DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Parte 2

tlengrep

November 30, 2023 Teoría de Lenguajes

Integrante	LU	Correo electrónico
Santiago Fiorino	516/20	fiorinosanti@gmail.com
Lucas Mas Roca	122/20	lmasroca@gmail.com
Juan Pablo Lebon	228/21	juanpablolebon98@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - Pabellón I
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

1 Introduction

$G = \langle \{\textbf{Start}, \textbf{Exp}, \textbf{Concat}, \textbf{Quant}, \textbf{Constant}, \textbf{Char\_Class\_Contents}, \textbf{Char\_Class\_Items}\},$
$\{+, *, ?, '|', [,], \{, \}, ,, \text{char}, \backslash\text{d}, \backslash\text{w}, -, \text{rsv}\}, P, \textbf{Start}\rangle$ con P:

$$
\begin{aligned}
\textbf{Start} &\rightarrow \textbf{Exp} \mid \lambda \\
\textbf{Exp} &\rightarrow \textbf{Exp}\,'|'\,\textbf{Concat} \mid \textbf{Concat} \\
\textbf{Concat} &\rightarrow \textbf{Concat}\ \textbf{Quant} \mid \textbf{Quant} \\
\textbf{Quant} &\rightarrow \textbf{Quant}^* \mid \textbf{Quant}^+ \mid \textbf{Quant}? \mid \textbf{Quant}\{n\} \mid \textbf{Quant}\{n,m\} \mid \textbf{Constant} \\
\textbf{Constant} &\rightarrow \backslash\text{w} \mid \backslash\,\text{d} \mid \text{char} \mid [\ \textbf{Char\_Class\_Contents}\ ] \mid \\
&\quad\ \text{rsv} \mid \textbf{-} \mid (\textbf{Start}) \mid [\ ] \\
\textbf{Char\_Class\_Contents} &\rightarrow \lambda \mid \textbf{-}\ \textbf{Char\_Class\_Items} \mid \textbf{Char\_Class\_Items} \\
\textbf{Char\_Class\_Items} &\rightarrow \text{char}\ \textbf{-}\ \text{char}\ \textbf{Char\_Class\_Items} \mid \text{char}\ \textbf{Char\_Class\_Items} \mid \\
&\quad\ \text{rsv}\ \textbf{Char\_Class\_Items} \mid \text{char}\ \textbf{-}\ \text{char} \mid \text{char} \mid \text{rsv} \mid \textbf{-}
\end{aligned}
$$

Donde *Char Class* hace referencia a las clases de caracteres $[x_1...x_n]$, $\{n\}$ es el token **SQUANTIFIER** y $\{n, m\}$ el token **DQUANTIFIER** (siendo n y m números naturales). Estos últimos se definen como tokens para aprovechar el poder del lexer y luego durante el parsing se descartan los corchetes y en el caso del cuantificador doble se separan los dos valores por la coma.

2 Gramática

La gramática construida es recursiva a izquierda (ver producción **Quant** $\rightarrow$ **Quant***) y no es ambigua. Un problema que surgió es que la concatenación de expresiones regulares no tiene un símbolo asociado, al no tener un token para esta producción, no es tan sencillo definir las precedencias y asociatividades de esas reglas en la tabla en PLY (tampoco conseguimos hacer que funcionen los tokens ficticios). Optamos entonces por resolver los conflictos dentro de la gramática en vez de hacerlo en la tabla – esto generó, por ejemplo, la producción *Char_Class_Items*, para evitar problemas ante ciertas cadenas que utilizaban el símbolo '-' al comienzo de una clase de caracteres.

3 Parser

El parser usado es del tipo *LALR(1)*. Este es el default de PLY; no vimos necesidad de cambiarlo, ya que era suficientemente expresivo, y resolver los conflictos que pueden surgir puede no presentar gran dificultad.

4 Decisiones de diseño

Las decisiones de diseño que tomamos fueron:

- Para poder utilizar los corchetes como símbolos no reservados y además usarlos para los cuantificadores, debemos definir estos últimos en el token **CHAR**. Sin embargo, si hacemos esto, por la forma en la que funciona el lexer, si definimos un token muy complejo, intentará matchear primero con este, con lo cual en lugar de interpretar los cuantificadores como cuantificadores, se interpretaran como strings. Para evitar este problema, definimos el token **RSV** para permitir el uso de corchetes sin escapar en cualquier lugar que se pueda usar un **CHAR**, de esta forma, al tener un token más simple que los tokens de cuantificadores, el lexer intentará matchear primero con los cuantificadores y luego con **RSV**.
- Para poder utilizar los símbolos reservados escapados en cualquier lugar, debemos definirlos dentro de **CHAR**. Al hacer esto, cuando parseamos un **CHAR** y realizamos su traducción debemos sacar los caracteres de escape de los símbolos reservados escapados, ya que queremos por ejemplo Char('+') y no Char('\+'). Debemos tener cuidado de no realizar esto para el carácter reservado '\' ya que este sí debe ser escapado en el string para que luego cuando se evalúa se transforme en el carácter '\'.

- Al usar un cuantificador, debemos tener cuidado de no estar aplicándoselo a otro cuantificador, esto está permitido en la gramática, con lo cual debemos detectarlo y tirar un error durante el proceso de traducción. Para esto definimos en *expression.py* que la clase *Expression* tiene un método *isQuantifier* que devuelve False y definimos la clase *Quantifier* como subclase de *Expression* que tiene el mismo método pero devuelve True. Luego las clases de las expresiones son subclases de *Expression*, mientras que las clases de los cuantificadores son subclases de *Quantifier* y al aplicar un cuantificador simplemente usamos el método *isQuantifier* para verificar que no estamos aplicando el cuantificador a otro cuantificador (en ese caso se levanta el *SyntaxError*). De esta forma, no permitimos cosas como a^{**}, si queremos eso, debemos usar paréntesis $(a^*)^*$.

- Si usamos un cuantificador doble con un rango inválido, como por ejemplo, $a\{5, 3\}$ esperamos que se produzca un error. Como esto no puede ser detectado durante el análisis léxico, debemos detectarlo durante el proceso de traducción.