Compiladores

An interpreter for a robot language

Autores:

Aguilar Carrejon, José Juan

Lorente García, Ester

Tutor:

Rivero Almeida, José Miguel

1 de Junio de 2015

Índice

1. Objetivo del proyecto

El objetivo principal del proyecto *An interpreter for a robot language* es realizar un intérprete para poder manipular y dirigir un robot *LEGO Mindstorms nxt 2.0*.

Deberemos definir un lenguaje similar al *RobotBASIC*, con las acciones que veamos necesarias para poder realizar acciones simples con el robot. Para acabar traduciendolo al lenguaje que entiende, el *leJOS*.

2. El lenguaje

Funcionalidades

Con las instrucciones de nuestro lenguaje:

- Movimiento directo del robot: avanzar, retroceder, parar, acelerar
- Detección de luz y colores con el sensor de colores
- Detección de tacto con el sensor de tacto
- Detección de obstáculos con el sensor de ultrasonidos

Con bucles/funciones programables por el usuario usando nuestro lenguaje:

- Seguir una línea de cualquier color con el sensor de colores
- Parar cuando el sensor de tacto toque algún obstáculo
- Rodear un objeto que identifique el sensor de ultrasonidos
- etc.

Principales instrucciones

Instrucciones del motor Mx (disponibles M1-M3)

- Constructoras
    - **MOTOR(Int)**: Para asignar el motor Mx a una variable. La variable numérica indica a qué puerto está conectado el motor.
- Consultoras
    - **Mx.getSpeed()**: devuelve la velocidad del motor Mx en grados por segundo.
- Modificadoras
    - **Mx.avanzar():** provoca que el motor Mx avance
    - **Mx.avanzar(Int,Bool)**: provoca que el motor Mx avance lo que indique la variable numérica en grados. La variable Bool indica si la acción es bloqueante: si es true entonces no espera a que finalice la acción.
    - **Mx.retroceder():** provoca que el motor Mx retroceda.
    - **Mx.retroceder(Int,Bool)**: provoca que el motor Mx retroceda lo que indique la variable numérica en grados. La variable Bool indica si la acción es bloqueante: si es true entonces no espera a que finalice la acción.
    - **Mx.parar()**: provoca que el motor Mx se pare.
    - **Mx.setSpeed(Int)**: modifica la velocidad del motor Mx en lo que indique la variable numérica en grados por segundo.

      La velocidad máxima a la que puede ir un motor es de 100*voltaje batería.
- Otras:
    - **Mx.isMoving()**: devuelve si el motor se está moviendo en ese instante.

Instrucciones del sensor de color Sx (disponibles S1-S4)

- **COLOR(Int)**: Para asignar el sensor Sx a una variable. La variable numérica indica en qué puerto está conectado y tiene valor x.
- **getColor()**: devuelve el color que el sensor de color Sx está apuntando.

  A continuación mostramos una tabla con los valores correspondientes a cada color.

Color	Valor
Ninguno	-1
Rojo	0
Verde	1
Azul	2
Amarillo	3
Magenta	4
Naranja	5
Blanco	6
Negro	7
Rosa	8
Gris	9
Gris claro	10
Gris oscuro	11
Cyan	12

Para una buena precisión del color, el objetivo no debería estar a más de 1cm de distancia y no deberían ser colores compuestos de los que indicamos en la tabla.

Instrucciones del sensor de tacto Sx (disponibles S1-S4)

- **TOUCH(Int)**: Para asignar el sensor de tacto Sx a una variable. La variable numérica indica en qué puerto está conectado y tiene valor x.
- **getTouch()**: devuelve si el sensor de tacto Sx siente algo. Si algo toca al sensor, la función devuelve True. En caso contrario, devuelve False.

Instrucciones del sensor de ultrasonidos Sx (disponibles S1-S4)

- **ULTRA(Int)**: Para asignar el sensor ultrasónico Sx a una variable. La variable numérica indica en qué puerto está conectado y tiene valor x.
- **getUltrasonic()**: devuelve la distancia a la que está un objeto delante del sensor de movimiento Sx.

  La distancia máxima a la que puede estar un objeto es de 170cm. Si no tiene ninguno, el valor que devuelve es de 255.

Otras instrucciones

- **sleep(Int)**: para permanecer igual durante los milisegundos que indique la variable numérica.
- **sleep()**: para permanecer igual hasta la pulsación del botón.

Para el uso de otros ficheros como includes tenemos:

- **include paquete/clase**: permite crear objetos de esa clase y usar sus funciones.

Para la creación y uso de objetos disponemos de:

- **OBJECT(clase)**: Para asignar un objeto de la clase a una variable.
- **clase.funcion(params)**: Para llamar a una función de esa clase.

3. Descripción de los módulos

Nuestro proyecto consiste en la realización de tres módulos. A continuación describimos cada uno según el orden que siguen para la realización del proyecto.

Analizador

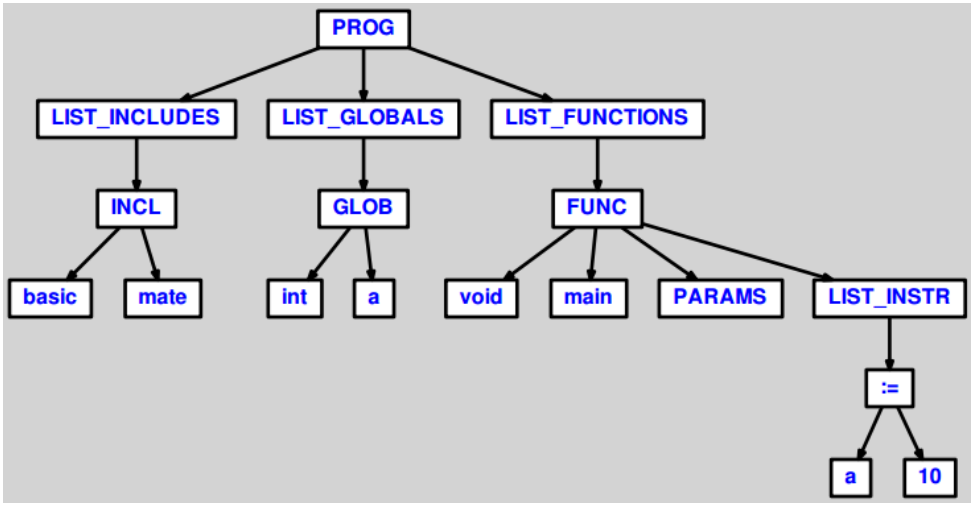
El analizador consiste en generar un lenguaje que nos hemos inventado (parecido al de los laboratorios de esta asignatura). Nuestro lenguaje contiene las instrucciones anteriormente mencionadas que nos sirven para poder mover a un robot e interactuar con él.

En el Apéndice comentamos nuestra gramática con las modificaciones de la gramática inicial usada en los laboratorios del curso.

Como ejemplo usaremos este programa, solo para ver como es la estructura:

```
include basic/mate

int a;

void main()
    a = 10;
endfunc
```

Con el analizador, generamos un árbol AST como el que tenemos a continuación.



Este árbol es una simplificación para ver su estructura:

- LIST_INCLUDES: una lista de librerías INCL con paquete y nombre de librería.
- LIST_GLOBALS: una lista de globales GLOB con tipo y nombre de variable.
- LIST_FUNCTIONS: una lista de funciones FUNC con tipo, nombre de función, parámetros y lista de instrucciones.

Traductor

El traductor recoge nuestro lenguaje y lo traduce a uno que reconoce un robot *LEGO Mindstorms nxt 2.0*. Por ello lo hemos tenido que traducir al lenguaje *leJOS*, un lenguaje Java para los robots de *LEGO Mindstorms*.
En el Apéndice comentamos nuestro traductor con las principales modificaciones sobre el intérprete usado en los laboratorios del curso.

Con el traductor y el árbol que se ha montado antes, generaríamos este programa en leJOS.

```
import basic.mate;
import lejos.nxt.*;
import lejos.util.Delay;

public class estructura {
    private static int a;

    public static void main(String args[]) {
        a = 10;
    }
}
```

Primero tenemos los imports que hemos declarado y los que usa el robot. Luego tenemos la clase con el nombre del fichero que contiene primero las variables globales y luego todas las funciones.

Software externo

Nuestro proyecto consiste en realizar un intérprete para un robot *LEGO Mindstorms nxt 2.0*. Por tanto, usaremos un robot como ése para probar si nuestra traducción funciona.
Tendremos que probar de construir con las piezas que queramos un robot y probar diferentes acciones (generadas por el analizador y traductor) para ver si las interpreta correctamente.

Nuestro principal objetivo ahora mismo es conseguir algo funcional con unas instrucciones básicas y a partir de aquí ir añadiendo más funcionalidades. Se intentará también mejorar la traducción de instrucciones y los tipos de datos en la medida de lo posible debido a que disponemos de una fecha límite de entrega.

4. Descripción de las fases

En la realización de nuestro proyecto se pueden distinguir tres fases importantes. Cada una de ellas ha aportado un cambio grande a la fase anterior.

Primera fase

Esta primera ha sido pensar en el objetivo del proyecto y empezar a pensar las partes del robot *LEGO Mindstorms nxt 2.0* que necesitábamos para poder realizar acciones con él.
Listamos todas las instrucciones que iba a necesitar nuestro lenguaje y realizamos la gramática para poder leer un programa en él. Esta gramática no dista mucho de la que se adjuntó en el informe preliminar.

Segunda fase (Informe preliminar)

En esta segunda fase tradujimos todas las instrucciones que teníamos listadas en la primera fase. A partir del intérprete proporcionado por los profesores de esta asignatura, lo modificamos para tener un traductor. La gran diferencia entre el intérprete y el traductor está en que nosotros solo teníamos que comprobar todos los tipos y no sus valores.
Al término de esta fase, nuestro lenguaje permitía programar código para mover el robot, sus sensores, etc. Todo programable por el usuario a partir de las instrucciones que tenía nuestro lenguaje.

Última fase (Informe final)

En esta última fase, las mejoras que se han hecho son el poder crear objetos de clases y el poder tener librerías y variables globales. Esta mejora la decidimos porque el lenguaje del robot *leJOS* tiene su base en Java que es un lenguaje orientado a objetos y nos facilitaría el poder usar librerías externas programadas en nuestro lenguaje con funciones que realicen acciones complejas del robot.
Al término de esta última fase, nuestro lenguaje permite que el usuario se programe su propio código o ayudarse de funciones disponibles en las librerías externas que le facilitamos. También podría programarse él mismo sus propias librerías siguiendo el formato de los includes.

5. Librerías

A continuación explicaremos las librerías externas de las clases que hemos definido para ayudar al usuario con funciones ya definidas.

Mate

Contiene funciones matemáticas básicas.

La librería completa se puede encontrar en el directorio DOCS/Libs. En nuestro lenguaje está en el fichero *mate.asl*, su correspondiente en *leJOS* en *mate.java* y su árbol AST en *mate.pdf*.

Piloto

Contiene funciones auxiliares para mover el motor. Usa las instrucciones principales de nuestro lenguaje para generar acciones más complejas.

Como la librería es muy extensa, a continuación mostraremos las funciones más destacables.

La siguiente función es la inicializadora del Objeto, es primordial ya que calcula la relación entre eje y radio para así girar los grados indicados de una forma precisa (aunque siempre dependerá de la superficie en la que nos movamos por temas de agarre de las ruedas y un pequeño deslizamiento interno del neumático.

```
void init(motor mizq, motor mder, int s, int e, int r)
	izq = mizq;
	der = mder;
	radio = r;
	sep = s;
	eje = e;
	relacion = eje/radio;
	limiteUS = -1;
	ultraSen = ULTRA(4);
	coloSen = COLOR(4);
	touchSen1 = TOUCH(4);
	touchSen2 = TOUCH(4);
	colorToF1 = -1;
	colorToF2 = -1;
endfunc
```

Su traducción a *leJOS* es:

```
public static void init(NXTRegulatedMotor mizq, NXTRegulatedMotor mder, int s, int e, int r) {
	izq = mizq;
	der = mder;
	radio = r;
	sep = s;
	eje = e;
	relacion = (eje / radio);
	limiteUS = -1;
```

```
        ultraSen = new UltrasonicSensor(SensorPort.S4);
        ColorSensor coloSen = new ColorSensor(SensorPort.S4);
        touchSen1 = new TouchSensor(SensorPort.S4);
        touchSen2 = new TouchSensor(SensorPort.S4);
        colorToF1 = -1;
        colorToF2 = -1;
}
```

Las funciones más destacables a parte de la inicializadora es la función que sigue una línea Bicolor:
Esta función una vez leídos o puestos los colores de la línea a seguir (definiendo color izquierdo y color derecho de dicha línea), intenta seguirla recuperándose de giros en los que la pierda de "vista".
Mientras el sensor de color detecte uno de los dos colores sigue andando hacia adelante con la velocidad que tenía puesta y mientras (si el parámetro check es true) no haya ningún conflicto con los demás sensores.
En el momento que no detecta ningún de los dos colores de la línea intenta recuperarse.

Como tenemos la información del último color visto, sabremos si nos hemos salido por la derecha o la izquierda de la línea. El algoritmo de recuperación, usaremos como ejemplo si nos salimos por la derecha y es totalmente análogo cambiando direcciones de giros si nos salimos por la izquierda, es el siguiente:

Nos salimos por la derecha, giramos hacia la izquierda hasta que vemos el color de la derecha de la línea, aquí sabemos que nos hemos recuperado pero para evitar una recuperación muy lenta en giros pronunciados pasamos a una segunda fase.
Ahora que sabemos que estamos justo en el borde de la derecha seguimos recuperando hacia la izquierda hasta que vemos el color de la izquierda de la línea, por lo tanto estaremos muy cerca del centro de la línea y la recuperación será mejor.

Una vez alcanzado este punto de equilibrio volvemos a avanzar hasta que nos salimos de la línea. Cada recuperación será como máximo de 170º para evitar volver hacia atrás en la línea. En tal caso si el giro fuese demasiado pronunciado y no lograse recuperarse la función devolverá false indicando que no ha podido recuperarse o que ha alcanzado el final de la línea y acabaría.

```
bool followBiColor(bool check)
        ret = true;
        if colorToF1 != -1 and colorToF1 != colorToF2 then
        while (ret) do
                estadoAct = colorSen.getColor();
                estadoAnt = -1;
                movi = (check and checkSensors()) or not check;
                while(movi and (estadoAct = colorToF1 or estadoAct = colorToF2) and ret) do
```

```
movi = mover(1,10,check);
estadoAnt = estadoAct;
estadoAct = colorSen.getColor();
endwhile;
if (movi) then
cont = 0;
//he salido por segunda clausula de colores
if (estadoAnt = colorToF2) then
	//me tengo que recuperar a la izq
	while (movi and estadoAct != colorToF2 and ret and cont<17) do
	movi = girar(1,-10,check);
	cont = cont+1;
	estadoAnt = estadoAct;
	estadoAct = colorSen.getColor();
	endwhile;
	if ( (cont = 17) or (not movi )) then
	// me he intentado recuperar a la izq y no he encontrado el color
	// derecho de la linea colorToF2 o me he parado por sensores
	// si check estaba activado
	ret = false;
	else
	//giro izq y he encontrado color derecho colorToF2 para corregir mejor
	//sigo hasta justo cuando encuentre el color izq colorToF1 asi intento
	//salirme menos
	while (movi and estadoAct = colorToF2 and ret and cont < 17) do
		movi = girar(1,-10,check);
		cont = cont+1;
		estadoAnt = estadoAct;
		estadoAct = colorSen.getColor();
	endwhile;
	if ((cont = 17) or (not movi) or (estadoAct != colorToF1)) then
		ret = false;
	endif;
	endif;
else
	// me tengo que recuperar a la derecha
	//me tengo que recuperar a la izq
	while (movi and estadoAct != colorToF1 and ret and cont<17) do
	movi = girar(-1,-10,check);
	cont = cont+1;
	estadoAnt = estadoAct;
	estadoAct = colorSen.getColor();
	endwhile;
	if ((cont = 17) or (not movi)) then
	// me he intentado recuperar a la izq y no he encontrado el color
	// derecho de la linea colorToF2 o me he parado por sensores
	// si check estaba activado
	ret = false;
	else
	//giro derecha y he encontrado color izq colorToF1 para corregir mejor
	//sigo hasta justo cuando encuentre el color der colorToF2 asi intento
	//salirme menos
	while (movi and estadoAct = colorToF1 and ret and cont<17) do
		movi = girar(-1,-10,check);
		cont = cont+1;
		estadoAnt = estadoAct;
		estadoAct = colorSen.getColor();
	endwhile;
	if ((cont = 17) or (not movi) or (estadoAct != colorToF2)) then
		ret = false;
	endif;
```

```
                    endif;
                endif;
                else
                // movi es false asi que problema con sensores y me bloqueo
                ret = false;
                endif;
        endwhile;
        else
        write "Colors To follow not defined or defined same color on both"
        endif;
        return ret;
endfunc
```

Su traducción a *leJOS* es:

```
public static boolean followBiColor(boolean check) {
        boolean ret = true;
        if(((colorToF1 != -1) && (colorToF1 != colorToF2))) {
            while(ret) {
                int estadoAct = colorSen.getColorID();
                int estadoAnt = -1;
                boolean movi = ((check && checkSensors()) || !check);
                while(((movi && ((estadoAct == colorToF1)||(estadoAct == colorToF2)))&& ret)) {
                        movi = mover(1, 10, check);
                        estadoAnt = estadoAct;
                        estadoAct = colorSen.getColorID();
                }
                if(movi) {
                    int cont = 0;
                    if((estadoAnt == colorToF2)) {
                        while((((movi && (estadoAct != colorToF2)) && ret) && (cont < 17))) {
                                movi = girar(1, -10, check);
                                cont = (cont + 1);
                                estadoAnt = estadoAct;
                                estadoAct = colorSen.getColorID();
                          }
                          if(((cont == 17) || !movi)) {
                                ret = false;
                          } else {
                                while((((movi &&(estadoAct == colorToF2))&& ret)&&(cont < 17))) {
                                movi = girar(1, -10, check);
                                cont = (cont + 1);
                                estadoAnt = estadoAct;
                                estadoAct = colorSen.getColorID();
                                }
                                if((((cont == 17) || !movi) || (estadoAct != colorToF1))) {
                                ret = false;
                                }

                          }
                    } else {
                        while((((movi && (estadoAct != colorToF1)) && ret) && (cont < 17))) {
                                movi = girar(-1, -10, check);
                                cont = (cont + 1);
                                estadoAnt = estadoAct;
                                estadoAct = colorSen.getColorID();
                          }
                          if(((cont == 17) || !movi)) {
                                ret = false;
```

```
                        } else {
                              while((((movi &&(estadoAct == colorToF1))&& ret) &&(cont<17))) {
                              movi = girar(-1, -10, check);
                              cont = (cont + 1);
                              estadoAnt = estadoAct;
                              estadoAct = colorSen.getColorID();
                              }
                              if((((cont == 17) || !movi) || (estadoAct != colorToF2))) {
                              ret = false;
                              }

                        }
                    }
                } else {
                        ret = false;
                }
            }
        } else {
                System.out.println("Colors To follow not defined or defined same color on both");
        }
        return ret;
}
```

La librería completa se puede encontrar en el directorio DOCS/Libs. En nuestro lenguaje está en el fichero *piloto.asl*, su correspondiente en *leJOS* en *piloto.java* y su árbol AST en *piloto.pdf*.

Sonar

Contiene funciones auxiliares para usar el sensor de ultrasonidos. Usa las instrucciones principales de nuestro lenguaje para generar acciones más complejas.

Como la librería es muy extensa, a continuación mostraremos las funciones más destacables.

La siguiente función es la usada en el Ejemplo 1. Realiza *lim* movimientos. En cada uno primero busca la dirección en la parte delantera del robot que esté más despejado. La función grados devuelve los grados que han de girar los motores. Si delante solo hay obstáculos cercanos, entonces gira 90º para realizar el siguiente movimiento.

```
void laberinto(int lim)
    i = 0;
    while (i < lim) do
        g = grados();
        write g;

        M1.retroceder(2 * g,true);
        M2.avanzar(2 * g,false);

        if (g != 180) then
            M1.avanzar(100,true);
            M2.avanzar(100,false);
        endif;

        i = i + 1;
```

```
    endwhile;
endfunc
```

Su traducción a *leJOS* es:

```
public static void laberinto(int lim) {
    int i = 0;
    while((i < lim)) {
        int g = grados();
        System.out.println(g);
        M1.rotate(-(2 * g),true);
        M2.rotate((2 * g),false);
        if((g != 180)) {
            M1.rotate(100,true);
            M2.rotate(100,false);
        }

        i = (i + 1);
    }
}
```

La siguiente función es la que usa la función *laberinto* para saber cuánto ha de girar para ir hasta un sitio despejado. Primero hace un barrido entre 75º de izquierda a derecha. En cada visualización, busca la dirección más despejada. Si delante todo son obstáculos, entonces devuelve 180 que significa que ha de girar porque por ahí no puede seguir y así el robot girará y seguirá buscando.

```
int grados()
    girarUltra(- 75);

    grad = 0;
    dist = 0;
    i = -65;
    cerca = true;
    while i <= 75 do

        if (U.getUltrasonic() > dist) then
            dist = U.getUltrasonic();
            grad = i;
        endif;

        if (U.getUltrasonic() > 50) then
            cerca = false;
        endif;

        girarUltra(10);
        i = i + 10;
    endwhile;

    girarUltra(- 65);

    if (cerca) then grad = 180; endif;

    return grad;
endfunc
```

Su traducción a *leJOS* es:

```
public static int grados() {
    girarUltra(-75);
    int grad = 0;
    int dist = 0;
    int i = -65;
    boolean cerca = true;
    while((i <= 75)) {
        if((U.getDistance() > dist)) {
            dist = U.getDistance();
            grad = i;
        }

        if((U.getDistance() > 50)) {
            cerca = false;
        }

        girarUltra(10);
        i = (i + 10);
    }
    girarUltra(-65);
    if(cerca) {
        grad = 180;
    }

    return grad;
}
```

La librería completa se puede encontrar en el directorio DOCS/Libs. En nuestro lenguaje está en el fichero *sonar.asl*, su correspondiente en *leJOS* en *sonar.java* y su árbol AST en *sonar.pdf*.

6. Ejemplos de programa

Ejemplo 1

Este programa hace uso de la librería *sonar* para intentar moverse por un laberinto. En esta librería, se hacen uso de dos motores para mover las ruedas del robot, un sensor ultrasónico y un motor para hacer un barrido con el sensor. Si éstos no están conectados correctamente en el robot, la librería no garantiza su funcionamiento correcto.

Las limitaciones de este programa son:

- El barrido del sensor ultrasónico solo puede ser puede realizar por la parte frontal del robot. El cable que conecta el sensor al robot impide que el barrido se pueda hacer en 360º. A causa de esto, el recorrido se hace más lento.
- Por problemas de sujeción de piezas, el movimiento constante del motor del sensor ultrasónico hace que se suelte su anclaje. A causa de esto, cuando lleva varios movimientos el sensor no empieza el barrido en su centro y por ello existe una desviación entre la dirección central del sensor y la de los motores de las ruedas del robot.

Primero mostraremos el programa principal en nuestro lenguaje.

```
include basic/sonar

void main()
    M1 = MOTOR(1);
    M2 = MOTOR(2);

    MU = MOTOR(3);
    U = ULTRA(4);
    s = OBJECT(sonar);
    s.init(M1,M2,MU,U);
    s.laberinto(10);
endfunc
```

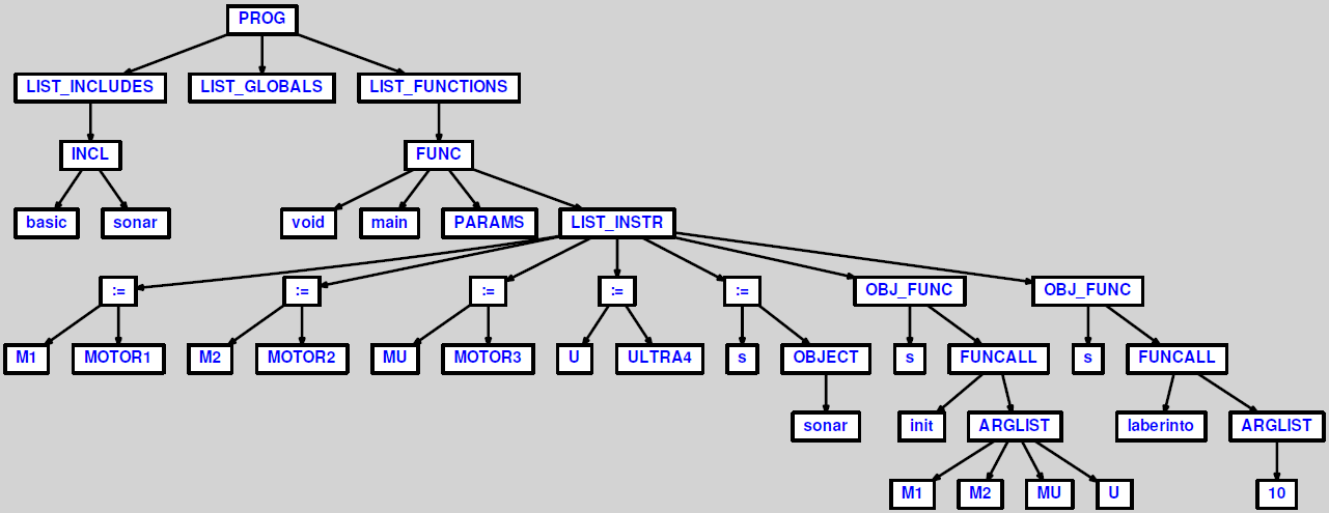
Gracias a la librería, el programa principal solo se encarga de llamar a la librería con los dos motores de las ruedas, el sensor ultrasónico y el motor del sensor.
Luego se llama a la función laberinto a la que se le pasa el número de movimientos que queremos que realice. En este caso, queremos que haga 10 movimientos dentro del laberinto.

A continuación mostraremos su traducción a *leJOS*.

```
import basic.sonar;
import lejos.nxt.*;
import lejos.util.Delay;

public class ejemplo1 {
    public static void main(String args[]) {
        NXTRegulatedMotor M1 = Motor.A;
        NXTRegulatedMotor M2 = Motor.B;
        NXTRegulatedMotor MU = Motor.C;
        UltrasonicSensor U = new UltrasonicSensor(SensorPort.S4);
        sonar s = new sonar();
        s.init(M1, M2, MU, U);
        s.laberinto(10);
    }
}
```

La traducción es una clase simple con un *import* para la librería y una función main con la creación del objeto *sonar* y las llamadas a funciones del objeto.

Y por último el árbol AST generado para ver su estructura.



En el árbol vemos que la *LIST_INCLUDES* no está vacía y contiene la librería *sonar* en el paquete *basic*. Luego tenemos la *LIST_GLOBALS* que en este caso está vacía porque no usamos ninguna variable global. Y finalmente tenemos las *LIST_FUNCTIONS* con una sola función main para crear un objeto y llamar a funciones de la clase *OBJ_FUNC*: su hijo izquierdo es la variable del objeto y derecho la función.

Ejemplo 2

Este programa mueve el robot para seguir un objeto cercano al sensor ultrasónico.

Hace uso de la librería *sonar* para saber los grados que han de moverse los motores para ir en la dirección deseada.

Este programa hace uso de la librería *sonar* para intentar mover el robot para seguir al objeto más cercano que tiene delante. Como se ha explicado en el Ejemplo 1, si los motores y el sensor no están correctamente conectados, no se podrá garantizar el correcto funcionamiento.

Las limitaciones de este programa son las mismas que las del Ejemplo 1: el barrido no es de 360º y el constante movimiento del sensor hace que se suelten las piezas de anclaje.

Primero mostraremos el programa principal en nuestro lenguaje.

```
include basic/sonar

// mueve el robot siguiendo un objeto

void main()
    M1 = MOTOR(1);
    M2 = MOTOR(2);

    MU = MOTOR(3);
    U = ULTRA(4);
    s = OBJECT(sonar);
    s.init(M1,M2,MU,U);
    s.sigueObjeto(10);
endfunc
```

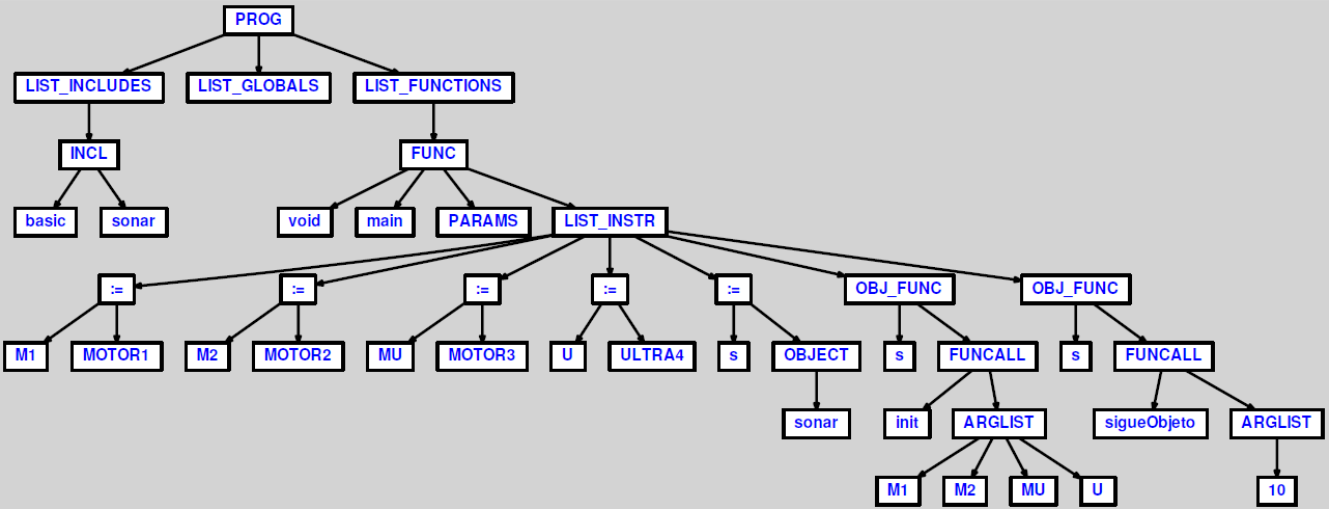
La estructura del programa es la misma que la del Ejemplo 1. Primero declara los motores y el sensor ultrasónico que usará y se lo pasa a la librería. Por último, usa una nueva función sigueObjeto al que le pasa el número de movimientos que quiere realizar.

A continuación mostraremos su traducción a *leJOS*.

```
import basic.sonar;
import lejos.nxt.*;
import lejos.util.Delay;

public class ejemplo2 {
    public static void main(String args[]) {
        NXTRegulatedMotor M1 = Motor.A;
        NXTRegulatedMotor M2 = Motor.B;
        NXTRegulatedMotor MU = Motor.C;
```

```
        UltrasonicSensor U = new UltrasonicSensor(SensorPort.S4);
        sonar s = new sonar();
        s.init(M1, M2, MU, U);
        s.sigueObjeto(10);
    }
}
```

Y por último el árbol AST generado para ver su estructura.



Ejemplo 3

Este programa realiza un pequeño “baile” y describe un movimiento imitando las iniciales de la asignatura “CL”, después pasa a modo seguimiento para seguir una línea bicolor.

Hace uso de la librería *piloto* para saber los grados que han de moverse los motores para ir en la dirección deseada.

Primero mostraremos el programa principal en nuestro lenguaje.

```
include basic/piloto

void main()
    m1 = MOTOR(1);
    m2 = MOTOR(2);
    m1.setSpeed(180);
    m2.setSpeed(180);

    p = OBJECT(piloto);
    p.init(m1,m2,9,12,2);
    p.setSensors(TOUCH(2),TOUCH(3),COLOR(1),ULTRA(4));

    ret = baileCL(p);
    if ret then
```

```
        write "baile realizado con exito"
    endif;
    write "pasamos a modo SEGUIMIENTO";
    p.readColorAndSet();
    ret = p.followBiColor(true);
    if not ret  then
        write "linea acabada o no he podido recuperarme :S";
    endif;
    write "pulsa cualquier boton para salir de este programa";
    sleep();


endfunc

bool baileCL(piloto p1)
    ret = true;
    vel = p1.getVelo();
    write "baile bienvenida demostracion CL";
    p1.setVelo(720);
    ret = p1.girarInSitu(1,720,true);
    sleep(2000);
    // letra C
    p1.setVelo(360);
    ret = p1.mover(1,360,true);
    ret = p1.girar(-1,90,true);
    ret = p1.mover(1,1080,true);
    ret = p1.girar(-1,90,true);
    ret = p1.mover(1,360,true);
    if ret then
        write "C"
    endif;
    sleep(2000);

    //posiciono para letra L
    p1.setVelo(720);
    ret = p1.girar(-1,90,true);
    ret = p1.mover(1,1080,true);
    ret = p1.girar(1,90,true);
    ret = p1.mover(1,360,true);
    ret = p1.girarInSitu(1,90,true);
    sleep(2000);

    //letra L
    p1.setVelo(360);
    ret = p1.mover(1,1080,true);
    ret = p1.girar(-1,90,true);
    ret = p1.mover(1,360,true);
    if ret then
        write "L"
    endif;
    sleep(2000);

    //segundo baile victoria!!!
    p1.setVelo(720);
    ret = p1.girarInSitu(1,1440,true);

    p1.setVelo(vel);
    return ret;
endfunc
```

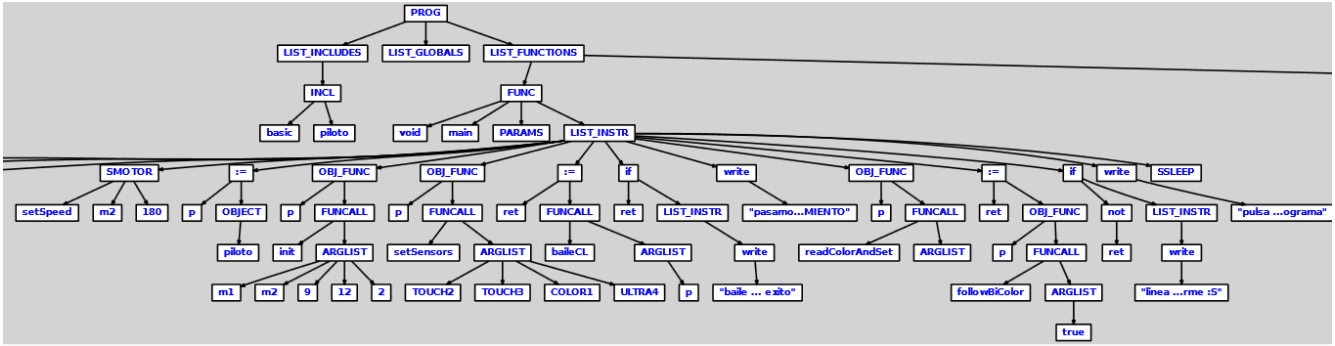
A continuación mostraremos su traducción a *leJOS*.

```
import basic.piloto;
import lejos.nxt.*;
import lejos.util.Delay;

public class sigueBi {
	public static boolean baileCL(piloto p1) {
	boolean ret = true;
	int vel = p1.getVelo();
	System.out.println("baile bienvenida demostracion CL");
	p1.setVelo(720);
	ret = p1.girarInSitu(1, 720, true);
	Delay.msDelay(2000);
	p1.setVelo(360);
	ret = p1.mover(1, 360, true);
	ret = p1.girar(-1, 90, true);
	ret = p1.mover(1, 1080, true);
	ret = p1.girar(-1, 90, true);
	ret = p1.mover(1, 360, true);
	if(ret) {
		System.out.println("C");
	}

	Delay.msDelay(2000);
	p1.setVelo(720);
	ret = p1.girar(-1, 90, true);
	ret = p1.mover(1, 1080, true);
	ret = p1.girar(1, 90, true);
	ret = p1.mover(1, 360, true);
	ret = p1.girarInSitu(1, 90, true);
	Delay.msDelay(2000);
	p1.setVelo(360);
	ret = p1.mover(1, 1080, true);
	ret = p1.girar(-1, 90, true);
	ret = p1.mover(1, 360, true);
	if(ret) {
		System.out.println("L");
	}

	Delay.msDelay(2000);
	p1.setVelo(720);
	ret = p1.girarInSitu(1, 1440, true);
	p1.setVelo(vel);
	return ret;
	}

	public static void main(String args[]) {
	NXTRegulatedMotor m1 = Motor.A;
	NXTRegulatedMotor m2 = Motor.B;
	m1.setSpeed(180);
	m2.setSpeed(180);
	piloto p = new piloto();
	p.init(m1, m2, 9, 12, 2);
	p.setSensors(new TouchSensor(SensorPort.S2), new TouchSensor(SensorPort.S3), new
ColorSensor(SensorPort.S1), new UltrasonicSensor(SensorPort.S4));
	boolean ret = baileCL(p);
	if(ret) {
		System.out.println("baile realizado con exito");
	}
```

```
        System.out.println("pasamos a modo SEGUIMIENTO");
        p.readColorAndSet();
        ret = p.followBiColor(true);
        if(!ret) {
                System.out.println("linea acabada o no he podido recuperarme :S");
        }

        System.out.println("pulsa cualquier boton para salir de este programa");
        Button.waitForAnyPress();
        }
}
```

Y por último el árbol AST generado para ver su estructura.

Como el árbol es bastante extenso, a continuación solo mostraremos la parte de la función main.



El resto del árbol generado se puede consultar en DOCS/Libs en el archivo sigueBi.pdf

7. Distribución y compartición del trabajo

La distribución del trabajo ha sido equitativa para los dos miembros. Ambos hemos realizado conjuntamente el analizador y el traductor.

8. Apéndice

Gramática (*Asl.g*)

La gramática la hemos desarrollado a partir de la gramática usada en los laboratorios de esta misma asignatura para ASL.

Primero mostraremos todas las reglas para analizar nuestro lenguaje.

```
// A program is a list of includes, a list of global variables and a list of functions
prog    : list_includes list_globals list_funcs -> ^(PROG list_includes list_globals list_funcs)
        ;

list_includes
        : include* -> ^(LIST_INCLUDES include*)
        ;

list_globals
        : (global ';')* -> ^(LIST_GLOBALS global*)
        ;

list_funcs
        : func+ EOF -> ^(LIST_FUNCTIONS func+)
        ;

include : INCLUDE p=ID'/'f=ID -> ^(INCL $p $f)
        ;

global  : t=ID id=ID -> ^(GLOB $t $id)
        | TIPO ID -> ^(GLOB TIPO ID)
        ;

// A function has a name, a list of parameters and a block of instructions
func    : t=ID id=ID params block_instructions return_stmt ENDFUNC -> ^(FUNC $t $id params block_instructions return_stmt)
        | TIPO ID params block_instructions return_stmt ENDFUNC -> ^(FUNC TIPO ID params block_instructions return_stmt)
        | VOID ID params block_instructions ENDFUNC -> ^(FUNC VOID ID params block_instructions)
        ;

// The list of parameters grouped in a subtree (it can be empty)
params  : '(' paramlist? ')' -> ^(PARAMS paramlist?)
        ;

// Parameters are separated by commas
paramlist
        : param (','! param)*
        ;

// Parameters with & as prefix are passed by reference
// Only one node with the name of the parameter is created
param   : TIPO id=ID -> ^(PVALUE TIPO ID)
        | t=ID id=ID -> ^(PVALUE $t $id)
        ;

// A list of instructions, all of them gouped in a subtree
block_instructions
        : instruction (';' instruction)* -> ^(LIST_INSTR instruction+)
        ;
```

```
// The different types of instructions
instruction
      : assign       // Assignment
      | ite_stmt     // if-then-else
      | while_stmt   // while statement
      | funcall      // Call to a procedure (no result produced)
      | read         // Read a variable
      | write        // Write a string or an expression
      |             // Nothing
      | motor        //
      | obj_fun
      | sleep        //
      ;

obj_fun : ID '.' funcall -> ^(OBJ_FUNC ID funcall)
      ;

motor   : ID '.' AVANZAR '(' (e1=expr (','e2=expr)?)? ')' -> ^(SMOTOR AVANZAR ID $e1? $e2?)
      | ID '.' PARAR '('')' -> ^(SMOTOR PARAR ID)
      | ID '.' MSETTER '(' expr ')' -> ^(SMOTOR MSETTER ID expr)
      ;

sleep   : SLEEP '(' expr ')' -> ^(SSLEEP expr)
      ;

// Assignment
assign  : esq eq=EQUAL expr -> ^(ASSIGN[$eq,":="] esq expr)
      ;

esq     : ID
      ;

// if-then-else (else is optional)
ite_stmt: IF^ expr THEN! block_instructions (ELSE! block_instructions)? ENDIF!
      ;

// while statement
while_stmt
      : WHILE^ expr DO! block_instructions ENDWHILE!
      ;

// Return statement with an expression
return_stmt
      : RETURN^ expr ';'!
      ;

// Read a variable
read    : READ^ esq
      ;

// Write an expression or a string
write   : WRITE^ (expr | STRING )
      ;

// Grammar for expressions with boolean, relational and aritmetic operators
expr    : boolterm (OR^ boolterm)*
      ;
```

```
boolterm: boolfact (AND^ boolfact)*
    ;

boolfact: num_expr ((EQUAL^ | NOT_EQUAL^ | LT^ | LE^ | GT^ | GE^) num_expr)?
    ;

num_expr: term ( (PLUS^ | MINUS^) term)*
    ;

term    : factor ( (MUL^ | DIV^ | MOD^) factor)*
    ;

factor  : (NOT^ | PLUS^ | MINUS^)? atom
    ;

// Atom of the expressions (variables, integer and boolean literals).
// An atom can also be a function call or another expression
// in parenthesis
atom    : ID
    | INT
    | m=MOTOR '(' b=INT ')' -> ^(DMOTOR[m, "MOTOR"+$b.text])
    | s=SENSOR '(' b=INT ')' -> ^(DSENSOR[s, $s.text+$b.text])
    | (b=TRUE | b=FALSE) -> ^(BOOLEAN[$b,$b.text])
    | funcall
    | obj_fun
    | '('! expr ')'!
    | ID '.' (b=GETCOLOR | b=GETULTRA | b=GETTOUCH) '(' ')' -> ^(GSENSOR $b ID)
    | ID'.' b=MGETTER '(' ')' -> ^(GMOTOR $b ID)
    | OBJECT '(' ID ')'-> ^(OBJECT ID)
    ;

// A function call has a lits of arguments in parenthesis (possibly empty)
funcall : ID '(' expr_list? ')' -> ^(FUNCALL ID ^(ARGLIST expr_list?))
    ;

// A list of expressions separated by commas
expr_list
    : expr (','! expr)*
    ;
```

Y por último mostraremos solo los token reservados para el uso del robot.

```
TIPO        : ('int'|'bool'|'motor'|'touch'|'ultra'|'color');
AVANZAR      : 'avanzar' | 'retroceder' ;
PARAR       : 'parar' ;
MSETTER      : ('setSpeed' | 'setPower');
MGETTER      : ('getSpeed' | 'getPower' | 'isMoving' );
GETCOLOR     : 'getColor' ;
GETULTRA     : 'getUltrasonic' ;
GETTOUCH     : 'getTouch' ;
MOTOR       : 'MOTOR';
SENSOR      : ('ULTRA'|'TOUCH'|'COLOR');
OBJECT      : 'OBJECT';
SLEEP       : 'sleep' ;
```

Traductor (*Interp.java*, *Data.java*, *Stack.java*)

El traductor lo hemos desarrollado a partir del intérprete usado en los laboratorios de esta misma asignatura para los ASL.

Como el código modificado es bastante extenso, resumimos los cambios más notables.

- La traducción se genera en archivos *.java* con el mismo nombre del fichero *.asl*
- No generamos indentación. Para ello, al finalizar la traducción usamos el comando *astyle* para que nos lo idente con los estándares de Java.
- La traducción se genera instrucción a instrucción sobre cada uno de los archivos.
    1. La traducción empieza con el archivo principal.
    2. Por cada include, se traduce su contenido en un archivo .java del include siguiendo estos mismo pasos 1 a 4.
    3. Se traducen las definiciones de variables globales.
    4. Se traduce el archivo principal.
- En Interp.java guardamos:
    - FuncName2Tree: guarda para cada nombre de función del mismo fichero su ASLTree. Nos sirve para saber si una función del propio fichero se llama correctamente.
    - IncludeName2Tree: guarda para fichero include su FuncName2Tree. Nos sirve para saber si se llama correctamente una función de alguno de los ficheros includes.
    - list_includes: guarda el nombre del paquete y fichero de cada include.
    - ruta: ruta del fichero que se está traduciendo.
    - filename: nombre del fichero que se está traduciendo.
- Para simplificar, los ficheros include no pueden tener otros includes.
- Para simplificar, las variables globales se definen luego de los includes y antes de las funciones. El valor se les asigna dentro de las funciones.
- Se obliga a que se declare una función *void main* sin parámetros en el archivo principal, porque será la función principal en la traducción a Java.
    - Si la función main no tiene un bucle, no se notarán las acciones sobre los motores que no sean bloqueantes.
- Las traducciones de las funciones serán *public static tipo nombre (parámetros)*.
    - Debido a que en Java todo se pasa por valor (los objetos se pasan por referencias por valor), en nuestra gramática eliminamos el paso por referencia ya que la traducción implicaría desarrollar clases envolventes con consultoras y modificadoras.

    - El nombre de una función nunca puede ser el nombre del fichero que la incluye porque en Java esa función es la constructora. En nuestro lenguaje la constructora de un objeto es siempre la constructora por defecto.
- Las llamadas a funciones se hacen de funciones que existan del mismo tipo, que usan el número y tipo de parámetros que pide y que devuelven el tipo correcto (si no son void).
    - En Java, los returns se pueden hacer en cualquier punto de una función, siempre y cuando todos los caminos llevan a uno y luego de uno de ellos no queden instrucciones por ejecutar. Para simplificar, en nuestra gramática solo dejamos usar un return al final.
- En nuestro lenguaje las declaraciones de tipo no se hacen explícitas sino implícitas. El tipo de una variable se lo damos cuando se le asigna el valor de ese tipo.
    - En ningún caso se podrá cambiar el tipo de esa variable, ya que en Java no está permitido. Para ello, usamos los tipos de data: INTEGER, BOOLEAN, MOTOR, ULTRA, TOUCH, COLOR, OBJECT.
    - Los tipo OBJECT son los que nos permiten crear objetos de clases. En ningún caso se podrá cambiar una variable objeto de una clase por otra clase distinta.
- En *Data.java* no guardamos valores porque nuestro proyecto no consiste en interpretar sino en traducir. Solo guardamos tipos para comprobar que todas las instrucciones se realizan con la tipación correcta.
    - Para los tipo OBJECT, se guarda además la clase a la que hace referencia. De este modo, podremos comprobar fácilmente el cambio de tipo objeto de una clase a otra.
- En *Stack.java* disponemos de dos HashMaps:
    - *CurrentAR*: guarda las variables locales de cada función.
    - *GlobalMap*: guarda las variables locales del fichero.

  En ningún caso se podrá tener dos variables locales o globales con el mismo nombre.