Universidade do Minho

Licenciatura em Engenharia Informática

Processamento de Linguagens
TP2 - PLY-Simple para PLY

Grupo 66

Diogo Casal Novo (a88276) João Lourenço (A93233)
João Ferreira (A93263)

maio, 2022

Conteúdo

Capítulo 1

Introdução

Uma das partes fundamentais da para o funcionamento dos computadores de hoje em dia é a análise sintática e interpretação de inputs. A técnica conhecida como parsing é a segunda fase do processo da compilação de um programa, precedido da análise lexical, onde uma sequência de caracteres é convertida numa sequência de tokens.

Este trabalho vai permitir criar um compilador que converta uma linguagem nova chamada *PLY-Simples* numa outra mais complexa, *PLY*. O nosso objetivo é simplificar a forma de escrita e retirando partes repetitivas e que possam ser automatizadas mas, ao mesmo tempo, tentando evitar restringir as possibilidades que o *PLY* fornece.

Capítulo 2

Análise do Problema

2.1 Descrição Informal do Problema

O gerador de parsers *PLY*, embora poderoso tem uma sintaxe um pouco complexa. Pretendemos criar um tradutor que a partir de uma sintaxe mais "limpa" *PLY-Simple*, gere as funções *PLY* convenientes.

2.2 Especificação

Do enunciado referido na secção anterior é possível levantar um conjunto de requisitos que o projeto deverá cumprir. O grupo, no entanto, decidiu ir mais além, pelo que foi levantado um conjunto de requisitos adicionais que tem o objetivo de complementar o funcionamento da ferramenta e permitir uma maior liberdade na escrita.

2.2.1 Requisitos base

Os requisitos base foram obtidos através da análise do enunciado, presente de forma resumida na secção 2.1. Estes são de seguida enunciados.

1. Deduzir um esquema de tradução *PLY-Simple* em *PLY*
2. A linguagem *PLY-Simple* deverá ter os comando para o *Yacc* e para o *Lex* no mesmo ficheiro.
3. A linguagem *PLY-Simple* deverá mecanizar a informação que é repetitiva ao longo de todas as definições sendo que apenas deverá ser escrito o essencial para o programa.
4. Permitir sempre que possível que funções escritas em *Python* possam também ser escritas de uma maneira idêntica em *PLY-Simple*.

2.2.2 Requisitos adicionais

Embora os requisitos acima mencionados suficientes para dar à aplicação bastantes capacidades, tal como dito anteriormente, o grupo decidiu incluir mais algumas de forma a permitir uma tolerância maior na escrita do programa em *PLY-Simple*. Estas novas capacidades são:

1. Permitir que *literals* sejam declarados como uma *string* de caracteres em que cada carácter simboliza um símbolo (incluindo caracteres como *new line, tab, backslash*), ou como uma lista tal como no caso dos *tokens* até porque este é o formato usado no *PLY*;
2. Permitir que seja possível declarar uma *string* quer no formato *f"<conteúdo>"* quer no formato *"<conteúdo>"* sendo que para o segundo caso a string possa incluir um ou mais argumentos separados pelo operador *'+'* que também deverão ser interpretados;
3. Dar liberdade nas declarações das gramáticas sendo que estas deverão possuir tanto a capacidade de interpretar declarações de instruções com uma ou mais linhas e não apenas uma linha como proposto pelo enunciado.
4. Permitir que se possa colocar comentários de linha em qualquer parte do código no ficheiro *PLY-Simple*
5. Detetar e avisar de alguns tipos de erros relacionados quer com o *Lex*, quer com o *Yacc*. Apenas erros que não permitam que o programa resultante da compilação execute devem impedir a sua criação sendo os restantes marcados apenas como *Warnings*.

Capítulo 3

Resolução

3.1 Abordagem

Sendo este um compilador de *PLY-Simple* para *PLY* tivemos como objetivo passar diretamente de uma lingugem para a outra. *PLY-Simple* é uma linguagem com uma estrutura bastante rija.

1. O programa deverá começar por uma marca para iniciar o *Lexer*, seguido das variáveis *literals*, *ignore* e *tokens*, podendo estas aparecer em qualquer ordem, e finalmente as defenições dos *tokens* anteriormente declarados;
2. Seguidamente deverá aparecer uma nova marca para iniciar o *Yacc* e as definições das varias regras da gramatica;
3. Finalmente devera haver uma parte final marcada apenas por %% que servirá para colocar tudo o que não puder se definido antes;

3.2 Lexer

3.2.1 Variáveis

Há três variáveis a serem definidas: *tokens*, *literals* e *ignore*. As variáveis têm de ter um caracter % a antecede-las.

Os *tokens* deverão ser declarados como uma lista de strings tal como no PLY, isto é, uma lista é delimitada por parênteses retos e cada elemento deverá estar separado por virgulas e entre plicas.

Os *literals* possuem duas formas de serem declarados sendo uma da mesma forma que os *tokens* e a outra usando uma string. Em ambas caracteres como especiais que usam \ (por exemplo o '\n') são aceites. Isto é feito com base numa expressão regular e o uso do *findall* aplicada à string lida.

Os caracteres a ignorar são definidos pelo *ignore* e são representados como uma string da mesma maneira que os *literals*.

Estas variáveis são interpretadas e guardadas num dicionário *parser.lex* para posteriormente serem usadas na criação do ficheiro. São ainda guardados os literals e os tokens num outro dicionário *parser.vars* para depois poderem ser utilizados para correção de erros.

3.2.2 Definições dos tokens

A seguir às variáveis vem as definições dos *tokens* anteriormente declarados. Estes são compostos por três campos sendo que os últimos dois estão entre parênteses, separados por uma virgula e antecedidos pela palavra "*return*":

Regex: o primeiro campo é uma expressão regular escrita como *r'<conteúdo>'*.

Nome da função: um dos nomes previamente definidos nos tokens.

Ação: este campo deverá ter a ação a ser feita pelo *lexer* e vai ser colocado após a expressão regular. Este pode tomar uma de três formas:

- vazio onde é colocado apenas "*return t*";
- *pass* onde apenas é colocado *pass*;
- um contudo ou uma função onde é colocado "*t.value = <conteúdo>* "seguido de "*return t*";

É ainda apanhado o caso de erro do *Lexer* que é dado por um ponto no inicio da linha, seguido da palavra "*error*"e, entre parênteses, uma string e uma ação para fazer em caso de erro.

O conteudo desta parte é guardado no dicionario *parser.lex* juntamente com as variáveis.

3.3 Yacc

O conteudo desta secção esta dividido em duas partes e, após ser interpretado, ira ser guardado na string *parser.yacc*

3.3.1 Gramática

Esta parte vem a seguir ao símbolo *YACC* e tem duas partes sendo a primeira a declaração das variáveis que pertencem ao *parser*, , e a segunda parte que é uma sequência de regras que compõem a gramática.

Para a primeira parte é opcional já que pode não ser necessário declarar nenhuma variável. Todas as declarações são sinalizadas com um % no início da linha

A segunda parte é composta por dois componentes: a regra e o conjunto de instruções dessa mesma regra.

Os nomes e símbolos usados nas regras podem ser *tokens*, *literals* ou nomes que não constam em nenhuma delas, mas, para o ultimo caso, estes são guardados no dicionário *parser.vars* para depois poderem ser utilizados para correção de erros.

O conjunto de instruções que se segue pode ser definido na própria linha ou pode ter varias linhas sendo apenas delimitado pelas chavetas.

3.3.2 Main

A ultima parte do ficheiro é separada por %% da parte anterior e contem um bloco onde pode ser definido qualquer função auxiliar e que não esteja diretamente ligado ao *PLY* pois o conteúdo deste bloco não é analisado.

3.4 writeFile

A função *writeFile* é responsável por criar e colocar os conteúdos nos ficheiros usando como base a informação previamente intrepretada e guardada no dicionário *parser.lex* e na string *parser.yacc*. Esta função só é executada caso o processo de *parse* tenha obtido sucesso.

Para que os ficheiros fiquem completos e possam ser executados ainda é necessário acrescentar algumas informações no topo e no final de dos mesmos como é o caso dos *import's*, ação que esta função também é responsável por fazer. Acrescentando a isto, esta função ainda ainda é responsável por reorganizar a informação vinda do *Lex* colocando os *tokens*, *literals*, definições dos *tokens* e finalmente os *ignore*.

3.5 Deteção de erros

A deteção e resposta a casos de erro é feita ao longo do processo de parsing. Para isso é utilizado o dicionário *parser.vars* para guarda as nomes utilizados como variaveis na gramatica, juntamente com os *tokens* e *literals* declarados e utilizados ao longo do processo.

São apanhadas algumas situações de erro e algumas situações em que o código fornecido poderá não estar correto sendo nessas apenas lançado um aviso.

As situações de erro obrigam a que o programa não acabe corretamente o que impede que sejam gerados os ficheiros **_lex.py* e **_yacc.py*. Estas são:

- duas ou mais definições para as variáveis *tokens*, *literals* e *ignore*;
- *tokens* e *literals* com nomes repetidos;
- *tokens* definidos mais que uma única vez no *lexer*;
- *tokens* definidos mas não estão declarados na variável *tokens*;
- varias definições para a função *t_error* (erro para do *lexer*);
- utilização de nomes reservados para *tokens* e *literals* como variáveis da gramática;
- utilização de *literals* que não estão definidos na gramática;
- variáveis da gramática não definidas;
- varias definições para a função *p_error* (erro para do *parser*);

As situações em que são lançados avisos não impedem que os ficheiros sejam criados mas apenas servem como aviso para potenciais situações de resultados inesperados. Estas são:

- *tokens* que estão declarados mas não são definidos;
- caso de erro para o *lexer* não esta definido;
- caso de erro para o *parser* não esta definido;

3.6 Comentarios

Em *PLY-Simple* é possível escrever comentários em qualquer local. Estes são marcados pelo caracter '#' no inicio sendo que, a partir dessa marca, qualquer caractere até ao '\n' são ignorados. Os comentários escritos no código em *PLY-Simple* não aparecem nos ficheiros gerados.

Capítulo 4

Conclusão

Conclui-se que que o projeto foi desenvolvido com sucesso, atingindo as metas e objetivos inicialmente propostos. Na realização do mesmo, foi aplicado e consolidado o conhecimento adquirido na UC de Processamento de Linguagens, nomeadamente na utilização das bibliotecas Lex e Yacc do Python.