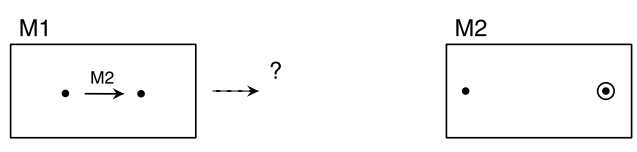
Recursão no autômato de pilha

A solução que nós acabamos de ver é legal, mas ela é um pouco enganadora.

Quer dizer, ela até resolve o problema.

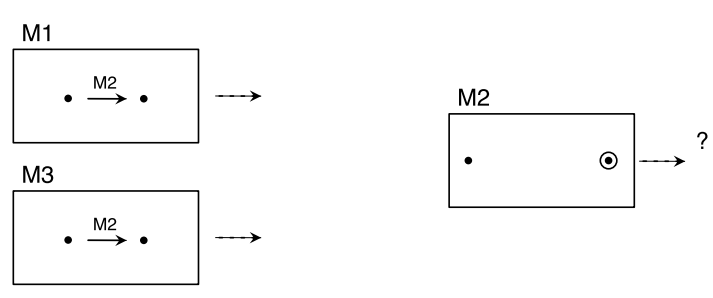
Mas ela não mostra todos os detalhes da solução.

Quer dizer, quando o módulo $\mathsf{M_1}$ faz uma chamada para o módulo $\mathsf{M_2}$



como é que o controle da computação é passado de um módulo para o outro?

E mais importante, se dois módulos $\mathsf{M_1}$ e $\mathsf{M_2}$ fazem chamadas para um terceiro módulo $\mathsf{M_3}$



quando a computação de $\mathsf{M_3}$ termina, como é que nós sabemos para onde nós devemos voltar?

Na prática, quando fazemos chamadas de função em nossos programas, é o sistema operacional quem cuida desses problemas.

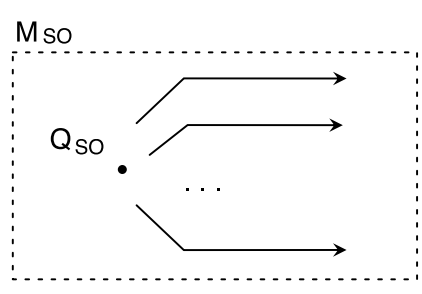
Mas, quando nós estudamos a teoria da computação, todos os detalhes devem estar à mostra para que a gente entenda como a computação realmente acontece.

A boa notícia é que tudo o que é preciso para gerenciar chamadas de função e retornos é uma pilha.

Quer dizer, agora nós vamos ver que tudo o que os autômatos recursivos são capazes de fazer (com a ajuda do sistema operacional) também pode ser feito por um autômato de pilha (sem a ajuda de ninguém).

E a ideia é bem simples: *nós vamos colocar o sistema operacional dentro do autômato de pilha.*

Mais especificamente, o papel do sistema operacional vai ser realizado por um estado $\mathsf{Q_{SO}}$



A ideia é que, momento em que um estado $\mathsf{q_i}$ quer fazer uma chamada para o módulo M

$$\mathsf{q_i} \xrightarrow{\mathtt{M}} \mathsf{q_j}$$

nós colocamos o nome do estado inicial do módulo M na pilha, e fazemos a transição para o "sistema operacional"

$$\mathsf{q_i} \xrightarrow{\mathtt{e[e,Q_M]}} \mathsf{Q_{SO}}$$

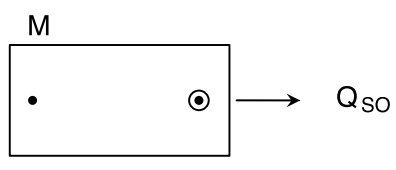
Daí, o sistema operacional remove esse nome do topo da pilha, e realiza a transição para o estado inicial do módulo M

$$\mathsf{Q_{SO}} \xrightarrow{\mathtt{e[Q_M,e]}} \mathsf{Q_M}$$

Certo.

Mas, ainda é preciso voltar para o estado $\mathsf{q_j}$ após o termino da computação do módulo M.

Bom, a ideia é que após o termino da computação de M o controle volta para o sistema operacional



E o sistema operacional sabe para onde voltar em seguida, porque no momento da chamada nós também colocamos essa informação na pilha

$$\mathsf{q_i} \xrightarrow{\mathtt{e[e,Q_M q_j]}} \mathsf{Q_{SO}}$$

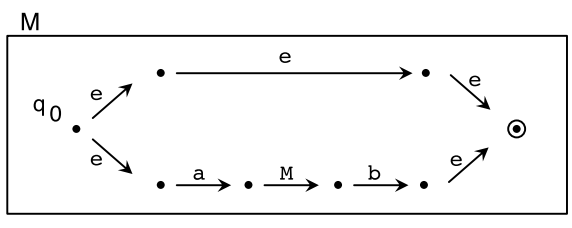
Quer dizer, após o termino da computação de M, o controle volta para o sistema operacional que realiza a seguinte transição

$$\mathsf{Q_{SO}} \xrightarrow{\mathtt{e[q_j,e]}} \mathsf{q_j}$$

Pronto, é só isso!

Vejamos como a coisa funciona em um exemplo concreto.

Abaixo nós temos o autômato recursivo que reconhece a linguagem $\mathtt{a^n b^n}$



As duas modificações que nós fazemos nesse módulo são as seguintes

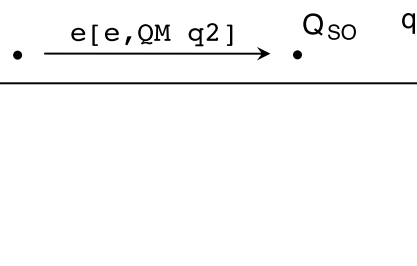
- a chamada recursiva é substituída por uma transição para o estado $\mathsf{Q_{SO}}$

$$\mathsf{q_1} \xrightarrow{\mathtt{e[e,Q_M q_2]}} \mathsf{Q_{SO}}$$

- o estado final do módulo agora também realiza uma transição para o estado $\mathsf{Q_{SO}}$

$$\odot \xrightarrow{\mathtt{e}} \mathsf{Q_{SO}}$$

O estado $\mathsf{Q_{SO}}$ implementa a chamada recursiva e o retorno para o local da chamada da seguinte maneira



Até aqui tudo bem.

Mas, como é que a computação começa e termina?

Bom, a ideia é ter um módulo principal $\mathsf{M_P}$, onde se encontram o estado inicial e o estado final do autômato de pilha, e que realiza a primeira chamada ao módulo M

A figura abaixo apresenta a implementação completa do autômato de pilha