Compiladors: Examen de laboratori

Març de 2016

Important

Al Racó trobareu un fitxer `examen.tgz` que conté:

- Aquest enunciat
- Intèrpret ASL original
- antlr3
- Jocs de proves per validar els exercicis proposats a l'examen
- Un script `avalua.sh` que executa l'ASL sobre els jocs de proves i diu si la resposta és l'esperada.
- Un script `empaqueta.sh` que crea un fitxer `entrega.tgz` per pujar al Racó

Al final de l'enunciat trobareu detalls sobre quins fitxers cal modificar, en quins directoris cal deixar-los, i com executar els scripts d'avaluació i empaquetat.

Exercici 1 : Gramàtica ANTLR (5 punts)

DOT és un llenguatge de descripció de grafs, en el que es codifiquen nodes i arestes, permetent-ne la renderització posterior amb algun motor gràfic que suporti aquest llenguatge.

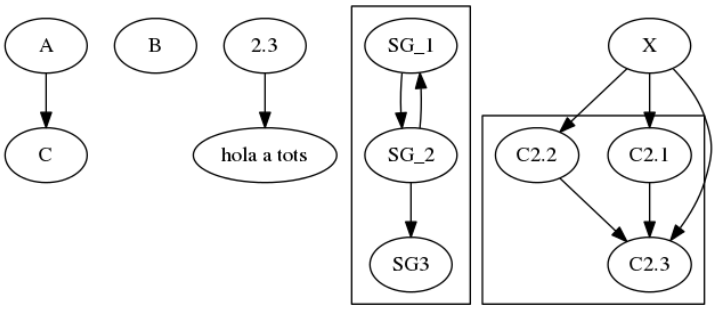
Volem escriure una gramàtica ANTLR per a una versió simplificada del llenguatge *DOT*. En aquesta versió simplificada, un graf consisteix en la paraula clau `digraph` seguida d'una etiqueta (opcional) i seguida d'una llista d'elements (que pot ser buida) entre '{' i '}'.

Els elements del graf poden ser nodes, subgrafs o arcs, en qualsevol ordre. Ente dos elements pot haver-hi (opcionalment) un punt i coma.

- Un node és simplement una etiqueta. P.ex.: `Node1`, `"Aquest és el node 2"`, etc.
- Un subgraf té la mateixa estructura que un graf, però comença amb la paraula clau `subgraph` enlloc de `digraph`.
- Un arc consisteix en el token '`->`' amb un node o subgraf a l'esquerra i un node o subgraf a la dreta. P.ex.: `Node1 -> Node2`.
  Els arcs poden encadenar-se de forma que el segon node d'un arc sigui el primer del següent. P.ex.: `Node1 -> Node2 -> Node3`.

Les etiquetes (tant les dels grafs i subgrafs com les dels nodes) poden ser un identificador, un string entre cometes, o un valor numèric (enter o real).

Per exemple, el graf:



es pot descriure amb el codi:

```
digraph Graf_1 {
   A // un node
   A -> C // un arc
   B // un altre node

   2.3 -> "hola a tots" // un arc entre un node amb una etiqueta
                        // real i un altre amb etiqueta string

   subgraph cluster_1 {  // un subgraf
      SG_1 -> SG2 -> SG3;
      SG_2 -> SG_1;
   }

   // un arc entre un node i un subgraf
   X -> subgraph cluster_2 { "C2.1" -> "C2.3"; "C2.2" -> "C2.3"; }
}
```

Es demana ampliar el fitxer `Dot.g` amb el següent:

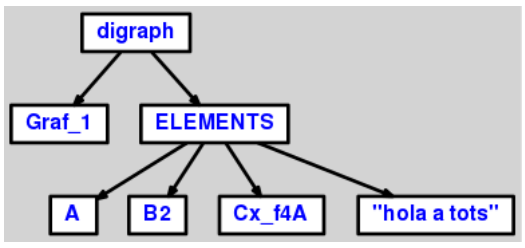
- La definició del token `NUM` per que reconegui qualsevol nombre enter o real. Valors vàlids són p.ex 23, 4.56, .002, 45., 029.
- Una gramàtica amb símbol inicial `graph` que reconegui els grafs descrits anteriorment i generi l'AST demanat en cada un dels jocs de proves següents (que trobareu al subdirectori `jps`).

Joc de Proves 1

Codi:

```
digraph Graf_1 {
   A // un node
   B2; // un altre node
   Cx_f4A // un node amb un nom estrany
   "hola a tots" // un node amb un string com a etiqueta
}
```

AST a generar:



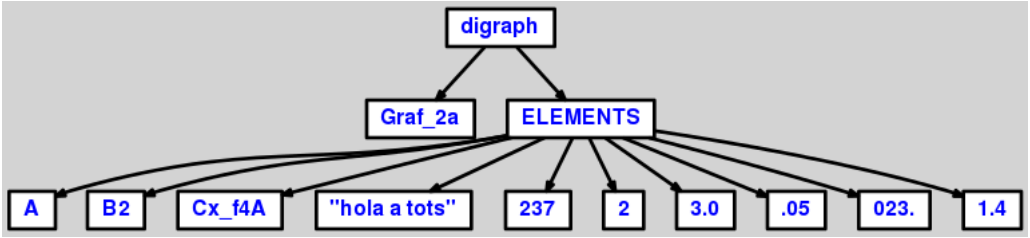
Joc de Proves 2

Codi:

```
digraph Graf_2a {
   A // un node
   B2; // un altre node
   Cx_f4A // un node amb un nom estrany
   "hola a tots" // un node amb un string com a etiqueta

   237; 2; 3.02  // diversos nodes amb etiquetes numèriques
   .05; 023.
   1.4;
}
```

AST a generar:



Codi:

```
digraph Graf_2b {
   A // un node
   B2; // un altre node
   Cx_f4A // un node amb un nom estrany
   "hola a tots" // un node amb un string com a etiqueta

   237; 2; 3:0 // diversos nodes amb etiquetes numèriques
   .05; 23,4;
   ,01
   1.4;
}
```

Resultat a generar (stderr):

```
jps/jp2b.dot line 7:12 no viable alternative at character ':'
jps/jp2b.dot line 8:10 no viable alternative at character ','
jps/jp2b.dot line 9:3 no viable alternative at character ','
```

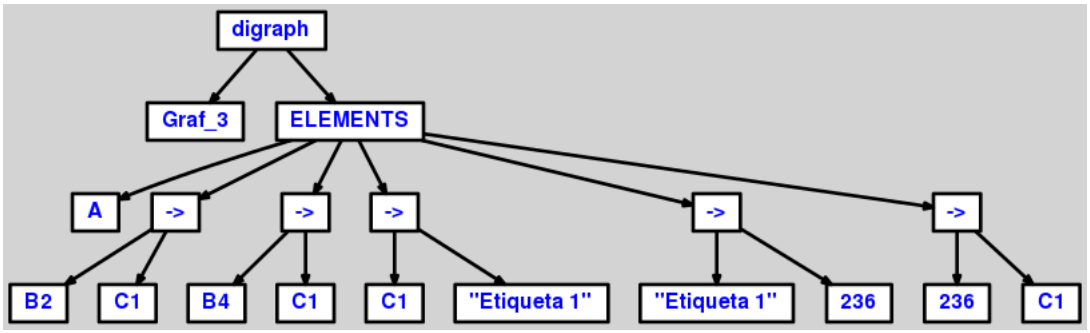
Joc de Proves 3

Codi:

```
digraph Graf_3 {
   A;
   B2 -> C1;  // un arc
   B3 -> C1
   B4 -> C1;
   C1 -> "Etiqueta 1"

   "Etiqueta 1" -> 236;
   236 -> C1
}
```

AST a generar:

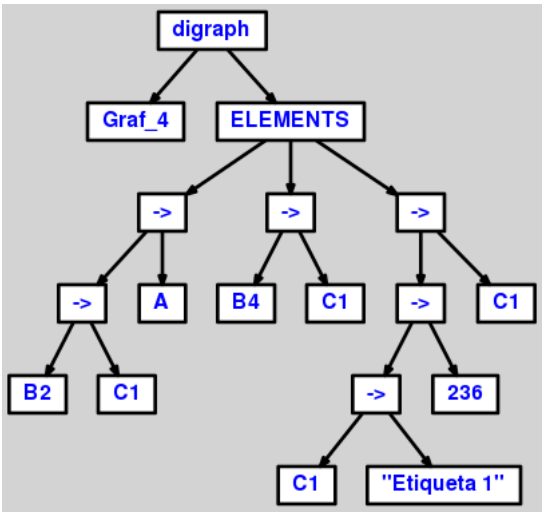


Joc de Proves 4

Codi:

```
digraph Graf_4 {
   B2 -> C1 -> A;
   B4 -> C1;
   C1 -> "Etiqueta 1" -> 236 -> C1
}
```

AST a generar:



Joc de Proves 5

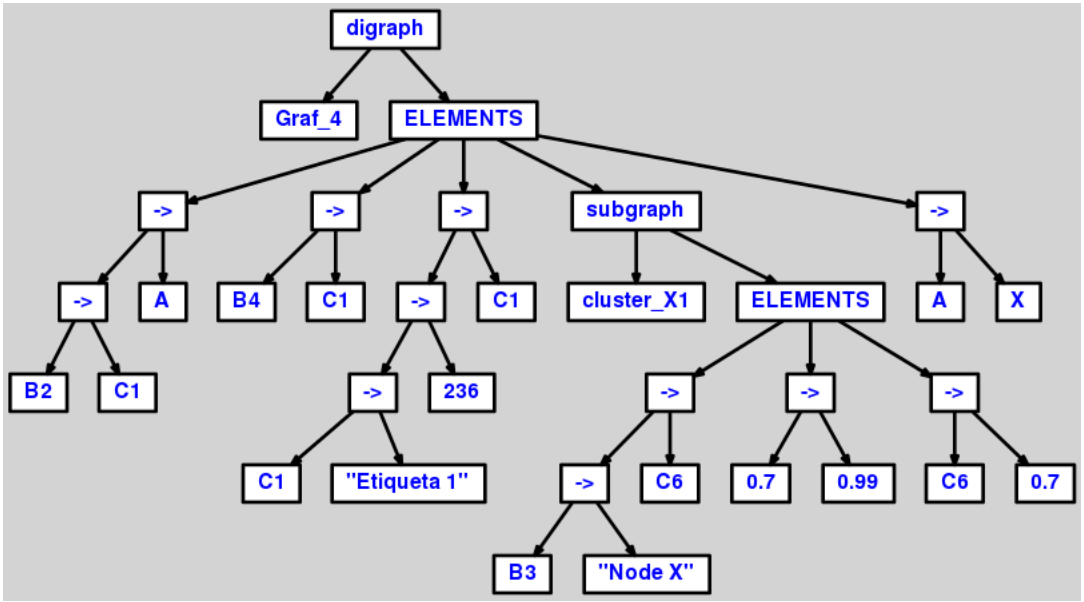
Codi:

```
digraph Graf_5 {
   B2 -> C1 -> A;
   B4 -> C1;
   C1 -> "Etiqueta 1" -> 236 -> C1

   subgraph cluster_X1 {
     B3 -> "Node X" -> C6
     0.7 -> 0.99
     C6 -> 0.7
   }

   A -> X;
}
```

AST a generar:



Joc de Proves 6

Codi:

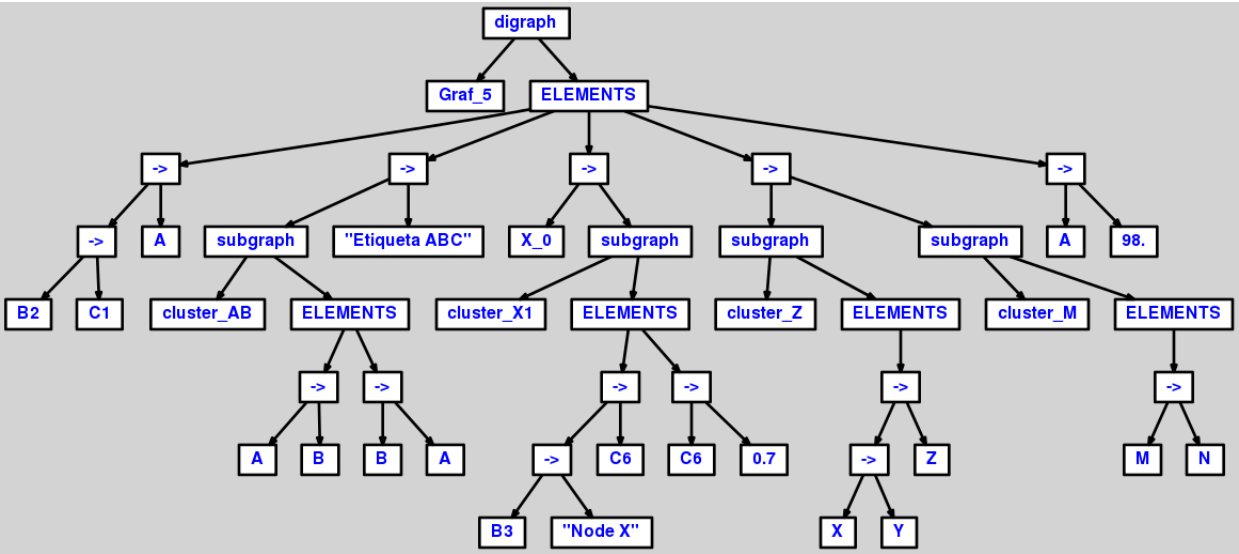
```
digraph Graf_5 {
   B2 -> C1 -> A;
   subgraph cluster_AB {
       A -> B;
       B -> A;
   } -> "Etiqueta ABC"

   X_0 -> subgraph cluster_X1 {
            B3 -> "Node X" -> C6
            C6 -> 0.7
          }

   subgraph cluster_Z { X -> Y -> Z; } -> subgraph cluster_M { M -> N };

   A -> 98.
}
```

AST a generar:



Exercici 2 : Modificació ASL

Instrucció iterativa *foreach* (3 punts)

Afegir una nova instrucció al intèrpret, la iteració *foreach*. Heu de fer dues versions com les que veieu en el següent exemple:

```
func main()
  x = 1; y = 3;
  s = 0;
  foreach a in [x, y-1, x*y, 14] do       // amb expressions entre cortxets
    s = s+a;
    x = x+1
  endfor;
  write s; write "%n";                    // escriu "20"
  foreach b in {x>3, a-1, x*y, false} do  // amb expressions entre claus
    write b; write "%n"                   // escriu "true" "13" "15" "false"
  endfor
endfunc
```

Les expressions de la llista, tant en el primer cas (entre `[` i `]`) com en el segon (entre { i }), han de ser avaluades **abans** de començar el bucle. Tant l'ordre de l'avaluació de les expressions com el del recorregut amb la variable de control és de esquerra a dreta.

- En la primera forma de *foreach* les expressions han de ser del mateix tipus (enter o booleà). El intèrpret **ha de comprovar-ho** una vegada han estat avaluades totes. Si no són del mateix tipus s'ha de llançar una excepció amb l'instrucció:

  ```
  throw new RuntimeException ("Expecting identical type expressions in foreach");
  ```

- En la segona forma de *foreach*, amb expressions entre claus, els tipus de les expressions poden ser diferents.

Poden haver tot tipus d'instruccions en el cos d'una instrucció *foreach*.

Jocs de proves: `foreach_1.asl`, `foreach_2.asl`, `foreach_3.asl` i `foreach_4.asl`.

Modificar l'intèrpret per a introduir la nova instrucció. Només cal modificar els fitxers `Asl.g` i `Interp.java`.

Funcions predefinides *isint* i *isbool* (2 punts)

Afegir les noves funcions booleanes *isint* i *isbool* que donada una expressió retornen `true` si el valor de l'expressió és del tipus corresponent. Altrament retornen `false`.

```
func main()
  i = 7;
  if i%2 = 0 then    n = i*i
  else               n = i>99
  endif;
  if isint(n) then
    write n+1;
  else
    if isbool(n) then  write n or true;  endif    // escriu "true"
  endif;
  write "%n"
endfunc
```

Jocs de proves: `istype_1.asl` i `istype_2.asl`.

Modificar l'intèrpret per a introduir les noves funcions. Només cal modificar els fitxers `Asl.g` i `Interp.java`.

Fitxers a usar

Al paquet `examen.tgz`, trobareu els següents fitxers:

- `enunciat.pdf`: Aquest document.
- `antlr3`: Directori amb les llibreries antlr3. No s'ha de modificar ni moure de lloc.
- `Exercici-ANTLR`: Directori amb els fitxers per al primer exercici. Veure més avall.
- `Exercici-ASL`: Directori amb els fitxers per al segon exercici. Veure més avall.
- `avalua.sh`: Script que compila els dos exercicis, els aplica els jocs de proves, i diu si el resultat és l'esperat o no.
- `empaqueta.sh`: Script que empaqueta els dos exercicis en un fitxer `entrega.txt` que cal puja al Racó.

Al directori `Exercici-ANTLR` trobareu els fitxers necessaris per al primer exercici:

- `Dot.g` : Esquelet de gramàtica a completar per fer l'exercici. Aquest és l'unic fitxer que cal modificar.
- `Dot.java` : Programa principal. No s'ha de modificar.
- `Makefile` : Fent `make`, compila la gramatica i construeix el parser. No s'ha de modificar.
- `jps` : directori amb els jocs de proves (`*.dot`). També conté les sortides esperades per a cada un (`*.ok.ast`, `*.ok.out`). No s'han de modificar ni canviar de lloc.
- `executa` : Script que executa el parser sobre el fitxer `.dot` donat com a parametre, i genera l'AST en format text (`.ast`) i `.pdf`. No s'ha de modificar.
  EXEMPLE: `./executa jps/jp1.dot`. Els resultats quedaran a `jps/*.student.*`.

Al directori `Exercici-ASL` trobareu els fitxers necessaris per al segon exercici:

- `Asl` : Codi original de l'intèrpret. Treballeu des d'aquest directori, usant `make` per construir el programa com habitualment.
  Haureu de modificar els fitxers `src/parser/Asl.g` i `src/interp/Interp.java`.
  Podeu fer una còpia de seguretat de la versió original, però la resposta a l'exàmen ha d'estar al directori `Asl`, mantenint la distribució original dels fitxers i subdirectoris.
- `jps` : directori amb els jocs de proves (`*.asl`). També conté les sortides esperades per a cada un (`*.ok.ast`, `*.ok.out`). No s'han de modificar ni canviar de lloc.
- `executa` : Script que executa el parser sobre el fitxer `.asl` donat com a parametre, i genera l'AST en format `.pdf`. No s'ha de modificar.
  EXEMPLE: `./executa jps/jp1.asl`. Els resultats quedaran a `jps/*.student.*`.