Prova Interativas

João Arthur Thiago Emmanuel

Coordenação de pós-graduação em Informática - COPIN

01/09/2008

Roteiro

Provas Convencionais

Teorema

For any positive integer $n, 1+2+...+n = n(n+1)/2$

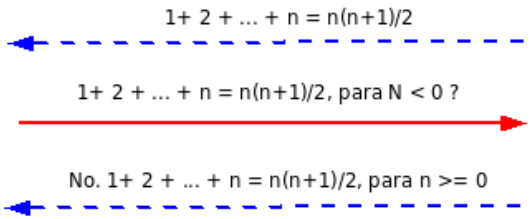
Demonstração.

- **Initial Step**. $P(1)$ asserts "$1 = 1(2)/2$", which is clearly true.
- **Inductive Step**. "If there is a k such that $P(k)$ is true, then (for this same k) $P(k+1)$ is true."
  $1+2+...+k+(k+1) = k(k+1)/2+(k+1) =$
  $(k(k+1)+2(k+1))/2 = (k+1)(k+2)/2.$

- O provador escreve todas as sentenças
- O verificador checa a corretude da prova. Não existe interação provador-verificador

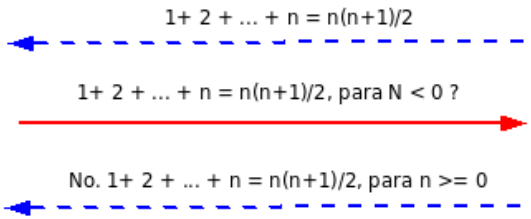
- O provador escreve todas as sentenças
- O verificador checa a corretude da prova. Não existe interação provador-verificador

Provas Interativas: Conceitos



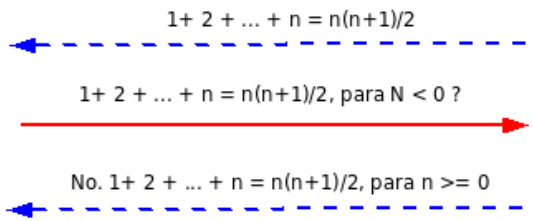
- Troca de mensagens entre o *provador* e o *verificador*
- Poder computacional:
  - Provador: Ilimitado
  - Verificador: Limitado

Provas Interativas: Conceitos



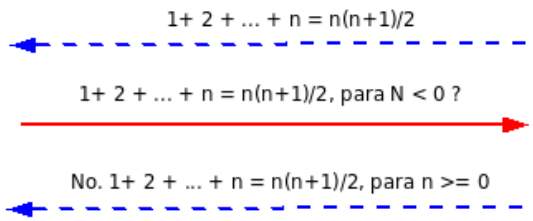
- Troca de mensagens entre o *provador* e o *verificador*
- Poder computacional:
  - Provador: Ilimitado
  - Verificador: Limitado

Provas Interativas: Conceitos



- Troca de mensagens entre o *provador* e o *verificador*
- Poder computacional:
  - Provador: Ilimitado
  - Verificador: Limitado

Provas Interativas: Formalismo

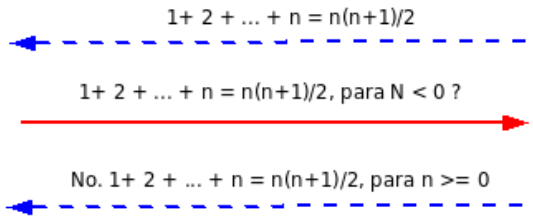


- As interações são modeladas por funções

$$a_1 = f(x)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, a_1, \dots, a_{2i})$$
$$a_{2i+2} = f(x, a_1, \dots, a_{2i+1})$$

- $(\langle f, g \rangle(x) - (2i+2) - round\ proof)$

Provas Interativas: Formalismo

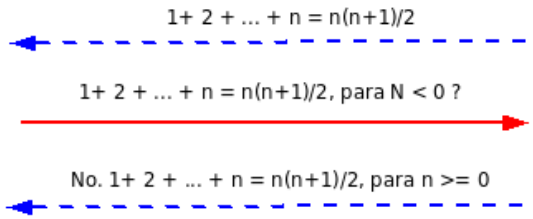


- As interações são modeladas por funções

$$a_1 = f(x)$$
$$a_2 = g(x, a_1)$$
$$\ldots$$
$$a_{2i+1} = f(x, a_1, \ldots, a_{2i})$$
$$a_{2i+2} = f(x, a_1, \ldots, a_{2i+1})$$

- $(\langle f, g \rangle(x) - (2i+2) - round\ proof)$

Provas Interativas: Formalismo

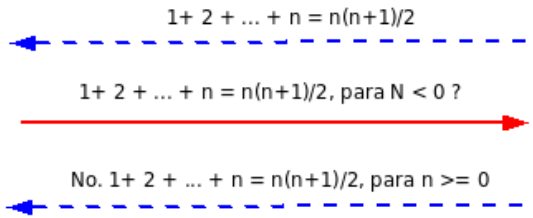


- As interações são modeladas por funções

$$a_1 = f(x)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, a_1, \dots, a_{2i})$$
$$a_{2i+2} = f(x, a_1, \dots, a_{2i+1})$$

- $(\langle f, g \rangle(x) - (2i+2) - round\ proof)$

Provas Interativas: Formalismo

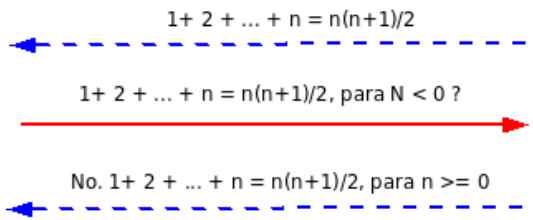


- As interações são modeladas por funções

$$a_1 = f(x)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, a_1, \dots, a_{2i})$$
$$a_{2i+2} = f(x, a_1, \dots, a_{2i+1})$$

- $(\langle f, g \rangle(x) - (2i+2) - round\ proof)$

Provas Interativas: Formalismo



- As interações são modeladas por funções

$$a_1 = f(x)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, a_1, \dots, a_{2i})$$
$$a_{2i+2} = f(x, a_1, \dots, a_{2i+1})$$

- $(\langle f, g \rangle(x) - (2i+2) - \textit{round proof})$

Provas Interativas: Propriedades

- *Completeness* - Qualquer teorema verdadeiro pode ser provado
- *Soundness* - Nenhum teorema falso pode ser tido como verdadeiro

Definição

- output
  - $out_f\langle f, g\rangle(x) = f(x, a_1, ..., a_k)$
  - $out_g\langle f, g\rangle(x) = g(x, a_1, ..., a_k)$

Definição

- *Completeness* $x \in L \Rightarrow \exists P\colon \{0,1\}^* \rightarrow \{0,1\}^* out_v\langle V, P\rangle(x) = 1$
- *Soundness* $x \notin L \Rightarrow \forall P\colon \{0,1\}^* \rightarrow \{0,1\}^* out_v\langle V, P\rangle(x) = 1$

Provas Interativas: Classe dIP

Definição

Uma linguagem L possui um $k - round$ deterministic interactive proof system, se existir uma *TM* que execute em tempo polinomial com a entrada $x, a_1, \ldots, a_k$, satisfazendo as propriedades *Completeness* e *Soundness*.

Definição

- **dIP** - Classe das linguagens que possuem $k - round$ deterministic interactive proof system

$NP = dIP$

Provas Interativas: Classe IP

$$a_1 = f(x, r)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, r, a_1, \dots, a_{2i})$$

Definição

IP - Uma linguagem pertece a $IP[k]$ se existir uma TM que execute em tempo polinomial com a entrada $x, r, a_1, \dots, a_k$ e que satisfaça as seguintes propriedades:

- *Completeness* $x \in L \Rightarrow \exists P Pr[out_V\langle V, P\rangle(x) = 1] \geqslant 2/3$
- *Soundness* $x \notin L \Rightarrow \forall P Pr[out_V\langle V, P\rangle(x) = 1] \leq 1/3$

Provas Interativas: Classe IP

$$a_1 = f(x, r)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, r, a_1, \dots, a_{2i})$$

Definição

IP - Uma linguagem pertece a $IP[k]$ se existir uma TM que execute em tempo polinomial com a entrada $x, r, a_1, \dots, a_k$ e que satisfaça as seguintes propriedades:

- *Completeness* $x \in L \Rightarrow \exists P Pr[out_v\langle V, P\rangle(x) = 1] \geqslant 2/3$
- *Soundness* $x \notin L \Rightarrow \forall P Pr[out_v\langle V, P\rangle(x) = 1] \leq 1/3$

Provas Interativas: Classe IP

$$a_1 = f(x, r)$$
$$a_2 = g(x, a_1)$$
$$\dots$$
$$a_{2i+1} = f(x, r, a_1, \dots, a_{2i})$$

Definição

IP - Uma linguagem pertece a $IP[k]$ se existir uma TM que execute em tempo polinomial com a entrada $x, r, a_1, \dots, a_k$ e que satisfaça as seguintes propriedades:

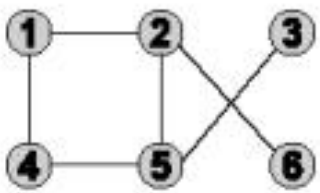
- *Completeness* $x \in L \Rightarrow \exists P Pr[out_v\langle V, P\rangle(x) = 1] \geqslant 2/3$
- *Soundness* $x \notin L \Rightarrow \forall P Pr[out_v\langle V, P\rangle(x) = 1] \leq 1/3$

Introdução

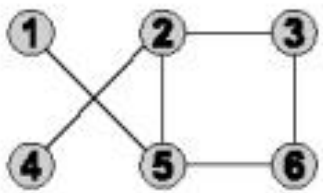
- Grafos
  - Importante objeto de estudo da computação
  - Modelagem de problemas
  - Representados por matrizes, listas de adjacência, incidência etc.
  - Algoritmos

Isomorfismo

Dois grafos G1 e G2 são isomorfos ($G1 \equiv G2$) se existe uma permutação π dos identificadores dos nós tal que $\pi(G1) = G2$.



Γ_1



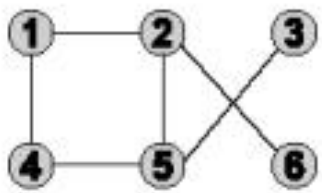
Γ_2

Introdução

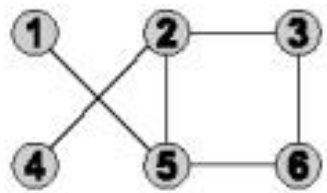
- Grafos
  - Importante objeto de estudo da computação
  - Modelagem de problemas
  - Representados por matrizes, listas de adjacência, incidência etc.
  - Algoritmos

Isomorfismo

Dois grafos G1 e G2 são isomorfos $(G1 \equiv G2)$ se existe uma permutação π dos identificadores dos nós tal que $\pi(G1) = G2$.



Γ_1



Γ_2

O Problema-Desafio

Dasafio

Decidir se dois grafos são isomorfos.

- GI está em NP, visto que um certificado é a descrição da permutação π.
- Por que é importante?
  - GI e o problema da fatoração são os representantes mais famosos de NP que não se sabe se estão em P ou NP-Completo.

O Problema-Desafio

Dasafio

Decidir se dois grafos são isomorfos.

- GI está em NP, visto que um certificado é a descrição da permutação π.
- Por que é importante?
  - GI e o problema da fatoração são os representantes mais famosos de NP que não se sabe se estão em P ou NP-Completo.

O Problema-Desafio

Dasafio

Decidir se dois grafos são isomorfos.

- GI está em NP, visto que um certificado é a descrição da permutação π.
- Por que é importante?
  - GI e o problema da fatoração são os representantes mais famosos de NP que não se sabe se estão em P ou NP-Completo.

Prova

Complemento de GI - GNI

Decidir se dois grafos não são isomorfos.

- Protocolo: *Private-coin Graph Non-isomorphism*
  - **V**: escolha $i \in \{1,2\}$ randomicamente (uniformemente). Permute os vértices de Gi para obter um novo grafo H. Envie H para **P**.
  - **P**: Identifique qual dos grafos G1 e G2 foi usado para gerar H. Seja Gj este grafo. Envie j para **V**.
  - **V**: Aceita se $i = j$; rejeita caso contrário.

Prova

Complemento de GI - GNI

Decidir se dois grafos não são isomorfos.

- Protocolo: *Private-coin Graph Non-isomorphism*
  - **V**: escolha $i \in \{1,2\}$ randomicamente (uniformemente). Permute os vértices de Gi para obter um novo grafo H. Envie H para **P**.
  - **P**: Identifique qual dos grafos G1 e G2 foi usado para gerar H. Seja Gj este grafo. Envie j para **V**.
  - **V**: Aceita se $i = j$; rejeita caso contrário.

Prova

Complemento de GI - GNI

Decidir se dois grafos não são isomorfos.

- Protocolo: *Private-coin Graph Non-isomorphism*
  - **V**: escolha $i \in \{1,2\}$ randomicamente (uniformemente). Permute os vértices de Gi para obter um novo grafo H. Envie H para **P**.
  - **P**: Identifique qual dos grafos G1 e G2 foi usado para gerar H. Seja Gj este grafo. Envie j para **V**.
  - **V**: Aceita se $i = j$; rejeita caso contrário.

Prova

Complemento de GI - GNI

Decidir se dois grafos não são isomorfos.

- Protocolo: *Private-coin Graph Non-isomorphism*
  - **V**: escolha $i \in \{1,2\}$ randomicamente (uniformemente). Permute os vértices de Gi para obter um novo grafo H. Envie H para **P**.
  - **P**: Identifique qual dos grafos G1 e G2 foi usado para gerar H. Seja Gj este grafo. Envie j para **V**.
  - **V**: Aceita se $i = j$; rejeita caso contrário.

Prova

- O protocolo acima satisfaz a definição 2, pois:
  - Se $(G1 \ncong G2)$, então existe um provador tal que:

$$Pr[Vaceita] = 1$$

    - Pelo fato de não serem isomorfos, um provador pode dizer com certeza qual dos dois é isomorfo à H
  - Se $(G1 \equiv G2)$ o provador pode apenas adivinhar, pois uma permutação em G1 é idêntica a uma permutação em G2. Neste caso, para todo provador:

$$Pr[Vaceita] \leq 1/2$$

    - A probabilidade pode ser reduzida para $1/3$ por repetições paralelas ou sequenciais

Moedas públicas e AM

- Provas interativas com moedas públicas: Modelo em que o provador tem acesso irrestrito às *strings* randômicas do verificador
- Metáfora Rei-Mágico (Arthur e Merlin).

Definição

$\forall k$, AM[k] é a classe das linguagens que podem ser decididas por uma prova interativa de k rodadas, onde as mensagens do verificador consistem em enviar uma *string* de tamanho polinomial e essas mensagens englobam todas as moedas lançadas anteriormente pelo verificador.

Moedas públicas e AM

- Provas interativas com moedas públicas: Modelo em que o provador tem acesso irrestrito às *strings* randômicas do verificador
- Metáfora Rei-Mágico (Arthur e Merlin).

Definição

$\forall k$, AM[k] é a classe das linguagens que podem ser decididas por uma prova interativa de k rodadas, onde as mensagens do verificador consistem em enviar uma *string* de tamanho polinomial e essas mensagens englobam todas as moedas lançadas anteriormente pelo verificador.

Moedas públicas e AM

Definição

AM[2] é a classe das linguagens com provas interativas que consiste no envio de uma mensagem com uma string **randômica** pelo verificador, o provador responde com uma mensagem e a decisão de aceitar é obtida aplicando uma função polinomial determinística à transcrição.

$$AM = AM[2]$$

- Nesta representação, Arthur (verificador) é uma máquina probabilística polinomial, enquanto Merlin (o provador) não possui limites de recursos computacionais
- Merlin não é totalmente confiável
- Arthur deve analisar as respostas e decidir por si só

Moedas públicas e AM - Quando um problema é decidível?

- Um problema é considerado decidível se:
  - Sempre que houver uma resposta “sim”, Merlin tem uma série de respostas que levam a Arthur aceitar no mínimo $2/3$ das vezes
  - Sempre que houver uma resposta “não”, Arthur nunca vai aceitar mais do que $1/3$ das vezes

Neste caso, Arthur age como um verificador BPP, assumindo que ele alocou tempo polinomial para decidir e fazer as perguntas

A classe MA

Definição

Classe dos problemas de decisão resolvíveis pelo protocolo MA

- Merlin envia para Arthur uma prova de tamanho polinomial cuja resposta é “sim”
- Arthur tem que verificar a prova em BPP (polinomial probabilística)
  - Sim: Existe uma prova tal que Arthur aceita com probabilidade no mínimo $2/3$
  - Não: Para todas as provas, Arthur aceita com probabilidade no máximo $1/3$

Observações

Observação

$AM[k] \subseteq IP[K]$

- A prova para GNI depende do fato de P não ver os bits randômicos de V. Do contrário, P conheceria *i* e adivinharia sempre corretamente.
- Surpreendentemente verifica-se que este sistema é tão poderoso quanto o de moedas privadas, em que o verificador não precisa enviar suas moedas ao provador

Relações entre as classes

Teorema

AM[k] = AM, exceto pelo fato de que há k iterações entre Arthur e Merlin

Teorema

Para k>2, $AM[k] = AM[2] = AM$

Relações entre as classes

Teorema

AM[k] = AM, exceto pelo fato de que há k iterações entre Arthur e Merlin

Teorema

Para k>2, $AM[k] = AM[2] = AM$

GNI está em AM[k]

Teorema

$GNI \in AM[K]$ para algum $K \geq 2$

- Re-modelando o problema
- $S = \{H : H \equiv G1 \text{ ou } H \equiv G2\}$
- permutação de G1 ou G2
- n! grafos equivalentes
  - Se $G1 \not\equiv G2$ então $|S| = 2n!$
  - Se $G1 \equiv G2$ então $|S| = n!$
- No caso geral, G1 ou G2 tem menos que n! grafos equivalentes

GNI está em AM[k]

Teorema

$GNI \in AM[K]$ para algum $K \geq 2$

- Re-modelando o problema
- $S = \{H : H \equiv G1 \text{ ou } H \equiv G2\}$
- permutação de G1 ou G2
- n! grafos equivalentes
  - Se $G1 \not\equiv G2$ então $|S| = 2n!$
  - Se $G1 \equiv G2$ então $|S| = n!$
- No caso geral, G1 ou G2 tem menos que n! grafos equivalentes

GNI está em AM[k]

Continuação...

Definição

$S = \{(H, \pi)\ H \equiv G1 \text{ ou } H \equiv G2 \text{ e } \pi \in aut(H)\ \}$

- Onde $\pi \in aut(H)$ se $\pi(H) = H$
- Convencer o verificador de que $G1 \not\equiv G2$

Set lower bound protocol!

GNI está em AM[k]

Continuação...

Definição

$S = \{(H, \pi)\ H \equiv G1$ ou $H \equiv G2$ e $\pi \in aut(H)\ \}$

- Onde $\pi \in aut(H)$ se $\pi(H) = H$
- Convencer o verificador de que $G1 \not\equiv G2$

Set lower bound protocol!

Set lower bound protocol

Protocolo que completa a prova 7.

- Provador mostra ao verificador que, dado um conjunto S:
  - Se $|S| > K$, o provador leva o verificador a aceitar com alta probabilidade
  - Se $|S| \leq K/2$, o verificador vai rejeitar com alta probabilidade, não importando o que o provador faça

Pairwise independent hash functions

- *Universal Hashing*: Algoritmo para escolher uma função *hash* **F** tal que:
  - Para duas entradas distintas x e y, a probabilidade de uma colisão é a mesma se a função fosse randômica
  - $F(x) \leq 1/|r|$

Two-Universal Family

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{|V|} \quad (1)$$

Strongly Two-Universal Family (Pairwise idependent hash functions)

$$\Pr_{h \in H}[h(x) = x' \text{ and } h(y) = y'] = \frac{1}{2^{2n}} \quad (2)$$

Pairwise independent hash functions

- *Universal Hashing*: Algoritmo para escolher uma função *hash* **F** tal que:
  - Para duas entradas distintas x e y, a probabilidade de uma colisão é a mesma se a função fosse randômica
  - $F(x) \leq 1/|r|$

Two-Universal Family

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{|V|} \tag{1}$$

Strongly Two-Universal Family (Pairwise idependent hash functions)

$$\Pr_{h \in H}[h(x) = x' \text{ and } h(y) = y'] = \frac{1}{2^{2n}} \tag{2}$$

Pairwise independent hash functions

- *Universal Hashing*: Algoritmo para escolher uma função *hash* **F** tal que:
  - Para duas entradas distintas x e y, a probabilidade de uma colisão é a mesma se a função fosse randômica
  - $F(x) \leq 1/|r|$

Two-Universal Family

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{|V|} \tag{1}$$

Strongly Two-Universal Family (Pairwise idependent hash functions)

$$\Pr_{h \in H}[h(x) = x' \text{ and } h(y) = y'] = \frac{1}{2^{2n}} \tag{2}$$

GoldWasserSipser Set LowerBound

- Condições:
  - $S \subseteq \{0,1\}^m$ é um conjunto cujo membros podem ser certificados
  - Ambos conhecem **K**
  - Provador precisa convencer o verificador que $|S| \geq K$
  - Verificador deve rejeitar se $|S| \leq \frac{K}{2}$
  - $2^{k-2} \leq K \leq 2^{k-1}$
- Protocolo:
  - **V**: Escolha randomicamente uma função $h : \{0,1\}^m \rightarrow \{0,1\}^k$. Escolha $y \in R\{0,1\}^k$ e envie h, y ao provador
  - **P**: Tente achar um $x \in S$ tal que $h(x) = y$. Envie esse x para **V**, juntamente com um certificado de que $x \in S$
  - **Saída de V**: Se o certificado valida que $x \in S$ e $h(x) = y$, aceita. Rejeita caso contrário.

GoldWasserSipser Set LowerBound

- Condições:
  - $S \subseteq \{0,1\}^m$ é um conjunto cujo membros podem ser certificados
  - Ambos conhecem **K**
  - Provador precisa convencer o verificador que $|S| \geq K$
  - Verificador deve rejeitar se $|S| \leq \frac{K}{2}$
  - $2^{k-2} \leq K \leq 2^{k-1}$
- Protocolo:
  - **V**: Escolha randomicamente uma função $h : \{0,1\}^m \rightarrow \{0,1\}^k$. Escolha $y \in R\{0,1\}^k$ e envie h, y ao provador
  - **P**: Tente achar um $x \in S$ tal que $h(x) = y$. Envie esse x para **V**, juntamente com um certificado de que $x \in S$
  - **Saída de V**: Se o certificado valida que $x \in S$ e $h(x) = y$, aceita. Rejeita caso contrário.

GoldWasserSipser Set LowerBound

- Condições:
  - $S \subseteq \{0,1\}^m$ é um conjunto cujo membros podem ser certificados
  - Ambos conhecem **K**
  - Provador precisa convencer o verificador que $|S| \geq K$
  - Verificador deve rejeitar se $|S| \leq \frac{K}{2}$
  - $2^{k-2} \leq K \leq 2^{k-1}$
- Protocolo:
  - **V**: Escolha randomicamente uma função $h : \{0,1\}^m \rightarrow \{0,1\}^k$. Escolha $y \in R\{0,1\}^k$ e envie h, y ao provador
  - **P**: Tente achar um $x \in S$ tal que $h(x) = y$. Envie esse x para **V**, juntamente com um certificado de que $x \in S$
  - **Saída de V**: Se o certificado valida que $x \in S$ e $h(x) = y$, aceita. Rejeita caso contrário.

GoldWasserSipser Set LowerBound

- Condições:
  - $S \subseteq \{0,1\}^m$ é um conjunto cujo membros podem ser certificados
  - Ambos conhecem **K**
  - Provador precisa convencer o verificador que $|S| \geq K$
  - Verificador deve rejeitar se $|S| \leq \frac{K}{2}$
  - $2^{k-2} \leq K \leq 2^{k-1}$
- Protocolo:
  - **V**: Escolha randomicamente uma função $h : \{0,1\}^m \rightarrow \{0,1\}^k$. Escolha $y \in R\{0,1\}^k$ e envie h, y ao provador
  - **P**: Tente achar um $x \in S$ tal que $h(x) = y$. Envie esse x para **V**, juntamente com um certificado de que $x \in S$
  - **Saída de V**: Se o certificado valida que $x \in S$ e $h(x) = y$, aceita. Rejeita caso contrário.

Prova para GNI

- A prova consiste em várias iterações entre verificador e provador
- Usa *set lower bound protocol* para o conjunto S
- O verificador aceita se a fração de iterações que aceitam for no mínimo 0.6p

Private coin X Public coin

- *Set lower bound protocol* não possui a propriedade de completude
- No entanto, qualquer protocolo de moedas privadas pode ser transformado em um protocolo perfeito e completo de moedas públicas, com um número similar de rodadas

IP=PSPACE

Teorema

$IP = PSPACE$

Demonstração.

1. $IP \subseteq PSPACE$ - Simular a execução de um prova interativa por uma máquina $PSPACE$
2. $PSPACE \subseteq IP$ - Mostrar que um problema $PSPACE - complete \in IP$

$TQBF \subseteq PSPACE$

Demonstração.

Aritmetização de TBQF

1. Substituir cada variável booleana x por um inteiro z
2. Substituir cada ocorrência da negação de x por $(1 - z)$
3. Substituir: $\wedge$ pela operação de multiplicação, $\vee$ pela operação de soma, $\exists$ por um somatório e $\forall$ pelo produtório.

Um TBQF é verdadeiro se o seu valor númerico é não negativo $\square$

Checagem de Programas

- Verificação de programas é indecidível
- *Program Checker*: Um programa **C** que executa outro programa **P** para uma entrada x e detecta se a saída está incorreta
- **C** precisa computar a saída de **P** para outras entradas
- Também chamada de *instance checking*

Observacão

Program testing can be used to show the presence of bugs, but never to show their absence.

Checker

Seja **P** um programa que computa uma tarefa **T**.

- Checador (**C**) é uma máquina polinomial que, dada uma entrada **x**, possui o seguinte comportamento:
  - Se $\forall y, P(y) = T(y)$, então $P[C^P \text{ aceitar } P(x)] \geq \frac{2}{3}$
  - Se $P(x) \neq T(x)$, então $P[C^P \text{ aceitar } P(x)] < \frac{1}{3}$

GNI checker

Para muitos problemas, construir um checador pode ser mais fácil que construir a solução.

- GNI possui um checador eficiente:
  - Protocolo definido anteriormente
  - No caso do provador admitir que $G1 \ncong G2$, repita K vezes:
  - Escolha $i \in R\{1,2\}$. Permute Gi randomicamente em H
  - Pergunte ao provador <G1, H>; <G2, H> e cheque para ver se a primeira resposta é consistente
- Se uma linguagem L possui um sistema de prova interativo onde o provador pode ser implementado usando acesso a oráculo, então esta linguagem possui um checador

GNI checker

Para muitos problemas, construir um checador pode ser mais fácil que construir a solução.

- GNI possui um checador eficiente:
  - Protocolo definido anteriormente
  - No caso do provador admitir que $G1 \neq G2$, repita K vezes:
  - Escolha $i \in R\{1,2\}$. Permute Gi randomicamente em H
  - Pergunte ao provador <G1, H>; <G2, H> e cheque para ver se a primeira resposta é consistente
- Se uma linguagem L possui um sistema de prova interativo onde o provador pode ser implementado usando acesso a oráculo, então esta linguagem possui um checador

GNI checker

Para muitos problemas, construir um checador pode ser mais fácil que construir a solução.

- GNI possui um checador eficiente:
  - Protocolo definido anteriormente
  - No caso do provador admitir que $G1 \not\equiv G2$, repita K vezes:
  - Escolha $i \in R\{1,2\}$. Permute Gi randomicamente em H
  - Pergunte ao provador <G1, H>; <G2, H> e cheque para ver se a primeira resposta é consistente
- Se uma linguagem L possui um sistema de prova interativo onde o provador pode ser implementado usando acesso a oráculo, então esta linguagem possui um checador

GNI checker

Para muitos problemas, construir um checador pode ser mais fácil que construir a solução.

- GNI possui um checador eficiente:
  - Protocolo definido anteriormente
  - No caso do provador admitir que $G1 \not\equiv G2$, repita K vezes:
  - Escolha $i \in R\{1,2\}$. Permute Gi randomicamente em H
  - Pergunte ao provador <G1, H>; <G2, H> e cheque para ver se a primeira resposta é consistente
- Se uma linguagem L possui um sistema de prova interativo onde o provador pode ser implementado usando acesso a oráculo, então esta linguagem possui um checador

GNI checker

Para muitos problemas, construir um checador pode ser mais fácil que construir a solução.

- GNI possui um checador eficiente:
  - Protocolo definido anteriormente
  - No caso do provador admitir que $G1 \not\equiv G2$, repita K vezes:
  - Escolha $i \in R\{1,2\}$. Permute Gi randomicamente em H
  - Pergunte ao provador <G1, H>; <G2, H> e cheque para ver se a primeira resposta é consistente
- Se uma linguagem L possui um sistema de prova interativo onde o provador pode ser implementado usando acesso a oráculo, então esta linguagem possui um checador

GNI checker

Para muitos problemas, construir um checador pode ser mais fácil que construir a solução.

- GNI possui um checador eficiente:
  - Protocolo definido anteriormente
  - No caso do provador admitir que $G1 \not\equiv G2$, repita K vezes:
  - Escolha $i \in R\{1,2\}$. Permute Gi randomicamente em H
  - Pergunte ao provador <G1, H>; <G2, H> e cheque para ver se a primeira resposta é consistente
- Se uma linguagem L possui um sistema de prova interativo onde o provador pode ser implementado usando acesso a oráculo, então esta linguagem possui um checador