Processamento de Linguagens

MiEI - 3º ano
2020 / 2021

A notícia do dia (31 de março de 2021):

"Turing Award goes to researchers who made programming easier and more powerful !
Alfred Aho and **Jeffrey Ullman** win the 2020 Turing Award for *pioneering compiler and algorithm work*."

https://www.cnet.com/news/turing-award-goes-to-researchers-who-made-programming-easier-and-more-powerful/

DOSSIER DA UC (DUC)

A) Objetivos do Ensino

Os objectivos fundamentais deste curso são: 1. Apresentar a noção de linguagem formal e caracterizar o conceito de processamento de linguagens, estabelecendo os objectivos, as tarefas de um processador, as exigências que lhe são impostas e as técnicas de desenvolvimento (face às restrições e à complexidade do problema). 2. Apresentar os conceitos de expressão regular de gramática e de autómato e a sua aplicação ao desenvolvimento de Processadores de Linguagens (PL's). 3. Introduzir formalismos para especificação da sintaxe e da semântica das linguagens e estudar a implementação dos analisadores que se podem derivar desses formalismos. 4. Tornar os alunos aptos a desenvolver Processadores de Linguagens segundo os métodos da Tradução Dirigida pela Sintaxe (TDS), recorrendo ao uso de ferramentas para geração automática dos ditos Processadores.

B) Resultados de Aprendizagem

- Trabalhar com métodos, técnicas e ferramentas para a especificação formal de linguagens e para a
construção automática de processadores de linguagens;
- Compreender as tarefas de processamento de linguagens, os algoritmos associados
e as estruturadas de dados necessárias para a construção de um compilador;
- Desenvolver processadores para linguagens de domínio

específico e linguagens de programação simples.

C) Programa detalhado de Processamento de Linguagens e Compiladores:

I. Introdução ao Processamento de Linguagens:
I.1 Conceitos básicos: Linguagem; Gramáticas - Gramáticas Independentes, Gramáticas Regulares e Contexto e Gramáticas Tradutoras.
I.2 Caracterização de Processador de Linguagens: objetivos; tarefas - análise (ou reconhecimento) e síntese (geração de código); requisitos de eficiência
I.3 Processamento de Texto (Filtragem) com base em Regras de Produção (regras Condição-Ação) recorrendo a Expressões Regulares.
I.4 Reconhecimento e Tradução de Linguagens de Programação ou similares recorrendo a Gramáticas Tradutoras

II. Análise Léxica especificada via Expressões Regulares (ER)
II.1 Expressões Regulares: conceitos, definições e aplicações
II.2 Autómatos Deterministas, Não-deterministas e Reativos;
Algoritmo de Reconhecimento guiado por um AD;
Conversões de ERs em ANDs e de ANDs em ADs
II.3 Uso do Gerador FLex

III. Análise Sintáctica especificada via Gramáticas Independentes de Contexto (GIC):
III.1 Parsing Top-Down: Recursivo-descendente (RD) e LL(1)
III.2 Parsing Bottom-Up: LR(0), SLR(1), LALR(1)

IV. Análise Semântica e Transformação (reação) especificada via Gramáticas Tradutoras (GT) ---Tradução Dirigida pela Sintaxe:
IV.1 Uso do Gerador Yacc.
IV.2 Geração de Código Assembly: Máquinas Virtuais.

D) Bibliografia

- Pedro Reis Santos & Thibault Langlois, "Compiladores – Da Teoria à Prática", FCA, http://www.fca.pt/cgi-bin/fca_main.cgi/?op=2&isbn=978-972-722-768-6, 2014.
- R. G. Crespo, "Processadores de Linguagens: da concepção à implementação", IST-Press, 1998;
- Aho & Sethi & Ullman, "Compiler Principles, Techniques and Tools", Addison-Wesley, 1986;
- William Waite & Carter, "An Introduction to Compiler Construction", HarperCollin College Publishers, 1993;
- Dick Grune & Kees van Reeuwijk & Henri E. Bal & Ceriel J.H. Jacobs & Koen Langendoen, "Modern Compiler Design", 2nd. edition, Springer, 2012;

- Keith D. Cooper & Linda Torczon, "Engineering a Compiler", 2nd. edition, Elsevier/Morgan-Kaufmann, 2011;
- Steven S. Muchnick, "Advanced Compiler Design and Implementation", Morgan Kaufmann, 1997;
- Andrew W. Appel & Jens Palsberg, "Modern Compiler Implementation in Java", Cambridge University Press, 2002;
- Andrew W. Appel & Maia Ginsburg, "Modern Compiler Implementation in C", Cambridge University Press, 2004;
- Andrew W. Appel, "Modern Compiler Implementation in ML", Cambridge University Press, 1997;
- Pittman & Peters, "The Art of Compiler Design: theory and practice", Prentice-Hall, 1992;
- M353 Programming and Programming Languages, "Unit 13: Compiling -- volume I", The Open University, 1994;
- M353 Programming and Programming Languages, "Unit 14: Compiling -- volume II", The Open University, 1994;
- M353 Programming and Programming Languages, "Unit 15: Compiling -- volume III", The Open University, 1994;
- J.R. Levine & T. Mason & D. Brown, "Lex & Yacc", Ed. Dale Dougherty, O'Reilly & Associates Inc., 1992;
- M. E. Lesk and E. Schmidt, "Lex - A Lexical Analyzer Generator", Computing Science Technical Report No. 39, Bell Laboratories, Murray hill, New Jersey 1975;
- Stephen C. Johnson, "Yacc: Yet Another Compiler Compiler", Computing Science Technical Report No. 32, BellLaboratories, Murray hill, New Jersey 1975;
- Doug Brown, John Levine, Tony Mason, "Lex & Yacc", 2nd Edition, Unix Programming Tools, O'Reilly, 2012.

E) Metodologia de Ensino / Aprendizagem

Este curso é dividido em aulas teóricas e práticas.
Nas aulas teóricas, o assunto de processamento de linguagens é abordado apresentando as suas motivações, conceitos, definições, métodos e justificações.
Nas aulas práticas, são feitos exercícios de consolidação, em papel (especificação de casos práticos) e no computador,
usando ferramentas específicas para resolver os problemas propostos.
Os assuntos teórico-práticos lecionados nas aulas T e TP são aplicados a situações concretas (do mundo real) através da realização/desenvolvimento, fora das aulas, de 2 Trabalhos Práticos de média dimensão e complexidade.

F) Metodologia de Avaliação

A avaliação tem uma componente teórica e uma componente prática, ambas obrigatórias.

A nota prática (NP) será obtida através da realização de 2 trabalhos práticos (a resolver extra-aulas em grupos de 2 alunos)
que envolverão o desenvolvimento completo de um filtro de texto e de um processador para uma linguagem específica.

A nota teórica (NT) será obtida através da realização de: um teste (constituído por 4 minitestes rápidos a responder em tempo letivo em datas marcadas e afixadas no Bb); ou de um exame final em data a marcar pela DC.

As datas para entrega dos TPs (que serão feitos em grupos de até 3 alunos) serão acertadas com os alunos e afixadas no Bb;
as datas para apresentação dos TPs aos docentes e avaliação, serão posteriormente definidas com os alunos.
Os TPs são classificados de 0 a 20 e têm o mesmo peso na Nota Prática (NP) final.

A **nota mínima** a cada parte (T e TP) é de **8 valores** e todos os **miniTestes e TPs são obrigatórios**.

A Nota Final é dada pela fórmula: **NF = 50% * NT + 50% * NP**

Obs: *à nota final pode ser subtraído um DELTA calculado da seguinte forma:*
DELTA = se (NP - NT) >= 5 então (NP - NT) / 2 senão 0

****

Datas dos miniTestes de resposta múltipla:

T1 (ER1): 05.Mar
T2 (ER2): 26.Mar
T3 (GIC/GT1): 23.Abr
T4 (GIC/GT2): 21.Mai

EXAME Recurso :

Datas de publicação dos Trabalhos Práticos:

TP1 (ER/FT): entrega na Seg. 05.abril (só o PDF)

Datas de submissão dos Trabalhos Práticos:

TP2 (Gra/PL): entrega no Dom. 30.Mai

EQUIPA DOCENTE

- Pedro Rangel Henriques
  - prh@di.uminho.pt
  - www.di.uminho.pt/~prh
  - 968412287

- José Carlos Ramalho
  - jcr@di.uminho.pt
  - www.di.uminho.pt/~jcr

- Pedro Moura
  - pedrorpmoura@gmail.com
  - d12318@di.uminho.pt

LINKS importante / úteis para documentação de apoio

1. Expressões Regulares
   https://regex101.com/

2. Expressões Regulares em Python
   https://docs.python.org/3/library/re.html#regular-expression-syntax

3. Artigo muito bom sobre partições de string (**split**) em Python:
   https://note.nkmk.me/en/python-split-rsplit-splitlines-re/

4. https://www.tutorialspoint.com/automata_theory/constructing_fa_from_re.htm

5. Python : Ply Lex and Yacc
   https://www.dabeaz.com/ply/

6. Árvore de Sintaxe Abstrata
   https://www.101computing.net/abstract-syntax-tree-generator/

7. Graphviz / DOT online
   https://dreampuf.github.io/GraphvizOnline/

Aulas Teóricas

Semana 1 - de 15 a 19 de Fev.

T1: férias

T2: (142)

+ Apresentação da UC
+ Introdução às ER

Texto = seq(caracter) EOT/EOF

Vocabulário / Alfabeto = { simbolo / token }
Frase= seq(SIMBOLO) escrita de acordo com 1 conjunto de regras
Linguagem = { Frase }

Expressões Regulares

Dado um um vocabulário T
€ (sequencia vazia) é uma ER
t in T é uma ER
e1 . e2 concatenação é uma ER
e1 + e2 união é uma ER
e1 ^ n = e1 . e1 …. . e1 (e1 concatenado n vezes consigo próprio)
[0-9]{4}

e1 ^ + (fecho transitivo) U(n=1 .. inf) e1^n
e1 ^ * (fecho Kleene) U(n=0 .. inf) e1^n

e1^+ = e1 . e1^*

Semana 2 - de 22 a 26 de Fev.

T1 (130):

+ Introdução às ER

Exemplos e Propriedades algébricas do operadores.
Axiomas e manipulação; Simplificação

exemplos:
T = { a, b, c, d }

Concatenação e União (alternativa)

a.b
b.a
concatenação NÃO É comutativa
a.€
€.a = a
a | €
a ?
[+\-]? ('+' | '-') | €
a | b
b | a
união É comutativa
a | b | c
[abc]
[0-9a-z]
[.,;:_?\-!]
\. \\ \? \] \[\(\) \< \> \{ \} \+ * \|
[^0-9] [a-zA-Z] \"[^"]*\" \{[^}]*\} \<[^>]+\>
\a "a" 'a' [a] a "hello"
ele Ele ELE (e|E)(l|L)(e|E) [eE][lL][eE]
(?i:ele)
(a | b) c
a c | b c
a (b | c)
a b | a c
a | (b . c)
a b | a c (FALSO)

T2 (120):

+ Introdução às ER

Sistemas de apoio e desenvolvimento
Linguagem derivada de uma ER, ***L(e)***; equivalência

L(e) - conjunto de todas as frases que ***derivam*** da ER 'e'

(T) f € L(e) sse e =>* f sendo f € T*

a | (b c) => a a € L(e)
a | (b c) => (b . c) => b. (c) => b . c € L(e)

a+ | (b* c) => a+ => a . a* => a . € => a €L(e)
a+ | (b* c) => a+ => a . a* => a . a . a* => a.a. a . a* => a.a.a a . a* =>
a.a.a.a . € => a a a a €L(e)
a+ | (b* c) => b* c => b.b* c => b b . b* c => b b € c => b b c €L(e)
a+ | (b* c) => b* c => € c => c €L(e)

ac ?€ L(e) ? não se pode obter por derivações sucessivas de 'e'
bcccccc ?€ L(e) ? não se pode obter por derivações sucessivas de 'e'

dados 2 ER, e1 e e2, as ER são **equivalentes** sse L(e1) = L(e2)

formaC(e1) == formaC(e2)

(a | b) . c ? (b | a) . c
(a | b) . c (a | b) . c*
ac € L(e1) ac € L(e2)
a nao-€ L(e1) a € L(e2)

…….

Semana 3 - de 1 de Mar. a 5 de Mar.

T1 (0):

faltei para arguir a tese de Hernâni Silva

T2 (128):

+ Introdução às ER
  - Reconhecedores (algoritmo de pattern-matching)
+ Autómatos Finitos Deterministas (AFD)
  - definição formal
  - linguagem definida por um autómato
  - equivalência ER <-> AFD
  - exemplos: "ab" e "(a+) | (b*c)"
+ Geração Automática de Programas a partir de uma especificação na forma de um ER;
  - algoritmos ***Table Driven***

Programa que Reconhece uma frase (Pattern-Matcher):
DADA 1 SEQUENCIA DE SIMBOLOS , SABER SE DERIVA DE UMA ER

e € ER < -- > ad € AFD

AFD ===> Algoritmo de ***pattern-matching***
é iterativo
é simples
é standard (não depende da ER dada)
é EFICIENTE

GERAÇÃO AUTOMÁTICA DE PROGRAMAS

A partir de uma ER dada, pretende-se derivar automaticamente o programa RECONHECEDOR:

```
q   <- S
Repetir:
   t <- dasimbolo(f)
   q  <- delta[  q,t ]          // TABELA DE DECISÃO
Ate:  q € Z  ou q € erro
```

este Reconhecedor é um algoritmo TABLE-DRIVEN

AFD = < T, Q, Z, S, delta : Q x T -> Q >

Semana 4 - de 08 de Mar. a 12 de Mar.

T1 (114):

+ Reconhecedores de ER e algoritmo de pattern-matching
+ Autómatos Finitos Deterministas (AFD)
    - mais exemplos da equivalência ER <- > AFD
+ Autómatos Finitos Não-Deterministas (AFND)
    - definição formal
    - equivalência ER <-> AFND e AFND <-> AFD
+ Os processos de programação por 'Generalização' de funções e por 'Especialização'

AFND = < T, Q, Z, S, delta' : Q x (T U €) -> { Q } >

e € ER < -- > and € AFND < -- > ad € AFD

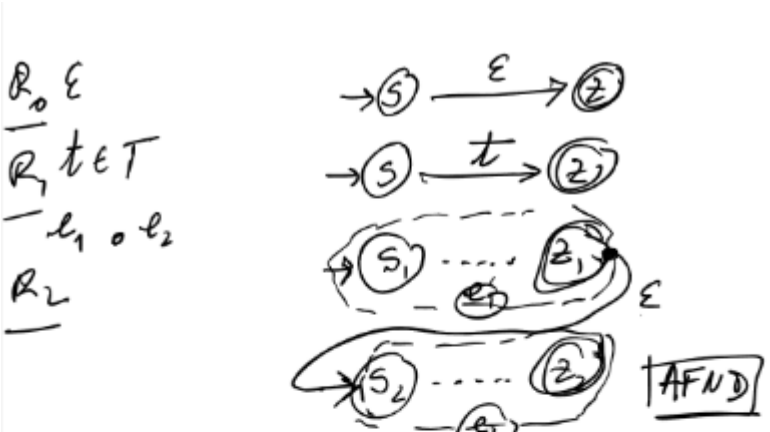
```
recER = re.compile( r'..e...' )   // especialização
recER.search( txf-fnt )

re.search( r'...e…' , txt-fnt )    // generalização
```

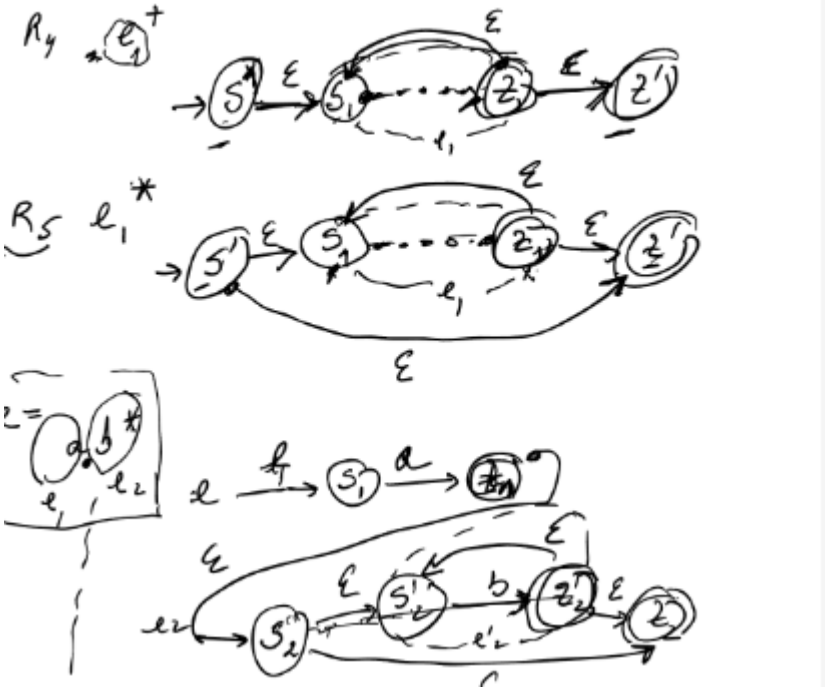
T2 (193): **1º miniTeste**

Semana 5 - de 15 de Mar. a 19 de Mar.

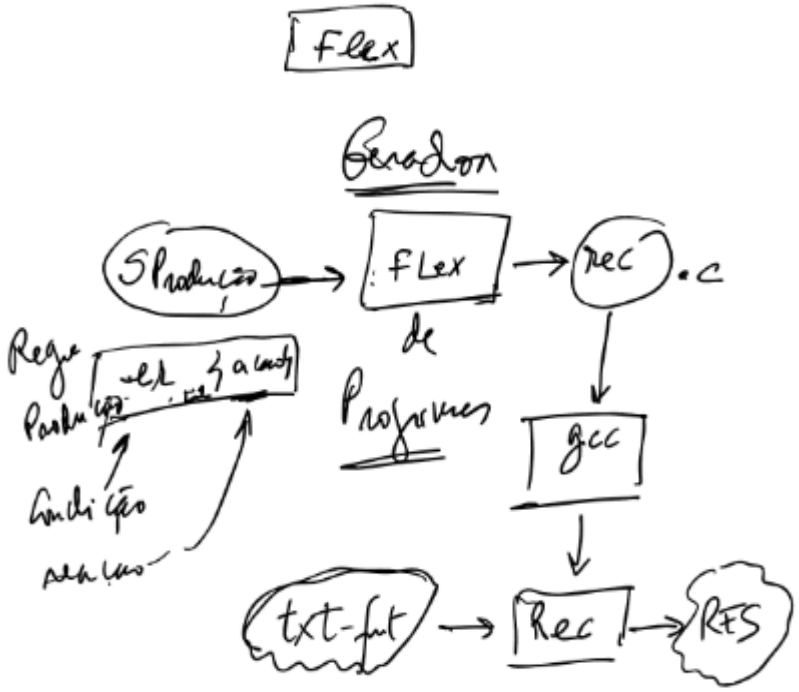
T1 (100):

+ Autómatos Finitos Não-Deterministas (AFND)
    - equivalência ER <-> AFND e AFND <-> AFD:
      as 6 Regras de Conversão ER => AFND
      a função €-fecho({Q}) para assimilar estados alcançáveis por €





- A Conversão ER ⇒ AFD sistematizada e automatizada;
  os Geradores Automáticos de Programas;
  caso concreto do Flex que gera um filtro de texto em C a partir de um Sistema de Produção (SP) -- conjunto de regras de produção da forma **< condição, reação >** , em que a condição é uma ER.



Exemplo de um SP em Flex:

```
%{
#define pal   1
#define num 2
%}
%%
[()]                    { return(yytext[0]); }
[a-zA-Z]+               { return(pal); }
[0-9]+(\.[0-9]+)?       { return(num); }
.|\n                    { ; }
%%
```

Exemplo de um ANALISADOR LÉXICO

T2 (120):

+ Processamento de Linguagens Formais
  - fases de um **PL** / **Compilador**
    (GPL(C, Java, Pascal, Python) / DSL(SQL))
  - Compilador : Ling de Programação AN -> Codigo Máquina
  - Fases de um PL
  - Análise Léxica

```
int     x;
char ch;

x  =
      12+ch
;             // reconhecer a semântica desta instrução
        x <- add(12,ch)         // o significado da instrução determina o processo de tradução
=/=
return(0);
END          For
```

INT(1) ID(2) PV(3) CHAR(4) ID(2) PV(3) ID(2) OPATR(5) NUM(6) OPAD (7) ID(2) PV(3)
RET(8) PE(9) NUM(6) PD(10) PV(3)

txt-fnt =compilador=> **txt-final** //preservar a semântica

Compilação (processamento de uma linguagem):
Reconhecimento -- (SIGNIficado) --> *Tradução/Transformação* (Geração de Código)
Análise
(Txt-fnt) -> Anal-Léxica -> Seq(T) -> Anal-Sintático -> (AD) -> Anal-Semantico -> (SIGN)

Nota: continuamos a usar Python o módulo **ply.lex**
tokens = ('T1', 'T2', …)

Semana 6 - de 22 de Mar. a 26 de Mar.

T1 (110):

+ Análise Sintática = Parsing
+ Gramática Independente de Contexto (GIC/CFG)
    - GIC estabelece regras estruturais ou sintáticas da linguagem

GIC = < T, N, S, P >

T : conj dos simbolos terminais (tokens)
N : conj dos simbolos não-terminais
S: Axioma / Start-symbol € N
P: conj de Produções em que p € P: LHS => RHS
X €N => X1 … Xi .. Xn , n >= 0
Xi € (N u T)

EXEMPLO

```
T =  { ID,  NUM,                          %Símbolos Variáveis
       OPATR, OPAD, PD, PE, PV,           % Sinais
       INT, CHAR                          % Palavras Reservadas   }
```

```
OPATR: r'(=)|(\<\-)'
….
PV : r';'
INT : r'[Ii][Nn][Tt]'
CHAR : r'CHAR'
INT32: r'INT32'
ID :     r'\w+'
NUM:      r' \d+'
REAL:     r'\d+\.\d+'
outro:            .
```

12 + 5.4 ;   NUM OPAD REAL PV

N={ Prg, BlocoDecl, DclVar, BlocoInstr, Instr, Atrib, Retorno, Exp, Oper }

S= Prg

P={   // notação BNF-puro
p1 Prg => BlocoDecl BlocoInstr
p2 BlocoDecl => €
p3 BlocoDecl => BlocoDecl DclVar   //   BlocDecl => DclVar *
p4 BlocoInstr => Instr
p5 BlocoInstr => BlocoInstr Instr
p6 DclVar => INT ID PV
p7 DclVar => CHAR ID PV
p8 Instr => Atrib
p9 Atrib => ID OPATR Exp PV
p10 Expr => Oper OPAD Oper PV
p11 Oper => ID
p12 Oper => NUM
}

considere a frase "**X = 12 + Y;**"
(Prg)
(BlocDecl)
(DclVar)INT(1) ID(2)   PV(3)
(DclVar)CHAR(4)   ID(2) PV(3)

(BlocoInstr)
(Instr)(Atrib)ID(2) OPATR(5) (Expr)NUM(6) OPAD (7) ID(2) PV(3)

T2 (189):

+ 2º miniTeste

Semana 7 - de 05.Abr. a 09.Abr.

T1 (90):

+ Análise Sintática = Parsing
+ Descrição detalhada da especificação do Analisador Léxico com base em ER e reações
+ Gramática Independente de Contexto (GIC/CFG)
    - Exemplo 2: linguagem para invocar uma função (lista mista de argumentos)

Exemplo2: Invocação de uma função
sqrt(44), power(3,4); power(base,exp), add(lista,31)
id()
id(id)
id(num)
id(id,num,id,num)
id (num, id, id, num) id(num,num)

T={ id, num, PE, PD, VIRG }
N={ InvF, Args, Arg, Cauda }
S= InvF
P={
p1 InvF => id PE Args PD
p2 Args => €
p3 Args => Arg Cauda
p4 Cauda => €
p5 | VIRG Arg Cauda
p6 Arg => id
p7 | num
}

T2 (90):

+ Análise Sintática = Parsing
+ Gramática Independente de Contexto (GIC/CFG)
  define a estrutura/forma ou SINTAXE de cada frase válida
  - ….

Exemplo3: Symbolic Expressions (**Lisp**)
12
atomo
(56)
(caso)
(add 1 (mul (sub 6 2) 3))
~~( )~~

T={ pal, num, PE, PD }
N={ Lisp, SExp, SExpList, Cauda }
S= Lisp
P={ // ***Sistema de Produção*** *= { Regra de Produção: COND,REACAO }*
p1 Lisp => SExp { REACAO }
p2 Sexp => num
p3 | pal
p4 | PE SExpList PD
p5 SExpList => SExp Cauda
p6 Cauda => €
p7 Cauda => SExpList // SExp Cauda
}

se for permitida a lista vazia
SExpList => €
| SExp SExpList

Semana 8 - de 12.Abr a 16.Abr

T1 (90):

+ Análise Sintática = Parsing
+ Gramática Independente de Contexto (GIC/CFG)
  define a estrutura/forma ou SINTAXE de cada frase válida

Exemplo4TP! : Turma com Alunos e Notas

TURMA PLT1
a12345 - "Ana d'Araujo Sousa e Ramos" : 1, 2 , 3
pg4321 - "Pedro Rangel Henriques" : 6, 5 e678 - "Romeo Bartolo" : 8

resultado possível
a12345 tem a nota final de 2 por isso Reprovou
pg4321 tem a nota final de 5.5

id - nome : notas
"c*" / 'c*' / { c* }

T={ id, num, nome, TURMA, '-', ':', ',' } // Tokens
N={ Turma, Aluno, Alunos, Cabec, Corpo, Cauda, Notas, RNotas, Nota }
S= Turma // axioma, ou símbolo inicial
P={ // ***Sistema de Produção*** *= { Regra de Produção: COND,REACAO }*
p1 Turma => Cabec Corpo
p2 Cabec => TURMA id
p3 Corpo => Alunos
p4 Alunos => Aluno Cauda
p5 Cauda => €
p6 Cauda => Aluno Cauda
p7 Aluno => id '-' nome ':' Notas
p8 Notas => Nota RNotas
p10 RNotas => €
p11 | ',' Nota RNotas
p12 Nota => num
}

T2 (60):

- + Análise Sintática = Parsing
- + **Gramática Independente de Contexto (GIC/CFG)**
  define a estrutura/forma ou SINTAXE de cada frase válida
- + Recursividade e Parsing
- + Introdução **as Gramáticas Tradutoras (GT)** e **Ações Semanticas (AS)**
- + Introdução Informal ao Parsing Top-Down (TD) e Bottom-Up (BU)

//**alternativa-1** a p4, p5 e p6
p4' Alunos => Alunos Aluno
p4'' Alunos => Aluno

//**alternativa-1** (Rec à Esquerda) a p8, p10, p11
p8' Notas => Nota
p8'' Notas => Notas ',' Nota //associa os elementos pela Esq
//alternativa-2 (Rec à Direita) a p8, p10, p11
p8''' Notas => Nota {as8'''}
p8'''' Notas => Nota ',' Notas {as8''''}

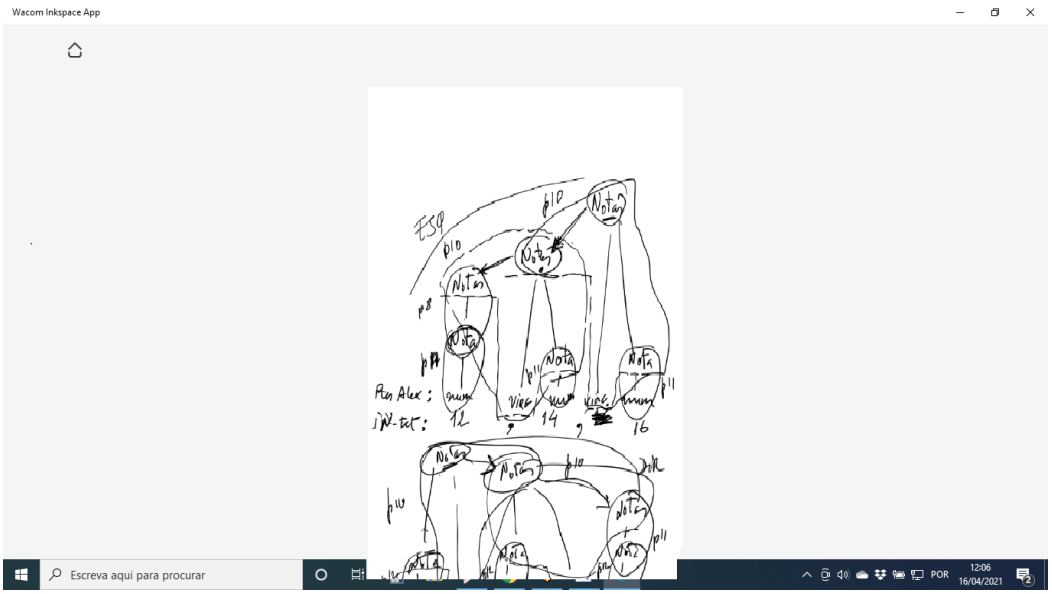
Exp: (((4 + 5) + 6)) - 12
Exp: (((34 * 2)) / 8) * 10 // Exp => Exp '*' Termo

/** Aproximação ERRADA
Notas => €
Notas => Notas ',' Nota

Notas
Notas
Nota Nota
num num
€ , 12 , 3
****/

/** Escrita padrão para Lista Vazia ou Lista com elementos Separados por ","
Notas => €

Notas => LstN
LstN => Nota
LstN => LstN ',' Nota
****/



Semana 9 - de 19.Abr a 23.Abr

T1 (80):

+ Análise Sintática = Parsing
    - Parsing Bottom-Up (BU) LR: algoritmo

GIC ==> Autómato Determinista LR ⇒ Tabelas de Reconhecimento (
ACTION :: Q x T -> AcRec
(aceita, erro, shift(q), red(p))
GOTO :: Q x N -> Q

Generating LALR Tables

Reconhecedor Bottom-UP (BU) *iterativo Table-driven*

Tabela de Decisão :
ACTION : Q x T -> acao
GOTO : Q x N -> Q

```
recLR {
 SP = NULL
 push(Q0,SP)   // Q0 identifica o estado inicial do AD
t <- getSymbol()
 repetir{
        q <- top(SP)
        acao <- ACTION [q,t]
        CASO (acao) SEJA
         aceita : ;
         erro   : error();
         shift  : // "q" estado associado a esta transicao
                  push(q, SP);
                  t <- getsymbol();
         reduce : // "p" identifca a produção reconhecida pela qual devo reduzir
                  n <- #RHS(p) // numero simbolos à direta da produção p, n>=0
                  executa( AS(p) ) //Acao Semantica associada a produção p
                  popn(SP,n)  // remove "n" estados do topo da stack
                  q <- top(SP)
                  q=GOTO[q, LHS(p)]
                  push( q, SP )
        FCASO
 } até ((acao = aceita) OU (acao == erro))
}
```

T2 (180):
- miniteste3

Semana 10 - de 26.Abr a 31.Abr

T1 (70):

- Análise Sintática = Parsing
  - Parsing Bottom-Up (BU) LR: construção das Tabelas ACTION e GOTO

AD-LR(0) = < (T U N), Q, Z, Q0, delta: Q x (TUN) -> acao >

GIC ===> AD-LR(0)

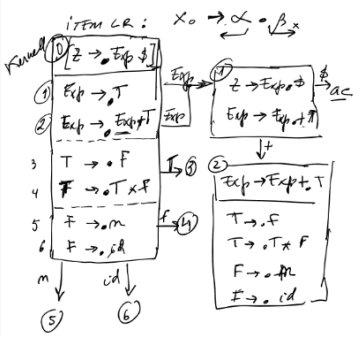
0: Z -> Exp $ // ESTENDER a GIC
1: Exp -> Termo
2: | Exp '+' Termo
3: Termo -> F
4: | Termo '*' F
5: F -> num
6: | id

T = { num , id, '+', '*' }
N = { Exp, Termo, F }

Exemplo de Textos de Entrada (Input) que são frases válidas (pertencem à Linguagem definida pela gramática)
12
abs
a + 2
b * 3
a + b * 4 + 6

Desenho da Construção do Autómato

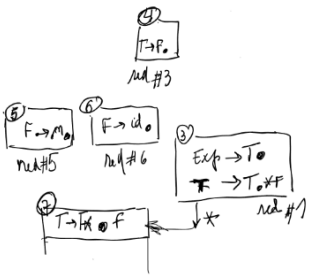
Wacom Inkspace App



Escreva aqui para procurar

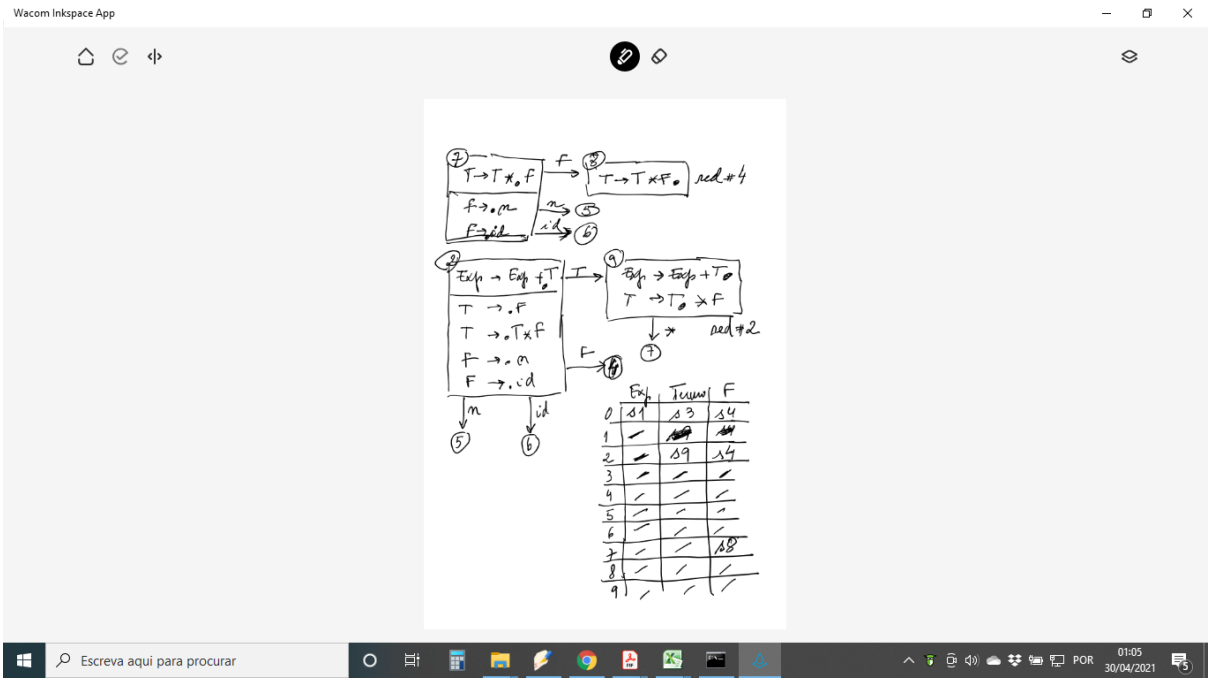
POR 12:10 28/04/2021

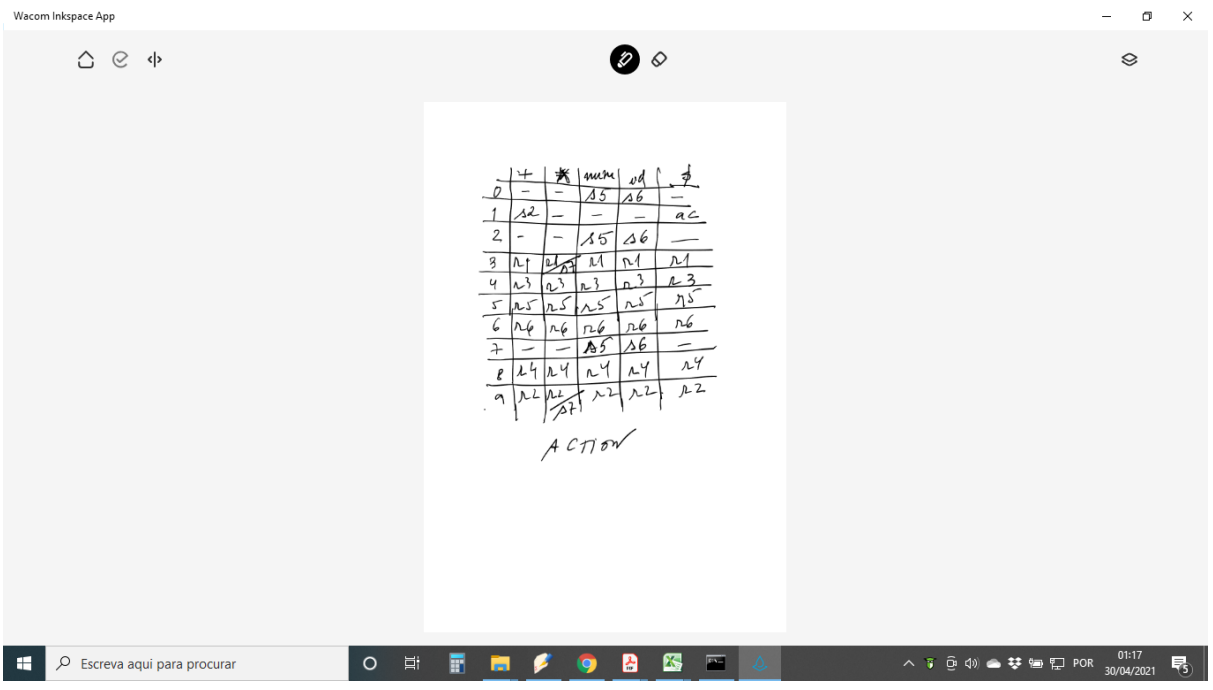
Wacom Inkspace App



Escreva aqui para procurar

POR 00:31 30/04/2021





T2 (70):

+ Análise Sintática = Parsing
  - Parsing Bottom-Up (BU) LR: conclusão do processo de construção das Tabelas ACTION e GOTO
+ Introdução à VM: arquitetura da máquina, princípio de funcionamento; conjunto de instruções

Semana 11 - 03 de Maio a 07 de Maio

T1 (65):

+ Análise Sintática = Parsing ; recapitulação do problema e do algoritmo Bottom-Up table-driven LR
+ Introdução à VM: arquitetura da máquina, princípio de funcionamento; conjunto de instruções; análise mais detalhada de algumas instruções

Explicação das instruções **PUSHI, PUSHG, LOADN, STOREN, PUSHGP**
Explicação da instrução **PADD**
que adiciona um offset inteiro a um endereço de base retornando um endereço

EXEMPLO do acesso a uma componente (indice) de um array

Tabela de Identificadores (pertence ao Compilador e existe em *Compile-time*)

v --> var cat 'array' -- e0 => endr-base(v)=10

i --> var cat 'atomica(int)' -- e1 =>endr(i)=3

Acessos

v[i] --> aceder à posição de memória cujo endereço é = e0 + valor(endr(i))
10 + valor(3)

v[i+2*4] --> aceder à posição de memória cujo endereço é = e0 + (valor(endr(i))+2*4)

Implementação da operação v[i] = v[i+2*4] + 1 i[v] = 9

```
//carregar para a stack o endr-base do array 'v'
 PUSHGP
 PUSHI   10 // e0=endr(v)
 PADD
//carregar para a stack o valor do indice i
 PUSHG    3 // valor(endr(i))
// fica na base da stack o endereço destino do STOREN
 //carregar para a stack o endr-base do array 'v'
 PUSHGP
 PUSHI   10 // e0=endr(v)
 PADD
 //carregar para a stack indice que é o valor da expressão i+2*4
 PUSHG    3 // valor(endr(i))
 PUSHI    2
 PUSHI    4
 MUL
 ADD
 //acesso para ir buscar o valor de  v[i+2*4]
 LOADN
 // incrementa v[i+2*4] de 1 unidade
 PUSHI    1
```

ADD
STOREN // armazena o resultado da expressão v[i+2*4] + 1 em v[i]

Outro Exemplo -- o estado da Stack após a execução das instruções abaixo

```
????    <= SP
0,astr
5,int
20,int   <= FP
--------
0,int
0
0
5
20,int    <= GP(0)
--------
```

```
PUSHI 20
PUSHI 5
PUSHN 3
PUSHGP 0
PUSHGP 1
PUSHS "O Resultado e"          O Resulto e\0
                               0
```

T2 (65)

+ Parsing Top-Down Recursivo Descendente: introdução ao conceito; construção de um primeiro exemplo

Semana 12 - 09 de Maio a 14 de Maio

T1 (65)

+ Parsing Top-Down Recursivo Descendente: reforço do conceito; conclusão da construção de um primeiro exemplo Listas Mistas)
+ Introdução informal aos conceitos de **Anulável**(alpha), **First**(alpha), **Follow**(N), **Lookahead**(P), sendo 'alpha' uma string (seq de símbolos), 'N' um não-termial e 'P' uma produção; noção de ***Conflito LL(1)***

Parsing Top-Down (TD)

parsing **preditivo** (olha para a GIC e depois verifica no Txt-Fnt)

```
p1 LstM -> INIC Elems FIM    $   // la(p1) = { INIC }
p2 Elems -> Elem Resto           // la(p2) = first(Elem) = { pal, num }
p3 Elem -> pal                   //  la(p3) = { pal }
p4        |   num                //  la(p4)  =  { num  }
p5 Resto -> €                    //  la(p5)  =  follow(Resto) = { FIM }
```

p6 | ',' Elems // la(p6) = first(',') = { ',' }

Algoritmo Recursivo-Descendente

PS designa o *próximo símbolo Terminal* (*a ser decidido se pode ser aceite*)
O Parser vai ter #N funções de Reconhecimento + #T ou 1

rec-LstM(PS)

```
{   se ( PS in la(p2) )
    entao { rec-INIC(PS);  rec-Elems(PS); rec-FIM(PS) }
                                  // rec-Term(PS,INIC).....rec-Term(PS,FIM)
    senao { erro(1,PS); }
}
```

rec-Elems(PS)

```
{   se ( PS in la(p2) )
    entao {  rec-Elem( PS ); rec-Resto( PS ) }
    senao {  erro(2,PS); }
}
```

rec-Elem(PS)

```
{   se ( PS in la(p3) )
    entao { rec-Term(pal,PS); }
    senao { se ( PS in la(p4) )
              então { rec-Term( num,PS); }
              senao [ erro(3,PS); }
            }
}
```

rec-Resto(PS)

```
{   se ( PS in la(p5) )
    entao { ; }
    senao { se ( PS in la(p6) )
              entao { rec-Term( PV,PS }; rec-Elems(PS); }
              senao { erro(4,PS); }
            }
}
```

((alternativa que facilita a geração automática))

rec-Resto(PS)

```
{   switch ( PS ) {
     casein la(p5) : skip ;
     casein la(p6) : rec-Term( PV,PS }; rec-Elems(PS);
     else : erro(4,PS);
    }
```

```
}
```

rec-INIC(PS)

```
{    se ( PS==INIC )
     entao { PS = daSimbolo()   } // lex()
     senao { erro(2,PS)  }
}
```

rec-FIM(PS)

```
{    se ( PS==FIM )
     entao { PS = daSimbolo()   } // lex()
     senao { erro(3,PS)  }
}
```

rec-pal(PS)

```
{    se ( PS==pal )
     entao { PS = daSimbolo()   } // lex()
     senao { erro(4,PS)  }
}
```

rec-num(PS)

```
{    se ( PS==num )
     entao { PS = daSimbolo()   } // lex()
     senao { erro(5,PS)  }
}
```

rec-PV(PS)

```
{    se ( PS==PV )
     entao { PS = daSimbolo()   } // lex()
     senao { erro(6,PS)  }
}
```

((--- otimização ---))

```
rec-Term( T, PS )   // T representa o Símbolo esperado
{    se ( PS==T )
     entao { PS = daSimbolo()   } // lex()
     senao { erro(100,PS.T)  }
}
```

lookahead : P -> set(T)
lookahead(p: X0 : X1… Xi … Xn) =
first (RHS(p)) (U follow (X0), se anulavel(RHS(p)))

first : seq((NUT)*) -> set(T)
first(€) = { }
first(t) = { t }
first(X1, X2, ,,, Xn) =
first(X1) [U first (X2, …, Xn) se anulavel(X1)]
first(N) = Up first (alpha), p: N -> alpha
follow : N -> set(T)
follow(N) = Up, N€RHS(p) first(delta),
U follow(LHS(p)) se anul(delta))
sendo RHS(p)=beta N delta
anulavel : seq((NUT)*) -> Bool
anul(€) = True
anul(t) = False
anul(N) = True, sse Existir p: N -> alpha & anulavel(alpha)
anul(X1, X2, …, Xn) = anul(X1) & anul(X2, .., Xn)

Pascal : Header Corpo "."
Corpo : BDecls BStats
first(Corpo) = first(BDecls BStats) = first(BDecls) U first(BStats)
= { DECLS } U { BEGIN }
follow(Corpo) = first(".") = { "." }

BDecls : €
| DECLS ----
first(BDecls) = first(€) U first(DECLS) = first(DECLS) = { DECLS }

BStats : €
| BEGIN ----
first(BStats) = first(€) U first(BEGIN) = first(BEGIN)

anulavel (Corpo) = anulavel(BDecls BStats) = anulavel(BDecls) &
anulavel(BStats) = True & True = True

```
la( Corpo : BDecls  BStats ) =
     first(  BDecls  BStats )  U follow( Corpo ) =
     first( BDecls ) U first( BStats ) U follow( Corpo ) =
     { DECLS } U { BEGIN } U follow( Corpo ) = { DECLS, BEGIN, "." }

 Z  ->  regPar  $
 regPar -> reg regPar      // la(p1) = { # }
         |  €              // la(p2) = { $ }
follow(regPar) = { $ }
anul(regPar) = True
first(regPar) = first(reg) = { '#' }

  reg    -> '#' data reg2   // la(p3) = { # }
  reg2   -> bat             // la(p5) = { 'batismoDe' }
         |  nasc            // la(p6) = { 'nascimentoDe' }

  bat    -> 'batismoDe' nome // la(p7) = { 'batismoDe' }

  nasc   -> 'nascimentoDe' nome 'MAE' nome 'PAI' nome  // la(p8) =
    { 'nascimentoDe' }
  nome   -> String   // la(p9) = { String }

follow(nome) = follow(bat) U  { 'MAE' } U { 'PAI' } U follow(nasc)
                = follow(reg2) U { 'MAE', 'PAI' } = { '#', '$' ,  'MAE', 'PAI' }
follow(bat) = follow(reg2)
follow(nasc)= follow(reg2)
follow(reg2) = follow(reg)
follow(reg) = first(regPar) U follow(regPar) = { '#', '$' }
```

T2 (56)

+ Parsing Top-Down Recursivo Descendente: reforço do conceito; conclusão da construção de um primeiro exemplo Listas Mistas)
+ Continuação da apresentação informal aos conceitos de **Anulável**(alpha), **First**(alpha), **Follow**(N), **Lookahead**(P), sendo 'alpha' uma string (seq de símbolos), 'N' um não-termial e 'P' uma produção; noção de ***Conflito LL(1)***

Semana 13 - 17 de Maio a 21 de Maio

T1 (60)

+ Parsing Top-Down Recursivo Descendente: reforço do conceito; conclusão da construção de um primeiro exemplo Listas Mistas)
+ Apresentação formal dos conceitos de **Anulável**(alpha), **First**(alpha), **Follow**(N), **Lookahead**(P), sendo 'alpha' uma string (seq de símbolos), 'N' um não-terminal e 'P' uma produção.

T2 (160)

+ miniTeste4

Semana 14 - 24 de Maio a 26 de Maio

T1 (10)

+ Balanço e Fecho da UC

Aulas Práticas

Semana (1) de 15 de Fev. a 19 de Fev

aulas só começam na 5ªfeira dia 18

Semana (2) de 22 de Fev. a 26 de Fev.

TP1 (31):

Introdução
Paradigmas e Linguagens de Programação ; Python
ERs em Python:
os métodos re.match, re.search e re.findall

TP2
TP3
TP4
TP5

Exercícios:

Semana (3) de 1 a 5 de Mar.

TP1 (20):

ERs em Python:

os métodos sub(), subn() e split()

conclusão dos exercícios da questão 0 da ficha pl20f00 (resolução detalhada)

Semana (4) de 8 de Mar. a 12 de Mar.

TP1 (15):

ERs em Python:

os métodos search, findall() e split() revisitados

conclusão do exercício 'search-geral' da questão 0 da ficha pl20f00

introdução ao conceito de 'Group ()' para manipular sub-ER

resolução do exercício IPV4/IPV6

resolução de 1 alínea do exercício 'arqson' da ficha pl20fpy01

Semana (5) de 15 de Mar. a 19 de Mar.

Exercícios:

TP1 (14)

ERs em Python:

os métodos search, findall() e split() revisitados

conclusão do exercício 'arqson' da ficha pl20fpy01

(uso de um dicionário em Phyton para contar por por regiões)

resolução de 2 exercícios com o módulo 'er' do miniT1

implementação de um tradutor XML para HTML

solução 1: transformação em bloco

Semana (6) de 22 de Mar. a 26 de Mar.

para instalar o PLY em qq um dos Sistemas Operativos

devem usar o package manager do Python:

pip3 install ply

TP1 (10)

Análise Léxica em Python:

+ o método lex do módulo PLY
  introdução ao analisadores léxicos simples (somador)

Semana () de 29 de Mar. a 02 de Abr.

Férias de Páscoa:

Semana (7) de 05 de Abr. a 09 de Abr.

TP1 (10)

Análise Léxica em Python:

+ o método lex do módulo PLY
  analisadores léxicos simples (calclex e somador-ply)

Semana (8) de 12 de Abr. a 16 de Abr.

TP1 (10)

Análise Léxica em Python:

+ o método lex do módulo PLY com condições de contexto / estados
  - analisadores léxicos com states (somador-cc-ply, rem-comments)
  - analisadores léxicos para emparelhar com os parsers (mistas-lex)

Análise Sintática em Python:

+ o método yacc do módulo PLY
  - parser para listas mistas não-vazias e vazias com e sem ações semânticas (mistas-only-grammar e mistas-yacc)

Notas:

Semana (9) de 19 de Abr. a 23 de Abr.

TP1 (10)

Análise Léxica em Python:

- analisadores léxicos para emparelhar com os parsers (SExp)

Análise Sintática em Python:

+ o método yacc do módulo PLY
  - parser para a linguagem Lisp, SExpression, com ações semânticas (Sexp)

Semana (10) de 26 de Abr. a 31 de Abr.

TP1 (10)

Escrita de GIC para definir novas linguagens :

- Exercício 10 da Folha 2 (Ling de Programação Infantil 2 para descrever uma BC e fazer questões)
- Exercício 14 da Folha 2 (Ling para descrever Ementas)

Semana (11) de 03 de Maio a 07 de Maio

TP1 (10)

Construção do Autómato LR(0) e Tabelas para a Linguagem Lisp

Escrita de GIC para definir novas linguagens :

- Exercício 13 da Folha 2 (Ling BibTeX); construção do Autómato LR(0)

Semana (12) de 10 de Maio a 14 de Maio

TP1 (10)

Escrita de GIC para definir novas linguagens :

- Exercício 13 da Folha 2 (Ling BibTeX); Implementação do ALex e Parser

Semana (13) de 17 de Maio a 21 de Maio

TP1 (10)

Construção do Autómato LR(0) e Tabelas LR para a Linguagem Bibtex

Cálculo dos Lookahead para a Linguagem Bibtex

Programação de um ciclo Repetir em Assembly da VM:

```
a = 5
repeat 4:
  a += 1   // a = a + 1
print(a)
```

```
TabIdent
a -> 0
```

```
PUSHI 5
START
pushi 4 // conta=4
ciclo: NOOP
pushl 0  //testa se conta já é 0
jz fimciclo

pushg 0
pushi 1
add
storeg 0

pushl 0
pushi 1
sub
storel 0
jump ciclo

fimciclo: NOOP
pushg  0
writei

stop
```

Semana (14) de 24 de Maio a 26 de Maio

TP1 (3)

+ Revisões e esclarecimento de dúvidas sobre o TP2
