Presentació

En aquesta Prova d'Avaluació Continuada es treballa la fase d'anàlisi sintàctic d'un compilador. La PAC combina preguntes teòriques sobre els algorismes de parsing amb un exercici pràctic on haureu de fer servir el software CUP per construir un analitzador sintàctic.

Competències

En aquesta PAC es desenvolupen les següents competències del Grau en Enginyeria Informàtica:

- Capacitat per analitzar un problema en el nivell d'abstracció adequat a cada situació i aplicar les habilitats i coneixements adquirits per a resoldre'l.
- Capacitat per dissenyar i construir aplicacions informàtiques mitjançant tècniques de desenvolupament, integració i reutilització.
- Capacitat per aplicar les tècniques específiques de tractament, emmagatzemament i administració de dades.
- Capacitat per proposar i avaluar diferents alternatives tecnològiques per resoldre un problema concret.

Objectius

Els objectius concrets d'aquesta Prova d'Avaluació Continuada són:

- Entendre el paper d'un analitzador sintàctic dins un compilador.
- Repassar com utilitzar gramàtiques independents de context per descriure un llenguatge.
- Saber aplicar algorismes d'anàlisi sintàctic ascendent i descendent per una gramàtica donada.
- Ser capaç de desenvolupar un analitzador sintàctic utilitzant CUP.

Descripció de la PAC

Exercici 1 (30%)

Donada la següent gramàtica, amb símbols terminals p, q i no terminals A, B, S:

$S \rightarrow AB \mid A \mid p$

$A \rightarrow pS \mid q$

$B \rightarrow p$

1. Calcular la col·lecció canònica LR(0).
2. Calcular la corresponent taula d'estats.
3. Calcular la taula d'accions SLR(1).
4. Existeix ambigüetat en la gramàtica proporcionada? Raona la teva resposta.

Per elaborar l'exercici, cal tenir en compte les següents consideracions:

- Si escau, es pot introduïr una regla addicional a la gramàtica: $S' \rightarrow S\#$
- Es valorarà positivament que la solució segueixi la recomanació del mòdul 3 (pàg. 51): "*Per no deixar-se res, és aconsellable seguir un ordre a l'hora de construir els conjunts; per exemple, primer avaluar els no terminals i després els terminals segons l'ordre d'aparició en la definició de la gramàtica*"

Solució:

S'inclou la solució en un full de càlcul adjunt, en formats ODS i PDF.

Exercici 2 (70%)

Continuem amb la definició del nostre llenguatge de definició de serveis. En aquesta segona PAC, en desenvoluparem l'analitzador sintàctic utilitzant el software CUP.

A nivell sintàctic, el llenguatge té les següents característiques:

Estructura general Un programa consisteix en zero o més declaracions de servei. Existeixen diferents tipus de declaració de servei, descrits més avall: servei d'**audio**, servei d'**interruptor**, servei **json** i servei de **script**. Tots els tipus de declaracions poden aparèixer al document intercalats i en qualsevol ordre.

Serveis d'audio La declaració d'un servei d'audio comença amb les paraules clau: `audio service`. A continuació, segueix un identificador que en denota el seu nom, seguit de la paraula clau `plays` i la ruta d'un fitxer. Finalment s'acaba la declaració amb un delimitador de punt i coma (`;`).

Serveis d'interruptor Els serveis d'interruptor segueixen una estructura similar als d'audio: En primer lloc, la declaració comença amb les paraules clau `switch service`. Seguidament, s'inclou un identificador que li dóna nom. A continuació, va el nom d'una acció d'interruptor, entre les quals s'inclouen `opens`, `closes` i `toggles`. Després de l'acció, segueix un identificador que denota l'interruptor sobre el que s'actuarà. Finalment la sentència de declaració acaba amb un punt i coma (`;`).

Servei JSON La declaració d'un servei JSON segueix la següent estructura:

- Les paraules clau `json service`, seguides d'un identificador que en denota el nom.
- Una **llista d'arguments**. Les llistes d'arguments estan delimitades per parèntesis (`(`, `)`). Dins de la llista d'arguments s'inclou una llista de zero o més declaracions d'arguments, separats per comes (`,`).
- Una **declaració d'argument** consisteix en una parella identificador i un tipus. Aquestes dues es troben separades entre sí pel delimitador de dos punts (`:`). El tipus pot ser una de les següents paraules reservades: `string`, `int`.
- Després de la llista d'arguments, segueix un **document JSON**[1]. Cal recordar que, tot i que JSON és un llenguatge real, en aquest exercici treballarem amb una versió simplificada. Així doncs, en la resta de l'enunciat, quan es parli de document JSON s'està fent referència a l'estructura sintàctica que es descriu dins la següent secció: "Document JSON".

[1] Més info a: `https://www.json.org/json-en.html`

Document JSON Un document JSON consisteix en zero o més definicions de camps, separades entre sí per comes i totes elles delimitades entre claus: { }.

- Una definició de camp consisteix en una **parella clau-valor**.
- Una **clau** consisteix en un identificador vàlid, seguit del delimitador de dos punts. A part dels identificadors, també es consideraran claus vàlides les paraules reservades `endpoint` i `request`.
- Un valor pot ser algun dels següents elements: Un nombre enter, una string, un identificador, una ruta de fitxer, una adreça web (URL), o bé, recursivament, un altre document JSON[2].
- La coma després de l'última definició de camp del document JSON és opcional.

Serveis de script Els script permeten definir comportaments complexos utilitzant una sintaxi lleugerament inspirada en el llenguatge de programació C. Per declarar un servei script, se segueix una estructura similar als serveis JSON: Les paraules clau `script service`, seguides d'un identificador, una llista d'arguments i finalment, un **bloc de codi** delimitat per claus {, }. Un bloc de codi, consisteix en zero o més sentències, d'entre les quals podem trobar:

- Una declaració de variable, que comença per la paraula clau `let`, el seu identifiacador, el separador de dos punts (`:`) i el tipus d'aquesta (`string` o `int`). A més, opcionalment la declaració pot portar una assignació inicial, que s'explica al següent punt.
- Una assignació consisteix en un identificador de variable, seguida de l'operador `=` i a la banda dreta d'aquest una *expressió*. Definim la sintaxi de les expressions més endavant.
- Una sentència especial, que consisteix en una paraula clau: `say` o `sleep`, seguida d'una única expressió.
- Una crida a servei, que consisteix en un identificador, seguit d'una llista de zero o més expressions separades per coma delimitades per parèntesi.

Després de cada tipus de sentència, s'indica sempre el final d'aquesta utilitzant el delimitador de punt i coma (`;`).

Expressions Dins un bloc de codi poden aparèixer expressions que combinen operands i operadors. Els operadors es poden aplicar sobre literals string, literals enters, identificadors i **crides feedback**, definides més avall. Cal notar que les comprovacions semàntiques de tipus no corresponen a aquesta pràctica. Així doncs, cal considerar que l'expressió `12 * "adeu"` és sintàcticament correcta. Tots els operadors **tenen associativitat per la dreta**. Per les expressions amb operadors s'apliquen les següents regles de precedència:

[2]Noteu que no s'inclouen les Array en aquesta definició. Això és una simplificació respecte el llenguatge JSON real.

- En primer lloc, s'avaluen els operadors de suma (`+`), resta (`-`) i concatenació de string (`++`). Tots al mateix nivell.
- Després, s'avaluen els operadors de multiplicació (`*`) i divisió (`/`). Ambdós al mateix nivell.
- Finalment, aquest ordre d'avaluació es pot modificar utilitzant parèntesis (`(`, `)`).

A part dels usuals, aquest llenguatge inclou les **crides feedback** com a possible operand. Les crides feedback permeten obtenir informació de l'usuari. Tenen una sintaxi similar a les sentències especials, però amb la diferència que poden aparèixer dins les expressions. Una crida feedback es representa per la paraula clau `feedback`, seguida d'una paraula clau de tipus: `int`, o `string`.

Es demana que, utilitzant CUP, desenvolupeu un analitzador capaç de reconèixer les estructures sintàctiques del llenguatge.

L'analitzador ha de ser capaç de **recuperar-se dels errors en la mesura del possible**. Concretament, l'analitzador s'ha de recuperar i ha d'imprimir un missatge d'error informatiu en les següents situacions: *(i)* Un audio service que es declara sense indicar una ruta de fitxer. *(ii)* Una sentència feedback sense indicar un tipus (`int` o `string`). *(iii)* Una declaració de servei JSON o script sense llista d'arguments.

Recordeu que per aquesta PAC no s'ha d'implementar cap analitzador semàntic.

Informe escrit

Juntament amb l'implementació del vostre analitzador, cal **entregar una descripció de les decisions adoptades en el desenvolupament del vostre analitzador**. En aquesta descripció, s'espera que inclogueu, per cadascuna de les funcionalitats sintàctiques descrites anteriorment, quina estratègia d'implementació heu seguit. Es valorarà positivament que l'informe faci referència al codi entregat allà on escau.

Jocs de proves i plantilla de codi

Per a la resolució d'aquest exercici, cal que **utilitzeu la plantilla de fitxers JLex i CUP proporcionada juntament a aquest enunciat**. La plantilla parteix d'una possible solució de la PAC 1.

De manera similar a la PAC anterior, aquesta plantilla també inclou un avaluador automàtic amb **fitxers de casos de prova** per comprovar el correcte funcionament del vostre analitzador. Degut a la major complexitat de l'analitzador semàntic, la naturalesa d'aquest avaluador és més limitada que en el cas lèxic. Per això és important assegurar-se que el vostre analitzador funciona més enllà dels casos de test proporcionats. En la correcció de la PAC **es valorarà positivament que la solució entregada faci un bon ús de la plantilla** (veure a sota) i **passi els tests de l'avaluador automàtic** inclosos amb l'enunciat.

Per a fer un bon ús de la plantilla, cal que tingueu en compte les següents consideracions:

- Després de reconèixer una declaració de servei d'audio, cal cridar la funció `emitAudioService`.
- Similarment, després de reconèixer una declaració de servei de tipus interruptor, cal cridar la funció `emitSwitchService`.
- Després de reconèixer un servei JSON, cal cridar la funció `emitJSONService`.
- En reconèixer un servei de tipus script, caldrà cridar la funció `emitScriptService`.
- Totes les funcions indicades anteriorment es troben declarades com a mètodes estàtics dins la classe `ParserEvaluator`, i reben un únic paràmetre de tipus String que correspon al nom del servei.
- Finalment, sempre que es detecti un error sintàctic reconegut, cal cridar la funció `emitError`, passant com a paràmetre un dels possibles valors: `ErrorType.AUDIO_NO_PATH`, `ErrorType.FEEDBACK_NO_TYPE`, o bé `ErrorType.NO_ARG_LIST`. Els tres valors fan referència als casos *(i), (ii), (iii)* descrits a la secció anterior.

Solució:

```
1  import java_cup.runtime.*;
2  import java.io.*;
3  import java.lang.String;
4
5  import Eval.ParserEvaluator;
6  import Eval.ErrorType;
7
8  parser code {:
9
10         static FileInputStream FInStr = null;
11         public static String fInName;
```

```
12
13          // The lexer class is stored as a member field to allow
14          // access to its getLine() method.
15          public static Yylex yy;
16
17          public static void main (String argv[]) throws Exception {
18              parser analyzer;
19
20              String mode = argv[0];
21              if (!mode.equals("run") && !mode.equals("evaluate") && !mode.equals("
    generate")) {
22                  System.err.println("[Arguments Error] The first parameter must be
    either \"run\" or \"evaluate\".");
23              } else {
24                  String caseName = argv[1];
25                  String filePath = caseName + "." + ParserEvaluator.
    LANGUAGE_EXTENSION;
26                  FileInputStream sourceCodeInputStream = null; // input file
27
28                  try {
29                      sourceCodeInputStream = new FileInputStream(filePath);
30                  } catch (FileNotFoundException e) {
31                      System.out.println(filePath + ": Unable to open file");
32                      return;
33                  }
34
35                  if (mode.equals("generate")) {
36                      ParserEvaluator.setGenerateTestCase(true);
37                  }
38
39                  yy = new Yylex(sourceCodeInputStream);
40                  analyzer = new parser(yy);
41                  analyzer.parse();
42
43                  if (mode.equals("evaluate")) {
44                      ParserEvaluator.evaluateTestCase(caseName + "." +
    ParserEvaluator.TESTCASE_EXTENSION);
45                  }
46              }
47          } //Fi main
48
49  :}
50
51
52  terminal STRING_LITERAL, PATH, URL, AUDIO, SWITCH, JSON, SCRIPT, SERVICE, PLAYS,
        OPENS, CLOSES, TOGGLES, INT, STRING, ENDPOINT, REQUEST, SAY, LET, SLEEP,
        ASSIGNMENT, CONCAT, PLUS, MINUS, PRODUCT, DIVISION, OPEN_PAREN, CLOSE_PAREN,
        OPEN_BRACKET, CLOSE_BRACKET, SEMICOLON, COLON, COMMA, IDENTIFIER,
        INTEGER_LITERAL, FEEDBACK;
53  non terminal program, declaration;
```

```
non terminal audio_decl;
non terminal switch_action, switch_decl;
non terminal type, arglist, arglist_inner, arglist_or_error;
non terminal key, value, keyval_pair;
non terminal json_decl, keyval_pairs, json_document;
non terminal expression, term, factor, feedback_call;
non terminal variable_declaration, variable_assignment, special_statement,
    arguments, service_call, statement, statements, code_block, script_decl;

////////////////////////////////////////
// Your parser definition goes here: //
////////////////////////////////////////
program ::= declaration program | /* Epsilon */;
declaration ::= audio_decl | switch_decl | json_decl | script_decl;


/** AUDIO SERVICE **/
audio_decl ::= AUDIO SERVICE IDENTIFIER:E1 PLAYS PATH SEMICOLON
{:
    ParserEvaluator.emitAudioService(E1.toString());
:}
| AUDIO SERVICE IDENTIFIER:E1 PLAYS SEMICOLON
{:
    ParserEvaluator.emitAudioService(E1.toString());
    ParserEvaluator.emitError(ErrorType.AUDIO_NO_PATH);
:};

/** SWITCH SERVICE **/
switch_action ::= OPENS | CLOSES | TOGGLES;
switch_decl ::= SWITCH SERVICE IDENTIFIER:E1 switch_action IDENTIFIER SEMICOLON
{:
    ParserEvaluator.emitSwitchService(E1.toString());
:};


/** COMMON RULES **/
type ::= STRING | INT;
arglist_or_error ::= arglist
    | {: ParserEvaluator.emitError(ErrorType.NO_ARGLIST); :};
arglist ::= OPEN_PAREN CLOSE_PAREN | OPEN_PAREN arglist_inner CLOSE_PAREN;
arglist_inner ::= IDENTIFIER COLON type |
    IDENTIFIER COLON type COMMA arglist_inner;

/** JSON SERVICE **/
json_decl ::= JSON SERVICE IDENTIFIER:E1 arglist_or_error json_document
{:
    ParserEvaluator.emitJsonService(E1.toString());
:};
key ::= REQUEST | ENDPOINT | IDENTIFIER;
value ::= INTEGER_LITERAL | STRING_LITERAL | IDENTIFIER | PATH | URL |
```

```
    json_document;
keyval_pair ::= key:K COLON value:V;
keyval_pairs ::= keyval_pair
    | keyval_pair COMMA
    | keyval_pair COMMA keyval_pairs ;
json_document ::= OPEN_BRACKET CLOSE_BRACKET
    | OPEN_BRACKET keyval_pairs CLOSE_BRACKET;

/** Expressions **/
expression ::= term PLUS expression
    | term MINUS expression
    | term CONCAT expression
    | term;
term ::= factor PRODUCT term
    | factor DIVISION term
    | factor;
factor ::= OPEN_PAREN expression CLOSE_PAREN
    | IDENTIFIER
    | INTEGER_LITERAL
    | STRING_LITERAL
    | feedback_call;
feedback_call ::= FEEDBACK type
    | FEEDBACK
    {:
        ParserEvaluator.emitError(ErrorType.FEEDBACK_NO_TYPE);
    :};

/** SCRIPT SERVICE **/
variable_declaration ::= LET IDENTIFIER COLON type
    | LET IDENTIFIER COLON type ASSIGNMENT expression;

variable_assignment ::= IDENTIFIER ASSIGNMENT expression;

special_statement ::= SAY expression | SLEEP expression;

arguments ::= expression
    | expression COMMA arguments;
service_call ::= IDENTIFIER OPEN_PAREN arguments CLOSE_PAREN
    | IDENTIFIER OPEN_PAREN CLOSE_PAREN;

statement ::= variable_declaration
    | variable_assignment
    | special_statement
    | service_call;
statements ::= statement SEMICOLON
    | statement SEMICOLON statements;
code_block ::= OPEN_BRACKET statements CLOSE_BRACKET
    | OPEN_BRACKET CLOSE_BRACKET;
script_decl ::= SCRIPT SERVICE IDENTIFIER:E1 arglist_or_error code_block
{:
```

```
152     ParserEvaluator.emitScriptService(E1.toString());
153 :};
```

Recursos

Recursos Bàsics

- Mòdul didàctic 3. Anàlisi Sintàctic
- Manual de JLex i CUP

Recursos Complementaris

- Materials de repàs de gramàtiques incontextuals
- Exercicis resolts detallats sobre anàlisi sintàctica.
- Aula de Laboratori de Java de Compiladors

Criteris d'avaluació

La ponderació dels exercicis és la següent:

- Exercici 1: 30%
- Exercici 2: 70%

Pel que fa a l'exercici pràctic de CUP, es valorarà:

- La correcta definició del programa principal que crida l'analitzador.

- L'enllaç correcte de l'analitzador lèxic i l'analitzador sintàctic.
- L'ús correcte de les directives de CUP (prioritats, ...).
- L'absència de conflictes en la gramàtica.
- La correctesa i llegibilitat de les regles sintàctiques.
- L'ús correcte de la taula de símbols, si s'escau.
- L'assoliment dels requisits descrits en l'enunciat.

En concret **es penalitzarà** la implementació de funcionalitats en Java que ja són ofertes per JLex/CUP.

Format i data de lliurament

Cal lliurar un **document en format PDF**, amb la resposta a la pregunta 1 i una justificació de les decisions adoptades a la pregunta 2. Addicionalment, s'hauran de lliurar **un únic fitxer .cup** implementant l'analitzador **a partir de la plantilla** que trobareu juntament a aquest enunciat. **Les solucions que no utilitzin la plantilla proporcionada no s'acceptaran.**

Aquest document s'ha de lliurar a l'espai de **Lliurament i Registre d'AC** de l'aula abans de les **23:59 del dia 17 de novembre de 2020.** **No s'acceptaran lliuraments fora de termini.**