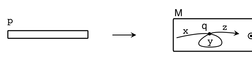
Lema do bombeamento

A ideia é saber se existe uma versão do lema do bombeamento para os autômatos de pilha.

Quer dizer, a linguagem $\mathtt{a^n b^n a^n}$ é uma linguagem infinita, que possui palavras arbitrariamente grandes.

Mas, um autômato de pilha sempre tem um número finito de estados.

Logo, a computação de um autômato de pilha sobre uma palavra muito, muito grande vai acabar passando mais de uma vez por algum estado $\mathtt{q}$



Sim, é verdade.

Mas, o diabo é a pilha ...

Quer dizer, o conteúdo da pilha no momento das duas passagens pelo estado $\mathtt{q}$ podem ser diferentes.

Daí que, nós não temos a garantia de que após a remoção (ou a duplicação) do fragmento $\mathtt{y}$, a computação ainda vai terminar no estado final — (*com a pilha vazia*)

E daí, o argumento do bombeamento já não funciona mais.

Certo.

Mas a gente pode levar a pilha em consideração.

E daí, a ideia fica assim

Quer dizer,

→ *Se o autômato passar duas vezes pelo estado* $\mathtt{q}$ *com o mesmo conteúdo na pilha, então o fragmento* $\mathtt{y}$ *(lido entre as duas passagens) pode ser removido ou bombeado sem afetar o resultado final da computação.*

Sim, isso é verdade.

Mas, o diabo é que a pilha pode ficar arbitrariamente grande ...

Daí que, nós não temos a garantia de que o autômato vai estar duas vezes no mesmo estado com o mesmo conteúdo na pilha durante a computação.

E daí, o argumento do bombeamento falha mais uma vez.

Certo.

Então a solução é pegar o diabo pelo chifre.

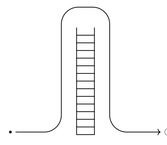
Quer dizer, imagine que a pilha do autômato fica arbitrariamente grande quando ele recebe as palavras da linguagem $\mathtt{L}$.

E considere uma palavra muito, muito grande da linguagem $\mathtt{L}$

Legal.

O truque é olhar para a pilha — (*ou o chifre ...*)

Quer dizer, a ideia é que a pilha vai ficar muito, muito grande durante a computação — (*e depois ela vai voltar a ficar vazia*)



E agora a gente tem que ver a repetição.

Bom, para ver a repetição a gente concentra a atenção sobre um símbolo qualquer da pilha

E daí, a gente observar que a computação passa duas vezes por esse ponto

subida: o símbolo estava no topo, e o autômato colocou outro símbolo sobre ele
descida: o símbolo estava no topo, e o autômato removeu ele da pilha

E a gente anota os dois estados em que o autômato estava quando essas coisas aconteceram

Daí a gente pode imaginar que essas anotações são feitas para todos os símbolos da pilha

Bom, o número de estados do autômato é finito.

E o número de símbolos que podem ser colocados em uma posição da pilha é finito.

Daí que, o número de triplas $(\mathtt{q_i}, \mathtt{a}, \mathtt{q_j})$ é finito.

Daí que, se a pilha é realmente muito, muito grande, vão haver dois pontos da pilha que tem a mesma anotação

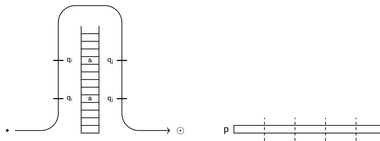
Pronto!

Agora nós vimos a repetição na computação do autômato de pilha.

E agora o resto é moleza ...

Quer dizer, o primeiro passo é quebrar a computação nos momentos de subida e descida associados a esses dois pontos.

E daí, claro, isso nos dá uma decomposição da palavra de entrada em 5 partes



Agora, é bem fácil ver que

→ *Após a remoção (ou a duplicação) dos fragmentos* $\mathtt{v}$ *e* $\mathtt{y}$ *da palavra de entrada a computação ainda termina no estado final — (com a pilha vazia)*

Quer dizer, é assim que as linguagens reconhecidas por autômatos de pilha bombeiam

$$\mathtt{u\,v\,x\,y\,z} \quad \in \quad \mathtt{L} \qquad \Rightarrow \qquad \mathtt{u\,x\,z}, \quad \mathtt{u\,v\,v\,x\,y\,y\,z} \quad \in \quad \mathtt{L}$$

— (*onde a gente não sabe quem são* $\mathtt{u}, \mathtt{v}, \mathtt{x}, \mathtt{y}, \mathtt{z}$, *e precisa considerar todas as possibilidades*)

Legal, não é?