Processamento de Linguagens

Trabalho Prático Grupo 6 "AAAAAA"

Relatório de Desenvolvimento

Francisca Barros (a96434) | Joana Pereira (a97588) | Rafael Correia (a94870)

28 de maio de 2023

Resumo

O presente relatório descreve o trabalho prático realizado pelo Grupo 6 no âmbito da Unidade Curricular de Processamento de Linguagens. O objetivo do trabalho foi desenvolver um conversor de *Pug* para *HTML*, utilizando os componentes *Lexer* e *Yacc* para realizar a análise léxica e sintática do código *Pug*. O grupo procurou implementar as funcionalidades necessárias para a maioria dos casos de ficheiros *Pug*, explorando as características da linguagem, como atributos, condicionais, iterações, entre outros. Ao longo do processo de implementação e testes, os resultados obtidos demonstraram a capacidade do conversor em converter corretamente o código *Pug* para *HTML*, preservando a estrutura e o conteúdo dos ficheiros.

Conteúdo

Capítulo 1

Introdução

1.1 Um belo Projeto

O presente relatório descreve o trabalho prático realizado pelo Grupo 6 no âmbito da Unidade Curricular de Processamento de Linguagens, inserida no curso de Licenciatura em Engenharia Informática durante o 2º Semestre do ano letivo 2022/2023.

Neste projeto, o grupo optou por abordar o tema 2.5, que consiste na criação de um conversor de *Pug* para *HTML*. O *Pug* é uma linguagem de modelação de *templates* para *Node.js*, reconhecida pela sua sintaxe simples e intuitiva. A escolha deste tema revelou-se interessante para o grupo devido à familiaridade com essa linguagem, adquirida na disciplina de Engenharia Web.

O objetivo deste trabalho é desenvolver um conversor de *Pug* para *HTML*, capaz de implementar as funcionalidades necessárias para a maioria dos casos de ficheiros *Pug*. Procurámos criar um conversor que seja eficiente e capaz de preservar corretamente a estrutura e o conteúdo dos ficheiros, garantindo a correta conversão para o formato *HTML*.

O relatório está dividido em várias secções. Iniciamos com a introdução, que apresenta uma visão geral do trabalho realizado. Em seguida, abordamos a análise e especificação, onde descrevemos informalmente o problema a ser resolvido e estabelecemos os requisitos necessários para a sua resolução. Posteriormente, apresentamos a conceção/desenho da solução, detalhando as estratégias adotadas para o desenvolvimento do conversor. Prosseguimos com a secção de codificação e testes, onde descrevemos o processo de implementação e os testes realizados para verificar a correta funcionalidade do conversor. Por fim, concluímos o relatório com uma síntese das principais conclusões obtidas ao longo do projeto.

O objetivo final deste trabalho é fornecer um conversor de *Pug* para *HTML* que atenda aos requisitos estabelecidos, permitindo a correta conversão de ficheiros *Pug* em *HTML* de forma eficiente e confiável.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O objetivo deste projeto é criar um conversor de ficheiros *.pug* para ficheiros *.html*, garantindo a preservação correta de toda a informação. Em outras palavras, ao utilizar um ficheiro em qualquer um dos formatos, não deverá haver diferenças visuais. Para atingir este objetivo, iremos utilizar a biblioteca [PLY], criando *tokens* com o *ply.lexer* e um *parser* com o *ply.yacc*. O nosso foco é assegurar a conversão adequada entre os formatos, mantendo a integridade dos elementos e a formatação correta, de modo a proporcionar um resultado consistente e de qualidade.

2.2 Especificação dos Requisitos

Como requisitos iniciais do nosso projeto, definimos as funcionalidades necessárias para ser possível converter o exemplo fornecido pelo professor. Após obtermos sucesso nessa etapa, avançamos para a implementação das várias funcionalidades extra do *Pug*. Para isso, consultámos a documentação do [Pug] e dedicámo-nos a incorporar cada uma das suas características no nosso conversor de *Pug* para *HTML*. Sendo estas:

- Atributos: Implementar a capacidade de lidar com atributos em várias linhas, entre aspas, com interpolação, booleanos e de estilo.
- Condicionais: Desenvolver a funcionalidade de condicionais, como o *if*, *else if*, *else* e *unless* permitindo controlar o fluxo do programa com base em determinadas condições.
- Iterações: Implementar iterações utilizando estruturas como *each* e *while*, permitindo repetir blocos de código com base em listas ou condições. Dessa forma, garantimos a correta renderização e repetição dos elementos no ficheiro *HTML*.
- Interpolação: Desenvolver a funcionalidade de interpolação, permitindo a inserção de variáveis ou expressões dentro do próprio texto. Isso possibilita a correta substituição e exibição dos valores no ficheiro *HTML* final.
- Herança de *templates*: Implementar a capacidade de herança de *templates*, permitindo a criação de *layouts* ou estruturas comuns que podem ser estendidos por outros ficheiros *Pug*. Isso garante a correta organização e reutilização de código no processo de conversão para *HTML*.

- *Mixins*: Desenvolver a funcionalidade de *mixins*, que são blocos de código reutilizáveis, permitindo a definição e invocação de trechos de código em diferentes partes de um ficheiro *Pug*. Isso proporciona uma maior modularidade e reutilização de código no processo de conversão.
- Texto simples: Implementar a capacidade de incluir texto simples, sem qualquer marcação específica. Dessa forma, garantimos a correta representação do texto no ficheiro *HTML* gerado.
- *Tags*: Desenvolver uma sintaxe simplificada para a criação de *tags HTML*, facilitando a estruturação e formatação do documento final. Isso garante a correta conversão das *tags* para o formato *HTML* correspondente.

Ao abordarmos cada uma dessas características, procuramos implementá-las adequadamente no nosso conversor de *Pug* para *HTML*, garantindo a correta conversão e preservação dos elementos e estruturas no ficheiro *HTML* gerado. Dessa forma, buscamos entregar um conversor funcional e eficiente, capaz de lidar com diversas funcionalidades do *Pug* e produzir resultados de alta qualidade.

Capítulo 3

Concepção/desenho da Resolução

No Capítulo 3, intitulado "Conceção/Desenho da Resolução", iremos abordar a fase de conceção e desenho da solução para o desenvolvimento do conversor de *Pug* para *HTML*. Nesta etapa, iremos explorar dois componentes essenciais: o *Lexer* (Analisador Léxico) e o *Parser* (Analisador Sintático). O *Lexer* será responsável por realizar a análise léxica do código fonte em *Pug*, identificando os diferentes *tokens* presentes no ficheiro. Já o *Parser* irá realizar a análise sintática, definindo a estrutura gramatical do código *Pug* e permitindo a construção do código HTML correspondente. Através da exploração destes componentes, iremos estabelecer as bases para a implementação eficiente e correta do conversor.
No nosso conversor de *Pug* para *HTML*, conseguimos implementar várias funcionalidades essenciais do *Pug*, enriquecendo a conversão de código de forma significativa. Estas são:

- Atributos e Atributos em multiplas linhas
- *Class Literal*: Implementamos a capacidade de utilizar a sintaxe do *Pug* para definir classes diretamente nos elementos *HTML*, facilitando a atribuição de estilos e seleção de elementos e evitando a constante definição de *tags div*.
- *ID Literal*: Da mesma forma, também permitimos a utilização de IDs literais nos elementos HTML, permitindo a identificação única de elementos específicos no código convertido.
- Declaração de variáveis: Além disso, o nosso conversor é capaz de lidar com declarações de variáveis (ex: var x = 0).
- Comentários: Implementamos o suporte para a maioria dos tipos de comentários disponíveis no *Pug*, permitindo a inclusão de notas e observações no código.
- Condicionais: Conseguímos interpretar e converter condicionais do Pug (*if,else if,else, unless*), possibilitando a execução condicional de trechos de código HTML com base em certas condições lógicas.
- *DOCTYPES*: Também conseguimos implementar o suporte aos diferentes tipos de *DOCTYPES* disponíveis no *Pug*, permitindo especificar a versão e o tipo de documento HTML gerado.
- *Includes* em *Plain Text*: com esta funcionalidade conseguímos implementar no ficheiro *Pug* textos contidos noutros ficheiros de texto.
- Interpolação de Variáveis - é possível substituir variáveis em condições ou em tags seguidas de um igual (ex: title= pageTitle).

- Iterações: Implementámos o *while* e o *each*, permitindo a repetição de blocos de código HTML com base em determinadas condições ou listas de dados.
- Texto simples

3.1 Lexer

Para o *Lexer* fomos definindo os *tokens* de acordo com as funcionalidades que íamos acrescentando e estão divididos em 3 tipos: *literals*, *reserved* e *tokens*.
Os *literals*, tal como é referido na documentação do [PLY], é definido por: *"simply a single character that is returned "as is"when encountered by the lexer. Literals are checked after all of the defined regular expression rules."*. Nós definimos como sendo:

```
literals = [',', ':','[',']','{','}','=']
```

Os *reserved*, também referidos na documentação, são um dicionário que associa palavras reservadas ao respetivo *token*.

```
reserved = {
   'if' : 'IF',
   'else' : 'ELSE',
   'while' : 'WHILE',
   'unless' : 'UNLESS',
   'each':'EACH',
   'in':'IN',
   'include':'INCLUDE'
}
```

Finalmente, os restantes *tokens* são definidos numa lista, posteriormente adicionada aos *tokens reserved*.

```
    tokens = [
    'POINT',
    'PA',
    'PF',
    'EQUALS',
    'TAG',
    'ID',
    'CLASS',
    'ATTRIBUTE',
    'TEXT',
    'NEWLINE',
    'INDENT',
    'DEDENT',
    'CONDITION',
    'LINECOMMENT',
    'BLOCKCOMMENT',
    'PIPED',
    'VALUE',
    'IGNORE_BLOCKCOMMENT',
```

```
    'IGNORE_LINECOMMENT',
    'COMMENT',
    'HASHTAG',
    'NUMBER',
    'STRING',
    'JS',
    'VAR'
] + list(reserved.values())
```

Cada token tem associado uma função especifica, no nosso caso, a função mais complexa acaba por ser a função que deteta *newlines*, visto que é quando aparece uma nova linha que normalmente se muda de estado. Esta função calcula a nova identação comparando-a com a antiga. Consoante o resultado, e o estado em que estamos, são feitas ações diferentes, como vai ser falado posteriormente.
Além dos *tokens*, foram também definidos vários estados para permitir a identificação dos diferentes *tokens* corretamente.

```
    states = (
    ('comment', 'exclusive'),
    ('ignoreComments', 'exclusive'),
    ('dedent','exclusive'),
    ('indent','exclusive'),
    ('firstWord', 'inclusive'),
    ('pointState', 'exclusive'),
    ('conditional', 'exclusive'),
    ('each', 'exclusive'),
    ('js','exclusive')

)
```

3.1.1 Estados comment e ignoreComments

Ambos os estados servem para idenficar comentários em bloco. Quando é detetado um "//"ou um "//-"com espaços no resto da linha entra-se no estado comment ou ignoreComments, dependendo do que é detetado. Estando neste estado, e visto que a função é exclusiva, existem apenas uma forma de sair do estado, através da função que lê new lines, visto que esta está declarada para todos os estados:

```
    def t_ANY_newline(t)
```

Assim, ambos os estados são acabados quando é reconhecido um DEDENT (uma diminuição de identação).

3.1.2 Estados dedent e indent

Estes estados são usados para criar os tokens INDENT e DEDENT. Quando o lexer entra na função do newline, verifica se a identação mudou e consoante a mudança muda uma variável que indica o número de tokens INDENT/DEDENT que devem ser criados e entra no estado respetivo. Estes são exclusivos e têm apenas uma função que cria o número de tokens necessário, saindo do estado após este processo.

3.1.3 Estado firstWord

Este estado permite saber que o lexer está no inicio de uma linha e ainda não descobriu nenhum token além de INDENT/DEDENT. Serve principalmente para detetar TAGs, ou simbolos reservados. É o único estado inclusivo, pois pode receber tokens do estado inicial, além dos tokens das suas funções exclusivas.

3.1.4 Estado pointState

Este estado serve para assinalar que começou um pedaço de texto em bloco. É começado quando é detatado um ponto (à frente de uma tag, ou dos seus atributos, ou da classe ou do id, ou mesmo numa nova linha sem nada). A partir deste ponto, sendo um estado exclusivo, tudo a que se segue é detatado como texto. Apenas sai deste estado quando aparece um DEDENT.

3.1.5 Estado conditional

Este estado serve para detetar condições. É ativado por um if ou por um unless e serve para captar toda a expressão que se segue até ao fim da linha. No final da condição, sai do estado.

3.1.6 Estado each

Este estado serve para os ciclos each. Estando neste estado, o objetivo é captar o que vem antes do IN, para o token VALUE. De seguida, capta-se o IN, e finalmente, este pode detetar strings, listas, sets e dicionários, visto que os simbolos para estes componentes se encontram nos literals e estão definidas as regras para detetar Strings e números. O estado é acabado com o final da linha.

3.1.7 Estado js

Este estado serve para captar código escrito em JavaScript. É iniciado com o carater '-'. Dentro do estado, no nosso lexer, pode aparecer a palavra 'var', que dá origem ao token VAR, ou podem aparecer números, strings e finalmente texto. O estado é terminado no final da linha.

3.2 Parser

No parser foi definida uma gramática para reconhecer ficheiros pug:

```
pug : conteudo

conteudo : conteudo elem
         | elem

elem : TAG tagProperties TEXT INDENT conteudo DEDENT
     | TAG tagProperties TEXT INDENT conteudo
     | TAG TEXT INDENT conteudo DEDENT
     | TAG TEXT INDENT conteudo
     | TAG tagProperties INDENT conteudo DEDENT
     | TAG tagProperties INDENT conteudo
     | TAG tagProperties EQUALS VALUE INDENT conteudo DEDENT
     | TAG tagProperties EQUALS VALUE INDENT conteudo
```

```
      | TAG INDENT conteudo DEDENT
      | TAG INDENT conteudo
      | TAG EQUALS VALUE INDENT conteudo DEDENT
      | TAG EQUALS VALUE INDENT conteudo
      | TAG tagProperties POINT  linhasTexto DEDENT
      | TAG tagProperties POINT  linhasTexto
      | TAG POINT linhasTexto DEDENT
      | TAG POINT linhasTexto
      | TAG tagProperties TEXT
      | TAG TEXT
      | TAG tagProperties
      | TAG
      | TAG tagProperties EQUALS VALUE
      | TAG EQUALS VALUE
      | INCLUDE TEXT
      | IF CONDITION INDENT conteudo DEDENT ELSE INDENT conteudo DEDENT
      | IF CONDITION INDENT conteudo DEDENT ELSE INDENT conteudo
      | IF CONDITION INDENT conteudo DEDENT elsesif ELSE INDENT conteudo DEDENT
      | IF CONDITION INDENT conteudo DEDENT elsesif ELSE INDENT conteudo
      | IF CONDITION INDENT conteudo DEDENT elsesif
      | IF CONDITION INDENT conteudo DEDENT
      | IF CONDITION INDENT conteudo
      | UNLESS CONDITION INDENT conteudo DEDENT
      | UNLESS CONDITION INDENT conteudo
      | LINECOMMENT
      | BLOCKCOMMENT listaComments DEDENT
      | BLOCKCOMMENT listaComments
      | POINT TEXT
      | WHILE CONDITION INDENT conteudo DEDENT
      | WHILE CONDITION INDENT conteudo
      | EACH VALUE IN iterable INDENT conteudo DEDENT
      | EACH VALUE IN iterable INDENT conteudo
      | pipedText
      | JS VAR TEXT "=" elemIterable

elsesif : elsesif ELSE IF CONDITION INDENT conteudo DEDENT
        | elsesif ELSE IF CONDITION INDENT conteudo
        | ELSE IF CONDITION INDENT conteudo DEDENT
        | ELSE IF CONDITION INDENT conteudo

pipedText : PIPED TEXT
          | pipedText PIPED TEXT

iterable : '[' elemsLista ']'
         | '{' elemsLista '}'
         | '{' elemsDict '}'

elemsLista : elemIterable
```

```
           | elemsLista ',' elemIterable

elemsDict : elemIterable ':' elemIterable
          | elemsDict ',' elemIterable ':' elemIterable

elemIterable : STRING
             | NUMBER

listaComments : COMMENT
              | listaComments COMMENT

tagProperties : ID
              | listaClasses
              | PA ATTRIBUTE PF
              | ID PA ATTRIBUTE PF
              | ID listaClasses
              | ID listaClasses PA ATTRIBUTE PF
              | listaClasses PA ATTRIBUTE PF

listaClasses : CLASS
             | listaClasses CLASS

linhasTexto : linhasTexto TEXT
            | TEXT
```

É possível perceber rapidamente que num autómato LR(0) existiriam bastantes conflitos *shift/reduce*, a maior parte relacionado com a identação, visto que regra geral cada *elem* se tiver um INDENT deve acabar com DEDENT, no entanto, se for o final do ficheiro, pode acontecer que este não ocorra.

O parser tem um dicionário global que é usado para guardar as variáveis e os respetivos valores. Para usar estes valores, nos locais onde podem aparecer estas variáveis, no nosso caso, em condições ou atribuições, é usado a biblioteca re para detetar se alguma das variáveis aparecem na expressão a testar e, se aparecer, esta é substituída pelo seu valor correspondente.

Por fim o parser produz um ficheiro texto (test.html) que contém o resultado da transformação dos dados.

Capítulo 4

Codificação e Testes

Neste capítulo vamos explorar a fase de implementação do conversor de *Pug* para *HTML*, bem como os testes realizados para garantir a qualidade e eficácia da solução desenvolvida. Nesta etapa crucial do projeto, iremos detalhar as decisões tomadas, as alternativas consideradas e os problemas de implementação que surgiram ao longo do processo. Além disso, apresentaremos os testes realizados, fornecendo informações sobre os valores utilizados e os resultados obtidos. Ao analisar esses resultados, poderemos avaliar a robustez e a correta funcionalidade do nosso conversor de *Pug* para *HTML*.

4.1 Alternativas, Decisões e Problemas de Implementação

Neste subcapítulo, abordaremos os problemas encontrados durante o desenvolvimento do nosso conversor e como conseguimos resolvê-los.

A maioria dos desafios surgiu na etapa inicial do trabalho, quando decidimos adotar *tokens* muito específicos. Essa abordagem acabou por complicar bastantes processos, como o reconhecimento de *tags*. Para superar esta dificuldade, introduzimos um estado no *Lexer* em vez de usar uma simples *flag*, como tínhamos feito inicialmente. Com essa nova metodologia, conseguimos evitar a execução de funções indesejadas no início da linha, o que resolveu a maioria dos problemas. Além disso, arranjámos também uma estratégia para a criação dos *divs*. A tag div, em Pug, não precisa de estar explicitamente escrita e no início era a gramática que controlava isto, no entanto, esta estratégia aumentava bastante o número de produções. Assim, no lexer, passou-se a detetar os casos em que era criada a tag div (inicio com uma classe ou um id identificador) e, aí é gerado um token TAG, com valor 'div'.

Além disso, outro problema que estava a surgir, vinha do facto de quando havia uma mudança de linha e a identação era alterada por mais de um valor, apenas estava a ser criado um token DEDENT, que não era o resultado desejado. Teve de ser criado um novo estado que permite criar vários tokens para uma dedentação maior que um.

Um dos desafios, que rápidamente foi resolvido, foi o de ignorar linhas apenas com espaços ou tabs, visto que estas estavam a influenciar a identação. Para tal, foi usado o módulo re:

```
re.match(r"[ \t]*\n",line)
```

Caso desse match, era ignorada a linha.

Outro desafio que tivemos, mas não houve tempo de resolver, foi no uso de variáveis em ciclos, por exemplo, visto que sendo uma gramática Bottom-up, o conteudo dos ciclos era calculado antes do ciclo ser corrido propriamente e, assim, não era possível ir alterando o resultado do mesmo. Por exemplo, os ciclos each apenas funcionam com conteúdo estático.

Finalmente, e tal como o problema anterior, nos else ifs, o valor destes era calculado antes do if e caso alguma das variáveis na condição não existisse, era enviada uma exceção. No entanto, essa exceção apenas devia ser enviada se não se verificasse a condição do if, para tal, no if é apanhada a exceção e só é lançada caso a condição do if não se verifique.

4.2 Testes realizados e Resultados

Para efetuar testes no nosso código, utilizamos não só o teste fornecido pelo professor, mas também criamos vários outros testes para verificar funcionalidades adicionais. Inicialmente, certificámos-nos de que o código funcionava corretamente para o teste fornecido. Em seguida, à medida que implementávamos novas funcionalidades, criávamos ficheiros de teste para verificar as mesmas. Além disso, à medida que eram acrescentadas novas funcionalidades, todos os testes eram corridos novamente, com o objetivo de verificar que nada parou de funcionar com as novas adições. Dessa forma, garantimos que todas as funcionalidades do conversor de *Pug* para *HTML* foram testadas e integradas de maneira adequada, assegurando o correto funcionamento do código como um todo.

```
- var pageTitle = "My Pug File"
- var showDetails = 1
- var loggedIn = 1

doctype html
html
    head
        title= pageTitle
        meta(charset='UTF-8')
        link(rel='stylesheet', href='w3.css')
    body
        header
            h1 Welcome to pageTitle
        section
            h2 About
            p This is a sample Pug file for testing purposes.

            if showDetails
                p Additional details are shown.

            else
                p No additional details are available.

        section
            h2 Features
            ul
                each c in [1,2,3,4,5]
                    li item
        if loggedIn
            section
                h2 User Dashboard
                p Welcome, username!

    footer
        p Copyright © currentYear. All rights reserved.
        p Contact: email
```

(a) Pug

Welcome to pageTitle

About

This is a sample Pug file for testing purposes.

Additional details are shown.

Features

- item
- item
- item
- item
- item

User Dashboard

Welcome, username!

Copyright © currentYear. All rights reserved.

Contact: email

(b) HTML

Figura 4.1: Um dos testes

Capítulo 5

Conclusão

Em suma, o nosso grupo considera que, embora não tenhamos conseguido implementar todas as funcionalidades do *Pug*, conseguimos desenvolver um conversor eficaz de *Pug* para *HTML*, que foi capaz de lidar com os exemplos fornecidos pelo professor, bem como diversas outras funcionalidades. Durante o processo, enfrentamos desafios significativos, mas fomos capazes de superá-los e entregar um produto funcional. Reconhecemos que há espaço para melhorias e expansões futuras, mas estamos satisfeitos com os resultados alcançados até o momento. Através deste projeto, aprofundamos nosso conhecimento de processamento de linguagens e adquirimos habilidades valiosas na implementação de conversores de linguagens. No geral, estamos orgulhosos do nosso trabalho e confiantes de que cumprimos os objetivos propostos, obtendo um resultado satisfatório.

Bibliografia

[PLY] Ply (python lex-yacc). https://www.dabeaz.com/ply/ply.html.

[Pug] Getting started - pug. https://pugjs.org/api/getting-started.html.