MoviScript

Lenguaje especifico de dominio para programar móviles

Pablo Dobry

October 2, 2024

1 Descripción general

El lenguaje MoviScript es un lenguaje especifico de dominio (DSL) orientado a la programación de móviles como el recorrido de pequeños autos eléctricos programables y robots. Este lenguaje tiene la finalidad de facilitar la programación de la movilidad, ofreciendo una sintaxis intuitiva y funcionalidades específicas para abordar los desafíos asociados con estos dispositivos. Este DSL está diseñado para simplificar tareas comunes en la programación del desplazamiento, promoviendo un desarrollo eficiente.
Este lenguaje cuenta con comandos clasicos de un lenguaje imperativo, como la posiblidad de operar sobre expresiones matemáticas y booleanas, condicionales y bucles. Cuenta con instrucciones específicas de la programación de móviles como la posiblidad de generar generar una lista de puntos en función de una expresión matemática y una lista de puntos donde evaluarla.
Cuenta con instrucciones para el seguimiento de un camino (una lista de puntos), como así también la posiblidad de que, en caso de verse impedido el avance del móvil elabore un plan de contingencia para poder llegar al siguiente punto del camino.
La implementación de un lenguaje, el DSL se construye directamente embebido dentro de Haskell, utilizando sus funcionalidades como base. Luego, se emplea la biblioteca Parsec de Haskell para llevar a cabo el análisis sintáctico del código fuente del DSL.
Una vez completado el paso de parseo, se procede a la construcción de un Árbol de Sintaxis Abstracta (AST). Este árbol representa la estructura jerárquica del código fuente y sirve como una representación interna que facilita la manipulación y evaluación del código.
Finalmente, para la evaluación semántica del DSL, se recurre a un evaluador monádico. Las mónadas en Haskell ofrecen un mecanismo para modelar cómputos con efectos secundarios, lo que resulta esencial para la interpretación y ejecución de expresiones en nuestro lenguaje.

2 Manual de uso

El lenguaje Movi esta embebido en Haskell por lo que para compilar el lenguaje MoviScript se deberá tener instalado un compilador de Haskell. Con ghci para su uso se deberá ejecutar el siguiente comando en la carpeta en la que se encuentran los archivos Main.hs, Parser.hs y EvalMovi.hs:

```
ghci Main.hs miPrograma.movi
```

De esta forma se ejecuta el programa *miPrograma.movi*. La ejecución en esta primer versión de MoviScript producirá una salida en pantalla con las primitivas ejecutadas por el móvil y la traza con los puntos por los que pasó el móvil. Se debe tener instalado ucblogo para poder ejecutar el script en logo con el que se evalua el programa en esta version. Para hacerlo se puede ejecutar en una distribucion Debian o Ubuntu el siguiente comando:

```
apt-get install ucblogo
```

3 Expresiones flotantes

El lenguaje incorpora funciones trigonométricas (seno, coseno, tangente), logaritmo, redondeo hacia abajo (floor) y redondeo hacia arriba (ceil), ofreciendo así una amplia gama de herramientas matemáticas. La inclusión de funciones especializadas como la exponencial y el redondeo a un entero mediante la función 'round' brinda flexibilidad a los desarrolladores para expresar de manera eficiente y concisa operaciones matemáticas complejas. Esta sintaxis proporciona una base robusta para la manipulación de cálculos numéricos en el contexto de programación de móviles.
La sintaxis del lenguaje es simple y se describe a continuacion.

- *expf* : Negación de una expresión flotante (expf).

(*expf* **)** : Agrupación de una expresión flotante.

expf **+** *expf* : Suma de dos expresiones flotantes.

expf ***** *expf* Multiplicación de dos expresiones flotantes.

expf **-** *expf* Resta de dos expresiones flotantes.

expf **/** *expf* División de dos expresiones flotantes.

sen (*expf* **)** : Seno de una expresión flotante.

cos (*expf* **)** : Coseno de una expresión flotante.

tan (*expf* **)** ' expf '): Tangente de una expresión flotante.

log (*expf* **)** : Logaritmo de base 10 una expresión flotante.

floor (*expf* **)** : Redondeo hacia abajo de una expresión flotante.

ceil (*expf* **)** : Redondeo hacia arriba de una expresión flotante.

exp (*expf , expf* **)** : Función exponencial con dos argumentos flotantes.

round (*expf , num* **)** Redondeo de una expresión flotante con una cantidad determinada (num) de decimales.

4 Expresiones booleanas

Las expresiones booleanas se pueden relacionar con los siguientes operadores.

true y **false**: Representan valores booleanos directos.

expf < *expf* y *expf* > *expf*: Permite comparaciones relacionales entre expresiones flotantes.

boolexp & *boolexp*: Operador lógico AND para conjunción de expresiones booleanas.

boolexp | *boolexp*: Operador lógico OR para disyunción de expresiones booleanas.

∼ *boolexp*: Operador unario NOT para negación de una expresión booleana.

(*boolexp*): Permite agrupar expresiones booleanas para establecer el orden de evaluación.

5 Comandos básicos y de control de flujo

En este lenguaje, se emplean comandos fundamentales para la manipulación eficiente de variables y el control del flujo de ejecución típicos en cualquier lenguaje imperativo.

varf **:=** *expf*

La asignacion de expresión flotante (expf) a una variable de tipo flotante (varf) se hará con el operador :=. En MoviScript sólo se permitirá el almacenamiento de valores flotantes en las variables. En versiones futuras se podrán incorporar variables de punto, de listas de expresiones flotantes y de listas de punto.

comm **;** *comm* : La secuenciación de comandos se realiza con el caracter ';'.

if (*boolexp***) then** *comm* **else** *comm* **end**

Estructura condicional clasica if-then-else que ejecuta un bloque de comandos u otro según la evaluación de una expresión booleana.

while (*boolexp***) do** *comm* **end**

Establece un bucle que repite la ejecución de un bloque de comandos mientras una condición booleana se mantenga verdadera.

6 Comandos sobre puntos y listas de puntos

En MoviScript, las instrucciones relacionadas con puntos proporcionan herramientas fundamentales para la manipulación espacial y el análisis geométrico. Estas instrucciones permiten la opreaciones de puntos en un plano bidimensional como la medición de la distancia entre el origen y un punto específico, así como la determinación del ángulo que dicho punto forma con el eje x. La capacidad de calcular la distancia y el ángulo facilita el desarrollo de algoritmos espaciales avanzados, permitiendo a los programadores abordar tareas relacionadas con la ubicación y la orientación en el contexto específico de dispositivos móviles.
La representación de un punto en nuestro lenguaje sera un par ordenado de expresiones flotantes.

$$point = (expf, expf)$$

Sobre un punto se pueden realiar los siguientes comandos.

dist(*point*). Establece la distancia entre un punto (point) y el origen de coordinadas

rangle(*point*)

El comando *rangle* establece el angulo (en radianes, de 0 a 2π) determinado entre la recta que pasa por el punto (point) y el eje x.

gangle(*point*)

El comando *gangle* establece el angulo (en grados, de 0 a 360°) determinado entre la recta que pasa por el punto (point) y el eje x.

path($expf, var, [expf]$)

Con el comando *path* podremos generar listas de puntos (x_i, y_i) en función de una expresión matemática (f), una variable (x) y una lista de expresiones flotantes donde evaluaremos nuestra expresión matemática (x_i). El comando toma tres argumentos:

- $expf$: Es una expresión matemática (f) que generara los valores y_i.
- var : Es la variable (x) en que se evaluará en la expresión matemática $expf$.
- $[expf]$: Es la lista de valores x_i en la que se evaluará la función f.

7 Comandos de movimientos

En esta sección describiremos las funciones que brindan la capacidad del dispositivo para moverse de manera inteligente y eficiente, ofreciendo a los programadores las herramientas necesarias para la creación de algoritmos de movilidad. Por un lado tenemos herramientas de navegación básica:

fd($dist, time$)

Con el comando Fd se indica al móvil el avance lineal de una distancia($dist$, de tipo expresión flotante) en durante en un tiempo determinado($time$, de tipo expresión flotante).

bk($dist, time$)

Con el comando Bk se indica al móvil el retroceso lineal de una distancia($dist$, de tipo expresión flotante) en durante en un tiempo determinado($time$, de tipo expresión flotante).

turn($ang, time$)

Con el comando $Turn$ se indica al móvil el giro de un angulo en grados(ang, de tipo expresión flotante) en durante en un tiempo determinado($time$, de tipo expresión flotante).

El lenguaje, también cuenta con instrucciones complejas que permiten elaboración de algoritmos avanzados:

lookat($point, vel$)

Con el comando *LookAt* se indica al móvil el giro en dirección al punto (*point*). La coordenada del punto será en caracter relativo al propio móvil tomando como origen de coordenadas el móvil

y el sentido positivo del eje x será la dirección en la que esta apuntando el móvil al momento de ejecutar la instrucción. La velocidad de giro será la indicada como *vel*.

$$\textbf{goline}(point, vel - lineal, vel - angular)$$

Con el comando *GoLine* se indica al móvil el giro en dirección al punto (*point*) y luego su desplazamiento al punto mencionado. La coordenada del punto será en caracter relativo al propio móvil tomando como origen de coordenadas el móvil y el sentido positivo del eje x será la dirección en la que esta apuntando el móvil al momento de ejecutar la instrucción. La velocidad de giro será la indicada como $vel - angular$ y la velocidad de desplazamiento será la indicada como $vel - lienal$.

$$\textbf{follow}(listpoint, vel - lineal, vel - angular)$$

Con el comando $follow$ se indica que el móvil haga un recorrido determinado por la lista de puntos ($listpoint = [(x_1, y_1), (x_2, y_2)...(x_n, y_n)]$).
La velocidad de giro será la indicada como $vel - angular$ y la velocidad de desplazamiento será la indicada como $vel - lienal$.

$$\textbf{followsmart}(listpoint - allow, listpoint - contingency - allow, vel - lineal, vel - angular)$$

De igual manera que el comando $follow$, el comando $followsmart$ indica que el móvil haga un recorrido determinado por la lista de puntos seguido de las posiciones en las que estará habilitado el móvil para circular. La lista de puntos-contingency será la lista de puntos ($listpoint - allow = def\_obstacles[(x_1, y_1), (x_2, y_2)...(x_n, y_n)], [x_1...x_i...x_n]$) donde x_i indica la lista de puntos en los que el móvil esta obstaculizado por el medio para ciruclar.
En este caso, el comando $followsmart$ incorpora información del medio. Lo hace mediante la lista de puntos que en este caso será una lista de índices en los que los puntos obstaculizados por el medio.
Si el móvil descure que su camino está impedido por algún obstaculo el lenguaje MoviScript tendrá un camino de contingencia (relativo al último punto de la lista *listpoint* que logró acceder sin obstaculos). Nótese que esta contingencia es relativo al último punto y se utilizará como contingencia de forma recursiva, de forma de que si hay obstaculos abordando el camino de contingencia el móvil también utilizará el camino de contingencia.

8 Semántica de los comandos

Lookat

$$\frac{p \downarrow (p_x, p_y)}{Lookat(p, v) \rightarrow Turn(\frac{180}{\pi} * atan^2(p_x, p_y), v)}$$

Goline

$$\frac{p \downarrow dist}{Goline(p, v_1, v_2) \rightarrow Seq(Lookat(p, v_1), Fd(\frac{dist}{v_2}, v_2)}$$

TurnAbs

$$\frac{}{TurnAbs(ang, vel) \rightarrow Turn\left(\left(\frac{ang - angAct}{v}\right), vel\right)}$$

GolineAbs

$$\frac{p \downarrow (p_x, p_y), p \downarrow dist}{Seq(TurnAbs\left(\frac{180}{\pi} \cdot atan2(p_x, p_y)\right)\ \ v_1, Fd(\frac{dist}{v_2}), v_2)}$$

Follow

$$\frac{}{Follow([], v_1, v_2) \rightarrow Skip} \quad (1)$$

$$\frac{}{Follow([p], v_1, v_2) \rightarrow GolineAbs(p, v_1, v_2)} \quad (2)$$

$$\frac{p \downarrow (p_x, p_y), q \downarrow (q_x, q_y)}{Follow(p : q : ps, v_1, v_2) \rightarrow Seq(GolineAbs(p, v_1, v_2), Follow(((q_x - p_x, q_y - p_y) : ps), v_1, v_2))} \quad (3)$$

Followsmart

$$\frac{}{FollowSmart([], contingency, v_1, v_2) \rightarrow Skip} \quad (1)$$

$$\frac{}{FollowSmart([p], contingency, v_1, v_2) \rightarrow GolineAbs(p, v_1, v_2)} \quad (2)$$

$$\frac{}{FollowSmart([], contingency, v_1, v_2) \rightarrow Skip} \quad (3)$$

$$\frac{}{FollowSmart([p], contingency, v_1, v_2) \rightarrow GolineAbs(p, v_1, v_2)} \quad (4)$$

$$\frac{}{\begin{array}{r}FollowSmart(((p, True) : (q, b) : xs), contingency, v_1, v_2) \rightarrow Seq(GolineAbs(p, v_1, v_2), \\ FollowSmart(((q_x - p_x, q_y - p_y), b) : xs), contingency, v_1, v_2)\end{array}} \quad (5)$$

$$\frac{p \downarrow (p_x, p_y), q \downarrow (q_x, q_y)}{\begin{array}{r}FollowSmart(((p, False) : (q, b) : xs), [], v1, v2) \rightarrow \\ FollowSmart((((q_x - p_x, q_y - p_y), b) : xs), [], v_1, v_2)\end{array}} \quad (6)$$

$$\frac{p \downarrow (p_x, p_y), c \downarrow (c_x, c_y)}{\begin{array}{r}FollowSmart((((p, False) : xs), ((c, b) : cs), v_1, v_2)) \rightarrow \\ FollowSmart((((c_x + p_x, c_y + p_y), b) : cs +\!\!+ xs), ((c, b) : cs), v_1, v_2)\end{array}} \quad (7)$$

9 Posibles próximas versiones

En futuras versiones, MoviScript podrá tomar información del medio con un sensor frontal del mḿovil. También se podrá compilar MoviScript a código NQC[1] y de esta forma podrá ser ejecutado en un móvil construido en $LEGOMINDSTORMS$.

[1]Not Quite C es un lenguaje simple que sintaxis similar a C utilizado para programar la ínea robótica de LEGO denominada Maindstorms. Para más información visitar la documentación: https://bricxcc.sourceforge.net/nqc/