Processamento de Linguagens
Trabalho Prático Nº1 - FLEX

Ana Catarina Sousa
a78029

Eduardo Jorge Barbosa
a83344

(31 de Março de 2019)

Resumo

O *FLEX* é uma ferramenta extremamente versátil que nos permite gerar programas que fazem *pattern matching* em texto. De forma a demonstrar o poder desta ferramenta, desenvolveu-se um programa que utiliza um analisador léxico gerado em *FLEX* para processar um conjunto de artigos. Estes artigos podem ser convertidos numa panóplia de formatos e é feita uma breve analise de como se interligam.

1 Introdução

A análise de texto, e consequente manipulação, é uma das tarefas mais comuns na informática. Analisando de forma espartana, pode-se mesmo argumentar que toda esta área tem por base a análise e manipulação de informação. No entanto, o desenvolvimento de analisadores léxicos com recurso a linguagens imperativas e/ou orientadas a objectos, tais como o *C* ou *C++* é um processo moroso, complexo e muito susceptível a erros. É neste contexto que surge o *FLEX*, um gerador de analisadores léxicos que adopta uma abordagem mais declarativa, mudando assim o foco para a especificação de padrões e a respectivas acções a realiza. Esta abordagem é extraordinariamente vantajosa para o programador, visto que muda o foco da implementação do mecanismo de detecção dos padrões para a implementação dos padrões que devem ser detectados e as acções a serem executadas. Todos os problemas introduzidos pela necessidade da implementação do analisador em si são postos de parte. O programador fica então apenas preocupado em expressar os padrões que devem ser encontrados, podendo dar-se ao luxo de esquecer a manutenção do estado do analisador, *off by one erros*, leitura correcta do *input*, e tantos outros problemas tradicionais. O *FLEX* utiliza um conjunto de *expressões regulares* para dar *match* com o texto pretendido.
Com o objectivo de demonstrar o uso do *FLEX* foi implementado um programa que analisa artigos de um jornal angolano. Por cada artigo pode ser gerado uma versão do mesmo em formato *HTML*, *TeX*, ou *Markdown*. Cada artigo é pré-processado antes de ser criado no novo formato. Visto que os artigos possuem

tags, é possível pesquisar artigos por *tag* e são gerados grafos que mostram como cada *tag* se liga às outras.

1.1 Estrutura do Relatório

O relatório encontra-se dividido nos seguintes capítulos:

1. Introdução;
2. Background;
3. Conceitos básicos;
4. Proposta;
5. Implementação;
6. Conclusão.

No capítulo 2, é apresentado o conceito por detrás das expressões regulares e como estas se ligam aos autómatos computacionais. É também abordado um pouco sobre como estes conceitos se relacionam com o *FLEX*.
No capítulo 3, descrevem-se alguns conceitos fundamentais sobre o *FLEX*.
No capítulo 4 é analisado o que foi pedido implementar e o foco do grupo. Finalmente, no capítulo 5 é abordada a arquitectura da solução, e a análise necessária para a sua implementação.
O capítulo 6 deixa algumas conclusões sobre o trabalho desenvolvido e trabalho futuro.

2 Background

Como já foi referido, o *FLEX* utiliza um conjunto de expressões regulares para realizar *pattern matching*.
Informalmente, uma expressão regular pode ser descrita como um pedaço de texto que descreve um padrão. De uma forma mais formal, uma expressão regular permite descrever uma *linguagem regular*, sendo então extremamente útil para descrever padrões.
Além do seu poder inato, as expressões regulares possuem uma álgebra, sendo possível combina-las de diversas formas.
Um conjunto finito de símbolos, Σ, é denominado por um **alfabeto**. Uma **palavra** deste alfabeto Σ é uma sequência finita de letras de Σ. Uma colecção de palavras de Σ é denotada por Σ^*. Uma **linguagem formal** de Σ é um subconjunto de Σ^*, e uma **linguagem regular** de Σ, é uma linguagem formal de Σ que é aceite por um **autómato determinístico finito.**
Em cima foram referidos autómatos determinísticos finitos e foi dado a entender que estes relacionam-se com as expressões regulares, *AFDs*. Informalmente

pode-se pensar num *AFD* como uma máquina de estados que aceitam, ou rejeitam, texto. À medida que vão consumindo pedaços do texto, vão alternando entre estados. Esta transição entre estados é fixa e definida à priori. Para cada expressão regular existe um *AFD*, o reciproco também é verdade. Como qualquer linguagem regular é aceite por um *AFD*, existe uma expressão regular para a descrever.
A tradução de uma expressão regular para um *AFD* permite optimizar o processamento do texto, visto que cada símbolo é lido apenas uma vez. Sendo o *FLEX* um gerador de analisadores léxicos, este utiliza expressões regulares para descrever que padrões de texto procurar e a acção respectiva caso seja encontrado. Internamente o conjunto de expressões regulares descritas pelo programador é traduzido num *AFD*.

3 Conceitos básicos de FLEX

De forma simples, o *FLEX* gera *scanners* que lêem input de um ficheiro, analisam o texto face à um conjunto de regras e executam as devidas acções.

3.1 Regras

As regras são um conjunto de expressões regulares que denotam os padrões a procurar. Seguem a forma:

```
padrão   acção
```

3.2 Definições

A secção das definições contem declarações de expressões regulares, de forma a simplificar as especificações, e declarações de *start conditions*.

```
definições
%%
regras
%%
```

3.3 Definições de expressões regulares

É possível dar nome a expressões regulares de forma a simplificar o *scanner*.

```
DIGITO    [0-9]
ID        [a-z][a-z0-9]*
```

3.4 Start conditions

De forma a activar regras de forma condicional, o *FLEX* oferece um mecanismo chamado *start conditions*. Acrescentando um prefixo a certas regras, estas só

irão ser activadas quando o *scanner* se encontrar nessa *start condition*.

```
%x example
%%

<example>foo   { do_something(); BEGIN INITIAL; }
<INITIAL,example>bar    { something_else(); }
<*>.*|\n { ; }
```

Pode-se então observar que as *start conditions*, exclusivas, são declaradas nas definições. É possível enumerar estados nos prefixos e utilizar o caractere * para denotar qualquer estado. Para entrar numa *start condition* utilizamos a instrução `BEGIN <condition>`.

3.4.1 Interesse

As *start conditions* aparecem para superar uma limitação conhecida das expressões regulares, lidar com níveis aninhados.

4 Proposta

Neste trabalho prático foi proposto a análise e processamento de vários artigos de um jornal angolano. Cada artigo segue o seguinte formato:

```
<pub>
#TAG: tag:{Eduardo dos Santos} tag:{Petróleo} tag:{mensagem} tag:{preços}
#ID:{post-6243 post type-post status-publish format-standard has-post-thumbnail hentry
category-nacional tag-eduardo-dos-santos tag-petroleo tag-mensagem tag-precos}
Nacional

2015 será um ano difícil, diz o Presidente. Igual aos outros, acrescenta o Povo
--------------------------------------------------------------------------------

PARTILHE VIA:
#DATE: [116eb] Redacção F8 - 29 de Dezembro de 2014
2015 será um ano difícil, diz o Presidente. Igual aos outros, acrescenta o Povo - Folha 8

A baixa no preço do barril de petróleo, verificada desde Junho, está a levar o Executivo de Eduardo
dos Santos a traçar estratégias para contornar as dificuldades desencadeadas. Ou seja, com o preço
do petróleo em alta ou em baixa, serão sempre os mais pobres a pagar a factura.
```

Artigo

4.1 Descrição Informal do Problema

Com este projecto, temos o objectivo de limpar os artigos lidos sem perder *meta-dados*. Isto significa processar não só o texto do artigo, como também o *id*, o *link* original, a categoria, as tags, entre outros. Além disso é requerido criar uma lista de abreviaturas por cada artigo.
O grupo decidiu formatar cada artigo lido, apresentado-os de forma mais elegante, como também permitir ao utilizador o formato do ficheiro resultante. O

utilizador tem a liberdade de escolher entre o formato *HTML*, *Markdown*, e/ou *TeX*. Por omissão é escolhido o formato *HTML*. Neste formato também é gerado um ficheiro *index*, à la *ezines*, que permites navegar entre os artigos por **tag** ou por **titulo**.

Articles per title

Articles per tag

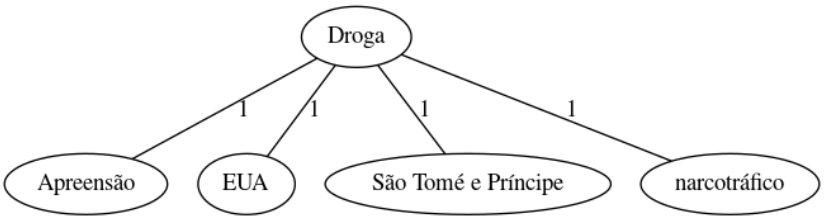
Index

Em cada *tag* é possível ver quantos artigos a possuem e navegar para os mesmos, é também apresentado um grafo que relaciona a *tag* com outras *tags* presentes em artigos que a contenham.

Number of articles: 1

Articles:

100 milhões de dólares em droga



Tag Droga

5 Implementação

5.1 Analisador Léxico

O analisador descrito em FLEX alterna entre 8 *start conditions*:

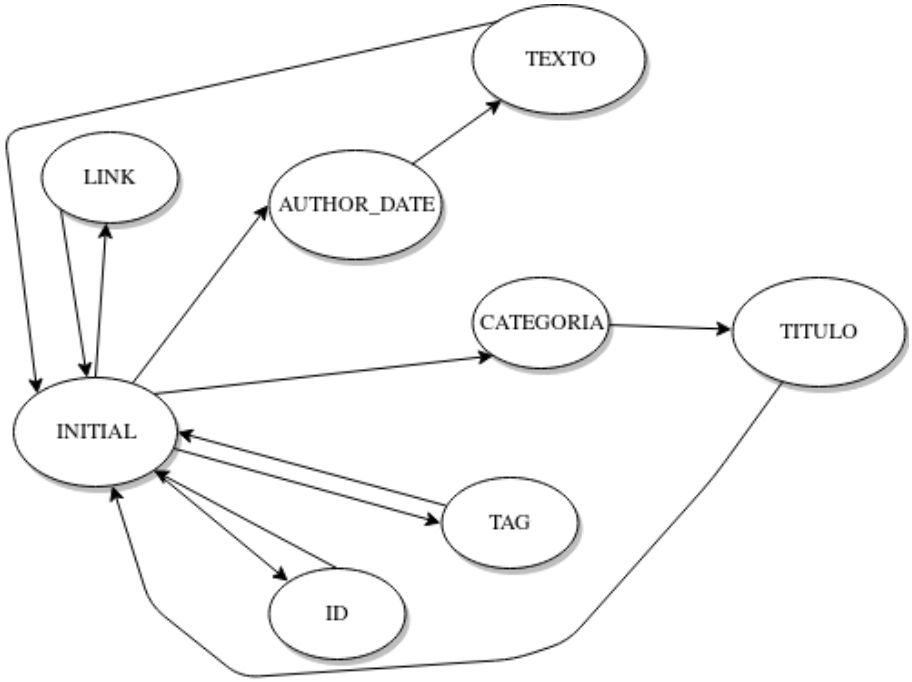
- LINK;
- TAG;
- ID;
- CATEGORIA;
- TITULO;
- TEXTO;
- AUTHORDATE.

Pode parecer contra intuitivo não existir uma *start condition* relativa ao artigo, mas visto que as expressões que permitem entrar em cada estado são tão distintas, tal não foi necessário.
Um artigo é delimitado inicialmente pelo seu *link* original, sendo este guardado. A parte interessante do link encontra-se num género de *tag HTML*, sendo então muito fácil a sua captura. A parte interessando está rodeada de símbolos `-`. De seguida é pretendido capturar as *tags* do artigo. Cada *tag* está contida na seguinte forma `tag:<TAG>`. O padrão `tag:{` denota a entrada no estado *TAG* e o } a saída.
Maior parte da informação é recolhida desta forma, alternando entre a *start condition INITIAL* e a correspondente ao pretendido. Uma excepção será o texto do artigo em si. A *start condition TEXT* é inicializada logo após a *AUTHORDATE*. O texto apresentou alguns desafios interessantes. De forma a facilitar a formatação o texto é lido linha a linha, isto é, até encontrar um *newline*. A expressão regular é simplesmente

```
.+\n
```

Sendo o *FLEX greedy and longest match*, isto vai consumir todo o input, perdendo a oportunidade de procurar qualquer abreviatura. Felizmente o *FLEX* oferece formas de lidar com isto. Utilizando as primitivas *REJECT* e *yymore*, é possível rejeitar o melhor padrão, fazer *match* com o segundo melhor padrão e manter o input original. Infelizmente isto requer a utilização de uma variável global visto que um input vai ser lido tantas vezes quantos caracteres tiver.

5.1.1 Autómato Finito Determinístico



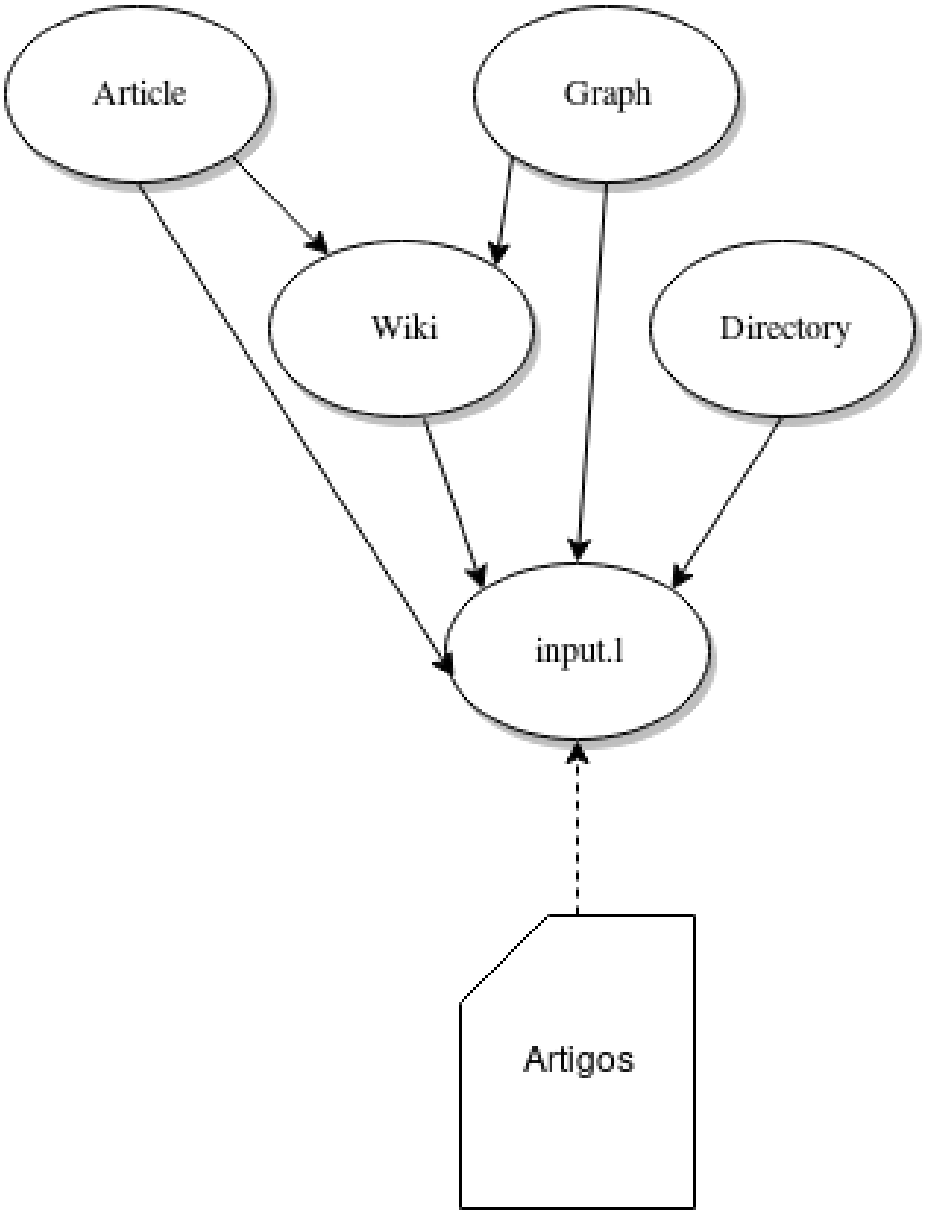
ADF

5.2 Estruturas auxiliares

De forma a facilitar o processamento da informação, após esta ter sido recolhida, foram criadas estruturas auxiliares.

1. Article.cpp;
2. Wiki.cpp;
3. Graph.cpp;
4. Directory.c.

A classe *Article* define o artigo, sabendo representar cada artigo nos vários formatos.
A classe *Wiki* define um conjunto de artigos, sendo responsável por relacionar as *tags* entre cada artigo, utilizando a classe Graph. Finalmente, o módulo *Directory* é responsável por criar a estrutura de pastas necessária ao bom funcionamento do programa.



Arquitetura

6 Conclusão

O grupo considera que o *FLEX* é uma escolha adequada para o processamento de texto, sobretudo devido à rapidez com que permite desenvolver analisadores léxicos. É também de realçar o seu poder expressivo proveniente do uso de expressões regulares e das*start conditions*, ultrapassando assim algumas limitações das mesmas. No entanto, algumas falhas são bastante notórias, nomeadamente a natureza greedy e a impossibilidade de atribuir graus de prioridade a cada expressão regular. Também no domínio das expressões regulares, convém notar que existem ferramentas que oferecem um conjunto mais rico e poderoso destas expressões. O grupo considera que fez um trabalho positivo face ao pedido, e conseguiu demonstrar grande parte do poder expressivo do *FLEX*, utilizando-o em conjunto com a linguagem de programação *C++*.