SÃO PAULO TECH SCHOOL – SPTECH

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATEUS CASTRO FORTES

LINGUAGENS FORMAIS E AUTÔMATOS

PESQUISA APLICADA

SÃO PAULO – SP

2022

SUMÁRIO

1. Introdução

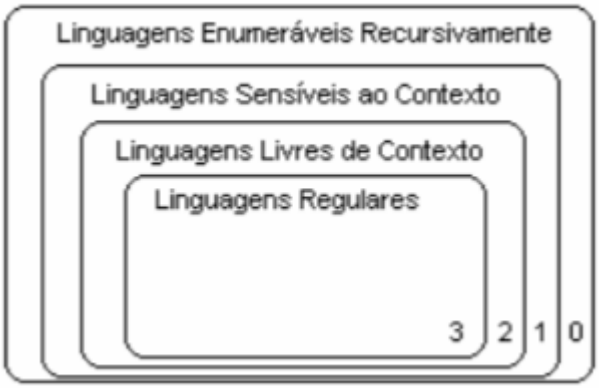
O objetivo desta pesquisa é estudar conceitos e princípios relacionados à teoria da computação para entender a natureza geral da computação. Para estudar esses princípios básicos constroem-se modelos de computadores abstratos para a resolução de pequenos problemas.

Para modelar o hardware de um computador é introduzido o conceito de **autômatos**.

2. Autômatos

Um autômato é uma construção que **possui todas as características indispensáveis de um computador** digital: entrada, saída, armazenagem temporária e tomadas de decisão. Pode-se sintetizar que autômato é um **reconhecedor da linguagem**. Através dos reconhecedores é possível identificar se a cadeia de símbolos pertence ou não a uma linguagem.

Estudar computação do ponto de vista teórico é sinônimo de **caracterizar o que é ou não computável**. E para isso é necessário um modelo matemático que represente o que se entende por computação. E um dos modelos principais, é o da **Máquina de Turing**.



2.1 Linguagem formal

Linguagem formal é uma abstração das características gerais de uma linguagem de programação, possuindo um conjunto de símbolos, regras de formação de sentenças, endentação etc.

A seguir uma tabela traz uma correspondência entre as classes de linguagens, gramáticas e reconhecedores.

Linguagem	Gramática	Reconhecedor
Tipo 0: Enumeráveis Recursivamente	Irrestritas	Máquinas de Turing
Tipo 1: Sensíveis ao Contexto	Sensíveis ao Contexto	Autômatos Limitados Linearmente
Tipo 2: Livres de Contexto	Livres de Contexto	Autômatos com Pilha
Tipo 3: Regulares	Regulares	Autômatos Finitos

3. Analisador Léxico

Podendo também ser chamado de ***scanner ou lexer***, um **analisador léxico** é um programa que é responsável por analisar o fluxo de entrada do compilador, e retornar para o **analisador sintático** a sequência de tokens identificados. Para isso ele implementa um **autômato finito**, reconhecendo (ou não) *Strings* como símbolos válidos de uma linguagem. E para poder implementá-lo é necessária uma descrição do autômato que reconhece as **sentenças da gramática ou expressões regulares** (*RegEx*).

Conhecendo essa informação, três procedimentos auxiliares podem ser executados para o analisador léxico.

- **Estado Inicial**: que recebe como argumento a referência para o autômato e retorna o seu estado inicial;

- **Estado Final**: que recebe como argumentos a referência para o autômato e a referência para o estado atual. O procedimento retorna um valor booleano baseado no estado especificado;

- **Próximo Estado**: que recebe como argumento a referência para o autômato, para o estado corrente e para o símbolo a ser analisado. O procedimento consulta a tabela de transições e retorna o próximo estado do autômato, ou o valor nulo se não houver transição possível.

O **analisador léxico** inicia sua operação definindo o estado inicial como o atual. Ele mantém uma **rotina de iteração** para reconhecer os próximos símbolos da sequência, analisando se estado do próximo símbolo é final.

Se houver símbolo a ser analisado, então o procedimento deve continuar o processo de reconhecimento. Se não houver transição possível para o símbolo corrente, então a sentença não foi reconhecida e o procedimento deve encerrar, retornando false.

3.1. Exemplo de aplicação

Em um compilador C um código para calcular a circunferência de um círculo é recebida como entrada.

```
ci = 2 * PI * radius
```

No primeiro momento do compilador, o trecho de código passa pelo analisador léxico. Após a leitura do primeiro caractere (c) o programa identifica que a sequência define um identificador. Assim que o primeiro caractere do símbolo de atribuição é encontrado, o valor é retornado ao analisador sintético, e o processo continua até o cessar de transições possíveis no símbolo atual.

4. Analisador Sintático

Também conhecido como ***parser***, o **analisador sintático** tem como tarefa principal determinar se o programa de entrada representado pelo fluxo de tokens possui as sentenças válidas para a linguagem de programação.

A análise sintática é a segunda etapa do processo de compilação e na maioria dos casos utiliza **gramáticas livres** de contexto para especificar a sintaxe de uma linguagem de programação. Essa então, que representam uma gramática formal e podem ser escritas através de **algoritmos** que fazem a derivação de todas as possíveis construções da linguagem.

As derivações têm como objetivo determinar se um fluxo de palavras se encaixa na sintaxe da linguagem de programação.

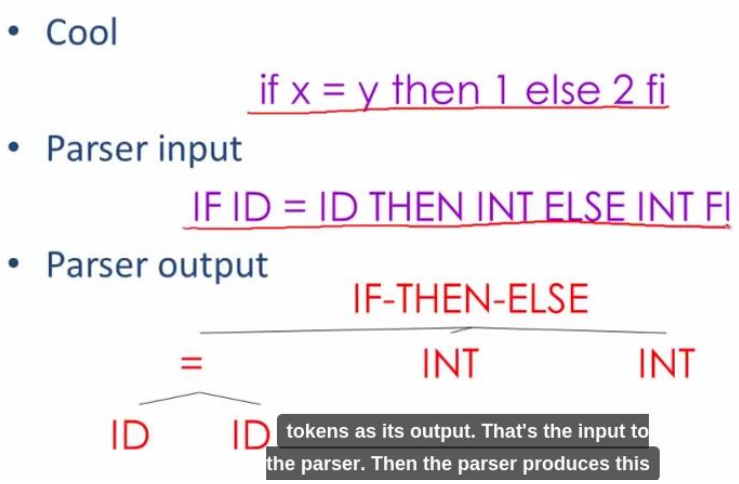
Alguns termos utilizados na definição de linguagens de programação são:

- **Símbolo**: são os elementos mínimos que compõe uma linguagem. Na linguagem humana são as letras.
- **Sentença**: É um conjunto ordenado de símbolos que forma uma cadeia ou uma *String*. Na linguagem humana são as palavras.
- **Alfabeto**: É um conjunto de símbolos. Na linguagem humana é o conjunto de letras {a, b, c, d, ...}
- **Linguagem**: É o conjunto de sentenças. Na linguagem humana são os conjuntos de palavras {compiladores, linguagem, ...}
- **Gramática**: É uma forma de representar as regras para formação de uma linguagem.

E assim como na linguagem humana, esses termos por si só, não oferecem nenhum sentido para o programa. Mas, a **sequência ordenada desses elementos** com base nas especificações de um desenvolvedor, totalizam um programa.

4.1. Exemplo de aplicação

Um exemplo de como um analisador sintático valida as expressões criadas na linguagem de programação, é a seguinte imagem:



O analisador sintático agrupa os tokens em frases gramaticais usadas pelo compilador com o objetivo de criar uma saída que representa a estrutura hierarquia do programa fonte.

Abaixo estão presentes as especificações de entrada e saída das etapas analisadas até o momento:

Fase	Entrada	Saída
Lexer	Conjunto de caracteres	Conjunto de tokens
Parser	Conjunto de Tokens	Árvores Sintática

5. Analisador Semântico

Esse programa é responsável por analisar os erros semânticos presentes no código que dizem respeito ao **escopo dos nomes, correspondência entre declarações e os próprios nomes, e compatibilidade dos tipos em diversas sintaxes de codificação**. Ele utiliza **a árvore sintática** e a **tabela de símbolos** para fazer a análise semântica.

As validações que não podem ser executadas pelas etapas anteriores devem ser executadas durante a análise semântica a fim de garantir que o programa fonte esteja **coerente** e o mesmo possa ser convertido para **linguagem de máquina**.

A análise semântica percorre a **árvore sintática** relacionando os identificadores com seus dependentes de acordo com **a estrutura hierárquica**.

5.1. Exemplo de aplicação

Um exemplo que ilustra muito bem essa etapa de validação de tipos é a atribuição de objetos de tipos ou classe diferentes. Em alguns casos, o compilador realiza a conversão automática de um tipo para outro que seja adequado à aplicação do operador. Por exemplo a expressão:

```
var s: String;
s := 2 + '2';
```

Agora, analisando um exemplo de código em Pascal:

```
function Soma(a, b : Integer) : Integer;
var
  i : Integer;
begin
  i := a + b;
  Result := i;
end;
```

No exemplo acima o analisador semântico deverá ter uma série de preocupações para validar o significado de cada regra de produção, como:

- Conferir se os identificadores foram declarados;
- Qual o tipo de informação computável os identificadores trazem;
- Qual o escopo desses identificadores.

Os tipos de dados são muito importantes nessa etapa da compilação, eles são **notações que as linguagens de programação utilizam para representar um conjunto de valores**. E com base nesses tipos o analisador semântico pode **definir quais valores podem ser manipulados, ou não**.

6. Conclusão e aplicação

O estudo desses processos e programas computacionais permitem uma especialização ímpar no universo das máquinas. A sequência desses processos e a existência destes conceitos são a base para grande parte da teoria da computação, e pode ser aplicada de diversas maneiras neste contexto.
Para a aplicação no projeto de Pesquisa e Inovação, é possível ressaltar a compreensão mais à fundo dos algoritmos envolvidos no sistema, contemplando novas métricas e conhecendo novas etapas presentes no funcionamento da aplicação.

7. Bibliografia

Teoria da Computação e Linguagens Formais:

http://wwwp.fc.unesp.br/~simonedp/disctcBCC.htm

Horário de acesso: 11/04/2022 – 20:50

Análise Sintática:

https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/syntax-analysis.html

Horário de acesso: 12/04/2022 – 17:10

A construção de Compiladores:

http://www.inf.ufes.br/~tavares/labcomp2000/intro2.html#:~:text=Analisador%20Semântico%3A%20verifica%20os%20erros,tipos%2C%20em%20expressões%20e%20comandos.

Horário de acesso: 12/04/2022 – 17:40

Analisadores Léxicos:

https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node50.html#:~:text=Um%20analisador%20léxico%2C%20ou%20scanner,ou%20expressão%20regular%20de%20interesse.

Horário de acesso: 13/04/2022 – 21:20