Decidibilidade

Decidibilidade

1. Introdução

- No tópico anterior a Máquina de Turing foi apresentada como um modelo de computador de propósito geral.
- A tese de Church-Turing foi utilizada para apresentar uma noção de algoritmo e discutir a universalidade das máquinas de Turing.
- No presente tópico começamos a investigar o poder dos algoritmos (computadores) em solucionar problemas.
- Serão demonstrados alguns problemas que podem ser resolvidos via algoritmos e outros que não podem.
- O objetivo é explorar os limites dos algoritmos (computadores) na resolução de problemas.
- Há duas motivações principais para se estudar quais problemas podem ser resolvidos e quais não podem:

  - Saber que um problema não pode ser resolvido por um algoritmo é útil pois a partir daí podemos tentar simplificá-lo ou alterá-lo para que ele seja resolvido por um algoritmo.
  - Mesmo sabendo que um problema pode ser resolvido, conhecer problemas não resolvíveis pode estimular a imaginação e nos ajudar a ganhar novas perspetivas sobre a computação e técnicas de resolução de problemas.
- *A discussão será apresentada abordando a questão sobre quais linguagens podem ser definidas por um dispositivo computacional (algoritmo). Esta questão está intimamente relacionada ao potencial dos computadores, pois o reconhecimento de uma string por uma linguagem é uma maneira formal de expressar qualquer problema, e a capacidade de resolver um problema é equivalente a estudar o que um computador pode fazer*.
- A palavra *decidível* é um sinônimo para *recursivo*, porém linguagens geralmente são denominadas de recursivas e problemas de decidíveis.

  - Se uma linguagem não é recursiva, então o problema expresso por esta linguagem não é decidível.
  - Na nossa abordagem utilizaremos ambas as terminologias para nos referir a linguagens.
- Vamos iniciar este tópico com um argumento informal.
  - Empregando um algoritmo simples escrito em linguagem de programação C, vamos apresentar um exemplo (problema) específico que não pode ser resolvido usando um computador.
- Posteriormente desenvolveremos uma teoria sobre problemas não-decidíveis.

2. Recapitulação Conceitual

- *Autômato finito*: 5-upla $(K,\sum,\delta,s,F)$; onde K é um conjunto de estados, $\sum$ é o alfabeto, δ é a função de transição, $s \in K$ é o estado inicial, e $F \subseteq K$ é o conjunto de estados aceitos.
- *Reconhecimento*: diz-se que uma máquina M (p. ex. autômato ou MT) *reconhece* uma linguagem A se $A = \{w \mid M \text{ aceita } w\}$.
- *Linguagens regulares*: uma linguagem é dita regular se algum autômato finito a reconhece.
- *Expressões regulares*: são expressões que descrevem linguagens.
- *Gramáticas livres de contexto*: 4-upla $(V,\sum,R,S)$, onde V é um conjunto finito de variáveis, $\sum$ é o conjuto finito de terminais, R é um conjunto finito de regras, e s é a variável início.
- *Linguagens livres de contexto*: coleção de linguagens associadas a gramáticas livres de contexto.

- **Nota**: autômatos finitos, expressões regulares e gramáticas livres de contexto são métodos distintos para descrever linguagens.

Exemplo de Problema Não-Decidível: "Hello, world"

- O problema em particular que vamos discutir aqui é se a primeira coisa que um programa em C imprime é "hello, world".
- Embora sejamos tentados a pensar que basta simular um programa para verificar o que ele faz, há problemas que podem levar uma quantidade infinita de tempo antes de fornecer uma saída.
- Entretanto, provar formalmente que não há um programa capaz de realizar uma determinada tarefa é difícil, e alguns mecanismos formais precisam ser desenvolvidos.
- Um primeiro programa implementado por alunos da linguagem C que usam o livro clássico de Ritchie[1] imprime a sentença "hello, world" na tela do computador.

[1] B. W. Kernighan & D. M. Ritchie, *The C Programming Language*, 1978, Prentice-Hall.

```
Main()
{
printf("hello, world\n");
}
```

- Há outros programas que também imprimem "hello, world" que não são tão óbvios quanto este.
- Por exemplo, considere o programa abaixo que recebe uma entrada n e procura inteiros positivos que sejam solução da seguinte equação:

  $x^n + y^n = z^n$
- Se o algoritmo encontrar uma tripla (x,y,z) que satisfaça a equação acima, então ele imprime "hello, world". Caso contrário ele continua a busca eternamente e nunca imprime "hello, world".
- Se o valor lido de n for 2 por exemplo, $n = 2$, então o programa irá eventualmente encontrar $x = 3$, $y = 4$ e $z = 5$ que satisfaz a equação.

- Para valores inteiros $n > 2$, o programa jamais encontrará uma tripla (x,y,z) que satisfaça $x^n + y^n = z^n$ e, portanto, não imprimirá "hello, world".
- O fato interessante é que até poucos anos atrás não se sabia se este programa iria ou não imprimir "hello, world" para algum inteiro grande n.

```
int exp(int i, n)
/* calcula i elevado a n */
{
    int ans, j;
    ans = 1;
    for (j=1; j<=n; j++) ans *= i;
    return(ans);
}
main ()
{
    int n, total, x, y, z;
    scanf("%d", &n);
    total = 3;
    while (1) {
      for (x=1; x<=total-2; x++)
          for (y=1; y<=total-x-1; y++) {
              z = total - x - y;
              if (exp(x,n)+exp(y,n)==exp(z,n))
                  printf("hello, world\n");
          }
      total++;
    }
}
```

3. Máquinas de Turing: Adendo

- No Tópico 5 uma MT foi formalmente definida como uma quádrupla $M = (K,\sum,\delta,s)$ onde:
  - K é um conjunto de *estados*, incluindo o *estado de parada* (*halt state – h*);
  - $\sum$ é um alfabeto contendo o símbolo vazio #, mas não contendo os símbolos de movimentação L, R e N;
  - δ é uma função de transição de estados δ: $K \times \sum \rightarrow (K \times \sum \times \{L,R,N\})$, onde $\{L,R,N\}$ indica o movimento do cabeçote para a esquerda (L), direita (R), ou nenhum movimento (N).
  - $s \in K$ é o estado inicial.
- Entretanto, é possível também definir uma MT considerando dois outros estados $M = (K,\sum,\delta,s,q_{accept},q_{reject})$ onde:
  - $q_{accept} \in K$ é o estado aceito; e
  - $q_{reject} \in K$ é o estado rejeitado, $q_{accept} \neq q_{reject}$.

- Neste caso, M computa até que ela entre em um dos dois estados q_{accept} ou q_{reject}.
- Uma MT M *aceita* uma entrada w se existe uma seqüência de configurações $C_1,C_2,...,C_k$ tal que:
  - C_1 é a configuração inicial de M,
  - Cada C_i resulta em um C_{i+1}, e
  - C_k é uma configuração cujo estado é q_{accept}.
- A coleção de strings aceitas por M é a *linguagem* de M, $L(M)$.
- Uma linguagem é dita *Turing-reconhecível*, ou *recursivamente enumerável*, se há alguma MT que a reconhece.
  - Uma MT reconhece uma linguagem L se dada uma string w de entrada ela é capaz de verificar se $w \in L$.

- Uma linguagem é dita *Turing-decidível*, ou *recursiva*, se alguma máquina de Turing a decide.
  - Uma MT decide uma linguagem se ela sempre toma uma decisão de *aceitar* ou *rejeitar* esta linguagem.

4. Problemas Decidíveis Envolvendo Linguagens Regulares

- Nesta seção apresentaremos exemplos de linguagens que são decidíveis por algoritmos.
- Inicialmente apresentaremos algoritmos para testar:
  - *se um autômato finito aceita uma string;*
  - *se uma linguagem de um autômato finito é vazia; e*
  - *se dois autômatos são equivalentes*.
- Por conveniência usaremos linguagens para representar vários problemas computacionais, uma vez que já definimos uma terminologia para tratar linguagens.

- Por exemplo, o *problema de aceitação* para autômatos finitos determinísticos (AFD), que testa se um determinado AFD aceita uma dada string pode ser expresso como uma linguagem A_{AFD}.
- Esta linguagem contém a codificação de todos os AFDs junto com as strings que eles aceitam.
- Seja:

  $A_{AFD} = \{\langle B,w\rangle \mid B$ é um AFD que aceita a string de entrada $w\}$.

  O problema de testar se um AFD B aceita uma string w é equivalente a testar se $\langle B,w\rangle$ é um membro da linguagem A_{AFD}.
- De maneira similar, é possível formular outros problemas computacionais como um problema de pertinência a uma linguagem.
- *Mostrar que uma linguagem é decidível é equivalente a mostrar que um problema computacional é decidível, ou seja, é resolvível por um algoritmo.*

- **Teorema 1**:
  - A_{AFD} é uma linguagem decidível.

  Idéia da prova: apresentar uma máquina de Turing que decida A_{AFD}.

  M = "dada uma entrada $\langle B,w \rangle$, onde *B* é um AFD e *w* uma string:

  1. Simule *B* para a entrada *w*."
  2. Se a simulação terminar em um estado aceito (q_{accept}), *aceite*.
  3. Se terminar em um estado não aceito, *rejeite*."

- **Exercício**:

  Prove que A_{AFD} é uma linguagem decidível.

- É possível provar um teorema similar para um autômato finito não-determinístico AFN.

- Seja:

  $A_{AFN} = \{\langle B,w \rangle \mid B \text{ é um AFN que aceita a string de entrada } w\}$.

- **Teorema 2**:
  - A_{AFN} é uma linguagem decidível.

  Prova: vamos apresentar uma máquina de Turing *N* que decida A_{AFN}.
  - É possível projetar *N* para operar como *M*, simulando um AFN ao invés de um AFD. Ao invés disso, façamos *N* usar *M* como uma sub-rotina.
  - Como *M* é projetado para operar com AFDs, *N* primeiramente converte o AFN recebido como entrada em um AFD antes de passá-lo para *M*.

  N = "dada uma entrada $\langle B,w\rangle$, onde *B* é um AFN e *w* uma string:

    1. Converta o AFN *B* em seu equivalente AFD *C* usando um procedimento para conversão.
    2. Rode a MT *M* do Teorema 1 para a entrada $\langle C,w\rangle$.
    3. Se *M aceitar*, aceite; senão *rejeite*."
- Rodar a MT *M* no procedimento acima significa incorporar *M* no projeto de *N* como um sub-procedimento.

- De forma similar, é possível testar se uma *expressão regular* gera uma dada string.
- Seja:

  $A_{ER} = \{\langle R,w\rangle \mid R \text{ é uma expressão regular que gera a string } w\}$.
- **Teorema 3**:
  - A_{ER} é uma linguagem decidível.

  Prova: a seguinte máquina de Turing *P* decide A_{ER}.

  P = "dada uma entrada $\langle R,w\rangle$, onde *R* é uma ER e *w* uma string:

  1. Converta a expressão regular *R* no AFD *A* equivalente usando o procedimento para conversão dado anteriormente.
  2. Rode a MT *M* para a entrada $\langle A,w\rangle$.
  3. Se *M* aceitar, *aceite*; se *M* rejeitar, *rejeite*.
- Os Teoremas 1, 2 e 3 ilustram que, para propósitos de decidibilidade, apresentar uma máquina de Turing com um AFD, um AFN ou uma ER é sempre equivalente, pois a MT é capaz de converter uma forma de codificação por outra.

- Vamos estudar agora outro tipo de problema em relação aos autômatos finitos: o problema de *testar se uma linguagem de um autômato finito é vazia*.
- Nos teoremas anteriores foi necessário testar se um autômato finito aceita uma determinada string. Na próxima prova testaremos se um autômato finito aceita pelo menos uma string.
- Seja:

  $E_{DFA} = \{\langle A \rangle \mid A \text{ é um AFD e } L(A) = \varnothing\}$.

- **Teorema 4**:

  E_{DFA} é uma linguagem decidível.

  Prova: um AFD aceita alguma string se e somente se for possível atingir um estado aceito a partir do estado inicial seguindo as setas do AFD.

  Para testar esta condição é possível projetar uma MT T que usa um algoritmo de marcação.

  T = "dada uma entrada $\langle A \rangle$, onde A é um AFD:

1. Marque o estado inicial A;
2. Repita até que nenhum novo estado seja marcado:
    a. Marque qualquer estado que possui uma transição vindo para ele de qualquer estado que já está marcado.
3. Se nenhum estado aceito está marcado, *aceite*; senão, *rejeite*.

- Por último vamos verificar a decidibilidade de AFDs equivalentes.
- O próximo teorema mostra que testar se dois AFDs reconhecem a mesma linguagem é um problema decidível.
- Seja:

  $EQ_{AFD} = \{\langle A,B\rangle \mid A \text{ e } B \text{ são AFDs e } L(A) = L(B)\}$.
- **Teorema 5**:

  EQ_{AFD} é uma linguagem decidível.

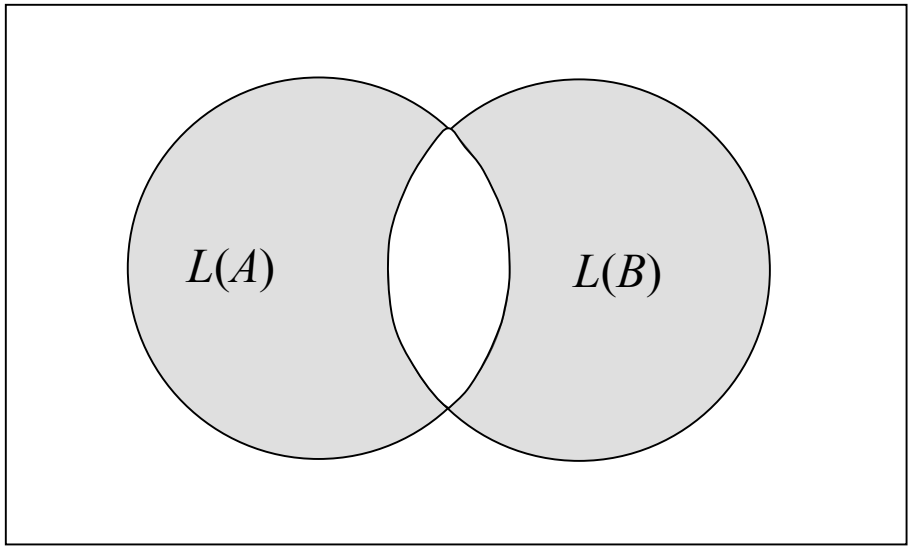
  Prova: Para provar este teorema usaremos o Teorema 4.

Construa um novo AFD C a partir de A e B, onde C aceita apenas aquelas strings que são aceitas ou por A ou por B, mas não por ambos. Assim, se A e B reconhecem a mesma linguagem, C não aceitará nada.

A linguagem de C é:

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

Esta expressão é conhecida como *diferença simétrica* de $L(A)$ e $L(B)$.



- A diferença simétrica é útil aqui, pois $L(C) = \varnothing$ se e somente se $L(A) = L(B)$.
- É possível construir C a partir de A e B com as construções para provar a classe de linguagens regulares fechadas sob o complemento, união e interseção.
    - Estas construções são algoritmos que podem ser executados por uma MT.
    - Uma vez que C foi construído é possível usar o Teorema 4 para testar se $L(C)$ é vazia. Se for, então $L(A) = L(B)$.

F = "dada uma entrada $\langle A,B \rangle$, onde A e B são AFDs:

1. Construa um AFD C como descrito;
2. Rode a MT T do Teorema 4 para a entrada $\langle C \rangle$;
3. Se T aceitar, *aceite*; senão, *rejeite*."

5. Problemas Decidíveis Envolvendo Linguagens Livres de Contexto

- Nesta seção vamos descrever algoritmos para testar se uma gramática livre de contexto (GLC) gera uma string particular e se uma linguagem de uma GLC é vazia.
- Seja:

  $\mathbf{A}_{\text{GLC}} = \{\langle G,w\rangle \mid G \text{ é uma GLC que gera } w\}$.
- **Teorema 6**:

  $\mathbf{A}_{\text{GLC}}$ é uma linguagem decidível.

  Idéia da Prova: Para a GLC G e a string w queremos testar se G gera w. Uma idéia é usar G para percorrer todas as derivações para determinar se alguma delas é uma derivação de w. Esta idéia não funciona pois infinitas derivações podem ter que ser testadas. Se G não gerar w, então este algoritmo nunca irá parar. Esta idéia faz com que a MT seja um reconhecedor, mas não um decisor, da linguagem $\mathbf{A}_{\text{GLC}}$.

Para tornar esta MT um decisor é preciso garantir que o algoritmo irá testar apenas uma quantidade finita de derivações. Já vimos que se G estiver na forma normal de Chomsky, qualquer derivação de w possui $2n - 1$ passos, onde n é o comprimento de w. Neste caso, checar somente derivações com $2n - 1$ passos para determinar se G gera w seria suficiente. Somente uma quantidade finita destas derivações existe. É possível converter G para a forma normal de Chomsky.

Prova: A MT S para $\mathbf{A}_{\text{GLC}}$ é:

S = "dada uma entrada $\langle G,w \rangle$, G é uma GLC e w é uma string:

- Converta G em sua gramática equivalente na forma normal de Chomsky;
- Liste todas as derivações com $2n - 1$ passos, onde n é o comprimento de w; exceto se $n = 0$, caso no qual todas as derivações de um passo são listadas;
- Se alguma destas derivações gerar w, *aceite*; senão, *rejeite*.

- Vamos considerar agora o problema de testar se uma linguagem de uma GLC está vazia.

- Como feito para um AFD, é possível mostrar que o problema de testar se uma GLC gera uma string qualquer é decidível.
- Seja:

  $\mathbf{E}_{\text{GLC}} = \{\langle G \rangle \mid G \text{ é uma GLC e } L(G) = \varnothing\}$.
- **Teorema 7**:

  $\mathbf{E}_{\text{GLC}}$ é uma linguagem decidível.

  Idéia da Prova: Para testar se uma linguagem de uma gramática está vazia, é preciso testar se a variável inicial pode gerar uma string de terminais. O algoritmo faz isso resolvendo um problema ainda mais genérico. Ele determina *para cada variável* se aquela variável é capaz de gerar uma string de terminais. Quando o algoritmo determinou que uma variável pode gerar alguma string de terminais, o algoritmo mantém esta informação marcando aquela variável.

  Primeiro o algoritmo marca todos os símbolos terminais da gramática. Depois ele lê todas as regras da gramática. Se ele encontrar uma regra que permita alguma

variável ser substituída por alguma string de símbolos já marcada, o algoritmo sabe que esta variável também pode ser marcada. O algoritmo procede desta forma até que ele não possa mais marcar nenhuma variável adicional. A MT *R* implementa este algoritmo.

Prova:

R = "dada uma entrada $\langle G \rangle$, onde *G* é uma GLC:

- Marque todos os símbolos terminais de *G*;
- Repita até que nenhuma nova variável possa ser marcada:
  - Marque qualquer variável *A* onde *G* tenha uma regra $A \rightarrow U_1U_2...U_k$ e cada símbolo $U_1,...,U_k$ já tenha sido marcado.
- Se o símbolo inicial não estiver marcado, *aceite*; senão, *rejeite*.

- Finalmente, vamos considerar o caso de testar se duas gramáticas livres de contexto geram a mesma linguagem.
- Seja:

$EQ_{GLC} = \{\langle G,H\rangle \mid G \text{ e } H \text{ são GLCs e } L(G) = L(H)\}$.

- **Teorema 8**:

  Toda linguagem livre de contexto (LLC) é decidível.

  Prova: Seja G uma GLC para A e projete uma MT M_G que decide A. Construa uma cópia de G em M_G. Funciona como a seguir:

  M_G = "para a entrada w:

  - Rode a MT S para a entrada $\langle G,w\rangle$
  - Se esta MT aceitar, *aceite*; senão, *rejeite*."

6. O Problema da Parada (The Halting Problem)

- Os computadores possuem uma capacidade de resolução de problemas tão grande que as vezes podemos imaginar que qualquer problema pode ser resolvido por um computador.
- Entretanto, através de um exemplo importante é possível mostrar que os computadores são limitados de uma maneira fundamental.

- Nesta seção vamos discutir um dos teoremas mais importantes filosoficamente em teoria da computação:
    - *Existe um problema específico que não pode ser resolvido por um algoritmo.*
- Em um tipo de problema não-resolvível, tem-se um programa computacional e uma especificação precisa do que este programa deve fazer (p. ex. ordenar uma lista de números).
    - É preciso verificar se este programa se comporta como especificado, ou seja, se está correto.
- Como tanto o programa quanto a especificação são objetos matemáticos precisos, é intuitivo que o processo de verificação possa ser automatizado dadas o programa e a especificação como entradas para um computador.
    - *Entretanto, o problema genérico de verificação se um software se comporta como desejado não é resolvível por um computador.*

- Um problema importante que demonstra a não-decidibilidade de uma linguagem específica é o problema de testar se uma máquina de Turing aceita uma dada string de entrada, ou seja, se dada uma string de entrada a MT irá parar.
  - Este problema é conhecido como *problema da parada*.
- Seja:

  $A_{TM} = \{\langle M,w\rangle \mid M \text{ é uma MT que aceita } w\}$.
- **Teorema 9**:

  A_{TM} não é decidível.
- Antes de provar o Teorema 9, observe que A_{TM} é Turing-reconhecível.
  - Este teorema mostra, portanto, que reconhecedores são mais poderosos que decisores.
  - Exigir que a MT pare para todas as entradas restringe os tipos de linguagens que ela pode reconhecer.
- A seguinte MT reconhece A_{TM}:

Dada a entrada $\langle M,w \rangle$, onde M é uma MT e w é uma string

1. Simule M para a entrada w.
2. Se M entrar em um estado aceito, *aceite*; se M entrar em um estado rejeitado, *rejeite*.

- Note que se o algoritmo tivesse alguma forma de determinar que M não iria parar dado w, ela poderia *rejeitar*.
    - *Nenhum algoritmo pode determinar se uma máquina irá parar de executá-lo*!

Bibliografia

J. E. Hopcroft, R. Motwani & J. D. Ullman (2001), *Introduction to Automata Theory, Languages, and Computation*, 2nd ed, Addison Wesley.

M. Sipser (1997), *Introduction to the Theory of Computation*, PWS Publishing Company.

H. R. Lewis & C. H. Papadimitriou (1981), *Elements of the Theory of Computation*, Prentice Hall Inc.