Llenguatges Formals
Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili

Pràctica 2: Bison

Josep Bello Curto
Oriol López Egea
Lautaro Russo Bertolez

18-05-2021

Índex

1. Lex

Primer que tot el fitxer que es vol convertir a CSV va passant per un analitzador sintactic generat en lex.

A aquest hem definit diverses expressions regulats per detectar cada part del XML i hem fet que per cada part del XML es retorni un token per tal que l'analitzador sintactic el detecti.

1.1. Expressions regulars

1.1.1. Apertura matriu simple

```
m := matriz

apertura_m_simple := <{m}>
```

Aquesta expressió composta ens dona match amb el que considerem el tag d'obertura d'una matriu que no consta d'atribut.

1.1.2. Apertura matriu amb atributs

```
nombre_tag := separador|N

atributo :=  {nombre_tag}+=(\"{texto}*\"|'{texto}*')

apertura_m_atr := <{m}({espacio}+{atributo})+{espacio}*>
```

Aquesta expressió composta ens fa match amb els tags d'opertura de matriu amb atributs. Admet cometes simples i dobles.

1.1.3. Tancament matriu

```
cierre_m := <\m>/
```

Aquesta expressió composta ens dona match amb el que considerem un tag de tancament de matriu.

1.1.4. Apertura fila

```
apertura_f := <fila>
```

Aquesta expressió simple ens dona match amb el que considerem el tag d'obertura d'una fila.

1.1.5. Apertura valor

```
apertura_v := <valor>
```

Aquesta expressió simple ens dona match amb el que considerem el tag d'obertura d'un valor.

1.1.6. Text

`texto := [^<>'"&\n]+`

Aquesta expressió simple ens dona match amb els textos que n'hi ha entre tags, que pot ser qualsevol caràcter menys els reservats pel nostre XML.

1.1.7. Tancament valor

`cierre_v := <\valor>/`

Aquesta expressió composta ens dona match amb el que considerem un tag de tancament de valor.

1.1.8. Tancament fila

`cierre_f := <\fila>/`

Aquesta expressió composta ens dona match amb el que considerem un tag de tancament de fila.

1.1.9. Apertura columna

`apertura_c := <columna>`

Aquesta expressió simple ens dona match amb el que considerem el tag d'obertura d'una fila.

1.1.10. Tancament columna

`cierre_f := <\columna>/`

Aquesta expressió composta ens dona match amb el que considerem un tag de tancament de columna.

1.1.11. Enter

`enter := \n`

Aquesta expressió simple ens dona match amb el salt de linea per detectar els enters del XML que no surtin del lex.

1.2. Tokenizer

Per poder processar el fitxer desade bison hem definit els següents tokens que seran retornats depenent del match amb les expressions regulars.

```
1  {apertura_m_simple} {return APERTURA_M_SIMPLE;}
2  {apertura_m_atr} {yylval.str = strdup(yytext); return APERTURA_M_ATR;}
3  {cierre_m} {return CIERRE_M;}
4
5  {apertura_f} {return APERTURA_F;}
6  {apertura_v} {return APERTURA_V;}
7  {texto} {yylval.str = strdup(yytext); return TEXTO;}
8  {cierre_v} {return CIERRE_V;}
9  {cierre_f} {return CIERRE_F;}
10
11 {apertura_c} {return APERTURA_C;}
```

```
12 {cierre_c} {return CIERRE_C;}
13 {enter} {}
```

2. Bison

D'altra banda a Bison hem definit els tokens previament explicats, i a mes hem afegit que els tokens de `"texto"` i `"apertura_m_atr"` guardin el text que ha fet saltar un match al lex.

2.1. Reglas de la gramàtica

2.1.1. Start

Es la primera regla a la qual s'entra, aquesta processa tot el fitxer cridant a al tres regles, no terminals, de la gramàtica.

També contemplem els casos de que la matriu esta buida, en el qual salta un error o be que nomes conte columnes, pel el qual es genera un warning

```
start : apertura_m columnas filas cierre_m |
    apertura_m cierre_m {yyerror("Matriz vacia");} |
    apertura_m columnas cierre_m {printf("[WARNING]: El fichero solo contiene columnas\n")
    ;};
```

2.1.2. Columnes i columna

Primer al fitxer xml ens trobem amb les columnes que ha de contenir el csv, per tal de fer la conversió de totes les columnes que conte el fitxer hem definit les dues regles següents.

Amb la primera regla processem de forma recursiva totes les columnes.

Amb la segona processem cada columna, on, si es tracta de la primera nomes es converteix el text que hi ha, però si es qualsevol altra columna es fica una coma abans.

```
columnas : columna | columnas columna;

columna : APERTURA_C TEXTO CIERRE_C {
    if(first_col) {
        first_col = 0;
        print_val($2);
    } else {
        fprintf(csv, "%c", delimiter);
        print_val($2);
    }

    cols++;
};
```

2.1.3. Valors i valor

Per processar els valors hem fet servir la mateixa idea que amb les columnes, dues regles, on una processa tots els valors i un altra un valor en particular, aquesta primera regla es la que s'utilitza de cada fila.

Per processar un valor simplement fem una copia a `$$`, el valor de retorn, el text del valor.

D'altra banda a la regla de processar múltiples valors es fa un processament recursiu guardant al valor de retorn de vals tots els texts a un array

```
val : APERTURA_V TEXTO CIERRE_V  {
    strcpy($$, $2);
};

vals : val {
    $$.str_array = malloc(sizeof(char*));
    $$.str_array[0] = $1;
    $$.n_elem = 1;
} | vals val {
    $$.str_array = realloc($1.str_array, sizeof(char*) * $1.n_elem);
    $$.str_array[$1.n_elem] = $2;
    $$.n_elem = $1.n_elem + 1;
};
```

Per poder fer tots els retorns correctament entre regles hem definit els següents elements

```
%union {
    char *str;
    struct {
        char **str_array;
        int n_elem;
    } str_array;
};

%type<str> val
%type<str_array> vals
```

2.1.4. Files i fila

Per processar les files hem fet servir la mateixa idea que amb les columnes i valors, dues regles on una processa de forma recursiva totes les files i un altra processa cada fila per separat.

Al processar una fila en particular s'utilitza la regla vals, descrita prèviament, que ens retorna els valors i nombre de valors que te una fila. Primer que tot es comprova que el nombre de columnes no sigui superior al que hi ha al xml, si es aixi es genera un warning.

A continuació recorrem la llista de valors d'una fila guardant al csv un valor o un valor precedit d'una coma, depenent de si es la primera iteració o no.

Finalment afegim la quantitat de delimitadors que facin falta per completar les columnes.

```
filas : fila | filas fila;

fila : APERTURA_F vals CIERRE_F {
    fprintf(csv, "\n");
    if($2.n_elem > cols) printf("[WARNING]: La fila %d tiene %d valores mas que columnas
    tiene la matriz\n", filas, $2.n_elem - cols);
    for(int i = 0; i < $2.n_elem && i < cols; i++) {
        if(i == 0) {
            print_val($2.str_array[i]);
        } else {
            fprintf(csv, "%c", delimiter);
            print_val($2.str_array[i]);
        }
    }

    for(int i = 0; i < cols - $2.n_elem; i++) fprintf(csv, "%c", delimiter);

    filas++;
};
```

2.1.5. Apertura matriu simple i amb atribut

Per les apertures de les matrius hem definit dues regles.

Per processar una apertura sense atributs simplement agafem el token retornat pel lex sense fer res mes.

Pero si es tracta d'una apertura amb algun atribut s'ha d'assignar el nombre de files i/o delimitador, per fer aixo s'utilitza la rutina set_atr.

```
apertura_m_simple : APERTURA_M_SIMPLE;

apertura_m_atr : APERTURA_M_ATR {
    set_atr($1);
};
```

2.1.6. Tancament matriu

Per processar el tancament d'una matriu donat que no s'ha de fer res en particular simplement agafem el token retornat pel lex sense fer res mes.

```
cierre_m : CIERRE_M;
```

2.2. Funcions auxiliars

2.2.1. set_atr: Agafar atributs del tag matriz

Per agafar els atributs que pugui tenir el tag matriz com s'ha vist a la regla Apertura matriu simple i amb atribut utilitzem el mètode `set_atr`.

Amb aquesta rutina recorrem tot el string del tag matriz buscant el `=` i mirant si ve precedit per una `N` o una `r`, en el primer cas es tractara del atribut que ens indica el nombre de files i en el segon del delimitador.

Un cop s'ha trobat aquest `=` es passa el string a una segona funció, `get_val`, que extraurà el contingut del interior de les `"` o `'` i el retornara a `set_atr` qui farà un atoi o una assignació a la variable que toqui.

```
char* get_val(char *s) {
    char *s_cpy = (char*)calloc(sizeof(s), sizeof(char));
    strcpy(s_cpy, s);

    char *token = strtok(s_cpy, "'\"");
    for(int i = 0; token != NULL && i < 1; i++) token = strtok(NULL, "'\"");

    char *token_cpy = (char*)calloc(sizeof(token), sizeof(char));
    strcpy(token_cpy, token);
    return token_cpy;
}

void set_atr(char *s) {
    for(int i = 0; s[i] != '\0'; i++) {
        if(s[i] == '=' && s[i - 1] == 'N') max_files = atoi(get_val(&s[i]));
        if(s[i] == '=' && s[i - 1]  == 'r') delimiter = get_val(&s[i])[0];
    }
}
```

2.2.2. print_val: Mostra valors sense separador ni espais

Per tal d'escriure correctament tot el text al csv hem definit aquesta rutina que s'encarrega de recórrer tot el string on com a cas general s'escriu al csv el caràcter del xml però en cas de trobar algun element que sigui el delimitador ficar un `$` i generar un warning i si hi ha un espai l'emet.

```
void print_val(char *s) {
    for (int i = 0;  s[i] != '\0' ; i++) {
        if(s[i] != delimiter && s[i] != ' ') fprintf(csv, "%c", s[i]);
        else if(s[i] == delimiter) {
            fprintf(csv, "$");
```

```
6               printf("[WARNING]:L'element \"%s\" conte el separador: '%c'\n", s, delimiter);
7           }
8       }
9   }
```

3. Execució

Quan comença el programa main, aquest rep per paràmetre de quin fitxer s'ha de processar y ho emmagatzemem a yyin.

Després s'executa yyparse per a executar tot el codi lex per aplicar les expressions regulars corresponents, quan una expressió fa match retorna un token al bison i en alguns casos també retorna l'element que ha fet el match per a poder verificar la gramàtica.

Finalment, abans d'acabar en cas que al tag d'obrir la matriu ens hagin indicat un nombre de files verifiquem que sigui l'esperat.

4. Joc de proves

Per fer el joc de proves hem escrit el nostre propi example per verificar el correcte funcionament de la practica.

```
<matriz separador=',' N="3">
<columna>nombre 1</columna>
<columna>nombre 122</columna>
<columna>nom,bre 132</columna>
<fila><valor>va,lor 0 1 </valor><valor>valor 0 2 </valor><valor>valor 0 3 </valor><
    valor>patata</valor></fila>
<fila><valor>valor 1 1 </valor><valor>valor 1 2 </valor></fila>
<fila><valor>valor 2 1 </valor></fila>
<fila><valor>valor 2 1 </valor></fila>
</matriz>
```

Quan executem el nostre codi amb aquest fitxer d'entrada ens dona el seguent output

```
nombre1,nombre122,nom$bre132
va$lor01,valor02,valor03
valor11,valor12,
valor21,,
valor21,,
```

```
[WARNING]: El elemento "nom,bre 132" contiene el separador: ','
[WARNING]: La fila 0 tiene 1 valores más que columnas tiene la matriz
[WARNING]: El elemento "va,lor 0-1" contiene el separador: ','
[WARNING]: El numero de filas es mas grande que el esperado, 4 > 3
```

Figura 4.1: Warnings generats al convertir el fitxer