Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 – Traductores e Interpretadores
Enero – Marzo 2020

Version: 12/01/2020

Proyecto Único
Interpretador de `Willy`*

El proyecto consiste en implementar el ambiente de programación `Willy`*. Este documento da una visión a grandes rasgos e informal del ambiente y el lenguaje `Willy`*. A medida que avance el proyecto, esta descripción pudiera sufrir modificaciones. `Willy`* está inspirado en el ambiente/lenguaje `Willy` definido por el Prof. Ernesto Hernández-Novich. Este documento sigue la estructura del documento similar que define `Willy`.

El ambiente consiste de un robot llamado Willy que se desenvuelve dentro de una cuadrícula finita en la cual existen paredes. Willy tiene sensores, brazos actuadores, una cesta para guardar objetos, y medios para desplazarse. El robot puede navegar libremente en la cuadrícula y puede recoger y dejar objetos en las celdas de la cuadrícula. Willy es controlado por un programa que contiene una o más definiciones del mundo, y la definición de los procedimentos y el programa principal.

El proyecto se desarrolla a lo largo del trimestre a través de 3 partes o entregas:

1. Análisis lexicográfico que consiste en reconocer la entrada y dividirla en sus componentes más básicos (tokens) que constituyen la entrada el análisis sintáctico.

2. Análisis sintáctico y construcción del árbol abstracto que consiste en obtener una representación abstracta del mundo, y los programas para Willy. En este proceso se detectan errores en la especificación del mundo y/o programa.

3. Análisis de contexto e interpretación que consiste en terminar de procesar la entrada y correr el programa en el mundo seleccionado.

Estructura de un Programa en `Willy`*

Un programa en `Willy`* consiste de instrucciones dos tipos distintos. El primer tipo de instrucciones definen uno o varios mundos. Un mundo define las condiciones iniciales sobre las cuales se ejecutan los programas de control. El segundo tipo de instrucciones definen un programa principal y, de ser necesario, procedimientos adicionales que pueden ser invocados desde el programa principal.

Definiendo un Mundo de `Willy`*

Un mundo se describe con un bloque de la forma

`begin-world` <identificador> <instr>* `end-world`

el cual contiene un identificador que le da nombre al mundo y cero o mas instrucciones separadas por ';' que definen el mundo. Las instrucciones que definen al mundo son las siguientes:

- `World` <columnas> <filas>

  Define las dimensiones de la cuadrícula. Todo mundo debe tener a lo sumo una instrucción de este tipo. Si el mundo no contiene dicha instrucciones, se asume que mundo tiene una fila y una columna. Las filas y columnas del mundo están enumeradas comenzando desde el índice 1.

- `Wall` <dirección> `from` <columna0> <fila0> `to` <columna1> <fila1>

  Define una pared a partir de la celda determinada por <fila0> y <columna0>, en la dirección estipulada, hasta la celda determinada por <fila1> y <columna1>. La dirección es una palabra en {`north`, `east`, `south`, `west`}. La dirección de la pared debe ser consistente con las coordenadas especificadas; e.g., para la dirección `north`, ambas columnas deben ser iguales, y <fila1> debe ser mayor o igual a <fila0>. Similarmente, las coordenadas deben ser consistentes con la dimensión de la cuadrícula.

- `Object-type` <identificador> `of color` <color>

  Define un tipo de objeto identificado por <identificador> y cuyo color es también especificado. El color puede ser `red`, `blue`, `magenta`, `cyan`, `green`, or `yellow`.

- `Place` <n> `of` <identificador> `at` <columna> <fila>

  Coloca n objetos de tipo <identificador> en la celda correspondiente a <columna> y <fila>. La cantidad debe ser un entero decimal positivo, el identificador debe referirse a un tipo de objeto, y las coordenadas deben ser consistentes con las dimensiones del mundo. Si existen dos o más instrucciones de este tipo para la misma celda y tipo de objeto, el resultado es colocar en dicha celda un número de objetos del tipo dado igual a la suma de las cantidades en todas esas instrucciones.

- `Place` <n> `of` <identificador> `in basket`

  Coloca n objetos de tipo <identificador> en la cesta del robot. La cantidad n debe ser un entero decimal positivo y el identificador debe referirse a un tipo de objeto. Si existen dos o más instrucciones de este tipo para el mismo <identificador>, el resultado es colocar en la cesta un número de objetos de tipo <identificador> igual a la suma de las cantidades en todas esas instrucciones.

- `Start at` <columna> <fila> `heading` <dirección>

  Define la posición y orientación inicial de `Willy`. La dirección es una en `north`, `east`, `south` o `west`. Sólo puede haber a lo sumo una instrucción de este tipo. Si dicha instrucción no se especifica, `Willy` estara inicialmente localizado en la primera fila y primera columna, y orientado hacia el norte.

- `Basket of capacity` <n>

  Indica que la cesta de `Willy` tiene capacidad para n objetos. Es un error tener dos instrucciones de este tipo. Si no se especifica una capacidad para la cesta, la capacidad por defecto es de 1 objeto.

- `Boolean` <identificador> `with initial value` <valor>

  Define un nuevo booleano de nombre <identificador> cuyo valor inicial es `true` o `false`. Es un error definir el mismo booleano mas de una vez.

- `Goal` <identificador> `is` <goal-test>

  Define una condición objetivo de nombre identificador expresada por <goal-test>. Es un error definir dos condiciones objetivo con el mismo nombre.

- `Final goal is` <final-goal>

  Define la condición objetivo final del mundo expresada por <final-goal>.

- Las condiciones <goal-test> pueden ser:

  - `willy is at` <columna> <fila>
  - <n> <identificador> `objects in Basket`
  - <n> <identificador> `objects at` <columna> <fila>

- La condición <final-goal> puede expresarse utilizando las siguientes expresiones:

- <identificador> de una condición objetivo previamente definida o identificador de booleano.
- <final-goal0> `and` <final-goal1>
- <final-goal0> `or` <final-goal1>
- `not` <final-goal>
- (<final-goal>)

Definiendo un Programa `Willy`*

Lo programas para Willy constan de un bloque principal y posiblemente definicioness de procedimientos adicionales. Dentro del bloque principal y las definiciones de procedimientos solamente pueden utilizarse las instrucciones que operan sobre Willy; es decir, las instrucciones que definen el mundo no pueden utilizarse dentro del programa principal o procedimientos auxiliares. Las instrucciones de programa permitidas son:

- `begin-work on` <identificador> <instrucción>* `end-work`

  Define un programa principal para el mundo especificado por el identificador. El programa consta de cero o más instrucciones separadas por ';'. Las instrucciones en el bloque se ejecutan de forma secuencial sobre el mundo. La ejecución del programa termina cuando la última instrucción del bloque finaliza, o cuando se ejecuta la instrucción `terminate`. Un programa es *existoso* si y sólo si la ejecución termina y la condición objetivo final se cumple en la configuración del mundo que resulta después de la ejecución.

- `if` <test> `then` <instrucción>

  Define una instrucción condicional mediada por <test>. Si <test> evalúa a cierto, se ejecuta <instrucción>.

- `if` <test> `then` <instrucción> `else` <instrucción>

  Similar a la anterior pero con una instrucción adicional que es ejecutada cuando <test> evalúa a falso.

- `repeat` <n> `times` <instrucción>

  Define una iteración acotada que ejecuta <instrucción> n veces. El parámetro n es un entero decimal no negativo.

- `while` <test> `do` <instrucción>

  Define una iteración abierta que ejecuta <instrucción> mientras la condición test evalue a cierto.

- `begin` <instrucción>* `end`

  Permite definir una instrucción compuesta que consiste de cero o mas instrucciones, separadas por ';'.

- `define` <identificador> `as` <instrucción>

  Permite asociar una instrucción con un nuevo nombre que luego puede ser utilizado como cualquier otra instrucción. Es un error definir dos veces la misma instrucción, así como definir instrucciones primitivas del lenguaje.

- Instrucciones primitivas que instruyen a Willy a realizar una tarea específica:
  - `move` para moverse un paso en la dirección actual.
  - `turn-left` para girar a la izquierda y así cambiar de orientación.
  - `turn-right` para girar a la derecha y así cambiar de orientación.
  - `pick` <identificador> para recoger un objeto de tipo <identificador> en la celda actual. Es un error de ejecución tratar de rocoger un objeto de tipo <identificador> cuando dichos objetos no existen en la celda actual.

  - `drop` <identificador> para dejar un objeto de tipo <identificador> en la celda actual. Es un error de ejecución tratar de dejar un objeto de tipo <identificador> cuando el robot no tiene en su cesta un objeto de dicho tipo.
  - `set` <identificador> para asignar el valor `true` al booleano <identificador>.
  - `set` <identificador> `to` <value> para asignar el valor <value> al booleano <identificador>.
  - `clear` <identificador> para asignar el valor `false` al booleano <identificador>.
  - `flip` <identificador> para complementar el valor del booleano <identificador>.
  - `terminate` para terminar la ejecución del programa.

- Expresiones booleanas se construyen a partir de los booleanos primitivos o definidos en el mundo, y las expresiones que se puedan construir con los conectores booleanos estándar:
  - <identificador> de booleano que puede ser uno de los definidos en el mundo, o un booleano primitivo en: `front-clear`, `left-clear`, `right-clear`, `looking-north`, `looking-east`, `looking-south`, `looking-west`.
  - `found(`<identificador>`)` que evalúa a cierto si y sólo si la celda actual contiene un objeto de tipo <identificador>.
  - `carrying(`<identificador>`)` que evalúa a cierto si y sólo si la cesta del robot contiene un objeto de tipo <identificador>.
  - <test0> `and` <test1> permite construir una expresión booleana.
  - <test0> `or` <test1> permite construir una expresión booleana.
  - `not` <test> permite construir una expresión booleana.
  - (<test>) permite construir una expresión booleana.