Analisador Semântico

Lucas Garavaglia

UNIOESTE

20 de setembro de 2021

Conteúdo

Participação de cada membro

Lucas:

- Criação da gramática
- Código fonte
- Documentação
- Slides

Descrição da Linguagem

Linguagem C:

- Compilada
- Procedural
- Estruturada

Tokens

```
Token para Início de bloco: [{]
Token para Final de bloco: [}]
Token para Início de Função: [(]
Token para final de Função: [)]
Token para separação: [,]
Token para Loop: [while]
Token para Condição: [if]
Token para Final de função: [return]
Token para Tipo de dado: [int]
Token para Atribuição: [=]
Token para Expressões Lógicas: [>=|<=|==|!=|>|<]
Token para Operadores matemáticos: [+|-|*|/|%]
Token para Final de linha: [;]
Token para Variável: [_|a-z|A-Z][_|a-z|A-Z|0-9]*
Token para Números: [0-9]+[[.][0-9]+]?
```

Automatos

Classes principais

- Transition
- State
- Automaton

Automatos

```
class transition {
    constructor(symbol, destinationState, currentState, symbolReg) {
        this.symbol = symbol;
        this.destinationState = destinationState;
        this.currentState = currentState;
        this.symbolReg = symbolReg;
    }
    ...
}
```

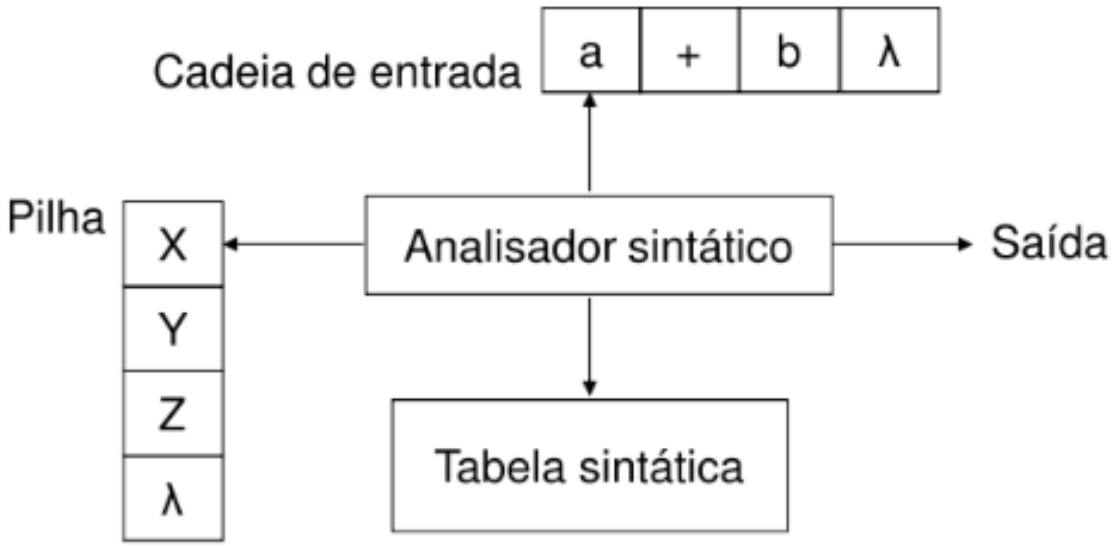
Automatos

```
class state {
    constructor(name, isFinal) {
        this.isFinal = isFinal;
        this.name = name;
        this.transitions = [];
    }
    ...
    newTransition(symbol, destinationState, currentState, symbolReg = /[]/) {
        this.transitions.push(
            new transition(symbol, destinationState, currentState, symbolReg)
        );
    ...
    getNextState(symbol) {
        for (var i = 0; i < this.transitions.length; i++) {
            if (
                this.transitions[i].getSymbol() == symbol ||
                this.transitions[i].getSymbolReg().test(symbol)
            ) {
                return this.transitions[i].getState();
            }
        }
        return false;
    }
}
```

Analisador Sintático

Classe principais

- parser
  - Analise Sintática
  - Tabela Sintática



Visão geral da gramática

```
<STA> ::= tokenConditional tokenStartFunction<EXPFUNC>
          | tokenStartBlockFunction<STA><ENDBLOCK>
          | tokenWhile tokenStartFunction<EXPFUNC>
          | tokenReturn<EXP>tokenEndLine<STA>
          | tokenIdentifier<K>tokenEndLine<STA>
          | tokenDataType <DECFUNC>
          | tokenUnsigned tokenDataType <DECFUNC>
          | tokenTypeDef <TD> <STA>
          | $

<ENDBLOCK> ::= <STA>tokenFinalBlockFunction

<EXPFUNC> ::= <EXP>tokenFinalFunction tokenStartBlockFunction<STA><ENDBLOCK>

<T> ::= tokenAssignments<EXP>
        | tokenSeparator tokenIdentifier<T>

<L> ::= tokenStartFunction<P>tokenFinalFunction
        |<T>tokenEndLine

<DECFUNC> ::= tokenIdentifier <L> <STA>

<EXP> ::= tokenIdentifier<S>
          | tokenStartFunction<EXP>tokenFinalFunction
          | tokenNumber<S>
```

Visão geral da gramática

```
<K> ::= tokenAssignments<EXP>
        | tokenStartFunction tokenIdentifier<B>tokenFinalFunction

<B> ::= tokenSeparator tokenIdentifier<B>
        | $

<S> ::= tokenOperator<EXP>
        | tokenExpression<EXP>
        | $

<P> ::= tokenDataType tokenIdentifier<Z>
        | $

<Z> ::= tokenSeparator tokenIdentifier<Z>
        | $

<TD> ::= tokenDataType tokenIdentifier tokenEndLine
```

Visão geral da gramática

```
this.syntacticTable = {
    "<STA>": [...],
    "<T>": [...],
    "<L>": [...],
    "<EXP>": [...],
    "<K>": [...],
    "<B>": [... ],
    "<S>": [...],
    "<F>": [...],
    "<P>": [...],
    "<P>": [...],
    "<TD>": [...]
    }
```

Visão geral analisador semântico

- Gramática de atributo
- Identificação de erro

Obrigado pela atenção!