Universidade Federal Fluminense
Instituto de Computação
TCC00305 Compiladores
Prof.: Christiano Braga
Niterói, 20 de julho de 2021
Pedro Paulo Bastos Teixeira 615031106

Relatório de Execução – SLR versus LALR

1. Descrição do Problema

Quando trata-se do parsing ascendente, tem-se , para que seja feito o parsing de uma string de entrada sob uma gramática livre de contexto, duas operações, são elas, *Shift* e *Reduce* (Empilha e Reduz, respectivamente) , onde uma pilha guarda os símbolos gramaticais correntes, e a redução, substitui a produção na pilha, pelo símbolo que a gerou, reduzindo para a mesma, como mostrado na figura 1.

STACK	INPUT	ACTION
$	$\mathbf{id}_1 * \mathbf{id}_2$ $	shift
$ $\mathbf{id}_1$	$* \mathbf{id}_2$ $	reduce by $F \to \mathbf{id}$
$ F	$* \mathbf{id}_2$ $	reduce by $T \to F$
$ T	$* \mathbf{id}_2$ $	shift
$ $T *$	$\mathbf{id}_2$ $	shift
$ $T * \mathbf{id}_2$	$	reduce by $F \to \mathbf{id}$
$ $T * F$	$	reduce by $T \to T * F$
$ T	$	reduce by $E \to T$
$ E	$	accept

Figure 1 Ações Shift/Reduce, Entrada e Pilha durante um processo de Parsing Ascendente

Durante o processo de parsing ascendente, as decisões de executar um Shift ou um Reduce, dependerá dos itens, isto é, do estado do autômato de itens do parser, gerado para uma determinada gramática em questão. Ocorre que durante este processo, a depender da gramática e do autômato, isto é, da abordagem em que se dará este processo de decisão entre ações Shift e Reduce, podem ocorrer conflitos .(estes conflitos podem ser Shift-Reduce , que é quando o analisador não consegue decidir entre alguma(s) ações de Shift e uma Reduce ; e podem ser Reduce-Reduce que é quando o analisador fica impossibilitado de decidir entre ações Reduce.) A priori, esse conflito depende da gramática quando a gramática é ambígua, entretanto, pode ocorrer da gramática em questão não ser ambígua, mas ser de nível maior que o Autômato/Parser/abordagem em questão, isto é, o Autômato/Parser/abordagem utilizado para realizar o processo , possuir limitações perante a gramática, uma vez que existem diversos níveis entre gramáticas livres de contexto , no âmbito do parsing ascendente, conforme ilustrado pela figura 2 , e neste caso ocorre algum(ns) conflito(s) durante o parsing.

O caso proposto, trata-se da gramática $\mathbb{G}$ mostrada a seguir na equação 1, esta gramática, objeto desta análse, representa a atribuição de variáveis e/ou ponteiros em C, que é uma das partes essenciais da linguagem . Esta gramática sofrerá um processo de

parsing ascendente, será gerado um parser para esta gramática, e se testará uma string de exemplo pertencente à esta gramática, como mostrado no algoritmo 1.

$$\mathbb{G} = (\{S, L, R\}, \{\boldsymbol{id}, =, *\}, \{\ S \to L = R \mid R \\ L \to * R \mid \boldsymbol{id} \\ R \to L \\ \}, S)$$

Equation 1 Gramática Proposta

```
var1 = var2
var1
var2 =  * var3
*var3
```

Algorithm 1 Exemplos de strings pertencentes à gramática

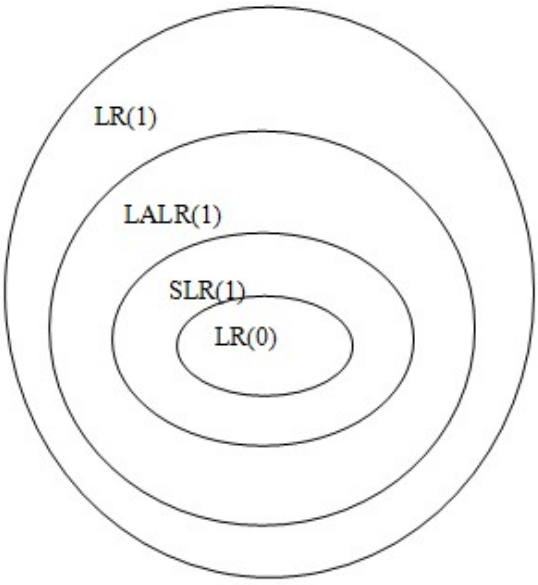


Figure 2 Hierarquia de Gramáticas Livres de Contexto

O objetivo deste experimento é demonstrar que esta GLC, embora não seja ambígua, não é SLR, e sim LALR, ou seja, na hierarquia de GLCs , esta gramática não consegue ser parseada por parsers SLR.

2. Desenvolvimento

Para a execução desta tarefa, foi utilizada a biblioteca PyLEX para Python 3.7 , que possui ferramentas léxicas e sintáticas, úteis aqui. Primeiramente foi implementado o analisador léxico desta gramática, nele, foram definidos três tipos de lexemas : { `ATRIBUICAO`, `PONTEIRO`, `ID` } o primeiro, demarca o operador $=$; o segundo, demarca o operador $*$; e o último marca os identificadores, presente na gramática como $\boldsymbol{id}$, este último, denota o identificador, que assume a forma mostrada pela expressão regular $\boldsymbol{id}$, mostrada na equação 2 e implementada em `exprelex.py`, anexo à esta atividade .

$$\boldsymbol{id} = \Pi\Phi^*$$
$$\Pi = \mathrm{a} + \mathrm{b} + \mathrm{c} + \cdots + \mathrm{z} + \mathrm{A} + \mathrm{B} + \mathrm{C} + \cdots + \mathrm{Z}$$
$$\Phi = \Pi + 0 + 1 + \cdots + 9 + \_$$

Equation 2 Expressão Regular que denota o identificador “id”

Já o analisador sintático nada mais é do que a mesma gramática supracitada, porém incluindo algumas ações semânticas, apenas para monitoramento da execução em tela, mostrada na equação 3 , e implementada em `slrvslalr.py`, anexo à esta atividade

$$S \rightarrow L = R \ \{\ \texttt{print(}L\texttt{.value , '←' , }R\texttt{.value)}\ \}$$
$$S \rightarrow R \ \{\ \texttt{print(}R\texttt{.value)}\ \}$$
$$L \rightarrow *R \ \{\ L\texttt{.value= '[*ptr]' + }R\texttt{.value}\ \}$$
$$L \rightarrow \boldsymbol{id} \ \{\ L\texttt{.value = get\_lexval(}\boldsymbol{id}\texttt{)}\ \}$$
$$R \rightarrow L\{\ R\texttt{.value = }L\texttt{.value}\ \}$$

Equation 3 Gramática com ações semânticas

Além disso, foi adicionado um trecho de código neste mesmo fonte, onde o usuário pode optar por executar o parse pelo YACC do PLY, tanto pelo parser SLR ou pelo parser LALR, e logo após se deve inserir trechos de código que pertençam a linguagem para execução do parse.

3. Evidenciamento do caso pela ferramenta PLY

Em seguida, foi feita a execução da gramática em dois casos, SLR, e LALR, usando como strings/trechos de código de teste para parsear, os mesmos exemplos em ambos os casos. Os outputs respectivos seguem nas imagens 3 e 4 abaixo.

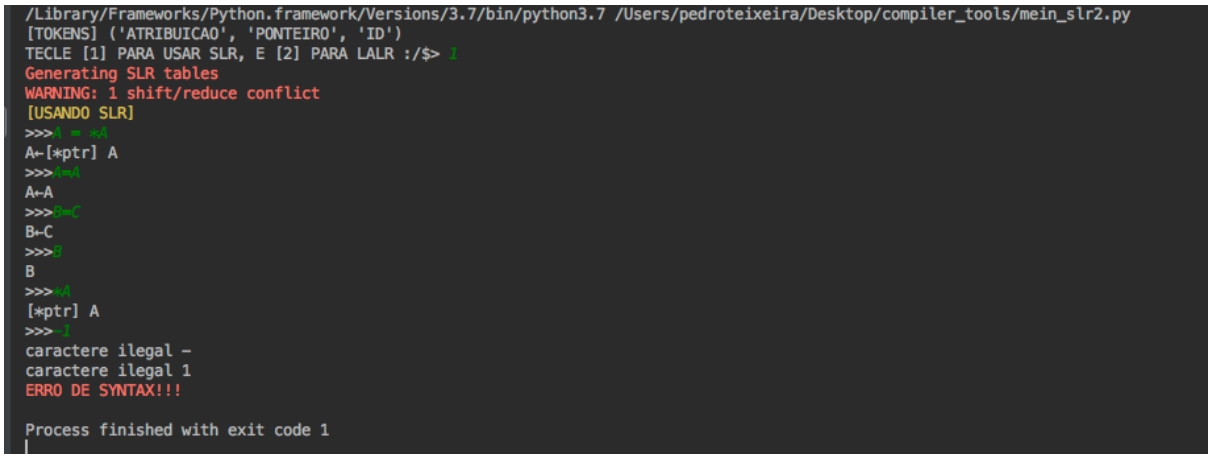


Figure 3 Parse SLR da Gramática

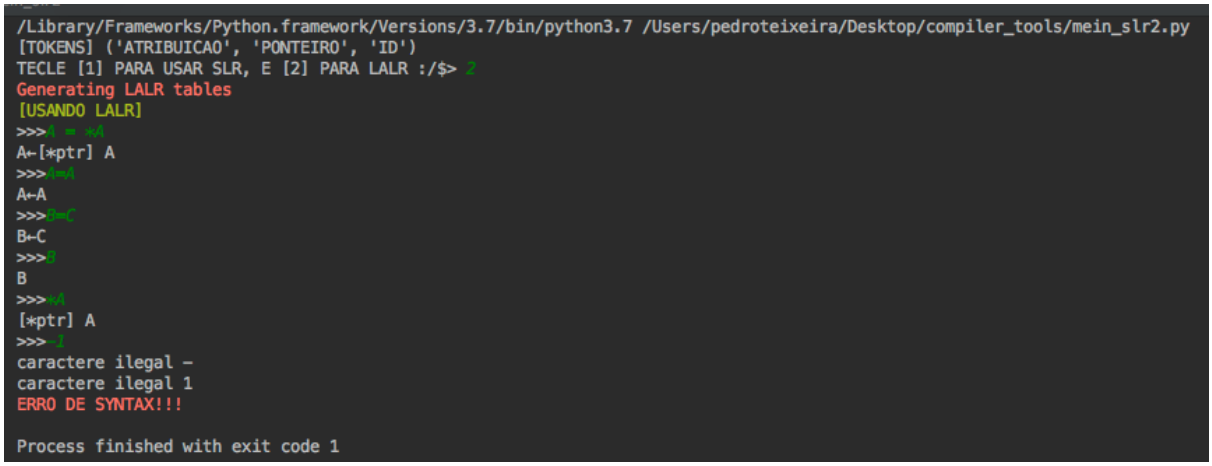


Figure 4 Parse LALR da Gramática

É notório que antes da execução do primeiro caso (SLR) no instante em que o YACC constrói o parser SLR para a gramática em questão, ele lança um aviso na dela, que diz que ele detectou algum estado onde ocorre um conflito Shift/Reduce, que não acontece no segundo caso (LALR).

O YACC, neste mesmo instante em que constrói os parsers, gera no mesmo diretório em que o fonte é executado, um output para cada caso (SLR e LALR), estes outputs (`slr_.out` e `lalr_.out` respectivamente) (presentes em anexos nesta atividade) contém a representação textual direta dos autômatos de cada parser, a partir dessa representação, os mesmos foram reproduzidos de forma gráfica (vide anexos I (SLR) e II (LALR)), e tanto na representação gráfica, quanto no output, no caso LALR, não foi detectado pelo YACC nenhum estado conflituoso, já no caso do SLR, o estado 2 (ou $\mathbb{I}_2$ na representação gráfica) foi detectado um conflito Shift/Reduce , para o símbolo “ATRIBUICAO” (representado na gramática por “ = “) . Onde para o mesmo símbolo, o parser SLR não consegue decidir entre fazer um Shift e transitar para o estado 6 (ou $\mathbb{I}_6$ na representação gráfica) ou fazer um

Reduce usando a regra $R \rightarrow L \cdot$. Ainda que o parser SLR do YACC trate o conflito para Shift, e segue em frente com a execução do parsing, realizando-o com sucesso nos exemplos supracitados.

```
state 2

    (1) S -> L . ATRIBUICAO R
    (5) R -> L .

  ! shift/reduce conflict for ATRIBUICAO resolved as shift
    ATRIBUICAO      shift and go to state 6
    $end            reduce using rule 5 (R -> L .)

  ! ATRIBUICAO      [ reduce using rule 5 (R -> L .) ]
```

Figure 5 Estado 2 SLR - Conflitos !

O que corrobora para a tese de que esta gramática não é SLR, e sim LALR.

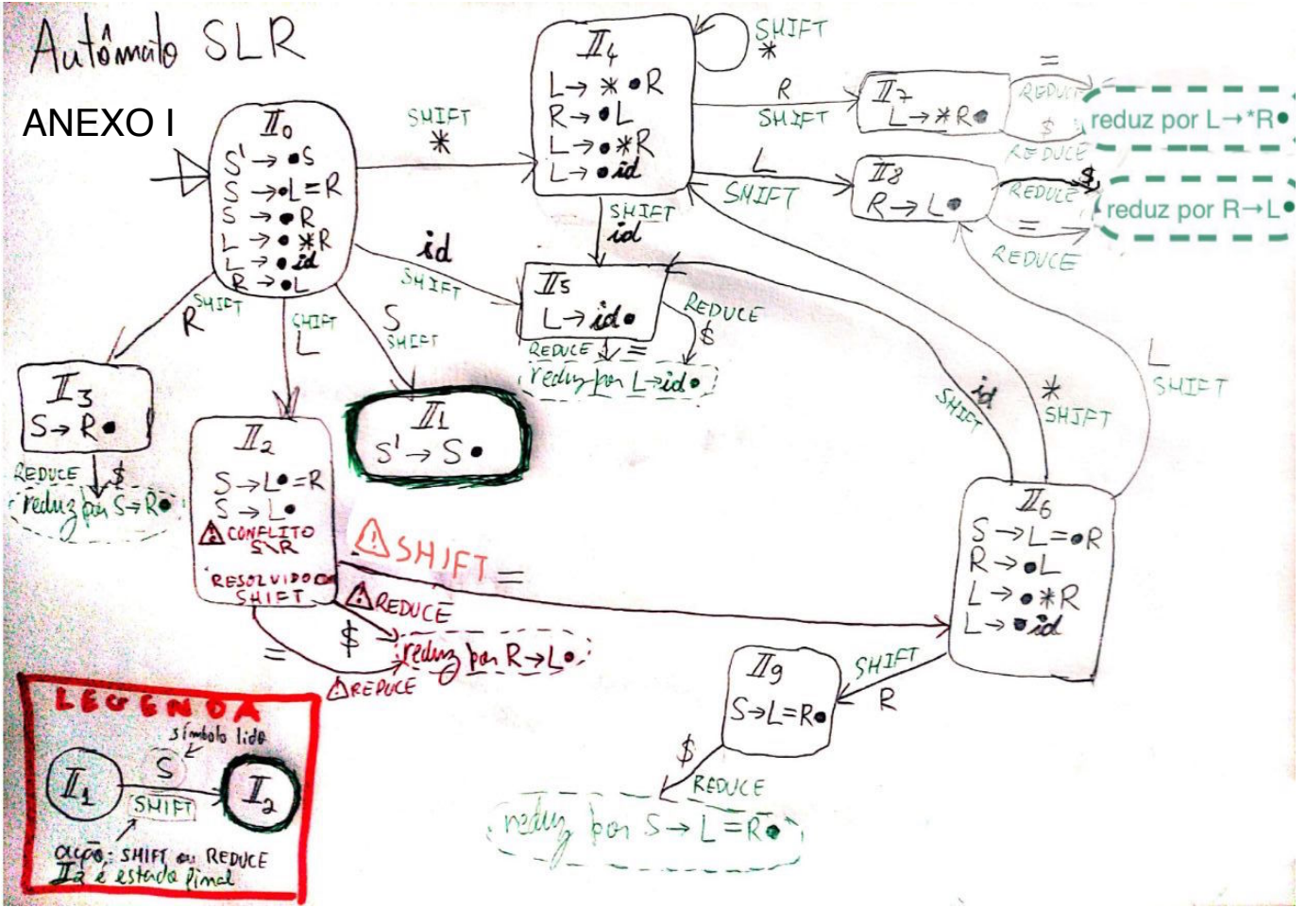
4. Conclusão

A ferramenta YACC do pyLEX , ao detectar conflitos em um estado do automato SLR, e ao realizar o parsing normalmente por LALR, comprova a tese de que a gramática $\mathbb{G}$ é LALR , mas não SLR .

5. Anexos

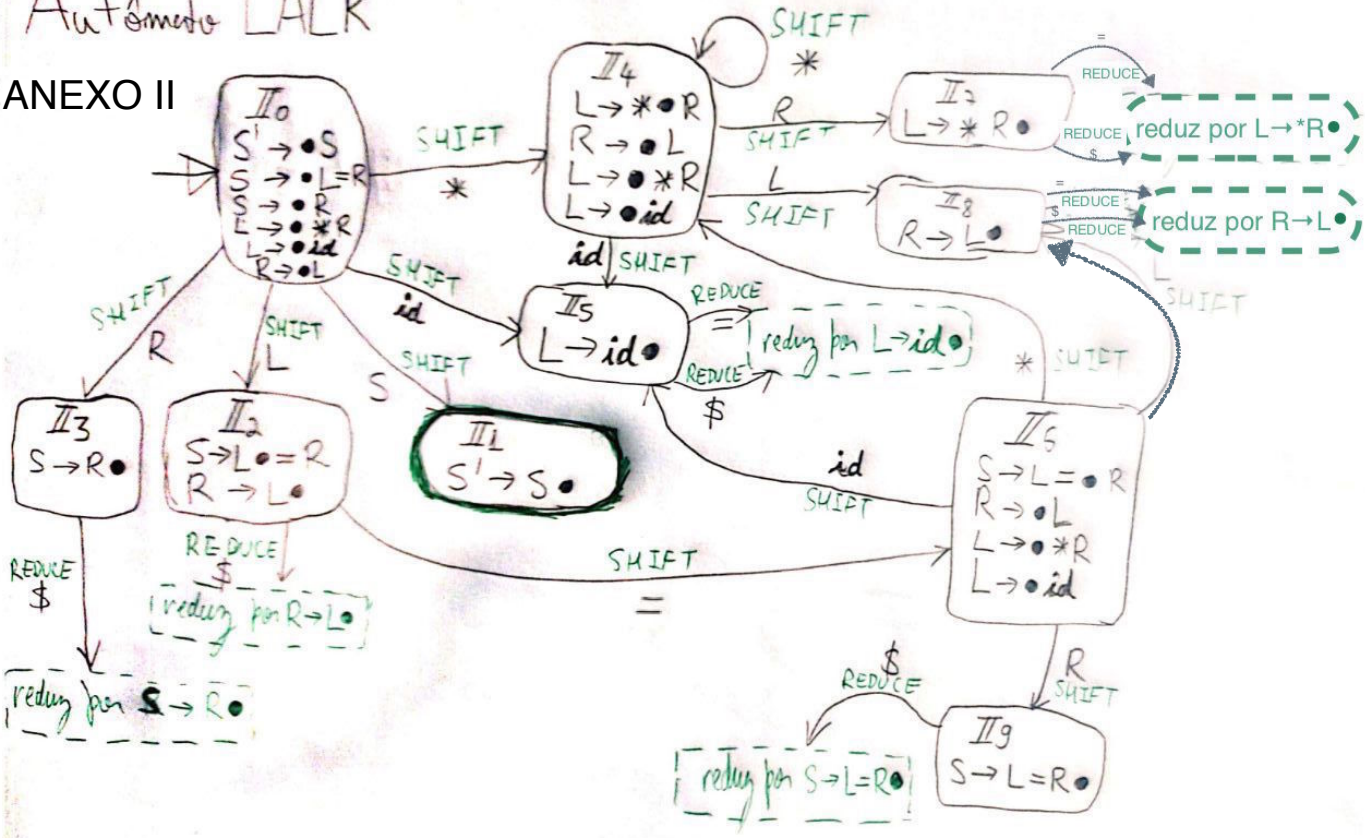
Autômato SLR

ANEXO I



Autômato LALR

ANEXO II



6. Referências

AHO, Alfred V; LAM, Monica s; SETHI, Ravi; ULLMAN, Jeffrey. **Compilers**: principles, techniques & tools. 2. ed. Boston, USA: Pearson Addison Wesley, 2007. 1038 p.

BEAZLEY, David M. **PLY (Python Lex-Yacc)**. 2021. Salt Lake City, USA. Disponível em: https://www.dabeaz.com/ply/ply.html. Acesso em: 20 jul. 2021.

BRAGA, Christiano. **Compiladores**: parsing ascendente - notas de aula. Niterói, Brasil: Universidade Federal Fluminense, 2021. 22 slides, color.

JUN, Yong. **LR(0),SLR(1),□范LR(1),LALR(1)**的关系. 2021. CN Blogs, Pequim, China. Disponível em: https://www.cnblogs.com/yongzhewudi/p/6048301.html. Acesso em: 20 jul. 2021.

MASCARENHAS, Fábio. **Compiladores - Parsing Ascendente**: parte 2. Rio de Janeiro, Brasil: Universidade Federal do Rio de Janeiro, 2015. 18 slides, color. Disponível em: https://www.dcc.ufrj.br/~fabiom/comp20152/10AscendentePt2.pdf. Acesso em: 20 jul. 2021.