

Analizador Sintático

Diogo Rodrigues dos Santos¹

¹Universidade Tecnológica Federal do Paraná - UTFPR

²Departamento de Computação - DACOM

Resumo. Criação de um analisador sintático utilizando a biblioteca automata da linguagem de programação python. Os testes executados no programa foram fornecidos pelo professor da disciplina Linguagens Formais, Autômatos e Computabilidade. O Projeto encontra-se disponível no repositório <https://github.com/DiogoRodrigues/PARSER> do GitHub.

1. Informações Gerais

O analisador sintático, recebe um arquivo '.exp' por parâmetro e imprime no terminal a condição de aceitação ou rejeição da expressão matemática presente no arquivo. Os símbolos aceitos nessa gramática são: *, /, +, -, (,), id, num. A gramática utilizada no processo inicial de criação pode ser observada na seção 2.

2. Gramática

Para a criação da tabela shift-reduce, é necessário uma gramática que identifique o que o analisador sintático irá reconhecer. Para esse projeto, foi utilizada uma gramática que reconheça expressões matemáticas. A FIGURA 1 mostra a tabela de produções utilizada.

S	→	E
S	→	V=E
E	→	E+T
T	→	T*F
F	→	(E)
E	→	E-T
E	→	T
T	→	T/F
T	→	F
F	→	V
V	→	id
F	→	num

Figura 1. Tabela de Produções

3. Tabela shift-reduce

Utilizando a tabela de produções da Figura 1 na função grammar do JFLAP, obtemos a tabela shift-reduce da Figura 2.

	()	*	+	-	/	=	d	i	m	n	u	\$	E	F	S	T	V
0	s1								s7		s8			2	3	4	5	6
1	s1								s7		s8			9	3		5	10
2				s11	s12								r1					
3		r9	r9	r9	r9	r9							r9					
4													acc					
5		r7	s13	r7	r7	s14							r7					
6		r10	r10	r10	r10	r10	s15						r10					
7								s16										
8												s17						
9		s18		s11	s12			Reject										
10		r10	r10	r10	r10	r10							r10					
11	s1								s7		s8			3			19	10
12	s1								s7		s8			3			20	10
13	s1								s7		s8			21				10
14	s1								s7		s8			22				10
15	s1								s7		s8			23	3		5	10
16		r11	r11	r11	r11	r11	r11						r11					
17									s24									
18		r5	r5	r5	r5	r5							r5					
19		r3	s13	r3	r3	s14							r3					
20		r6	s13	r6	r6	s14							r6					
21		r4	r4	r4	r4	r4							r4					
22		r8	r8	r8	r8	r8							r8					
23				s11	s12								r2					
24		r12	r12	r12	r12	r12							r12					

Figura 2. Tabela shift-reduce

4. Automato Gerado

Ao utilizar a gramática da Figura 2 obtemos o seguinte automato representado na Figura 3.

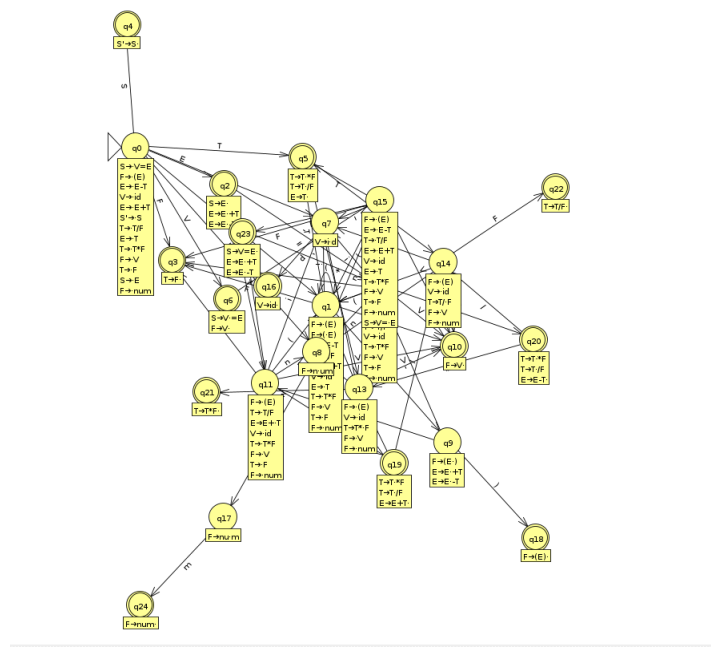


Figura 3. Automato Gerado

5. Como Executar o Projeto

Com o terminal aberto na pasta implementação, digite o comando 'python Analisador-Sintatico.py ;caminho-arquivo-teste;', substituindo ';caminho-arquivo-teste' pelo cami-

no onde o teste a ser realizado se encontra. A Figura 4 é um exemplo de execução e a Figura 5 é um exemplo de saída.

```
diogo@ubuntu:~/Documentos/FACULDADE/4º Período/AUTOMATOS/PROJETO_2_ANALISADOR_SINTATICO/implementaçã  
o$ python AnalisadorSintatico.py ./testes-exp/teste-002.exp
```

Figura 4. Teste de Arquivo Unitário

```
diogo@ubuntu:~/Documentos/FACULDADE/4º Período/AUTOMATOS/PROJETO_2_ANALISADOR_SINTATICO/implementaçã  
o$ python AnalisadorSintatico.py ./testes-exp/teste-002.exp  
Expressão id=id*id+num: Aceita!
```

Figura 5. Resultado Teste Unitario

Também é possível executar todos os testes disponibilizados, através do comando **make**. Esse comando deve ser utilizado no terminal, dentro da pasta **implementação**. A Figura 6 é um exemplo de execução e a Figura 7 é um exemplo de saída.

```
diogo@ubuntu:~/Documentos/FACULDADE/4º Período/AUTOMATOS/PROJETO_2_ANALISADOR_SINTATICO$ make
```

Figura 6. Executando Comando Make

```
diogo@ubuntu:~/Documentos/FACULDADE/4º Período/AUTOMATOS/PROJETO_2_ANALISADOR_SINTATICO/implementaçã  
o$ make  
python AnalisadorSintatico.py ./testes-exp/teste-001.exp  
Expressão num: Aceita!  
  
python AnalisadorSintatico.py ./testes-exp/teste-002.exp  
Expressão id=id*id+num: Aceita!  
  
python AnalisadorSintatico.py ./testes-exp/teste-003.exp  
Expressão (num+num)*id: Aceita!
```

Figura 7. Resultado do Comando Make

6. Exemplos de Entrada e Saída

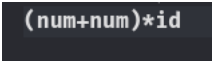
As Figuras 8,9 e 10 são alguns exemplos de possíveis entradas.

```
num
```

Figura 8. Entrada 1

```
id=id*id+num
```

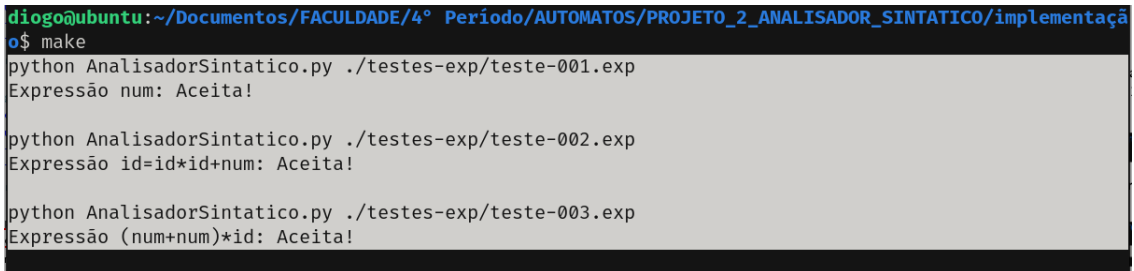
Figura 9. Entrada 2



```
(num+num)*id
```

Figura 10. Entrada 3

A Figura 11 mostra os resultados das entradas 1, 2 e 3.



```
diogo@ubuntu:~/Documentos/FACULDADE/4º Período/AUTOMATOS/PROJETO_2_ANALISADOR_SINTATICO/implementaçã
o$ make
python AnalisadorSintatico.py ./testes-exp/teste-001.exp
Expressão num: Aceita!

python AnalisadorSintatico.py ./testes-exp/teste-002.exp
Expressão id=id*id+num: Aceita!

python AnalisadorSintatico.py ./testes-exp/teste-003.exp
Expressão (num+num)*id: Aceita!
```

Figura 11. Exemplos de Saídas

7. Conclusão

Para uma expressão verdadeira, podemos torná-la uma subexpressão e substituí-la por algo simbólico que represente uma expressão que já foi aceita anteriormente. Um vez que essa expressão é validada, não existe a necessidade de verificarmos toda ela novamente.