Taller 2, punto palíndromos

Sergio Pérez, Jhon Ramírez, Alegandro Gallego

13 de diciembre de 2013

1. Introducción

En este punto del taller dos se debía construir un programa con YACC que dijera si una palábra era palíndromo o no. Una palabra es palíndromo cuando, reversando las letras, la palabras es la misma que originalmente. Ejemplos:

- ana
- salas
- yatay
- anilina

El programa en YACC debe implementar una gramática y código para reconocer si la palabra es palíndromo.

2. Autómatas de pila no deterministas

Sea L el conjunto de palíndromos $\omega\hat{\omega}$ sobre algún alfabeto de almenos dos símbolos, donde $\hat{\omega}$ es el reverso de ω. Un autómada de pila no determinista para este lenguaje puede simplemente añadir símbolos a su pila, y en algún momento *decidir* que llegó a la mitad, y gradualmente desocupar su pila. Esto puede tomar tiempo, pero eventualmente la condición de existencia se cumple, de modo que debe existir una suposición correcta y la palabra será aceptada.

Después de varios intentos con diferentes configuraciones de autómatas de pila deterministas, se llega a la conclusión de que la mayor dificultad es determinar el punto del proceso en el que el centro de la cadena de entrada ha sido alcanzado. De modo que este problema no es solucionable por medio de autómatas deterministas.

Aunque es posible transformar autómatas finitos deterministas en autómatas finitos no deterministas expandiendo al máximo el grafo, con los autómatas de pila determinísticos no es posible realizar la conversion ya que, en general, no es posible definir una pila para todas lass transiciones posibles.

Ahora, los *parsers* que YACC genera son, escencialmente, máquinas de estado finito con una pila. Repetidamente insertan símbolos hasta que la parte

derecha de una regla gramatical esté en la pila, y luego los recuperan hacia la parte izquierda correspondiente.

3. Programáticamente

Dado el hecho de que dicha gramática no puede ser definida con YACC, se optó por la implementación de una gramática que acepta cualquier cadena de texto de longitud mayor a uno, y luego, programáticamente en código C, se determina si se trata de un palíndromo o no.

4. Código fuente

Código de palindromo.lex:

```
%{
    #include "y.tab.h"
    #include <stdlib.h>
%}

%%

[a-z]+      {
                yylval = atoi(yytext);
                return WORD;
            }

[\n]        { return *yytext; }

[ \t]       ;

.           ;

%%

int yywrap(void) {
    return 1;
}
```

Código de palindromo.yacc:

```
%{
    #include <stdio.h>
    int yylex(void);
    void yyerror(char *);
    char* isPal(char *s);
    extern char * yytext;
```

```
    int l = 1;
%}

%token WORD

%%

program:
        program expr '\n'                {isPal($2);l=1;}
        |
        ;

expr:
        WORD                                        { $$ = strdup(yytext);}
        ;

%%

void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
}

char* isPal(char *s){
        printf("%s ==> ", s);
        int l = 0;
        while(s[l] != NULL){
                l++;
        }
        int pal = 1;
        if(l == 1) pal = 1;
        else{
                        int i = 0;
                        for (i = 0; i < (l/2);i++){
                                printf("es %c ", s[i]);
                                printf("igual a %c?->", s[l - (i+1)]);
                                if(s[i] != s[l - (i+1)]){
                                        printf("no\n");
                                        pal = 0;
                                        break;
                                }
                                printf("si\n");
                        }
        }
        if(pal){
                printf("%s", "Es un palindromo");
        }else
```

```
                printf("%s", "No es un palindromo");
        printf("%s", "\n>>");
        return s;
}

int main(void) {
    yyparse();
    return 0;
}
```

5. Conclusiones

Se implementó un programa usando YACC y LEX para el reconocimieto de palíndromos pares e impares. En el proceso de investigación para llevar cabo el punto del taller, se aprendió que no es posible definir una gramática en YACC para dicho problema, de modo que se decidió usa código C para atacarlo.

Un aprendizaje importante fue el relacionado con manipular TOKENS como si se tratasen de cadenas de caracteres de C. Esto se hace mediante el *snippet* $$ = strdup(yytext);. El algoritmo seguido para la determinación fue el más obvio al momento de la implememtación, y funcionó.

Referencias

[1] Klaus Draeger *http://cstheory.stackexchange.com/questions/9673/why-is-non-determinism-push-down-automata-necessary* 2012.

[2] Jaime Seguel *http://ece.uprm.edu/ jseguel/L13ToC.pdf*

[3] Pete Jinks *http://www.cs.man.ac.uk/ pjj/cs2112/ho/node6.html* Inside LEX and YACC: State Machines