Robolang

Magí Toneu, Víctor Massagué i Pau Argelaguet

Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya

Abstract

Robolang és un llenguatge de programació pensat per a facilitar la programació dels robots Lego Mindstorms[1]. Concretament, ha estat dissenyat per al model NXT, però la seva naturalesa el fa fàcilment extensible a altres models de la marca.

Per aconseguir això, Robolang incorpora una sintaxi simple i certes característiques similars a les dels llenguatges de *scripting*, per tal de disposar d'un entorn de desenvolupament àgil i ràpid. A més, també inclou comandes predefinides per a accions comunes del robot, que altrament comportarien un número significatiu de línies de codi.

Finalment, Robolang inclou un compilador que, agafant com a entrada el programa, el compila i el volca al robot automàticament (sempre que aquest estigui connectat via USB o Bluetooth).

[1] https://www.lego.com/es-es/mindstorms

1. Definició del llenguatge

1.1. Descripció general

Robolang és un llenguatge interpretat per una màquina virtual[2], de tipat fort i dinàmic. Els fitxers que contenen codi Robolang són fitxers de text pla amb l'extensió `.rl`.

L'execució del codi és totalment seqüencial, començant per la primera línia del fitxer de text i acabant per la última en estricte ordre d'aparició. No hi ha cap funció `main` ni similars[3].

El format del codi està inspirat en els llenguatges com C/C++: els blocs estan limitats per claudàtors (`{, }`), les instruccions han d'acabar en punt i coma (`;`) i els comentaris poden ser d'una única línia (`// comentari`) o de múltiples (`/* comentari */`).

Com es comenta més endavant en els tipus de dades, no tenim tipus de dades definits explícitament, i això inclou les funcions. Per a definir-les, es fa servir la paraula clau `def` (d'una manera similar a com ho fa Python) i no s'especifica de tipus de retorn. En el cas que una funció no contingui cap expressió `return`, es considera que no retorna res (similar a una funció `void` de C).

Les estructures de control de flux també són molt similars als llenguatges anteriorment mencionat. Tenim els habituals `if, elif, else` i bucles del tipus `while` i `for` (en aquest cas, la seva sintaxi està especialment pensada

[2]Tot i ser interpretat, al córrer sobre la JVM el *bytecode* generat és molt eficient i a més disposem d'un JIT.

[3]Una funció definida com a *main* es comporta com qualsevol altra funció.

per recórrer arrays). També suportem els habituals operadors de comparació `<, >, <=, >=, ==, !=`.

Conté les operacions bàsiques en reals (suma, resta, multiplicació, divisió i mòdul), no existeixen operacions entre MR, i existeix l'operador afegir en arrays i es representa de la següent forma: `a = a + [1,2,3]`, on a és un array.

Per accedir a sensors i motors, fem servir el símbol del dòlar (`$A, $1`). L'expressió ens retornarà una referència als ports (A, B, C, 1, 2, 3, 4) per a efectuar operacions directament a un motor o a un sensor.

1.2. Tipus de dades i estructures

A Robolang, el tipus de les variables no es defineix explícitament, sinó que el compilador és capaç d'inferir-los. Això no vol dir que no existeixin, és a dir, sí que hi ha comprovació de tipus (inferit) quan es realitzen operacions entre variables o crides a funcions.

Els tipus i les estructures suportades són relativament pocs, ja que s'ha optat per la simplicitat del llenguatge reduïnt-los als estrictament necessaris per a operar el robot. Són els següents:

- **Cadenes de text**. Conjunt de caràcters alfanumèrics, com `string` a la majoria de llenguatges.
- **Enters**. Nombres enters, com `int` a la majoria de llenguatges. La seva longitud depèn de la plataforma on s'estigui executant, en el cas del NXT, 32 bits.
- **Reals**. Nombres enters de doble precisió, com `double` a la majoria de

llenguatges. Noti's que no tenim nombres de coma flotant de precisió simple.

- **Booleans**. Valor cert (`true`) o fals (`false`). En el cas de ser avaluats en una expressió numèrica, prenen el valor de `1` i `0` respectivament.

- **Arrays**. Estructura de dades que conté diferents valors del mateix tipus. La seva llargada s'ajusta automàticament a mesura que se li afegeixen/treuen elements.

- **MR**. Indica moviment o rotació, de la forma: x-y, on x té un valor d'entre {mf,mb,ra,rh}, mf indica moviment frontal, mb moviment cap enrere, ra rotació antihoraria i rh rotació horària.

1.3. Funcions predefinides

Una part fonamental del llenguatge són les funcions predefinides, que permeten fer interaccions habituals amb el robot d'una manera molt més ràpida i eficient. Seguidament adjuntem un recull de les més importants.

Certes funcions dependran de l'estat actual del robot a més de la seva estructura física (quins motors i quins sensors té muntats).

- `move(paràmetre_tipus_MR)`, el robot executa l'acció que es passa per paràmetre.

- `move_front(unitats)` i `move_back(unitats)`

- `rotate(unitats)`, el sentit és el signe de les unitats (negatiu: sentit horari, positiu: sentit antihorari).

- `distance(angle)`, retorna la distància que hi ha fins l'objecte més proper a la direcció donada.
- `explore()`, el robot es mou en línia recta fins que troba un obstacle, analitza l'entorn i canvia la trajectòria.
- `follow_line(color)`, el robot segueix una línia del color que li passis com a paràmetre (degut a les limitacions del sensor només podrà ser blanc, 0, o negre, 1).
- `light(x, r, g, b)` Fa llum de color r, g, b si al port x hi ha connectat un sensor lluminós.
- `print(x)`, imprimeix per la pantalla del robot la representació textual del valor passat com a paràmetre.

1.4. Arquitectura

Robolang consisteix d'un compilador que crea codi per a ser executat a Lego NXT. Per a fer-ho, empra ANTLR[4] (en la seva versió 3) i leJOS[5], un *firmware* alternatiu que permet executar programes Java a la plataforma. Fa servir una versió modificada de la JVM i proporciona al desenvolupador un seguit de APIs que permeten interactuar amb el *brick* i els diferents sensors i motors del robot.

Robolang genera codi interpretable (*bytecode*) per aquesta màquina virtual. Per qüestions de practicitat, si el robot està connectat, compila el codi, l'enllaça i el transfereix.

[4]http://www.antlr3.org
[5]http://www.lejos.org

El llenguatge està elaborat com a un projecte Maven[6], el gestor de paquets recomanat per a projectes grans elaborats amb ANTLR[7]. S'ha optat per aquesta alternativa per la raó que gestiona dependències, compila i empaqueta els fitxers d'una manera molt més automàtica, ràpida i còmoda.

Quan es munta una *release* de Robolang, es genera un JAR[8] que interpreta (podent generar l'arbre amb `-dot`) mitjançant ANTLR i després aquest programa es compila, s'enllaça i s'executa usant les eines que proporciona leJOS (`nxcompile, nxlink, nxupload`).

[6]https://maven.apache.org

[7]https://theantlrguy.atlassian.net/wiki/display/ANTLR3/Building+ANTLR+Projects+with+Maven

[8]De fet, es generen dos JAR, un dels quals és autocontingut i inclou totes les dependències necessàries.

2. Exemples de codi

2.1. Joc de proves 1

En aquest primer joc podem veure una de les maneres en que pot ser útil fer servir els arrays. Creem l'array a, i mitjançant el bucle for podem anar executant les diferents accions que conté el vector.

```
x = 2;
a = [mf-10,mb-3.4,ra-3,rh-25];
l = a[x];
for (i in a) {
    move(i);
}
for(i in [rh-2,mf-4,ra-2]){
    move(i);
}
```

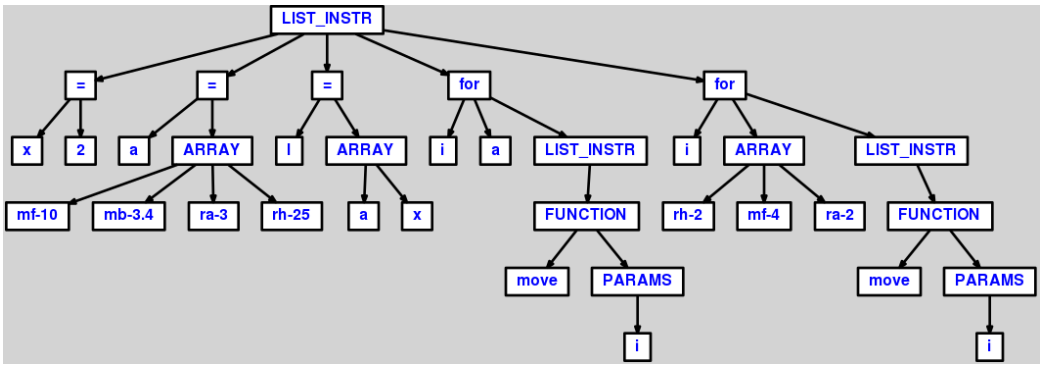

Figura 1: Arbre generat joc proves 1

2.2. Joc de proves 2

A continuació podem veure un joc de proves en el que fem ús de les diferents clàusules de control de flux i d'iteració, i també la definició d'una funció.

```
if(distance(45) > distance(-45)) {
    rotate(45);
} else {
    rotate(-45);
}
while(distance() > 0.5){
    move_front(0.5);
}
def random_moves(){
    moves = [mf-4,ra-45,mb-10,rh-3,mf-7];
    for(i in moves){
        move(i);
    }
}
i = 0;
while(i < 7) {
    random_moves();
    i = i + 1;
}
```

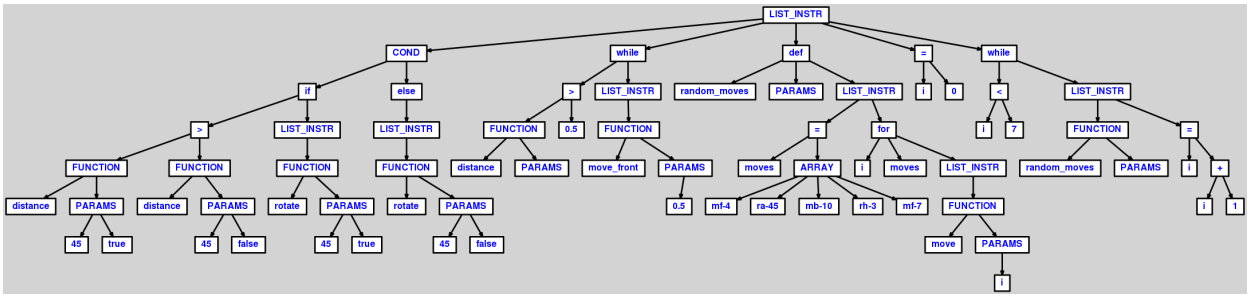

Figura 2: Arbre generat joc proves 2

2.3. Joc de proves 3

En aquest joc de proves bàsicament analitzem les clàusules condicionals, operadors de comparació. També com es pot observar una definició de funció i concatenació de dos o més arrays utilitzant el signe "+ï referència als actuadors o sensors del robot amb el signe $VAR.

```
def shoot2(a) {
    while(a > 0) {
        $x1 = true;
        $x1 = false;
        a = a-1;
    }
}
x = 2;
y = x*3;
b = [rh-45,mf-10];
if(x < y) {
  for(i in [mf-2,mb-5,rh-90]) {
    move(i);
```

```
    x = x + i;
   }
} elif (x > y and true) {
    move_back(5);
    while (x > 0 and y >= x) {
        shoot2(2);
        move_front(1);
    }
} elif (true or y < x) {
    explore();
} else { b = b[2] + [mf-5]; }
```

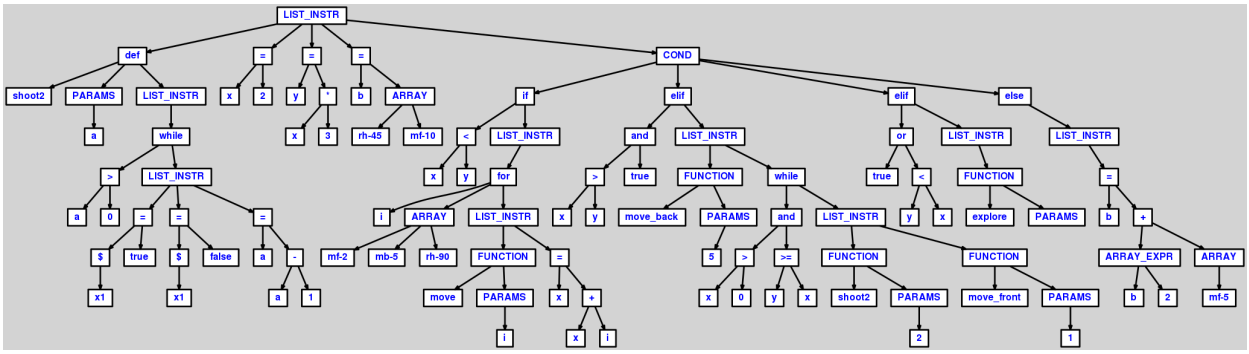

Figura 3: Arbre generat joc proves 3

2.4. Joc de proves 4

Aquí podem veure l'ús de comentaris (segueixen la mateixa estructura que en c++), i algunes de les funcions predefinides entre d'altres.

```
/*
```

```
*   Comentari de
*   múltiples
*   linies.
*/

//Definim una funció que intenta allunyar un objecte si aquest es proper
def allunya(){
    while(distance(0,true) < 0.3) {
        shoot(1);
    }
}
//Li assignem a la variable "a" un moviment frontal de 0.3 unitats.
a = mf-0.3;
i = 0;
while(i < 3){
allunya();
    //Allunyem l'objecte i ens avancem cap a ell.
    move(a);
    i = i + 1;
}
rotate(-45);
follow_line(1);
```

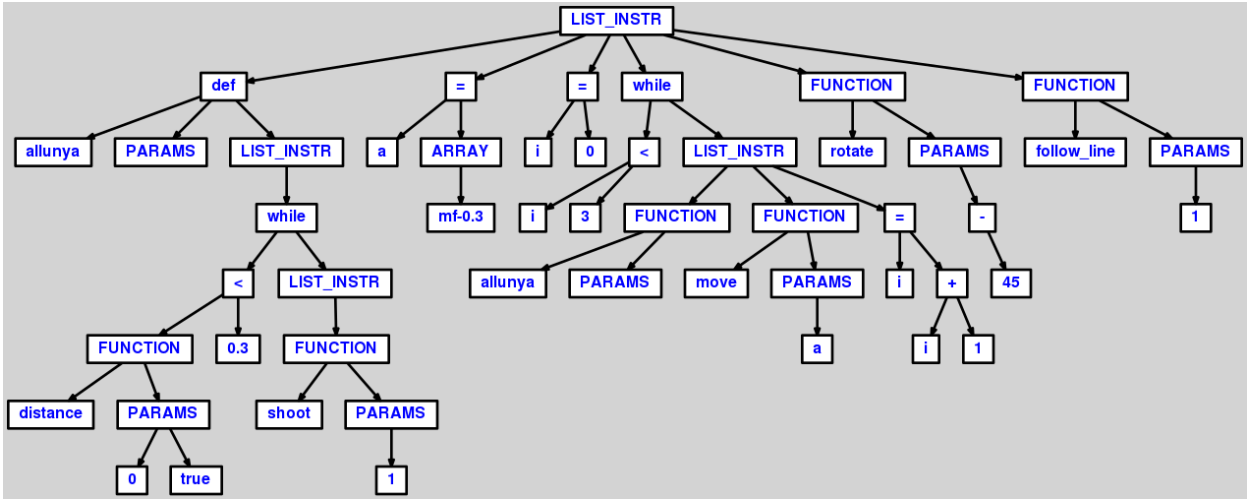

Figura 4: Arbre generat joc proves 4

2.5. Joc de proves 5

En aquest joc de proves bàsicament utilitzem el modul en la condició per veure si la variable x és un nombre parell, i posteriorment un seguit de operacions parentitzades. També es pot veure l'ús de l'operador "!="a *not equal* i operacions barrejant reals enters i booleans.

```
x = 3;
y = 0.177;
if(x%2 == 0) {
    x = (x-1*(y+0.4)) != y and y;
}
else {
    while(x > y) {
        move_front(1);
        x = x - 0.1;
```

```
    }
}
```

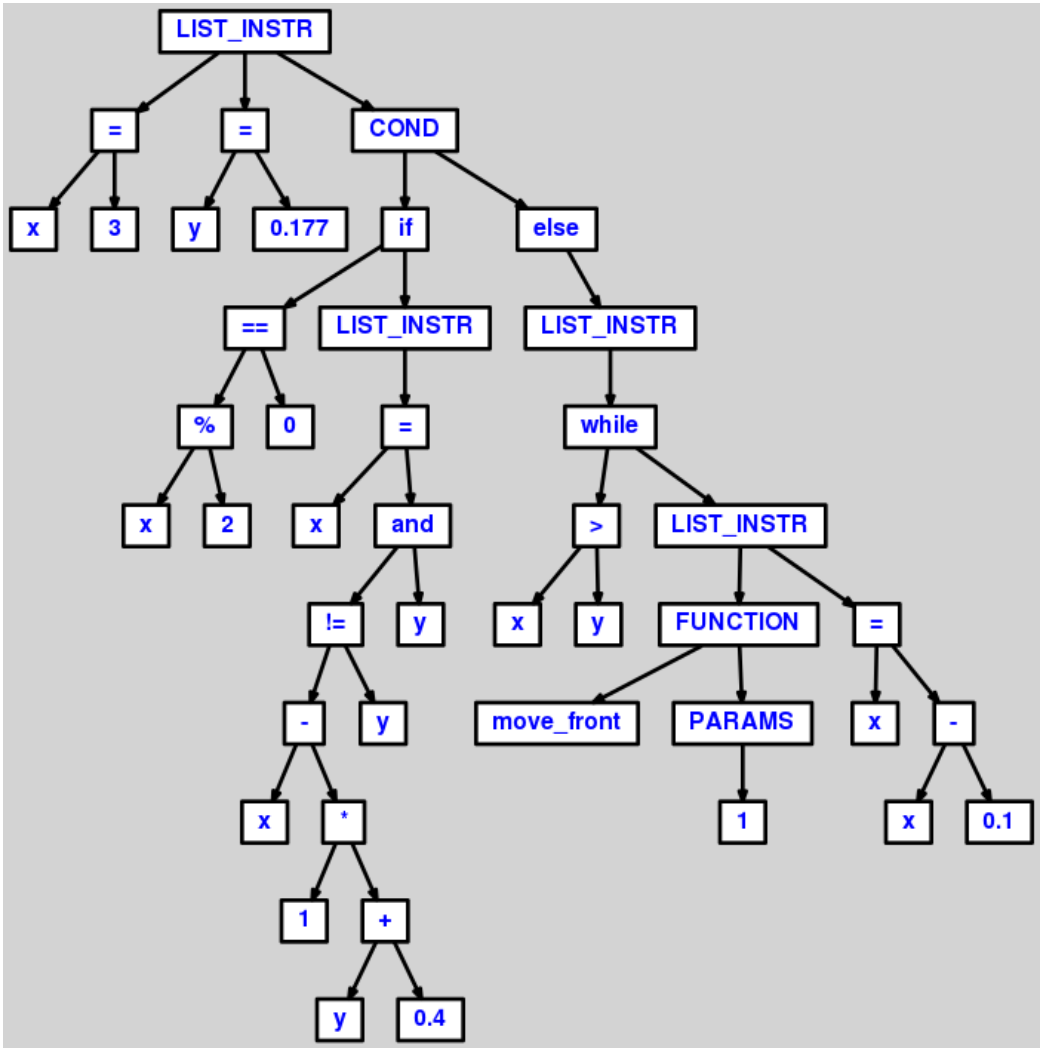

Figura 5: Arbre generat joc proves 5