A Hierarquia de Tempo Não Determinístico

Nicollas Sdroievski

25 de Março de 2019

1 Ideia Geral

- **Dificuldade:** como lidamos com MTs não determinísticas, não é claro como negar a resposta de maneira fácil assim como fizemos com MTs determinísticas.

Relembramos alguns fatos sobre máquinas não determinísticas.

- Existe uma MT não determinística universal com *overhead* de simulação constante. Isso indica que, em teoria, poderíamos obter um teorema tão forte quando o teorema de hierarquia de espaço determinístico.
- É possível simular uma máquina não determinística de tempo $f(n)$ em tempo determinístico $2^{f(n)}$.

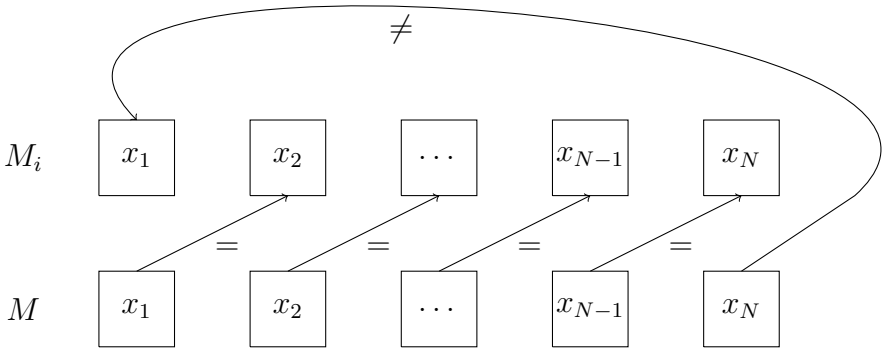
Usando a simulação determinística, poderíamos obter um teorema de hierarquia com *gap* gigantesco, pois em tempo exponencial podemos negar a resposta de uma máquina não determinística.

Ideia: simular máquinas com menos tempo em um intervalo exponencialmente grande, garantindo pelo menos uma resposta diferente $\rightarrow$ *Lazy Diagonalization*.

Digamos que a MTND M está tentando negar pelo menos uma resposta da MTND M_i em um determinado intervalo $I_i = \{x_1, x_2, \ldots, x_N\}$. Imagine o seguinte comportamento para $M(x_j)$:

1. $M(x_j) = M_i(x_{j+1})$ caso $j < N$. Para isso basta que M possa simular (não deterministicamente) o comportamento de M_i.

2. $M(x_N) \neq M_i(x_1)$. Para isso é preciso que M simule deterministicamente o comportamento de M_i e então negue sua resposta. Nesse caso é preciso que o tamanho de x_N seja exponencialmente maior que o tamanho de x_1.



Nesse caso, é garantido que para pelo menos um elemento x_j do intervalo I_i, as respostas de $M(x_j)$ e $M_i(x_j)$ sejam diferentes (**Exercício**).

2 Teorema e Demonstração

Teorema 1. *Sejam h e g duas funções tempo-construtíveis tais que $h(n+1) = o(g(n))$, então* $\mathsf{NTIME}(h(n)) \subsetneq \mathsf{NTIME}(g(n))$.

Demonstração. Vamos mostrar o resultado mais simples $\mathsf{NTIME}(n) \subsetneq \mathsf{NTIME}(n^{1.5})$, e para isso apresentamos uma linguagem L tal que $L \in \mathsf{NTIME}(n^{1.5})$ porém $L \notin NTIME(n)$. L será definida a partir da MT M que a decide usando as ideias da seção anterior.

O intervalo I_i no qual M tentará negar uma resposta de M_i será determinado pela função $f : \mathbb{N} \to \mathbb{N}$ definida por $f(1) = 2$ e $f(i+1) = 2^{f(i)^{1.2}}$. É **possível**, dado n, determinar o valor de i tal que que $f(i) < n \leq f(i+1)$ em tempo $O(n^{1.5})$. O conjunto I_i no qual M negará alguma resposta de M_i será o conjunto $\{1^n \mid f(i) < n \leq f(i+1)\}$. Definimos o comportamento de M:

Na entrada x, se x possui algum zero, rejeite. Caso $x = 1^n$, compute i tal que $f(i) < n \leq f(i+1)$ e então

1. Se $n < f(i+1)$, então simule (não deterministicamente) M_i na entrada 1^{n+1} com limite de tempo $n^{1.1}$ e responda o que M_i responder (se M_i não parar nesse tempo, então pare e responda não).

2. Se $n = f(i+1)$, então simule deterministicamente $M_i(1^{f(i)+1})$ com limite de tempo $(f(i)+1)^{1.1}$ e negue sua resposta. Novamente, se M_i não parar nesse tempo, pare e responda não.

Perceba que a condição de que $h(n+1) = o(g(n))$ é necessária para garantir que a máquina M consiga cumprir a parte 1 e simular, na entrada 1^n, a máquina M_i na entrada 1^{n+1}. Além disso, perceba que para negar a resposta de M_i com $1^{f(i)+1}$ na parte 2 é necessário tempo $O(2^{(f(i)+1)^{1.1}})$, o que não é um problema pois o tamanho da entrada é $2^{f(i)^{1.2}}$.

Junto com as ideias da seção anterior, terminamos a demonstração. □

Completando a demonstração: fiquei confuso com a parte da demonstração que diz que é fácil encontrar o valor de i tal que $f(i) < n \leq f(i+1)$ em tempo $n^{1.5}$, então pesquisei um pouco e encontrei algumas coisas interessantes.

Para encontrar esse valor seria necessário testar valores de i iniciando em 1 até que $f(i+1) \geq n$, caso em que $n > f(i)$. Como o crescimento de f é maior que exponencial, o tempo para realizar essas operações é dominado pelo tempo de calcular $f(i+1)$, o último valor.

Também vi que a função $x \mapsto 2^x$ para $x \in \mathbb{N}$ pode ser computada em tempo $O(x)$ (levando em consideração que $|x| = O(\log x)$ e $|2^x| = O(x)$, esse tempo é exponencial). Para isso, basta escrever 1 e então x zeros.

Para calcular $f(i+1)$ precisamos computar (em binário) $2^{f(i)^{1.2}}$. Como $n > f(i)$, a complexidade de tempo para computar esse valor é $O(n^{1.2})$. Como o valor de n é recebido em unário, esse tempo não só é polinomial como está dentro do limite de $n^{1.5}$ que foi estabelecido.

Curiosidade: A condição de que $h(n+1) = o(f(n))$ faz com que o teorema falhe para algumas funções super exponenciais. Esse é o caso de $h(n) = 2^{2^{n-1}}$ e $f(n) = 2^{2^n}$. Nesse caso temos $h(n) = o(f(n))$ porém $h(n+1) = f(n)$.