[Aula 21] Máquina de Turing – Computabilidade

Prof. João F. Mari
joaof.mari@ufv.br

BIBLIOGRAFIA

- MENEZES, P. B. **Linguagens formais e autômatos**, 6. ed., Bookman, 2011.
  - Capítulo 8.
  - + Slides disponibilizados pelo autor do livro.

- HOPCROFT J. E.; MOTWANI, R. ; ULLMAN, J. D. **Introdução a teoria dos autômatos, linguagens e computação**, 1. Ed., Campus, 2002.

- Nelma Moreira. **Computabilidade: uma introdução**, Departamento de Ciência de Computadores - Faculdade de Ciências, Universidade do Porto, 1996 (Revisão em 2003).
  - Disponível em: http://www.dcc.fc.up.pt/~nam/publica/compdec.pdf

Computabilidade

- O que significa ser computável?
  - Existir um método algorítmico de resolver,
  - Isto é, uma sequência finita de instruções que possa ser efetuada mecanicamente.

  - David Hilbert no início do sec. XX, pretendia-se reduzir a Matemática a manipulação pura de símbolos, e encontrar um algoritmo que determinasse a veracidade ou a falsidade de qualquer proposição matemática.

  - Foram propostos vários formalismos que se demonstrou serem equivalentes (no sentido de aceitarem as mesmas linguagens):
    - Máquinas de Turing (Alan Turing, 1936)
    - Funções recursivas parciais (Kurt Gödel, 1931)
    - λ-Calculus (Alonso Church, 1933)
    - Logica combinatoria (Haskell Curry, 1929)
    - Gramáticas não restritas ou Tipo 0 (Noam Chomsky, 1956)

Máquinas Universais

- Todos os formalismos referidos são suficientemente poderosos para aceitarem a si próprios! Isto é,
  - Tem programas que aceitam como dados codificações de programas.
  - Existem máquinas de Turing que aceitam como dados palavras que são a descrição de máquinas de Turing.
  - Podemos escrever em C um programa que interprete programas em C.

- Note que a noção de Universalidade esta na base da noção de **programa armazenado em memória** e, portanto, na de software.

- Por exemplo (e infelizmente), pode-se demonstrar que não existem algoritmos que dado um programa em C determinem o seu resultado ou se ele para. → → → →

Tese de Church-Turing

- Embora não demonstrado, um problema **computável** (tem um método algorítmico para o resolver) se e somente se existe uma Máquina de Turing que o resolve.

- As razões pelas quais a quase totalidade dos pesquisadores pensa que a Tese de Church é válida são várias.
  1. As máquinas de Turing são tão gerais que parecem poder simular qualquer possível computação.
  2. Nunca ninguém descobriu um modelo “algorítmico" mais geral que as maquinas de Turing.
  3. Vários pesquisadores, ao estudar modelos de computação suficientemente gerais, têm sempre chegado a “algo" que nunca é mais geral do que as MT, sendo apenas equivalentes.

- Um modelo de computação diz-se universal se todos os problemas efetivamente computável podem ser resolvido utilizando esse modelo.

Problemas não computáveis - indecidíveis

- Existência de problemas não computáveis (indecidíveis)
  - Um problema corresponde a uma linguagem em um alfabeto .
  - Resolve-lo é determinar se uma palavra pertence a essa linguagem.

- O número de linguagens sobre um alfabeto não é numerável:

- Uma máquina (programa) que o resolva também pode ser visto como uma palavra num dado alfabeto.
  - Mas então só existe um numerável de máquinas...

- **[CONCLUSÃO]** Existem mais problemas que programas...

- Em que:
  - Têm que existir problemas indecidíveis;
  - Pode ser difícil demonstrar que um dado problema é indecidível.

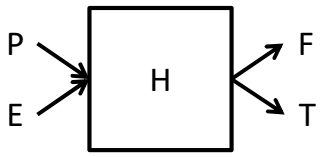
O Problema da Parada

Um exemplo clássico de problema indecidível.

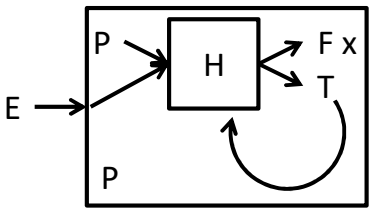
- Escreva um programa H que receber um outro programa P juntamente com sua entrada E.
  - H para e imprime "sim" se P **para** ao receber E.
  - H para e imprime "não" se P **entra em *loop*** ao receber E.
- Alguma estratégia para acompanhar o fluxo de execução de P.
- Para que H emita a resposta "sim" ou "não", é necessário que esta estratégia forneça uma conclusão em tempo finito.
- Porém, se o programa P entrar em *loop, a* estratégia reproduzirá o *loop, e a resposta "não" nunca será emitida.*

O Problema da Parada

- Um algoritmo H, que toma como entrada um procedimento P e sua entrada E.
  - H retorna **true** se P termina para a entrada E.
  - H retorna **false** se P não termina com a entrada E.



- O algoritmo H não existe. A prova é por **contradição**.
  - Suponha que H exista: `boolean H(P, E) { ... }`
- Então eu posso escrever a minha função P como:
  - P NÃO termina com E. P termina com E.
  - `void P(int E) {`
  - `  while H(P, E) { }`
  - `}`
  - TRUE: P termina com E. FALSE: P NÃO termina com E.



- Pergunta: P termina com a entrada E?
  - **SIM**: P termina sua execução quando recebe a entrada E.
    - Então H(P, x) é **true** e P não termina com a entrada E***. Contradição!!!***
  - **NÃO:** P não termina sua execução com a entrada E.
    - Então H(P,x) é **false** e P termina a sua execução com a entrada E***. Contradição!!!***

MT como calculadoras de funções parciais

- Números inteiros
  - Representação binária:
    - Sistema posicional (aula anterior!).
  - Representação Unária
    - O número inteiro i >= 0 é representado pela cadeia 0^i.
    - Se a função possui mais de dois ou mais argumento, separar por 1's.
    - Ex: 000 → f(3); 001000 → f(2,3); 00010100 → f(3,1,2); 00110→f(2,0,1)
- MT: Soma de dois números naturais: f(a,b) = a + b
  - Fita (inicio): $0_a10_b\beta\ldots$; Fita (final): $0_a0_b\beta\ldots$;
- MT: Multiplica um número natural por 2: f(a) = 2*a
  - Fita (inicio): $0_a\beta\ldots$; Fita (final): $0_{a+a}\beta\ldots$;
- MT: Subtração própria: f(a,b) = a – b, se a >= b; 0, se a < b;
  - Fita (inicio): $0_a10_b\beta\ldots$; Fita (final): $0_{a-b}\beta\ldots$ (a>=b); $\beta\ldots$ (a<b) ;

Máquina de Turing Universal

- Uma máquina de Turing pode simular outras máquinas de Turing adequadamente codificadas:
- Considere a linguagem definida pelos pares (MT, x):
  - MT é uma máquina de Turing codificada em binário e alfabeto de entrada {0, 1}
  - $x \in \{0, 1\}^*$
  - MT aceita x
- Máquinas que aceitam essas linguagens são denominadas Máquinas de Turing Universais MT_U:
  - ACEITA(MTU) = { (MT, x) | $x \in$ ACEITA(MT) }

Enumeração das palavras {0, 1}*

- Considere a bijeção entre {0, 1}* e N
  - {0, 1}* f→ **N**
  - ε 1
  - 0 2
  - 1 3
  - 00 4
  - 01 5
  - 10 6
  - 11 7
  - 000 8
  - ... ...

Codificação de Máquinas de Turing

- Podemos codificar qualquer MT com alfabeto {0,1} em palavra de {0,1}*.
  - Para isso, considere a definição alternativa da MT M = (S; Σ; Γ; δ ; s_0; •; F):
    - S : Conjunto finito de estados;
    - Σ : Alfabeto de entrada;
    - Γ : Σ ∪ •
    - s_0 : Estado inicial;
    - • : branco;
    - F : Subconjunto de estados finais;
  - Representamos os estados, símbolos lidos da fita e direção por valores em unário:
    - Estados, q: $q_0 \rightarrow 0$, $q_1 \rightarrow 00$, $q_2 \rightarrow 000$, $q_k \rightarrow 0^k$
    - Símbolos lidos (ou escritos na fita), X: 0 → 0, 1 → 00, • → 000
    - Direção da leitura, D: D1 (esq) → 0, D2 (dir) → 00.
  - A transição $\delta(q_i, X_j) = (q_k, X_l, D_m)$
    - $0^i10^j10^k10^l10^m$
  - Os códigos das transições são separadas por 11:
    - $C_1 11 C_2 11 C_3 11 C_n$
  - A máquina de Turing é separada da palavra por 111

[EX] Codificação de MT

- MT = ($\{s_1, s_2, s_3\}$, $\{0, 1\}$, $\{0, 1, \bullet\}$, δ, s_1, $\bullet$, $\{s_2\}$)
  - $\delta(s_1,1) = (s_3, 0, d)$
  - $\delta(s_3,0) = (s_1, 1, d)$
  - $\delta(s_3,1) = (s_2, 0, d)$
  - $\delta(s_3,1) = (s_3, 1, d)$

Linguagem de diagonalização

- Deve-se construir uma tabela N x N -> {0,1}:
  - Cada célula (i,j) tem valor 1 se MT_i aceita x_j e 0 caso contrário.

$i \backslash j$	1	2	3	4	...
1	0	1	1	0	...
2	1	1	0	0	...
3	0	0	1	1	...
4	0	1	0	1	...
.	.	.	.	.	.
.	.	.	.	.	...

- A linguagem de diagonalização LD é constituída pelas palavras x tal que:
  - MT codificada como x não aceita x;
  - L_D é obtida pelo complemento da diagonal da matriz N x N.
    - Como L_D é diferente em pelo menos uma coluna de todas as linhas da tabela:
      - Não existe MT que aceita L_D.
      - Então L_D não é recursivamente enumerável (não é computável).

BIBLIOGRAFIA AUXILIAR

- O Problema da Parada: Alan Turing, de Leibniz a Gödel - Centenário de Alan Turing – Unicamp
  - https://www.youtube.com/watch?v=593mK9I2P6Q
- Teoria da Computação: uma abordagem prática - Centenário de Alan Turing - Unicamp
  - https://www.youtube.com/watch?v=mNlogpMQnG8
- Palestra Especial: A Vida e o Legado de Alan Turing para a Ciência
  - https://www.youtube.com/watch?v=QmXnc2Ljid8

[FIM]

- FIM:
  - **[AULA 21]** Máquina de Turing – Computabilidade
- **FIM DA DISCIPLINA!**