em $|y|$; mesmo assim existe um algoritmo óbvio que inverte f_{len} sobre $y = f_{\text{len}}(x)$ em tempo polinomial em $|x|$ (e.g., por $|x| \mapsto 0^{|x|}$). Em geral, a entrada auxiliar $1^{|x|}$, fornecida em conjunto com a entrada $f(x)$, permite que o algoritmo de inversão execute em tempo polinomial no comprimento total da entrada principal e a saída desejada. Note que no caso especial de funções que preservam o comprimento f (i.e., $|f(x)| = |x|$ para todo x), essa entrada auxiliar é redundante. Mais geralmente, a entrada auxiliar é redundante se, dada apenas $f(x)$, pode-se gerar $1^{|x|}$ em tempo polinomial em $|x|$. (Veja o Exercício 4 e Seção 2.2.3.2.)

Discussão Adicional

Dificuldade para inverter é interpretada (pela definição supra) como um limitante superior na probabilidade de sucesso dos algoritmos de inversão eficientes. A probabilidade é medida com respeito a tanto as escolhas aleatórias do algoritmo de inversão quanto a distribuição da entrada (principal) para esse algoritmo (i.e., $f(x)$). A distribuição da entrada para o algoritmo de inversão é obtida aplicando-se f a uma $x \in \{0,1\}^n$ uniformemente selecionada. Se f induz a uma permutação sobre $\{0,1\}^n$, então a entrada para o algoritmo de inversão é uniformemente distribuída sobre $\{0,1\}^n$. Entretanto, no caso geral onde f não é necessariamente uma função um-para-um, a distribuição da entrada para o algoritmo de inversão pode diferir substancialmente da distribuição uniforme. Qualquer que seja o caso, é necessário que a probabilidade de sucesso, definida sobre o espaço de probabilidade supramencionado, seja desprezível (como uma função do comprimento de x). Para esclarecer ainda mais a condição imposta sobre a probabilidade de sucesso, consideramos dois algoritmos triviais.

Algoritmo de Adivinhação-Aleatória $\mathbf{A_1}$. Sobre a entrada $(y, 1^n)$, o algoritmo A_1 uniformemente seleciona e dá saída numa cadeia de comprimento n. Notamos que a probabilidade de sucesso de A_1 iguala a probabilidade de colisão da variável aleatória $f(U_n)$ (i.e., $\sum_y \mathsf{Pr}[f(U_n) = y]^2$). Ou seja, fazendo U'_n representar uma variável aleatória uniformemente distribuída sobre $\{0,1\}^n$ independentemente de U_n, temos

$$\begin{aligned}\mathsf{Pr}[A_1(f(U_n), 1^n) \in f^{-1}(f(U_n))] &= \mathsf{Pr}[f(U'_n) = f(U_n)] \\ &= \sum_y \mathsf{Pr}[f(U_n) = y]^2 \ \geq \ 2^{-n}\end{aligned}$$

onde a igualdade é devida ao fato de que, para x_i's não-negativos cuja soma é 1, o somatório $\sum_i x_i^2$ é minimizado quando todos os x_i's são iguais. Por conseguinte, a última desigualdade torna-se uma igualdade se e somente se f é uma função 1-1. Consequentemente:

1. Para qualquer função f, a probabilidade de sucesso do algoritmo trivial A_1 é estritamente positiva. Portanto, não se pode requerer que qualquer algoritmo eficiente *sempre* falhará na inversão de f.

2. Para qualquer função 1-1 f, a probabilidade de sucesso de A_1 na inversão de f é desprezível. Obviamente, isso não indica que f é unidirecional (mas sim que A_1 é trivial).

3. Se f é unidirecional, então a probabilidade de colisão da variável aleatória $f(U_n)$ é desprezível. Isso segue do fato de que A_1 se encontra no escopo da definição, e sua probabilidade de sucesso é igual à probabilidade de colisão.

Algoritmo de Saída-Fixa A_2. Um outro algoritmo trivial, representado por A_2, é aquele que computa uma função que é constante sobre todas as entradas de mesmo comprimento (e.g., $A_2(y, 1^n) = 0^n$). Para toda função f, temos

$$\begin{aligned}\mathsf{Pr}[A_2(f(U_n), 1^n) \in f^{-1}(f(U_n))] &= \mathsf{Pr}[f(0^n) = f(U_n)] \\ &= \frac{|f^{-1}(f(0^n))|}{2^n} \ \geq \ 2^{-n}\end{aligned}$$

com a igualdade se verificando no caso em que $f(0^n)$ tem uma única pré-imagem (i.e,, a própria 0^n) sob f. Novamente, observamos fatos análogos:

1. Para qualquer função f, a probabilidade de sucesso do algoritmo trivial A_2 é estritamente positiva.

2. Para qualquer função 1-1 f, a probabilidade de sucesso de A_2 na inversão de f é desprezível.

3. Se f é unidirecional, então a fração de x's em $\{0,1\}^n$ que são mapeados por f a $f(0^n)$ é desprezível.

Obviamente, a Definição 2.2.1 considera todos os algoritmos probabilísticos de tempo-polinomial, não simplesmente os triviais discutidos anteriormente. Em um certo sentido essa definição assevera que para funções unidirecionais, nenhum algoritmo probabilístico pode superar "significativamente" esses algoritmos triviais.

Probabilidade Desprezível

Umas poucas palavras em relação à noção de probabilidade desprezível se fazem necessárias. A definição e a discussão anteriores consideram a probabilidade de sucesso de um algoritmo como sendo *desprezível* se, como uma função do comprimento da entrada, a probabilidade de sucesso é limitada por cima por toda fração polinomial. Segue que repetindo o algoritmo uma quantidade de vezes polinomial (no comprimento da entrada) produz um novo algoritmo que também tem probabilidade de sucesso desprezível. Em outras palavras, eventos que ocorram com probabilidade desprezível (em n) permanecem desprezíveis mesmo se o experimento for repetido uma quantidade de vezes polinomial (em n). Logo, definir uma taxa de sucesso desprezível como "ocorrendo com probabilidade menor que qualquer fração polinomial" está naturalmente acoplado com definir computação factível como "computado em tempo polinomial."

Uma "negação forte" da noção de uma fração/probabilidade desprezível é a noção de uma fração/probabilidade detectável. Dizemos que uma função $\nu : \mathbb{N} \to \mathbb{R}$ é *detectável* se existe um polinômio $p(\cdot)$ tal que para todos os n's suficientemente grandes, verifica-se que $\nu(n) > \frac{1}{p(n)}$. Enfatizamos que funções podem não ser nem desprezíveis nem detectáveis.

2.2.2 Funções Unidirecionais Fracas

Funções unidirecionais, conforme definidas acima, são unidirecionais em um sentido muito forte. A saber, qualquer algoritmo inversor eficiente tem sucesso desprezível em invertê-las. Uma definição muito mais fraca, apresentada a seguir, requer apenas que todos os algoritmos inversores eficientes falhem com probabilidade detectável.

Definição 2.2.2 (Funções Unidirecionais Fracas): *Uma função $f : \{0,1\}^* \to \{0,1\}^*$ é chamada* **fracamente unidirecional** *se as duas condições seguintes se verificam:*

1. Fácil de computar: *Como na definição de uma função unidirecional.*

2. Levemente difícil de inverter: *Existe um polinômio $p(\cdot)$ tal que para todo algoritmo probabilístico de tempo-polinomial A' e todos os n's suficientemente grandes,*

$$\mathsf{Pr}[A'(f(U_n), 1^n) \notin f^{-1}(f(U_n))] > \frac{1}{p(n)}$$

Chamamos a atenção do leitor para a ordem dos quantificadores: Existe um *único* polinômio $p(\cdot)$ tal que $1/p(n)$ limita por baixo a probabilidade de falha de *todos* os algoritmos probabilísticos de tempo-polinomial tentando inverter f sobre $f(U_n)$.

Funções unidirecionais fracas falham em prover o tipo de dificuldade aludida nas discussões motivadoras anteriores. Mesmo assim, como veremos adiante, elas podem ser convertidas em funções unidirecionais fortes, que por sua vez realmente provêem tal dificuldade.

2.2.3 Duas Convenções de Comprimento Úteis

No que se segue será conveniente usar as duas seguintes convenções relativas aos *comprimentos* das pré-imagens e imagens de funções unidirecionais. Na presente seção justificamos o uso dessas convenções.

2.2.3.1 Funções Definidas Apenas para Alguns Comprimentos

Em muitos casos é mais conveniente considerar funções unidirecionais com domínios parciais ao conjunto de todas as cadeias. Em particular, isso facilita a introdução de alguma estrutura no domínio da função. Um caso particularmente importante, usado em todo o restante desta seção, é aquele de funções com domínio $\bigcup_{n\in\mathbb{N}}\{0,1\}^{l(n)}$, onde $l(\cdot)$ é algum polinômio. Fornecemos um tratamento mais geral desse caso.

Seja $I \subseteq \mathbb{N}$, e represente por $s_I(n)$ o sucessor de n com respeito a I; a saber, $s_I(n)$ é o menor inteiro que é maior que n e que pertence ao conjunto I (i.e., $s_I(n) \stackrel{\text{def}}{=} \min\{i \in I : i > n\}$). Um conjunto $I \subseteq \mathbb{N}$ é chamado de *enumerável-em-tempo-polinomial* se existe um algoritmo que sobre a entrada n pára em $\mathrm{poly}(n)$ passos e dá como saída $1^{s_I(n)}$. (A saída unária força s_I a ser limitado polinomialmente; i.e., $s_I(n) \leq \mathrm{poly}(n)$.) Seja I um conjunto enumerável-em-tempo-polinomial e f uma função com domínio $\bigcup_{n\in\mathbb{N}}\{0,1\}^n$. Chamamos f de fortemente (respectivamente, fracamente) *unidirecional sobre comprimentos em* l se f é computável-em-tempo-polinomial e é difícil invertê-la sobre n's em I. Por exemplo, a condição de dificuldade para funções que são fortemente unidirecionais sobre comprimentos em I é enunciada como segue:

> *Para todo algoritmo probabilístico de tempo-polinomial A', todo polinômio $p(\cdot)$ e todos os n's suficientemente grandes em I,*
>
> $$\mathsf{Pr}[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

Funções unidirecionais comuns, como definidas nas subseções anteriores, podem ser vistas como sendo unidirecionais sobre comprimentos em $\mathbb{N}$.

Funções unidirecionais sobre comprimentos em qualquer conjunto enumerável-em-tempo-polinomial podem ser facilmente transformadas em funções unidirecionais comuns (i.e., definidas sobre todo o conjunto $\{0,1\}^*$). Em particular, para qualquer função f com domínio $\bigcup_{n\in\mathbb{N}}\{0,1\}^n$, podemos construir uma função $g : \{0,1\}^* \to \{0,1\}^*$ fazendo

$$g(x) \stackrel{\text{def}}{=} f(x') \tag{2.1}$$

onde x' é o prefixo mais longo de x com comprimento em I. No caso da função f ser tal que preserve o comprimento (i.e., $|f(x)| = |x|$ para toda x), podemos preservar essa propriedade modificando a construção para obter uma função que preserva o comprimento $g' : \{0,1\}^* \to \{0,1\}^*$ tal que

$$g'(x) \stackrel{\text{def}}{=} f(x')x'' \tag{2.2}$$

onde $x = x'x''$, e x' é o prefixo mais longo de x com comprimento em I.

Proposição 2.2.3: *Seja I um conjunto enumerável-em-tempo-polinomial, e suponha que f seja fortemente (respectivamente, fracamente) unidirecional sobre comprimentos em I. Então g e g' (como definidas em* Eq. (2.1) *e* Eq. (2.2)*, respectivamente) são fortemente (respectivamente, fracamente) unidirecionais (no sentido comum).*

Embora a validade da proposição acima seja muito atraente, rogamos ao leitor a não pular a prova seguinte. A prova, que é na verdade bastante simples, usa (pela primeira

vez neste livro) um argumento que é usado extensivamente no que se segue. O argumento usado para provar a propriedade de dificuldade-de-inverter da função g (respectivamente, g') procede assumindo, em direção à contradição, que g (respectivamente, g') pode ser eficientemente invertida com probabilidade de sucesso não-permissível. A contradição é derivada deduzindo-se que f pode ser eficientemente invertida com probabilidade de sucesso não-permissível. Em outras palavras, inverter f é "reduzido" a inverter g (respectivamente, g'). O termo "redução" é aqui usado em um sentido mais-forte-que-o-padrão. Aqui uma redução precisa preservar a probabilidade de sucesso dos algoritmos. Esse tipo de argumento é chamado de um *argumento de redutibilidade*.

Prova: Primeiro provamos que g e g' podem ser computadas em tempo polinomial. Para esse fim usamos o fato de que I é um conjunto enumerável-em-tempo-polinomial, o que implica que podemos decidir pertinência em I em tempo polinomial (e.g., observando que $m \in I$ se e somente se $s_I(m-1) = m$). Segue que sobre a entrada x pode-se encontrar em tempo polinomial o maior $m \leq |x|$ que satisfaz $m \in I$. Computar $g(x)$ significa encontrar esse m e aplicar a função f ao prefixo de m-bits de x. Igualmente para g'.

A seguir provamos que g mantém a propriedade de dificuldade-de-inverter de f. Uma prova similar estabelece a propriedade de dificuldade-de-inverter de g'. Em nome da brevidade, apresentamos aqui apenas a prova para o caso em que f é fortemente unidirecional. A prova para o caso em que f é fracamente unidirecional é análoga.

A prova procede por contradição. Assumimos, contrário à afirmação (da proposição), que existe um algoritmo eficiente que inverte g com probabilidade de sucesso que não é desprezível. Usamos esse algoritmo inversor (para g) para construir um algoritmo eficiente que inverte f com probabilidade de sucesso que não é desprezível, portanto derivando uma contradição (à hipótese da proposição). Em outras palavras, mostramos que inverter f (com probabilidade de sucesso não-permissível) é eficientemente redutível a inverter g (com probabilidade de sucesso não-permissível) e portanto concluimos que essa última não é factível. A redução é baseada na observação de que inverter g sobre imagens de comprimentos arbitrários resulta em inverter g também sobre imagens de comprimentos em I, e que sobre tais comprimentos g colide com f.

Intuitivamente, qualquer algoritmo invertendo g pode ser usado para inverter f como segue. Sobre a entrada $(y, 1^n)$, onde y supostamente pertence à imagem de $f(U_n) = g(U_m)$ para qualquer $m \in \{n, \ldots, s_I(n) - 1\}$, podemos invocar o inversor de g sobre a entrada $(y, 1^n)$ e dar como saída o prefixo mais longo com comprimento em I da cadeia que o inversor de g retorna (e.g., se o inversor de g retorna uma cadeia de m-bits, então damos como saída seu prefixo de n-bits). Por conseguinte, nossa probabilidade de sucesso em inverter f sobre $f(U_n)$ é igual à probabilidade de sucesso do inversor de g sobre $g(U_m)$. A questão é qual $m \in \{n, \ldots, s_I(n)-1\}$ devemos usar, e a resposta é tentar todos (capitalizando no fato de que $s_I(n) = \mathrm{poly}(n)$). Note que os inteiros são particionados em intervalos da forma $\{n, \ldots, s_I(n)\}$, cada um associado com um único $n \in I$. Por conseguinte, a probabilidade de sucesso de qualquer inversor de g sobre uma quantidade infinita de comprimentos $m \in \mathbb{N}$ traduz para a probabilidade de sucesso de nosso inversor de f sobre uma quantidade infinita de comprimentos $n \in I$. Detalhes seguem.

Dado um algoritmo B' para inverter g, construimos um algoritmo A' para inverter f tal que A' tem complexidade e probabilidade de sucesso relacionadas àquelas de B'. (Por simplicidade, assumiremos que $B'(y, 1^m) \in \{0,1\}^m$ se verifica para toda

$y \in \{0,1\}^*$ e $m \in \mathbb{N}$; essa suposição é imaterial, e mais adiante comentamos sobre esse aspecto em duas notas de pé-de-página.) O algoritmo A' usa o algoritmo B' como uma subrotina e procede como segue. Sobre a entrada y e 1^n (supostamente y pertence ao contradomínio de $f(U_n)$, e $n \in I$), o algoritmo A' procede como segue:

1. Computa $s_I(n)$ e faz $k \stackrel{\text{def}}{=} s_I(n) - n - 1 \geq 0$. Por conseguinte, para todo $i = 1, \ldots, k$, temos que $n + i \notin I$.)

2. Para $i = 0, 1, \ldots, k$, o algoritmo A' invoca o algoritmo B' sobre a entrada $(y, 1^{n+i})$, obtendo $z_i \leftarrow B'(y, 1^{n+i})$; se $g(z_i) = y$, então A' dá como saída o prefixo de n-bits[2] de z_i.

Note que para todo $x' \in \{0,1\}^n$ e $|x''| \leq k$, temos que $g(x'x'') = f(x')$, e portanto se $g(x'x'') = y$, então $f(x') = y$, o que estabelece a corretude da saída de A'. Usando $s_I(n) = \text{poly}(n)$ e o fato de que $s_I(n)$ é computável em tempo polinomial, segue que se B' é um algoritmo probabilístico de tempo-polinomial, então A' também o é. A seguir analisamos a probabilidade de sucesso de A' (mostrando que se B' inverte g com probabilidade de sucesso não-permissível, então A' inverte f com probabilidade de sucesso não-permissível.

Suponha que B' inverte g sobre $(g(U_m, 1^m)$ com probabilidade $\varepsilon(m)$. Então existe um n tal que $m \in \{n, \ldots, \text{poly}(n)\}$ e tal que $A'(f(U_n), 1^n)$ invoca B' sobre a entrada $(f(U_n), 1^m) = (g(U_m), 1^m)$. Segue que $A'(f(U_n), 1^n)$ inverte f com probabilidade pelo menos $\varepsilon(m) = \varepsilon(\text{poly}(n))$. Por conseguinte, $A'(f(U_n), 1^n)$ herda o sucesso de $B'(g(U_m), 1^m)$. Uma análise tediosa (que pode ser pulada) segue.[3]

> Suponha, contrário à nossa afirmação, que g não é fortemente unidirecional, e seja B' um algoritmo demonstrando essa hipótese de contradição. A saber, existe um polinômio $p(\cdot)$ tal que para uma quantidade infinita de m's a probabilidade de que B' inverta g sobre $g(U_m)$ é pelo menos $\frac{1}{p(m)}$. Vamos representar o conjunto desses m's por M. Defina uma função $\ell_I : \mathbb{N} \to I$ tal que $\ell_I(m)$ é o maior limite inferior de m em I (i.e., $\ell_I(m) \stackrel{\text{def}}{=} \max\{i \in I : i \leq m\}$). Claramente, $m \geq \ell_I(m)$ e $m \leq s_I(\ell_I(m)) - 1$ para todo m. As duas seguintes afirmações relacionam a probabilidade de sucesso do algoritmo A' com aquela do algoritmo B'.
>
> **Afirmação 2.2.3.1:** Seja m um inteiro e $n = \ell_I(m)$. Então
>
> $$\mathsf{Pr}[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] \geq \mathsf{Pr}[B'(g(U_m), 1^m) \in g^{-1}(g(U_m))]$$
>
> (A saber, a probabilidade de sucesso do algoritmo A' sobre $f(U_{\ell_I(m)})$ é limitada por baixo pela probabilidade de sucesso do algoritmo B' sobre $g(U_m)$.)
>
> **Prova:** Pela construção de A', sobre a entrada $(f(x'), 1^n)$, onde $x' \in \{0,1\}^n$, o algoritmo A' obtém o valor $B'(f(x'), 1^t$ para todo $t \in \{n, \ldots, s_I(n) - 1\}$. Em particular, como $m \geq n$ e $m \leq s_I(\ell_I(m)) - 1 = s_I(n) - 1$, segue que o algoritmo A' obtém o valor $B'(f(x'), 1^m)$. Pela definição de g, para todo $x'' \in \{0,1\}^{m-n}$, verifica-se que $f(x') = g(x'x'')$. A afirmação segue. □

[2] Aqui usamos a suposição $z_i \in \{0,1\}^{n+i}$, o que implica que n é o maior inteiro que pertence a I e que é no máximo $n + i$. Em geral, A' dá como saída o prefixo mais longo x' de z_i satisfazendo $|x'| \in I$. Note que $f(x') = g(z_i) = y$ se verifica.

[3] O leitor pode verificar que a análise a seguir não se refere ao comprimento da saída de B' e portanto não depende da suposição simplificadora feita anteriormente.

Afirmação 2.2.3.2: Existe um polinômio $q(\cdot)$ tal que $m < q(\ell_I(m))$ para todo os m's.

Prova: Seja q um polinômio (conforme garantido pela enumerabilidade em tempo-polinomial de I) tal que $s_I(n) < q(n)$. Então, para todo m, temos que $m < s_I(\ell_I(m)) < q(\ell_I(m))$. □

Pela Afirmação 2.2.3.2, o conjunto $S \stackrel{\text{def}}{=} \{\ell_I(m) : m \in M\}$ é infinito (pois, do contrário, para u limitando superiormente os elementos em S obtemos $m < q(\ell_I(m)) \leq q(u)$ para todo $m \in M$, o que contradiz a hipótese de M é infinito). Usando a Afirmação 2.2.3.1, segue que para todo $n = \ell_I(m) \in S$, a probabilidade de que A' inverta f sobre $f(U_n)$ é pelo menos

$$\frac{1}{p(m)} > \frac{1}{p(q(\ell_I(m)))} = \frac{1}{p(q(n))} = \frac{1}{\text{poly}(n)}$$

onde a desigualdade é devida à Afirmação 2.2.3.2. Segue que f não é fortemente unidirecional, em contradição à hipótese da proposição. ■

2.2.3.2 Funções Comprimento-Regulares e Comprimento-Preservantes

Uma segunda convenção útil relativa a funções unidirecionais é assumir que a função f é *comprimento-regular* no sentido de que para todas $x, y \in \{0,1\}^*$, se $|x| = |y|$, então $|f(x)| = |f(y)|$. Apontamos para o fato de que a transformação apresentada acima (i.e., ambas Eq. (2.1) e Eq. (2.2)) preserva a regularidade do comprimento. Um caso especial de regularidade de comprimento, preservado pela Eq. (2.2), é aquele das funções *comprimento-preservantes*.

Definição 2.2.4 (Funções Comprimento-Preservantes): *Uma função é* **comprimento-preservante** *se para toda $x \in \{0,1\}^*$ verifica-se que $|f(x)| = |x|$.*

Dada uma função fortemente (respectivamente, fracamente) unidirecional f, podemos construir uma função fortemente (respectivamente, fracamente) unidirecional f'' que é comprimento-preservante, como segue. Seja p um polinômio limitando a expansão do comprimento de f (i.e., $|f(x)| \leq p(|x|)$). Tal polinômio tem que existir porque f é computável-em-tempo-polinomial. Primeiro construímos uma função comprimento-regular f' definindo

$$f'(x) \stackrel{\text{def}}{=} f(x)10^{p(|x|)-|f(x)|} \tag{2.3}$$

(Usamos um acolchoamento da forma 10^* de modo a facilitar a varredura de $f'(x)$ para $f(x)$ e o acolchoamento da "sobra".) A seguir, definimos f' apenas sobre cadeias de comprimento $p(n)+1$, para $n \in \mathbb{N}$, fazendo

$$f''(x'x'') \stackrel{\text{def}}{=} f'(x'), \text{ onde } |x'x''| = p(|x'|) + 1 \tag{2.4}$$

Claramente, f'' é comprimento-preservante.

Proposição 2.2.5: *Se f é uma função fortemente (respectivamente, fracamente) unidirecional, então f' e f'' também o são (conforme definido em* Eq. (2.3) *e* Eq. (2.4)*, respectivamente).*

Esboço de Prova: É um tanto fácil ver que ambas f' e f'' são computáveis-em-tempo-polinomial. Usando "argumentos de redutibilidade" análogos ao usado na prova precedente, podemos estabelecer a dificuldade-de-inverter de ambas f' e f''. Por exemplo, dado um algoritmo B' para inverter f', construímos um algoritmo A' para inverter f como segue. Sobre a entrada y e 1^n (supostamente y pertence ao codomínio de $f(U_n)$), o algoritmo A' pára com saída $B'(y10^{p(n)-|y|}, 1^n)$. ■

Sobre a Retirada da Entrada Auxiliar $1^{|x|}$. O leitor pode facilmente verificar que se f é comprimento-preservante, então é redundante fornecer ao algoritmo inversor a entrada auxiliar $1^{|x|}$ (além de $f(x)$). O mesmo se verifica se f for comprimento-regular e não encurta sua entrada de mais que uma quantidade polinomial (i.e., existe um polinômio $p(\cdot)$ tal que $p(|f(x)|) \geq |x|$ para toda x). No que se segue, *deveremos lidar apenas com funções unidirecionais que são comprimento-regulares e que não encurtam suas entradas de mais que uma quantidade polinomial*. Além do mais, deveremos lidar sobretudo com funções comprimento-preservantes. Em todos esses casos, *podemos assumir, sem perda de generalidade, que o algoritmo inversor recebe apenas $f(x)$ como entrada*.

Sobre Funções Unidirecionais 1-1. Se f é 1-1, então f' também o é (conforme definido na Eq. (2.3)). Por conseguinte, quando nos é dada uma função unidirecional 1-1, podemos assumir sem perda de generalidade que ela é comprimento-regular, mas não podemos assumir que ela seja comprimento-preservante. Além do mais, a suposição de que funções unidirecionais 1-1 existem parece mais forte que a suposição de que funções unidirecionais arbitrárias (e portanto comprimento-preservantes) existem. Para maiores discussões, veja a Seção 2.4.

2.2.4 Candidatas a Funções Unidirecionais

A seguir estão várias candidatas a funções unidirecionais. Claramente, não se sabe se essas funções são ou não de fato unidirecionais. Essas são apenas conjecturas suportadas por pesquisa extensiva que até agora falhou em produzir um algoritmo inversor eficiente (um que tenha considerável probabilidade de sucesso).

2.2.4.1 Fatoração Inteira

Apesar da pesquisa extensiva direcionada à construção de algoritmos de fatoração factíveis, os melhores algoritmos conhecidos para fatorar inteiros têm tempos de execução subexponenciais. Daí é razoável se acreditar que a função f_{mult} que particiona sua cadeia de entrada em duas partes e retorna o inteiro (ou melhor, a representação binária dele) resultando da multiplicaccão dessas partes (ou melhor, dos inteiros representados por elas) é unidirecional. A saber, seja

$$f_{\text{mult}}(x, y) = x \cdot y$$

onde $|x| = |y|$, e $x \cdot y$ denota (a cadeia representando) o inteiro resultando da multiplicação dos inteiros (representados pelas cadeias) x e y. Claramente f_{mult} pode ser computada em tempo polinomial. Assumindo a intratabilidade da fatoração (e.g., que dado o produto de dois primos de n-bits de comprimento uniformemente escolhidos, é infactível achar os fatores primos), e usando o teorema da densidade-de-primos (que garante que pelo menos $\frac{N}{\log_2 N}$ dos inteiros menores que N são primos), segue

que f_{mult} é pelo menos fracamente unidirecional. (Para maiores discussões, veja o Exercício 8.) Outras funções populares relacionadas à fatoração inteira (e.g., a função RSA) são discutidas na Seção 2.4.3.

2.2.4.2 Decodificação de Códigos Lineares Aleatórios

Um dos mais destacados problema em aberto na área de códigos corretores de erros é aquele de apresentar algoritmos decodificadores eficientes para códigos lineares aleatórios. De particular interesse são os códigos lineares aleatórios com taxas de informação constantes que podem corrigir uma fração constante de erros. Um (n, k, d) *código linear* é uma matriz binária k-por-n na qual a soma vetorial (mod 2) de qualquer subconjunto não-vazio de linhas resulta num vetor com pelo menos d entradas de 1 (1-entradas). (Uma mensagem de k-bits de comprimento é codificada multiplicando-a pela matriz k-por-n, e o vetor de n-bits de comprimento resultante tem uma única pré-imagem mesmo quando se inverte até $\frac{d}{2}$ de suas entradas.) O limitante de Gilbert–Varshamov para códigos lineares garantem a existência de tal código desde que $\frac{k}{n} < 1 - H_2(\frac{d}{n})$, onde $H_2(p) \stackrel{\text{def}}{=} -p \log_2 p - (1-p) \log_2 (1-p)$ se $p < \frac{1}{2}$ e $H_2(p) \stackrel{\text{def}}{=} 1$ caso contrário (i.e., $H_2(\cdot)$ é uma modificação da função de entropia binária). Igualmente, se para algum $\varepsilon > 0$ se verifica que $\frac{k}{n} < 1 - H_2(\frac{(1+\varepsilon)d}{n})$, então quase todas as matrizes binárias k-por-n constituirão (n, k, d) códigos lineares. Considere três constantes $\kappa, \delta, \varepsilon > 0$ satisfazendo $\kappa < 1 - H_2((1+\varepsilon)\delta)$. A função f_{codifica} parece ser um candidato plausível a função unidirecional:

$$f_{\text{codifica}}(C, x, i) \stackrel{\text{def}}{=} (C, xC + e(i))$$

onde C é uma matriz binária κn-por-n, x é um vetor binário κn-dimensional, i é o índice de um vetor binário n-dimensional tendo no máximo $\frac{\delta n}{2}$ 1-entradas dentro de uma enumeração correspondente de tais vetores (o vetor propriamente dito é representado por $e(i)$), e a aritmética pertence ao espaço vetorial binário n-dimensional. Claramente, f_{codifica} é computável-em-tempo-polinomial, desde que usemos uma enumeração de vetores eficiente. Um algoritmo eficiente para inverter f_{codifica} produziria um algoritmo eficiente para decodificar uma fração não-desprezível dos códigos lineares de taxa-constante (o que se constituiria num resultado de abalar o mundo na teoria dos códigos).

2.2.4.3 O Problema da Soma-de-Subconjuntos

Considere a função f_{somasub} definida como segue:

$$f_{\text{somasub}}(x_1, \ldots, x_n, I) = \left(x_1, \ldots, x_n, \sum_{i \in I} x_i \right)$$

onde $|x_1| = \ldots = |x_n| = n$, e $I \subseteq \{1, 2, \ldots, n\}$. Claramente, f_{somasub} é computável-em-tempo-polinomial. O fato de que o problema da soma-de-subconjuntos é $\mathcal{NP}$-completo não pode servir como evidência para a unidirecionalidade de f_{somasub}. Por outro lado, o fato de que o problema da soma-de-subconjuntos é fácil para casos especiais (tais como se ter uma "estrutura ocultada" e/ou "baixa densidade") não descarta essa proposta. A conjectura de que f_{somasub} é unidirecional é baseada na falha dos algoritmos conhecidos em lidar com instâncias "de-alta-densidade" aleatórias (i.e., instâncias nas quais o comprimento dos elementos é aproximadamente igual ao seu número, tal qual na definição de f_{somasub}).

2.2.5 Funções Não-Uniformemente Unidirecionais

Nas duas definições anteriores de funções unidirecionais o algoritmo inversor é um algoritmo probabilístico de tempo-polinomial. Versões mais fortes de ambas as definições requerem que as funções não possam ser invertidas sequer por famílias não-uniformes de circuitos de tamanho-polinomial. Enfatizamos que a condição fácil-de-computar está ainda enunciada em termos de algoritmos uniformes. Por exemplo, a que se segue é uma versão não-uniforme da definição de funções unidirecionais (comprimento-preservantes) fortes.

Definição 2.2.6 (Funções Não-Uniformemente Unidirecionais Fortes): *Uma função $f : \{0,1\}^* \to \{0,1\}^*$ é chamada* **não-uniformemente unidirecional** *se as seguintes condições se verificam:*

1. Fácil de computar: *Existe um algoritmo de tempo-polinomial A tal que sobre a entrada x o algoritmo A dá $f(x)$ como saída.*

2. Difícil de inverter: *Para toda família (mesmo não-uniforme) de circuitos de tamanho-polinomial $\{C_n\}_{n\in\mathbb{N}}$, todo polinômio positivo $p(\cdot)$, e todos os n's suficientemente grandes,*

$$\mathsf{Pr}[C_n(f(U_n)) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

A probabilidade na segunda condição é tomada apenas sobre todos os possíveis valores de U_n. Observamos que qualquer função não-uniformemente unidirecional é unidirecional (i.e., no sentido uniforme).

Proposição 2.2.7: *Se f é não-uniformemente unidirecional, então ela é unidirecional. Isto é, se f satisfaz Definição 2.2.6, então ela também satisfaz Definição 2.2.1.*

Prova: Convertemos qualquer algoritmo inversor probabilístico de tempo-polinomial em uma família não-uniforme de circuitos de tamanho-polinomial, sem decrementar a probabilidade de sucesso. Isso está de acordo com nosso meta-teorema (veja a Seção 1.3.3). Detalhes seguem.

Seja A' um algoritmo (inversor) probabilístico de tempo-polinomial. Suponha que r_n represente uma seqüência de cara-ou-coroa para A' maximizando a probabilidade de sucesso de A' (tomada como média sobre a entrada $f(U_n)$). A saber, r_n satisfaz

$$\mathsf{Pr}[A'_{r_n}(f(U_n)) \in f^{-1}(f(U_n))] \geq \mathsf{Pr}[A'(f(U_n)) \in f^{-1}(f(U_n))]$$

onde a primeira probabilidade é tomada sobre todos os possíveis valores de U_n, e a segunda probabilidade também é tomada sobre todas as possíveis cara-ou-coroa para A'. (Recordemos que $A'_r(y)$ representa a saída do algoritmo A' sobre a entrada y e cara-ou-coroa internos r.) O circuito desejado C_n incorpora o código do algoritmo A' e a seqüência r_n (que é de comprimento polinomial em n). ■

Observamos que, tipicamente, argumentos sobre a média (da forma aplicada mais atrás) nos permite converter algoritmos polinomiais probabilísticos em circuitos não-uniformes polinomiais. Por conseguinte, em geral, noções não-uniformes de segurança (i.e,, robustez contra algoritmos probabilísticos de tempo-polinomial). A recíproca

não é necessariamente verdadeira. Em particular, é possível que funções unidirecionais existam (no sentido uniforme) e ainda assim não existam quaisquer funções não-uniformemente unidirecionais. Entretanto, essa situação (i.e., que funções unidirecionais existem *apenas* no sentido uniforme) parece improvável, e acredita-se largamente que funções não-uniformemente unidirecionais existam. Na verdade, acredita-se que todas as candidatas mencionadas na subseção precedente sejam funções não-uniformemente unidirecionais.

2.3 Funções Unidirecionais Fracas Implicam em Fortes

Primeiro observamos que nem toda função unidirecional fraca é necessariamente uma forte. Considere, por exemplo, uma função unidirecional f (que, sem perda de generalidade, é comprimento-preservante). Modifique f para uma função g tal que $g(p,x) = (p, f(x))$ se p começa com $\log_2 |x|$ zeros, e $g(p,x) = (p,x)$ caso contrário, onde (em ambos os casos) $|p| = |x|$.[4] Afirmamos que g é uma função unidirecional fraca mas não é uma forte. Claramente, g não pode ser uma função unidirecional forte (porque para todas exceto uma fração $\frac{1}{n}$ das cadeias de comprimento $2n$ a função g coincide com a função identidade). Para provar que g é fracamente unidirecional, usamos um "argumento de redutibilidade."

Proposição 2.3.1: *Suponha que f seja uma função unidirecional (mesmo no sentido fraco). Então g, construída anteriormente, é uma função fracamente unidirecional.*

Prova: Intuitivamente, inverter g sobre entradas sobre as quais ela *não* coincide com a transformação identidade está relacionado com inverter f. Portanto, se g for invertida, sobre entradas de comprimento $2n$, com probabilidade que é consideravelmente maior que $1 - \frac{1}{n}$, então g deve ser invertida com probabilidade considerável sobre entradas às quais g aplica f. Por conseguinte, se g não for fracamente unidirecional, então f também não o é. A prova completa, imediata porém tediosa segue.

> Dado um algoritmo probabilístico de tempo-polinomial B' para inverter g, construimos um algoritmo probabilístico de tempo-polinomial A' que inverte f com probabilidade de sucesso "relacionada." A seguir está a descrição do algoritmo A'. Sobre a entrada y, o algoritmo A' faz $n \stackrel{\text{def}}{=} |y|$ e $l \stackrel{\text{def}}{=} \log_2 n$, seleciona p' uniformemente em $\{0,1\}^{n-1}$, computa $z \stackrel{\text{def}}{=} B'(0^l p', y)$, e pára dando como saída o sufixo de n-bits de z. Suponha que S_{2n} represente os conjuntos de todas as cadeias de $2n$-bits de comprimento que começam com $\log_2 n$ zeros (i.e., $S_{2n} \stackrel{\text{def}}{=} \{0^{\log_2 n}\alpha : \alpha \in \{0,1\}^{2n-\log_2 n}\}$). Então pela construção de A' e g, temos
>
> $$\begin{aligned}
> &\mathsf{Pr}[A'(f(U_n)) \in f^{-1}(f(U_n))] \\
> &\quad \geq \mathsf{Pr}[B'(0^l U_{n-l}, f(U_n)) \in (0^l U_{n-l}, f^{-1}(f(U_n)))] \\
> &\quad = \mathsf{Pr}[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))|U_{2n} \in S_{2n}] \\
> &\quad \geq \frac{\mathsf{Pr}[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \mathsf{Pr}[U_{2n} \notin S_{2n}]}{\mathsf{Pr}[U_{2n} \in S_{2n}]} \\
> &\quad = n \cdot \left(\mathsf{Pr}[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \left(1 - \frac{1}{n}\right)\right) \\
> &\quad = 1 - n \cdot (1 - \mathsf{Pr}[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))])
> \end{aligned}$$

[4] Em todo o texto, tratamos $\log_2 |x|$ como se fosse um inteiro. Um argumento preciso pode ser derivado substituindo-se $\log_2 |x|$ por $\lfloor \log_2 |x| \rfloor$ e alguns ajustes menores.

(Para a segunda desigualdade, usamos $\mathsf{Pr}[A|B] = \frac{\mathsf{Pr}[A \cap B]}{\mathsf{Pr}[B]}$ e $\mathsf{Pr}[A \cap B] \geq \mathsf{Pr}[A] - \mathsf{Pr}[\neg B]$.) Não deve vir como surpresa o fato de que a expressão acima tem significado apenas no caso em que $\mathsf{Pr}[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] > 1 - \frac{1}{n}$.

Segue que para todo polinômio $p(\cdot)$ e todo inteiro n, se B' inverte g sobre $g(U_{2n})$ com probabilidade maior que $1 - \frac{1}{p(2n)}$, então A' inverte f sobre $f(U_n)$ com probabilidade maior que $1 - \frac{n}{p(2n)}$. Portanto, se g não for fracamente unidirecional (i.e., para todo polinômio $p(\cdot)$ existe uma quantidade infinita de m's tais que g pode ser invertido sobre $g(U_m)$ com probabilidade $\geq 1 - 1/p(m)$), então f também não é fracamente unidirecional (i.e., para todo polinômio $q(\cdot)$ existe uma quantidade infinita de n's tais que f pode ser invertida sobre $f(U_n)$ com probabilidade $\geq 1 - 1/q(n)$, onde $q(n) = p(2n)/n$). Isso contradiz nossa hipótese (de que f é fracamente unidirecional).

Para resumir, dada um algoritmo probabilístico de tempo-polinomial que inverte g sobre $g(U_{2n})$ com probabilidade de sucesso $1 - \frac{1}{n} + \alpha(n)$, obtemos um algoritmo probabilístico de tempo-polinomial que inverte f sobre $f(U_n)$ com probabilidade de sucesso $n \cdot \alpha(n)$. Por conseguinte, como f é (fracamente) unidirecional, $n \cdot \alpha(n) < 1 - (1/q(n))$ deve se verificar para algum polinômio q, e portanto g deve ser fracamente unidirecional (pois cada algoritmo probabilístico de tempo-polinomial tentando inverter g sobre $g(U_{2n})$ deve falhar com probabilidade de pelo $\frac{1}{n} - \alpha(n) > \frac{1}{n \cdot q(n)}$). ■

Acabamos de mostrar que a menos que nenhuma função unidirecional exista, existem funções unidirecionais fracas que não são fortes. Isso descarta a possibilidade de que todas as funções unidirecionais fracas sejam fortes. Felizmente, podemos também descartar a possibilidade de que todas as funções unidirecionais sejam (apenas) fracas. Em particular, a existência de funções unidirecionais fracas implica a existência de fortes.

Teorema 2.3.2: *Funções unidirecionais fracas existem se e somente se funções unidirecionais fortes existem.*

Recomendamos fortemente que o leitor não pule a prova (dada na Seção 2.3.1), pois acreditamos que a prova é muito instrutiva para o restante do livro. Além do mais, a prova demonstra que amplificação de dificuldade computacional é muito mais complicado que amplificação de um evento probabilístico análogo. Ambos os aspectos são melhor discutidos na Seção 2.3.3. Uma ilustração da prova no contexto de um exemplo "*mascote*" é dada na Seção 2.3.2. (É possível ler a Seção 2.3.2 antes da Seção 2.3.1; na realidade, a maioria dos leitores pode preferir fazer assim.)

2.3.1 A Construção e Sua Análise (Prova do Teorema 2.3.2)

Seja f uma função unidirecional, e p o polinômio garantido pela definição de uma função unidirecional fraca. A saber, todo algoritmo probabilístico de tempo-polinomial falha em inverter f sobre $f(U_n)$ com probabilidade de pelo menos $\frac{1}{p(n)}$. Assumimos, por simplicidade, que f é comprimento-preservante (i.e., $|f(x)| = |x|$ para toda as x's). Esta suposição, que não é realmente essencial, é justificada pela Proposição 2.2.5. Definimos uma função g da seguinte forma:

$$g(x_1, \ldots, x_{t(n)}) \stackrel{\text{def}}{=} f(x_1), \ldots, f\left(x_{t(n)}\right) \tag{2.5}$$

onde $|x_1| = \cdots = |x_{t(n)}| = n$ e $t(n) \stackrel{\text{def}}{=} n \cdot p(n)$. A saber, a entrada de g de $n^2p(n)$-bits de comprimento é particionada em $t(n)$ blocos, cada um de comprimento n, e f é aplicada a cada bloco.

Claramente, g pode ser computada em tempo polinomial (por um algoritmo que quebra a entrada em blocos e aplica f a cada bloco). Além do mais, é fácil ver que inverter g sobre $g(x_1, \ldots, x_{t(n)})$ requer encontrar uma pré-imagem para cada $f(x_i)$. Pode-se ser levado a deduzir que está também claro que g é uma função fortemente unidirecional. Um argumento ingênuo poderia proceder assumindo implicitamente (sem justificativa) que o algoritmo inversor trabalhasse separadamente em cada $f(x_i)$. Se isso fosse realmente o caso, então a probabilidade de que um algoritmo inversor pudesse inverter com sucesso todo $f(x_i)$ seria no máximo $\left(1 - \frac{1}{p(n)}\right)^{n \cdot p(n)} < 2^{-n}$ (o que é desprezível também como uma função de $n^2p(n)$). Entretanto, a suposição de que um algoritmo tentando inverter g funciona independentemente sobre cada $f(x_i)$ não pode ser justificada. Daí, um argumento mais complexo é necessário.

A seguir está um esboço de nossa demonstração. A demonstração de g é fortemente unidirecional procede por um argumento por contradição. Assumimos, ao contrário, que g não é fortemente unidirecional; a saber, assumimos que existe um algoritmo de tempo-polinomial que inverte g com probabilidade que não é desprezível. Derivamos uma contradição apresentando um algoritmo de tempo-polinomial que, para uma quantidade infinita de n's, inverte f sobre $f(U_n)$ com probabilidade maior que $1 - \frac{1}{p(n)}$ (em contradição à nossa hipótese). O algoritmo inversor para f usa o algoritmo inversor para g como uma subrotina (sem assumir nada sobre a maneira pela qual esse último algoritmo opera). (Enfatizamos que não assumimos que o g-inversor funciona de uma forma específica, mas sim usamos qualquer g-inversor para construir, de uma forma genérica, um f-inversor.) Detalhes se seguem.

Suponha que g não seja fortemente unidirecional. Pela definição, segue que existe um algoritmo probabilístico de tempo-polinomial B' e um polinômio $q(\cdot)$ tal que para uma quantidade infinita de m's,

$$\mathsf{Pr}[B'(g(U_m)) \in g^{-1}(g(U_m))] > \frac{1}{q(m)} \tag{2.6}$$

Vamos representar por M' o conjunto infinito de inteiros para os quais isso se verifica. Suponha que N' represente o conjunto infinito de n's para os quais $n^2 \cdot p(n) \in M'$ (note que todos os m's considerados são da forma $n^2 \cdot p(n)$, para algum inteiro n).

Usando B', apresentamos agora um algoritmo probabilístico de tempo-polinomial A' para inverter f. Sobre a entrada y (supostamente no contradomínio de f), o algoritmo A' procede aplicando o seguinte procedimento probabilístico, representado por I, sobre a entrada y por $a(|y|)$ vezes, onde $a(\cdot)$ é um polinômio que depende dos polinômios p e q (especificamente, fazemos $a(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2p(n))$).

Procedimento I

Entrada: y (faça $n \stackrel{\text{def}}{=} |y|$).

`Para` $i = 1$ `até` $t(n)$ `faça`

1. Selecione uniformemente e independentemente uma seqüência de cadeias $x_1, \ldots, x_{t(n)} \in \{0,1\}^n$.

2. Compute $(z_1, \ldots, z_{t(n)}) \leftarrow B'(f(x_1), \ldots, f(x_{i-1}), y, f(x_{i+1}), \ldots, f(x_{t(n)}))$.
(Note que y é substituído na i-ésima posição no lugar de $f(x_i)$.)

3. Se $f(z_i) = y$, então páre e dê como saída z_i.
(Isso é considerado um *sucesso*).
`fim`

Usando a Eq. (2.6), apresentamos agora um limitante inferior sobre a probabilidade de sucesso do algoritmo A'. Para esse fim definimos um conjunto, representado por S_n, que contém todas as cadeias de n-bits sobre as quais o procedimento I é bem sucedido com probabilidade não-desprezível (especificamente, maior que $\frac{n}{a(n)}$). (A probabilidade é tomada apenas sobre os cara-ou-coroa do procedimento I.) A saber,

$$S_n \stackrel{\text{def}}{=} \left\{ x : \mathsf{Pr}[I(f(x)) \in f^{-1}(f(x))] > \frac{n}{a(n)} \right\}$$

Nas próximas duas afirmações mostraremos que S_n contém todas exceto uma fração das cadeias de comprimento $n \in N'$ e que para cada cadeia $x \in S_n$ o algoritmo A' inverte f sobre $f(x)$ com probabilidade exponencialmente próxima a 1. Seguirá que A' inverte f sobre $f(U_n)$, para todo $n \in N'$, com probabilidade maior que $q - \frac{1}{p(n)}$, em contradição à nossa hipótese.

Afirmação 2.3.2.1: Para toda $x \in S_n$

$$\mathsf{Pr}[A'(f(x)) \in f^{-1}(f(x))] > 1 - \frac{1}{2^n}$$

Demonstração: Pela definição do conjunto S_n, o procedimento I inverte $f(x)$ com probabilidade pelo menos $\frac{n}{a(n)}$. O algoritmo A' simplesmente repete I por $a(n)$ vezes, e portanto

$$\mathsf{Pr}[A'(f(x)) \notin f^{-1}(f(x))] < \left(1 - \frac{n}{a(n)}\right)^{a(n)} < \frac{1}{2^n}$$

A afirmação segue. □

Afirmação 2.3.2.2: Para todo $n \in N'$

$$|S_n| > \left(1 - \frac{1}{2p(n)}\right) \cdot 2^n$$

Demonstração: Assumimos, ao contrário, que $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. Chegaremos a uma contradição à Eq. (2.6) (i.e., nossa hipótese relativa à probabilidade de sucesso de B'). Recordemos que por essa hipótese (para $n \in N'$),

$$s(n) \stackrel{\text{def}}{=} \mathsf{Pr}[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))] > \frac{1}{q(n^2p(n))} \tag{2.7}$$

Suponha que $U_n^{(1)}, \ldots, U_n^{(n \cdot p(n))}$ representem os blocos de n-bits de comprimento na variável aleatória $U_{n^2p(n)}$ (i.e., esses $U_n^{(i)}$'s são variáveis aleatórias independentes cada uma uniformemente distribuídas em $\{0,1\}^n$). Particionamos o evento considerado na

Eq. (2.7) em dois eventos disjuntos correspondentes a se um dos $U_n^{(i)}$'s reside ou não fora de S_n. Intuitivamente, B' não pode desempenhar bem em tal caso, pois esse caso corresponde à probabilidade de sucesso de I sobre pré-imagens fora de S_n. Por outro lado, a probabilidade de que todos os $U_n^{(i)}$'s residam em S_n é pequena. Especificamente, definimos

$$s_1(n) \stackrel{\text{def}}{=} \mathsf{Pr}[B'(g(U_{n^2\cdot p(n)})) \in g^{-1}(g(U_{n^2\cdot p(n)})) \wedge (\exists i \text{ s.t. } U_n^{(i)} \notin S_n)]$$

e

$$s_2(n) \stackrel{\text{def}}{=} \mathsf{Pr}[B'(g(U_{n^2\cdot p(n)})) \in g^{-1}(g(U_{n^2\cdot p(n)})) \wedge (\forall i : U_n^{(i)} \in S_n)]$$

Claramente, $s(n) = s_1(n) + s_2(n)$ (pois os eventos considerados nos s_i's são disjuntos). Derivamos uma contradição ao limitante inferior sobre $s(n)$ (dado na Eq. (2.7)) apresentando limitantes superiores para ambos $s_1(n)$ e $s_2(n)$ (que somam menos).

Primeiro, apresentamos um limitante superior sobre $s_1(n)$. A observação chave é que o algoritmo I inverte f sobre a entrada $f(x)$ com probabilidade que está relacionada com o sucesso de B' em inverter g sobre uma seqüência de f-imagens aleatórias contendo $f(x)$. Especificamente, para toda $x \in \{0,1\}^n$ e todo $1 \leq i \leq n \cdot p(n)$, a probabilidade de que I inverta f sobre $f(x)$ é maior ou igual à probabilidade de que B' inverta g sobre $g(U_{n^2p(n)})$ condicionada sobre $U_n^{(i)} = x$ (pois qualquer sucesso de B' em inverter g significa que f foi invertida no i-ésimo bloco, e por conseguinte contribui para a probabilidade de sucesso de I). Segue que, para toda $x \in \{0,1\}^n$ e todo $1 \leq i \leq n \cdot p(n)$,

$$\begin{aligned}&\mathsf{Pr}[I(f(x)) \in f^{-1}(f(x))] \\ &\quad \geq \mathsf{Pr}[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))|U_n^{(i)} = x]\end{aligned} \tag{2.8}$$

Como para $x \notin S_n$ o lado esquerdo não pode ser grande, mostraremos que (o lado direito, e portanto) $s_1(n)$ não pode ser grande. Especificamente, usando Eq. (2.8), segue que

$$\begin{aligned}
s_1(n) &= \mathop{\mathsf{Pr}}_{n\cdot p(n)}[\exists i \text{ t.q. } B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \mathsf{Pr}[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \sum_{x\notin S_n} \mathsf{Pr}[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} = x] \\
&= \sum_{i=1}^{n\cdot p(n)} \sum_{x\notin S_n} \mathsf{Pr}[U_n^{(i)} = x] \cdot \mathsf{Pr}[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \mid U_n^{(i)} = x] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \max_{x\notin S_n}\{\mathsf{Pr}[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \mid U_n^{(i)} = x]\} \\
&\leq \sum_{i=1}^{n\cdot p(n)} \max_{x\notin S_n}\{\mathsf{Pr}[I(f(x)) \in f^{-1}(f(x))]\} \\
&\leq n \cdot p(n) \cdot \frac{n}{a(n)} = \frac{n^2 \cdot p(n)}{a(n)}
\end{aligned}$$

(A última desigualdade usa a definição de S_n, e a anterior usa Eq. (2.8).)

Agora apresentamos um limitante superior sobre $s_2(n)$. Relembremo-nos de que pela hipótese da contradição, $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. Segue que

$$\begin{aligned} s_2(n) &\leq \mathsf{Pr}[\forall i : U_n^{(i)} \in S_n] \\ &\leq \left(1 - \frac{1}{2p(n)}\right)^{n \cdot p(n)} \\ &< \frac{1}{2^{n/2}} < \frac{n^2 \cdot p(n)}{a(n)} \end{aligned}$$

(A última desigualdade se verifica para n suficientemente grande.)

Combinando os limitantes superiores sobre os s_i's, temos $s_1(n)+s_2(n) < \frac{2n^2 \cdot p(n)}{a(n)} = \frac{1}{q(n^2 p(n))}$, onde a igualdade é pela definição de $a(n)$. Mesmo assim, por outro lado, $s_1(n)+s_2(n) = s(n) > \frac{1}{q(n^2 p(n))}$, onde a desigualdade é devida à Eq. (2.7). Contradição é atingida, e a afirmação segue. □

Combinando as Afirmações 2.3.2.1 e 2.3.2.2, obtemos

$$\begin{aligned} &\mathsf{Pr}[A'(f(U_n)) \in f^{-1}(f(U_n))] \\ &\quad \geq \mathsf{Pr}[A'(f(U_n)) \in f^{-1}(f(U_n)) \wedge U_n \in S_n] \\ &\quad = \mathsf{Pr}[U_n \in S_n] \cdot \mathsf{Pr}[A'(f(U_n)) \in f^{-1}(f(U_n)) \mid U_n \in S_n] \\ &\quad \geq \left(1 - \frac{1}{2p(n)}\right) \cdot (1 - 2^{-n}) > 1 - \frac{1}{p(n)} \end{aligned}$$

Segue que existe um algoritmo probabilístico de-tempo-polinomial (i.e., A') que inverte f sobre $f(U_n)$, para $n \in N'$, com probabilidade maior que $1 - \frac{1}{p(n)}$. Essa conclusão, que segue da hipótese de que g não é fortememente unidirecional (i.e., Eq. (2.6)), se põe em contradição à hipótese de que todo algoritmo probabilístico de-tempo-polinomial falha em inverter f com probabilidade no mínimo $\frac{1}{p(n)}$, e o teorema segue. ■

2.3.2 Ilustração por um Exemplo Mascote

Vamos tentar esclarecer ainda mais as idéias algorítmicas subjacentes à prova do Teorema 2.3.2. Para fazer isso, considere a seguinte noção quantitativa de funções unidirecionais fracas. Dizemos que uma f (computável em-tempo-polinomial) é ρ-unidirecional se para todos os algoritmos probabilísticos de-tempo-polinomial A', para somente uma quantidade finita de n's, a probabilidade de que sobre a entrada $f(U_n)$ o algoritmo A' *falha* em encontrar uma pré-imagem sob f é *no mínimo* $\rho(n)$. (Cada função unidirecional fraca é $1/p()$-unidirecional para algum polinômio p, enquanto que funções unidirecionais são $(1 - \mu())$-unidirecionais, onde μ é uma função desprezível.)

Proposição 2.3.3 (Exemplo Mascote): *Suponha que f seja $\frac{1}{3}$-unidirecional, e suponha que $g(x_1, x_2) \stackrel{\text{def}}{=} (f(x_1), f(x_2))$. Então g é 0,55-unidirecional (onde $0,55 < 1 - (\frac{2}{3})^2$).*

Esboço de Prova: Suponha, para que se chegue uma contradição, que existe um algoritmo de-tempo-polinomial A' que inverte $g(U_{2n})$ com probabilidade de sucesso maior que $1 - 0,55 = 0,45$, para uma quantidade infinita de n's. Considere um n qualquer desses, e faça $N \stackrel{\text{def}}{=} 2^n$. Assuma por simplicidade que A' é determinístico. Considere uma matriz N-por-N com entradas correspondendo a pares $(x_1, x_2) \in \{0,1\}^n \times \{0,1\}^n$

tais que a entrada (x_1, x_2) é marcada com 1 se A' inverte g com sucesso sobre a entrada $g(x_1, x_2) = (f(x_1), f(x_2))$ e é marcada com zero caso contrário. Nossa hipótese de contradição é que a fração de entradas 1 na matriz é maior que 45%.

A suposição ingênua (injustificada) é de que A' opera separadamente sobre cada elemento do par $(f(x_1), f(x_2))$. Se isso fosse o caso, então a região de sucesso de A' teria sido um retângulo generalizado $R \times C \subseteq \{0,1\}^n \times \{0,1\}^n$ (i.e., correspondendo a todos os pares (x_1, x_2) tais que $x_1 \in R$ e $x_2 \in C$ para alguns conjuntos $R \subseteq \{0,1\}^n$ e $C \subseteq \{0,1\}^n$.) Usando a hipótese de que f é $\frac{1}{3}$-unidirecional, temos $|R|$, $|C| \leq \frac{2}{3} \cdot N$, e portanto $\frac{|R \times C|}{N^2} \leq \frac{4}{9} < 0,45$, em contradição com nossa hipótese a respeito de A'.

Entretanto, conforme enunciado anteriormente, a suposição ingênua não pode ser justificada, e portanto um argumento mais complexo é necessário. Em geral, a região de sucesso de A', denotada por S, pode ser um subconjunto arbitrário de $\{0,1\}^n \times \{0,1\}^n$ satisfazendo $|S| > 0,45 \cdot N^2$ (pela hipótese de contradição). Vamos chamar uma linha x_1 (resp., coluna x_2) de *boa* se ela contém pelo menos $0,1\%$ de entradas 1; caso contrário ela é chamada de *ruim*. (Veja a Figura **??**.) A parte algorítmica principal da prova é estabelecer a seguinte afirmação.

Afirmação 2.3.3.1: *A proporção de linhas (resp., colunas) boas é no máximo* $66,8\%$.

Uma vez que essa afirmação esteja provada, tudo o que resta é combinatória trivial (i.e., contagem). Ou seja, limitamos por cima o tamanho de S contando separadamente o número de 1-entradas na interseção das linhas boas e colunas boas e as 1-entradas em linhas ruins e colunas ruins: Pela Afirmação 2.3.3.1, existem no máximo $(0,668N)^2$ entradas na interseção de linhas boas e colunas boas, e por definição o número de 1-entradas em cada linha ruim (respectivamente, coluna ruim) é no máximo $0,001N$. Portanto, $|S| \leq (0,668N)^2 + 2 \cdot N \cdot 0,001N < 0,449 \cdot N^2$, em contradição à nossa hipótese (i.e., $|S| > 0,45 \cdot N^2$).

Prova da Afirmação 2.3.3.1: Suponha, para efeito de uma contradição, que a fração de linhas boas é maior que $66,8\%$ (o argumento para colunas é análogo). Então, para atingir uma contradição, construimos um algoritmo para inverter f da seguinte forma. Sobre a entrada y, o algoritmo repete os seguintes passos 10.000 vezes:

1. Selecione x_2 uniformemente em $\{0,1\}^*$.
2. Invoque A' sobre a entrada $(y, f(x_2))$, e obtenha sua saída (x', x'').
3. Se $f(x') = y$, então páre com saída x'.

Claramente, esse algoritmo funciona em tempo polinomial, e falta analisar seu sucesso em inverter f. Para toda boa x_1, a probabilidade de que o algoritmo falhe em inverter f sobre a entrada $y = f(x_1)$ é no máximo $(1 - 0,001)^{10.000} < 0,001$. Por conseguinte, a probabilidade de que o algoritmo seja bem sucedido em inverter f sobre a entrada $f(U_n)$ é no mínimo $0,668 \cdot 0,999 > \frac{2}{3}$, em contradição à hipótese de que f é $\frac{1}{3}$-unidirecional. □

2.3.3 Discussão

2.3.3.1 Argumentos de Redutibilidade: Uma Discussão

Vamos relembrar a estrutura da prova do Teorema 2.3.2. Dada uma função unidirecional fraca f, primeiro construimos uma função computável-em-tempo-polinomial g. Isso foi feito com a intenção de mais adiante provar que g é fortemente unidirecional. Para provar que g é fortemente unidirecional, usamos um *argumento de redutibilidade*. O argumento transforma algoritmos eficientes que supostamente contradizem a unidirecionalidade forte de g em algoritmos eficientes que contradizem a hipótese de que f é fracamente unidirecional. Daí g tem que ser fortemente unidirecional. Enfatizamos que nossa transformação algorítmica, que é na verdade uma redução de Cook aleatorizada,[5] não faz quaisquer suposições implícitas ou explícitas sobre a estrutura dos possíveis algoritmos para inverter g. Suposições tais como a suposição "natural" de que o inversor de g funciona independentemente sobre cada bloco não pode ser justificada (pelo menos não no nosso estado atual do entendimento da natureza das computações eficientes).

Usamos o termo *argumento de redutibilidade*, ao invés de simplesmente dizer uma redução, de modo a enfatizar que *não* nos referimos aqui a reduções (no-pior-caso) padrão. Vamos esclarecer a distinção: Em ambos os casos nos referimos a *reduzir* a tarefa de resolver um problema para a tarefa de resolver um outro problema; ou seja, usamos um procedimento que resolve a segunda tarefa de modo a construir um procedimento que resolve a primeira tarefa. Entretanto, nas reduções padrão assume-se que a segunda tarefa tem um procedimento perfeito que a resolve em todas as instâncias (i.e., no pior-caso) e constrói tal procedimento para a primeira tarefa. Por conseguinte, a redução pode invocar o procedimento dado (para a segunda tarefa) sobre instâncias "não-típicas". Isso não pode ser feito em nossos argumentos de redutibilidade. Aqui, nos é dado um procedimento que resolve a segunda tarefa *com certa probabilidade com respeito a certa distribuição*. Por conseguinte, ao empregar um argumento de redutibilidade, não podemos invocar esse procedimento sobre qualquer instância. Ao contrário, temos que considerar a distribuição de probabilidade, sobre instâncias da segunda tarefa, induzida por nossa redução. Em muitos casos essa última distribuição é igual à distribuição à qual a hipótese (relativa à solubilidade da segunda tarefa) se refere, mas outros casos podem ser tratados também (e.g., essas distribuições podem ser "suficientemente próximas" para o propósito específico). Em todo caso, uma análise cuidadosa da distribuição induzida pelo argumento de redutibilidade é devido.

2.3.3.2 O Análogo Informação-Teórico

O Teorema 2.3.2 tem um análogo informação-teórico (ou "probabilístico") natural que assevera que repetir um experimento que tem uma probabilidade de falha detectável uma quantidade suficiente de vezes acarretará em alguma falha com probabilidade muito alta. O leitor está provavelmente convencido nesse ponto de que a prova do Teorema 2.3.2 é muito mais complexa que a prova do análogo informação-teórico. No contexto informação-teórico, os eventos repetidos são independentes por definição, enquanto que em nosso contexto computacional nenhuma tal independência (que corresponde ao argumento ingênuo dado no início da prova do Teorema 2.3.2) pode ser garantida. Uma outra indicação da diferença entre os dois cenários segue. No cenário

[5] Uma redução de Cook (aleatorizada) de um problema computacional Π_i para um outro problema, denotado Π_2, é uma máquina oráculo (probabilística) de tempo-polinomial que resolve Π_1, enquanto fazendo consultas ao oráculo Π_2.

informação-teórico, a probabilidade de que nenhum dos eventos de falha ocorrerá decresce exponencialmente com o número de repetições. Em contraste, no cenário computacional podemos atingir somente um limitante desprezível não-especificado sobre as probabilidades inversoras de algoritmos de tempo-polinomial. Além do mais, pode ser o caso de que a g construída na prova do Teorema 2.3.2 pode ser eficientemente invertida sobre $g(U_{n^2p(n)})$ com probabilidade de sucesso que é sub-exponencialmente decrescente (e.g., com probabilidade $2^{-(\log_2 n)^3}$, enquanto que o limitante informação-teórico análogo é exponencialmente decrescente (i.e., e^{-n}).

2.3.3.3 Funções Unidirecionais Fracas Versus Fortes: Um Resumo

Pelo Teorema 2.3.2, sempre que assumimos a existência de funções unidirecionais, não há necessidade de especificar se nos referimos a fracas ou fortes. Ou seja, com relação à mera existência de função unidirecional, as noções de funções unidirecionais fracas e fortes são equivalentes. Entretanto, no que concerne a considerações de eficiência, as duas noções não são de fato equivalentes, pois a transformação acima de funções unidirecionais fracas em fortes não é prática. Uma transformação alternativa, que é muito mais eficiente, realmente existe para o caso de permutações unidirecionais e outras classes específicas de funções unidirecionais. O leitor interessado é remetido à Seção 2.6.

2.4 Funções Unidirecionais: Variações

Nesta seção discutimos várias questões concernentes a funções unidirecionais. Na primeira subseção apresentamos uma função que é (fortemente) unidirecional, desde que funções unidirecionais existam. A construção dessa função é de estrito interesse abstrato. Em contraste, as questões discutidas nas outras subseções são de importância prática. Primeiro, apresentamos uma formulação alternativa de funções unidirecionais. Essa formulação é mais apropriada para descrever muitas candidatas naturais a funções unidirecionais, e de fato usamo-la de modo a descrever algumas candidatas populares a funções unidirecionais. A seguir, usamos essa formulação para apresentar funções unidirecionais com propriedades adicionais; especificamente, consideramos permutações (unidirecionais) com alçapão e pares de função livres-de-presas. Observamos que essas propriedades são usadas em várias construções apresentadas em outros capítulos deste livro (e.g., permutações com alçapão são usadas na construção de esquemas de encriptação de chave-pública, enquanto que permutações livres-de-presas são usadas na construção de dispersão livre-de-colisão). Concluimos esta seção com observações concernentes à "arte" de propor candidatas a funções unidirecionais.

2.4.1 * Funções Unidirecionais Universais

Usando a noção de uma maáquina universal e o resultado da seção precedente, é possível provar a existência de uma função unidirecional *universal*; ou seja, apresentamos uma função (fixada) que é unidirecional, desde que funções unidirecionais existam.

> **Proposição 2.4.1:** *Existe uma função computável em tempo-polinomial que é (fortemente) unidirecional se e somente se funções unidirecionais existem.*

Esboço da Prova. Uma observação chave é que existem funções unidirecionais se e somente se existem funções unidirecionais que possam ser calculadas por um algoritmo de tempo-quadrático. (A escolha do limitante de tempo específico é imaterial; o que é importante é que tal limitante de tempo específico existe.) Esse enunciado é provado usando um argumento de preenchimento. Detalhes seguem.

Seja f uma função unidirecional arbitrária, e suponha que $p(\cdot)$ seja um polinômio limitando a complexidade de tempo de um algoritmo para computar f. Defina $g(x'x'') \stackrel{\text{def}}{=} f(x')x''$, onde $|x'x''| = p(|x'|)$ e então aplica f a x'. A análise sintática e as outras operações extra podem ser implementadas em tempo quadrático (em $|x'x''|$), enquanto que computar $f(x')$ é feito dentro do tempo $p(|x'|) = |x'x''|$ (que é linear no comprimento da entrada. Daí, g pode ser computada (por uma máquina de Turing) em tempo quadrático. O leitor pode verificar que g é unidirecional usando um "argumento de redutibilidade" (análogo àquele usado na prova da Proposição 2.2.5).

Agora apresentamos uma função (unidirecional universal), denotada f_{univ}:

$$f_{\text{univ}}(\text{desc}(M), x) \stackrel{\text{def}}{=} (\text{desc}(M), M(x)) \tag{2.9}$$

onde $\text{desc}(M)$ é uma descrição de uma máquina de Turing M, e $M(x)$ é definida como a saída de M sobre a entrada x se M roda no máximo em tempo quadrático sobre x, e $M(x)$ é definida como x caso contrário. (Sem perda de generalidade, podemos ver qualquer cadeia como a descrição de alguma máquina de Turing.) Claramente f_{univ} pode ser computada em tempo polinomial por uma máquina universal que usa um contador de passos. Para mostrar que f_{univ} é fracamente unidirecional (desde que funções unidirecionais realmente existam), usamos um "argumento de redutibilidade."

Assumindo que funções unidirecionais existem, e usando a observação acima, segue que existe uma função unidirecional g que é computada em tempo quadrático. Seja M_g a máquina de tempo quadrático que computa g. Claramente, um algoritmo (eficiente) que inverte f_{univ} sobre entradas da forma $f_{\text{univ}}(\text{desc}(M), U_n)$ com probabilidade $p(n)$ pode ser facilmente modificado para um algorimo (eficiente) que inverte g sobre entradas da forma $g(U_n)$ com probabilidade $p(n)$. Como na prova da Proposição 2.3.1, segue que um algoritmo que inverte f_{univ} com probabilidade no mínimo $1 - \varepsilon(n)$ sobre cadeias de comprimento $|\text{desc}(M_g)| + n$ produz um algoritmo que inverte g com probabilidade no mínimo $1 - 2^{|\text{desc}(M_g)|} \cdot \varepsilon(n)$ sobre cadeias de comprimento n. (Enfatizamos que $|\text{desc}(M_g)|$ é uma constante, dependento apenas de g.) Daí, se f_{univ} não é fracamente unidirecional (i.e., a função ε não é detectável), então g também não pode ser (fracamente) unidirecional (i.e., também $2^{|\text{desc}(M_g)|} \cdot \varepsilon(n)$ não é detectável.)

Usando o Teorema 2.3.2 (para transformar a função unidirecional fraca f_{univ} em uma forte), a proposição segue. ■

Discussão. A observação pela qual basta considerar funções unidirecionais que podem ser calculadas dentro de um limitante de tempo específico é crucial para a construção

de f_{univ}, a razão sendo que não é possível construir uma máquina de tempo-polinomial que seja universal para a classe de todas as máquinas de tempo-polinomial (i.e., uma máquina que possa "simular" todas as máquinas de tempo-polinomial). É, no entanto, possível construir, para todo polinômio $p(\cdot)$, uma máquina de tempo-polinomial que é universal para a classe de máquinas com tempo de execução limitado por $p(\cdot)$.

A impraticabilidade da construção de f_{univ} provém do fato de que f_{univ} é provavelmente difícil de inverter somente sobre entradas de comprimento enorme (i.e., comprimento permitindo a codificação de algoritmos não-triviais conforme requeridos para o cálculo de funções unidirecionais). Além do mais, para obter uma função fortemente unidirecional a partir de f_{univ}, precisamos aplicar essa última sobre uma seqüência de mais que 2^L entradas, cada uma de comprimento $L+n$, onde L é um limitante inferior sobre o comprimento da codificação de potenciais funções unidirecionais, e n é nosso real parâmetro de segurança.

Ainda assim, a Proposição 2.4.1 diz que, em princípio, a questão de se saber se funções unidirecionais existem ou não "se reduz" à questão de se saber se uma função específica é ou não unidirecional.

2.4.2 Funções Unidirecionais como Coleções

A formulação de funções unidirecionais usada até agora é adequada para uma discussão abstrata. Entretanto, para descrever muitas candidatas naturais a funções unidirecionais, a seguinte formulação (embora sendo mais enfadonho) é mais usável. Ao invés de ver funções unidirecionais como funções operando sobre um domínio infinito (i.e., $\{0,1\}^*$), consideramos coleções finitas de funções cada uma operando sobre um domínio finito. As funções na coleção compartilham um único algoritmo de cálculo que quando recebem como entrada uma representação sucinta de uma função e um elemento no seu domínio retorna o valor da função especificada no ponto dado. A formulação de uma coleção de funções também é útil para a apresentação de permutações do tipo alçapão e funções livres-de-presas (veja as Seções 2.4.4 e 2.4.5, respectivamente). Começamos com a seguinte definição.

> **Definição 2.4.2 (Coleção de Funções):** *Uma* **coleção de funções** *consiste de um conjunto infinito de* **índices***, denotado $\overline{I}$, e um conjunto correspondente de funções finitas, denotado $\{f_i\}_{i\in\overline{I}}$. Ou seja, para cada $i \in \overline{I}$, o domínio da função f_i, denotado D_i, é um conjunto finito.*

Tipicamente, o conjunto de índices $\overline{I}$ será um subconjunto "denso" do conjunto de todas as cadeias; ou seja, a fração de cadeias de n-bits de comprimento em $\overline{I}$ será detectável (i.e., $|\overline{I} \cap \{0.1\}^n| \geq 2^n/\mathrm{poly}(n)$).

Estaremos interessados somente em coleções de funções que podem ser usadas em aplicações criptográficas. Como sugerido anteriormente, uma condição necessária para usar uma coleção de funções é a existência de um algoritmo eficiente para cálculo-da-função (denotado F) que sobre a entrada $i \in \overline{I}$ e $x \in D_i$ retorne $f_i(x)$. Mesmo assim essa condição por si só não basta. Precisamos ser capazes de selecionar (aleatoriamente) um índice especificando uma função sobre um domínio suficientemente grande, assim como ser capazes de selecionar (aleatoriamente) um elemento do domínio (quando recebemos o índice do domínio). A propriedade de amostragem do conjunto de índices é capturada por um algoritmo eficiente (denotado I) que sobre a entrada de um inteiro n (apresentado em unário) seleciona aleatoriamente um índice de $\mathrm{poly}(n)$-bits de comprimento especificando uma função e seu domínio associado. (Como de costume,

apresentação em unário é usada de modo a estar de acordo com a associação padrão de algoritmos eficientes com aqueles rodando em tempo polinomial nos comprimentos de suas entradas.) A propriedade de amostragem dos domínios é capturada por um algoritmo eficiente (denotado D) que sobre a entrada de um índice i seleciona aleatoriamente um elemento em D_i. A propriedade unidirecional da coleção é capturada ao se exigir que todo algoritmo eficiente, quando recebe um índice de uma função e um elemento no seu codomínio, falha em inverter a função, exceto com probabilidade desprezível. A probabilidade é tomada sobre a distribuição induzida pelos algoritmos de amostragem I e D. Tudo acima é capturado pela seguinte definição.

Definição 2.4.3 (Coleção de Funções Unidirecionais): *Uma coleção de funções unidirecionais* $\{f_i : d_i \to \{0,1\}^*\}_{i\in\overline{I}}$ *é chamada* fortemente *(respectivamente,* fracamente*)* **unidirecional** *se existem três algoritmos probabilísticos de tempo-polinomial* I, D, *e* F *tais que as duas condições seguintes se verificam:*

1. Fácil de amostrar e computar: *A distribuição de saída do algoritmo* I *sobre a entrada* 1^n *é uma variável aleatória com valores atribuídos do conjunto* $\overline{I} \cap \{0,1\}^n$. *A distribuição de saída do algoritmo* D *sobre a entrada* $i \in \overline{I}$ *é uma variável aleatória com valores atribuídos de* D_i. *Sobre a entrada* $i \in \overline{I}$ *e* $x \in D_i$, *o algoritmo* F sempre *dá como saída* $f_i(x)$.
   (Portanto, $D_i \subseteq \bigcup_{m\leq \mathrm{poly}(|i|)}\{0,1\}^m$. *Sem perda de generalidade, podemos assumir que* $D_i \subseteq \{0,1\}^{\mathrm{poly}(|i|)}$. *Também sem perda de generalidade, podemos assumir que o algoritmo* F *é determinístico.)*
2. Difícil de inverter (versão para fortemente unidirecional): *Para todo algoritmo probabilístico de tempo-polinomial* A', *todo polinômio positivo* $p(\cdot)$, *e todo* n *suficientemente grande,*

$$\Pr[A'(I_n, f_{I_n}(X_n)) \in f^{-1}(f_{I_n}(X_n))] < \frac{1}{p(n)}$$

   onde I_n *é uma variável aleatória descrevendo a distribuição de saída do algoritmo* I *sobre a entrada* 1^n, *e* X_n *é uma variável aleatória descrevendo a saída do algoritmo* D *sobre a entrada (variável aleatória)* I_n.
   (A versão para coleções fracamente unidirecionais é análoga.)

Enfatizamos que a saída do algoritmo I sobre a entrada 1^n *não é* necessariamente distribuída *uniformemente* sobre $\overline{I} \cap \{0,1\}^n$. Além do mais, não é sequer exigido que $I(1^n)$ não esteja inteiramente concentrado sobre uma única cadeia. Igualmente, a saída do algoritmo D sobre a entrada i *não é* necessariamente distribuída *uniformemente* sobre D_i. Mesmo assim a condição de dificuldade-de-inverter implica que $D(i)$ não pode estar principalmente concentrada sobre uma quantidade polinomial (em $|i|$) de cadeias. Enfatizamos que a coleção é difícil de inverter com respeito à distribuição induzida pelos algoritmos I e D (em adição a depender, como de costume, do mapeamento induzido pela função propriamente dita).

Podemos descrever a coleção de funções unidirecionais indicando a tripla correspondente de algoritmos. Daí, podemos dizer que uma *tripla de algoritmos probabilísticos de tempo-polinomial* (I, D, F) *constitui uma coleção de funções unidirecionais* se existe uma coleção de funções para as quais esses algoritmos satisfazem as duas condições acima.

Claramente, qualquer coleção de funções unidirecionais pode ser representada como uma função unidirecional e vice-versa (veja o Exercício 18), e mesmo assim cada formulação tem suas próprias vantagens. No que se segue, usaremos a formulação de uma coleção de funções unidirecionais de modo a apresentar candidatas populares para funções unidirecionais.

Relaxações. Para permitir uma apresentação menos incômoda das candidatas naturais a coleções unidirecionais (de funções), relaxamos a Definição 2.4.3 de duas maneiras. Primeiro, permitimos que o algoritmo de amostragem do índice dê como saída, na entrada 1^n, índices de comprimento $p(n)$ ao invés de n, onde $p(\cdot)$ é algum polinômio. Segundo, permitimos que todos os algoritmos falhem com probabilidade desprezível. Mais importante, permitimos que o amostrador de índice I dê como saída cadeias que não estão em $\overline{I}$ desde que a probabilidade de que $I(1^n) \notin \overline{I} \cap \{0,1\}^{p(n)}$ seja uma função dezprezível em n. (As mesmas relaxações podem ser usadas quando discutindo permutações com alçapão e funções livres-de-presas.)

Propriedades Adicionais: Índices e Domínios Eficientemente Reconhecíveis. Várias propriedades adicionais que se verificam para algumas coleções candidatas a funções unidirecionais serão explicitamente discutidas em subseções subseqüentes. Aqui mencionamos duas propriedades adicionais (úteis) que se verificam em algumas coleções candidatas a funções unidirecionais. As propriedades são (1) ter um conjunto de índices eficientemente reconhecível e (2) ter uma coleção de domínios eficientemente reconhecível; ou seja, nos referimos à existência de um algoritmo eficiente para decidir pertinência em $\overline{I}$ e a existência de um algoritmo eficiente que, dado $i \in \overline{I}$ e x, pode determinar se $x \in D_i$ ou não. Note que para a Definição 2.4.3 não-relaxada, as moedas usadas para gerar $i \in \overline{I}$ (respectivamente, $x \in D_i$) constituem um certificado (i.e., uma $\mathcal{NP}$-testemunha) para a afirmação correspondente; ainda assim esse certificado de que $i \in \overline{I}$ (respectivamente, $x \in D_i$) pode assistir em inverter da função f_i (respectivamente, sempre produzir a pré-imagem x).

2.4.3 Exemplos de Coleções Unidirecionais

2.4.4 Permutações Unidirecionais do tipo Alçapão

2.4.5 Funções Livres-de-Presas

2.4.6 Sobre Propor Candidatas

2.5 Predicados Núcleo-Duro

2.5.1 Definição

2.5.2 Predicados Núcleo-Duro para Qualquer Função Unidirecional

2.5.3 Funções Núcleo-Duro

2.6 Amplificação Eficiente de Funções Unidirecionais

2.6.1 A Construção

2.6.2 Análise

2.7 Miscelânea

Enfatizamos que os relacionamentos supramencionados entre as várias formas de funções unidirecionais são os únicos que sabidamente se verificam. Especificamente:

- Funções unidirecionais fracas (respectivamente, permutações (respectivamente, com alçapão)) podem ser transformadas em funções unidirecionais fracas (respectivamente, permutações (respectivamente, com alçapão)). A outra direção é trivial.
- Dificuldade não-uniforme implica em dificuldade uniforme, mas não o contrário.
- Permutações com alçapão são casos especiais de permutações unidirecionais, que por sua vez são casos especiais de funções unidirecionais. Não sabemos se é possível transformar funções unidirecionais arbitrárias em permutações unidirecionais ou essas últimas em permutações com alçapão.[6] Evidência para o contrário tem sido apresentada ([140] e [133], respectivamente, onde é mostrado que é improvável que reduções "caixa-preta" forneçam tais transformações).
- Coleções de pares de funções sem-garra (respectivamente, permutações) dão origem a coleções de funções unidirecionais, mas a outra direção não é conhecida.

2.7.1 Notas Históricas

As noções de função unidirecional e de permutação com alçapão têm origem no artigo seminal de Diffie e Hellmann [63]. Funções unidirecionais foram introduzidas por Yao

[6] Mencionamos que funções com alçapão (nas quais, dado o alçapão, pode-se recuperar alguma pré-imagem) podem ser construídas a partir de funções unidirecionais arbitrárias (cf. [8]), mas o número de pré-imagens de cada imagem da função construída é exponencial.

[210]. A função RSA foi introduzida por Rivest, Shamir e Adleman [191], enquanto que elevar-ao-quadrado módulo um composto foi sugerida e estudada por Rabin [187]. Outros autores têm sugerido basear funções unidirecionais na suposta intratabilidade da decodificação de códigos lineares aleatórios [24,108] e no problema da soma-de-subconjuntos [132].

A equivalência da existência de funções unidirecionais fracas e fortes (i.e., Teorema 2.3.2) está implícita no trabalho de Yao [210], com a primeira prova aparecendo em [91]. A amplificação eficiente de funções unidirecionais apresentada na Seção 2.6 é tirada de Goldreich et al. [104], que por sua vez usa uma ferramenta técnica originada em [4] (veja também [55,135]). A existência de funções unidirecionais universais é enunciada no trabalho de Levin [150].

O conceito de predicados núcleo-duro tem origem no trabalho de Blum e Micali [36]. Eles também provaram que um predicado específico constitui um núcleo-duro para a "função DLP" (i.e., exponenciação em um corpo finito), desde que essa última função seja unidirecional. Consequentemente, Yao mostrou como transformar qualquer função unidirecional em um predicado núcleo-duro (i.e., o resultado não é enunciado em [210], mas sim devido a apresentações orais daquele trabalho). Uma prova primeiro apareceu no trabalho de Levin [150] (veja detalhes em [114]). Entretanto, a construção de Yao, que é análoga à construção usada na prova do Teorema 2.3.2, é de pouco valor prático.

O fato de que o produto interno mod 2 é um núcleo duro para qualquer função unidirecional (da forma $g(x,r) = (f(x),r)$) foi provado por Goldreich e Levin [110]. A prova apresentada neste livro, que segue idéias que tiveram origem em [5], foi descoberta independentemente por Leonid Levin e Charles Rackoff. O melhoramento capturado pela Proposição 2.5.4 é devido a Levin [151].

O Teorema 2.5.6 (funções núcleo-duro de um número logarítmico de bits baseadas em qualquer função unidirecional) é também devida a [110]. O Lema do XOR Computacional (Lema 2.5.8) é devido a [208], mas a prova apresentada aqui é devida a Leonid Levin. (Uma construção alternativa de funções núcleo-duro é apresentada em [117].)

Predicados (e funções) núcleo-duro para coleções específicas de permutações têm sido sugeridas [36,141,5,208]. Especificamente, Alexi et al. [5] provaram que a intratabilidade da fatoração dá origem a predicados núcleo-duro para permutações induzidas pela exponenciação quadrática módulo um número composto. Uma prova mais simples e mais justa foi subsequentemente encontrada [82].

2.7.2 Sugestões para Leitura Adicional

2.7.3 Problemas Abertos

2.7.4 Exercícios

Capítulo 3

Geradores Pseudoaleatórios

Neste capítulo discutimos geradores pseudoaleatórios. Informalmente falando, tais geradores pseudoaleatórios são programas determinísticos que expandem sementes curtas aleatoriamente selecionadas para seqüências de bits "pseudoaleatórias" muito mais longas (veja a ilustração na Figura 3.1). Seqüências pseudoaleatórias são definidas como computacionalmente indistinguíveis de seqüências verdadeiramente aleatórias por meio de algoritmos eficientes. Portanto a noção de indistinguibilidade computacional (i.e., indistinguibilidade por meio de procedimentos eficientes) tem um papel pivotante em nossa discussão. Além do mais, a noção de indistinguibilidade computacional desempenha um papel chave também em capítulos subseqüentes, em particular nas discussões de encriptação segura, provas de conhecimento-zero, e protocolos criptográficos.

A teoria da pseudoaleatoriedade também é aplicada a funções, resultando na noção de funções pseudoaleatórias, que é uma ferramenta útil para muitas aplicações criptográficas.

Em adição a definições de distribuições pseudoaleatórias, geradores pseudoaleatórios, e funções pseudoaleatórias, este capítulo contém construções de geradores pseudoaleatórios (e funções pseudoaleatórias) baseadas em vários tipos de funções unidirecionais. Em particular, geradores pseudoaleatórios muito simples e eficientes são construídos baseado na existência de permutações unidirecionais. Destacamos a *técnica híbrida*, que desempenha um papel central em muitas das provas. (Para o primeiro uso e discussão adicional dessa técnica, veja a Seção 3.2.3.)

Organização. Discussões básicas, definições, e construções de geradores pseudoaleatórios aparecem nas Seções 3.1–3.4: Começamos com uma discussão motivadora (Seção 3.1), prosseguimos com uma definição geral de indistingüibilidade computacional (Seção 3.2), a seguir apresentamos e discutimos definições de geradores pseudoaleatórios (Seção 3.3), e finalmente apresentamos algumas construções simples (Seção 3.4). Construções mais gerais são discutidas na Seção 3.5. Funções pseudoaleatórias são definidas e construídas (baseadas em qualquer gerador pseudoaleatório) na Seção 3.6. Permutações pseudoaleatórias são discutidas na Seção 3.7.

Dica de Ensino. A *técnica híbrida*, primeiramente usada para mostrar que indistingüibilidade computacional é preservada sob múltiplas amostragens (Seção 3.2.3), tem um papel importante em muitas das provas que se referem a indistingüibilidade computacional. Por conseguinte, caso você escolha pular essa prova específica, não deixe de incorporar uma discussão da técnica híbrida na primeira ocasião que você a usa.

3.1 Discussão Motivadora

A natureza da aleatoriedade tem intrigado os pensadores há séculos. Acreditamos que a noção de computação, e em particular aquela da computação eficiente, provê uma boa base para se entender a natureza da aleatoriedade.

3.1.1 Abordagens Computacionais à Aleatoriedade

Uma abordagem computacional à aleatoriedade foi iniciada por Solomonov e Kolmogorov no início dos anos 1960 (e redescoberta por Chaitin no início dos anos 1970). Essa abordagem é "ontológica" por natureza. Informalmente falando, uma cadeia s é considerada *Kolmogorov-aleatória* se seu comprimento (i.e., $|s|$) é igual ao comprimento do programa mais curto que produz s. O programa mais curto pode ser considerado a "explicação" "mais simples" para o fenômeno descrito pela cadeia s. Daí a cadeia s ser considerada Kolmogorov-aleatória se ela não possui uma explicação "simples" (i.e., uma explicação que é substancialmente mais curta que $|s|$). Enfatizamos que não se pode determinar se uma dada cadeia é ou não Kolmogorov-aleatória (e, em geral, a complexidade de Kolmogorov é uma função que não pode ser computada). Além do mais, essa abordagem parece não ter qualquer aplicação à questão dos "geradores pseudoaleatórios."

Uma abordagem computacional alternativa à aleatoriedade é apresentada no restante do capítulo. Essa abordagem foi iniciada no começo dos anos 1980. Em contraste à abordagem de Kolmogorov, essa nova abordagem é behaviorista por natureza. Ao invés de considerar a "explicação" para um fenômeno, consideramos o efeito do fenômeno sobre o ambiente. Informalmente falando, uma cadeia é considerada *pseudoaleatória* se nenhum observador eficiente pode distingüí-la de uma cadeia uniformemente escolhida de mesmo comprimento. O postulado subjacente é que objetos que não podem ser diferenciados por procedimentos eficientes são considerados equivalentes, embora eles possam ser diferentes em sua natureza (e.g., podem ter complexidades (de Kolmogorov) fundamentalmente diferentes). Além do mais, a nova abordagem naturalmente leva ao conceito de gerador pseudoaleatório, que é um conceito fundamental com muitas aplicações práticas (particularmente no campo da criptografia).

3.1.2 Uma Abordagem Rigorosa aos Geradores Pseudoaleatórios

A abordagem aos geradores pseudoaleatórios apresentada neste livro se põe em contraste com a abordagem heurística que ainda é comum em discussões relativas a "geradores pseudoaleatórios" que estão sendo usados em computadores reais. A abordagem heurística considera "geradores pseudoaleatórios" como programas que produzem seqüências de bits que podem "ser aprovadas" em *alguns* testes estatísticos *específicos*. As escolhas dos testes estaítiscos aos quais esses programas são submetidos

são um tanto arbitrárias e carecem de qualquer formulação sistemática. Além do mais, é possível construir testes estatísticos eficientes que

3.2 Indistinguibilidade Computacional

Como enunciado anteriormente, o conceito de indistinguibilidade computacional é a base para nossa definição de pseudoaleatoriedade. Por conseguinte, começamos com uma definição geral e uma discussão sobre esse conceito fundamental.

O conceito de computação eficiente leva naturalmente a uma nova equivalência entre objetos: *Objetos são considerados como sendo computacionalmente equivalentes se eles não podem ser diferenciados por nenhum procedimento eficiente*. Observamos que considerar *objetos indistingüíveis* como equivalentes é uma dos paradigmas básicos tanto da ciência quanto de situações da vida real. Daí, acreditamos que a noção de indisguinbilidade é uma noção muito natural.

3.2.1 Definição

A noção de indistinguibilidade computacional é formulada de uma maneira que é padrão no campo da complexidade computacional: considerando-se objetos como seqüências de cadeias. Daí, as seqüências $\{x_n\}_{n\in N}$ e $\{y_n\}_{n\in N}$ são ditas computacionalmente indistingüíveis se nenhum procedimento eficiente pode separá-las. Em outras palavras, nenhum algoritmo D pode aceitar uma quantidade infinita de x_n's enquanto rejeita suas contrapartidas y (i.e., para todo algoritmo eficiente D e todos os n's suficientemente grandes, verifica-se que D aceita x_n sse D aceita y_n). Objetos que são computacionalmente indistingüíveis nesse sentido podem ser considerados equivalentes para todos os propósitos práticos (porque propósitos práticos são capturados por algoritmos eficientes, e eles não podem distinguir esses objetos).

A discussão acima se estende naturalmente ao cenário probabilístico. Além do mais, como veremos adiante, essa extensão dá origem a muitas conseqüências úteis. Informalmente falando, duas distribuições são chamadas de computacionalmente indistingüíveis se nenhum algoritmo eficiente pode separá-las. Dado um algoritmo eficiente D, consideramos a probabilidade de que D aceite (e.g., dá saída 1 sobre a entrada) uma cadeia tomada da primeira distribuição. Igualmente, consideramos a probabilidade de que D aceite uma cadeia tomada da segunda distribuição. Se essas duas probabilidades são próximas, dizemos que D não distingue as duas distribuições. Novamente, a formulaç— ao dessa discussão é com respeito a duas seqüências infinitas de distribuições (ao invés de com respeito a duas distribuições fixas). Tais seqüências são chamadas de agrupamentos de probabilidade.

Definição 3.2.1 (Agrupamentos de Probabilidade): *Seja I um conjunto contável de índices. Um* **agrupamento indexado por** *I é uma seqüência de variáveis aleatórias indexadas por I. A saber, qualquer $X = \{X_i\}_{i\in I}$, onde cada X_i é uma variável aleatória, é um agrupamento indexado por I.*

Usaremos $\mathbb{N}$ ou um subconjunto $\{0,1\}^*$ como o conjunto de índices. Tipicamente em nossas aplicações, um agrupamento da forma $X = \{X_n\}_{n\in\mathbb{N}}$ tem cada X_n variando sobre cadeias de comprimento $\mathrm{poly}(n)$, enquanto que um agrupamento da forma $X = \{X_w\}_{w\in\{0,1\}^*}$ terá cada X_w variando sobre cadeias de comprimento $\mathrm{poly}(|w|)$. No restante deste capítulo trataremos de agrupamentos indexados por $\mathbb{N}$, enquanto que

em outros capítulos (e.g., na definição de encriptação segura e conhecimento-zero) trataremos com agrupamentos indexados por cadeias. Para evitar confusão, apresentaremos variações da definição de indistinguibilidade computacional para cada um desses dois casos. As duas formulações podem ser unificadas se associamos os números naturais a sua representação unária (i.e., associar $\mathbb{N}$ e $\{1^n : n \in \mathbb{N}\}$).

Definição 3.2.2 (Indistinguibilidade de Tempo-Polinomial):

1. Variante para agrupamentos indexados por $\mathbb{N}$*: Dois agrupamentos* $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ *e* $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$*, são* indistingüíveis em tempo polinomial *se para todo algoritmo probabilístico de tempo-polinomial* D*, todo polinômio positivo* $p(\cdot)$*, e todos os* n*'s suficientemente grandes,*

$$|\mathsf{Pr}[D(X_n, 1^n) = 1] - \mathsf{Pr}[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

2. Variante para agrupamentos indexados por um conjunto de cadeias S*: Dois agrupamentos,* $X =^{\text{def}} \{X_w\}_{w\in S}$ *e* $Y =^{\text{def}} \{Y_w\}_{w\in S}$*, são* indistingüíveis em tempo-polinomial *se para todo algoritmo probabilístico de tempo-polinomial* D*, todo polinômio positivo* $p(\cdot)$*, e toda* $w \in S$ *suficientemente longa,*

$$|\mathsf{Pr}[D(X_w, w) = 1] - \mathsf{Pr}[D(Y_w, w) = 1]| < \frac{1}{p(|w|)}$$

Frequentemente dizemos indistinguibilidade computacional *ao invés de indistinguibilidade em tempo polinomial.*

As probabilidades na definição acima são tomadas sobre as variáveis aleatórias correspondentes X_i (ou Y_i) e os cara-ou-coroa internos do algoritmo D (que é permitido ser um algoritmo probabilístico). A segunda variante dessa definição terá um papel chave em capítulos subseqüentes, e discussão adicional sobre ela fica adiada para tais ocasiões. No restante deste capítulo, nos referimos apenas a primeira variante da definição acima. A cadeia 1^n é dada como entrada auxiliar para o algoritmo D de modo a tornar a primeira variante consistente com a segunda. Comentamos que *em casos típicos*, o comprimento de X (respectivamente, Y_n) e n são polinomialmente relacionados (i.e., $|X_n| < \mathrm{poly}(n)$ e $n < \mathrm{poly}(|X_n|)$) e além do mais pode ser computada uma da outra em $\mathrm{poly}(n)$ de tempo. Em tais casos, *dar* 1^n *como entrada auxiliar é redundante*. De fato, em todo este capítulo tipicamente omitimos essa entrada auxiliar e assumimos que n pode ser eficientemente determinado a partir de X_n.

O seguinte experimento mental pode ser instrutivo. Para cada $\alpha\{0,1\}^*$, considere a probabilidade, a partir de agora representada por $d(\alpha)$, de que o algoritmo D dá como saída 1 sobre a entrada α. Considere a esperança de d tomada sobre cada uma das coleções: Isto é, faça $d_X(n) = \mathsf{E}[d(X_n)]$ e $d_Y(n) = \mathsf{E}[d(Y_n)]$. Então X e Y são ditas indistingüíveis por D se a (função) diferença $\delta(n) \stackrel{\text{def}}{=} |d_X(n) - d_Y(n)|$ é desprezível em n. Recordemos que uma função $\mu : \mathbb{N} \to [0,1]$ é chamada de *desprezível* se para todo polinômio positivo p e todos os n's suficientemente grandes, $\mu(n) < 1/p(n)$.

Um par de exemplos pode ajudar a esclarecer a definição. Considere um algoritmo D_1 que, indenpendentemente da entrada, arremessa uma moeda (valorada 0-1) e dá como saída seu resultado. Claramente, sobre toda entrada, o algoritmo D_1 dá como saída 1 com probabilidade exatamente $\frac{1}{2}$ e portanto não distingue nenhum par de

coleções. A seguir, considere um algoritmo D_2 que dá como saída 1 se e somente se a cadeia de entrada contém mais zeros que uns. Pelo fato de D_2 poder ser implementado em tempo polinomial, segue se X e Y são indistingüíveis-em-tempo-polinomial, então a diferença $|\mathsf{Pr}[wt(X_n) < \frac{n}{2}] - |\mathsf{Pr}[wt(Y_n) < \frac{n}{2}]|$ é desprezível (em n), enquanto que $wt(\alpha)$ representa o número de uns na cadeia α. De modo semelhante, coleções indistingüíveis-em-tempo-polinomial têm que exibir o mesmo "perfil" (a menos de erro desprezível) com respeito a qualquer "estatística de cadeia" que pode ser computada em tempo polinomial. Entretanto, não é exigido que coleções indistingüíveis-em-tempo-polinomial tenham "perfis" semelhantes com respeito a quantidades que não podem ser computadas em tempo polinomial (e.g., complexidade de Kolmogorov, ou a função apresentada após a Proposição 3.2.3).

3.2.2 Relação com Proximidade Estatística

Indistinguibilidade computacional é um engrossamento de uma noção tradicional da teoria da probabilidade. Chamamos duas coleções $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ e $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$ de *estatisticamente próximos* se sua diferença estatística é desprezível, onde a *diferença estatística* (também conhecida como *distância de variação*) entre X e Y é definida pela função

$$\Delta(n) \stackrel{\text{def}}{=} \frac{1}{2} \cdot \sum_{\alpha} |\mathsf{Pr}[X_n = \alpha] - \mathsf{Pr}[Y_n = \alpha]| \tag{3.1}$$

Claramente, se as coleções X e Y são estatisticamente próximas, então elas também são indistingüíveis-em-tempo-polinomial (veja o Exercício 6). A recíproca, entretanto, não é verdadeira. Em particular:

Proposição 3.2.3: *Existe uma coleção $X = \{X_n\}_{n\in\mathbb{N}}$ tal que X não é estatisticamente próxima à coleção uniforme $U \stackrel{\text{def}}{=} \{U_n\}_{n\in\mathbb{N}}$, e mesmo assim X e U são indistingüíveis-em-tempo-polinomial. Além do mais, X_n atribui toda a sua massa de probabilidade a no máximo $2^{n/2}$ cadeias (de comprimento n).*

Recordemos que U_n é uniformemente distribuída sobre cadeias de comprimento n. Embora X e U sejam indistingüíveis-em-tempo-polinomial, pode-se definir uma função $f : \{0,1\}^* \to \{0,1\}$ tal que f tem média 1 sobre X enquanto que tem média quase 0 sobre U (e.g., $f(x) = 1$ se e somente se $\mathsf{Pr}[X = x] > 0$). Daí, X e U têm diferentes "perfis" com respeito à função f, e mesmo assim é (necessariamente) impossível computar f em tempo polinomial.

Prova: Afirmamos que para todo n suficientemente grande, existe uma variável aleatória X_n, distribuída sobre algum conjunto de no máximo $2^{n/2}$ cadeias (cada uma de comprimento n), tal que para todo circuito C_n de tamanho (i.e., número de portas) $2^{n/8}$, se verifica que

$$|\mathsf{Pr}[C_n(U_n) = 1] - \mathsf{Pr}[C_n(X_n) = 1] > 2^{-n/8} \tag{3.2}$$

A proposição segue dessa afirmação, porque diferenciadores-em-tempo-polinomial (mesmo os probabilísticos; veja o Exercício 10 (Parte 1)) dão origem a circuitos de tamanho-polinomial com intervalo diferenciador pelo menos tão grande quanto.

A afirmação acima é provada usando um argumento probabilístico. Isto é, na verdade mostramos que a maioria das distribuições de uma certa classe podem "enganar"

todos os circuitos de tamanho $2^{n/8}$. Especificamente, mostramos que se selecionarmos uniformemente um multiconjunto de $2^{n/2}$ cadeias em $\{0,1\}^n$ e supor X_n uniforme sobre esse multiconjunto, então Eq. (3.2) se verifica com probabilidade esmagadoramente alta (sobre as escolhas do multiconjunto).

Seja C_n um certo circuito com n entradas, e seja $p_n \stackrel{\text{def}}{=} \mathsf{Pr}[C_n(U_n) = 1]$. Selecionamos, independentemente e uniformemente, $2^{n/2}$ cadeias, representadas por $s_1, \ldots, s_{2^{n/2}}$, em $\{0,1\}^n$. Definimos variáveis aleatórias ζ_i's tais que $\zeta_i = C_n(s_i)$; isto é, essas variáveis aleaórias dependem das escolhas aleatórias das s_i's correspondentes. Usando o limitante de Chernoff, obtemos que

$$\mathsf{Pr}\left[\left|p_n - \frac{1}{2^{n/2}} \cdot \sum_{i=1}^{2^{n/2}} \zeta_i\right| \geq 2^{-n/8}\right] \leq 2e^{-2\cdot 2^{n/2}\cdot(2^{-n/8})^2} < 2^{-2^{n/4}} \tag{3.3}$$

Pelo fato de que existem no máximo $2^{2^{n/4}}$ circuitos diferentes de tamanho (número de portas) $2^{n/8}$, segue que existe uma seqüência $s_1, \ldots, s_{2^{n/2}} \in \{0,1\}^n$ tal que para todo circuito C_n de tamanho $2^{n/8}$ se verifica que

$$\left|\mathsf{Pr}[C_n(U_n) = 1] - \frac{\sum_{i=1}^{2^{n/2}} C_n(s_i)}{2^{n/2}}\right| < 2^{-n/8}$$

Fixando uma tal seqüência de s_i's, e supondo que X_n seja uniformemente distribuída sobre os elementos na seqüência, a afirmação segue. ■

Comentário de Alto-Nível. A Proposição 3.2.3 apresenta um par de coleções que são computacionalmente indistingüíveis, embora elas sejam estatisticamente distantes uma da outra. Uma das duas coleções não é construtível em tempo polinomial (veja a Definição 3.2.5). É interessante notar que um par de coleções construtíveis-em-tempo-polinomial que são ambas computacionalmente indistingüíveis e têm uma diferença estatística observável podem existir apenas se geradores pseudoaleatórios existem. Saltando adiante, observamos que essa condição necessária é também suficiente. (A última observação segue do fato de que geradores pseudoaleatórios dão origem a uma coleção construtível-em-tempo-polinomial que é computacionalmente indistingüível da coleção uniforme e mesmo assim estatisticamente distante dela.)

Comentário de Baixo-Nível. Um exame mais próximo da prova supra revela que todas exceto uma fração das seqüências de comprimento $2^{n/2}$ podem ser usadas para definir a variável aleatória X_n. Especificamente, a segunda desigualdade na Eq. (3.3) é uma superestimação grosseira, e um limitante superior de $2^{-2^{O(n)}} \cdot 2^{-2^{n/4}}$ na realidade se verifica. Observando que a maioria das seqüências não contêm repetições, podemos fixar uma tal seqüência. Consequentemente, X_n será uniforme sobre $2^{n/2}$ elementos distintos da seqüência.

3.2.3 Indistinguibilidade por Experimentos Repetidos

Pela Definição 3.2.2, duas coleções são consideradas computacionalmente indistingüíveis se nenhum procedimento eficiente pode separá-las baseado em uma única amostra. Agora mostramos que para coleções "eficientemente construtíveis", indistinguibilidade computacional (baseada numa única amostra) implica indistinguibilidade computacional baseada em múltiplas amostras. Começamos apresentando definições de "indistinguibilidade por múltiplas amostras" e "coleções eficientemente construtíveis."

Definição 3.2.4 (Indistinguibilidade por Amostragens Repetidas): *Duas coleções, $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ e $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$ são* **indistingüíveis por amostragem de tempo-polinomial** *se para todo algoritmo probabilístico de tempo-polinomial D, todo par de polinômios positivos $m(\cdot)$ e $p(\cdot)$, e todo n suficientemente grande,*

$$|\mathsf{Pr}[D(X_n^{(1)},\ldots,X_n^{(m(n))})=1] - \mathsf{Pr}[D(Y_n^{(1)},\ldots,Y_n^{(m(n))})=1]| < \frac{1}{p(n)}$$

onde $X_n^{(1)}$ até $X_n^{(m(n))}$ e $Y_n^{(1)}$ até $Y_n^{(m(n))}$ são variáveis aleatórias independentes com cada $X_n^{(i)}$ idêntica a X_n e cada $Y_n^{(i)}$ idêntica a Y_n.

Definição 3.2.5 (Agrupamentos Eficientemente Construtíveis): *Uma coleção $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ é dita ser* **construtível-em-tempo-polinomial** *se existe um algoritmo probabilístico de tempo-polinomial S tal que para todo n, as variáveis aleatórias $S(1^n)$ e X_n são identicamente distribuídas.*

Teorema 3.2.6: *Seja $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ e $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$ duas coleções construtíveis em tempo-polinomial, e suponha que X e Y sejam indistingüíveis em tempo-polinomial (como na Definição 3.2.2). Então X e Y são indistingüíveis por amostragens de tempo-polinomial (como na Definição 3.2.4).*

Uma formulação alternativa do Teorema 3.2.6 procede da seguinte forma. Para toda coleção $Z \stackrel{\text{def}}{=} \{Z_n\}_{n\in\mathbb{N}}$ e todo polinômio $m(\cdot)$, defina o $m(\cdot)$-produto de Z como a coleção $\{(Z_n^{(1)},\ldots,Z_n^{(m(n))})\}_{n\in\mathbb{N}}$, onde os $Z_n^{(i)}$'s são cópias independentes de Z_n. O Teorema 3.2.6 assevera que *se as coleções X e Y são indistingüíveis em tempo polinomial e cada uma é construtível em tempo polinomial, então para todo polinômio $m(\cdot)$ o $m(\cdot)$-polinômio de X e o $m(\cdot)$-produto de Y são indistingüíveis em tempo polinomial.*

O análogo informação-teórico do teorema acima é bastante óbvio: Se duas coleções são estatisticamente próximas, então seus produtos-polinomiais são estatisticamente próximos (veja o Exercício 7). Adaptar a prova ao cenário computacional requer, como de costume, um argumento de redutibilidade. Esse argumento usa, pela primeira vez neste livro, a *técnica híbrida*. A técnica híbrida desempenha um papel central na demonstração da indistinguibilidade computacional de coleções complexas. Aplicações subseqüentes da técnica híbrida envolverão mais tecnicalidades. Daí o leitor é fortemente recomendado a não pular a prova seguinte.

Prova: A prova é por um argumento de redutibilidade. Mostramos que a existência de um algoritmo eficiente que distingue as coleções X e Y usando várias amostras implica na existência de um algoritmo eficiente que distingue as coleções X e Y usando uma única amostra. A implicação é provada usando o seguinte argumento, que mais tarde será chamado de "argumento híbrido."

Suponha que, ao contrário, existe um algoritmo probabilístico de tempo-polinomial D, assim como polinômios $m(\cdot)$ e $p(\cdot)$, tais que para uma quantidade infinita de n's verifica-se que

$$\begin{aligned}\Delta(n) &\stackrel{\text{def}}{=} |\mathsf{Pr}[D(X_n^{(1)},\ldots,X_n^{(m)})=1] - \mathsf{Pr}[D(Y_n^{(1)},\ldots,Y_n^{(m)})=1]| \\ &> \frac{1}{p(n)}\end{aligned} \tag{3.4}$$

onde $m \stackrel{\text{def}}{=} m(n)$, e os $X_n^{(i)}$'s e $Y_n^{(i)}$'s são como na Definição 3.2.4. No que se segue, derivaremos uma contradição apresentando um algoritmo probabilístico de tempo-polinomial D que distingue as coleções X e Y (no sentido da Definição 3.2.2).

Para todo k, com $0 \leq k \leq m$, definimos a variável aleatória *híbrida* H_n^k como uma seqüência (de comprimento m) consistindo de k cópias independentes de X_n seguidas por $m - k$ cópias independentes de Y_n

Capítulo 4

Sistemas de Prova de Conhecimento-Zero

Neste capítulo discutimos sistemas de prova de conhecimento-zero (CZ). Informalmente falando, tais sistemas de prova têm a notável propriedade de ser convincente *e* de produzir nada (além da validade da assertiva). Em outras palavras, receber uma prova de conhecimento-zero de que uma assertiva se verifica é equivalente a ouvir de uma parte acreditada que a assertiva se verifica (veja a ilustração na Figura 4.1). O principal resultado apresentado neste capítulo é um método para construir sistemas de prova de conhecimento-zero para toda linguagem em $\mathcal{NP}$. Esse método pode ser implementado usando qualquer esquema de comprometimento-de-bit, que por sua vez pode ser implementado usando qualquer gerador pseudoaleatório. A importância desse método provém de sua generalidade, que é a chave para suas muitas aplicações. Especificamente, quase todos os enunciados que se pode desejar provar na prática podem ser codificados como afirmações relativas a pertinência em linguagens em $\mathcal{NP}$. Adicionalmente, discutimos aspectos mais avançados do conceito de conhecimento-zero e seus efeitos na aplicabilidade desse conceito.

Organização. O material básico está apresentado nas Seções 4.1 até 4.4. Em particular, começamos com a motivação (Seção 4.1), a seguir definimos e exemplificamos as noções de provas interativas (Seção 4.2) e de conhecimento-zero (Seção 4.3), e finalmente apresentamos um sistema de prova de conhecimento-zero para toda linguagem em $\mathcal{NP}$ (Seção 4.4). As seções dedicadas a tópicos avançados se seguem (veja Figura 4.2). A menos que seja enunciado diferentemente (na lista seguinte e na Figura 4.3), cada uma dessas seções avançadas podem ser lidas independentemente das outras.

- Na Seção 4.5 apresentamos alguns *resultados negativos* relativos a provas de conhecimento-zero. Esses resultados demonstram a "otimalidade" dos resultados na Seção 4.4 e motivam as variantes apresentadas nas Seções 4.6 e 4.8.

- Na Seção 4.6 apresentamos uma relaxação grande de conhecimento-zero e provamos que ela é fechada sob composição paralela (que não é o caso, em geral, para conhecimento-zero). Aqui remetemos a uma noção chamada *indistinguibi-*

lidade de testemunha, que está relacionada a *ocultação de testemunha* (também definida e discutida).

- Na Seção 4.7 definimos e discutimos *provas de conhecimento* (de conhecimento-zero).
- Na Seção 4.8 discutimos uma relaxação de provas interativas, chamadas *provas* (ou *argumentos*) *computacionalmente seguras*.
- Na Seção 4.9 apresentamos duas construções de sistemas de conhecimento-zero *número-de-rodadas-constante*. A primeira é um sistema de prova interativa, enquanto que a segunda é um sistema de argumento. A Seção 4.8.2 (discutindo perfeitamente esquemas de comprometimento de ocultação) é um pré-requisito para a primeira construção, enquanto que as Seções 4.8, 4.7, e 4.7 constituem um pré-requisito para a segunda.
- Na Seção 4.10 discutimos provas *de conhecimento-zero não-interativas*. A noção de indistinguibilidade de testemunha (definida na Seção 4.6) é um pré-requisito para os resultados apresentados na Seção 4.10.3.1.
- Na Seção 4.11 discutimos sistemas de prova *multi-provadores*.

Concluimos, como usual, com uma seção de miscelânea (Seção 4.12).

Dica de Ensino. O sistema de prova interativa para Não-Isomorfismo de Grafos (apresentado na Seção 4.2 e a prova de conhecimento-zero de Isomorfismo de Grafos (apresentada na Seção 4.3) são exemplos meramente ilustrativos. Por conseguinte, deve-se evitar analisar aqueles exemplos em detalhe.

4.1 Provas de Conhecimento-Zero: Motivação

Um problema criptográfico arquétipo consiste de prover partes mutuamente desconfiadas com a finalidade de desvendar "peças (pré-determinadas) de informação." Refere-se aos cenários nos quais as partes possuem segredos, e elas desejam revelar partes desses segredos. Os segredos são integral ou parcialmente determinados por alguma informação publicamente conhecida, e portanto faz sentido falar sobre revelar o valor correto do segredo. A questão é como permitir a verificação de partes recentemente reveladas do segredo sem desvendar outras partes do segredo. Para esclarecer a questão, vamos considerar um exemplo específico.

> Suponha que todos os usuários em um sistema mantêm *backups* de seus sistemas de arquivos, cifradas usando suas chaves secretas, em meios de armazenamento acessivel publicamente. Suponha que em um certo ponto, um usuário, chamado `Alice`, deseja revelar a um outro usuário, chamado `Bob`, o conteúdo de algum registro em um de seus arquivos (que aparece no *backup* dela). Uma "solução" trivial é `Alice` simplesmente enviar o registro (de conteúdo) a `Bob`. O problema com essa "solução" é que `Bob` não tem como verificar que `Alice` de fato o enviou o registro verdadeiro (como apareceu cifrado no *backup* público dela), em vez de simplesmente lhe enviar um registro qualquer. `Alice` poderia provar que ela enviou o registro correto simplesmente revelando a `Bob` a chave de segredo dela. Entretanto, fazendo assim revelaria a `Bob` o conteúdo de todos os seus

arquivos, o que é certamente algo que `Alice` não quer. A questão é se `Alice` pode ou não convencer `Bob` de que ela de fato revelou o registro correto sem produzir qualquer "conhecimento" adicional.

Um problema análogo pode ser fraseado formalmente da seguinte maneira. Seja f uma *permutação* unidirecional e b um predicado núcleo-duro com respeito a f. Suponha que uma parte A, tem uma cadeia x, enquanto que uma outra parte, representada por B, tem apenas $f(x)$. Além do mais, suponha que A deseja revelar $b(x)$ à parte B, sem produzir qualquer informação adicional. A "solução" trivial é deixar A enviar $b(x)$ a B, mas, como explicado anteriormente, B não terá como verificar que A realmente enviou o bit correto (e não o seu complemento). A parte A poderia de fato provar que ela enviou o bit correto (i.e., $b(x)$) enviando x também, mas revelando x a B seria muito mais do que A tinha originalmente em mente. Novamente, a questão é se A pode ou não convencer B de que ela de fato revelou o bit correto (i.e., $b(x)$), sem produzir qualquer "conhecimento" adicional.

Em geral, a questão é se *é ou não possível provar um enunciado sem produzir nada além de sua validade*. Tais provas, sempre que elas existem, são chamadas *de conhecimento-zero*, e elas têm um papel central na construção de protocolos "criptográficos."

Informalmente falando, *provas de conhecimento-zero são provas que não produzem nada* (i.e., "nenhum conhecimento") *além da validade da assertiva*. No restante desta seção introdutória, discutimos a noção de uma "prova" e um possível significado da frase "produzir nada (i.e., nenhum conhecimento) além de algo."

4.1.1 A Noção de Prova

Uma prova é o que quer que me convence.
Shimon Even, respondendo a uma pergunta de um estudante
em sua aula de algoritmos em grafos (1978)

Discutimos a noção de prova com a intenção de trazer à tona alguns de seus aspectos subjacentes.

4.1.1.1 Um Objeto Estático versus um Processo Interativo

Tradicionalmente em matemática, uma "prova" é uma seqüência *fixa* consistindo de enunciados que ou são auto-evidentes ou são derivados de enunciados anteriores através de regras auto-evidentes. Na realidade, é mais preciso substituir a frase "auto-evidente" pela frase "comumente acordado." Na verdade, no estudo formal de provas (i.e., lógica), os enunciados comumente acordados são chamados *axiomas*, enquanto que regras comumente acordadas são chamadas *regras de derivação*. Queremos enfatizar duas propriedades de provas matemáticas:

1. Provas são vistas como objetos fixos.
2. Provas são consideradas no mínimo tão fundamentais quanto suas conseqüências (i.e., os teoremas).

Entretanto, em outras áreas da atividade humana, a noção de "prova" tem uma interpretação muito mais ampla. Em particular, uma prova não é um objeto fixo, mas

sim um processo pelo qual a validade de uma assertiva é estabelecida. Por exemplo, passar por uma interrogação num tribunal pode produzir o que pode ser considerado uma prova na lei, e a falha em prover uma resposta adequada a uma afirmação de um rival é considerada uma prova em discussões filosóficas, políticas, e às vezes técnicas. Adicionalmente, em muitas situações do cotidiano, provas são consideradas secundárias (em importância) a suas conseqüências.

Para resumir, em matemática "canônica", provas têm uma natureza estática (e.g., elas são "escritas"), enquanto que em situações do cotidiano provas têm uma natureza dinâmica (i.e., elas são estabelecidas através de uma interação). Uma interpretação dinâmica da noção de prova é mais apropriada ao nosso cenário, no qual provas são usadas como ferramentas (i.e., subprotocolos) dentro de protocolos "criptográficos.". Além do mais, uma interpretação dinâmica (ao menos no sentido fraco) é essencial à não-trivialidade da noção de prova de conhecimento-zero.

4.1.1.2 Provador e Verificador

A noção de provador está implícita em todas as discussões de provas, seja em matemática ou em situações do cotidiano: O provador é a entidade (às vezes escondida ou transcendental) fornecendo a prova. Em contrapartida, a noção de verificador tende a ser mais explícita em tais discussões, que tipicamente enfatizam o *processo de verificação*, ou em outras palavras o papel do verificador. Tanto em matemática como em situações do cotidiano, provas são definidas em termos do procedimento de verificação. O procedimento de verificação é considerado como sendo relativamente simples, e a responsabilidade é colocada na parte/pessoa que fornece a prova (i.e., o provador).

A assimetria entre a complexidade da tarefa de verificação e a complexidade da tarefa de provar-teoremas é capturada pela classe de complexidade $\mathcal{NP}$, que pode ser vista como uma classe de sistemas de prova. Cada linguagem $L \in \mathcal{NP}$ tem um procedimento eficiente de verificação para provas de enunciados da forma "$x \in L$." Recordemos que cada $L \in \mathcal{NP}$ é caracterizada por uma relação reconhecível-em-tempo-polinomial R_L tal que

$$L = \{x \,:\, \exists y \text{ t.q. } (x, y) \in R_L\}$$

e $(x, y) \in R_L$ somente se $|y| \leq \mathrm{poly}(|x|)$. Portanto, o procedimento de verificação para afirmações de pertinência da forma "$x \in L$" consiste em aplicar o algoritmo (de tempo-polinomial) para reconhecer R_L à afirmação (codificada por) x e uma suposta prova, representada por y. Qualquer y satisfazendo $(x, y) \in R_L$ é considerada uma *prova* de pertinência $x \in L$. Por conseguinte, enunciados corretos (i.e., $x \in L$) e somente esses têm provas nesse sistema de prova. Note que o procedimento de verificação é "fácil" (i.e., de tempo-polinomial), enquanto que chegar a provas pode pode ser "difícil" (se, de fato, $\mathcal{NP}$ não estiver contida em $\mathcal{BPP}$).

Vale a pena notar a "atitude desconfiada" para com o provador que está por trás de qualquer sistema de prova. Se o verificador confia no provador, então nenhuma prova é necessária. Daí, sempre que discutindo um sistema de prova, considera-se um cenário no qual o verificador não está confiando no provador e além do mais é cético de qualquer coisa que o provador diga.

4.1.1.3 Completude e Corretude

Duas propriedades fundamentais de um sistema de prova (i.e., um procedimento de verificação) são sua *corretude* (ou *validade*) e *completude*. A propriedade de corretude assevera que o procedimento de verificação não pode ser "enganado" para aceitar enunciados falsos. Em outras palavras, *corretude* captura a capacidade do verificador de se proteger de ser convencido de enunciados falsos (não importa o que o provador faz para enganar o verificador). Por outro lado, *completude* captura a capacidade de algum provador de convencer o verificador de enunciados verdadeiros (pertencentes a algum conjunto pré-determinado de enunciados verdadeiros). Note que ambas as propriedades são essenciais à própria noção de sistema de prova.

Observamos que nem todo conjunto de enunciados verdadeiros tem um sistema de prova "razoável" no qual cada um daqueles enunciados pode ser provado (enquanto que nenhum enunciado falso pode ser "provado"). Esse fato fundamental ganha significado preciso em resultados como o *Teorema da Incompletude de Gödel* e o *Teorema de Turing* relativo à *indecidibilidade do Problema da Parada*. Enfatizamos que neste capítulo nos restringimos à classe de conjuntos (de enunciados válidos) que realmente têm "sistemas de prova eficientes." Na verdade, a Seção 4.2 é dedicada à discussão e à formulação do conceito de "sistemas de prova eficientes." Dando um salto à frente, adiantamos que a eficiência de um sistema de prova estará associada à eficiência de seu procedimento de verificação.

4.1.2 Ganhando Conhecimento

Recordemos que motivamos provas de conhecimento-zero como provas através das quais o verificador não ganha "nenhum conhecimento" (além da validade da assertiva). O leitor pode corretamente se perguntar que conhecimento é esse e o que é um ganho em conhecimento. Quando discutindo provas de conhecimento-zero, evitamos a primeira questão (que é um tanto complexa) e tratamos a segunda questão diretamente. A saber, *sem* apresentar uma definição de conhecimento, apresentamos um caso genérico no qual é certamente justificado se dizer que nenhum conhecimento é ganho. Felizmente, essa abordagem parece bastar no que concerne à criptografia.

Para motivar a definição de conhecimento-zero, considere uma conversação entre as duas partes, `Alice` e `Bob`. Assuma primeiro que essa conversação é unidirecional; especificamente, `Alice` apenas fala, e `Bob` apenas escuta. Claramente, podemos dizer que `Alice` não ganha qualquer conhecimento da conversação. Por outro lado, `Bob` pode ou não ganhar conhecimento da conversação (dependendo do que `Alice` diz). Por exemplo, se tudo o que `Alice` diz é "$1 + 1 = 2$,' então claramente `Bob` não ganha qualquer conhecimento da conversação, porque ele já conhece aquele fato. Se, por outro lado, `Alice` revela a `Bob` uma prova de que $\mathcal{P} \neq \mathcal{NP}$, então ele certamente ganha conhecimento da conversação.

Para dar um melhor gostinho da definição, agora consideramos uma conversação entre `Alice` e `Bob` na qual `Bob` pergunta a `Alice` sobre um grafo grande (que é conhecido de ambos). Considere primeiro o caso no qual `Bob` pergunta a `Alice` se o grafo[1] é ou não euleriano. Claramente, `Bob` não ganha qualquer conhecimento da resposta de `Alice`, porque ele poderia facilmente ter determinado a resposta por si próprio (executando um procedimento em tempo-linear[2]). Por outro lado, se `Bob`

[1] Veja nota de pé de página 13.

[2] Por exemplo, baseado no teorema de Euler, que assevera que um grafo é euleriano se e somente se ele é conexo e todos os seus vértices têm grau par.

pergunta a Alice se o grafo é ou não hamiltoniano, e Alice (até certo ponto) responde essa questão, então não podemos dizer que Bob não tenha ganho qualquer conhecimento (porque não conhecemos um procedimento eficiente através do qual Bob poderia ter determinado a resposta por si próprio, e supondo $\mathcal{P} \neq \mathcal{NP}$, nenhum tal procedimento eficiente existe). Daí, dizemos que Bob *ganhou conhecimento* da interação se sua *capacidade computacional*, relativa ao grafo publicamente conhecido, *aumentou* (i.e., se após a interação ele pode facilmente computar algo que ele não poderia ter eficientemente computado antes da interação). Por outro lado, se o que quer que Bob possa eficientemente computar sobre o grafo *após interagir* com Alice ele também pode eficientemente computar *por si próprio* (a partir do grafo), então dizemos que Bob *não ganhou qualquer conhecimento* da interação. Ou seja, Bob ganha conhecimento somente se ele recebe o resultado de uma computação que é infactível para ele. A questão de como Alice poderia conduzir essa computação infactível (e.g., responder à questão de Bob se o grafo é ou não hamiltoniano) foi ignorada até agora. Fazendo um salto adiante, observamos que Alice pode ser uma mera abstração ou pode estar em poder de dicas adicionais que a habilitam a conduzir eficientemente computações que caso contrário são infactíveis (e em particular são infactíveis para Bob, que não tem essas dicas).

Conhecimento versus Informação

Desejamos enfatizar que *conhecimento* (tal como discutido aqui) é muito diferente de *informação* (no sentido da teoria da informação). Dois principais aspectos da diferença são como segue:

1. *Conhecimento* está relacionado a dificuldade computacional, enquanto que *informação* não está. Nos exemplos acima, existe uma diferença entre o conhecimento revelado no caso em que Alice responde a questões da forma "O grafo é euleriano?" e o caso em que ela responde a questões da forma "O grafo é hamiltoniano?." Do ponto de vista da teoria da informação não existe diferença entre os dois casos (i.e., em cada caso a resposta é determinada pela questão, e portanto Bob não obtém qualquer informação).

2. *Conhecimento* se relaciona principalmente com objetos conhecidos, enquanto que *informação* se relaciona principalmente com objetos sobre os quais apenas informação parcial é conhecida publicamente. Considere o caso em que Alice responde a cada questão jogando uma moeda não viciada e contando a Bob o resultado. De um ponto de vista da teoria da informação, Bob obtém de Alice informação relativa a um evento. Entretanto, dizemos que Bob não ganha qualquer conhecimento de Alice, porque ele poderia jogar moedas por si próprio.

4.2 Sistemas de Prova Interativa

Nesta seção introduzimos a noção de sistema de prova interativa e apresentamos um exemplo não-trivial de um tal sistema (especificamente para afirmações da forma "os dois grafos seguintes *não* são isomorfos"). A apresentação é direcionada para a introdução de provas interativas de conhecimento-zero. Sistemas de prova interativa são interessantes em si próprios e têm aplicações complexidade-teóricas importantes.[3]

[3] Veja as sugestões para leitura adicional no final do capítulo.

4.2.1 Definição

A definição de um sistema de prova interativa refere-se explicitamente a duas tarefas computacionais relacionadas a um sistema de prova: "produzir" uma prova e verificar a validade de uma prova. Essas tarefas são realizadas por duas partes diferentes, chamadas *provador* e *verificador*, que interagem entre si. Em alguns casos, a interação pode ser muito simples e, em particular, unidirecional (i.e., o provador envia um texto, chamado prova, ao verificador). Em geral, a interação pode ser mais complexa e pode tomar a forma do verificador interrogando o provador. Começamos definindo tal processo de interrogação.

4.2.1.1 Interação

Interação entre duas partes é definida da maneira natural. O único ponto que vale a pena notar é que a interação é parametrizada por uma entrada comum (dada a ambas as partes). No contexto de sistemas de prova interativa, a entrada comum representa o enunciado a ser provado. Primeiro definimos a noção de uma máquina interativa e a seguir a noção de interação entre duas tais máquinas. O leitor pode pular para a Seção 4.2.1.2, que introduz algumas convenções importantes (relativas a máquinas interativas), com pouca perda (se é que alguma).

Definição 4.2.1 (Uma Máquina Interativa):

- *Uma* **máquina de Turing interativa** (MTI) *é uma máquina de Turing (determinística) multi-fita. As fitas são uma* fita de entrada *apenas-leitura, uma* fita aleatória *apenas-leitura, uma* fita de trabalho *leitura-e-escrita, uma* fita de saída *apenas-escrita, um par de* fitas de comunicação*, e uma* fita de chaveamento *leitura-e-escrita consistindo de uma única célula. Uma fita de comunicação é apenas-leitura, e a outra é apenas-escrita.*

- *A cada MTI é associado um único bit* $\sigma \in \{0,1\}$*, chamado de sua* **identidade**. *Uma MTI é dita* ativa*, em uma configuração, se o conteúdo de sua fita de chaveamento é igual à identidade da máquina. Do contrário a máquina é dita* ociosa*. Enquanto inativo, o estado da máquina, as localizações de duas cabeças nas várias fitas, e os conteúdos das fitas escrevíveis da MTI não são modificados.*

- *O conteúdo da fita de entrada é chamado* **entrada***, o conteúdo da fita aleatória é chamado* entrada aleatória*, e o conteúdo da fita de saída ao término é chamado* **saída**. *O conteúdo escrito na fita de comunicação apenas-escrita durante um período (de tempo) no qual a máquina está ativa é chamado* **mensagem enviada** *naquele período. Igualmente, o conteúdo lido da fita de comunicação apenas-leitura durante um período ativo é chamado* **mensagem recebida** *(naquele período).*

  (Sem perda de generalidade, os movimentos da máquina em ambas as fitas de comunicação são em apenas uma direção, e.g., da esquerda para a direita.)

Essa definição, tomada isoladamente, parece um tanto não-intuitiva. Em particular, pode-se dizer que uma vez estando ociosa, a máquina nunca passará a ser ativa novamente. Pode-se também se perguntar qual é a finalidade de se distinguir a fita de comunicação apenas-leitura da fita de entrada (e respectivamente distinguir a fita de

comunicação apenas-escrita da fita de saída). A idéia é que nunca vamos considerar uma única máquina interativa, mas sim um par de máquinas combinadas de tal forma que algumas de suas fitas coincidem. Intuitivamente, as mensagens enviadas por uma máquina interativa são recebidas por uma segunda máquina que compartilha suas fitas de comunicação (de modo que a fita de comunicação apenas-leitura de uma máquina coincide com a fita apenas-escrita da outra máquina). A máquina ativa pode vir a ficar ociosa trocando o conteúdo da fita de chaveamento compartilhada, e quando ela faz isso, a outra máquina (tendo identidade oposta) passará a ficar ativa. A computação de um tal par de máquinas consiste das máquinas alternativamente enviando mensagens uma para a outra, baseadas na sua entrada (comum) inicial, suas entradas (distintas) aleatórias, e as mensagens que cada máquina recebeu até então.

Definição 4.2.2 (Computação Conjunta das Duas MTIs):

- *Duas máquinas interativas são ditas* **ligadas** *se elas têm identidades opostas, suas fitas de entrada coincidem, suas fitas de chaveamento coincidem, e a fita de comunicação apenas-leitura de uma máquina coincide com a fita de comunicação apenas-escrita da outra máquina, e vice-versa. Enfatizamos que as outras fitas de ambas as máquinas (i.e., a fita aleatória, a fita de trabalho, e a fita de saída) são distintas.*

- *A* **computação conjunta** *de um par ligado de MTIs, sobre uma entrada comum x, é uma seqüência de pares representando as configurações locais de ambas as máquinas. Ou seja, cada par consiste de duas cadeias, cada uma representando a configuração local de uma das máquinas. Em cada par de configurações locais, uma máquina (não necessariamente a mesma) está ativa, enquanto que a outra máquina está ociosa. O primeiro par na seqüência consiste de configurações iniciais correspondendo à entrada comum x, com o conteúdo da fita de chaveamento zerada.*

- *Se uma máquina pára enquanto a fita de chaveamento ainda mantém sua identidade, então dizemos que ambas as máquinas pararam. As* **saídas** *de ambas as máquinas são determinadas nesse momento.*

Nesse ponto, o leitor pode fazer objeção a essa definição, dizendo que as máquinas individualmente estão desprovidas de entradas locais individuais (enquanto que elas dispõem de fitas individuais e não-compartilhadas). Essa restrição é removida na Seção 4.2.4, e na verdade permitir entradas locais individuais (em adição à entrada comum compartilhada) é um tanto importante (ao menos no que tange aos propósitos práticos). Mesmo assim, para uma primeira apresentação de provas interativas, assim como para demonstrar o poder desse conceito, preferimos a definição mais simples acima. Por outro lado, a convenção de fitas aleatórias individuais é essencial para o poder de provas interativas (veja o Exercício 4).

4.2.1.2 Convenções Relativas às Máquinas Interativas

Tipicamente, consideramos execuções nas quais o conteúdo da fita aleatória de cada máquina é uniforme e independentemente escolhida (entre todas as seqüências infinitas de bits). A convenção de se ter uma seqüência infinita de cara-ou-coroa internos não deve incomodar o leitor, porque durante uma computação finita apenas um prefixo

finito é lido (e importa). O conteúdo de cada uma dessas fitas aleatórias pode ser visto como cara-ou-coroa internos da máquina correspondente (como na definição de máquinas probabilísticas ordinárias apresentadas no Capítulo 1). Portanto, máquinas interativas são na verdade probabilísticas.

Notação. Sejam A e B um par ligado de MTIs, e suponha que todas as possíveis interações de A e B sobre cada entrada comum terminam em um número finito de passos. Designamos por $(A, B)(x)$ a variável aleatória representando a saída local de B quando interagindo com a máquina A sobre a entrada comum x, quando a entrada aleatória para cada máquina é uniforme e independentemente escolhida. (De fato, essa definição é assimétrica, pois ela considera apenas as saídas de B.)

Uma outra convenção importante é considerar a complexidade-de-tempo de uma máquina interativa como uma função de apenas o comprimento de sua entrada.

Definição 4.2.3 (A Complexidade de uma Máquina Interativa): *Dizemos que uma máquina interativa A tem* **complexidade-de-tempo** $t : \mathbb{N} \to \mathbb{N}$ *se para toda máquina interativa B e toda cadeia x, verifica-se que quando interagindo com a máquina B, sobre a entrada comum x, a máquina A sempre (i.e., independente do conteúdo de sua fita aleatória e da fita aleatória de B) pára em $t(|x|)$ passos. Em particular, dizemos que A é de* **tempo-polinomial** *se existe um polinômio p tal que A tem complexidade-de-tempo p.*

Enfatizamos que a complexidade-de-tempo, assim definida, é independente do conteúdo das mensagens que a máquina A recebe. Em outras palavras, é um limitantee superior que se verifica para todas as possíveis mensagens que chegam (assim como todos os cara-ou-coroa internos). Em particular, uma máquina interativa com complexidade-de-tempo $t(\cdot)$ pode ler, sobre a entrada x, apenas um prefixo do comprimento total $t(|x|)$ das mensagens enviadas a ela.

4.2.1.3 Sistemas de Prova

Em geral, sistemas de prova são definidos em termos do procedimento de verificação (que pode ser visto como uma entidade, chamada verificador). Uma "prova" para uma afirmação específica é sempre considerada como proveniente do exterior (que pode ser visto como uma outra entidade, chamada provador). O procedimento de verificação por si só não gera "provas," mas simplesmente verifica sua validade. Sistemas de prova interativa têm o propósito de capturar o que quer que possa ser verificado eficientemente via interação com o exterior. Em geral, a interação com o exterior pode ser muito complexa e pode consistir de muitas trocas de mensagens, desde que o tempo total gasto pelo verificador seja polinomial (na entrada comum).

Nossa escolha de considerar verificadores probabilísticos de tempo-polinomial é justificada pela associação de procedimentos eficientes com algoritmos probabilísticos de tempo-polinomial. Além do mais, o veredito do verificador de aceitar ou rejeitar a afirmação é probabilístico, e uma probabilidade de erro limitada é permitida. (Saltando adiante, mencionamos que o erro pode ser diminuído até ser desprezível repetindo-se o procedimento de verificação um número de vezes suficientemente.)

Informalmente falando, requeremos que o provador seja capaz de convencer o verificador da validade de enunciados verdadeiro, enquanto que ninguém possa induzir o verificador a acreditar em enunciados falsos. Ambas as condições recebem uma interpretação probabilística: É requerido que o verificador aceite enunciados válidos

com probabilidade "alta", enquanto que a probabilidade de que ele aceitará um enunciado falso é "baixa" (independente da máquina com a qual o verificador interage). Na definição seguinte, a saída do verificador é interpretada como sua decisão sobre se aceita ou rejeita a entrada comum. Saída 1 é interpretada como "aceita,", enquanto que saída 0 é interpretada como "rejeita."

Definição 4.2.4 (Sistema de Prova Interativa): *Um par de máquinas interativas (P, V) é chamado* **sistema de prova interativa** *para uma linguagem L se a máquina V é de tempo-polinomial e as duas condições seguintes se verificam:*

- Completude: *Para toda $x \in L$,*

$$\mathsf{Pr}[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$$

- Corretude: *Para toda $x \notin L$ e toda máquina interativa B,*

$$\mathsf{Pr}[\langle B, V\rangle(x) = 1] \leq \frac{1}{3}$$

Algumas observações são necessárias. Primeiro enfatizamos que a condição de corretude se refere a todos os "provadores" potenciais, enquanto que a condição de completude se refere apenas ao provador prescrito P. Segundo, ao verificador é requerido que seja uma máquina (probabilística) de tempo-polinomial, mas nenhum limitante de recursos são colocados no poder de computação do provador (em nenhuma das condições de completude ou corretude). Terceiro, como no caso de $\mathcal{BPP}$, a probabilidade de erro na definição acima pode ser tornada exponencialmente pequena repetindo-se a interação muitas (polinomialmente) vezes.

Toda linguagem em $\mathcal{NP}$ tem um sistema de prova interativa. Especificamente, seja $L \in \mathcal{NP}$, e seja R_L uma relação testemunha associada à linguagem L (i.e., R_L é reconhecível em tempo polinomial, e L é igual ao conjunto $\{x : \exists \text{ t.q } |y| = \text{poly}(|x|) \wedge (x, y) \in R_L\}$). Então uma prova interativa para a linguagem L consiste de um provador que sobre a entrada $x \in L$ envia uma testemunha y (tal qual anteriormente), e um verificador que ao receber y (sobre a entrada comum x) dá como saída 1 se $|y| = \text{poly}(|x|)$ e $(x, y) \in R_L$) (e dá como saída 0 caso contrário). Claramente, quando interagindo com o provador prescrito, esse verificador sempre aceitará entradas na linguagem. Por outro lado, independente do que um provador "enganador" faça, esse verificador nunca aceitará entradas que não estão na linguagem. Destacamos que nesse sistema de prova específico, ambas as partes são determinísticas (i.e., não fazem uso de suas fitas aleatórias). É fácil ver que apenas linguagens em $\mathcal{NP}$ têm sistemas de prova interativa nos quais ambas as partes são determinísticas (veja o Exercício 2).

Em outras palavras, $\mathcal{NP}$ pode ser vista como uma classe de sistemas de prova interativa na qual a interação é unidirecional (i.e., do provador para o verificador) e o verificador é determinístico (e nunca erra). Em provas interativas gerais, *ambas* restrições são removidas: A interação é bidirecional, e o verificador é probabilístico (e pode errar, com uma certa probabilidade pequena). Tanto interação bidirecional quanto aleatorização parecem essenciais para o poder de sistemas de prova interativa (veja o Exercício 2).

Definição 4.2.5. (A Classe $\mathcal{IP}$): *A classe $\mathcal{IP}$ consiste de todas as linguagens que têm sistemas de prova interativa.*

Pela discussão acima, $\mathcal{NP} \subseteq \mathcal{IP}$. Pelo fato de que linguagens em $\mathcal{BPP}$ podem ser vistas como cada uma tendo um verificador que decide a pertinência sem qualquer interação, segue que $\mathcal{BPP} \cup \mathcal{NP} \subseteq \mathcal{IP}$. Lembramos o leitor que não se sabe se $\mathcal{BPP} \subseteq \mathcal{NP}$ ou não.

A seguir mostramos que a definição da classe $\mathcal{IP}$ permanece invariante se substituirmos os limitantes (constantes) nas condições de completude e corretude por duas funções $\mathsf{c}, \mathsf{s} : \mathbb{N} \to [0,1]$ satisfazendo $\mathsf{c}(n) < 1 - 2^{-\mathrm{poly}(n)}$, $\mathsf{s}(n) > 2^{-\mathrm{poly}(n)}$, $\mathsf{c}(n) > \mathsf{s}(n) + \frac{1}{\mathrm{poly}(n)}$. A saber, consideramos a seguinte generalização da Definição 4.2.4.

Definição 4.2.6 (Prova Interativa Generalizada): *Sejam $\mathsf{c}, \mathsf{s} : \mathbb{N} \to \mathbb{R}$ funções satisfazendo $\mathsf{c}(n) > \mathsf{s}(n) + \frac{1}{\mathrm{poly}(n)}$ para algum polinômio $p(\cdot)$. Um par interativo (P, V) é chamado sistema de prova interativa (generalizada) para a linguagem L, com* **limitante de completude** $\mathsf{c}(\cdot)$ *e* **limitante de corretude** $\mathsf{s}(\cdot)$*, se*

- completude (modificada): *para toda $x \in L$,*

$$\mathsf{Pr}[\langle P, V\rangle(x) = 1] \geq \mathsf{c}(|x|)$$

- corretude (modificada): *para toda $x \notin L$ e toda máquina interativa B,*

$$\mathsf{Pr}[\langle B, V\rangle(x) = 1] \leq \mathsf{s}(|x|)$$

A função $\mathsf{g}(\cdot)$ definida como $\mathsf{g}(n) \stackrel{\mathrm{def}}{=} \mathsf{c}(n) - \mathsf{s}(n)$ é chamada intervalo de aceitação *de (P, V), e a função $\mathsf{e}(\cdot)$ definida como $\mathsf{e}(n) \stackrel{\mathrm{def}}{=} \max\{1 - \mathsf{c}(n), \mathsf{s}(n)\}$, é chamada* probabilidade de erro *de (P, V). Em particular, s é o* erro de corretude *de (P, V), e $1 - \mathsf{c}$ é seu* erro de completude*.*

Enfatizamos que c é um limitante inferior, enquanto que s é um limitante superior.

Proposição 4.2.7: *As três condições seguintes são equivalentes:*

1. *$L \in \mathcal{IP}$. A saber, existe um sistema de prova interativa, com limitante de completude $\frac{2}{3}$ e limitante de corretude $\frac{1}{3}$, para a linguagem L.*
2. *L* tem sistemas de prova interativa muito fortes*: Para todo polinômio $p(\cdot)$ existe um sistema de prova interativa para a linguagem L, com probabilidade de erro limitada acima por $2^{-p(\cdot)}$.*
3. *L* tem uma prova interativa muito fraca: *Existe um polinômio $p(\cdot)$ e um sistema de prova interativa generalizada para a linguagem L, com intervalo de aceitação limitada abaixo por $1/p(\cdot)$. Além do mais, limitantes de completude e corretude para esse sistema, a saber, os valores $\mathsf{c}(n)$ e $\mathsf{s}(n)$, podem ser computados em tempo polinomial em n.*

Claramente, qualquer um dos dois primeiros itens implica no terceiro (incluindo os requisitos de limitantes eficientemente computáveis). A capacidade de computar eficientemente limitantes de completude e corretude é usada na prova da direção (não-trivial) oposta. A prova é deixada como um exercício (i.e., Exercício 1).

4.2.2 Um Exemplo (Não-Isomorfismo de Grafo em $\mathcal{IP}$)

Todos os exemplos de sistemas de prova interativa apresentados até agora têm sido degenerados (e.g., a interação, se alguma, tem sido unidirecional). Agora apresentamos um exemplo de um sistema de prova interativa não-degenerado. Além do mais, apresentamos um sistema de prova interativa para uma linguagem para a qual não se sabe se está em $\mathcal{BPP} \cup \mathcal{NP}$. Especificamente, a linguagem é o conjunto de *pares de grafos não-isomorfos*, representado por NIG. A idéia subjacente a esse sistema de prova é apresentada através da seguinte estória:

> Petra von Kant afirma que a cerveja Goldstar[4] em garrafas grandes tem sabor diferente da cerveja Goldstar em garrafas pequenas. Virgílio não acredita nela. Para provar sua afirmação, Petra e Virgílio repetem o seguinte processo um número de vezes suficiente para convencer Virgílio além de dúvida razoável.
>
> Virgílio seleciona aleatoriamente uma garrafa pequena ou grande e põe um pouco de cerveja num copo, sem Petra ver que garrafa ele usa. Virgílio então entrega a Petra o copo e lhe pede para dizer qual das garrafas ele usou.
>
> Se Petra nunca erra em suas respostas, então Virgílio é convencido da validade de sua afirmação. (Na verdade, ele deveria ser convencido mesmo se ela responde corretamente com probabilidade substancialmente maior que 50%, porque se a cerveja tem o mesmo sabor independente da garrafa, então não haveria maneira de Petra adivinhar corretamente com probabilidade maior que 50% que garrafa foi usada.)

Agora voltamos à exposição formal. Vamos primeiro definir a linguagem em foco: Dois grafos,[5] $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, são chamados *isomorfos* se existe um mapeamento 1-1 e sobrejetor, π, do conjunto de vértices V_1 para o conjunto de vértices V_2 tal que $(u, v) \in E_1$ se e somente se $(\pi(v), \pi(u)) \in E_2$. O mapeamento π, se existe, é chamado *isomorfismo* entre os grafos. O conjunto de pares de grafos não-isomorfos é representado por NIG.

Construção 4.2.8 (Um Sistema de Prova Interativa para Não-Isomorfismo de Grafo):

- Entrada comum: *Um par de dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$. Suponha, sem perda de generalidade, que $V_1 = \{1, 2, \ldots, |V_1|\}$, e igualmente para V_2.*
- Primeiro passo do verificador (V1): *O verificador seleciona aleatoriamente um dos dois grafos de entrada e envia ao provador uma cópia isomorfa aleatória desse grafo. A saber, o verificador seleciona uniformemente $\sigma \in \{1, 2\}$ e uma permutação aleatória π do conjunto de permutações sobre o conjunto de vértices V_σ. O verificador constrói um grafo com conjunto de vértices V_σ e conjunto de arestas*
$$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_\sigma\}$$
e envia (V_σ, F) ao provador.

[4] Goldstar é uma cerveja israelense, disponível em garrafas de 330ml e 500ml. Na realidade, a estória vai lá atrás na afirmação de Atenas relativa a jarras de néctar, que foi contestada por Zeus próprio. Infelizmente, Ovídio não diz o resultado da interação deles.

[5] Veja nota de pé de página 13.

- Observação motivadora: *Se os grafos de entrada são não-isomorfos, como o provador afirma, então o provador deve ser capaz de distinguir (não necessariamente por um procedimento eficiente) cópias isomorfas de um grafo de cópias isomorfas do outro grafo. Entretanto, se os grafos de entrada são isomorfos, então uma cópia isomorfa aleatória de um grafo será distribuída identicamente a uma cópia isomorfa aleatória do outro grafo.*

- Passo final do provador (P1): *Ao receber um grafo $G' = (V', E')$ do verificador, o provador encontra $\tau \in \{1, 2\}$ tal que o grafo G' é isomorfo ao grafo de entrada G_τ. (Se ambos $\tau = 1$ e $r = 2$ satisfazem a condição, então τ é selecionado arbitrariamente. NO caso em que nenhum $\tau \in \{1, 2\}$ satifaz a condição, coloca-se 0 em τ.) O provador envia τ ao verificador.*

- Segundo passo do verificador (V2): *Se a mensagem τ recebida do provador é igual a σ (escolhido no Passo V1), então o verificador dá como saída 1 (i.e., aceita a entrada comum). Caso contrário o verificador dá como saída 0 (i.e, rejeita a entrada comum).*

Esse programa do verificador é facilmente implementado em tempo polinomial probabilístico. Não conhecemos uma implementação probabilística de tempo-polinomial do programa do provador, mas isso não é requerido. Deveremos agora mostrar que o par de máquinas interativas acima constitue um sistema de prova interativa (no sentido geral) para a linguagem NIG (Não-Isomorfismo de Grafo).

Proposição 4.2.9: *A linguagem NIG pertence à classe $\mathcal{IP}$. Além do mais, os programas especificados na Construção 4.2.8 constituem um sistema de prova interativa generalizada para NIG, com limitante de completude 1 e limitante de corretude $\frac{1}{2}$. A saber:*

1. *Se G_1 e G_2 não são isomorfos (i.e., $(G_1, G_2) \in NIG$), então o verificador sempre aceita (quando interagindo com o provador).*

2. *Se G_1 e G_2 são isomorfos (i.e., $(G_1, G_2) \notin NIG$), então independente da máquina com a qual o verificador interage, esse último rejeita a entrada com probabilidade pelo menos $\frac{1}{2}$.*

Prova: Claramente, se G_1 e G_2 não são isomorfos, então nenhum grafo pode ser isomorfo a ambos G_1 e G_2. Segue que existe um único τ tal que o grafo G' (recebido pelo provador no passo P1) é isomorfo ao grafo de entrada G_τ. Logo, τ encontrado pelo provador no passo P1 sempre é igual ao σ escolhido no Passo V1. Parte 1 segue.

Por outro lado, se G_1 e G_2 são isomorfos, então o grafo G' é isomorfo a ambos os grafos de entrada. Além do mais, mostraremos que nesse caso o grafo G' não dá qualquer informação sobre σ, e consequentemente nenhuma máquina pode (sobre a entrada G_1, G_2, e G') ajustar τ tal que igualará σ com probabilidade maior que $\frac{1}{2}$. Detalhes seguem.

Seja π uma permutação sobre o conjunto de vértices de um grafo $G = (V, E)$. Designamos por $\pi(G)$ o grafo com vértice V e conjunto de arestas $\{(\pi(u), \pi(v)) : (u, v) \in E\}$. Seja ξ uma variável aleatória uniformemente distribuída sobre $\{1, 2\}$, e seja Π uma variável aleatória uniformemente distribuída sobre o conjunto de permutações sobre V. Enfatizamos que essas duas variáveis aleatórias são independentes. Estamos

interessados na distribuição da variável aleatória $\Pi(G_\xi)$. Vamos mostrar que embora $\Pi(G_\xi)$ seja determinada pelas variáveis aleatórias Π e ξ, as variáveis aleatórias ξ e $\Pi(G_\xi)$ são estatisticamente independentes. Na verdade, mostramos o seguinte:

Afirmação 4.2.9.1: *Se os grafos G_1 e G_2 são isomorfos, então para todo grafo G' que é isomorfo a G_1 (e G_2), verifica-se que*

$$\mathsf{Pr}[\xi = 1 \mid \Pi(G_\xi) = G'] = \mathsf{Pr}[\xi = 2 \mid \Pi(G_\xi) = G'] = \frac{1}{2}$$

Prova: Primeiro afirmamos que os conjuntos $S_1 \stackrel{\text{def}}{=} \{\pi : \pi(G_1) = G'\}$ e $S_2 \stackrel{\text{def}}{=} \{\pi : \pi(G_2) = G'\}$ são da mesma cardinalidade. Isso segue da observação de que existe uma correspondência 1-1 e sobrejetora entre o conjunto S_1 e o conjunto S_2 (a correspondência é dada pelo isomorfismo entre os grafos G_1 e G_2). Logo,

$$\begin{aligned}\mathsf{Pr}[\Pi(G_\xi) = G' \mid \xi = 1] &= \mathsf{Pr}[\Pi(G_1) = G'] \\ &= \mathsf{Pr}[\Pi \in S_1] \\ &= \mathsf{Pr}[\Pi \in S_2] \\ &= \mathsf{Pr}[\Pi(G_\xi) = G' \mid \xi = 2]\end{aligned}$$

Usando a regra de Bayes, a afirmação segue. □

Intuitivamente, a Afirmação 4.2.9.1 diz que para todo par (G_1, G_2) de grafos isomorfos, a variável aleatória $\Pi(G_\xi)$ não dá qualquer informação sobre ξ, e portanto nenhum provador pode enganar o verificador de modo a aceitar com probabilidade maior que $\frac{1}{2}$. Especificamente, tomamos R como sendo um processo aleatorizado arbitrário (representando uma estratégia do provador-enganador que depende de (G_1, G_2)) que, dada mensagem do verificador no Passo V1, tenta adivinhar o valor de ξ. Então $R(\Pi(G_\xi)) = \xi$ representa o evento no qual o verificador aceita, e temos

$$\mathsf{Pr}[R(\Pi(G_\xi)) = \xi] = \sum_{G'} \mathsf{Pr}[\Pi(G_\xi) = G'] \cdot \mathsf{Pr}[R(G') = \xi \mid \Pi(G_\xi) = G']$$

Usando a Afirmação 4.2.9.1 para a terceira igualdade, temos (para qualquer G' no suporte de $\Pi(G_\xi)$):

$$\begin{aligned}\mathsf{Pr}[R(G') = \xi \mid \Pi(G_\xi) = G'] &= \sum_{v} \mathsf{Pr}[R(G') = v \;\&\; \xi = v \mid \Pi(G_\xi) = G'] \\ &= \sum_{v} \mathsf{Pr}[R(G') = v] \cdot \mathsf{Pr}[\xi = v \mid \Pi(G_\xi) = G'] \\ &= \sum_{v \in \{1,2\}} \mathsf{Pr}[R(G') = v] \cdot \mathsf{Pr}[\xi = v] \\ &= \frac{\mathsf{Pr}[(R(G') \in \{1,2\}]}{2} \\ &\leq \frac{1}{2}\end{aligned}$$

com igualdade no caso em que R sempre dá como saída um elemento do conjunto $\{1, 2\}$. Parte 2 da proposição segue. ■

Observações Relativas à Construção 4.2.8. No sistema de prova da Construção 4.2.8, o verificador *sempre* aceita entradas *na* linguagem (i.e., o erro de completude é igual a zero). O fato de que $NIG \in \mathcal{IP}$, enquanto que não se sabe se $NIG \in \mathcal{NP}$ ou não, é uma indicação do poder de interação e aleatoricidade no contexto de prova-de-teoremas. Finalmente, notamos que é essencial que o provador não conheça o resultado dos cara-ou-coroa internos do verificador. Para uma perspectiva mais ampla sobre essas questões, veja a subseção seguinte.

4.2.3 A Estrutura da Classe $\mathcal{IP}$

Dando seguimento às observações acima, discutimos brevemente vários aspectos relativos ao "poder de prova" de sistemas de prova interativa.

1. *Os limitantes de completude e corretude*: Todos os sistemas de prova apresentados neste livro por acaso têm *completude perfeita*; isto é, o verificador *sempre* aceita entradas NA linguagem (i.e., o erro de completude é igual a zero). Na verdade, pode-se *transformar* qualquer sistema de prova interativa em um sistema de prova interativa (para a mesma linguagem) no qual o verificador sempre aceita entradas na linguagem.

Por outro lado, como mostrado no Exercício 5, apenas linguagens em $\mathcal{NP}$ têm sistemas de prova interativa nos quais o verificador *sempre rejeita* entradas NÃO PERTENCENTES À linguagem (i.e., tendo erro de corretude igual a zero).

2. *A privacidade das moedas do verificador*: *provas de Arthur–Merlin* (também conhecidas como *sistemas de prova de moeda-pública*) são um caso especial de provas interativas, onde o verificador tem que enviar o resultado de qualquer moeda que ele joga (e por conseguinte não precisa enviar qualquer outra informação). Como enunciado anteriormente, o sistema de prova da Construção 4.2.8 *não* é do tipo moeda-pública. Mesmo assim pode-se *transformar* qualquer sistema de prova interativa em um sistema de prova interativa *de moeda-pública* (para a mesma linguagem), preservando a completude perfeita.

3. *Que linguagens têm sistemas de prova interativa*? (Ignoramos essa questão natural até agora.) Constata-se que toda linguagem em $\mathcal{PSPACE}$ tem um sistema de prova interativa. Na verdade,

$$\mathcal{IP} \text{ é igual a } \mathcal{PSPACE}$$

Comentamos que acredita-se que $\mathcal{PSPACE}$ é muito maior que $\mathcal{NP}$; em particular, co$\mathcal{NP} \subseteq \mathcal{PSPACE}$, enquanto que é comum se acreditar que co$\mathcal{NP} \neq \mathcal{NP}$. Também, pelo fato de que $\mathcal{PSPACE}$ ser fechado sob complementação, $\mathcal{IP}$ também o é.

4. *Sistemas de prova interativa de número-de-rodadas-constante*: A Construção 4.2.8 constitui um protocolo de número-de-rodadas-constante (i.e., um número de mensagens constante são enviadas). Diferentemente, no sistema de prova interativa genérico para $\mathcal{PSPACE}$, o número de rodadas de comunicação é polinomialmente relacionado ao comprimento da entrada. Comentamos que acredita-se que co$\mathcal{NP}$ NÃO tem provas interativas de *número-de-rodadas-constante*.

Mencionamos que qualquer linguagem tendo um sistema de prova interativa de *número-de-rodadas-constante* também tem um sistema de prova interativa de

moeda-pública no qual apenas duas mensagens são enviadas: Esse último consiste de um desafio aleatório do verificador que é respondido pelo provador. Em geral, para qualquer função $r : \mathbb{N} \to \mathbb{N}$, qualquer sistema de prova de $2r$-rodadas pode ser transformado em um sistema de provas de r-rodadas (para a mesma linguagem).

4.2.4 Ampliação do Modelo

Para propósitos que ficarão mais claros nas Seções 4.3 e 4.4, ampliamos a definição básica de um sistema de prova interativa permitindo que cada uma das partes tenha uma entrada privada (em adição à entrada comum). Informalmente falando, essas entradas são usadas para capturar informação adicional para cada uma das partes. Especificamente, ao utilizar sistemas de prova interativa como subprotocolos dentro de protocolos maiores, as entradas privadas são associadas à configuração local das máquinas antes de entrar no subprotocolo. Em particular, a entrada privada do provador pode conter informação que habilita uma implementação eficiente da tarefa do provador.

Definição 4.2.10 (Sistemas de Prova Interativa, Revisitados):

1. *Uma* máquina interativa *é definida como na Definição 4.2.1, exceto que a máquina tem uma fita leitura-escrita adicional chamada* **a fita de entrada-auxiliar**. *O conteúdo dessa fita ́chamado* entrada auxiliar.

2. *A complexidade de tal máquina interativa é ainda medida como uma função do comprimento da entrada (comum). A saber, a máquina interativa A tem* complexidade-de-tempo $t : \mathbb{N} \to \mathbb{N}$ *se para toda máquina interativa B e toda cadeia x, verifica-se que quando interagindo com a máquina B, sobre a entrada comum x, a máquina A sempre (i.e., independente do conteúdo de sua fita aleatória e de sua fita de entrada-auxiliar), assim como do conteúdo das fitas de B) pára dentro de $t(|x|)$ passos.*

3. *Designamos por $(A(y), B(z))(x)$ a variável aleatória representando a saída (local) de B quando interagindo com a máquina A sobre a entrada comum x, quando a entrada aleatória para cada máquina é uniforme e independentemente escolhida, e A (respectivamente, B) tem entrada auxiliar y (respectivamente, z).*

4. *Um par de máquinas interativas (P, V) é chamado de um* sistema de prova interativa para uma linguagem L *se a máquina V é de tempo-polinomial e as duas condições seguintes se verificam:*

   - Completude*: Para toda $x \in L$, existe uma cadeia y tal que para toda $z \in \{0,1\}^*$,*
   $$\mathsf{Pr}[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$$
   - Corretude*: Para toda $x \notin L$, toda máquina interativa B, e todas $y, z \in \{0,1\}^*$,*
   $$\mathsf{Pr}[\langle B(y), V(z)\rangle(x) = 1] \leq \frac{1}{3}$$

Enfatizamos que ao dizer que uma máquina interativa é de tempo-polinomial, queremos dizer que seu tempo de execução é polinomial no comprimento da entrada comum. Consequentemente, não é garantido que tal máquina tenha tempo suficiente para ler sua entrada auxiliar por inteiro.

Dica de Ensino. O modelo ampliado de provas interativas é o primeiro usado neste livro na Seção 4.3.3, onde a noção de conhecimento-zero é estendida para lidar com informação a priori que o verificador possa ter. Pode-se portanto preferir à atual Definição 4.2.10 após apresentar as definições básicas de conhecimento-zero, ou seja, adiar a Definição 4.2.10 para a Seção 4.3.3. (Entretanto, conceitualmente falando, a Definição 4.2.10 pertence à atual seção.)

4.3 Provas de Conhecimento-Zero: Definições

Nesta seção introduzimos a noção de sistema de prova interativa de conhecimento-zero e apresentamos um exemplo não-trivial de um tal sistema (especificamente, para afirmações da forma "os dois grafos seguintes são isomorfos").

4.3.1 Conhecimento-Zero Perfeito e Conhecimento-Zero Computacional

Informalmente falando, dizemos que um sistema de prova interativa (P, V) para uma linguagem L é de conhecimento-zero se o que quer que possa ser eficientemente computado após interagir com P sobre a entrada $x \in L$ pode ser também eficientemente computado a partir de x (sem qualquer interação). Enfatizamos que isso se verifica com respeito a qualquer maneira eficiente de interagir com P, não necessariamente a maneira definida pelo programa verificador V. Na realidade, conhecimento-zero é uma propriedade do provador prescrito P. Ela captura a robustez de P contra tentativas de ganhar conhecimento pela interação com ele. Uma maneira simples e direta de capturar a discussão informal segue.

> Seja (P, V) um sistema de prova interativa para alguma linguagem L. Dizemos que (P, V) ou, na realidade, P, é *de conhecimento-zero perfeito* se para toda máquina interativa probabiliística de tempo-polinomial V^* existe um algoritmo probabilístico (*comum*) de tempo-polinomial M^* tal que para toda $x \in L$ as duas seguintes variáveis aleatórias são identicamente distribuídas.
>
> - $(P, V^*)(x)$ (i.e., a saída da máquina interativa V^* após interagir com a máquina interativa P sobre a entrada comum x)
> - $M^*(x)$ (i.e., a saída da máquina M^* sobre a entrada x)
>
> A máquina M^* é chamada de *simulador* para a interação de V^* com P.

Enfatizamos que requeremos que *para toda* V^* interagindo com P, não simplesmente para V, existe um simulador ("perfeito") M^*. Esse simulador, embora não tendo acesso à máquina interativa P, é capaz de simular a interação de V^* com P. O fato de que tais simuladores existem significa que V^* não ganha qualquer conhecimento de P (pois a mesma saída poderia ser gerada sem qualquer acesso a P).

O Paradigma da Simulação

A discussão acima segue um paradigma definicional geral que é também usado em outros capítulos deste livro (especificamente, no Volume 2). O *paradigma de simulação* postula que o que quer que uma parte possa fazer por si só não pode ser considerado um ganho da interação com o exterior. A validade desse paradigma é evidente, desde

que tenhamos em mente que por "fazer" queremos dizer "fazer eficientemente" algo (e mais ainda se a complexidade de "fazê-lo sozinho" está firmemente relacionada à complexidade de "fazê-lo após interação com o exterior").[6] Evidentemente, a falha de prover uma simulação de uma interação com o exterior NÃO significa necessariamente que sua interação resulta em algum "ganho real" (em algum sentido intuitivo). Mesmo assim o que importa é que qualquer "ganho real" NÃO pode ocorrer sempre que somos capazes de apresentar uma simulação. Em resumo, a abordagem subjacente ao paradigma de simulação pode ser excessivamente cautelosa, mas é certamente válida. (Além do mais, no mínimo parece muito mais difícil prover uma definição robusta de "ganho real.")

Casos Triviais. Note que toda linguagem em $\mathcal{BPP}$ tem um sistema de prova de conhecimento-zero perfeito no qual o provador não faz nada (e o verificador verifica sozinho se aceita ou rejeita a entrada comum). Para demonstrar a propriedade de conhecimento-zero desse "provador postiço," pode-se apresentar para todo verificador V^* um simulador M^* que é essencialmente idêntico a V^* (exceto que as fitas de comunicação de V^*, que nunca são usadas, são consideradas como fitas de trabalho normais de M^*).

4.3.1.1 Conhecimento-Zero Perfeito

Infelizmente, a formulação precedente de conhecimento-zero (perfeito) é um pouco estrita demais (pelo menos até onde sabemos).[7] Relaxamos a formulação permitindo que o simulador deixe, com probabilidade limitada, de produzir uma interação.

Definição 4.3.1 (Conhecimento-Zero Perfeito): *Seja (P,V) um sistema de prova interativa para alguma linguagem L. Dizemos que (P,V) é de* **conhecimento-zero perfeito** *se para toda máquina interativa probabilística de tempo-polinomial V^* existe um algoritmo probabilístico de tempo-polinomial tal que para toda $x \in L$ as duas seguintes condições se verificam:*

1. *Com probabilidade no máximo $\frac{1}{2}$, sobre a entrada x, a máquina M^* dá como saída um símbolo especial representado por $\perp$ (i.e., $\mathsf{Pr}[M^*(x) = \perp] \leq \frac{1}{2}$).*

2. *Seja $m^*(x)$ uma variável aleatória descrevendo a distribuição de $M^*(x)$ condicionada a $M^*(x) \neq \perp$ (i.e., $\mathsf{Pr}[m^*(x) = \alpha] = \mathsf{Pr}[M^*(x) = \alpha \mid M^*(x) \neq \perp]$ para toda $\alpha \in \{0,1\}^*$). Então as seguintes variáveis aleatórias são identicamente distribuídas:*

   - *$(P,V^*)(x)$ (i.e., a saída da máquina interativa V^* após interagir com a máquina interativa P sobre a entrada comum x)*
   - *$m^*(x)$ (i.e., a saída da máquina M^* sobre a entrada x, condicionada a não ser $\perp$)*

   A máquina M^* é chamada **simulador perfeito** *para a interaçãoA de V^* com P.*

[6] Veja a discussão de firmeza de conhecimento na Seção 4.4.4.2.

[7] Isto é, não conhecemos nenhum caso não-trivial no qual aquele requisito seja satisfeito. Por outro lado, casos não-triviais satisfazendo a definição relaxada dada a seguir são conhecidos, e na realidade apresentamos uma (i.e., uma prova de conhecimento-zero perfeito para Isomorfismo de Grafo).

A condição 1 pode ser substituída por uma condição mais forte requerendo que M^* dê como saída o símbolo especial (i.e., $\perp$) apenas com probabilidade desprezível. Por exemplo, pode-ser requerer que (sobre a entrada x) a máquina M^* dará como saída $\perp$ com probabilidade limitada acima por $2^{-p(|x|)}$, para qualquer polinômio $p(\cdot)$; veja o Exercício 6. Consequentemente, a diferença estatística entre as variáveis aleatórias $(P, V^*)(x)$ e $M^*(x)$ pode ser tornadas desprezíveis (em $|x|$); veja o Exercício 8. Logo, o que quer que o verificador eficiente compute após interagir com o provador pode ser eficientemente computado (com apenas um erro extremamente pequeno) pelo simulador (e portanto pelo verificador propriamente dito).

4.3.1.2 Conhecimento-Zero Computacional

Seguindo o espírito do Capítulo 3, observamos que para propósitos práticos não há necessidade de ser capaz de "simular perfeitamente" a saída de V^* depois que ela interage com P. Ao invés disso, basta gerar uma distribuição de probabilidade que seja computacionalmente indistingüível da saída de V^* depois que ela interage com P. A relaxação é consistente com nosso requisito original de que "o que quer que seja eficientemente computado após interagir com P sobre a entrada $x \in L$ pode ser também eficientemente computado a partir de x (sem qualquer interação)," a razão sendo que consideramos coleções computacionalmente indistingüíveis como sendo o mesmo. Antes de apresentar a definição relaxada de conhecimento-zero geral, recordamos a definição de coleções computacionalmente indistingüíveis (veja o Item 2 na Definição 3.2.2). Aqui consideramos coleções indexadas por cadeias de uma linguagem L. Dizemos que as coleções $\{R_x\}_{x\in L}$ e $\{S_x\}_{x\in L}$ são **computacionalmente indistingüíveis** se para todo algoritmo probabilístico de tempo-polinomial D, para todo polinômio $p(\cdot)$, e para toda $x \in L$ suficientemente longa, verifica-se que

$$|\mathsf{Pr}[D(x, R_x) = 1] - \mathsf{Pr}[D(x, S_x) = 1]| < \frac{1}{p(|x|)}$$

Definição 4.3.2 (Conhecimento-Zero Computacional): *Seja (P, V) um sistema de prova interativa para alguma linguagem L. Dizemos que (P, V) é de* **conhecimento-zero computacional** *(ou simplesmente de* **conhecimento-zero***) se para toda máquina interativa probabilística de tempo-polinomial V^* existe um algoritmo probabilístico de tempo-polinomial M^* tal que as duas seguintes coleções são computacionalmente indistingüíveis:*

- *$\{\langle P, V^*\rangle(x)\}_{x\in L}$ (i.e., a saída da máquina interativa V^* depois que ela interage com a máquina interativa P sobre a entrada comum x)*
- *$\{M^*(x)\}_{x\in L}$ (i.e., a saída da máquina M^* sobre a entrada x)*

A máquina M^* é chamada simulador *para a interação de V^* com P.*

O leitor pode facilmente verificar (veja o Exercício 9) que permitir ao simulador dar como saída o símbolo especial $\perp$ (com probabilidade limitada acima por, digamos, $\frac{1}{2}$) e considerar a distribuição condicional da saída (como foi feito na Definição 4.3.1) não aumenta o poder da Definição 4.3.2.

O Escopo de Conhecimento-Zero. Enfatizamos que ambas definições de conhecimento-zero se aplicam a sistemas de prova interativa no sentido geral (i.e., tendo qualquer intervalo notável entre as probabilidades de aceitação para entradas dentro e fora da linguagem). Na verdade, as definições de conhecimento-zero se aplicam a qualquer par de máquinas interativas (na realidade a cada máquina interativa): A saber, podemos dizer que a máquina interativa A é de *conhecimento-zero sobre* L se o que quer que seja eficientemente computado após a interação com A sobre a entrada comum $x \in L$ pode também ser eficientemente computado a partir da própria x.

4.3.1.3 Uma Formulação Alternativa de Conhecimento-Zero

Uma formulação alternativa de conhecimento-zero considera a visão do verificador da interação com o provador, ao invés de apenas a saída do verificador após tal interação. Por "visão do verificador da interação" queremos dizer a seqüência inteira das configurações locais do verificador durante uma interação (execução) com o provador. Claramente, basta considerar apenas o conteúdo da fita aleatória do verificador e a seqüência de mensagens que o verificador recebeu do provador durante a execução (pois a seqüência inteira de configurações locais e a saída final são determinadas por aqueles objetos).

Definição 4.3.3 (Conhecimento-Zero, Formulação Alternativa): *Seja* (P,V)*,* L*, e* V^* *como na Definição 4.3.2. Designamos por* $\text{visão}^P_{V^*}(x)$ *uma variável aleatória descrevendo o conteúdo da fita aleatória de* V^* *e as mensagens que* V^* *recebe de* P *durante uma computação conjunta sobre a entrada comum* x*. Dizemos que* (P,V) *é de* **conhecimento-zero** *se para toda máquina interativa probabilística de tempo-polinomial* V^* *existe um algoritmo probabilístico de tempo-polinomial* M^* *tal que as coleções* $\{\text{visão}^P_{V^*}(x)\}_{x\in L}$ *e* $\{M^*(x)\}_{x\in L}$ *são computacionalmente indistingüíveis.*

Umas poucas observações são necessárias. Primeiro, note que a Definição 4.3.3 é obtida da Definição 4.3.2 substituindo-se $\langle P,V^*\rangle(x)$ por $\text{visão}^P_{V^*}(x)$. O simulador M^* usado na Definição 4.3.3 está relacionado, embora não seja igual, ao simulador usado na Definição 4.3.2 (mesmo assim esse fato não é refletido no texto daquelas definições). Claramente, $\langle P,V^*\rangle(x)$ pode ser computado em tempo polinomial (determinístico) a partir de $\text{visão}^P_{V^*}(x)$ para cada V^*. Embora isso não seja verdadeiro para a direção oposta, a Definição 4.3.3 é equivalente à Definição 4.3.2 (em virtude da quantificação universal sobre os V^*'s; veja o Exercício 10). Esse último fato justifica o uso da Definição 4.3.3, que é mais conveniente para trabalhar, embora pareça menos natural que a Definição 4.3.2. Um formulação alternativa análoga de conhecimento-zero perfeito pode ser obtida da Definição 4.3.1 e é claramente equivalente a ela.

4.3.1.4 Conhecimento-Zero (Estatístico) Quase-Perfeito

Um relaxação menos drástica (que conhecimento-zero computacional) da noção de conhecimento-zero perfeito é a seguinte:

Definição 4.3.4 (Conhecimento-Zero (Estatístico) Quase-Perfeito): *Seja* (P,V) *um sistema de prova interativa para alguma linguagem* L*. Dizemos que* (P,V) *é de* **conhecimento-zero quase-perfeito** *(ou de* **conhecimento-zero estatístico***) se para toda máquina interativa probabilística de tempo-polinomial* V^* *existe um algoritmo probabilístico de tempo-polinomial* M^* *tal que as duas seguintes coleções são estatisticamente próximas como funções de* $|x|$*:*

- $\{\langle P, V^*\rangle(x)\}_{x\in L}$ *(i.e., a saída da máquina interativa V^* depois que ela interage com a máquina interativa P sobre a entrada comum x)*
- $\{M^*(x)\}_{x\in L}$ *(i.e., a saída da máquina M^* sobre a entrada x).*

Isto é, a diferença estatística entre $\langle P, V^*\rangle(x)$ e $M^*(x)$ é desprezível em termos de $|x|$.

Como no caso de conhecimento-zero computacional, permitir ao simulador dar como saída o símbolo especial $\perp$ (com probabilidade limitada acima por, digamos, $\frac{1}{2}$) e considerar a distribuição condicional de saída (como foi feito na Definição 4.3.1) não aumenta o poder da Definição 4.3.4; veja o Exercício 8. Também é fácil mostrar que conhecimento-zero perfeito implica conhecimento-zero quase-perfeito, o que por sua vez implica conhecimento-zero computacional.

As três definições (i.e., conhecimento-zero perfeito, quase-perfeito, e computacional) correspondem a uma hierarquia natural em três-estágios de interpretações da noção de pares "próximos" de coleções de probabilidade. (Em todos os três casos, os pares de coleções sendo postulados como sendo próximos são $\{\langle P, V^*\rangle(x)\}_{x\in L}$ e $\{M^*(x)\}_{x\in L}$.)

1. A interpretação mais estrita de proximidade é o requisito de que duas coleções sejam identicamente distribuídas. Esse é o requisito no caso de conhecimento-zero perfeito.

2. Uma interpretação levemente mais relaxada de proximidade é que as duas coleções sejam estatisticamente indistingüíveis (ou estatisticamente próximas). Esse é o requisito no caso de conhecimento-zero quase-perfeito (ou estatístico).

3. Uma interpretação muito mais relaxada de proximidade, que basta para todos os propósitos práticos, é que duas coleções sejam computacionalmente indistingüíveis. Esse é o requisito no caso de conhecimento-zero computacional.

4.3.1.5 Classes de Complexidade Baseadas em Conhecimento-Zero

As várias definições de conhecimento-zero dão origem a classes de complexidade naturais:

Definição 4.3.5 (Classe de Linguagens Tendo Provas de Conhecimento-Zero): *Designamos por $\mathcal{ZK}$ (também por $\mathcal{CZK}$) a classe de linguagens tendo sistemas de prova interativa de conhecimento-zero. Igualmente, $\mathcal{PZK}$ (respectivamente, $\mathcal{SZK}$) denota a classe de linguagens tendo sistemas de prova interativa de conhecimento-zero perfeito (respectivamente, estatístico).*

Claramente

$$\mathcal{BPP} \subseteq \mathcal{PZK} \subseteq \mathcal{SZK} \subseteq \mathcal{CZK} \subseteq \mathcal{IP}$$

Assumindo a existência de funções (não-uniformemente) unidirecionais, a última inclusão é uma igualdade (i.e., $\mathcal{CZK} = \mathcal{IP}$); veja a Proposição 4.4.5 e os Teoremas 3.5.12 e 4.4.12. Por outro lado, acreditamos que a primeira e a terceira inclusões são estritas (pois igualdades em qualquer um dos casos contradizem suposições largamente acreditadas).

4.3.1.6 Simuladores de Tempo-Polinomial Esperado

A formulação de conhecimento-zero perfeito apresentada na Definição 4.3.1 é diferente da definição usada em algumas publicações anteriores na literatura. A definição original requer que o simulador *sempre* dê como saída um transcrito legal (que tem que ser distribuído identicamente à interação real), e mesmo assim permite ao simulador rodar em tempo polinomial *esperado* ao invés de estritamente tempo polinomial. Enfatizamos que a esperança é tomada sobre os cara-ou-coroa do simulador (enquanto que a entrada para o simulador é fixa). Isso produz o seguinte:

Definição 4.3.6 (Conhecimento-Zero Perfeito, Formulação Liberal): *Dizemos que* (P, V) *é de* **conhecimento-zero perfeito no sentido liberal** *se para toda máquina interativa probabilística de tempo-polinomial* V^* *existe um algoritmo de tempo-polinomial* esperado M^* *tal que para toda* $x \in L$ *as variáveis aleatórias* $\langle P, V^* \rangle(x)$ *e* $M^*(x)$ *são identicamente distribuídas.*

Enfatizamos que por *tempo polinomial probabilístico* queremos dizer um limitante estrito no tempo de execução em todas as possíveis execuções, enquanto que por *tempo polinomial esperado* permitimos execuções não-polinomiais mas requeremos que o tempo de execução seja "polinomial na média." Claramente, a Definição 4.3.1 implica a Definição 4.3.6 (veja o Exercício 7). Curiosamente, existem provas interativas que são de conhecimento-zero com respeito à definição liberal mas não se sabe se são de zero-conhecimento perfeito com respeito à Definição 4.3.1. Chamamos atenção para o fato de que a forma ingênua de transformar um algoritmo probabilístico de tempo-polinomial esperado em um que rode em tempo polinomial estrito não é adequada ao presente contexto.[8]

Preferimos adotar a Definição 4.3.1, ao invés da Definição 4.3.6, porque desejamos adotar a noção de tempo polinomial esperado. A principal razão para nossa intenção de evitar essa última noção é que a correspondência entre tempo polinomial esperado e computações eficientes é mais controvertida que a associação largamente aceita de tempo polinomial estrito com computações eficientes. Além do mais, a noção de tempo polinomial esperado é mais sutil do que se imagina à primeira vista:

> A interpretação ingênua de tempo polinomial esperado é ter um tempo de execução *médio* que é *limitado por um polinômio* no comprimento da entrada. Essa definição de tempo polinomial esperado é insatisfatória porque ela *não é fechada sob reduções* e é (demasiado) *dependente-de-máquina*. Ambos fenômenos agravantes seguem do fato de que uma função pode ter uma média (digamos sobre $\{0,1\}^n$) que é limitada por um polinômio (em n) e ainda assim elevando a função ao quadrado produzirá uma função que não é limitada por um polinômio (em n). Por exemplo, a função $f(x) \stackrel{\text{def}}{=} 2^{|x|}$ se $x \in \{0\}^*$, e $f(x) \stackrel{\text{def}}{=} |x|^2$ caso contrário, satisfaz $\mathsf{E}[f(U_n)] < n^2 + 1$, mas $\mathsf{E}[f(U_n)^2] > 2^n$.

[8] A transformação ingênua trunca execuções do algoritmo (em nosso caso, o simulador) que tomam mais que t vezes o número esperado de passos. (Tal execução truncada é dita produzir alguma saída fixa.) A diferença estatística entre a distribuição de saída do algoritmo original do algoritmo modificado é no máximo $1/t$. O problema é que t tem que ser limitado por um polinômio fixo no tempo de execução, e portanto a diferença estatística não é desprezível. Para ver que a análise dessa transformação ingênua é justa, considere seu efeito sobre o seguinte algoritmo: Sobre a entrada 1^n, o algoritmo primeiro seleciona uniformemente $r \in \{0,1\}^n$, a seguir leva 2^i passos inativos, onde i é o comprimento do prefixo de zeros mais longo de r, e finalmente executa $S(1^n)$, onde S é um algoritmo probabilístico (estrito) de tempo-polinomial arbitrário.

> Portanto, uma melhor interpretação de tempo polinomial esperado é ter um tempo de execução que é *limitado por um polinômio em uma função que tem uma taxa de crescimento linear médio*. Isto é, usando a definição ingênua de linear na média, dizemos que f é *polinomial na média* se existe um polinômio p e uma função linear-na-média ℓ tal que $f(x) \leq p(\ell(x))$ para todas as x's suficientemente longas. Note que se f é polinomial na média, então f^2 também o é.

Uma discussão análoga aplica-se a conhecimento-zero computacional. Mais especificamente, a Definição 4.3.2 requer que o simulador trabalhe em tempo polinomial, enquanto que uma noção mais liberal permitiria que ele trabalhasse em tempo polinomial *esperado*.

Comentamos que em nome da elegância é comum modificar as definições que permitem simuladores de tempo-polinomial *esperado* requerendo que tais simuladores também existam para a interação de verificadores de tempo-polinomial *esperado* com o provador.

4.3.1.7 Conhecimento-Zero de Verificador-Honesto

Discutimos brevemente uma noção fraca de conhecimento-Zero. A noção, chamada *conhecimento-zero de verificador-honesto*, requer simulabilidade da visão de apenas o verificador *prescrito* (ou *honesto*), ao invés de simulabilidade da visão de *qualquer possível* verificador (probabilístico de tempo-polinomial). Embora essa noção não seja suficiente para aplicações criptográficas típicas, ela é interessante para pelo menos um par de razões: Primeiro, essa noção fraca de conhecimento-zero é altamente não-trivial e fascinante em si própria. Segundo, protocolos de moeda-pública que são de conhecimento-zero com respeito ao verificador honesto podem ser transformados em protocolos semelhantes que são de conhecimento-zero em geral. Enfatizamos que no presente contexto (do verificador prescrito único) as formulações de *simulabilidade de saída* (como na Definição 4.3.2) e *simulabilidade de visão* (como na Definição 4.3.3) NÃO são equivalentes, e é importante usar a última.[9]

Definição 4.3.7 (Conhecimento-Zero com Respeito a um Verificador Honesto): *Seja* (P, V)*,* L*, e* $\text{visão}_V^P(x)$ *tal qual na Definição 4.3.3. Dizemos que* (P, V) *é de* **conhecimento-zero de verificador-honesto** *se existe um algoritmo probabilístico de tempo-polinomial* M *tal que as coleções* $\{\text{visão}_V^P(x)\}_{x\in L}$ *e* $\{M(x)\}_{x\in L}$ *são computacionalmente indistingüíveis.*

A definição precedente se refere a conhecimento-zero computacional e é uma restrição da Definição 4.3.3. Versões para conhecimento-zero perfeito e conhecimento-zero estatístico são definidas analogamente.

4.3.2 Um Exemplo (Isomorfismo de Grafo em $\mathcal{PZK}$)

Como mencionado anteriormente, toda linguagem em $\mathcal{BPP}$ tem um sistema de prova de conhecimento-zero trivial (i.e., degenerado). Agora apresentamos um exemplo de

[9] Note que para qualquer prova interativa de completude perfeita, a saída do verificador honesto é trivialmente simulável (por um algoritmo que sempre dá como saída 1). Diferentemente, muitos dos resultados negativos apresentados na Seção 4.5 também se aplicam a conhecimento-zero com respeito a um verificador honesto, como definido a seguir. Por exemplo, apenas linguagens em $\mathcal{BPP}$ têm sistemas de prova *unidirecionais* que são de conhecimento-zero com respeito a um verificador honesto.

um sistema de prova de conhecimento-zero não-degenerado. Além do mais, apresentamos um sistema de prova de conhecimento-zero para uma linguagem para a qual não se sabe se pertence a $\mathcal{BPP}$. Especificamente, a linguagem é o conjunto de *pares de grafos isomorfos*, representada por IG (veja a definição na Seção 4.2). Novamente, a idéia subjacente a esse sistema de prova é apresentada através de uma estória:

> Nesta estória, Petra von Kant afirma que existe um caminho entre a porta norte e a porta sul de seu labirinto (i.e., um caminho entrando no labirinto). Virgílio não acredita nela. Petra está querendo provar sua afirmação a Virgílio, mas não quer lhe prover qualquer conhecimento adicional (e, em particular, não quer ajudá-lo a encontrar um caminho para dentro a partir da porta norte para a porta sul). Para provar sua afirmação, Petra e Virgílio repetem o seguinte processo um número de vezes suficiente para convencer Virgílio além de dúvida razoável.
>
> Petra milagrosamente transporta Virgílio a um local aleatório em seu labirinto. Então Virgílio pede que lhe mostre o caminho ou para a porta norte ou para a porta sul. A escolha dele supõe-se ser aleatória, mas ele pode tentar enganar. Petra então escolhe uma caminhada aleatória (suficientemente longa) a partir da localização atual até o destino desejado e guia Virgílio ao longo da caminhada.
>
> Claramente, se o labirinto tem um caminho tal qual afirmado (e Petra sabe caminhar no labirinto), então Virgílio será convencido da validade da afirmação dela. Se, por outro lado, o labirinto não tem tal caminho, então a cada iteração, com probabilidade no mínimo 50%, Virgílio irá detectar que Petra está mentindo. Finalmente, Virgílio não ganhará qualquer conhecimento da tournée guiada, a razão sendo que ele pode simular uma tournée guiada por si próprio, como segue: Primeiro, ele seleciona norte ou sul (como faz na tournée guiada) e vai para a porta apropriada (de fora do labirinto). A seguir, ele toma uma caminhada aleatória a partir da porta para dentro do labirinto enquanto vai desenrolando um carretel de linha atrás dele, e finalmente ele segue a linha de volta à porta. (Uma caminhada aleatória suficientemente longa cujo comprimento é igual ao comprimento da tournée guiada por Petra garantirá que Virgílio visitará um local aleatório no labirinto, e o caminho de volta vai parecer uma caminhada aleatória da localização no final de sua linha para a porta escolhida.)

Agora voltamos à exposição formal:

Construção 4.3.8 (Uma Prova de Conhecimento-Zero Perfeito para Isomorfismo de Grafo):

- Entrada comum: *Um par de grafos, $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$. Seja ϕ um isomorfismo entre os dois grafos de entrada; a saber, ϕ é um mapeamento 1-1 e sobrejetor do conjunto de vértices V_1 para o conjunto de vértices V_2 tal que $(u, v) \in E_1$ se e somente se $(\phi(u), \phi(v)) \in E_2$.*

- Primeiro passo do provador (P1): *O provador seleciona uma cópia isomorfa de G_2 e a envia ao verificador. A saber, o provador seleciona aleatóriamente, com distribuição uniforme de probabilidade, uma permutação π do conjunto de permutações sobre o conjunto de vértices V_2 e constrói um grafo com conjunto de vértices V_2 e conjunto de arestas*

$$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_2\}$$

O provador envia (V_2, F) ao verificador.

- Observação motivadora: *Se os grafos de entrada são isomorfos, como afirma o provador, então o grafo enviado no Passo P1 é isomorfo a ambos os grafos de entrada. Entretanto, se os grafos de entrada não são isomorfos, então nenhum grafo pode ser isomorfo a ambos.*

- Primeiro passo do verificador (V1): *Ao receber um grafo $G' = (V', E')$ do provador, o verificador pede ao provador para mostrar um isomorfismo entre G' e um dos grafos de entrada, escolhido aleatoriamente, pelo verificador. A saber, o verificador seleciona uniformemente $\sigma \in \{1, 2\}$ e o envia ao provador (que supõe-se que deve responder com um isomorfismo entre G_σ e G').*

- Segundo passo do provador (P2): *Se a mensagem σ recebida do verificador é igual a 2, então o provador envia π ao verificador. Caso contrário (i.e., $\sigma \neq 2$), o provador envia $\pi \circ \phi$ (i.e., a composição de π com ϕ, definida como $\pi \circ \phi(v) \stackrel{\text{def}}{=} \pi(\phi(v))$) ao verificador. (Observação: O provador trata qualquer $\sigma \neq 2$ como $\sigma = 1$.)*

- Segundo passo do verificador (V2): *Se a mensagem, representada por ψ, recebida do provador é um isomorfismo entre G_σ e G', então o verificador dá como saída 1; caso contrário, ele dá como saída 0.*

Vamos representar por P_{IG} o programa do provador.

O programa do verificador que acaba de ser apresentado é facilmente implementado em tempo polinomial probabilístico. No caso em que o provador recebe um isomorfismo entre os grafos de entrada como entrada auxiliar, o programa do provador pode também ser implementado em tempo polinomial probabilístico. Agora mostramos que esse par de máquinas interativas constitui um sistema de prova interativa de *conhecimento-zero* (no sentido geral) para a linguagem IG (Isomorfismo de Grafo).

Proposição 4.3.9: *A linguagem IG tem um sistema de prova interativa de conhecimento-zero perfeito. Além do mais, os programas especificados na Construção 4.3.8 satisfazem o seguinte:*

1. *Se G_1 e G_2 são isomorfos (i.e., $(G_1, G_2) \in IG$, então o verificador sempre aceita (quando interagindo com P_{IG}).*

2. *Se G_1 e G_2 não são isomorfos (i.e., $(G_1, G_2) \notin IG$, então não importa com qual máquina o verificador interage, ele rejeitará a entrada com probabilidade pelo menos $\frac{1}{2}$.*

3. *O provador (i.e., P_{IG}) é de conhecimento-zero perfeito. A saber, para toda máquina probabilística interativa de tempo-polinomial V^*, existe um algoritmo probabilístico de tempo-polinomial M^* dando como saída $\perp$ com probabilidade no máximo $\frac{1}{2}$, de modo que para toda $x \stackrel{\text{def}}{=} (G_1, G_2) \in IG$, as duas variáveis aleatórias seguintes são identicamente distribuídas:*

   - $\text{visão}_{V^*}^{P_{IG}}(x)$ *(i.e., a visão de V^* após interagir com P_{IG}, sobre a entrada comum x)*
   - $m^*(x)$ *(i.e., a saída da máquina m^*, sobre a entrada x, condicionada a não ser $\perp$).*

Um sistema de prova interativa de conhecimento-zero para IG com probabilidade de erro 2^{-k} (apenas na condição de corretude) pode ser derivada executando o protocolo acima, *sequencialmente*, k vezes. Enfatizamos que em cada repetição do protocolo, ambos provador (prescrito) e verificador têm que usar cara-ou-coroa "frescos" que são independentes dos cara-ou-coroa usados em repetições anteriores do protocolo. Para discussões adicionais, veja a Seção 4.3.4. Observamos que k execuções *paralelas* também decrementarão o erro na condição de corretude para 2^{-k}, mas a prova interativa resultante não se sabe se é de conhecimento-zero no caso em que k cresce mais rapidamente que logaritmicamente no comprimento da entrada. Na verdade, acreditamos que tal prova interativa *não* é de conhecimento-zero. Para discussões adicionais, veja a Seção 4.5.

Enfatizamos que não se sabe se $IG \in \mathcal{BPP}$ ou não. Daí, a Proposição 4.3.9 assevera a existência de uma prova de conhecimento-zero perfeito para uma linguagem para a qual não se sabe se pertence a $\mathcal{BPP}$.

Prova: Primeiro mostramos que esses programas de fato constituem um sistema de prova interativa (geral) para IG. Claramente, se os grafos de entrada G_1 e G_2 são isomorfos, então o grafo G' construído no Passo P1 será isomorfos a ambos. Daí, se cada parte segue seu programa prescrito, então o verificador sempre aceitará (i.e., dará como saída 1). A parte 1 segue. Por outro lado, se G_1 e G_2 não são isomorfos, então nenhum grafo pode ser isomorfo a ambos G_1 e G_2. Segue que não importa como o provador (possivelmente enganando) constrói G', existe $\sigma \in \{1,2\}$ tal que G' e G_σ *não* são isomorfos. Daí, se o verificador segue seu programa, então ele rejeitará (i.e., dará como saída 0) com probabilidade no mínimo $\frac{1}{2}$. A parte 2 segue.

Resta mostrar que P_{IG} é de fato de conhecimento-zero perfeito sobre IG. Isso é de fato a parte difícil da prova inteira. É fácil simular a saída do verificador especificado na Construção 4.3.8 (pois sua saída é identicamente 1 para entradas na linguagem IG). Também, não é difícil simular a saída de um verificador que segue o programa especificado na Construção 4.3.8, exceto que ao término ele dará como saída a transcrição inteira de sua interação com P_{IG} (veja o Exercício 12). A parte difícil é simular a saída de um verificador eficiente que desvia arbitrariamente do programa especificado.

Usaremos aqui a formulação alternativa de conhecimento-zero (perfeito) e mostraremos como simular a visão de V^* da interação com P_{IG} para toda máquina interativa probabilística de tempo-polinomial V^*. Como mencionado anteriormente, não é difícil simular a visão do verificador da interação com P_{IG} quando o verificador segue o programa especificado. Entretanto, precisamos simular a visão do verificador no caso geral (no qual ele usa um programa interativo de tempo-polinomial arbitrário). O que se segue é uma visão geral de nossa simulação (i.e., de nossa construção de um simulador M^* para cada V^*).

O simulador M^* incorpora o código do programa interativo V^*. Sobre a entrada (G_1, G_2), o simulador M^* primeiro seleciona aleatoriamente um dos grafos de entrada (i.e., G_1 ou G_2) e gera uma cópia isomorfa aleatória, representar por G'', desse grafo de entrada. Ao fazer isso, o simulador se comporta diferentemente de P_{IG}, mas o grafo gerado (i.e., G'') é distribuído identicamente à mensagem enviada no Passo P1 da prova interativa. Digamos que o simulador gerou G'' permutando aleatoriamente G_1. Agora, se V^* pede para ver o isomorfismo entre G_1 e G'', então o simulador pode de fato respnder corretamente, e ao fazê-lo completa a simulação da visão do verificador da interação com P_{IG}. Entretanto, se V^* pede para ver o isomorfismo entre G_2 e G'', então o simulador (que, diferentemente de P_{IG}, não "conhece" ϕ) não tem

como responder corretamente, e admitimos que ele pára com $\perp$. Enfatizamos que o simulador "não tem como saber" se V^* pedirá para ver um isomorfismo para G_1 ou para G_2. O fato é que o simulador pode tentar uma das possibilidades aleatoriamente, e se ele tiver sorte (o que acontece com probabilidade exatamente $\frac{1}{2}$), então ele pode dar como saída uma distribuição que será idêntica à visão de V^* quando interagindo com P_{IG} (sobre a entrada comum (G_1, G_2)). Um fato chave (veja a Afirmação 4.3.9.1, a seguir) é que a distribuição de G'' é estocasticamente independente da escolha do simulador de qual dos dois grafos de entrada usar, e portanto V^* não pode afetar a probabilidade de que o simulador terá sorte. Uma descrição detalhada do simulador segue.

Simulador M^*. Sobre a entrada $x \stackrel{\text{def}}{=} (G_1, G_2)$, o simulador M^* procede como segue:

1. *Preparando a fita aleatória de V^**: Suponha que $q(\cdot)$ represente um polinômio limitando o tempo de execução de V^*. O simulador M^* começa selecionando uniformemente uma adeia $r \in \{0,1\}^{q(|x|)}$ para ser usada como o conteúdo da fita aleatória de V^*. (Alternativamente, poder-se-ia produzir moedas para V^* "na hora," isto é, durante o Passo 3, que segue.)

2. *Simulando o primeiro passo do provador (P1)*: O simulador M^* seleciona aleatoriamente, com distribuição uniforme de probabilidade, um "bit" $\tau \in \{1,2\}$ e uma permutação ψ do conjunto de permutações sobre o conjunto de vértices V_τ. Ele então constrói um grafo com conjunto de vértices V_τ e conjunto de arestas

$$F \stackrel{\text{def}}{=} \{(\psi(u), \psi(v)) : (u,v) \in E_\tau\},$$

   e faz $G'' \stackrel{\text{def}}{=} (V_\tau, F)$.

3. *Simulando o primeiro passo do verificador (V1)*: O simulador M^* inicia uma execução de V^* colocando x na fita de entrada-comum de V^*, colocando r (selecionada no Passo 1) na fita aleatória de V^*, e colocando G'' (construído no Passo 2) na fita de mensagens-recebidas de V^*. Após executar um número polinomial de passos de V^*, o simulador pode ler a mensagem enviada de V^*, representada por σ. Para simplificar o restante da descrição, normalizamos σ fazendo $\sigma = 1$ se $\sigma \neq 2$ (e deixamos σ sem modificação se $\sigma = 2$).

4. *Simulando o segundo passo do provador (P2)*: Se $\sigma = \tau$, então o simulador pára com saída (x, r, G'', ψ).

5. *Falha da simulação*: Caso contrário (i.e., $\sigma \neq \tau$), o simulador pára com saída $\perp$.

Usando a hipótese de que V^* é de tempo-polinomial, segue que o simulador M^* também o é. Faltaria mostrar que M^* dá como saída $\perp$ com probabilidade no máximo $\frac{1}{2}$ e que, condicionada a não dar como saída $\perp$, a saída do simulador é distribuída como a visão do verificador em uma "interação real com P_{IG}." A afirmação seguinte é a chave para a prova de ambas as afirmações.

Afirmação 4.3.9.1: Suponha que os grafos G_1 e G_2 são isomorfos. Seja ξ uma variável aleatória uniformemente distribuída em $\{1,2\}$, e seja Π uma variável aleatória

uniformemente distribuída sobre o conjunto de permutações sobre V_ξ. Então para todo grafo G'' que é isomorfo a G_1 (e G_2), verifica-se que

$$\mathsf{Pr}[\xi = 1 \mid \Pi(G_\xi) = G''] = \mathsf{Pr}[\xi = 2 \mid \Pi(G_\xi) = G''] = \frac{1}{2}$$

onde, tal qual na Afirmação 4.2.9.1, $\pi(G)$ representa o grafo obtido do grafo G pela modificação na rotulação dos nós usando a permutação π.

A Afirmação 4.3.9.1 é idêntica à Afirmação 4.2.9.1 (usada para demonstrar que a Construção 4.2.8 constitui uma prova interativa para NIG).[10] Como no restante da prova da Proposição 4.2.9, segue que qualquer processo aleatório com saída em $\{1, 2\}$, dado $\Pi(G_\xi)$, dá como saída ξ com probabilidade exatamente $\frac{1}{2}$. Logo, dado G'' (construído pelo simulador no Passo 2), o programa do verificador produz σ (normalizado), de modo que $\sigma \neq \tau$ com probabilidade exatamente $\frac{1}{2}$. Concluimos que o simulador dá como saída $\perp$ com probabilidade $\frac{1}{2}$. Resta provar que, condicionada a não dar como saída $\perp$, a saída do simulador é idêntica à "visão de V^* das interações reais." A saber:

Afirmação 4.3.9.2: Seja $x = (G_1, G_2) \in IG$. Então para toda cadeia r, grafo H, e permutação ψ, verifica-se que

$$\mathsf{Pr}[\text{visão}_{V^*}^{P_{IG}}(x) = (x, r, H, \psi)] = \mathsf{Pr}[M^*(x) = (x, r, H, \psi) \mid M^*(x) \neq \perp]$$

Prova: Suponha que $m^*(x)$ descreva $M^*(x)$ condicionada a não ser $\perp$. Primeiro observe que ambas $m^*(x)$ e visão$_{V^*}^{P_{IG}}(x)$ são distribuídas sobre quádruplas da forma $(x, r, \cdot, \cdot)$, com $r \in \{0,1\}^{q(|x|)}$ uniformemente distribuída. Seja $\nu(x, r)$ uma variável aleatória descrevendo os dois últimos elementos de visão$_{V^*}^{P_{IG}}(x)$ condicionada ao segundo elemento ser igual a r. Similarmente, suponha que $\mu(x, r)$ descreva os dois últimos elementos de $m^*(x)$ (condicionada ao segundo elemento ser igual a r). Precisamos mostrar que $\nu(x, r)$ e $\mu(x, r)$ são identicamente distribuídas para toda x e r. Observe que uma vez que r é fixada, a mensagem enviada por V^* sobre a entrada comum x, fita aleatória r, e mensagem recebida H, é univocamente definida. Vamos representar essa mensagem por $\nu^*(x, r, H)$. Mostramos que ambas $\nu(x, r)$ e $\mu(x, r)$ são uniformemente distribuídas sobre o conjunto

$$C_{x,r} \stackrel{\text{def}}{=} \{(H, \psi) : H = \psi(G_{\nu^*(x,r,H)})\}$$

onde (novamente) $\psi(G)$ representa o grafo obtido de G modificando-se a rotulação dos vértices usando a permutação ψ (i.e., se $G = (V, E)$, então $\psi(G) = (V, F)$, de modo que $(u, v) \in E$ sse $(\psi(u), \psi(v)) \in F$). A prova desse enunciado é um tanto cansativa e não está relacionada com os assuntos deste livro (e portanto pode ser pulada sem prejuízo).

> A prova é ligeiramente não-trivial porque ela está relacionada (ao menos implicitamente) ao grupo de automorfismos do grafo G_2 (i.e., o conjunto de permutações π para as quais $\pi(G_2)$ é idêntico a G_2, não apenas isomorfo a G_2). Por simplicidade, considere primeiro o caso especial no qual o grupo de automorfismos de G_2 consiste de meramente

[10] Na Construção 4.2.8, o grafo $\Pi(G_\xi)$ foi apresentado ao provador, e a Afirmação 4.2.9.1 foi usada para estabelecer a corretude do sistema de prova (i.e., analisar o que acontece no caso em que $(G_1, G_2) \notin NIG$, o que significa que $(G_1, G_2) \in IG$. Aqui o grafo $\Pi(G_\xi)$ é apresentado ao verificador, e a afirmação é usada para estabelecer a propriedade de conhecimento-zero (e portanto também se refere a $(G_1, G_2) \in IG$).

a permutação identidade (i.e., $G_2 = \pi(G_2)$ se e somente se π é a permutação identidade). Nesse caso, $(H, \psi) \in C_{x,r}$ se e somente se H é isomorfo a (ambos G_1 e) G_2 e ψ é o (único) isomorfismo entre H e $G_{\nu^*(x,r,H)}$. Daí, $C_{x,r}$ contém exatamente $|V_2|!$ pares, cada um contendo um grafo H diferente como primeiro elemento. No caso geral, $(H, \psi) \in C_{x,r}$ se e somente se H é isomorfo a (ambos G_1 e) G_2 e ψ é um isomorfismo entre H e $G_{\nu^*(x,r,H)}$. Enfatizamos que $\nu^*(x, r, H)$ é o mesmo em todos os pares contendo H. Suponha que $\mathrm{aut}(G_2)$ represente o tamanho do grupo de automorfismos de G_2. Então cada H (isomorfo a G_2) aparece em exatamente $\mathrm{aut}(G_2)$ pares de $C_{x,r}$, e cada par desses contém um isomorfismo diferente entre H e $G_{\nu^*(x,r,H)}$. O número de H's diferentes que são isomorfos a G_2 é $|V_2|!/\mathrm{aut}(G_2)$, e portanto $|C_{x,r}| = |V_2|!$ também no caso geral.

Primeiro consideramos a variável aleatória $\mu(x, r)$ (descrevendo o sufixo de $m^*(x)$). Recordemos que $\mu(x, r)$ é definida pelo seguinte processo aleatório em dois passos.A No *primeiro* passo, seleciona-se uniformemente um par (τ, ψ), sobre o conjunto dos pares $(\{1, 2\} \times$ permutação $)$, e faz-se $H = \psi(G_\tau)$. No *segundo* passo, dá-se como saída (i.e., faz $\mu(x, r)$ igual a) $(\psi(G_\tau), \psi)$ se $\nu^*(x, r, H) = \tau$ (e ignora-se o par (τ, ψ) caso contrário). Daí, cada grafo H (isomorfo a G_2) é gerado, no primeiro passo, por exatamente $\mathrm{aut}(G_2)$ $(1, \cdot)$-pares diferentes (i.e., os pares $(1, \psi)$ satisfazendo $H = \psi(G_1)$) e por exatamente $\mathrm{aut}(G_2)$ $(2, \cdot)$-pares diferentes (i.e., os pares $(2, \psi)$ satisfazendo $H = \psi(G_2)$). Todos esses $2 \cdot \mathrm{aut}(G_2)$ pares dão origem ao mesmo grafo H e portanto conduzem ao mesmo valor de $\nu^*(x, r, H)$. Segue que dos $2 \cdot \mathrm{aut}(G_2)$ pares da forma (τ, ψ) que dão origem ao grafo $H = \psi(G_\tau)$, apenas os $\mathrm{aut}(G_2)$ pares satisfazendo $\tau = \nu^*(x, r, H)$ conduzem a uma saída. Daí, para cada H (que é isomorfo a G_2), a probabilidade de $\mu(x, r) = (H, \cdot)$ é igual a $\mathrm{aut}(G_2)/(|V_2|!)$. Além do mais, para cada H (que é isomorfo a G_2),

$$\mathsf{Pr}[\mu(x, r) = (H, \psi)] = \begin{cases} \frac{1}{|V_2|!} & \text{se} \quad H = \psi(G_{\nu^*(x,r,H)}) \\ 0 & \text{caso contrário} \end{cases}$$

Daí $\mu(x, r)$ é uniformemente distribuída sobre $C_{x,r}$.

Agora consideramos a variável aleatória $\nu(x, r)$ (descrevendo o sufixo da visão do verificador em uma "interação real" com o provador). Recordemos que $\nu(x, r)$ é definida selecionando-se uniformemente uma permutação π (sobre o conjunto V_2) e fazendo-se $\nu(x, r) = (\pi(G_2), \pi)$ se $\nu^*(x, r, \pi(G_2)) = 2$, e $\nu(x, r) = (\pi(G_2), \pi \circ \phi)$ caso contrário, onde ϕ é o isomorfismo entre G_1 e G_2. Claramente, para cada H (que é isomorfo a G_2), a probabilidade de que $\nu(x, r) = (H, \cdot)$ é igual a $\mathrm{aut}(G_2)/(|V_2|!)$. Além do mais, para cada H (que é isomorfo a G_2),

$$\mathsf{Pr}[\nu(x, r) = (H, \psi)] = \begin{cases} \frac{1}{|V_2|!} & \text{se} \quad \psi = \pi \circ \phi^{2-\nu^*(x,r,H)} \\ 0 & \text{caso contrário} \end{cases}$$

Observando que $H = \psi(G_{\nu^*(x,r,H)})$ se e somente se $\psi = \pi \circ \phi^{2-\nu^*(x,r,H)}$, concluimos que $\mu(x, r)$ e $\nu(x, r)$ são identicamente distribuídas.

A afirmação segue. □

Isso completa a prova da Parte 3 da proposição. ■

4.3.3 Conhecimento-Zero com Respeito a Entradas Auxiliares

As definições de conhecimento-zero apresentadas anteriormente estão aquém do que é requerido em aplicações práticas, e consequentemente uma modificação menor deve ser usada. Recordamos que essas definições garantem que o que quer que seja eficientemente computado após interação com o provador sobre qualquer *entrada comum* possa ser eficientemente computado *a partir da entrada propriamente dita*. Entretanto,

em aplicações típicas, (e.g., quando uma prova interativa é usada como um subprotocolo dentro de um protocolo maior) o verificador interagindo com o provador sobre a entrada comum x pode ter alguma informação a priori adicional, codificada como uma cadeia z, que pode assistí-lo nas suas tentativas de "extrair conhecimento" do provador. Esse perigo pode ficar ainda mais agudo no provável caso em que z está relacionada com x. (Por exemplo, considere o protocolo da Construção 4.3.8 e o caso em que o verificador tem uma informação a priori relativa a um isomorfismo entre os grafos de entrada.) O que é tipicamente requerido é que o que quer que seja eficientemente computado a partir de x e z após interação com o provador sobre qualquer entrada comum x possa ser eficientemente computado a partir de x e z (sem qualquer interação com o provador). Esse requisito é formulado a seguir usando a noção ampliada de provas interativas apresentada na Definição 4.2.10.

Definição 4.3.10 (Conhecimento-Zero, Revisitado): *Seja (P,V) uma prova interativa para uma linguagem L (como na Definição 4.2.10). Designe por $P_L(x)$ o conjunto de cadeias y satisfazendo a condição de completude com respeito a $x \in L$ (i.e., $\mathsf{Pr}[\langle P(y), V(z)\rangle(x) =] \geq \frac{2}{3}$ para toda $z \in \{0,1\}^*$). Dizemos que (P,V) é de* **conhecimento-zero com respeito à entrada auxiliar** *(ou é de* **conhecimento-zero de entrada-auxiliar***) se para toda máquina interativa probabilística de tempo-polinomial V^* existe um algoritmo probabilístico M^*, rodando em tempo polinomial no comprimento de sua primeira entrada, tal que as duas seguintes coleções são computacionalmente indistingüíveis (quando o intervalo de diferenciação é considerado como uma função de $|x|$):*

- $\{\langle P(y_x), , V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$ *para $y_x \in P_L(x)$ arbitrária*
- $\{M^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$

A saber, para todo algoritmo probabilístico D com tempo de execução polinomial no comprimento da primeira entrada, para todo polinômio $p(\cdot)$, e toda $x \in L$ suficientemente longa, toda $y \in P_L(x)$, e $z \in \{0,1\}^*$, se verifica que

$$|\mathsf{Pr}[D(x,z,\langle P(y), V^*(z)\rangle(x) = 1] - \mathsf{Pr}[D(x,z,M^*(x,z)) = 1]| < \frac{1}{p(|x|)}$$

Nessa definição, y representa a informação a priori para o provador, enquanto que z representa a informação a priori para o verificador. Ambas y e z podem depender da entrada comum x; por exemplo, se y facilita a tarefa de provar, então y tem que depender de x (e.g., no caso em que y é uma $\mathcal{NP}$-testemunha para $x \in L \in \mathcal{NP}$). Enfatizamos também que a entrada auxiliar z (mas não y) é também dada ao algoritmo diferenciador (que pode ser pensado como uma extensão do verificador).

Recordemos que pela Definição 4.2.10, dizer que a máquina interativa V^* é probabilística de tempo-polinomial significa que seu tempo de execução é limitado por um polinômio no comprimento da entrada comum. Daí, o programa verificador, o simulador, e o algoritmo diferenciador todos rodam em tempo polinomial no comprimento de x (e não em tempo polinomial no comprimento total de todas as suas entradas). Essa convenção é essencial em muitos aspectos (a menos que se limite explicitamente o comprimento das entradas auxiliares por um polinômio no comprimento de x; veja o Exercício 11). Por exemplo, tendo permitido o algoritmo diferenciador rodar em

tempo proporcional ao comprimento da entrada auxiliar teria colapsado conhecimento-zero computacional para conhecimento-zero perfeito (e.g., considerando-se verificadores que rodam em tempo polinomial na entrada comum, e mesmo assim têm entradas auxiliares enormes de comprimento exponencial na entrada comum).

A Definição 4.3.10 se refere a conhecimento-zero computacional. Uma formulação de conhecimento-zero perfeito com relação à entrada auxiliar é imediata. Observamos que a prova de conhecimento-zero perfeito para Isomorfismo de Grafos, apresentada na Construção 4.3.8, é na verdade de conhecimento-zero perfeito em relação à entrada auxiliar. Esse fato segue facilmente por meio de uma argumentação pequena para o simulador construído na prova da Proposição 4.3.9 (i.e., quando invocando o verificador, o simulador deve provê-lo com a entrada auxiliar que é dada ao simulador). Em geral, uma demonstração de conhecimento-zero pode ser estendida para produzir conhecimento-zero com relação à entrada auxiliar sempre que o simulador usado na demonstração original funciona invocando o programa do verificador como uma caixa preta (veja a Definição 4.5.10 na Seção 4.5.4). Todos os simuladores apresentados neste livro têm essa propriedade.

Comentário Avançado: Não-Uniformidade Implícita na Definição 4.3.10

A natureza não-uniforme da Definição 4.3.10 é capturada pelo fato de que o diferenciador obtém uma entrada auxiliar. É verdade que essa entrada auxiliar também é dada a ambos o programa do verificador e o simulador; entretanto, se a entrada auxiliar é suficientemente longa, então apenas o diferenciador pode fazer uso de seu sufixo (pois o diferenciador pode ser determinado após o limitante de tempo-polinomial do simulador ser fixado). Segue que o simulador garantido na Definição 4.3.10 produz saída que é indistingüível das interações reais também por meio de circuitos não-uniformes de tamanho-polinomial (veja a Definição 3.2.7). A saber, para toda família (até mesmo não-uniforme) de circuitos de tamanho-polinomial $\{C_n\}_{n\in\mathbb{N}}$, todo polinômio $p(\cdot)$, todo n suficientemente grande, toda $x \in L \cap \{0,1\}^n$, toda $y \in P_L(x)$, e $z \in \{0,1\}^*$,

$$|\mathsf{Pr}[C_n(x,z,\langle P(y),V^*(z)\rangle(x))=1] - \mathsf{Pr}[C_n(x,z,M^*(x,z))=1]| < \frac{1}{p(|x|)}$$

O que se segue é um esboço da prova dessa afirmação. Assumimos, ao contrário, que existe uma família de circuitos de tamanho-polinomial $\{C_n\}_{n\in\mathbb{N}}$ tal que para um número infinito de n's existem triplas (x,y,z) para as quais C_n tem um intervalo diferenciador não-desprezível. Derivamos uma contradição incorporando a descrição de C_n juntamente com a entrada auxiliar z em uma entrada auxiliar mais longa, representada por z'. Isso é feito de tal maneira que ambos V^* e M^* tenham tempo insuficiente para chegar à descrição de C_n. Por exemplo, seja $q(\cdot)$ um polinômio limitando os tempos de execução de ambos V^* e M^*. Assuma, sem perda de generalidade, que $|z| \leq q(n)$ (ou do contrário o restante de z, que é ilegível por ambos V^* e M^*, pode ser ignorado). Então supomos que z' seja a cadeia que resulta da concatenação de brancos a z até um comprimento total de $q(n)$ e da descrição das descrições do circuito C_n no seu final (i.e., z é um prefixo de z'). Claramente, $M^*(x,z') = M^*(x,z)$ e $\langle P(y),v^*(z')\rangle(x) = \langle P(y),v^*(z)\rangle(x)$. Por outro lado, usando um algoritmo universal de cálculo-de-circuitos, obtemos um algoritmo probabilístico de tempo-polinomial D tal que $D(x,z',\alpha) = C_n(x,z,\alpha)$, e contradição (à hipótese de que M^* produz saída que é indistingüível em tempo-polinomial probabilístico da saída de (P,V^*) segue.

Mencionamos que a Definição 4.3.2 ela própria tem um certo sabor não-uniforme, pois ela requer indistinguibilidade para todas exceto um número finito de x's. Em contraste, um análogo inteiramente uniforme da definição requeriria apenas fosse impraticável encontrar x's sobre as quais a simulação falharia (com relação a algum diferenciador probabilístico de tempo-polinomial). Isto é, uma definição inteiramente uniforme de conhecimento-zero requer apenas que seja impraticável encontrar x's sobre as quais um verificador possa ganhar conhecimento (e não que tais instâncias não existam de forma alguma). Veja maior discussão na Seção 4.4.2.4.

Comentário Avançado: Por Que Não Caminhar para uma Formulação Inteiramente Não-Uniforme?

Uma versão supersimplificada da Definição 4.3.10 permite que o verificador seja modelado por uma família (não-uniforme) de circuitos (de tempo-polinomial), e permite o mesmo para o simulador. Os circuitos não-uniformes são supostamente responsáveis pelas entradas auxiliares, e portanto essas são tipicamente omitidas de tal versão supersimplificada. Por exemplo pode-se requerer o seguinte:

> *Para toda família de circuitos de tamanho-polinomial* $\{V_n\}_{n\in\mathbb{N}}$ (represetando uma possível máquina de estratégia do verificador) *existe uma família de circuitos de tamanho-polinomial* $\{M_n\}_{n\in\mathbb{N}}$ (representando um simulador) *tal que as coleções* $\{\langle P, V_{|x|}\rangle(x)\}_{x\in L}$ *e* $\{M_{|x|}(x)\}_{x\in L}$ *são indistingüíveis por circuitos de tamanho-polinomial.*

Entretanto, a impressão de que circuitos não-uniformes dão conta de entradas auxiliares é errada, e em geral encontramos tais versões super-simplificadas insatisfatórias. Primeiro, essas versões não garantem uma transformação "efetiva" de verificadores para simuladores. De fato, tal transformação também não é requerida na Definição 4.3.10, mas lá os objetos (i.e., máquinas) são de tamanho fixo, enquanto que aqui lidamos com objetos infinitos (i.e., famílias de circuitos). Por conseguinte, o nível de "segurança" oferecido pela definição super-simplificada é insatisfatório. Segundo, a versão supersimplificada não garante uma relação entre o tamanho da parte não-uniforme do verificador e a parte correspondente do simulador, enquanto que na Definição 4.3.10 a única parte não-uniforme é a entrada auxiliar, que permanece a mesma. Ambas as questões surgem ao tentar provar um teorema de composição-seqüencial para um número não-constante de iterações de sistemas de prova de conhecimento-zero. Finalmente, observamos que a versão super-simplificada *não* implica a versão básica (i.e., Definição 4.3.2); considere, por exemplo, um provador que sobre a entrada comum x envia alguma cadeia de comprimento de $\mathrm{poly}(|x|)$-bits difícil-de-computar que depende apenas de $|x|$ (e.g., a fatoração-prima de todos os inteiros no intervalo $[2^{|x|}+1,\ldots,2^{|x|}+|x|^3]$).

4.3.4* Composição Seqüencial de Provas de Conhecimento-Zero

Um requisito intuitivo que uma definição de provas de conhecimento-zero deve satisfazer é que provas de conhecimento-zero devem ser fechadas sob composição seqüencial. A saber, se executamos uma prova de conhecimento-zero após uma outra, então a execução composta tem que ser de conhecimento-zero. O mesmo deve permanecer válido mesmo se executarmos um número polinomial de provas uma após a outra. De fato, como será mostrado em breve, a definição revisada de conhecimento-zero (i.e.,

Definição 4.3.10) satisfaz esse requisito. É interessante que provas de conhecimento-zero como definidas na Definição 4.3.2 não são fechadas sob composição seqüencial, e esse fato é realmente uma outra indicação da necessidade de ampliar essa definição (como foi feito na Definição 4.3.10).

Em adição à sua importância conceitual, o lema da composição-seqüencial é uma ferramenta importante no desenho de sistemas de prova de conhecimento-zero. Tipicamente, tal sistema de prova consiste de muitas repetições de uma prova de conhecimento-zero atômica. Informalmente falando, a prova atômica provê *alguma* (porém não muita) evidência estatística para a validade da afirmação. Repetindo-se a prova atômica suficientemente, a confiança na validade da afirmação é incrementada. Mais precisamente, a prova atômica oferece um intervalo entre as probabilidades de aceitação para cadeias na linguagem e cadeias fora da linguagem. Por exemplo, na Construção 4.3.8, pares de grafos isomorfos (i.e., entradas em IG) são aceitos com probabilidade 1, enquanto que pares de grafos não-isomorfos (i.e., entradas que não pertencem a IG) são aceitos com probabilidade no máximo $\frac{1}{2}$. Repetindo-se a prova atômica, o intervalo entre as duas probabilidades é decrementado ainda mais. Por exemplo, repetindo a prova da Construção 4.3.8 k vezes produzirá uma nova prova interativa na qual entradas em IG são ainda aceitas com probabilidade 1, enquanto que entradas fora de IG são aceitas com probabilidade no máximo $\frac{1}{2^k}$. O lema de composição-seqüencial garante que se o sistema de prova-atômica é de conhecimento-zero, então o sistema de prova resultante de se repetir a prova atômica um número polinomial de vezes também o é.

Antes de enunciarmos o lema da composição-seqüencial, lembramos o leitor que a propriedade de conhecimento-zero de uma prova interativa é na verdade uma propriedade do provador. Além do mais, é requerido que o provador seja de conhecimento-zero apenas sobre entradas na linguagem. Finalmente, enfatizamos que quando falando sobre conhecimento-zero com respeito à entrada auxiliar, nos referimos a todas as entradas auxiliares possíveis para o verificador.

Lema 4.3.11 (Lema da Composição-Seqüencial): *Seja P uma máquina interativa (i.e., um provador) que é de* conhecimento-zero com respeito à entrada auxiliar *sobre alguma linguagem L. Suponha que a última mensagem enviada por P, sobre a entrada x, contenha um símbolo especial de fim-de-prova. Seja $Q(\cdot)$ um polinômio, e suponha que P_Q seja uma máquina interativa que, sobre a entrada comum x, procede em $Q(|x|)$ fases, cada uma delas consistindo de rodar P sobre a entrada comum x. (Enfatizamos que no caso em que P é probabilística, a máquina interativa P_Q usa cara-ou-coroa independentes para cada uma das $Q(|x|)$ fases.) Então P_Q é de conhecimento-zero (com respeito à entrada auxiliar) sobre L. Além do mais, se P é de conhecimento-zero perfeito (com respeito à entrada auxiliar), então P_Q também o é.*

A convenção relativa ao símbolo de fim-de-prova é introduzida por propósitos técnicos (e é redundante em todos os sistemas de prova conhecidos, e, além do mais, sempre que o número de mensagens enviadas durante a execução é facilmente computado a partir da entrada comum). Claramente, toda máquina P pode ser facilmente modificada de tal maneira que sua última mensagem conterá um símbolo apropriado (como assumido tal maneira que sua última mensagem conterá um símbolo apropriado (como assumido anteriormente), e ao fazê-lo preservará as propriedades de conhecimento-zero de P (assim como as condições de completude e corretude).

O lema ignora outros aspectos de se repetir uma prova interativa diversas vezes, especificamente, o efeito sobre o intervalo entre as probabilidades de aceitação para

entradas dentro e fora da linguagem. Esse último aspecto de repetir um sistema de prova interativa é discutido na Seção 4.2.1.3 (veja também o Exercício 1).

Prova. Seja V^* uma máquina interativa probabilística de tempo-polinomial *arbitrária* interagindo com o provador composto P_Q. Nossa tarefa é construir um simulador (de tempo-polinomial) M^* que simulará as interações reais de V^* com P_Q. A seguir vai uma descrição de muito alto nível da simulação. A idéia chave é simular a interação real sobre a entrada comum x em $Q(|x|)$ fases correspondendo às fases da operação de P_Q. Cada fase da operação de P_Q é simulada usando o simulador garantido para o provador atômico P. A informação acumulada pelo verificador em cada fase é passada à fase seguinte usando a entrada auxiliar.

> (Na exposição a seguir, ignoramos a entrada auxiliar para o provador. Isso meramente simplifica nossa noção. Isto é, ao invés de escrever $P(y)$ e $P_Q(y)$, onde y é a entrada auxiliar do provador, escrevemos P e P_Q.)

O primeiro passo na realização desse plano é particionar a execução de uma máquina interativa arbitrária V^* em fases. A partição pode não existir no código do programa V^*, e mesmo assim ela pode ser imposta às execuções desse programa. Isso é feito usando a estrutura de fases *do provador prescrito* P_Q, que por sua vez é induzido pelos símbolos de fim-de-prova. Portanto, afirmamos que independentemente de como V^* opera, a interação de V^* com P_Q sobre a entrada comum x pode ser capturada por $Q(|x|)$ interações sucessivas de uma máquina relacionada, representada por V^{**}, com P. A saber:

Afirmação 4.3.11.1: Existe uma V^{**} probabilística de tempo-polinomial tal que para toda entrada comum x e entrada auxiliar z, verifica-se que

$$\langle P_Q, V^*(z)\rangle(x) = Z^{(Q(|x|))}$$

onde $Z^{(0)} \stackrel{\text{def}}{=} z$ e

$$Z^{(i+1)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i)})\rangle(x) \quad \text{para } i = 0, \ldots, Q(|x|) - 1$$

A saber, $Z^{(Q(|x|))}$ é uma variável aleatória descrevendo a saída de V^{**} após $Q(|x|)$ interações sucessivas com P, sobre a entrada comum x, onde a entrada auxiliar de V^{**} na $i+1$-ésima interação iguala a saída de V^{**} após a i-ésima interação (i.e., $Z^{(i)}$).

Prova. Intuitivamente, V^{**} captura a funcionalidade de V^* durante cada fase individualmente. Pela *funcionalidade de V^* durante uma fase* queremos dizer a maneira pela qual V^* transforma o conteúdo de suas fitas de trabalho no início da fase para o conteúdo no final da fase, assim como a maneira pela qual V^* determina as mensagens que ela envia durange essa fase. De fato, essa transformação depende das mensagens recebidas durante a fase corrente. Enfatizamos que podemos realizar essa transformação sem fazer "engenharia-reversa" com (o código de) V^*, mas sim emulando sua execução enquanto monitorando todas as suas fitas. Detalhes seguem.

De modo a facilitar esse processo, primeiro modificamos V^* de modo que todas as suas atividades "essenciais" se refiram apenas a suas fitas de trabalho. A máquina V^* pode ser levemente modificada de modo que ela começa sua execução lendo a entrada comum, a entrada aleatória, e a entrada auxiliar para regiões especiais na sua

fita de trabalho e nunca acessa novamente as fitas somente-leitura acima mencionadas. Igualmente, V^* é modificada de modo que ela começa cad período ativo[11] (veja a Definição 4.2.2) lendo a mensagem recebida corrente da fita de comunicação para uma região especial na fita de trabalho (e nunca acessa novamente a fita de mensagens-recebidas durante a esse período). Na realidade, essa descrição deveria ser modificada de modo que V^* copie apenas um prefixo polinomialmente longo (na entrada comum) de cada uma dessas fitas, o polinômio sendo aquele limitando o tempo de execução da máquina V^* (original).

> (Formalmente falando, dada uma V^* arbitrária, construimos uma máquina W^* que emula V^* de uma maneira que satisfaz as condições acima; isto é, W^* vai satisfazer essas condições mesmo se V^* não as satisfaz. A máquina W^* começará copiando sua própria entrada comum, entrada aleatória, e entrada auxiliar para as fitas designadas correspondentes. Igualmente, W^* iniciará cada período ativo copiando a mensagem recebida corrente de sua própria fita de comunicação para a fita designada correspondente (i.e., a fita de comunicação de entrada de V^*). Após completar essas atividades de cópia, W^* simplesmente emula a execução de V^*. Claramente, W^* satisfaz os requisitos postulados. Por conseguinte, formalmente falando, sempre que nos referirmos adiante a V^*, queremos dizer W^*.)

Considere uma interação de $V^*(x)$ com P_Q, sobre a entrada comum x. Pela modificação acima, a interação consiste de $Q(|x|)$ fases, tal que, exceto na primeira fase, a máquina V^* nunca acessa suas fita-de-entrada, fita-aleatória, e fita-de-entrada-auxiliar. (Na primeira fase, a máquina V^* começa copiando o conteúdo dessas fitas para suas fitas de trabalho e nunca acessa novamente aquelas primeiras.) Igualmente, quando executando a fase corrente, a máquina V^* não tenta ler mensagens de fases *anteriores* de sua fita de comunicação de entrada (mesmo assim ela pode ler essas mensagens "velhas" da memória nas suas fitas de trabalho). Considerando o conteúdo das *fitas de trabalho de* V^* no final de *cada* uma das $Q(|x|)$ fases (da interação com P_Q) naturalmente nos leva à construção de V^{**}.

Estamos agora finalmente prontos para apresentar a construção de V^{**}: Sobre a entrada comum x e entrada auxiliar z', a máquina V^{**} começa copiando z' na fita de trabalho de V^*. A seguir, a máquina V^{**} emula uma *única fase* da interação de V^* com P_Q (sobre a entrada x), começando com o conteúdo anterior da fita de trabalho de V^* (ao invés de começar com uma fita de trabalho vazia). A máquina emulada V^* considera as fitas de comunicação da máquina V^{**} como suas próprias fitas de comunicação. Quando V^* completa a interação na fase corrente, a máquina V^{**} conclui dando como saída o conteúdo corrente da fita de trabalho de V^*. Por conseguinte, quando z' iguala um conteúdo possível da fita de trabalho de V^* após $i \geq 1$ fases, a máquina emulada V^* se comporta como na fase $i+1$, e a saída de V^{**} é distribuída como o conteúdo da fita de trabalho de V^* após $i+1$ fases. Na realidade, a descrição acima deveria ser levemente modificada para lidar com a primeira fase na interação com P_Q (i.e., o caso $i = 0$ ignorado anteriormente). Especificamente, V^{**} copia z' para a fita de trabalho de V^* apenas se z' codifica o conteúdo da fita de trabalho de V^* (assumimos, sem perda de generalidade, que o conteúdo da fita de trabalho de V^* é codificada diferentemente da codificação de uma entrada auxiliar para V^*). No caso em que z' codifica uma entrada auxiliar para V^*, a máquina V^{**} invoca V^* sobre uma fita de trabalho vazia, e V^*

[11] Recordemos que um *período ativo* durante uma execução de uma máquina interativa M consiste dos passos que M dá a partir do momento que a última mensagem é recebida até o momento no qual M finaliza enviando sua mensagem de resposta.

considera as fitas legíveis de V^{**} (i.e., fita de entrada-comum, fita de entrada-aleatória, e fita de entrada-auxiliar) como suas. Observe que $Z^{(1)} \stackrel{\text{def}}{=} \langle P, V^{**}(z)\rangle(x)$ descreve o conteúdo da fita de trabalho de V^* após a primeira fase (na interação com P_Q sobre a entrada comum x e a entrada auxiliar z). Igualmente, para todo $i = 2, \ldots, Q(|x|)$, a variável aleatória $Z^{(i)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i-1)})\rangle(x)$ descreve o conteúdo da fita de trabalho de V^* após i fases. A afirmação segue. □

Pelo fato de que V^{**} é uma máquina interativa de tempo-polinomial (com entrada auxiliar) interagindo com P, segue pela hipótese do lema que existe uma máquina probabilística que simula essas interações em tempo polinomial no comprimento da primeira entrada. Suponha que M^{**} designe esse simulador.[12] Então para todo algoritmo probabilístico de tempo-polinomial (em x) D, todo polinômio $p(\cdot)$, toda entrada $x \in L$ suficientemente longa, e toda $z \in \{0,1\}^*$, temos

$$|\mathsf{Pr}[D(x, z, \langle P, V^{**}(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x, z, M^{**}(x, z)) = 1]| < \frac{1}{p(|x|)} \qquad (4.1)$$

Agora estamos prontos para apresentar a construção de um simulador M^* que simula a saída "real" de V^* após interação com P_Q. Podemos assumir, sem perda de generalidade, que a saída de V^* iguala o conteúdo de suas fitas de trabalho no final da interação (pois a saída de V^* é computável em tempo-polinomial probabilístico a partir do conteúdo de suas fitas de trabalho naquele momento). A máquina M^* usa o simulador M^{**} (como uma caixa preta).

O simulador M^*: Sobre a entrada (x, z) a máquina M^* faz $z^{(0)} = z$ e procede em $Q(|x|)$ fases. Na i-ésima fase, a máquina M^* computa $z^{(i)}$ rodando a máquina M^{**} sobre a entrada $(x, z^{(i-1)})$. Depois que $Q(|x|)$ fases são completadas, a máquina M^* pára dando como saída $z^{(Q(|x|))}$.

Claramente, a máquina M^*, como construída aqui, roda em tempo polinomial na sua primeira entrada. Falta mostrar que a máquina M^* de fato produz uma saída que é computacionalmente indistingüível da saída de V^* (após interagir com P_Q). A saber:

Afirmação 4.3.11.2: Para todo algoritmo probabilístico D com tempo de execução polinomial na sua primeira entrada, todo polinômio $p(\cdot)$, toda $x \in L$ suficientemente longa, e toda $z \in \{0,1\}^*$, temos

$$|\mathsf{Pr}[D(x, z, \langle P_Q, V^*(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x, z, M^*(x, z)) = 1]| < \frac{1}{p(|x|)}$$

Além do mais, se P é de conhecimento-zero perfeito, então $\langle P_Q, V^*(z)\rangle(x)$ e $M^*(x, z)$ são identicamente distribuídas.

Esboço de prova: Usamos um argumento híbrido (veja o Capítulo 3). Em particular, definimos os seguintes $Q(|x|) + 1$ híbridos. O i-ésimo híbrido, $0 \leq i \leq Q(|x|)$, corresponde ao seguinte processo aleatório. Primeiro fazemos V^{**} interagir com P para i fases, começando com a entrada comum x e a entrada auxiliar z, e representamos

[12] Recordemos que no caso de conhecimento-zero perfeito (veja a Definição 4.3.1) a máquina M^{**} pode parar sem qualquer saída real (mas sim com saída $\perp$). Entretanto, através de um número suficiente de repetições, podemos tornar a probabilidade desse evento exponencialmente desprezível. No restante da exposição, assumimos em nome da simplicidade que M^{**} sempre pára com alguma saída.

por $Z^{(i)}$ a saída de V^{**} após a i-ésima fase, A seguir iteramos repetidamente M^{**} para as $Q(|x|)-i$ fases restantes. Em ambos os casos, usamos a saída da fase anterior como entrada auxiliar para a nova fase. Formalmente, o hïbrido $H^{(i)}$ é definido como segue:

$$H^{(i)}(x,z) \stackrel{\text{def}}{=} M^{**}_{Q(|x|)-i}(x, Z^{(i)})$$

onde os $Z^{(j)}$'s são tal qual definidos na Afirmação 4.3.11.1, e onde

$$M^{**}_0(x,z') \stackrel{\text{def}}{=} z' \quad \text{e} \quad M^{**}_k(x,z') \stackrel{\text{def}}{=} M^{**}_{k-1}(x, M^{**}(x,z')) \text{ para } k = 1, \ldots, Q(|x|)-i$$

Pela Afirmação 4.3.11.1, o $Q(|x|)$-ésimo híbrido iguala $\langle P_0, V^*(z)\rangle(x)$. Por outro lado, recordando a construção de M^*, vemos que o híbrido zero (i.e., $H^{(0)}(x,z) = M^{**}_{Q(|x|)}(x,z)$) iguala $M^*(x,z)$. Portanto, tudo o que é requerido para completar a prova é mostrar que todos os pares de dois híbridos adjacentes são computacionalmente indistingüíveis (pois isso implicará que os híbridos extremos, $H^{(Q(|x|))}$ e $H^{(0)}$, são também indistingüíveis). Para esse fim, reescrevemos os híbridos i e $i-1$ como segue:

$$\begin{aligned} H^{(i)}(x,z) &= M^{**}_{Q(|x|)-i}(x, Z^{(i)}) \\ &= M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(Z^{(i-1)})\rangle(x)) \\ H^{(i-1)}(x,z) &= M^{**}_{Q(|x|)-(i-1)}(x, Z^{(i-1)}) \\ &= M^{**}_{Q(|x|)-i}(x, M^{**}(x, Z^{(i-1)})) \end{aligned}$$

onde $Z^{(i-1)}$ é tal qual definido na Afirmação 4.3.11.1.

Usando um argumento baseado na média, segue que se um algoritmo D distingue os híbridos $H^{(i)}(x,z)$ e $H^{(i-1)}(x,z)$, então existe um z' (no suporte de $Z^{(i-1)}$) tal que o algoritmo D distingue as variáveis aleatórias $M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x))$ e $M^{**}_{Q(|x|)-i}(x, M^{**}(x,z'))$ pelo menos tão bem quanto. (Em todos os casos, D também é alimentado com x e z.) Usando os algoritmos M^{**} e D, obtemos um novo algoritmo D', com tempo de execução polinomialmente relacionado àqueles dois algoritmos, que distingue as variáveis aleatórias $(x,z,i,z', \langle P, V^{**}(z')\rangle(x))$ e $(x,z,i,z', M^{**}(x,z'))$ pelo menos tão bem quanto. Especificamente, sobre a entrada $(x,(z,i,z'),\alpha)$ (onde α é tomada ou de $\langle P, V^{**}(z')\rangle(x)$ ou de $M^{**}(x,z')$), o algoritmo D' invoca D sobre a entrada $(x, z, M^{**}_{Q(|x|)-i}(x,\alpha))$ e dá como saída o que quer que D dê. Claramente,

$$\begin{aligned} &|\mathsf{Pr}[D'(x,(z,i,z'), \langle P, V^{**}(z')\rangle(x)) = 1] - \mathsf{Pr}[D'(x,(z,i,z'), M^{**}(x,z')) = 1]| \\ &\geq |\mathsf{Pr}[D(x,z,H^{(i)}(x,z)) = 1] - \mathsf{Pr}[D(x,z,H^{(i-1)}(x,z)) = 1]| \end{aligned}$$

Note que D' usa a entrada adicional (x,z,i,z'), enquanto que distingue $\langle P, V^{**}(z')\rangle(x)$ de $M^{**}(x,z')$. Isso não encaixa na definição de um diferenciador para conhecimento-zero (com entrada auxiliar), pois o último deve receber apenas (x,z') e a cadeia a ser distingüida. Em outras palavras, construímos na verdade um $D' = D'_{i,z}$ não-uniforme que, dependendo de i e z, distingue $\langle P, V^{**}(z')\rangle(x)$ de $M^{**}(x,z')$. Ainda assim, no caso de conhecimento-zero perfeito, fazendo de D uma função arbitrária (ao invés de um algoritmo eficiente), isso basta para contradizer a hipótese de que M^{**} simula perfeitamente (P, V^{**}). Para o caso de conhecimento-zero computacional, usamos o fato de que a definição de conhecimento-zero com entrada-auxiliar implica robustez contra diferenciadores não-uniformes (de tamanho-polinomial), e observamos que Di, z' recai nessa categoria (desde que D também recaia). Por conseguinte, em ambos os casos, a contradição (à hipótese de que M^{**} simula (P, V^{**})) segue. □

Detalhes adicionais relativos à prova da Afirmação 4.3.11.2: Neste estágio (assumindo que o leitor tenha passado pelo Capítulo 3), o leitor deveria ser capaz de transformar o esboço de prova acima numa prova detalhada. A principal coisa que está faltando é o detalhe concernente à maneira pela qual um algoritmo contradizendo a hipótese de que M^{**} é um simulador para (P, V^{**}) é derivado de um algoritmo contradizendo o enunciado da Afirmação 4.3.11.2. Esses detalhes são apresentados a seguir.

Assumimos, para chegar à contradição, que existe um algoritmo probabilístico de tempo-polinomial D e um polinômio $p(\cdot)$ tal que para um número infinito de $x \in L$, existe $z \in \{0,1\}^*$ tal que

$$|\mathsf{Pr}[D(x, z, \langle P_Q, V^*(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x, z, M^*(x, z)) = 1] > \frac{1}{p(|x|)}$$

Segue que para todas tais x e z, existe um $i \in \{1, \ldots, Q(|x|)\}$ tal que

$$|\mathsf{Pr}[D\left(x, z, H^{(i)}(x, z)\right) = 1] - \mathsf{Pr}[D\left(x, z, H^{(i-1)}(x, z)\right) = 1]| > \frac{1}{Q(|x|) \cdot p(|x|)}$$

onde os híbridos $H^{(j)}$'s são tal qual definidos anteriormente. Designe $\varepsilon(n) \stackrel{\text{def}}{=} 1/(Q(n) \cdot p(n))$. Combinando, como antes, as definições do híbridos i e $i-1$ com um argumento baseado na média, segue que para cada tais x, z, e i, existe uma z' tal que

$$\begin{aligned}|\mathsf{Pr}[D\left(x, z, M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x)\right) = 1] \\ -\mathsf{Pr}[D\left(x, z, M^{**}_{Q(|x|)-i}(x, M^{**}(x, z'))\right) = 1]| > \varepsilon(|x|)\end{aligned}$$

Isso quase leva à contradição desejada. A saber, as variáveis aleatórias $(x, z', \langle P, V^{**}(z')\rangle(x))$ e $(x, z', M^{**}(x, z'))$ podem ser distinguidas usando os algoritmos D e M^{**}, *desde que "conheçamos" i e z*. Mas como chegamos a "conhecer" i e z? O problema é resolvido usando o fato, apontado anteriormente, de que a saída de m^{**} deveria ser indistingüível das interações de V^{**} com P mesmo com respeito a circuitos não-uniformes de tamanho-polinomial. Por conseguinte, de modo a derivar uma contradição, basta construir um diferenciador não-uniforme que incorpore i e z na sua descrição. Alternativamente, podemos incorporar i e z em uma nova entrada auxiliar, representada por z'', de modo que z' seja um prefixo de z'', mas que z'' pareça a mesma que z' para ambas V^* e M^*. A seguir seguiremos a última alternativa.

Suponha que T represente um limitante superior polinomial sobre a complexidade-de-tempo de ambas V^* e M^*. Note que para toda z' determinada para um par (x, z), como antes, deve-se verificar que $|z'| \leq T(|x|)$ (pois z' é um possível registro de uma computação parcial de $M^*(x, z)$). Seja $z'' = z'\flat^{T(|x|)-|z'|}, i, z)$, onde i e z são como antes (e $\flat$ representa o símbolo em-branco da fita de trabalho). Construimos um algoritmo probabilístico de tempo-polinomial D' que distingue $(x, z'', \langle P, V^{**}(z'')\rangle(x))$ e $(x, z'', M^{**}(x, z''))$ para as (x, z, i, z')-tuplas supramencionadas. Sobre a entrada (x, z'', α) (onde α supostamente está em $\langle P, V^{**}(z'')\rangle(x) = \langle P, V^{**}(z')\rangle(x)$ ou em $M^{**}(x, z'') = M^{**}(x, z')$), o algoritmo D' primeiro extrai i e z de z''. A seguir, ele usa M^{**} para computar $\beta = M^{**}_{Q(|x|)-i}(x, \alpha)$. Finalmente, D' pára com saída $D(x, z, \beta)$. Usando o fato de que V^{**} e M^{**} não podem distinguir as entradas auxiliares z' e z'', temos

$$\begin{aligned}&|\mathsf{Pr}[D'(x, z'', \langle P, V^{**}(z'')\rangle(x)) = 1] - \mathsf{Pr}[D'(x, z'', M^{**}(x, z'')) = 1]| \\ &\quad = |\mathsf{Pr}[D'(x, z'', \langle P, V^{**}(z')\rangle(x)) = 1] - \mathsf{Pr}[D'(x, z'', M^{**}(x, z')) = 1]| \\ &\quad = |\mathsf{Pr}[D\left(x, z, M_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x))\right) = 1] \\ &\qquad -\mathsf{Pr}[D\left(x, z, M_{Q(|x|)-i}(x, M^{**}(x, z'))\right) = 1]| \\ &\quad > \varepsilon(|x|)\end{aligned}$$

Contradição (à hipótese de que M^{**} é um simulador para (P, V^{**})) segue. □

O lema segue. ■

E Que Tal A Composição Paralela?

Infelizmente, não podemos provar que conhecimento-zero (mesmo com respeito à entrada auxiliar) é preservado sob composição paralela. Além do mais, existem provas de conhecimento-zero (com entrada-auxiliar) que quando executadas duas vezes em paralelo de fato produzem conhecimento (para um "verificador enganador"). Para maiores detalhes, veja a Seção 4.5.4.

O fato de que conhecimento-zero não é preservado sob composição paralela de protocolos é realmente má notícia. Poder-se-ia até dizer que esse fato é um fenômeno conceitualmente incômodo. Discordamos dessa avaliação. Nosso sentimento é que o comportamento de protocolos e "jogos" sob composição paralela é, em geral (i.e., não apenas no contexto de conhecimento-zero), uma questão muito mais complexa do que seu comportamento sob composição seqüencial: na verdade, em vários outros casos (e.g., provas computacionalmente corretas, provas de conhecimento, e sistemas de prova multi-provadores; veja Seções 4.8, 4.7, e 4.11, respectivamente), a composição paralela está atrasada em relação à composição seqüencial. Além do mais, a única vantagem da composição paralela sobre a composição seqüencial é em eficiência. Daí, não consideramos o não-fechamento sob composição paralela como sendo uma fraqueza fundamental da formulação de conhecimento-zero. Ainda assim o "não-fechamento" de conhecimento-zero motiva a busca por noções alternativas (relacionadas) que são preservadas sob composição paralela. (Tais noções podem ser mais fracas ou mais fortes que a formulação de conhecimento-zero. Para maiores detalhes, o leitor é remetido às Seções 4.9 e 4.6.

4.4 Provas de Conhecimento-Zero para $\mathcal{NP}$

Esta seção apresenta a principal força deste capítulo, a saber, um método para construir provas de conhecimento-zero para *toda* linguagem em $\mathcal{NP}$. A importância deste método provém de sua generalidade, que é a chave para suas muitas aplicações. Especificamente, quase todos os enunciados que se poderia querer provar na prática podem ser codificados como afirmações relativas a pertinência em linguagens em $\mathcal{NP}$. Em particular, a construção de provas de conhecimento-zero para tais enunciados provê uma ferramenta para "forçar" as partes a executar apropriadamente qualquer protocolo dado.

O método para construir provas de conhecimento-zero para linguagens $\mathcal{NP}$ faz uso essencial de *comprometimento de bit*. Portanto, começamos com uma apresentação desse último conceito. (Um leitor que desejar ter um pouco mais do sabor dessa aplicação de esquemas de comprometimento antes de estudá-los é encorajado a ler a Seção 4.4.2.1 primeiro.)

4.4.1 Esquemas de Comprometimento

Esquemas de comprometimento são ingredientes básicos em muitos protocolos criptográficos. Eles são usados para habilitar uma parte a se comprometer com um valor enquanto mantendo-o em segredo. Em um estágio posterior o comprometimento é

"aberto," e é garantido que a "abertura" pode produzir apenas um único valor determinado na fase de comprometimento. Esquemas de comprometimento são os análogos digitais de envelopes selados não-transparentes. Colocando um aviso em um tal envelope, uma parte se compromete com o conteúdo do aviso enquanto mantendo o conteúdo em segredo.

4.4.1.1 Definição

Informalmente falando, um esquema de comprometimento é um protocolo de duas-partes *bidirecional* eficiente através do qual uma parte, chamada *emissor*, pode se comprometer com um *valor* tal que os seguintes requisitos conflitantes são satisfeitos.

1. *Sigilo* (ou *ocultação*): Ao final da primeira fase, a outra parte, chamada de *receptor*, não ganha qualquer conhecimento do valor do emissor. Este requisito tem que ser satisfeito mesmo se o receptor tenta fraudar.

2. *Desambigüidade* (ou *amarração*): Dado o registro da interação na primeira fase, existe no máximo um valor que o receptor pode mais tarde (i.e., na segunda fase) aceitar como uma abertura "legal" do comprometimento. Este requisito tem que ser satisfeito mesmo se o emissor tenta fraudar.

Em adição, dever-se-ia requerer que o protocolo seja *viável*, no sentido de que se ambas as partes o seguem, então ao final da segunda fase o receptor obtém o valor com o qual se comprometeu o emissor. A primeira fase é chamada de *fase de comprometimento*, e a segunda fase é chamada de *fase de revelação*. Estamos requerendo que a fase de comprometimento não produza qualquer conhecimento (pelo menos nenhum conhecimento do valor do emissor) para o receptor, enquanto que a fase de comprometimento de fato "amarra" o emissor a um valor único (no sentido de que na fase de revelação o receptor pode aceitar apenas esse valor). Enfatizamos que o protocolo é eficiente no sentido de que os programas pré-determinados de ambas as partes podem ser implementados em tempo polinomial probabilístico. Sem perda de generalidade, a fase de revelação pode consistir de simplesmente deixar o emissor enviar, ao receptor, o valor original e a seqüência de cara-ou-coroa aleatórios que ele usou durante a fase de comprometimento. O receptor aceitará o valor se e somente se a informação suprida casa com seu registro da interação na fase de comprometimento. Essa última convenção leva à seguinte definição (que se refere explicitamente apenas à fase de comprometimento).

Definição 4.4.1 (Esquema de Comprometimento-de-Bit): *Um* **esquema de comprometimento-de-bit** *é um par de máquinas interativas probabilísticas de tempo polinomial, representado por* (E, R) *(para* emissor *e* receptor*), satisfazendo o seguinte:*

- Especificação de entrada: *A* entrada comum *é um inteiro* n *apresentado em unário (servindo como o parâmetro de segurança).*

  A entrada privada *para o receptor é um bit, representado por* v*.*

- Sigilo *(ou* ocultação*): O receptor (mesmo quando desviando arbitrariamente do protocolo) não pode distinguir um comprometimento com* 0 *de um comprometimento com* 1*. A saber, para toda máquina probabilística de tempo-polinomial* R^* *interagindo com* E*, as coleções descrevendo a saída de* R^* *nos dois casos, a saber* $\{\langle E(0), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$ *e* $\{\langle E(1), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$*, são computacionalmente indistingüíveis.*

- Desambigüidade *(ou* amarração*): Preliminares ao requisito:*

  1. Uma visão do receptor de uma interação *com o emissor, representada por* $(e, \overline{m})$*, consiste das moedas aleatórias usadas pelo receptor* (e) *e a seqüência de mensagens recebidas do emissor* $(\overline{m})$.
  2. *Seja* $\sigma \in \{0,1\}$. *Dizemos que uma visão do receptor (de tal interação),* $(r, \overline{m})$*, é um* σ**-comprometimento possível** *se existe uma cadeia* s *tal que* $\overline{m}$ *descreve as mensagens recebidas por* R *quando* R *usa moedas locais* r *e interage com a máquina* E *que usa moedas locais* e *e tem entrada* $(\sigma, 1^n)$. *(Usando a notação da Definição 4.3.3, dizemos que* $(r, \overline{m})$ *é um* σ*-comprometimento possível se* $(r, \overline{m}) = \text{visão}_{R(1^n,r)}^{E(\sigma,1^n,e)}$.*)*
  3. *Dizemos que a visão do receptor* $(r, \overline{m})$ *é* **ambígua** *se ela é tanto um* 0*-comprometimento possível quanto um* 1*-compromentimento possível.*

  O requisito de desambigüidade *assevera que para todos exceto uma fração desprezível de arremessos de moedas do receptor não existe qualquer seqüência de mensagens (do emissor) que juntamente com esses arremessos de moedas forme uma visão do receptor ambígua. A saber, para todos exceto uma fração dezprezível dos* $r \in \{0,1\}^{\text{poly}(n)}$ *não existe qualquer* $\overline{m}$ *tal que* $(r, \overline{m})$ *seja ambígua.*

O requisito de sigilo é um requisito computacional. Por outro lado, o *requisito de desambigüidade* tem sabor informação-teórico (i.e., ele não se refere a poderes computacionais) e é às vezes referido como *perfeito* (ou *absoluto*). Por conseguinte, um esquema de comprometimento como na Definição 4.4.1 é às vezes referido como *computacionalmente ocultante* e *perfeitamente amarrador*. Uma definição dual, requerendo sigilo informação-teórico e impraticabilidade computacional de se criar ambigüidades, é apresentada na Seção 4.8.2. (Essa última é referida como *perfeitamente ocultante* e *computacionalmente amarradora*.)

Fase Canônica de Revelação. O requisito de sigilo se refere explicitamente à situação no final da fase de comprometimento. Por outro lado, enfatizamos que o requisito de desambigüidade implicitamente assume que a fase de revelação toma a seguinte forma:

1. O emissor envia ao receptor sua entrada inicial privada v e as moedas aleatórias e que ele usou na fase de comprometimento.

2. O receptor verifica que v e e (juntamente com as moedas (r) usadas por R na fase de comprometimento) de fato produzem as mensagens que R recebeu na fase de comprometimento. A verificação é feita em tempo polinomial (rodando os programas E e R).

Note que o requisito de viabilidade (i.e., asseverando que se ambas as partes seguem o protocolo, então ao final da fase de revelação o receptor obtém v) é implicitamente satisfeito por essa convenção.

4.4.1.2 Construção Baseada em Qualquer Permutação Unidirecional

Alguns esquemas de encriptação de chave-pública podem ser usados como um esquema de comprometimento. Isso pode ser feito fazendo-se com que o emissor gere um par de

chaves e use a chave pública juntamente com a encriptação de um valor como seu comprometimento para com o valor. De modo a satisfazer o requisito de desambigüidade, o esquema de chave-pública subjacente precisa satisfazer requisitos adicionais (i.e., o conjunto de chaves públicas legítimas devem ter uma decriptação única). De qualquer modo, esquemas de encriptação de chave-pública têm propriedades adicionais não exigidas dos esquemas de encriptação, e suas existências parecer requerer suposições de intratabilidade mais fortes. (Por conseguinte, consideramos a abordagem supramencionada como sendo "conceitualmente errada.") Uma construção alternativa, apresentada a seguir, usa qualquer *permutação unidirecional*. Especificamente, usamos uma permutação unidirecional, representada por f, e um predicado núcleo-duro para ela, representada por b (veja a Seção 2.5). Na verdade, podemos usar qualquer função unidirecional 1-1.

Construção 4.4.2 (Comprometimento de Bit Simples): *Seja $f : \{0.1\}^* \to \{0,1\}^*$ uma função, e seja $b : \{0,1\}^* \to \{0,1\}$ um predicado.*

1. Fase de comprometimento: *Para comprometer um valor $v \in \{0,1\}$ (usando o parâmetro de segurança n), o emissor seleciona uniformemente $e \in \{0,1\}^*$ e envia o par $(f(e), b(e) \oplus v)$ ao receptor.*

2. Fase (canônica) de revelação: *Na fase de revelação, o emissor revela o bit v e a cadeia e usada na fase de comprometimento. O receptor aceita o valor v se $f(e) = \alpha$ e $b(e) \oplus v = \sigma$, onde (α, σ) é a visão do receptor da fase de comprometimento.*

Proposição 4.4.3: *Seja $f : \{0,1\}^* \to \{0,1\}^*$ uma função unidirecional 1-1, e $b : \{0,1\}^* \to \{0,1\}$ um predicado núcleo-duro de f. Então o protocolo apresentado na Construção 4.4.2 constitui um esquema de comprometimento-de-bit.*

Prova: O requisito de sigilo segue diretamente do fato de que b é um núcleo-duro de f. O requisito de desambigüidade segue da propriedade 1-1 de f. Na verdade, não existe *qualquer* visão ambígua do receptor. A saber, para cada visão possível do receptor (α, σ), existe um único $e \in \{0,1\}^{|\alpha|}$ tal que $f(e) = \alpha$, e portanto um único $v \in \{0,1\}$ tal que $b(e) \oplus v = \sigma$. ■

4.4.1.3 Construção Baseada em Qualquer Função Unidirecional

Agora apresentamos uma construção de um esquema de comprometimento-de-bit que é baseado na suposição mais fraca possível: a existência de funções unidirecionais. Provar que a suposição é de fato mínima é deixado como um exercício (i.e., Exercício 13). Por outro lado, pelos resultados do Capítulo 3 (especificamente, Teoremas 3.3.3 e 3.5.12), a existência de funções unidirecionais implica na existência de geradores pseudoaleatórios expandindo cadeias de n-bits em cadeias de $3n$-bits. Usaremos tal gerador pseudoaleatório na construção apresentada a seguir.

Começamos motivando a construção. Seja G um gerador pseudoaleatório satisfazendo $|G(e)| = 3 \cdot |e|$. Assuma que G tem a propriedade de que os conjuntos $\{G(e) : e \in \{0,1\}^n\}$ e $\{G(e) \oplus 1^{3n} : e \in \{0,1\}^n\}$ são disjuntos, onde $\alpha \oplus \beta$ representa o OU-EXCLUSIVO bit-a-bit das cadeias α e β. Então o emissor pode se comprometer com o bit v selecionando uniformemente $e \in \{0,1\}^n$ e enviando a mensagem $G(e) \oplus v^{3n}$ (v^k representa a cadeia toda-v de comprimento k-bits). Infelizmente,

a suposição não pode ser justificada em geral, e uma variante um pouco mais complexa é requerida. A observação chave é que para a maioria das cadeias $r \in \{0,1\}^{3n}$ os conjuntos $\{G(e) : e \in \{0,1\}^n\}$ e $\{G(e) \oplus r : e \in \{0,1\}^n\}$ são disjuntos. Tal cadeia r é dita *boa*. Essa observação sugere o seguinte protocolo: O receptor seleciona uniformemente $r \in \{0,1\}^{3n}$, esperando que seja boa, e envia r ao emissor. Tendo recebido r, o emissor se compromete com o bit v selecionando uniformemente $e \in \{0,1\}^n$ e enviando a mensagem $G(e)$ se $v = 0$, e $G(e) \oplus r$ caso contrário.

Construção 4.4.4 (Comprometimento de Bit sob Suposições Gerais): *Seja $G : \{0,1\}^* \to \{0,1\}^*$ uma função tal que $|G(e)| = 3 \cdot |e|$ para toda $e \in \{0,1\}^*$.*

1. Fase de comprometimento:
   - *Para receber um comprometimento a um bit (usando o parâmetro de segurança n), o receptor seleciona uniformemente $r \in \{0,1\}^{3n}$ e a envia ao emissor.*
   - *Ao receber a mensagem r (do receptor), o emissor se compromete com o valor $v \in \{0,1\}$ selecionando uniformemente $e \in \{0,1\}^n$ e enviando $G(e)$ se $v = 0$ e $G(e) \oplus r$ caso contrário.*
2. Fase (Canônica) de Revelação: *Na fase de revelação, o emissor envia a cadeia e usada na fase de comprometimento. O receptor aceita o valor 0 se $G(e) = \alpha$ e aceita o valor 1 se $G(e) \oplus r = \alpha$, onde (r, α) é a visão do receptor da fase de comprometimento.*

Tal definição da fase (canônica) de revelação permite ao receptor aceitar ambos os valores, mas mostraremos que isso acontece muito raramente (se é que acontece).

Proposição 4.4.5: *Se G é um gerador pseudoaleatório, então o protocolo apresentado na Construção 4.4.4 constitui um esquema de comprometimento-de-bit.*

Prova: O requisito de sigilo segue do fato de que G é um gerador pseudoaleatório. Especificamente, suponha que U_k represente a variável aleatória uniformemente distribuída sobre cadeias de comprimento k. Então para toda $d \in \{0,1\}^{3n}$, as variáveis aleatórias U_{3n} e $U_{3n} \oplus d$ são identicamente distribuídas. Logo, se é factível encontrar uma $r \in \{0,1\}^{3n}$ tal que $G(U_n)$ e $G(U_n) \oplus r$ sejam computacionalmente distingüíveis, então U_{3n} e $G(U_n)$ são computacionalmente distingüíveis ou $U_{3n} \oplus r$ e $G(U_n) \oplus r$ são computacionalmente distingüíveis. Em qualquer dos casos, contradição à aleatoricidade de G segue.

Agora nos voltamos ao requisito de desambigüidade. Seguindo a discussão motivadora, chamamos $r \in \{0,1\}^{3n}$ de *boa* se os conjuntos $\{G(e) : e \in \{0,1\}^n\}$ e $\{G(e) \oplus r : e \in \{0,1\}^n\}$ forem disjuntos. Dizemos que $r \in \{0,1\}^{3n}$ *produz uma colisão entre as sementes e_1 e e_2* se $G(e_1) = G(e_2) \oplus r$. Claramente, r é boa se ela não produz uma colisão entre qualquer que seja o par de sementes. Por outro lado, existe no máximo uma cadeia r que produz uma colisão entre um dado par de sementes (e_1, e_2); isto é, $r = G(e_1) \oplus G(e_2)$. Pelo fato de que existem no máximo $\binom{2^n}{2} < 2^{2n}$ possíveis pares de sementes, menos que 2^{2n} cadeias produzirão uma colisão entre pares de sementes, e portanto todas as outras cadeias de $3n$-bits de comprimento são boas. Segue que com probabilidade pelo menos $1 - 2^{2n-3n}$ o receptor seleciona uma cadeia boa, caso em que sua visão (r, α) é desambígua (pois se r é boa e $G(e_1) = \alpha$ se verifica para alguma e_1, então $G(e_2) \neq \alpha \oplus r$ deve se verificar para todas as e_2's.). O requisito de desambigüidade segue. ■

4.4.1.4 Extensões

A definição e as construções de esquemas de comprometimento-de-bit são facilmente esten'ıveis a esquemas gerais de comprometimento, habilitando o emissor a se comprometer com uma cadeia ao invés de um único bit. Na verdade, para os propósitos do restante desta seção, necessitamos de um esquema de comprometimento pelo qual pode haver comprometimento com um valor ternário. Estendendo a definição e as contruções para lidar com esse caso especial é ainda mais direto.

No restante desta seção necessitaremos de esquemas de comprometimento com um requisito de sigilo aparentemente mais forte que o definido anteriormente. Especificamente, ao invés de requerer sigilo com respeito a todas as máquinas de tempo-polinomial, requeremos sigilo com respeito a todas as famílias (não necessariamente uniformes) de circuitos de tamanho-polinomial. Assumindo a existência de funções não-uniformemente unidirecionais (veja a Definição 2.2.6 na Seção 2.2), esquemas de comprometimento com sigilo não-uniforme podem ser construídas, usando a mesma construção que no caso uniforme. Por conseguinte, temos o seguinte:

Teorema 4.4.6: *Suponha que existam funções não-uniformemente unidirecionais (como na Definição 2.2.6). Então existe um esquema de comprometimento-de-bit (como na Definição 4.4.1) para o qual a condição de sigilo também se verifica com respeito a circuitos de tamanho-polinomial.*

4.4.2 Prova de Conhecimento-Zero de Coloração de Grafos

Apresentar um sistema de prova de conhecimento-zero para uma linguagem $\mathcal{NP}$-completa implica a existência de um sistema de prova de conhecimento-zero para toda linguagem em $\mathcal{NP}$. Esse enunciado intuitivamente atraente de fato requer uma prova, que adiamos para um estágio mais na frente. Na seção corrente apresentamos um sistema de prova de conhecimento-zero para uma linguagem $\mathcal{NP}$-completa, especificamente 3-Colorabilidade de Grafos. Essa escolha é de fato arbitrária.

A linguagem *3-Coloração de Grafos*, representada por $3CG$, consiste de todos os grafos (finitos) simples (i.e., nenhuma aresta em paralelo ou laços)[13] que pode ser *vértice-colorido* usando três cores tais que nenhum par de vértices adjacentes recebem a mesma cor. Formalmente, um grafo $G = (V, E)$ é *3-colorível* se existe um mapeamento $\phi : V \to \{1, 2, 3\}$ tal que $\phi(u) \neq \phi(v)$ para todo $(u, v) \in E$.

4.4.2.1 Discussão Motivadora

A idéia subjacente ao sistema de prova de conhecimento-zero para $G3C$ é quebrar a prova da afirmação de que um grafo é 3-colorível em uma quantidade polinomial de *pedaços* arranjados em *modelos*[14] de modo que cada modelo por si só não produzirá nenhum conhecimento e mesmo assim todos os modelos colocados juntos garantirá a validade da afirmação principal. Suponha que o provador gera tais pedaços de informação, coloca cada um deles em um envelope selado, separado e não-transparente, e permite que o verificador abra e inspecione os pedaços participantes de um dos modelos. Então

[13] Um grafo finito simples é um par (V, E), onde V é um conjunto finito e E é um conjunto de subconjuntos-de-2 de V; isto é, $E \subseteq \{e \subseteq V : |e| = 2\}$. Os elementos de V são chamados de **vértices**, e os elementos de E são chamados de **arestas**. Embora cada aresta seja um par não-ordenado de dois elementos de V, usamos a notação de par-ordenado $(u, v) \in E$ ao invés da notação $\{u, v\} \in E$. Para $e = (u, v) \in E$, dizemos que u e v são as extremidades de e e que u é adjacente a v.

[14] Nota do tradutor: usamos 'modelo' como tradução para 'template'.

certamente o verificador não ganha nenhum conhecimento no processo, muito embora sua confiança na validade da afirmação (de que o grafo é 3-colorível) aumenta. Uma implementação concreta dessa idéia abstrata vai a seguir.

Para provar que o grafo $G = (V, E)$ é 3-colorível, o provador gera uma 3-coloração aleatória do grafo, representada por ϕ (na verdade uma re-rotulação de uma coloração fixa servirá). A cor de cada vértice constitui um pedaço da informação relativa à 3-coloração. O conjunto de modelos corresponde ao conjunto de arestas (i.e., cada par $(\phi(u), \phi(v))$, onde $(u, v) \in E$, constitui um modelos para a afirmação de que G é 3-colorível). Cada modelo separadamente (sendo meramente um par de elementos distintos em $\{1, 2, 3\}$) não dará origem a qualquer conhecimento. Entretanto, se todos os modelos estão OK (i.e., cada um contém um par de elementos distintos em $\{1, 2, 3\}$), então o grafo tem que ser 3-colorível. Consequentemente, grafos que não são 3-coloráveis têm que conter pelo menos um mau modelo e portanto será rejeitado com probabilidade observável. A seguir vai uma descrição abstrata do sistema de prova de conhecimento-zero resultante para $G3C$.

- *Entrada comum:* Um grafo simples $G = (V, E)$.
- *Primeiro passo do provador:* Seja ψ uma 3-coloração de G. O provador seleciona uma permutação aleatória π sobre $\{1, 2, 3\}$ e faz $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$ para cada $v \in V$. Daí, o provador forma uma re-rotulação aleatória da 3-coloração ψ. O provador envia ao verificador uma seqüência de $|V|$ caixas trancadas e não-transparentes tais que a v-ésima caixa contém o valor $\phi(v)$.
- *Primeiro passo do verificador:* O verificador seleciona uniformemente uma aresta $(u, v) \in E$ e a envia ao provador.
- *Observação motivadora:* O verificador pede para inspecionar as cores dos vértices u e v.
- *Segundo passo do provador:* O provador envia ao verificador as chaves para as caixas u e v.
- *Segundo passo do verificador:* O verificador abre as caixas u e v e aceita se e somente se elas contêm dois elementos diferentes de $\{1, 2, 3\}$.

Claramente, se o grafo de entrada é 3-colorível, então o provador pode fazer com que o verificador sempre aceite. Por outro lado, se o grafo de entrada não é 3-colorível, então qualquer conteúdo colocado nas caixas tem que estar inválido em pelo menos uma aresta, e consequentemente o verificador rejeitará com probabilidade no mínimo $1/|E|$. Daí, o protocolo acima exibe um intervalo observável nas probabilidades de aceitação entre o caso de entradas em $G3C$ e o caso de entradas que não estão em $G3C$. A propriedade de conhecimento-zero segue facilmente nesse cenário abstrato, porque pode-se simular a interação real colocando-se um par aleatório de cores diferentes nas caixas indicadas pelo verificador. Enfatizamos que esse argumento simples não será possível na implementação digital, porque as caixas não são totalmente livres de serem afetadas pelo seu conteúdo (mas sim, são afetadas, mesmo que de uma maneira indistingüível). Finalmente, observamos que a confiança na validade da afirmação (de que o grafo de entrada é 3-colorível) pode ser aumentada aplicando-se sequencialmente a prova acima um número suficiente de vezes. (Na verdade, se as caixas são perfeitas, como se supõe, então pode-se também usar repetições paralelas; entretanto, as caixas não são perfeitas na implementação digital apresentada a seguir).

4.4.2.2 A Prova Interativa

Agora nos voltamos para a implementação digital do protocolo abstrato. Nessa implementação as caixas são implementadas por um esquema de comprometimento. A saber, para cada caixa, invocamos uma execução independente do esquema de comprometimento. Isso nos habilitará a executar a fase de revelação para apenas alguns dos comprometimentos, uma propriedade que é crucial para nosso esquema. Por simplicidade de exposição, usamos o esquema simples de comprometimento apresentado na Construção 4.4.2 (ou, de forma geral, qualquer esquema de comprometimento de *interação-unidirecional*). Representamos por $C_r(\sigma)$ o comprometimento do emissor, usando moedas r, ao valor (ternário) σ.

Construção 4.4.7 (Uma Prova de Conhecimento-Zero para 3-Coloração de Grafo):

- Entrada comum: *Um grafo (3-colorível) simples $G = (V, E)$. Seja $n \stackrel{\text{def}}{=} |V|$ e $V = \{1, \ldots, n\}$.*
- Entrada auxiliar para o provador: *Uma 3-coloração de G, representada por ψ.*
- Primeiro passo do provador (P1): *O provador seleciona uma permutação π sobre $\{1, 2, 3\}$ e faz $\phi(v) \stackrel{\text{def}}{=} \phi(\psi(v))$ para cada $v \in V$. O provador usa o esquema de comprometimento propriamente dito para a cor de cada um dos vértices. A saber, o provador seleciona uniforme e independentemente $r_1, \ldots, r_n \in \{0,1\}^n$, computa $c_i = C_{r_i}(\phi(i))$ para cada $i \in V$, e envia $c_1, \ldots, c_n$ ao verificador.*
- Primeiro passo do verificador (V1): *O verificador seleciona uniformemente uma aresta $(u, v) \in E$ e a envia ao provador.*
- Segundo passo do provador (P2): *Sem perda de generalidade, podemos assumir que a mensagem recebida do verificador é uma aresta, representada por (u, v). (Do contrário, o provador fixa (u, v) como sendo uma aresta pré-determinada de G.) O provador usa a fase (canônica) de revelação do esquema de comprometimento de modo a revelar as cores dos vértices u e v ao verificador. A saber, o provador envia $(r_u, \phi(u))$ e $(r_v, \phi(v))$ ao verificador.*
- Segundo passo do verificador (V2): *O verificador verifica se os valores correspondentes ao comprometimentos u e v foram ou não recebidos corretamente e se esses valores são ou não diferentes. A saber, ao receber (r, σ) e (r', τ), o verificador verifica se $c_u = C_r(\sigma)$, $c_v = C_{r'}(\tau)$, e $\sigma \neq \tau$ ou não (e se ambos pertencem ou não a $\{1, 2, 3\}$). Se todas as condições se verificam, então o verificador aceita. Caso contrário, rejeita.*

Vamos representar esse programa do provador por P_{G3C}.

Enfatizamos que o programa do verificador e o do provador podem ser implementados em tempo polinomial probabilístico. No caso do programa do provador, essa propriedade é tornada possível pelo uso da *entrada auxiliar* para o provador. Como veremos adiante, o protocolo acima constitui uma prova interativa fraca para $G3C$. Como de costume, a confiança pode ser aumentada (i.e., a probabilidade de erro pode ser diminuída) por um número suficientemente grande de aplicações sucessivas. Entretanto, a mera existência de uma prova interativa para $G3C$ é óbvia (já que $G3C \in \mathcal{NP}$). O

arremate é que esse protocolo é de conhecimento-zero (também com respeito à entrada auxiliar). Usando o lema-da-composição-sequencial (Lema 4.3.11), segue que uma quantidade polinomial de aplicações sequenciais desse protocolo preservará a propriedade de conhecimento-zero.

Proposição 4.4.8: *Suponha que o esquema de comprometimento usado na Construção 4.4.7 satisfaça os requisitos de sigilo e desambigüidade (não-uniforme), Então a Construção 4.4.7 constitui uma prova interativa (generalizada) de conhecimento-zero com-entrada-auxiliar.*

Para uma maior discussão da Construção 4.4.7, veja a Seção 4.4.2.4.

4.4.2.3 O Simulador: Prova da Proposição 4.4.8

Primeiro provamos que a Construção 4.4.7 constitui uma prova interativa fraca para $G3C$. Suponha primeiro que o grafo de entrada é de fato 3-colorível. Então se o provador segue o programa especificado, o verificador sempre aceitará (i.e., aceitará com probabilidade 1). Por outro lado, se o grafo de entrada não é 3-colorível, então independentemente do que o provador faça, os n comprometimentos enviados no Passo P1 não podem corresponder a uma 3-coloração do grafo (pois tal coloração não existe). Frisamos que a única correspondência de comprometimentos a valores é garantida pela propriedade de desambigüidade do esquema de comprometimento. Segue que deve existir uma aresta $(u, v) \in E$ tal que c_u e c_v, enviados no Passo P1, não são comprometimentos a dois elementos *diferentes* de $\{1, 2, 3\}$. Daí, independentemente de como o provador se comporta, o verificador rejeitará com probabilidade pelo menos $1/|E|$. Por conseguinte, existe uma lacuna detectável (no comprimento de entrada) entre as probabilidades de aceitação no caso em que a entrada pertence a $G3C$ e no caso em que ela não pertence.

Mostraremos que P_{G3C}, o programa do provador especificado na Construção 4.4.7, é de fato de conhecimento-zero para $G3C$. A afirmação é provada sem referência a entrada-auxiliar (para o verificador), mas uma extensão do argumento a conhecimento-zero com-entrada-auxiliar é imediata. Novamente, usamos a formulação alternativa de conhecimento-zero (i.e., Definição 4.3.3) e mostramos como simular a visão de V^* da interação com P_{G3C} para toda máquina interativa probabilística de tempo-polinomial V^*. Como no caso do sistema de prova para Isomorfismo de Grafos (i.e., Construção 4.3.8), é fácil simular a visão do verificador da interação com P_{G3C}, *desde que* o verificador siga o programa especificado. Entretanto, precisamos simular a visão do verificador no caso geral (no qual o verificador usa um programa interativo de tempo-polinomial *arbitrário*). A seguir vai um panorama de nossa simulação (i.e., de nossa construção de um simulador M^* para um V^* arbitrário).

O simulador M^* incorpora o código

Bibliografia

Glossário de traduções

Tradução	**Termo original em inglês**
amarração	binding
encriptação	encryption
esquemas de encriptação	encryption schemes
ocultação	hiding
sigilo	secrecy
texto-cifrado	ciphertext
texto-puro	plaintext

Índice Remissivo