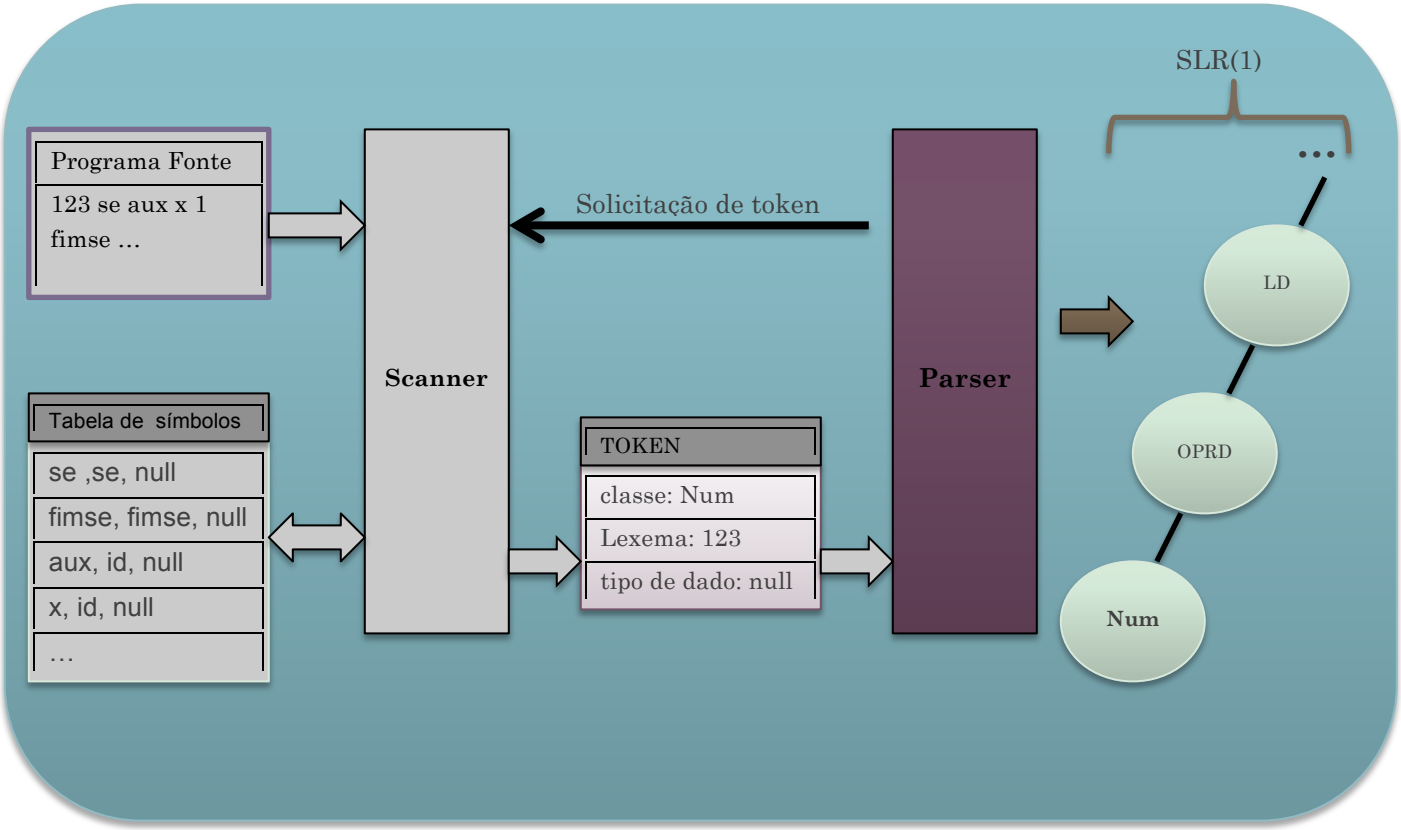
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

COMPILADORES – TRABALHO 2

Analisador Sintático e Tratamento e recuperação do Erro no parser



Profa. Dra. Deborah Fernandes
2021

1 . Descrição

A atividade prática Trabalho 2 (T2) – Analisador Sintático em Compiladores é um componente para a avaliação e desenvolvimento dos conhecimentos desenvolvidos nas disciplinas ofertadas para Ciência da Computação e Engenharia de Computação - Compiladores e Compiladores 1. O valor dessa atividade é 10,0 e compõe a média de aprovação na disciplina conforme definido no plano de curso.

2 – Entregáveis

* Todas as atividades e código derverão ser entregues EXCLUSIVAMENTE via plataforma Turing.
2.1 – (Atividade 1 Complementar do T2 – T2.1) - Conjuntos *First* e *Follow* dos não terminais da gramática do T2. A atividade é INDIVIDUAL e vale 0,5 na nota final do trabalho T2. Não será computada nota para atividade entregue após a data determinada.

2.2 – (Atividade 2 Complementar do T2 – T2.2) - AUTÔMATO LR(0) com itens da gramática do T2. Atividade INDIVIDUAL e vale 1,0 na nota final do trabalho T2. Não será computada nota para atividade entregue após a data determinada.

2.3 – (Atividade 3 Complementar do T2 – T2.3) - Pesquisa e escrita sobre tratamento de erros no analisador sintático ascendente. Atividade INDIVIDUAL, vale 0,5 na nota final do trabalho T2. Não será computada nota de atividade entregue após a data determinada.

2.4 – T2 – Código da implementação do T2: A ser desenvolvido conforme as definições descritas nas próxima seções. Caso seja realizado em duplas, apenas um componente deverá entregá-lo na plataforma conforme padrão de nome definido no documento de regras gerais. á. **Exemplo**: T2-DeborahFernandes-Fulano.c .

- Se for entregar um projeto com vários arquivos, junte-os em uma pasta com o nome conforme o padrão T2-NomeAluno1-NomeAluno2 e inclua dentro da pasta um arquivo .txt explicando como abrir e rodar os códigos do programa.

3 – Notas

3.1 – A entrega e arguição oral terão o valor total de 8,0 pontos.

3.2 – Nota total = Nota T2.1 + Nota T2.2 + Nota T2.3 + Nota T2.

4 – O que fazer?

O programa a ser desenvolvido deverá estar de acordo com as definições de projeto descritas abaixo e será avaliado pelo professor em relação a cada critério estabelecido. Portanto, leia com atenção.

Desenvolver um programa computacional na linguagem escolhida para o projeto que, acoplado ao T1 (analisador léxico), implemente:

4.1 Um analisador sintático SLR(1) que reconheça as sentenças que podem ser formadas a partir da gramática livre de contexto disponível na TABELA 1.

4.2 Passos de projeto (itens realizados em 3.1 e 3.2):

4.2.1 Construir o autômato LR(0) para a gramática livre de contexto da TABELA 1 (item 2.2);

4.2.2 Obter os conjuntos FIRST/FOLLOW dos não terminais da gramática (item 2.1);

4.2.3 Construir a tabela de análise sintática **SLR** com as colunas AÇÃO (*shift, reduce, accept e error*) e DESVIOS (*goto*), baseadas nos itens 4.2.1 e 4.2.2. À critério do programador, pode ser criada uma ou duas tabelas (uma para ações –ACTION- e outra para os desvios - GOTO).

i. A tabela pode ser construída em um arquivo .csv. O upload pode ser realizado em uma matriz, estrutura de dados, map, à critério do programador, ou poderá ser construída diretamente em uma estrutura no programa.

ii. As lacunas da tabela sintática – coluna AÇÕES (espaços sem ações de redução/empilhamento/aceita) devem ser preenchidas com códigos de erros que deverão indicar o tipo de erro sintático encontrado (se falta operador aritmético, relacional, atribuição, aguarda um id, um se, um (, etc.).

4.3 Implementar o algoritmo de análise sintática *shift-reduce* da FIGURA 1 - PARSER.

4.3.1 Uma estrutura de dados do tipo pilha deverá ser criada para apoiar o reconhecimento da sentença(implementação do autômato de pilha). Ela é inicializada com o estado 0 (estado inicial do autômato LR) ao topo. As operações de empilhamento e desempilhamento apontadas no algoritmo serão realizadas sobre esta pilha.

4.3.2 No algoritmo de análise, todas as vezes em que houver um movimento com o apontador de entrada ***a***, o programa deverá chamar a função **"SCANNER"** do trabalho T1 que retornará um TOKEN e seus atributos em ***a***. O campo de a que será utilizado na análise é a "classe".

4.3.3 Todas as vezes que for acionada uma consulta ACTION ou GOTO, a(s) tabela(s) desenvolvida(s) no item 4.2.3 deverá ser consultada.

4.3.4 Imprimir a produção significa apresentar na saída (tela do computador) a regra que foi reduzida.

4.3.5 Ao invocar uma rotina de **recuperação de ERRO (item 4.4 abaixo),** além de reestabelecer a análise sintática, esta deverá imprimir uma mensagem na saída (tela do computador) informando o

tipo do erro sintático encontrado a linha e a coluna onde ocorreu no fonte (linha e coluna podem ser obtidos pelo léxico – scanner) .

4.4 Implementar uma rotina de tratamento ou recuperação do Erro.

4.4.1 Conforme a pesquisa realizada em T2.3 (item 2.3), escolher e implementar um modelo (modo pânico ou outro ou a junção de dois modelos);

4.4.2 Ao encontrar um erro, o PARSER emite mensagem conforme item 4.3.5, reestabelece a análise conforme 4.4.1 e continua o processo de análise para todo o restante do código fonte.

4.5 O PARSER invocará :

4.5.1 O SCANNER nas linhas (1) e (6) do algoritmo de análise na FIGURA1;

4.5.2 Uma rotina que emitirá o tipo do erro sintático encontrado (mensagem na tela informando que houve erro sintático e qual terminal era aguardado para leitura, linha e coluna onde ocorreu o erro), linha (13) do algoritmo de análise na FIGURA1;

4.5.3 Uma rotina que fará uma recuperação do erro (modo pânico ou outro) para continuar a análise sintática até que o final do fonte seja alcançado, linha (13) do algoritmo de análise na FIGURA1.

Algoritmo de análise

```
(1)  Seja a o primeiro símbolo de w$;
(2)  while {  /*Repita indefinidamente*/
(3)  seja s o estado no topo da pilha;
(4)       if (ACTION [s,a] = shift t ) {
(5)                 empilha t na pilha;
(6)                 seja a o próximo símbolo da entrada;
(7)       }else if (ACTION [s,a] = reduce A-> β )  {
(8)                   desempilha símbolos | β | da pilha;
(9)                   faça o estado t agora ser o topo da pilha;
(10)                  empilhe GOTO[t,A] na pilha;
(11)                  imprima a produção A-> β ;
(12)      }else if (ACTION [s,a] = accept )  pare; /* a análise terminou*/
(13)      else  chame uma rotina de recuperação do erro;
```

FIGURA 1 – Algoritmo de análise sintática ascendente *shift-reduce.*

TABELA 1 – Produções da gramática livre de context para o Trabalho 2.

Identificação	Regra gramatical
1	P' → P
2	P→ **inicio** V A
3	V→ **varinicio** LV
4	LV→ D LV
5	LV→ **varfim** pt_v
6	D→ TIPO L pt_v
7	L→ id vir L
8	L→ id
9	TIPO→ **inteiro**
10	TIPO→ **real**
11	TIPO→ **literal**
12	A→ ES A
13	ES→ **leia** id pt_v
14	ES→ **escreva** ARG pt_v
15	ARG→ lit
16	ARG→ num
17	ARG→ id
18	A→ CMD A
19	CMD→ id rcb LD pt_v
20	LD→ OPRD opm OPRD
21	LD→ OPRD
22	OPRD→ id
23	OPRD→ num
24	A→ COND A
25	COND→ CAB CP
26	CAB→ **se** ab_p EXP_R fc_p **então**
27	EXP_R→ OPRD opr OPRD
28	CP→ ES CP
29	CP→ CMD CP
30	CP→ COND CP
31	CP→ **fimse**
32	A→ R A
33	R → **repita** ab_p EXP_R fc_p CP_R
34	CP_R→ ES CP_R
35	CP_R→ CMD CP_R
36	CP_R→ COND CP_R
37	CP_R→ **fimrepita**
38	A→ **fim**

5 – Resultado final do Parser

O Parser (FIGURA 2) realizará o processo de análise sintática:

- invocando o SCANNER (T1), sempre que necessitar de um novo TOKEN;
- inserindo e removendo o topo da pilha;
- consultando as tabelas ACTION e GOTO para decidir sobre as produções a serem aplicadas até a raiz da árvore sintática seja alcançada e não haja mais tokens a serem reconhecidos pelo SCANNER;
- Mostrando na tela os erros cometidos, bem como sua localização do fonte (linha, coluna);
- Reestabelecendo a análise para que o restante do código fonte seja analisado.

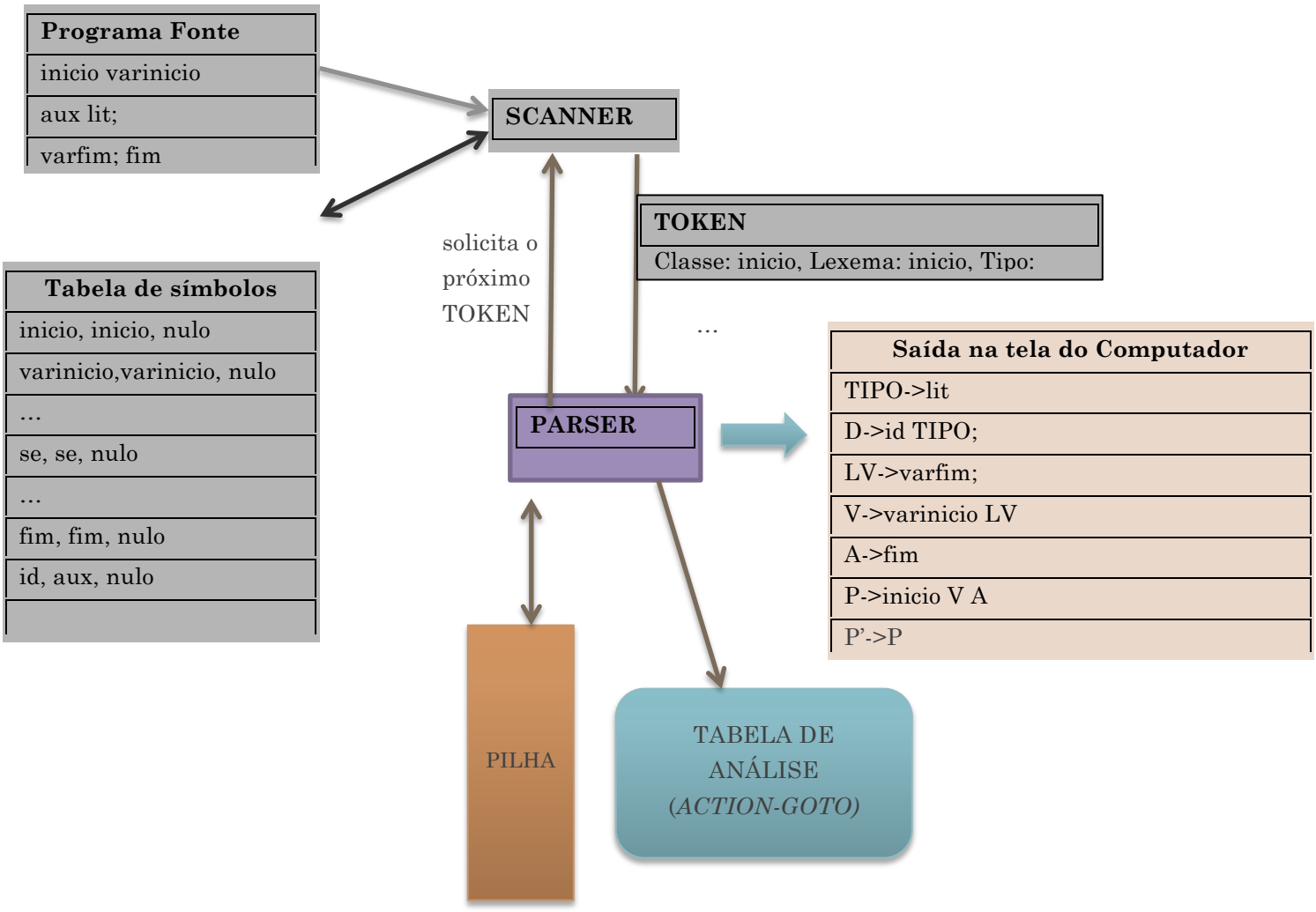


FIGURA 2 – Resultado do PARSER.