Profesores: José Miguel Cañete Valdeón y Francisco José Galán Morillo

PROCESADORES DE LENGUAJES Proyecto

Lenguaje D (Especificación y Evaluación del Proyecto. Nivel de Expresividad 1)

D es un lenguaje de programación de descripciones. La **descripción** representa un conjunto de individuos relacionados. El programa D (programa en adelante) consta de 5 secciones: (1) declaración de individuos (`INDIVIDUOS`), (2) relaciones binarias entre individuos (`RELACIONES`), (3) tipos (`TIPOS`), (4) tipificación de los individuos (`INICIALIZACION`) y (5) descripciones (`DESCRIPCIONES`):

Ejemplo de localización de secciones en un programa:

```
PROGRAMA

INDIVIDUOS: ...
RELACIONES: ...
TIPOS: ...
INICIALIZACION: ...
DESCRIPCIONES: ...
```

El programa declara un conjunto (finito y no cambiante) de **individuos**. Es lo que se conoce como un dominio cerrado de interpretación. Los individuos se formalizan con nombres en minúscula (se admite guion y subrayado).
Ej:

```
INDIVIDUOS: juan, pedro, ismael, maria, manuela,
            grado-ic, grado-it, grado-is,
            tesis1, publicacion1, articulo1,
            compania1, universidad1,
            eval_alt, eval_ofic
```

El programa también declara un conjunto de **relaciones** entre individuos. Las relaciones se formalizan con predicados binarios con nombres en minúscula (se admite guion y subrayado).
Ej:

```
RELACIONES: publica, obtiene, supera, trabaja
```

En las siguientes definiciones, las variables se formalizan con nombres en minúsculas prefijados con el símbolo _.

Los **tipos** del programa describen clases de individuos que persisten en cualquier descripción del programa (sección `DESCRIPCIONES`). Los tipos se formalizan con predicados unarios cuyos nombres comienzan con una primera letra mayúscula (se admite guion y subrayado). Ej: `Profesor-Titular, Tesis.`
Los tipos pueden ser primitivos y no primitivos.
Los tipos primitivos se definen con una fórmula de la forma: `Tipo(_x) <-` siendo `_x` una variable que representa un individuo.
Los tipos no primitivos se definen no recursivamente con fórmulas de la forma: `Tipo(_x) <- Definición(_x).`
La parte derecha de la definición (`Definición`) se construye desde tipos primitivos, tipos no primitivos y conjunción (`&&`).

Se dice que un individuo `i` es de tipo `Tipo` si `i` pertenece al conjunto de individuos definidos según la semántica de `Tipo`. Se dice que un individuo `i` no es de tipo `Tipo` en caso contrario.
La semántica del tipo se formaliza recursivamente de la siguiente manera:

- (caso base) `Tipo(_x) <-`
  se interpreta como el conjunto de individuos `i` tal que `Tipo(i)` ocurre en la inicialización del programa (sección `INICIALIZACION`).
- (caso recursivo I) `Tipo(_x) <- Tipo1(_x) && ... && Tipok(_x)`
  se interpreta como `Tipo(_x)<- Definición1(_x) && Tipo2(_x) && ... && Tipok(_x)`
  si la definición de `Tipo1` es `Tipo1(_x) <- Definición1(_x)`
- (caso recursivo II) `Tipo(_x) <- Tipo1(_x) && ... && Tipok(_x)`

se interpreta como `Tipo(_x)<- Tipo2(_x) && ... && Tipok(_x)` con `_x` instanciable en el conjunto de individuos de tipo `Tipo1` si la definición de `Tipo1` es `Tipo1(_x) <-` (ver caso base).

Ej:

```
TIPOS:
   Profesor(_x) <-,
   PersonaConVocacion(_x) <-,
   Compania(_c) <-,
   Tesis(_t) <-,
   Articulo(_a) <-,
   Grado(_g) <-,
   ProfesorConVocacion(_x) <- Profesor(_x) && PersonaConVocacion(_x),
   Documento(_d) <- Tesis(_d),
   Documento(_d) <- Articulo(_d)
```

La **inicialización** de los tipos primitivos de los individuos del programa se realiza en la sección `INICIALIZACION`. No es obligatorio que todo individuo tenga tipo, ni que el tipo del individuo sea único pero una vez que éste se asigna permanece a lo largo de todas las descripciones del programa.

.

Ej.

```
INICIALIZACION:
   Profesor(juan), Profesor (manuela),
   PersonaConVocacion(pedro), PersonaConVocacion(juan),
   Grado(grado-it), Grado(grado-is), Grado(grado-it),
   Articulo(articulo1), Tesis(tesis1),
   Compania(compania1)
```

Las **descripciones** del programa expresan relaciones (binarias) entre individuos. Las descripciones son independientes (no dependen unas de otras) y se organizan de forma secuencial. Un número entero positivo identifica cada descripción en el programa.

Ej:

```
DESCRIPCIONES:
1: obtiene(maria,grado-it), obtiene(pedro,grado-is), trabaja(ismael,compania),
   trabaja(maria,compania1), trabaja(manuela,universidad1), trabaja(juan,universidad1),
   publica(juan,tesis1), supera(ismael,grado-ic)


2: publica(juan,articulo1),publica(manuela,publicacion1), publica(ismael,publicacion1),
   trabaja(manuela,universidad1), trabaja(juan,universidad1), obtiene(maria,grado-it),
   obtiene(pedro,grado-is)
```

Cada descripción del programa puede incluir al final un conjunto de consultas.

La **consulta** es una expresión de la forma `?_x tal que (condición(_x))`. La condición de la consulta (ej. `condición`) es una formula construida desde los siguientes recursos: (a) relaciones cuyo primer parámetro es la variable cuestionada (ej. `?_x`), (b) tipos cuyo parámetro es la variable cuestionada (ej. `?_x`), (c) conjunción (`&&`), disyunción (`||`) y (d) paréntesis.

Ej:

```
//consulta de individuos que tiene un grado en ingeniería del software, son profesores
//con vocación y trabajan en la universidad1
?_t tal que (trabaja(_t,universidad1) &&
             obtiene(_t,grado-is) && ProfesorConVocacion(_t)),

//consulta de profesores con publicacion1 o autores del artículo1
?_i tal que ((Profesor(_i) && publica(_i,publicacion1)) || publica(_i,articulo1))
```

Profesores: José Miguel Cañete Valdeón y Francisco José Galán Morillo

EJEMPLO

A continuación, se muestran un ejemplo de programa en el lenguaje D (Nivel de expresividad 1):

```
PROGRAMA

INDIVIDUOS: juan, pedro, ismael, maria, manuela,
            grado-ic, grado-it, grado-is,
            tesis1, publicacion1, articulo1,
            compania1, universidad1,
            eval_alt, eval_ofic

RELACIONES: publica, obtiene, supera, trabaja

TIPOS:
   Profesor(_x) <-,
   PersonaConVocacion(_x) <-,
   Compania(_c) <-,
   Tesis(_t) <-,
   Articulo(_a) <-,
   Grado(_g) <-,
   ProfesorConVocacion(_x) <- Profesor(_x) && PersonaConVocacion(_x),
   Documento(_d) <- Tesis(_d),
   Documento(_d) <- Articulo(_d)

INICIALIZACION:
   Profesor(juan), Profesor (manuela),
   PersonaConVocacion(pedro), PersonaConVocacion(juan),
   Grado(grado-it), Grado(grado-is), Grado(grado-it),
   Articulo(articulo1), Tesis(tesis1),
   Compania(compania1)

DESCRIPCIONES:
1: obtiene(maria,grado-it), obtiene(pedro,grado-is), trabaja(ismael,compania),
   trabaja(maria,compania1), trabaja(manuela,universidad1), trabaja(juan,universidad1),
   publica(juan,tesis1), supera(ismael,grado-ic)

//consulta de individuos que tiene un grado en ingeniería del software, son profesores
//con vocación y trabajan en la universidad1
?_t tal que (trabaja(_t,universidad1) &&
             obtiene(_t,grado-is) && ProfesorConVocacion(_t)),

//consulta de profesores con publicacion1 o autores del artículo1
?_i tal que ((Profesor(_i) && publica(_i,publicacion1)) || publica(_i,articulo1))

2: publica(juan,articulo1),publica(manuela,publicacion1), publica(ismael,publicacion1),
   trabaja(manuela,universidad1), trabaja(juan,universidad1), obtiene(maria,grado-it),
   obtiene(pedro,grado-is)

//consulta de individuos que trabajan en universidad1 y son autores de publicacion1
?_t tal que (trabaja(_t,universidad1) && publica(_t,publicacion1)),


//consulta de profesores o personas con vocación
?_i tal que (Profesor(_i) || PersonaConVocacion(_i))
```

Profesores: José Miguel Cañete Valdeón y Francisco José Galán Morillo

SE PIDE:

Especificación e Implementación de un procesador para programas D compuesto de 4 partes:

1.Analizador Léxico/Sintáctico

2.Analizador Semántico

3.Intérprete

4.Compilador (o Traductor) a lenguaje Java.

Llamaremos Front-end del procesador a las dos primeras partes (Analizador Léxico/Sintáctico, Analizador Semántico) y Back-end a las dos últimas (Intérprete, Compilador).

Analizador Léxico/Sintáctico

[ESPECIFICACIÓN 1] Gramática independiente de tecnología que describa la sintaxis de D.

[IMPLEMENTACIÓN 1] Lexer y Parser Antlr4 para D basado en ESPECIFICACIÓN 1.

Analizador Semántico

[ESPECIFICACIÓN 2] Especificación del analizador semántico que detecte los 2 errores semánticos propuestos. El formato de la especificación debe ser:

```
//OBJETIVO: especificación de analizador semántico que detecte los errores
//propuestos (se debe dar por consola tanto el mensaje de error como la línea del programa
//            en la que se produce dicho error):
//          error 1: definición recursiva de tipos
//          error 2: consulta con uso incorrecto de la variable cuestionada
//DECISIONES DE DISEÑO error 1:
// (DECISIÓN 1.1) …
// (DECISIÓN 1.2) …
//DECISIONES DE DISEÑO error 2:
// (DECISIÓN 2.1) …
// (DECISIÓN 2.2) …

//GRAMÁTICA ATRIBUIDA: …
```

[IMPLEMENTACIÓN 2] Implementación Antlr4 de la gramática atribuida propuesta en ESPECIFICACIÓN 2.

IMPORTANTE: Muestre que su implementación se deriva de la gramática atribuida propuesta añadiendo comentarios que lo aclaren (p.e. qué reglas de la gramática generan qué fragmentos en la implementación).

Intérprete

[ESPECIFICACIÓN 3] Especificación de un intérprete para programa D.

El formato de la especificación debe ser:

```
//OBJETIVO: (no hace falta)
```

```
//DECISIONES DE DISEÑO: explicar cómo se realizarán las interpretaciones de las
//                      diferentes elementos de un programa.
// (DECISIÓN 1) …
// (DECISIÓN 2) …
//  …
//GRAMÁTICA ATRIBUIDA: …
```

[IMPLEMENTACIÓN 3] Implementación Antlr4 de la gramática atribuida propuesta en ESPECIFICACIÓN 3.

IMPORTANTE: Muestre que su implementación se deriva de la gramática atribuida propuesta añadiendo comentarios que lo aclaren (p.e. qué reglas de la gramática generan qué fragmentos en la implementación).

Compilador

[ESPECIFICACIÓN 4] Especificación de un compilador que traduzca programa D a Java.

El formato de la especificación debe ser:

```
//OBJETIVO: (no hace falta)
//EJEMPLOS DE COMPILACIONES: proponer un conjunto de ejemplos de traducción.
//      Dicho conjunto sea válido y razonablemente completo.
//      ejemplo 1
//      …
//      ejemplo n
//DECISIONES DE DISEÑO: explicar cómo se realizarán las traducciones de los diferentes
//                      elementos del programa.
// (DECISIÓN 1) …
// (DECISIÓN 2) …
//  …
//GRAMÁTICA ATRIBUIDA: …
```

[IMPLEMENTACIÓN 4] Implementación Antlr4 de la gramática atribuida propuesta en ESPECIFICACIÓN 4.

IMPORTANTE: Muestre que su implementación se deriva de la gramática atribuida propuesta añadiendo comentarios que lo aclaren (p.e. qué reglas de la gramática generan qué fragmentos en la implementación).

EVALUACIÓN:

Nota asignada al proyecto: Hasta un 6.

La evaluación será *individual* (tanto el proyecto como la prueba de conocimiento/autoría).

Es importante escribir bien las especificaciones (**legibles y simples**), en particular, las gramáticas atribuidas porque serán las herramientas usadas para razonar e implementar soluciones.

El procesador consta de 2 partes principales:

Front-end (Analizador Léxico/Sintáctico y Analizador Semántico). Debe incluir al menos un Analizador Léxico/ Sintáctico.

Back-end (Intérprete y Compilador): Debe incluir al menos un Intérprete o un Compilador.

Ejemplos de procesadores mínimos viables serían:

Analizador Léxico/Sintáctico + Intérprete

Analizador Léxico/Sintáctico + Compilador