Introdução

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Linguagens de programação

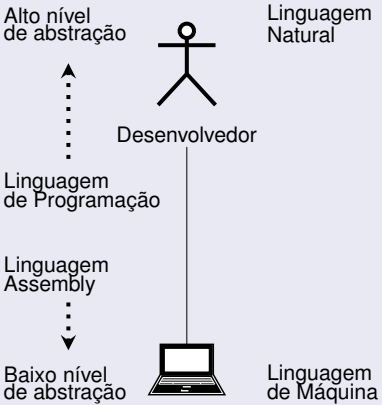
- “É qualquer notação para a descrição de algoritmos e estruturas de dados” (*Pratt, 1975. Programming Languages: Design and Implementation*).
- “Uma linguagem que tem por objetivo expressar programas de computador e é capaz de expressar qualquer programa de computador” (*MacLennan, 2016. Principles of Programming Languages: Design, Evaluation, and Implementation*).

Abstração

- A abstração permite a distinção entre categorias e indivíduos, ou classes e instâncias.
- Por exemplo: Um carro, do nível mais abstrato para o mais concreto:
  - um veículo automotor (pode ser moto, caminhão, caro, etc...)
  - um carro propriamente dito
  - um carro de passeio ou uma SUV
  - um Koenigsegg CC
  - um Koenigsegg CC 2003
  - um Koenigsegg CC 2003 do Geraldo
  - um Koenigsegg CC 2003 do Geraldo no momento em que foi comprado

Abstração

Nível de abstração da linguagem



Domínios de programação

- Aplicações científicas
  - Grande quantidade de operações de ponto-flutuante. Uso de arranjos (vetores e matrizes). Exemplo de linguagem: Fortran.
- Aplicações de negócios
  - Produz relatórios, usa números decimais e caracteres. Exemplo de linguagem: COBOL.
- Inteligência artificial
  - Manipulação simbólica. uso de listas. Exemplos de linguagem: Prolog, LISP.
- Programação de sistemas
  - Foco em eficiência. Exemplo de linguagem: C.
- Sistemas para a Web
  - Coleção de linguagens para marcação (ex. HTML), script (ex. JavaScript), programação geral (ex. Java).

Critérios de avaliação de linguagens

- *Readability* (legibilidade): facilidade com que uma linguagem pode ser lida e compreendida. Dimensões: simplicidade, ortogonalidade, tipos de dados, sintaxe.
- *Writability*: facilidade com que uma linguagem pode ser usada para criar programas. Dimensões: simplicidade, ortogonalidade, suporte a abstração e expressividade.
- Confiabilidade: conformidade com as especificações. Dimensões: checagem de tipos, tratamento de erros, naturalidade, *aliasing*.
- Custo: custo total. Dimensões: treinamento (curva de aprendizado), produtividade, tempo de compilação e de execução, custo de compiladores, custo de confiabilidade, manutenção.

Requisitos de linguagens de programação

- Toda linguagem de programação deve ser **universal**: se um dado problema possui solução algorítmica, então a LP deve prover uma implementação que resolva este problema.
- Toda linguagem de programação deve ser **natural**: Está relacionado com a legibilidade. Dado um domínio de aplicação, as estruturas semânticas da linguagem devem prover as operações básicas deste domínio.
- Toda linguagem deve poder ser **implementável** em um computador;
- Toda linguagem deve possuir uma implementação **eficiente**.

Sintaxe e semântica

Sintaxe (forma)

define a forma ou a estrutura como cada um dos elementos da linguagem de programação é escrito. Os principais elementos de uma LP são expressões, instruções ou unidades de computação.

Semântica (significado)

define o significado de cada elemento da linguagem de programação. Qual a ideia ou abstração ele representa.

- Sintaxe + Semântica = Definição de uma linguagem de programação.

Exemplo de sintaxe e semântica

- Semântica: **fim de instrução**
- Sintaxe:
  - Em **C**/**C++**: “;” (ponto e vírgula)
  - Em **Haskell**: “\n” (fim de linha – notação posicional)

- Semântica: **bloco de comandos**
- Sintaxe:
  - Em **C**/**C++**: “{ }” (chaves)
  - Em **Pascal**: “**begin end**”
  - Em **Haskell**: indentação (notação posicional)

Descrevendo uma sintaxe

Sentença

é uma *string* de caracteres sobre algum alfabeto.

Linguagem

um conjunto de sentenças.

Lexema

é a unidade sintática de mais baixo nível de uma linguagem. Por exemplo: if, while.

Token

É uma categoria de lexema. Por exemplo: identificador, operador.

Gramáticas

- Descrevem a estrutura de uma linguagem de programação. As principais formas são:
  - Gramática livre de contexto
  - Bakus-Naur Form (BNF)

Exemplo de gramática

```
<program> -> <stmts>
  <stmts> -> <stmt> | <stmt> ; <stmts>
   <stmt> -> <var> = <expr>
    <var> -> a | b | c | d
   <expr> -> <term> + <term> | <term> - <term>
   <term> -> <var> | const
```

a

Semântica estática

Semântica estática

não está relacionada com o significado. As gramáticas livres de contexto não conseguem descrevem toda a sintaxe de uma linguagem de programação. Exemplo: Tipo dos operandos em uma expressão.

- Construtores não-livre de contexto: por exemplo, variáveis precisam ser declaradas antes do uso.
- Gramáticas de atributos: adicionam informações semânticas às gramáticas livres de contexto.
  - Especificam a semântica estática.
  - Permitem a checagem semântica (compiladores).

Gramáticas de atributos

Exemplo de gramática de atributos

```
  <assign> -> <var> = <expr>
    <expr> -> <var> + <var> | <var>
     <var> A | B | C

  actual_type: synthesized for <var> and <expr>
expected_type: inherited for <expr>
```

[a]

[a] SEBESTA (2019). *Conceitos de Linguagens de Programação. 11ed*

Gramáticas de atributos

Exemplo de gramática de atributos (cont)

```
Syntax rule:  <expr> -> <var>[1] + <var>[2]
Semantic rules:
<expr>.actual_type <- <var>[1].actual_type
Predicate:
<var>[1].actual_type == <var>[2].actual_type
<expr>.expected_type == <expr>.actual_type

Syntax rule:  <var> -> id
Semantic rule:
<var>.actual_type <- lookup (<var>.string)
```

a

[a] SEBESTA (2019). *Conceitos de Linguagens de Programação. 11ed*

Gramáticas de atributos

Exemplo de gramática de atributos (cont)

```
<expr>.expected_type <- inherited from parent

<var>[1].actual_type <- lookup (A)
<var>[2].actual_type <- lookup (B)
<var>[1].actual_type =? <var>[2].actual_type

<expr>.actual_type <- <var>[1].actual_type
<expr>.actual_type =? <expr>.expected_type
```

a

[a] SEBESTA (2019). *Conceitos de Linguagens de Programação. 11ed*

Semântica

- Programadores precisam saber o que as instruções significam.
- Compiladores precisam saber o que os construtos das linguagens fazem.
- Prova de correção deveria ser possível.
- Geração de compiladores deveria ser possível.
- Projetistas de linguagens devem detectar ambiguidades e inconsistências.

Semântica

- **Semântica operacional**: descreve o significado das execuções de instruções em uma máquina (mudanças de estado). Para alto nível de abstração, uma máquina virtual é necessária.
  - Usada para criar manuais de linguagens de programação.
- **Semântica denotacional**: notação formal e matemática do estado de entidades da linguagem. Cada instrução mapeia instâncias dessas entidades em objetos matemáticos.
  - Podem provar a correção de um programa.
- **Semântica axiomática**: baseada na lógica de predicados.
  - Difícil de ser definida e utilizada por compiladores e implementadores de linguagens.

Gerações de Linguagens de Programação

- Segundo MacLennan (1999) a história das linguagens de programação se dividem em 5 gerações.
  - 1ª geração: linguagens orientadas à máquina.
  - 2ª geração: linguagens que minimizam o uso de desvios e sequenciadores (*GOTO*).
  - 3ª geração: linguagens voltadas a aplicação (ex. C, C++, Java).
  - 4ª geração: linguagens voltadas à abstração de dados (ex. BD, web, GUI).
  - 5ª geração: linguagens funcionais, lógicas e declarativas em geral.

Paradigma de Programação

- Paradigma denota um modelo baseado em um conceito prévio.
- Paradigma de programação é um modelo, padrão ou estilo de programação suportado por determinado grupo de linguagens.
- Cada paradigma irá se basear em um modelo prévio de como problemas são resolvidos em outros domínios do conhecimento humano, tais como matemática, física, etc.
- podem ser agrupados em dois grandes grupos: linguagens assertivas e declarativas.

Linguagens assertivas

- Baseadas no conceito de comandos que afetam variáveis.
- São linguagens de aplicação geral.
- O programador fica encarregado de traçar, no algoritmo, os comandos que devem ser executados (como o problema será resolvido).
- Principais modelos de programação:
  - Programação Imperativa
  - Programação Orientada para Objetos
  - Programação Orientada a Eventos
  - Programação Orientada a Aspectos

Linguagens declarativas

- Baseadas no conceito de declaração.
- Declarações permitem dizer **o que** deve ser feito, e não **como** deve ser feito.
- Ideais para se construir cadeias de conhecimentos e relações (muito utilizadas nas áreas de inteligência artificial).
- Exemplo comercial: SQL (*Structured Query Language*), para manipulação de bancos de dados.
- Principais modelos de programação:
  - Programação Lógica
  - Programação Funcional
  - Programação Relacional

Paradigma imperativo

- Definição:
  - Programas centrados no conceito de um estado (“variáveis”) e ações (“comandos”) que manipulam o estado.
  - Primeiro paradigma a surgir e ainda é o dominante.
  - Baseado no Modelo de von Neumann.
- Vantagens:
  - Eficiência (arquiteturas atuais são uma evolução do modelo von Neumann).
  - Modelagem “natural” de aplicações do mundo real.
  - Paradigma mais do que estabelecido.
- Desvantagens:
  - Implementações demasiadamente operacionais (definem “como fazer” e não “o que fazer”)

Paradigma orientado para objetos

- Definição:
  - Programas estruturados em módulos (classes) que agrupam um estado (atributos) e operações (métodos) sobre este estado.
  - Classes podem ser estendidas, via herança, e usadas como tipos (cujos elementos são objetos).
- Ênfase em reutilização de código
- Uso mais comum junto com o paradigma imperativo. Embora existam linguagens funcionais com orientação para objetos, como CLOS (*Common LISP Object System*).

Programação orientada a eventos

- É mais uma extensão à programação imperativa pura, ou à programação orientada para objetos.
- Migração da interação usuário-computador de síncrona para assíncrona (o programa não mais deve interromper sua execução para receber entradas do usuário).
- Sistema operacional avisa que um evento acabou de ocorrer.
- O programa deverá então responder a este evento.

Programação orientada a aspectos

- Permite separar e organizar o código de acordo com a sua importância para a aplicação (*separation of concerns*).
- Complementa a programação orientação para objetos permitindo ao desenvolvedor modificar dinamicamente o modelo para criar um sistema que pode atender a novos requisitos que são transversais ao sistema.
- Maior desvantagem é a depuração.

Paradigma funcional

- Programa é uma função (ou um grupo de funções), composta de funções mais simples.
- Uso de funções e expressões na forma como elas são definidas na matemática: implementa um mapeamento de entradas em saídas, sem produzir efeito colateral.
- Linguagens de programação funcionais tratam funções como valores de primeira classe.
- Vantagens:
  - Maior facilidade para se implementar prova de propriedades
  - Possibilidade de se explorar concorrência de forma natural.
- Desvantagem:
  - As implementações em geral são ineficientes.

Programação lógica

- Programa implementa uma relação ao invés de um mapeamento.
- Relações podem ser unárias, binárias, ternárias etc.
- Como relações são mais gerais que mapeamentos, a programação em lógica é, potencialmente, de nível mais alto que a programação imperativa ou funcional.
- Restringe sua implementação a cláusulas de Horn, baseadas na relação SE-ENTÃO.
- A computação de um programa baseado no paradigma lógico é realizada, testando-se uma dada consulta.
- As vantagens e desvantagens são as mesmas do paradigma funcional.

Linguagens multiparadigma

- Tendência é combinar paradigmas.
- Por exemplo, praticamente todas as LPs orientadas para objetos possuem avaliação tardia e funções *lambda*, típicos da programação funcional.
- Ex.: C#, Java 8, CLOS, etc.

Histórico das linguagens de programação