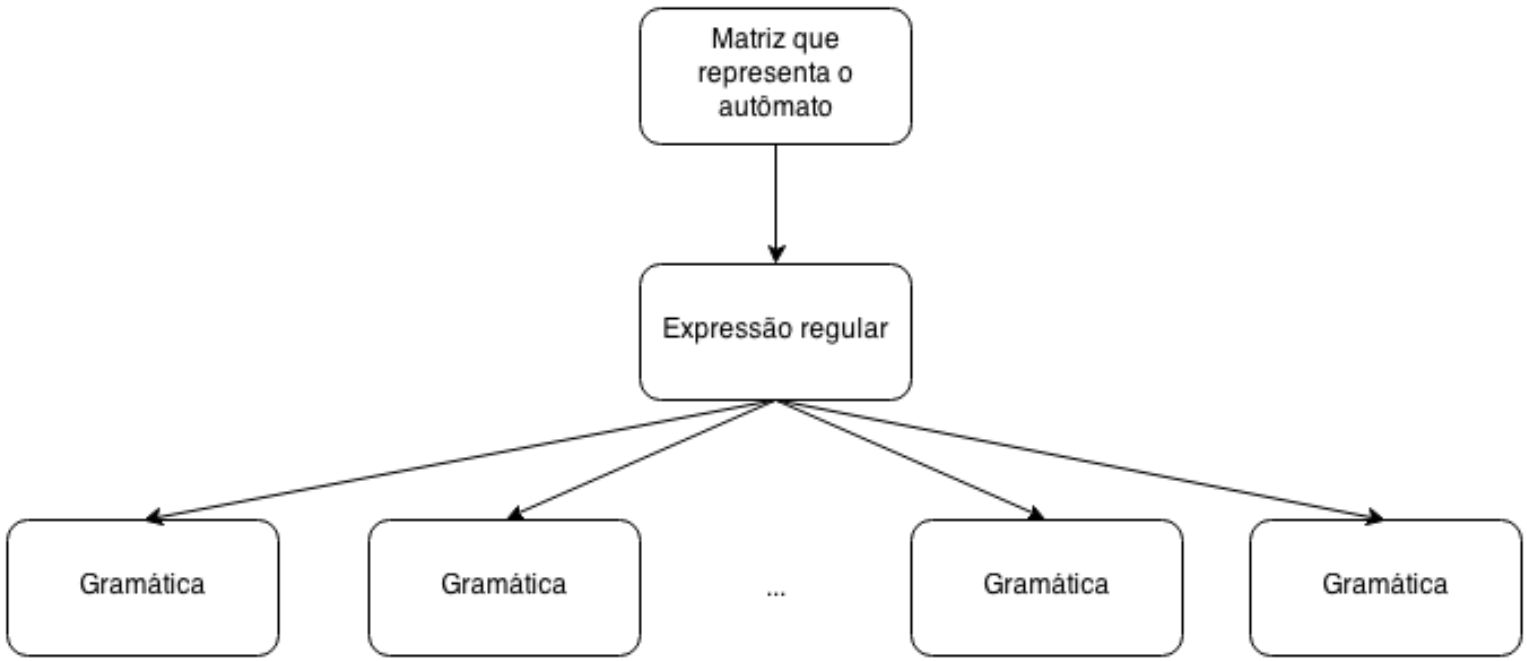
There and back again
Convertendo autômatos para gramáticas

Lucas Virgili

The map


Matriz que representa o autômato
Expressão regular
Gramática
Gramática
...
Gramática
Gramática

De matriz para regex

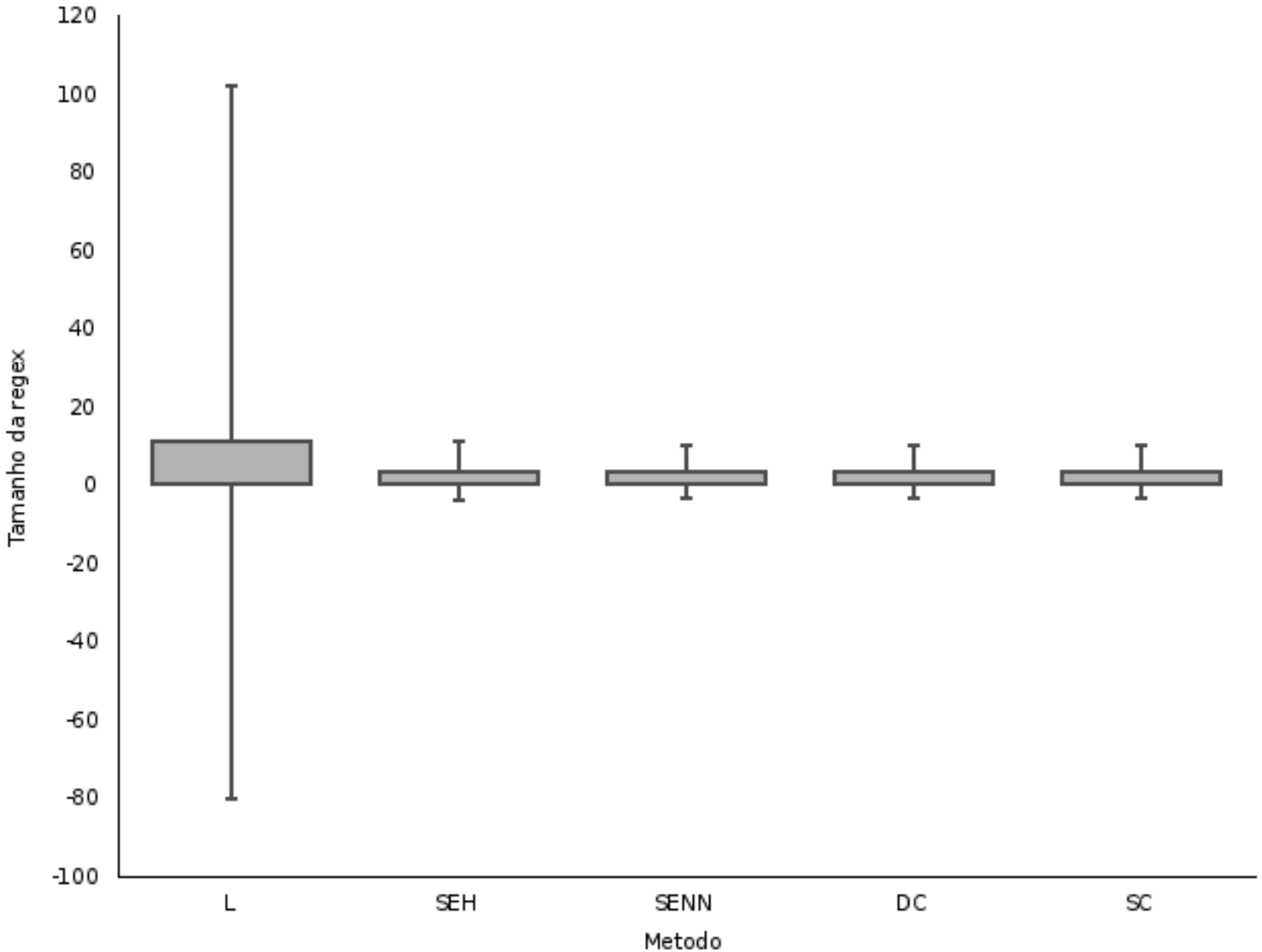
- Implementei o método idiota, doravante chamado de método chamado de método L. (de Lucas)
  - Basicamente uma "reinterpretação" do algoritmo de Floyd-Warshall
- Comecei a implementar o algoritmo de *state removal*
- Enquanto eu procurava por heurísticas para qual estado remover primeiro, encontrei um módulo de python chamado FAdo
- Já tinha implementado as heurísticas!

Comparando os métodos

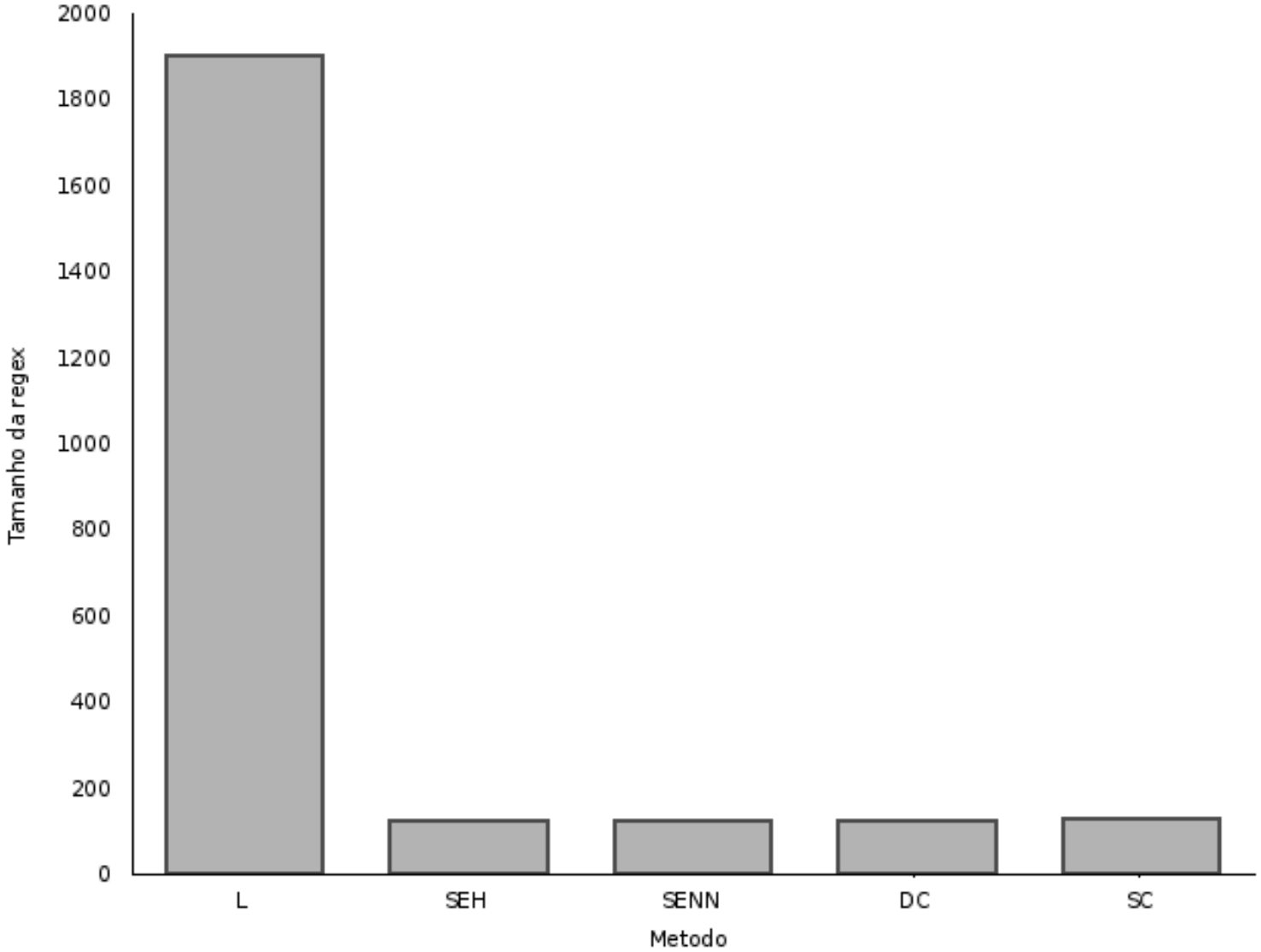
- Usando os métodos implementados no FAdo e o método L em autômatos finitos aleatórios, eu analisei qual o método gerava a menor regex.
- Os testes não puderam ser feitos com autômatos muito grandes (muitos estados) nem com um alfabeto muito extenso, já que meu computador é uma droga e os algoritmos tem complexidades que deixam a desejar.

0 e 1!

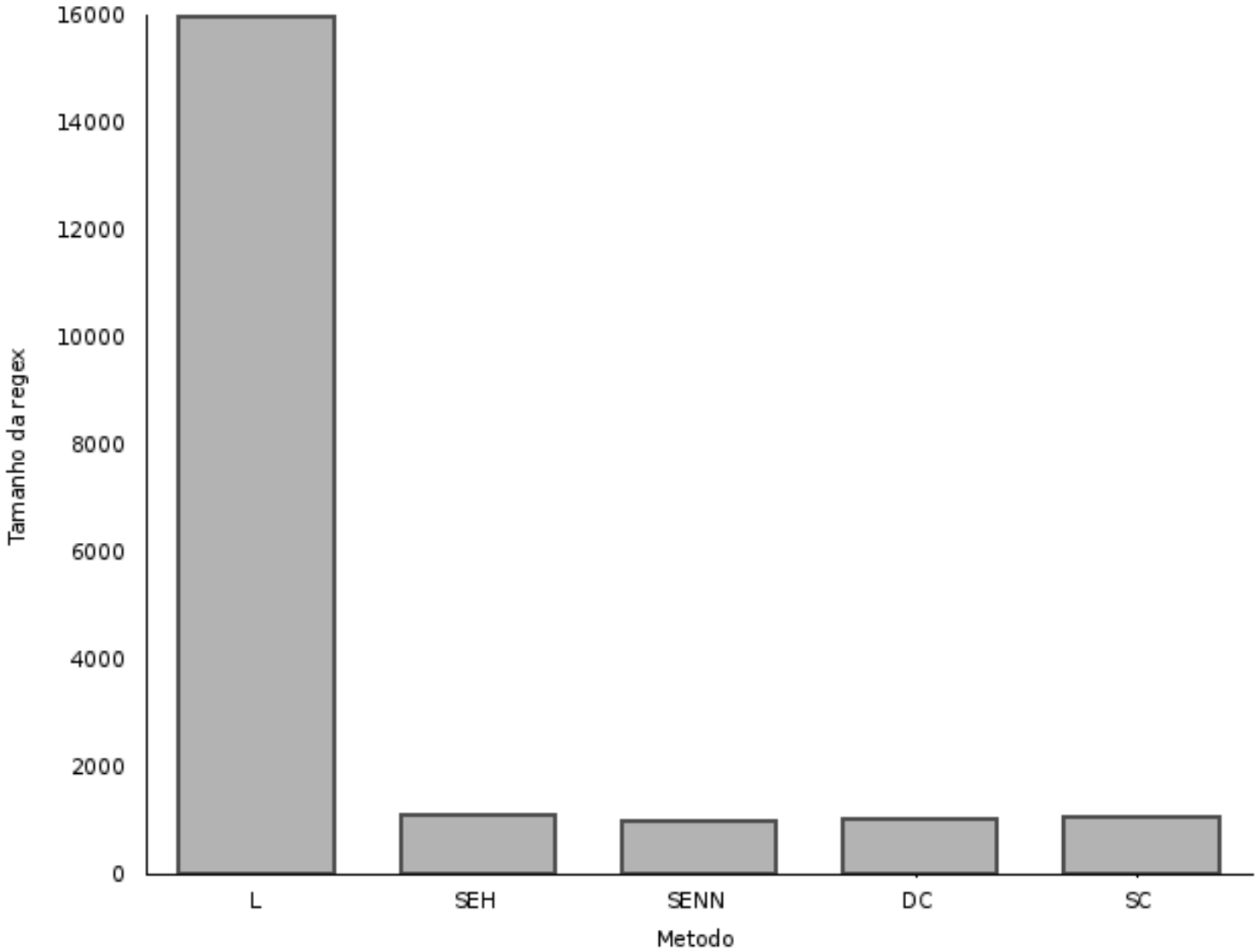
- $\Sigma = \{0, 1\}$ e 2 estados.



$|\Sigma| = 5$ e 6 estados



$|\Sigma| = 10$ e 7 estados



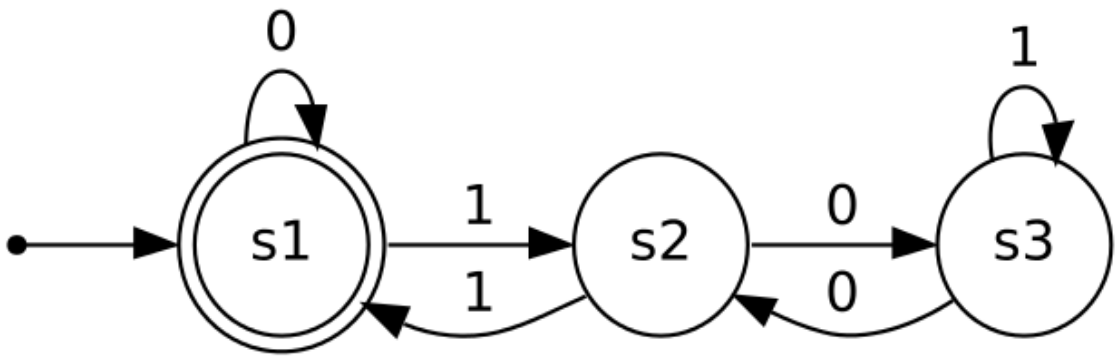
O veredicto

- Usar eliminação de estados é rápido e gera expressões de tamanhos semelhantes aos outros métodos.

De regex à gramática

- Dada a regex, é simples obter uma gramática equivalente: basta nomear cada não terminal e, para cada um deles, formalizar as uniões e os fechos.

A cobaia



A cobaia virou uma regex...

- Sua regex:
  $((0 + (1\ 1)) + (((1\ 0)\ (1 + (0\ 0))^*)\ (0\ 1)))^*$

E a regex virou uma gramática!

- A = {0}
- B = (1 1)
- C = (A B)
- D = {C}
- E = (1 0)
- F = {1}
- G = (0 0)
- H = (F G)
- I = [H]
- J = (E I)
- K = (0 1)
- L = (J K)
- M = (D L)
- N = [M]