

Trabalho Prático Nº2 - Compiladores

Marco Sousa^{1,2[62608]}, José Malheiro^{1,2[93271]}, and Miguel Fernandes^{1,2[94269]}

¹ Universidade do Minho

² Licenciatura em Engenharia Informática, Braga, Portugal

Abstract. Keywords: Expressões regulares · Python · Templates · Gramática Independente de Contexto · Compiladores.

1 Introdução

1.1 Contextualização

O presente relatório foi escrito no âmbito da Unidade Curricular (UC) de Processamento de Linguagens (PL), apresentando como objetivo a descrição da solução desenvolvida para o segundo trabalho prático, assim como as decisões tomadas para a sua conceção.

A solução é relativa ao primeiro enunciado: **Linguagens de Templates** (inspirada nos *templates Pandoc*).

Neste sentido, é proposto o desenvolvimento de um compilador de *templates*, de modo a criar ficheiros *markup*, a partir do processamento de *templates* pré-definidos. Para tal usou-se a linguagem *YAML* para guardar toda a informação a ser usada no preenchimento do *template*.

Assim, engloba o desenvolvimento do **analisador léxico** e o **parser**, bem como a gramática e a estrutura da árvore sintática criadas para poder gerar o texto final.

1.2 Breve Descrição do Trabalho Proposto

O enunciado escolhido, **Linguagem de templates** (inspirada nos *templates Pandoc*), propõe a criação de um compilador de *templates*. Deste modo, utiliza-se:

1. um *Template*.
2. um Dicionário.

...para construir o texto final, um **ficheiro *markup***, com todos os dados submetidos pelo utilizador.

Os *templates* a usar devem incluir um conjunto de regras, para permitir substituir os dados fornecidos pelo utilizador nos parâmetros desejados. Neste sentido, propõe-se o desenvolvimento da **gramática** que cada *template* deve seguir, independentemente da linguagem que está a utilizar.

Para o desenvolvimento do **compilador** segue-se a criação de um:

- Analisador Léxico.
- *Parser*.

...que conjuntamente permitem capturar os campos de **metadados** dos *templates*, para serem posteriormente tratados. Para tal deve-se recorrer ao módulo *PLY* do *Python*, nomeadamente o gerador de analisador léxicos - *Lex* - e o gerador de compiladores (com base em gramáticas tradutoras) - *Yacc*.

No final, após o documento ser corrido na sua totalidade, o texto final está completo.

Os dados a serem inseridos no *template* encontram-se num ficheiro à parte, um **dicionário**, que será submetido aquando da execução do programa. No projeto a realizar utilizar-se-á ficheiros *YAML*, usufruindo da organização estrutural da linguagem.

2 HandleDoc

2.1 Gramática Criada

Com o intuito de descrever a estrutura hierárquica da linguagem definida para os *templates*, construiu-se a **gramática** a usar no compilador.

Como o programa na íntegra dependia das decisões tomadas durante a definição da gramática, foi o primeiro passo no processo de desenvolvimento da solução.

Usando a biblioteca *PLY* para facilmente aplicar as regras sintáticas, a **gramática independente de contexto** (GIC) usada no âmbito do presente projeto encontra-se em 4.1.

Mediante os símbolos **terminais**, **literais** e as palavras **reservadas** definidas no contexto do analisador léxico do compilador, em 2.2, seguem-se as **produções**:

Document Correspondente à primeira produção criada no contexto gramatical, *Document* caracteriza o *template* na sua totalidade, com todos os elementos constituintes(?). A produção dada à sua simplicidade permaneceu imutável desde o início do desenvolvimento do projeto até ao fim, sendo todas as alterações ao nível atómico das produções.

Elements Os elementos que compõem um documento são caracterizados como uma lista de elemento(2.1). Não é do interesse do projeto tratar *templates* vazios, pelo que é obrigatório um documento ter **pelo menos** um elemento constituinte, seja qual for.

Element Componente atómico do documento. Um elemento pode ser do tipo:

- *Statement* (Stmt)
- Texto (TEXT)
- Variável (Var)
- *Backslash* (BACK)
- Iterável (It)
- *Nesting* (Nesting)

...

2.2 Analisador Léxico

- máquina de estados

2.3 Estratégia Utilizada

- estratégia utilizada – árvore de parsing concreta – classes – hierarquia – descentralização das responsabilidades – diagrama de classes

- Funcionalidades – if – elseif – else

2.4 Ifs

A condição *if* começa com a palavra reservada *if* seguida de uma variável entre parênteses(dentro de '\$'). Tem a possibilidade ainda de ser sucedido por um ou mais *elseif*, também com uma variável entre parênteses(dentro de '\$') ou então um *else*(dentro de '\$').

```
$if(example)$
    $example$
$endif$

$if(example1)$
    $example1$
$elseif(exemple2)$
    $example2$
$else$
    Lorem ipsum
$endif$
```

– for

2.5 For

Um ciclo *for* é iniciado pela palavra reservada *for* seguido de uma variável entre parênteses(dentro de '\$'), terminando com um *endfor*(dentro de '\$'). Caso a variável a iterar seja:

- **Lista:** Os elementos são percorridos ordenadamente, sendo a variável definida para cada elemento.
- **Dicionário:** As chaves são percorridas ordenadamente, sendo a variável definida para o valor correspondente...
- **Texto:** O ciclo só irá realizar uma única iteração.

```
$for(example)$
    $example.ex1$, $example.ex2$
$endfor$

$for(list)$ $list$ $endfor$
```

Caso se pretenda utilizar um separador entre valores consecutivos numa iteração do ciclo *for*, é possível utilizar o *sep*(dentro de '\$'), indicando o separador a utilizar. Por exemplo: Caso queiramos iterar a lista:

```
autores = ["José", "Marco", "Miguel"]
no seguinte template:
$for(autores)$autor = $autores$$sep$, $endfor$
Será produzido:
autor = José, autor = Marco, autor = Miguel
```

Estratégia utilizada A classe *for* é inicializada com a variável a iterar, a lista de elemento que compõe o corpo do *for* e o separador no caso de este ser utilizado.

Uma vez que a forma de iterar o ciclo, depende do tipo da variável, tal como já foi explicado anteriormente, foram criadas 3 funções: *handleStr*, *handleList* e *handleDict*...

Em todas as funções é percorrido a lista de elementos, sendo estes imprimidos consoante o valor a ser iterado. No caso das listas é passado o elemento e também a condição, desta forma, caso no corpo apareça a variável da condição pretende-se imprimir apenas o valor e não a lista inteira.

FALTA FALAR DOS MAPAS

– pipes

2.6 Pipes

Tal como no *Pandoc*, foram criados *Pipes* que transformam o valor de uma variável ou de um *Partial*.

Os *Pipes* são especificados com o uso de uma barra (/), depois do nome da variável ou do *partial*. Por exemplo:

```
$author/uppercase$

$for(authors/reverse)$
  $it.name$
$endfor$

$subtemplate()/lowercase$
```

Os *Pipes* também podem ser encadeados:

```
$author/uppercase/reverse$
```

Quanto aos *Pipes* criados, foram feitos todos os que o *Pandoc* tem implementados até ao momento da realização do presente relatório, mais concretamente:

- *pairs*: Converte um dicionário ou uma lista para um dicionário de listas. Caso o valor seja uma lista, a chave corresponde ao índice do array, começando em 1.
- *uppercase*: Converte um texto todo para letras maiúsculas.
- *lowercase*: Converte um texto todo para letras minúsculas.
- *length*: Retorna o tamanho do valor, no caso de um valor textual o número de caracteres, numa lista ou dicionário o número de elementos.
- *reverse*: Inverte um valor textual ou uma lista.
- *first*: Retorna o primeiro elemento de uma lista.
- *last*: Retorna o último elemento de uma lista.
- *rest*: Retorna o todos os elementos de uma lista menos o primeiro.
- *allbutlast*: Retorna o todos os elementos de uma lista menos o último.
- *chomp*: Substitui *newlines* seguidos por apenas um.
- *nowrap*:
- *alpha*: Em valores textuais, converte números inteiros, entre 1 e 26, para a correspondente letra minúscula.

- *roman*: Em valores textuais, converte número inteiros para números romanos minúsculos.
- *left n "leftboarder" "righthboarder"*: Coloca um valor textual num bloco de tamanho *n*, alinhado à esquerda, com um limite opcional à esquerda e à direita. Caso só seja passado um limite, por defeito assume-se que se trata do limite à esquerda.
- *right n "leftboarder" "righthboarder"*: Coloca um valor textual num bloco de tamanho *n*, alinhado ao centro, com um limite opcional à esquerda e à direita. Caso só seja passado um limite, por defeito assume-se que se trata do limite à esquerda.
- *center n "leftboarder" "righthboarder"*: Coloca um valor textual num bloco de tamanho *n*, alinhado à direita, com um limite opcional à esquerda e à direita. Caso só seja passado um limite, por defeito assume-se que se trata do limite à direita.

Estratégia utilizada A classe *Pipes* é inicializada com uma lista de objetos da classe *Pipe*. Tem ainda uma função, *handlePipes*, que aplica iterativamente cada *pipe* ao valor ou *partial* recebido, retornando o valor transformado.

Na classe *Pipe* foi criado um dicionário, em que a chave é o nome do *pipe* e o valor é a função corresponde.

```
_pipes = {
    'pairs'      : lambda x : pairs(x),
    'uppercase'  : lambda x : x.upper(),
    'lowercase'  : lambda x : x.lower(),
    'length'     : lambda x : len(x),
    'reverse'    : lambda x : reverse(x),
    'first'      : lambda x : first(x),
    'last'       : lambda x : last(x),
    'rest'       : lambda x : rest(x),
    'allbutlast' : lambda x : allbutlast(x),
    'chomp'      : lambda x : chomp(x),
    'nowrap'     : lambda x : nowrap(x),
    'alpha'      : lambda x : alpha(x),
    'roman'      : lambda x : roman(x),
    'left'       : lambda x, n, l, r : left(x, n, l, r),
    'center'     : lambda x, n, l, r : center(x, n, l, r),
    'right'      : lambda x, n, l, r : right(x, n, l, r)
}
```

Desta forma, a função responsável por aplicar o *pipe* ao valor apenas precisa de fazer:

```
Pipe._pipes[self.id](args)
```

...sendo o *self.id* o nome do *pipe* e *args* os argumentos passados, podendo variar de 1 a 4.

– subtemplating

2.7 subtemplating

– range

2.8 Range

– it

– sep

2.9 Separator

2.10 Manual de Ajuda

CONSTRUIR MANUAL!!

3 Conclusões

4 Anexos

4.1 Gramática

```
#####
#                               #
#####

Document -> Elems

#####
#                               #
#####

Elems    -> Elems Elem
Elems    |   Elem

#####
#                               #
#####

Elem     -> Stmt Newline
          | TEXT NewLine
          | Var NewLine
          | BACK NewLine
          | It
          | Nesting

#####
#                               #
#####

Stmt     -> If
          | For
          | Subtemplate

#####
#                               #
#####

If       -> IF OPAR Cond CPAR Newline Elems Else ENDIF

#####
#                               #
```

```

#####

Else    -> ELSE Newline Elems
        |   ElseIf Else
        |   Empty

#####
#           ElseIf           #
#####

ElseIf  -> ELSEIF OPAR Cond CPAR Newline Elems

#####
#           ForStmt           #
#####

For      -> FOR OPAR Cond CPAR Newline Elems Sep ENDFOR

Sep      -> SEP TEXT
        |   Empty

#####
#           Subtemplate           #
#####

StmtSubtemplate  -> Var OPAR CPAR Pipes Newline
SubIt             -> VarAtomic OPAR CPAR Pipes

#####
#           IT           #
#####

It        -> IT ItOpt Newline

ItOpt     -> COLON SubIt
        |   DOT Var COLON SubIt
        |   DOT Var
        |   DOT Var OSQBRAC NUM COMMA NUM CSQBRAC
        |   OSQBRAC NUM COMMA NUM CSQBRAC
        |   Empty

#####
#           Nesting           #
#####

Nesting -> Var TEXT '^' NestElems '^' Newline

NestElems -> NestElems
        |   NestElem

NestElem  -> Var
        |   TEXT
#####

```

```

#                               Conds                               #
#####

Cond    -> Var
        | It

#####
#                               Vars                               #
#####

Var -> VarAtomic Pipes

VarAtomic -> VarAtomic DOT ID
          | ID

#####
#                               Pipes                               #
#####

Pipes    -> Pipes SLASH ID
          | Pipes SLASH ID NUM QUO TEXT QUO
          | Pipes SLASH ID NUM QUO TEXT QUO QUO TEXT QUO
          | Empty

#####
#                               Num                               #
#####

Num      -> NUM
          | Empty

#####
#                               Newline                             #
#####

Newline  -> NL
          | Empty

```
