**Teoria de Compilador – Tradutor**

(**Fonte => Objeto**)

- Processador de Linguagens: **Compilador**

**Entrada**: Programa Fonte

**Computador**

**Saída**: Linguagem Máquina

Código Fonte

Erros de sintaxe; processador de linguagens; mapeamento de entrada/saída mais rápido; diagnóstico de erros é mais dificultoso.

Máquina

**Entrada**

**Saída**

Ações

Código Objeto

Compilador

- Processador de Linguagens: **Interpretador**

Máquina

Ações

**Saída**

**Entrada**

Interpretador

Código Fonte

- Processador de Linguagens: **Abordagem Híbrida (Java)**

**Saída**

JIT (Just in Time) **Compilador** (Objeto)

Interpretador do Código JVM

**Entrada**

Byte Code

Compilador

Código Fonte

Máquina

Ações

Processador de Linguagens (Linguagens – Forma de Comunicação - Formalismo)

**Aplicações Linguagens**: SQL; HTML; LaTeX

**Alan Turing** (Teoria da Computação) – Pai da Computação.

Turing define que, algoritmo é o que o computador consegue fazer.

**Teoria do Formalismo** => Algoritmos são linguagens – forma de comunicação – formalismo - Linguagem Formal)

Linguagem (Conjunto)

Cadeia (**Entrada**)

{W | W são os caracteres do alfabeto romana/latino}

abbc ...

Leitura

Máquina Abstrata

Fita; Fita Infinita; Máquina de Estado; Código Regulares

Não Pertence a Linguagem

Pertence a Linguagem

Computador

**Exemplo**:

Tipo **Máquina Abstrata**: PDA

C -> As | SbA

A -> cbX (**formalismo**)

Cadeia (**Entrada**)

{Linguagem Livre de Contexto}

ações

Autômatos Finitos

Leitura

PAD (Pilha)

Computador

Não Pertence

Pertence

Linguagem – não livre de contexto;

Analisador Sintático –> **gramática** (expressão; palavras reservadas, comandos, etc)

Em **Compiladores**

S -> Expr;

Expr -> Expr + Expr | Expr – Expr | ...

Cadeia (Entrada: **código fonte**)

{Linguagem Livre de Contexto}

Int x = 13

Int y = 17

If ( x > y) {

System.Out.Prinbln

(“X > Y); }

Leitura

Compilador

Máquina Abstrata

Pertence (**Aceita**) + Ações Semânticas

Não Pertence (**Rejeita**) + Erros

Computador

mov R1, #47F3

mov R2, #BB5E

sub R1, R2, R3

JN2 #47FE

Mensagem de erro

1. **Análise Sintática** (forma) – sintaxe, tudo que está na gramática (livre de contexto) é sintático, o resto é considerado semântico.
2. **Análise Semântica** (significado), checa a consistência com a definição da linguagem; coleta informações sobre os tipos e armazena na árvore de sintaxe ou na tabela de símbolos; checagem de tipos / coerção (adequação dos tipos)

Exemplo: int nome = “Hello World”;

Na gramática (**dentro – sintático**), fora semântico.

**LFA** (Linguagens Formais e Autômatos), temos: Árvore de análise sintática e somente com elementos da gramática.

**Estrutura do Compilador**

**Duas Etapas**: Análise (**frontend – código fonte**); Síntese (**backend – código objeto**)

Gramática (estrutura gramatical); Erros (sintáticos; semânticos); Tabela de Símbolos

**Fases do Compilador (Azul – frontend; Cinza – backend)**

Cadeia (Fluxo de Caracteres) ->

Analisador Léxico (**Scanning**)

* Fluxo de Tokens: (key : value) **Tabela de Símbolos** ->

Analisador Sintático (**Parsing**)

* Árvore de Sintaxe ->

Analisador Semântico

* Árvore de Sintaxe ->

Gerador de Código Intermediário (Objeto)

* Representação Intermediária ->

Otimizador de Código (independente da máquina)

* Representação Intermediária ->

Gerador de Código

* Código da Máquina Alvo ->

Otimizador de Código (dependente da máquina)

* Código da Máquina Alvo

**Análise Léxica (Lexemas - Scannig)**

position = initial + rate \* 60

|  |  |  |  |
| --- | --- | --- | --- |
| **Entrada** | **Lexema** | **Tipo** | **...** |
| 1 | position | int | ... |
| 2 | initial | int | ... |
| 3 | rate | float | ... |
| 4 | 60 | int | ... |

**Tabela de Símbolos**

|  |  |
| --- | --- |
| **Lexema** | **Token** |
| position | <id,1> |
| = | <=> |
| initial | <id,2> |
| + | <+> |
| rate | <id,3> |
| \* | <\*> |
| 60 | <num, 4) |

**<id, 1>** <=> <id,2> <+> <id,3> <\*> <num,4>

**Análise Sintático (Parsing)**

Utiliza os tokens produzidos pelo analisador léxico (despreza os aspectos não livre de contexto); produz uma árvore de análise sintática (representa a estrutura gramatical do fluxo de tokens); As fases seguintes utilizam a estrutura gramatical para realizar tarefas outras análises e gerar o programa objeto.

Exemplo:

**<id, 1>** <=> <id,2> <+> <id,3> <\*> <num,4>

**atribuição**

<id,1> **soma**

<id,2> **multiplicação**

<id,3> <num,4>

**Análise Semântico (Símbolos; Tipos/Coerção)**

Checa a consistência com a definição da linguagem; informações sobre tipos de dados; checagem de tipos/coerção.

Exemplo:

**<id, 1>** <=> <id,2> <+> <id,3> <\*> <num,4>

|  |  |  |  |
| --- | --- | --- | --- |
| **Entrada** | **Lexema** | **Tipo** | **...** |
| 1 | position | int | ... |
| 2 | initial | int | ... |
| 3 | rate | float | ... |
| 4 | 60 | int | ... |

**Tabela de Símbolos**

**Tipos incompatíveis: int** vs **float**

**Coerção: intToFloat**

**atribuição**

<id,1> **soma**

<id,2> **multiplicação**

**intToFloat**

**Análise Semântico (Símbolos; Tipos/Coerção)**

**Fase Gerador de Código**

num4 = 60

t1 = **intToFloat**(num4)

t2 = id3 \* t1

t3 = id2 + t2

id1 = t3

**Fase Otimizador de Código**

num4 = 60

**t1 = id3 \* 60.0**

**id1 = id2 + t1**

t1 = **intToFloat**(num4)

t2 = id3 \* t1

t3 = id2 + t2

id1 = t3

**Fase Geração de Código**

**LDF R2, id3**

**MULF R2, R2, #60.0**

**LDF R1, id2**

**ADDF R1, R1, R2**

**STF id1, R1**

t1 = id3 \* 60.0

id1 = id2 + t1

**Fase Gerenciamento da Tabela de Símbolos / Manipulação de Erros**

# (**violação de tamanho de memória**: **erro de geração de código de máquina**)

int **main**() {

int i, a [100000000000000000000];

float j@; # (**violação de formação de identificador**: **erro léxico**)

i = “7”; # (**violação de significado**: **erro semântico**)

while (i < 3 # (**violação de formação de comando**: **erro sintático**)

**printf**(“%d \n”, i);

k = i; # (**violação de identificadores conhecidos**: **erro contexto semântico**)

**return** (0);

}

**Máquina de Turing**

**MT** = (Q, ∑, I, q1, F) ;

**Eventos**: Estado -> Valor Lido -> Valor Escrito -> Movimento -> Próximo Estado

**Exemplo1**: Escrever o símbolo “1” após uma cadeia de símbolos “1” e rebobinar o cabeçote de leitura para o primeiro “1” novamente.

**FITA (**configuração da fita**)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ... | 0 | 0 | 1 | 1 | .. | 1 | 1 | 0 | 0 | 0 | ... |

**q1**

**Estados** => Q = {q1, q2, q3}

**Alfabeto** => ∑ = {0, 1}

**Instruções** => I = {i1, i2, i3, i4} = {(q1, 1, Dir, q1), ..., (q2, 0, Dir, q3)}

**Estado Inicial:** q1

**Estado Final:** F **= {**q3}

**INSTRUÇÕES**

**i1 =** (q1, 1, Dir, q1)

**i2 =** (q1, 0, 1, q2)

**i3 =** (q2, 1, Esq, q2)

**i4 =** (q2, 0, Dir, q3)

**Resultado Final**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ... | 0 | 0 | 1 | 1 | .. | 1 | 1 | 1 | 0 | 0 | ... |

**q3**

**Exemplo2**: Loop infinito.

**FITA (**configuração da fita**)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ... | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | ... |

**q1**

**INSTRUÇÕES**

**i1 =** (q1, 1, 0, q2)

**i2 =** (q2, 0, 1, q1)