6. Máquinas de Turing e Decidibilidade

Copiar link

Recordemos que a noção mais geral de gramática que estudámos admite regras da forma $\alpha \to \beta$ com muito poucas restrições, em particular α pode conter símbolos terminais e não-terminais (pelo menos um não-terminal).

Por exemplo:

```
S → XZaY
Yc → acY
ZY → WY
aW → Wb
```

Este tipo de gramática tem interesse teórico.

Os autómatos capazes de fazer o seu reconhecimento são conhecidos por Máquinas de Turing.

Numa máquina de Turing é suprimida a pilha presente nos PDAs, estando em vez disso presente uma *fita*: uma estrutura de dados linear, podendo a máquina alterar o conteúdo de qualquer posição (escrever em qualquer célula).
De forma resumida:

- a máquina possui uma *cabeça* (como nos gravadores de cassetes de fita magnética de áudio e vídeo), sendo em cada transição lida a célula da fita que se encontra sob essa cabeça, e escrito um valor nessa mesma célula
- como resultado de cada transição de estado, haverá um efeito que poderá fazer deslocar essa cabeça de leitura e escrita para a esquerda ou a direita.
- considera-se que a fita é “duplamente infinita”, i.e. é possível a cabeça deslocar-se indefinidamente em ambos os sentidos
- ao contrário dos PDAs, em que a palavra lida e a pilha eram estruturas separadas, agora a palavra será usada para inicializar a máquina: na sua configuração inicial a fita conterá esta palavra (um símbolo por célula)
- finalmente, consideraremos que se trata de mecanismos não-determinísticos: para cada estado e valor lido da fita, poderá haver várias transições possíveis da máquina

A execução da máquina termina (‘halts’) se numa determinada configuração não existe nenhuma transição possível. Isto acontece devido à parcialidade da função de transição, que pode não estar definida para um determinado par (estado, símbolo).

Nas máquinas de Turing existe um único critério de aceitação: uma palavra é **aceite** se a execução termina num estado final. É **rejeitada** se a execução termina num estado que não é final. Mas note-se que a execução não pára sempre que se atinge um estado final! Apenas pára se não existe nenhuma transição possível.

Duas diferenças fundamentais em relação aos autómatos anteriores são:

- A execução pode não terminar! Enquanto nos DFAs, NFAs, e PDAs era lido um símbolo da palavra em cada transição, uma vez que não era possível a escrita na palavra, a terminação era garantida. Nas MTs é possível escrever na fita, pelo que a informação nela contida não constitui uma medida que garanta a terminação

- Enquanto num autómato a palavra a aceitar era lida exactamente uma vez, nas MTs pode ser lida apenas parcialmente, e cada símbolo pode ser lido mais do que uma vez. Não existem quaisquer restrições, uma vez que a leitura e a escrita são feitas na mesma estrutura — a palavra a aceitar é apenas usada como inicializarão desta estrutura.

Definição Formal
Uma máquina de Turing é um tuplo $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ em que:

- Q é um conjunto *finito* e *não vazio* de estados
- Σ é o habitual alfabeto (*finito* e *não vazio*) de símbolos lidos pelo autómato
- Γ é também um alfabeto (igualmente *finito* e *não vazio*), dos símbolos que podem ser inseridos na fita. Será sempre $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, S\}$ é uma função de transição
- $q_0 \in Q$ é o estado inicial do autómato
- $B \in \Gamma$ é o símbolo vazio ('blank')
- $F \subseteq Q$ é um conjunto *finito* e *possivelmente vazio* de estados finais

Atente-se no tipo da função de transição:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$$

O conjunto $\{L, R, S\}$ contém três instruções para a cabeça da máquina: mover para a esquerda (L), mover para a direita (R), e não mover (S). Em cada transição será executada uma destas instruções, depois de lido o símbolo sob a cabeça e de escrito um novo símbolo nessa mesma célula.

Relação de Transição

Um ponto prévio: quando dizemos que a fita contém a palavra w, sendo a fita infinita e w uma palavra finita, está implícito que na realidade a fita contém os símbolos $\cdots BBBBBBwBBBBBBB\cdots$, ou seja todas as células à esquerda e à direita de w contém o símbolo especial 'Blank'.

Existem duas formas comuns de representar as configurações de uma MT.

1. como triplos (q, w, i), em que q é um estado, w é a palavra que se encontra na fita (um símbolo por célula), e i é o índice em que se encontra a cabeça da máquina, admitindo-se índices negativos

2. como triplos (a, q, b), em que q é um estado e a, b são palavras. Nesta representação a ideia é que a fita contém a palavra ab, encontrando-se a cabeça na célula que contém o primeiro símbolo de b. Estas configurações podem ser escritas de forma muito simples como aqb (mas note-se que *q não é um símbolo!*)

   Por exemplo
   - $000q\#111$ denota uma configuração em que uma máquina com $\Gamma = \{0, 1, \#\}$ se encontra no estado q, sendo a fita $\cdots \mid 0 \mid 0 \mid 0 \mid \underline{\#} \mid 1 \mid 1 \mid 1 \mid \cdots$ (_ identifica a posição da cabeça)
   - q denota uma configuração em que a máquina se encontra no estado q, estando a fita vazia (contém a palavra ε)

Definimos depois a *relação de transição $\rightarrow$ sobre configurações* da seguinte forma:

- Se $(q', y, S) \in \delta(q, x)$ então $aqxb \rightarrow aq'yb$
- Se $(q', y, R) \in \delta(q, x)$ então $aqxb \rightarrow ayq'b$
- Se $(q', y, L) \in \delta(q, x)$ então $azqxb \rightarrow aq'zyb$

Trata-se de uma relação não-determinística: para cada par (estado, símbolo lido) podem acontecer diferentes transições. Escreveremos $\rightarrow^*$ para designar o fecho reflexivo e transitivo desta relação, e escreveremos $aqb \Rightarrow a'q'xb'$ se $aqb \rightarrow^* a'q'xb'$ e $\delta(q', x) = \emptyset$, i.e. existe uma execução da máquina que termina com a configuração $a'q'xb'$, podendo q' ser ou não um estado final.

Linguagem de uma Máquina de Turing

A linguagem reconhecida por uma MT é definida da forma que seria expectável pelo critério de *final state*:

$$L = \{w \mid w \in \Sigma^* \wedge q_0 w \Rightarrow aqb \text{ com } a, b \in \Gamma^* \text{ e } q \in F\}$$

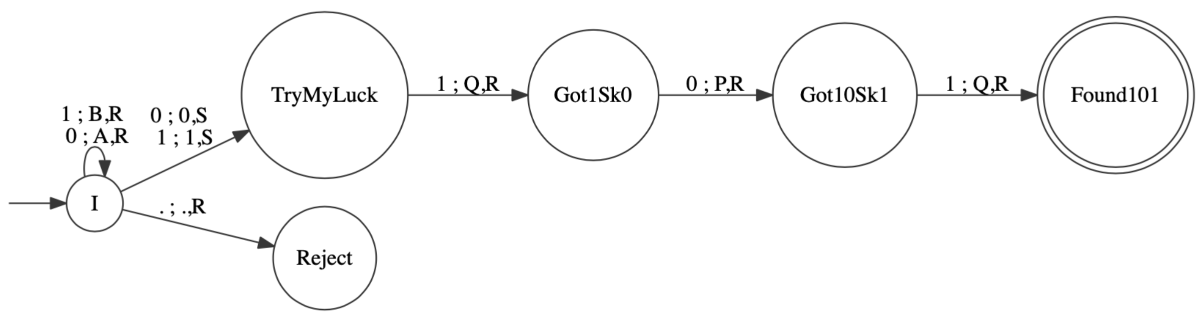
Se além de reconhecer a linguagem L a máquina termina sempre a execução para qualquer $w \in \Sigma^*$, i.e. rejeita todas as palavras $w \notin L$ terminado a execução num estado $q \notin F$, diz-se que a máquina *decide* a linguagem L.

Note-se que uma linguagem L pode ser reconhecida e não decidida por uma máquina, porque a sua execução pode não terminar para algumas palavras não pertencentes a L.

Exemplo

O interesse das MTs é na verdade teórico, e não está propriamente nas linguagens que reconhecem, no sentido tradicional que temos estudado.

Vejamos uma MT que reconhece uma linguagem que é na verdade regular, o que já é suficiente para o nosso propósito neste ponto. Trata-se da linguagem das palavras que contêm como sub-palavra “101”.

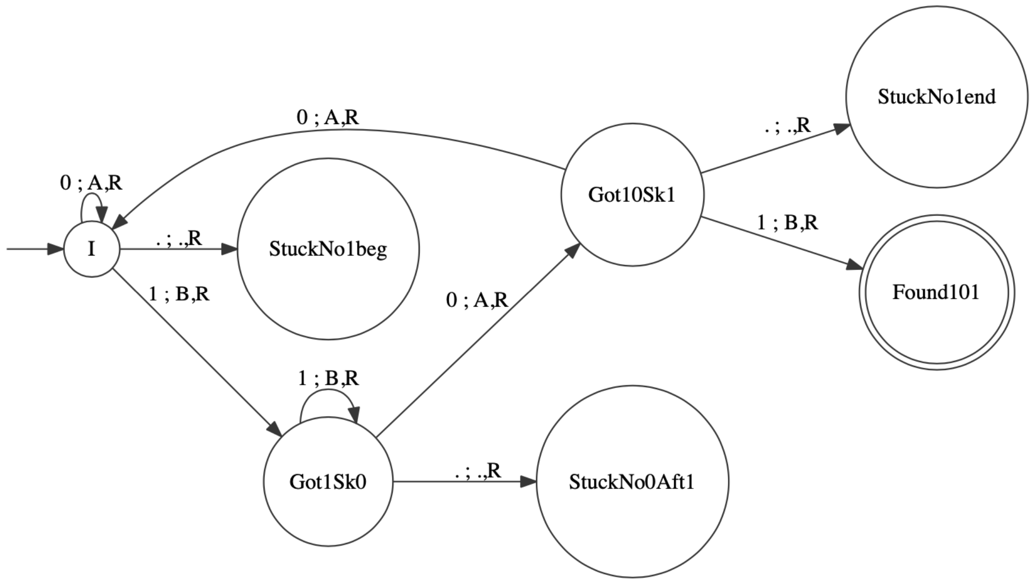


Uma etiqueta como 1 ; B , R denota uma transição em que é lido o símbolo ‘1’, sendo substituído na fita por ‘B’, e avançando a cabeça para a direita. O símbolo ‘.’ corresponde ao que eu cima designámos por ‘B’ (*Blank*).
Note-se que:

- o não-determinismo garante que existem execuções que tentam começar o teste de “101” a partir do estado TryMyLuck, em qualquer posição da fita.
- O anel no estado I vai avançando na fita substiuindo ’0’ por ‘A’ e ’1’ por ‘B’, por forma a que em cada posição da fita o teste só seja iniciado no máximo uma vez.

- Quando termina a palavra, sendo encontrado um *blank '.'*, a máquina transita para um estado Reject no qual não estão definidas quaisquer transições, sendo assim rejeitadas todas as palavras que levam a estas execuções
- No final da execução observa-se um efeito lateral: caso a substring tenha sido encontrada, ela foi marcada na fita com os símbolos "QPQ"

Vejamos uma versão determinística para o mesmo problema. Naturalmente, mas MTs determinísticas são casos particulares de MTs.



Observa-se aqui a necessidade de incluir estados "Stuck" para a a rejeição de palavras que terminam antes de ser encontrada "101".

Problemas de Decisão e Decidibilidade

Em Ciência da Computação um *problema de decisão* é um qualquer problema de resposta SIM ou NÃO. Por outras palavras, qualquer problema que consista, dado um *input* concreto, em decidir se aquele *input* satisfaz ou não uma determinada propriedade.

Naturalmente, para qualquer problema de interesse é desejado que exista um procedimento que o resolva, i.e., que exista uma função que seja mecanizável, que para cada *input* produza a resposta sim/não correcta.

Um problema diz-se *decidível* se existe um procedimento cuja execução:
1. termina sempre
2. e produz a resposta SIM/NÃO correcta

Um problema diz-se semi-decidível (uma noção mais fraca) se existe um procedimento cuja execução:
1. termina sempre, com resultado SIM, para os *inputs* do problema para os quais a resposta ao problema é SIM
2. para os inputs para os quais a resposta ao problema é NÃO,
   - ou termina com resultado NÃO,
   - ou não termina

A noção de problema, abrange, naturalmente, uma variedade enorme de áreas e graus de dificuldade. Vejamos alguns exemplos

- Dado como *input* uma lista de números inteiros: decidir se a lista se encontra ou não ordenada.
  - Pode ser resolvido facilmente por uma travessia recursiva ou iterativa da lista: é decidível e tem pode ser resolvido de forma eficiente

- Em Lógica, dado como *input* uma *fórmula*, decidir se ela é válida. Por exemplo:

  - $p \wedge (p \to q) \to q$ (fórmula *proposicional*)
    é válida, porque é verdadeira para todos os valores das variáveis proposicionais p e q.

O problema de validade preposicional pode ser resolvido com recurso a uma tabela de verdade: é decidível, apesar de ser extremamente difícil de resolver em geral (é um problema NP-completo)

- $(\exists x.P(x)) \to \forall x.P(x)$, com P um *predicado* (fórmula de *primeira ordem*) não é válida, porque a existência de um valor x_0 para o qual $P(x_0)$ é verdade não significa que $P(x)$ seja sempre verdade para qualquer x!
  O problema de validade de primeira ordem é indecidível, mas é semi-decidível.

- Em teoria de linguagens e autómatos (a área em que nos temos situado), dada uma palavra e uma gramática como *inputs*: decidir se a palavra pertence ou não à linguagem especificada pela gramática
  - Pode ser resolvido construindo um autómato que reconheça a linguagem especificada pela gramática.
  - Quanto menos restrita for a gramática, mais sofisticado terá de ser o autómato, e mais difícil será resolver o problema

Uma visão possível é pois esta última, de vermos o problema de reconhecimento de uma linguagem como um caso particular de problema de decisão. Existe no entanto uma outra visão, que explica a proximidade que existe entre as áreas de
- Autómatos e Linguagens Formais
  e de
- Teoria da Computação

É que de facto todo o estudo das noções centrais da Teoria da Computação, nomeadamente a
- **computabilidade** (é ou não possível resolver um problema), e a
- **complexidade** (quão difícil é resolver, sendo possível)

dos problemas de decisão, **pode ser realizado no quadro da teoria das linguagens**, uma vez que qualquer problema de decisão pode ser visto como um problema de reconhecimento de uma linguagem.

Basta para isto, para um qualquer problema de decisão P:
1. Definir um alfabeto Σ tal que todos os inputs de P possam ser codificados como palavras de Σ^*
2. Seja T a função que traduz um input de P numa palavra de Σ^*
3. Considerar a linguagem L_P de todas as palavras $T(x)$ correspondentes a inputs x de P para os quais a resposta ao problema P é SIM
4. Então $T(x) \in L_P$ sse a resposta de P sobre X é SIM, o que significa que resolver P é equivalente a resolver o problema de reconhecimento da linguagem L_P.

À noção de procedimento corresponde, na Teoria das Linguagens, a de máquina de Turing. É fácil constatar que:

- P é decidível sse existe uma máquina de Turing que **decide** a linguagem L_P, i.e. L_P é **recursiva**
- P é semi-decidível sse existe uma máquina de Turing que **aceita** a linguagem L_P, i.e. L_P é **recursivamente enumerável**