

Paradigma procedimental



Prof. Dr. Rafael Giusti
rgiusti@icomp.ufam.edu.br

Agenda

- » Máquinas de Turing
- » Três linguagens procedimentais
- » Fluxo: avaliação de expressões
- » Fluxo: sequência e seleção
- » Fluxo: repetições

Modelos matemáticos para algoritmos

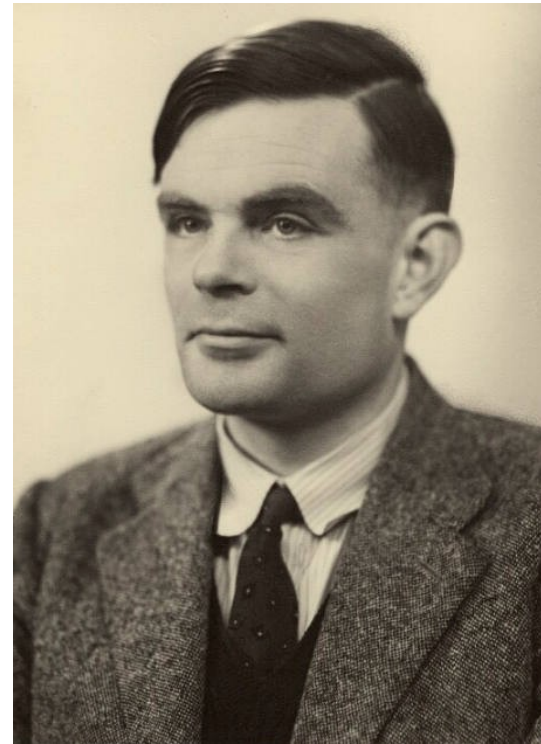
- » Na primeira metade do século XX, matemáticos formalizavam os conceitos de
 - » Algoritmos
 - ~ Como expressar formalmente uma solução para um problema
 - » Computabilidade
 - ~ Quais são os problemas que podem ser resolvidos por meio de algoritmos

Modelos matemáticos para algoritmos

- » Quase simultaneamente, dois modelos equivalentes foram propostos



Cálculo lambda, proposto por Alonzo Church ^[1]



Máquina de Turing, proposto por Alan Turing ^[2]

[1] https://en.wikipedia.org/wiki/File:Alonzo_Church.jpg
[2] <https://en.wikipedia.org/wiki/File:Alan-Turing.jpg>

Máquina de Turing

- » Modelo matemático proposto por Alan Turing por volta de 1936
 - » Aproximadamente seis anos antes do Colossus
- » Objetivo
 - » Formalizar o conceito de **algoritmo**
 - » Solucionar o desafio do "problema de decisão" (*Entscheidungsproblem*) que havia sido proposto por David Hilbert
 - ~ Definir os limites da computabilidade

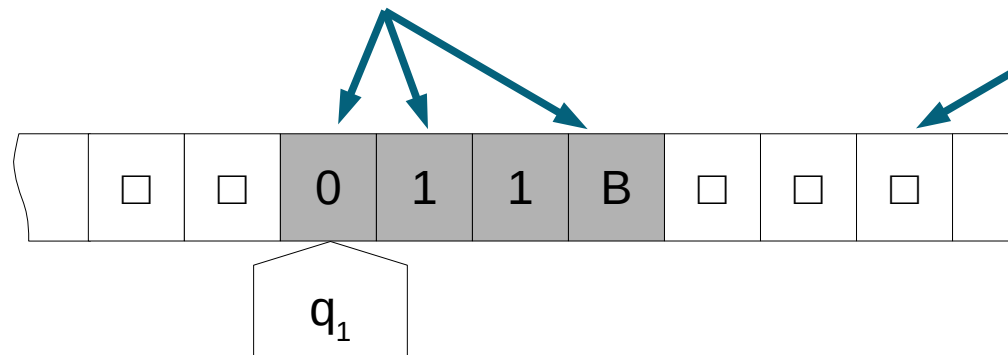
Máquinas de Turing

- » As máquinas de Turing são máquinas de estados que possuem
 - » Uma **fita** de tamanho infinito
 - » Uma **cabeça de leitura e escrita**
 - » Um conjunto finito de **estados internos**
 - ~ Inicial, de aceitação e de rejeição
 - » Um conjunto finito de **regras de transição**

Máquinas de Turing

Os símbolos 1, B e 0 fazem parte do alfabeto da fita

□ é o símbolo “branco”



O polígono indica a posição da cabeça de leitura e escrita

q_1 é o estado interno atual da máquina

Para uma definição formal e descrição detalhada, consulte Lewis e Papadimitriou, “Elementos de Teoria da Computação”

Transições

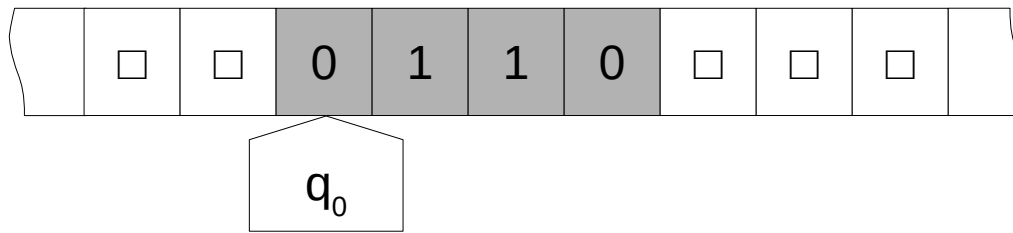
- » Dependendo do símbolo na fita e do estado interno da máquina de Turing, uma decisão diferente pode ser tomada
- » A ação da máquina é especificada por uma **regras de transição**

$$\delta(q_0, X_1) = (q_1, X_2, M)$$

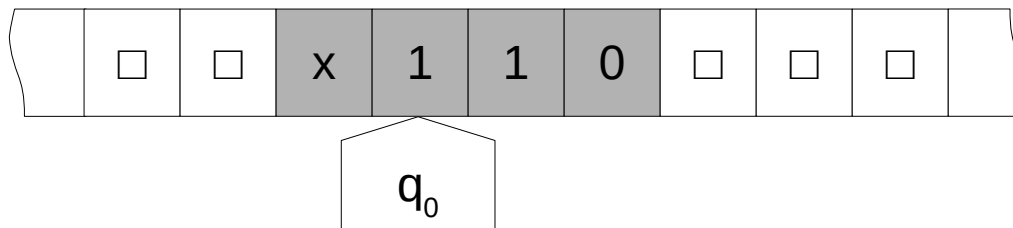
Transições

» Exemplo de transição

- » Troca um símbolo por outro e move a cabeça para a direita



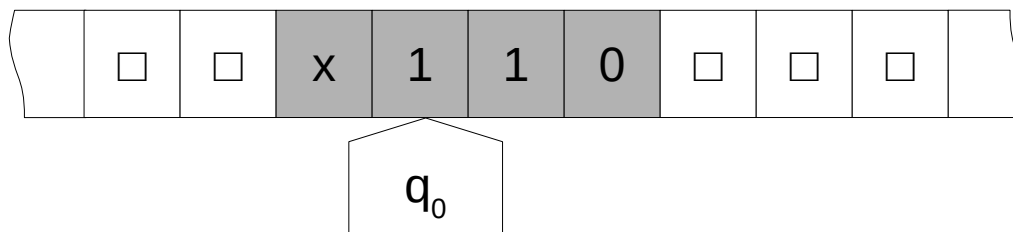
$$\delta(q_0, 0) = (q_0, x, R)$$



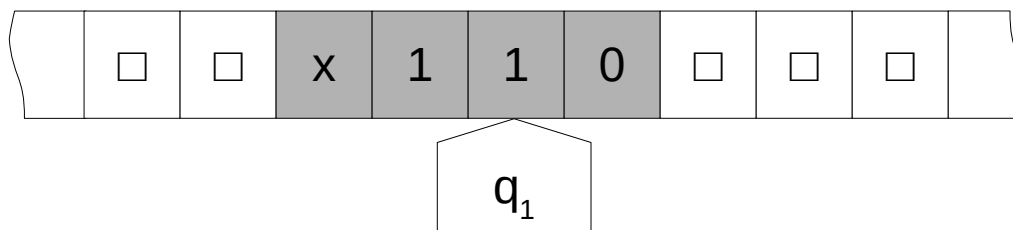
Transições

» Exemplo de transição

- » Mantém o mesmo símbolo, muda de estado e move a cabeça para a direita



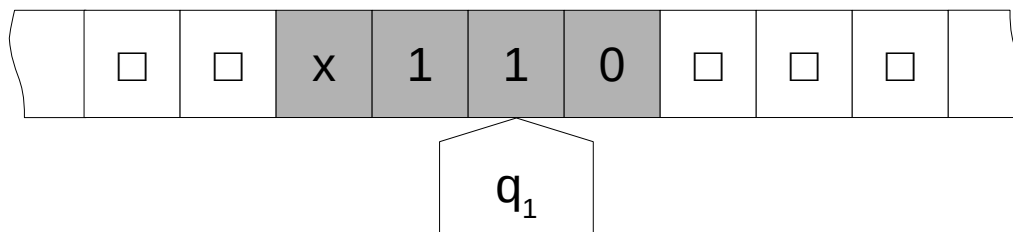
$$\delta(q_0, 1) = (q_1, 1, R)$$



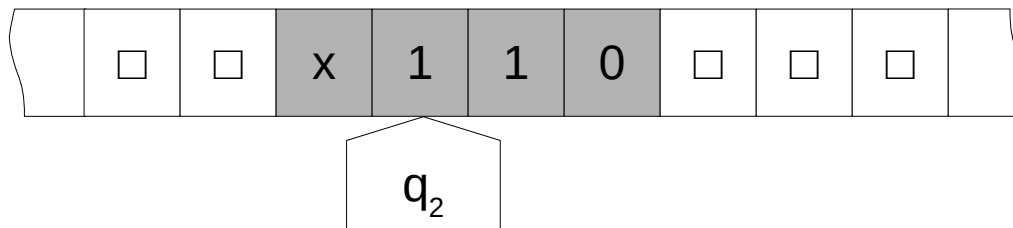
Transições

» Exemplo de transição

- » Mantém o mesmo símbolo, muda de estado e move a cabeça para a esquerda



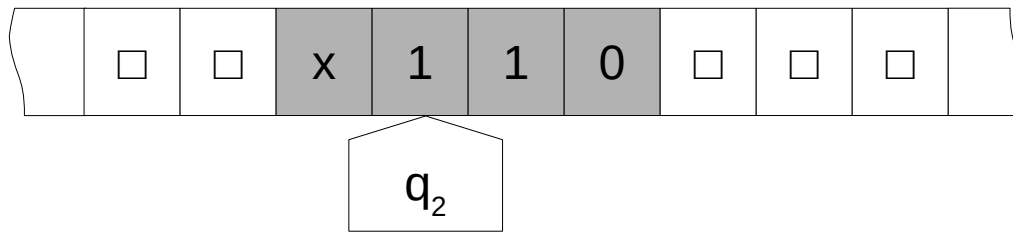
$$\delta(q_1, 1) = (q_2, 1, L)$$



Transições

» Exemplo de transição

- » Se não houver nenhuma transição disponível, rejeita a entrada

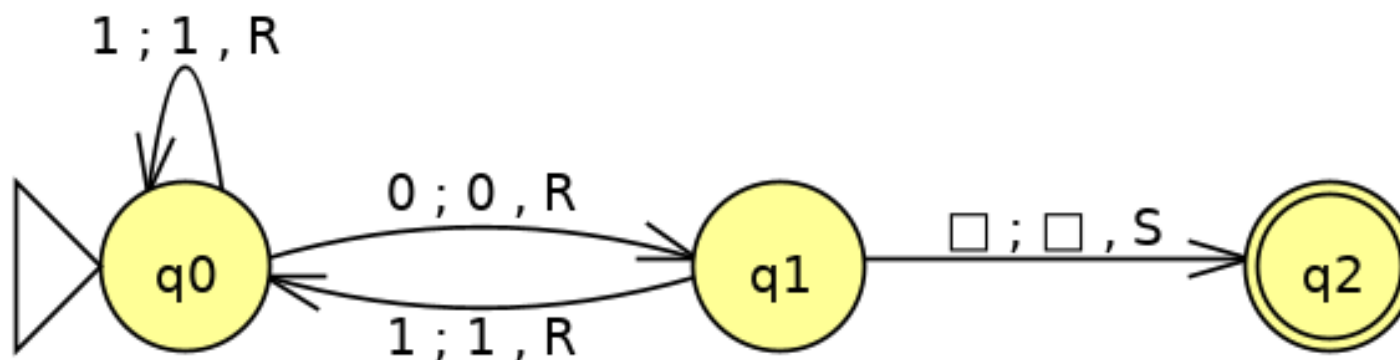


$$\delta(q_2, 0) = (q_3, 1, R)$$

$$\delta(q_2, x) = (q_1, x, R)$$

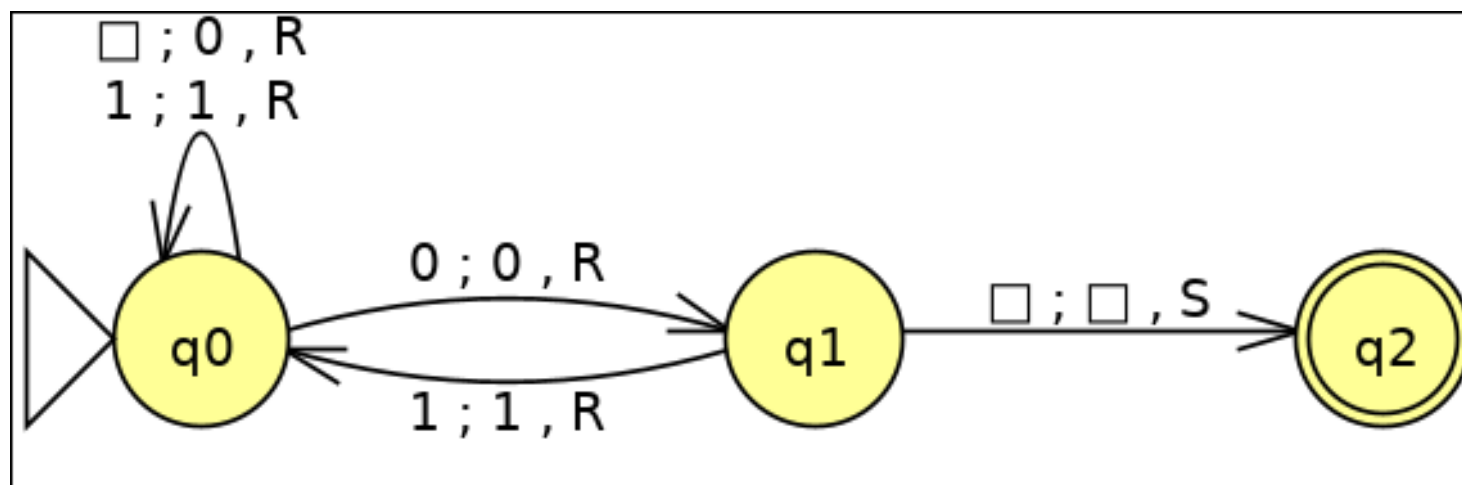
Exemplo de MT

- » Uma máquina de Turing para verificar se um número binário é par



Exemplo de MT

- » Uma máquina de Turing que entra em *loop* infinito se o número for ímpar ou se a entrada for vazia



Algoritmos e MT

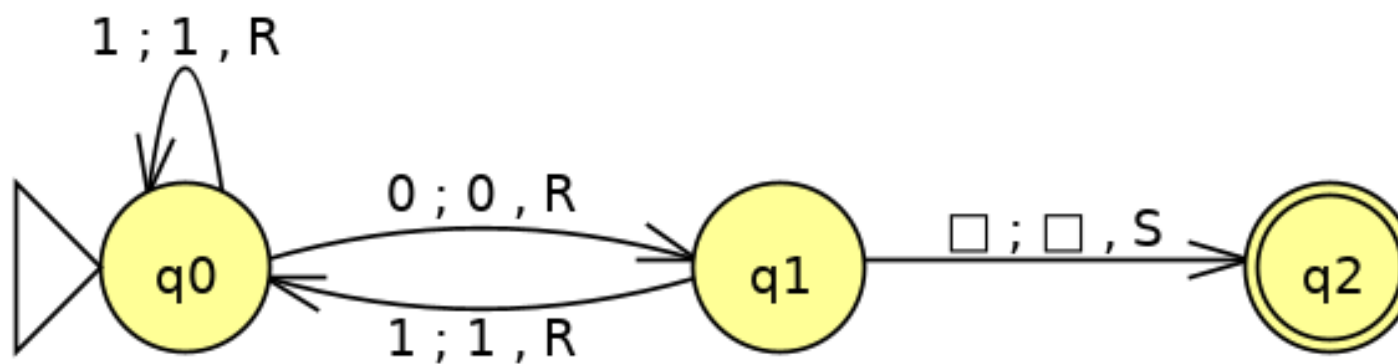
- » Uma máquina de Turing que sempre para para qualquer entrada é um "modelo de algoritmo"
- » Todo **algoritmo** possui uma máquina de Turing equivalente que para para qualquer entrada
- » Portanto algoritmos e máquinas de Turing são conceitos equivalentes

Linguagens e MT

- » Máquinas de Turing, por sua vez, são equivalentes a gramáticas livres de contexto
- » Uma máquina de Turing pode ser usada para
 - » Descrever uma linguagem
 - » Verificar se uma cadeia pertence à linguagem
- » Portanto um algoritmo pode ser usado para
 - » Descrever uma linguagem
 - » Verificar se uma cadeia pertence à linguagem

Máquina de Turing Universal

- » Uma máquina de Turing pode ser descrita como uma cadeia



$q_0 1 : q_0 1 R \# q_0 0 : q_1 0 R \# q_1 1 : q_0 1 R \# q_1 \square : q_2 \square S \# q_2^* : \text{HALT}$

Máquina de Turing Universal

- » Assim, uma máquina de Turing pode ser entrada de outra máquina de Turing
- » Uma **máquina de Turing universal** U é uma máquina de Turing que aceita como entrada
 - » A descrição de uma máquina de Turing M
 - » Uma cadeia de entrada w
- » A máquina de Turing universal simula a execução de M com a cadeia de entrada w
 - » Um computador é "equivalente" a uma máquina de Turing universal (com memória finita)

Completude de Turing

- » Dizemos que uma linguagem de programação é **Turing-completa** se ela pode ser utilizada para simular uma máquina de Turing
- » Todas as linguagens de programação “úteis” são Turing-completas
 - ~ Se uma linguagem de programação é Turing-completa, então qualquer algoritmo pode ser descrito nela

Completude de Turing

- » Podemos mostrar que uma linguagem de programação é Turing-completa de duas formas
 - » Escrevendo um programa que equivale a uma máquina de Turing universal
 - » Implementando um interpretador de outra linguagem Turing-completa
 - ~ Se uma linguagem de programação X é Turing-completa...
 - ~ então um interpretador de X em uma linguagem de programação Y mostra que Y também é

Linguagens Turing-completa

» Exemplos

» Brainf*ck

~ <https://en.wikipedia.org/wiki/Brainfuck>

» Regra 110

~ https://en.wikipedia.org/wiki/Rule_110

» A microinstrução MOV da arquitetura x86 (!)

~ <https://www.cl.cam.ac.uk/~sd601/papers/mov.pdf>

~ https://en.wikipedia.org/wiki/One_instruction_set_computer

Agenda

- » Máquinas de Turing
- » Três linguagens procedimentais
- » Fluxo: avaliação de expressões
- » Fluxo: sequência e seleção
- » Fluxo: repetições

Python: uma linguagem “simples”

- » Projetada por Guido van Rossum em 1990
- » Projetada para ser simples
 - » Princípio norteador: “deve haver um jeito óbvio (e preferencialmente apenas um) de se fazer alguma coisa em Python”
- » Desenvolvimento contínuo
 - » PEP (*Python Enhancement Proposal*)
 - ~ Propostas da comunidade para acrescentar funcionalidades ou corrigir problemas do Python

Python: algumas PEP

- » PEP 8: estilo de código
 - » Define um estilo de código comum para todos os programadores
 - ~ Usar espaços ou TAB para indentar?
 - ~ Onde usar linhas em branco?
 - ~ <https://www.python.org/dev/peps/pep-0008>
- » PEP 20: o “Zen” do Python
 - » Define os princípios norteadores da linguagem
- » PEP 3099
 - » Coisas que não vão mudar no Python 3000

Python: fluxo sequencial

```
#!/usr/bin/python3

def fib(n):
    if n <= 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

s = input("Digite um n: ")
x = int(s)
fx = fib(x)
print(f"Fibonacci de {x} e' {fx}")
```

Perl: uma linguagem “expressiva”

- » Projetada por Larry Wall em 1987
- » Projetada para substituir awk e sed
 - » Anúncio na Usenet
 - ~ <https://groups.google.com/forum/#!topic/comp.sources.unix/Njx6b6TiZos>
- » Designada para ser prática (fácil de usar, eficiente, completa) em vez de elegante
- » Muito influenciada por comandos de terminal
 - ~ Chamadas de função não exigem parênteses

Perl: uma linguagem “expressiva”

- » Perl possui três tipos de dados
 - » `$x` é um escalar
 - ~ Pode ser uma string, um inteiro, um ponto flutuante ou uma referência
 - » `@x` é um vetor
 - ~ Pode conter qualquer elemento escalar
 - ~ Equivalente às listas de Python
 - » `%x` é uma *hash*
 - ~ Contém entradas do tipo chave => valor
 - ~ Equivalente aos dicionários de Python

Perl: fluxo sequencial

```
#!/usr/bin/perl

@vetor = (1, 2, 3, 4);
$numel = $#vetor;
print "O vetor contem $numel elementos:\n";
print @vetor;
print "\n";

$primeiro = $vetor[0];
print "O primeiro elemento da lista eh $primeiro\n";

$primeiro = shift @vetor;
print "O primeiro elemento da lista era $primeiro:\n";
print @vetor;
print "\n";
```

Perl: fluxo sequencial

```
#!/usr/bin/perl

@vetor = (1, 2, 3, 4);
$numel = $#vetor;
print("O vetor contem $numel elementos:\n");
print(@vetor);
print("\n");

$primeiro = $vetor[0];
print("O primeiro elemento da lista eh $primeiro\n");

$primeiro = shift(@vetor);
print("O primeiro elemento da lista era $primeiro:\n");
print(@vetor);
print("\n");
```

O mesmo código, com os parênteses opcionais

C: uma linguagem “genérica”

- » A linguagem C foi criada por Dennis Ritchie e Brian Kernighan em 1972
 - » Uma linguagem de baixo nível
 - » Baseada em B
 - » Com tipos de dados
 - ~ Apesar de não impor muitas restrições
 - » Programação estruturada
 - ~ Todos os programas contêm uma função chamada `main()`
 - ~ A partir do seu ponto de entrada, a execução segue o fluxo sequencial

C: uma linguagem “genérica”

- » C foi projetada para ser compilada
 - » Todas as variáveis *devem* ser declaradas
 - » Os tipos das variáveis *devem* ser conhecidos em tempo de execução
 - » As variáveis não podem mudar de tipo durante a execução
 - ~ Tipagem estática
 - ~ Tipagem relativamente fraca
 - ~ Mais fraca que Java, mais forte que JavaScript

C: uma linguagem “genérica”

- » Atualmente a linguagem C é um padrão ISO
 - » *International Organization for Standardization* ou Organização Internacional para Padronizações
 - ~ O nome ISO não é um acrônimo; ele vem do grego *isos* (ἴσος), que significa “igual”
 - » ISO/IEC 9899:2011 (C11)
- » A linguagem C foi base da linguagem C++
 - » C++ também é hoje um padrão ISO
 - » ISO/IEC 14882:2017 (C++17)

C++: uma linguagem “para a todos conquistar”

- » C++ cresceu muito desde sua versão inicial em 1985
- » Sofreu influência de muitas linguagens
 - » Ada, ALGOL 68, C, CLU, ML, Simula, ...
- » É um “monstro” multi-paradigma
 - » Procedimental
 - » Orientado a objetos
 - » Funcional
 - » Paralelo
 - » Genérico

C++: uma linguagem “para a todos conquistar”

```
#include <iostream>

using std::cout;
using std::cin;

int main()
{
    int n, i, j;
    cout << "Digite um numero\n";
    cin >> n;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            cout << '*';
        cout << '\n';
    }
    return 0;
}
```

Agenda

- » Máquinas de Turing
- » Três linguagens procedimentais
- » Fluxo: avaliação de expressões
- » Fluxo: sequência e seleção
- » Fluxo: repetições

Precedência e associatividade

- » A maioria das linguagens de programação permite que operadores sejam associados em expressões
 - » Isso é um exemplo de **ortogonalidade**
- » Questões de projeto
 - » Qual a precedência dos operadores?
 - » Qual a associatividade dos operadores?

Precedência e associatividade

- » Linguagens têm tabelas de associatividade e precedência
 - » C tem 15 níveis
 - » Python tem 13 níveis
 - » C++ tem 18 níveis
 - » Perl tem 24

Precedência e associatividade

- » Uma expressão típica em C

```
int numeros[] = {1, 3, 6, 16, 32, 42};  
int *ptr = &numeros[3];
```

- » A linguagem assegura que o operador & tem menor precedência que o operador [] para que a expressão acima signifique o "*endereço do quarto elemento*"

Precedência e associatividade

- » Uma expressão típica em Perl

```
$nome_completo =~ /\s*(\w+)\s+(\w+)/ or die "No match";  
$nome = $1;  
$sobrenome = $2;
```

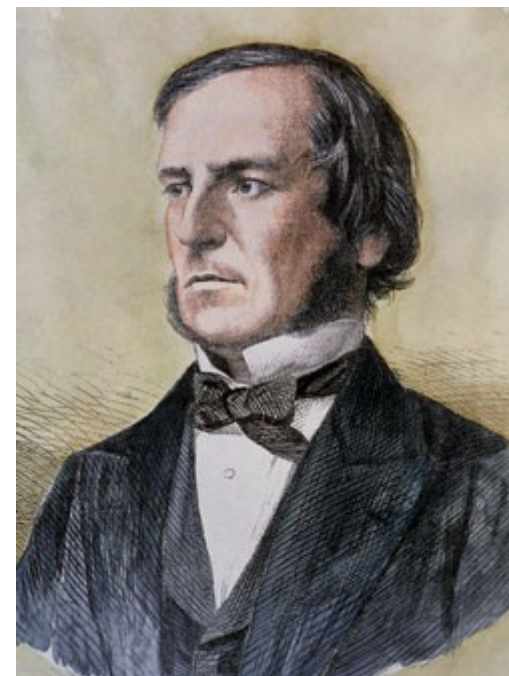
- » O operador `or` tem menor precedência que os outros operadores
 - ~ O comando **die** só é executado se a expressão à esquerda retornar um valor falso

Precedência e associatividade

- » As regras de precedência da linguagem influenciam muito na facilidade de escrita e na facilidade de leitura
 - » Algumas regras são intuitivas ou devem ser internalizadas pelos programadores
 - ~ `&estrutura->vetor[10]`
 - » Em outros casos, é melhor utilizar parênteses
 - ~ `terceiro_bit = (num >> 2) & 1`

Álgebra booleana

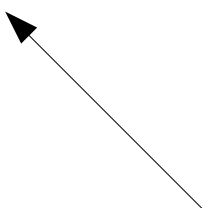
- » “Álgebra com valores verdades”
 - » Variáveis: $x, y, z...$
 - » Constantes: 1 e 0
 - ~ 1: verdadeiro
 - ~ 0: falso
 - » Operadores
 - ~ \wedge : conjunção (e)
 - ~ \vee : disjunção (ou)
 - ~ \neg : negação



George Boole

Álgebra booleana

- » Uma expressão booleana pode ter um valor verdadeiro ou falso
- » $x \wedge y$
- » $x \vee y$
- » $\neg(x \wedge y) = (\neg x \vee \neg y)$ (lei de DeMorgan)



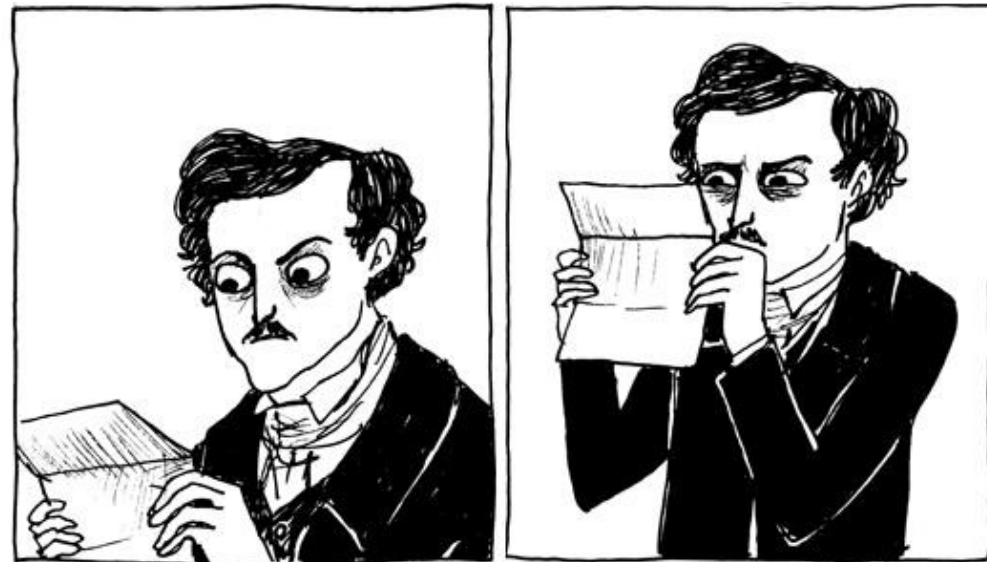
Em álgebra booleana **existe** igualdade entre as variáveis, ao passo que em lógica proposicional falamos apenas de **equivalência** entre proposições

Operadores booleanos

- » Qualquer linguagem “que se preze” oferece operadores booleanos
 - » C, C++, Perl, Python, JavaScript etc.
 - ~ A constante “falso” é 0
 - ~ Qualquer outro valor é a constante “verdadeiro”
 - ~ &&, || e ! representam conjunção, disjunção e negação
 - » Perl, C++, Python, Pascal etc.
 - ~ A constante “falso” é False e/ou false
 - ~ A constante “verdadeiro” é True e/ou true
 - ~ and, or e not

Operadores booleanos

- » Qualquer linguagem “que se preze” oferece operadores booleanos
 - » **C, C++, Perl, Python**, JavaScript etc.
 - ~ A constante “falso” é 0
 - ~ Qualquer outro valor é a constante “verdadeiro”
 - ~ &&, || e ! representam conjunção, disjunção e negação
 - » **Perl, C++, Python**, Pasca
 - ~ A constante “falso” é Fa
 - ~ A constante “verdadeirc
 - ~ and, or e not



Avaliação de curto-circuito

- » Os operandos de conjunção e disjunção podem interromper a avaliação de uma expressão se o resultado puder ser calculado antecipadamente
- » Utilizado para proteger expressões que não podem ser avaliadas

```
#!/usr/bin/perl  
use strict;  
open FILE, ">arquivo.txt";  
print FILE "Hello, world!\n" or die "$!";  
close FILE;
```

Avaliação de curto-circuito

» Sintaxe e semântica do operador `or` em Perl

» `<or_expr> ::= <expr>[1] or <expr2>[2]`

»
 `parse(<expr>[1])`
 `if <expr>[1] == true goto out`
 `parse(<expr>[2])`
 `out: ...`

Agenda

- » Máquinas de Turing
- » Três linguagens procedimentais
- » Fluxo: avaliação de expressões
- » Fluxo: sequência e seleção
- » Fluxo: repetições

Sequência e seleção

- » A primeira linguagem a oferecer um comando de seleção foi FORTRAN
- » $\langle \text{if_stmt} \rangle ::= \text{if } (\langle \text{expr} \rangle) \langle \text{L1} \rangle \langle \text{L2} \rangle \langle \text{L3} \rangle$
- »
`evaluate($\langle \text{expr} \rangle$)`
`if $\langle \text{expr} \rangle < 0$ goto $\langle \text{L1} \rangle.\text{value}$`
`if $\langle \text{expr} \rangle > 0$ goto $\langle \text{L3} \rangle.\text{value}$`
`goto $\langle \text{L2} \rangle.\text{value}$`

Python e Perl: if..else

- » COBOL introduziu o conceito de IF..ELSE
 - » Utilizava comandos no lugar de linhas
 - » Hoje é uma construção existente em praticamente todas as linguagens

```
#!/usr/bin/python3
```

```
if condição:  
    comandos  
else:  
    comandos
```

```
#!/usr/bin/perl
```

```
if (condição) {  
    comandos  
}  
else {  
    comandos  
}
```

C++: if..else if..else if..else

```
if (condição)
    comando
else if (condição)
    comando
else if (condição)
    comando
else
    comando
```

```
if (condição) {
    comandos
}
else if (condição) {
    comandos
}
else if (condição) {
    comandos
}
else {
    comandos
}
```

Python: if..else if..else if..else

- » Para evitar o “senão pendurado”, Python exige indentação ou o comando elif

```
if condição:  
    comandos  
else:  
    if condição:  
        comandos  
    else:  
        if condição:  
            comandos  
        else:  
            comandos
```

```
if condição:  
    comandos  
elif condição:  
    comandos  
elif condição:  
    comandos  
else:  
    comandos
```

Perl: if..else if..else if..else

- » Em Perl é elsif, mas o funcionamento é o mesmo

```
if (condição) {  
    comandos  
}  
else {  
    if (condição) {  
        comandos  
    }  
    else {  
        if (condição) {  
            comandos  
        }  
        else {  
            comandos  
        }  
    }  
}
```

```
if (condição) {  
    comandos  
}  
elsif (condição) {  
    comandos  
}  
elsif (condição) {  
    comandos  
}  
else {  
    comandos  
}
```

Ortogonalidade do if em ALGOL

- » ALGOL permite que if..then..else seja utilizado como expressão

```

» x := if x = y then
      p
    else
      q
    fi

```

Para evitar o “senão pendente”, o **if** é finalizado por **fi**

- » Ou como blocos

```

» x := if x = y then
      begin
        y := funcao(x);
        p := y;
      end
    else
      q
    fi

```

O valor do bloco é a última expressão avaliada

Note que aqui não há ;

Agenda

- » Máquinas de Turing
- » Três linguagens procedimentais
- » Fluxo: avaliação de expressões
- » Fluxo: sequência e seleção
- » Fluxo: repetições

Laços de repetição: while..do

- » Todas as nossas linguagens possuem o laço de repetição while..do
 - » `<while_stmt> ::= while (<expr>) <stmt>`
 - » `início: evaluate(<expr>)
if <expr> == false goto out
parse(<stmt>)
goto início
out: ...`

Laços de repetição: while..do

» Mas nem todas possuem o laço do..while

» `<do_while> ::= do <stmt> while (<expr>)`

```
» inicio: parse( <stmt> )  
          evaluate( <expr> )  
          if <expr> == false goto out  
          goto inicio  
out:     ...
```


Expressividade em Perl

- » Perl permite escrever comandos de controle em notação pós-fixa

- » `<while_pf> ::= <stmt> while <expr>`

- »

```
início: evaluate( <expr> )  
      if <expr> == false goto out  
      parse( <stmt> )  
      goto início  
out:   ...
```

Laços de repetição: for

- » C, C++ e Perl possuem o laço for com 3 expressões, mas Python tem uma sintaxe diferente...

» `<for_stmt> ::= for (<expr>[1] ;
 <expr>[2] ; <expr>[3]) <stmt>`

»
 `evaluate(<expr>[1])`
 `inicio: evaluate(<expr>[2])`
 `if <expr>[2] == false goto out`
 `parse(<stmt>)`
 `evaluate(<expr>[3])`
 `goto inicio`
 `out: ...`

Laços de repetição: for

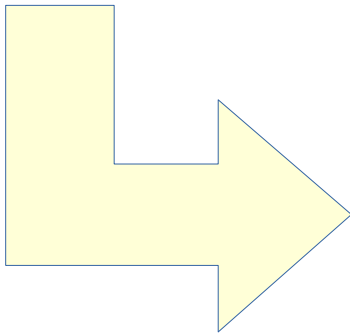
- » Em Perl e Python, o laço for pode iterar sobre uma coleção de objetos
 - » `<for_stmt> ::=`
`for <var> in <colecao>: <stmt>`
 - » Semântica
 - ~ Se a `<colecao>` for vazia (ex.: um vetor vazio), nada é feito
 - ~ Senão, para cada elemento da coleção, `<var>.value` recebe o valor do próximo elemento e o comando `<stmt>` é executado. A coleção não é alterada

Laços de repetição: for

- » Em Perl e Python, o laço for pode iterar sobre uma coleção de objetos

```
#!/usr/bin/python3
```

```
vetor = [1, 1, 2, 3, 5, 12]  
for num in vetor:  
    print(num, vetor)
```



```
1 [1, 1, 2, 3, 5, 12]  
1 [1, 1, 2, 3, 5, 12]  
2 [1, 1, 2, 3, 5, 12]  
3 [1, 1, 2, 3, 5, 12]  
5 [1, 1, 2, 3, 5, 12]  
12 [1, 1, 2, 3, 5, 12]
```