Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 – Traductores e Interpretadores
Septiembre – Diciembre 2015

Proyecto
Intérprete de `BOT`

Es el año 3043 y la Tierra ha quedado virtualmente inhabitable. Sólo algunas pequeñas ciudades restan, resguardadas bajo tierra en las regiones más apartadas del mundo. Hace décadas que ningún humano toca la superficie del planeta, sin embargo en ésta se encuentra nutrientes y combustible que son necesarios para prolongar la supervivencia de la especie. Por esta razón, la humanidad desarrolló robots especializados que podrían recorrer la superficie y realizar cualquier trabajo que fuese necesario.

Cada robot tiene la capacidad para comunicarse eficientemente con la base humana y para moverse en cualquier dirección, sin embargo la programación de los mismos no puede cambiar una vez dejan la base. Por tanto, sus programas deben ser lo suficientemente sofisticados como para adaptarse a las diferentes situaciones que pudieran encontrar. Para trabajar en este problema se crea la `USB` *(Unidad Sintáctica Biónica)*, encargada de diseñar e implementar un lenguaje de programación imperativo sencillo, que permitiese a los robots una programación completa y adecuada a cada necesidad. Además, debe proveer un controlador principal que organice a los robots. Para esto, representarán a la superficie como una matriz de dos dimensiones (cada una de tamaño infinito), dividida en celdas de $1m^2$. Cada celda en esta matriz está representada por dos números enteros (no necesariamente positivos).

Después de incontables reuniones y seminarios, los miembros de alto rango de la `USB` finalmente presentan su propuesta de lenguaje, al cual llamaron `BOT`. Sin embargo, necesitan ayuda para su implementación y puesta a prueba. Para trabajar en esto contactan a la comunidad más brillante que resta en cualquiera de las últimas ciudades, los llamados *"estudiantes de traductores"*. Su talento legendario para programar, bien podría ser la salvación de la humanidad. Por esto se les encarga implementar un intérprete para `BOT`, pero dada la inminente escasez de recursos, sólo tendrán 12 semanas para lograrlo.

El diseño original del lenguaje obvia muchos elementos usuales de los antiguos lenguajes de programación, como el manejo de estructuras de datos compuestas y procedimientos. Sin embargo, futuras versiones del lenguaje para robots más avanzados pudieran ser posibles. Siendo así y previendo tales cambios, se les pedirá durante el transcurso del proyecto analizar cómo podrían incluirse algunas de estas características adicionales en su implementación.

A continuación se describe entonces el lenguaje `BOT`, para el cual Ud. creará un intérprete. El desarrollo se realizará en 4 etapas: (I) Análisis lexicográfico; (II) Análisis sintáctico y construcción del árbol sintáctico abstracto; (III) Análisis de contexto e (IV) Intérprete final del lenguaje.

1 Estructura de un Programa en `BOT`

[`create` ⟨*Lista de Declaraciones*⟩] `execute`
⟨*Instrucción de Controlador*⟩
`end`

donde las palabras claves `execute` y `end` indican el principio y el fin del controlador principalmente respectivamente. Note que los corchetes "[" y "]" no son parte del programa, sino que son utilizados para indicar que lo que encierran es opcional, que en este caso corresponde a la creación de los robots, precedida por la palabra clave `create`. Por otra parte, los signos "⟨" y "⟩" son utilizados para indicar que lo que encierran es un componente del programa cuya sintaxis será explicada más adelante.

La ⟨*Lista de Declaraciones*⟩ es una lista no vacía que contiene las declaraciones de robots, sus tipos respectivos y su programación. Estas definiciones serán utilizadas luego en la ⟨*Instrucción de Controlador*⟩ que representa al controlador principal. La sintaxis de las instrucciones de controlador (i.e. de ⟨*Instrucción de Controlador*⟩) se describe en la Sección 1..

Cada definición de robots tiene la forma siguiente:

⟨*Tipo*⟩ `bot` ⟨*Lista de Identificadores*⟩
[⟨*Lista de Comportamientos*⟩]
`end`

La ⟨*Lista de Identificadores*⟩ es una lista no vacía de identificadores (nombres) de robots. Los identificadores estarán separadas por comas ("`,`") en la lista en cuestión. Cada identificador es el nombre de un robot y estará formado por una letra seguida de cualquier cantidad de letras y dígitos decimales. No se aceptará como identificador de variable secuencias alfabéticas que correspondan a palabras claves utilizadas en la sintaxis de `BOT` (por ejemplo: `create`, `execute`, `if`, etc.). En `BOT` se hace distinción entre mayúsculas y minúsculas, por lo que los identificadores `abcde` y `aBcdE` son diferentes; por tanto, identificadores como `Execute` no serían palabras clave.

Cada robot tiene un tipo asociado, especificado en ⟨*Tipo*⟩ (que comparten todos los robots presentes en ⟨*Lista de Identificadores*⟩). Cada robot puede almacenar un valor del tipo que tiene asociado y el controlador principal podrá utilizarlo en cualquier momento. Todos los robots son activados en la posición $(0, 0)$, donde está la base humana.

El lenguaje manejará solamente valores de tipo entero (representados por la palabra clave `int`), booleano (representados por la palabra clave `bool`) y caracteres (representados por la palabra clase `char`). Los robots toman valores solamente a través de las instrucciones de almacenamiento (`store`) o de lectura (`recieve`), e inicialmente no tienen valor alguno. Se considerará un error de ejecución el tratar de usar el valor de un robot que no haya almacenado nada aún.

La ⟨*Lista de Comportamientos*⟩ es una lista, posiblemente vacía, que contiene los comportamientos del robot en cuestión. Cada comportamiento tiene la siguiente forma:

```
on ⟨Condición⟩ :
    ⟨Instrucción de Robot⟩
end
```

La sintaxis de las instrucciones para robots (i.e. de ⟨*Instrucción de Robot*⟩) se describe en la Sección 2.. La sintaxis de las condiciones (i.e. de ⟨*Condición*⟩) se describe en la Sección 3..

Como adelanto, mostramos a continuación un ejemplo de programa escrito en `BOT` que imprime "Hello BOT!" (sin las comillas):

```
create
    char bot b
        on activation:
            store 'H'. send.
            store 'e'. send.
            store 'l'. send.
            store 'l'. send.
            store 'o'. send.
            store ' '. send.
            store 'B'. send.
            store 'O'. send.
            store 'T'. send.
            store '!'. send.
        end
    end

execute
    activate b.
end
```

Las partes involucradas en este ejemplo serán explicadas en las secciones siguientes.

2 Instrucciones

A continuación se describen las instrucciones disponibles en `BOT`, tanto en el controlador como en los robots. Todas las instrucciones deben ser terminadas por un punto ("`.`").

1. Instrucciones de Controlador

Las instrucciones permitidas en el controlador principal de un programa en `BOT` son:

Activación: Una activación "`activate` ⟨*Lista de Identificadores*⟩" tiene el efecto de activar todos los robots cuyos nombre aparezcan en la ⟨*Lista de Identificadores*⟩. Si los robots tienen

definido un comportamiento "`on activation`", el código correspondiente es ejecutado. Los robots son activados en orden de aparición en la ⟨*Lista de Identificadores*⟩ (de izquierda a derecha). Los robots a activar deben haber sido creados y no estar ya activos; en caso contrario se dará un mensaje de error.

Avance: Un avance "`advance` ⟨*Lista de Identificadores*⟩" tiene el efecto de ejecutar un paso de todos los robots cuyos nombre aparezcan en la ⟨*Lista de Identificadores*⟩. Para cada robot, primeramente se revisa la lista de comportamientos de mismo y se toma el primer comportamiento cuya condición sea satisfecha y se ejecuta la instrucción asociada (no realiza acción alguna si ninguna condición es satisfecha). Los robots son avanzados en orden de aparición en la ⟨*Lista de Identificadores*⟩ (de izquierda a derecha). Los robots a avanzar deben haber sido creados y activados; en caso contrario se dará un mensaje de error.

Desactivación: Una desactivación "`deactivate` ⟨*Lista de Identificadores*⟩" tiene el efecto de desactivar todos los robots cuyos nombre aparezcan en la ⟨*Lista de Identificadores*⟩. Si los robots tienen definido un comportamiento "`on deactivation`", el código correspondiente es ejecutado. Los robots son desactivados en orden de aparición en la ⟨*Lista de Identificadores*⟩ (de izquierda a derecha). Los robots a desactivar deben haber sido creados y activados; en caso contrario se dará un mensaje de error.

Secuenciación: La composición secuencial de las instrucciones ⟨*Instr0*⟩ e ⟨*Instr1*⟩ es la instrucción compuesta "⟨*Instr0*⟩ ⟨*Instr1*⟩". Ésta corresponde a ejecutar la instrucción ⟨*Instr0*⟩ y, a continuación, la instrucción ⟨*Instr1*⟩.

Note que "⟨*Instr0*⟩ ⟨*Instr1*⟩" es *una* instrucción compuesta. La secuenciación permite combinar varias instrucciones en una sola que puede entonces ser, por ejemplo, *la* instrucción del cuerpo del controlador principal o de un comportamiento para algún robot.

Condicional: Las instrucciones condicionales de `BOT` son de la forma

"`if` ⟨*Bool*⟩ `:` ⟨*Instr0*⟩ [`else :` ⟨*Instr1*⟩] `end`"

donde de nuevo es importante notar que hemos usado corchetes, "[" y "]", para indicar que lo que éstos encierran es opcional (la rama "`else`"). ⟨*Bool*⟩ es una expresión booleana (ver Sección 4.), e ⟨*Instr0*⟩ e ⟨*Instr1*⟩ son instrucciones cualesquiera.

La semántica para esta instrucción es la convencional: Se evalúa la expresión booleana ⟨*Bool*⟩; si ésta es verdadera, se ejecuta ⟨*Instr0*⟩ y, en caso contrario, se ejecuta ⟨*Instr1*⟩ (si la rama "`else`" está presente). En caso de que la expresión booleana sea falsa y la rama del "`else`" no se encuentre presente, la instrucción no tendrá efecto alguno. Es decir, no se ejecutará ninguna acción.

Iteración Indeterminada: Las instrucciones de iteración indeterminadas (esto es, con condiciones generales de salida) de `BOT` son de la forma

"`while` ⟨*Bool*⟩ `:` ⟨*Instr*⟩ `end`"

con ⟨*Bool*⟩ una expresión booleana e ⟨*Instr*⟩ una instrucción cualquiera.

La semántica para esta instrucción es la convencional: Se evalúa la expresión ⟨*Bool*⟩; si ésta es verdadera, se ejecuta el cuerpo ⟨*Instr*⟩ y se vuelve al inicio de la ejecución (preguntando nuevamente por la condición anterior) y, en caso contrario, se abandona la ejecución de la iteración.

Visto de otra forma, sea `I` una instrucción de iteración indeterminada, con forma: `while B do I0 end`, esta instrucción es equivalente a la secuenciación `I0 ; I` si `B` evalúa en cierto y a la instrucción vacía (una instrucción que no tiene ningún efecto) si `B` evalúa en falso.

Como ejemplo, el siguiente programa imprime los primeros n números de Fibonnaci:

```
create
    int bot n
        on activation:
            recieve.
        end
        on default:
            store me - 1.
        end
    end
    int bot f
        on activation:
            store 0.
            send.
            drop 0.
            store 1.
        end
        on default:
            send.
            collect as x.
            drop me.
            store me + x.
        end
    end

execute
    activate n, f.
    while n > 0:
        advance n, f.
    end
end
```

Incorporación de alcance: Una instrucción de incorporación de alcance en `BOT` tiene la siguiente estructura:

[`create` ⟨*Lista de Declaraciones*⟩] `execute`
⟨*Instrucción de Controlador*⟩
`end`

Así es, exactamente la misma estructura que la de un programa. Esta instrucción incorpora las nuevas declaraciones de variables (de existir) y las hace visibles/usables únicamente en la ⟨*Instrucción de Controlador*⟩.

2. Instrucciones de Robot

Almacenamiento: Una instrucción de almacenamiento "`store` ⟨*Expresión*⟩" tiene el efecto de evaluar la expresión en ⟨*Expresión*⟩ y almacenar el valor resultante como *valor asociado del robot*. Todas las variables presentes en ⟨*Expresión*⟩ deben haber sido previamente declaradas e inicializadas, de lo contrario se deberá reportar un error. Además, el tipo del valor resultante debe ser igual al tipo asociado al robot, de lo contrario se deberá reportar un error. La sintaxis para expresiones se detallará en la sección 3.

Colección: Una instrucción de colección "`collect` [`as` ⟨*Identificador*⟩]" tiene el efecto de tomar el valor que se encuentra almacenado en la posición de la matriz en la que el robot se encuentre. De estar presente el `as`, se declarará una variable cuyo nombre será ⟨*Identificador*⟩ y el valor recopilado será almacenado en dicha variable. Tal variable no debe haber sido declarada antes en el mismo alcance, de lo contrario se deberá reportar un error. Si no está presente el `as`, se almacenará el valor como *valor asociado al robot* y su tipo debe ser igual al tipo asociado al robot. Si la posición de la matriz en donde se encuentra el robot está vacía o si el tipo del robot no es igual al del valor almacenado en la matriz, se deberá reportar un error.

Soltado: Una instrucción de soltado "`drop` ⟨*Expresión*⟩" tiene el efecto de evaluar la expresión en ⟨*Expresión*⟩ y almacenar el valor resultante en la matriz, en la posición actual del robot. Todas las variables presentes en ⟨*Expresión*⟩ deben haber sido previamente declaradas e inicializadas, de lo contrario se deberá reportar un error. El tipo del valor resultante puede ser igual o diferente al tipo asociado del robot.

Movimiento: Las instrucciones de la forma "⟨*Dirección*⟩ [⟨*Expresión*⟩]" moverán la posición del robot en la dirección propuesta en ⟨*Dirección*⟩. De estar presente ⟨*Expresión*⟩, la expresión se evaluará y se moverá al robot tantas unidades como el valor resulte. Dicha expresión debe ser de tipo entero y ser *no negativa*. Además, todas las variables presentes en ⟨*Expresión*⟩ deben haber sido previamente declaradas e inicializadas, de lo contrario se deberá reportar un error.

La dirección en ⟨*Dirección*⟩ puede ser una de entre "`left`" (izquierda), "`right`" (derecha), "`up`" (arriba) y "`down`" (abajo).

Entrada y Salida: `BOT` cuenta con instrucciones que le permite a los robots interactuar con la base humana a través de la entrada/salida estándar del sistema de operación (indistinto para muchos sistemas de operación conocidos).

Para leer un valor de la entrada las instrucciones serán de la forma "`read` [`as` ⟨*Identificador*⟩]" y tendrán en el efecto de pedir un valor al usuario. De estar presente el `as`, se declarará una variable cuyo nombre será ⟨*Identificador*⟩ y el valor recopilado será almacenado en dicha variable. Tal variable no debe haber sido declarada antes en el mismo alcance, de lo contrario se deberá reportar un error. Si no está presente el `as`, se almacenará el valor como *valor asociado al robot* y su tipo debe ser igual al tipo asociado al robot.

Para escribir en la salida las instrucciones serán de la forma "send" y tendrán el efecto de imprimir el valor asociado al robot a la salida estándar.

Secuenciación: De manera análoga a las instrucciones de controlado, la composición secuencial de las instrucciones ⟨*Instr0*⟩ e ⟨*Instr1*⟩ es la instrucción compuesta "⟨*Instr0*⟩ ⟨*Instr1*⟩". Ésta corresponde a ejecutar la instrucción ⟨*Instr0*⟩ y, a continuación, la instrucción ⟨*Instr1*⟩.

3. Condiciones

Las condiciones disponibles para los robots, que funcionan como guardias de cada comportamiento son las siguientes:

- `activation`: Sólo se ejecuta el comportamiento asociado cuando el robot en cuestión está siendo activado.
- `deactivation`: Sólo se ejecuta el comportamiento asociado cuando el robot en cuestión está siendo desactivado.
- ⟨*Expresión*⟩: La expresión en ⟨*Expresión*⟩ debe evaluar a un valor booleano. Sólo se ejecuta el comportamiento asociado cuando el robot en cuestión está siendo avanzado y la expresión evalúa en `true`.
- `default`: Sólo se ejecuta el comportamiento asociado cuando el robot en cuestión está siendo avanzado y ninguna de las expresiones anteriores es cumplida. Sólo puede haber un comportamiento `default` por robot, y de existir debe estar después de todos los demás comportamientos (con excepción de `activation` y `deactivation`).

3 Expresiones

A continuación se listan los tipos de expresiones con los que cuenta `BOT`.

1. Literales

En `BOT` existen tres tipos básicos: los enteros, los booleanos y los caracteres. Los literales enteros serán todos los números naturales, precedidos con una cantidad arbitraria de ceros y posiblemente un símbolo (`-`) para expresar que el número es negativo. Los literales booleanos serán `true` y `false`. Los literales para caracteres serán el caracter encerrado en comillas simples. Además, se deben soportar los siguientes caracteres especiales:

- `'\n'`: Salto de linea.
- `'\t'`: Tabulador horizontal.
- `'\''`: Comilla simple.

2. Variables

Las variables en `BOT` pueden llevar por nombre cualquier cadena, tal que empiece con un caracter alfabético (mayúscula o minúscula) y el resto esté formado por caracteres alfanuméricos (mayúsculas, minúsculas o dígitos) o el caracter de *underscore*.

Para las instrucciones de controlador, las variables visibles serán únicamente los nombres de los robots creados. Para las instrucciones de robot, las variables visibles serán únicamente aquellas creadas dentro de un comportamiento en específico, únicamente desde su declaración hasta el final la ejecución de tal comportamiento. Además, en las instrucciones de robot estará disponible la variable especial `me`, que tendrá siempre el *valor asociado al robot*.

3. Expresiones Aritméticas

Una expresión aritmética estará formada por números naturales, identificadores de variables, y operadores convencionales de aritmética entera. Los operadores a ser considerados serán suma (`+`), resta (`-` binario), multiplicación (`*`), división entera (`/`), resto de división entera o módulo (`%`), e inverso (`-` unario). Tal como se acostumbra, las expresiones serán construidas con notación infija para los operadores binarios, por ejemplo `1+2`, y con notación prefija para el operador unario, por ejemplo `-3`. La tabla de precedencia es también la convencional (donde los operadores más fuertes están hacia abajo):

`+`, `-` binario
`*`, `/`, `%`
`-` unario

y, por supuesto, se puede utilizar paréntesis para forzar un determinado orden de evaluación. Por tanto, evaluar `2+3/2` da como resultado `3`, mientras que evaluar `(2+3)/2` da `2`. Los operadores con igual precedencia se evalúan de izquierda a derecha. Por tanto, evaluar `60/2*3` resulta en `90`, mientras que evaluar `60/(2*3)` resulta en `10`.

Expresiones con variables serán evaluadas de acuerdo al valor que éstas tengan en ese momento, para lo cual se requiere que tales variables hayan sido declaradas y previamente inicializadas. Por ejemplo, la evaluación de `x+2` resulta en `5`, si `x` fue declarada y en su última asignación tomó valor `3`. Si `x` no fue declarada o es de un tipo no compatible para la operación, se da un mensaje de error; a este tipo de errores se les llama *estáticos*, pues pueden ser detectados antes de la ejecución del programa. Si `x` fue declarada pero no ha sido inicializada previamente, también debe darse un mensaje de error; este error sería *dinámico*, pues sólo puede ser detectado durante la ejecución del programa.

4. Expresiones Booleanas

Análogamente a las expresiones aritméticas, una expresión booleana estará formada por las constantes `true` y `false`, identificadores de variables, y operadores convencionales de lógica booleana. Los operadores a ser considerados serán conjunción (/\), disyunción (\/), y negación (∼). Tal como se acostumbra, las expresiones serán construidas con notación infija para los operadores binarios, por ejemplo `true /\ false`. Sin embargo, el operador unario (negación) será construido con notación prefija, por ejemplo ∼`true`. La tabla de precedencia es también la convencional (donde las operadores más fuertes están hacia abajo):

```
\/
/\
∼
```

y, por supuesto, se puede utilizar paréntesis para forzar un determinado orden de evaluación. Por tanto, evaluar `true \/ true /\ false` resulta en `true`, mientras que la evaluación de `(true \/ true) /\ false` resulta en `false`. Los operadores con igual precedencia se evalúan de izquierda a derecha. Sin embargo, note que en este caso, en realidad dicho orden es irrelevante.

Expresiones con variables serán evaluadas de acuerdo al valor que éstas tengan en ese momento, para lo cual se requiere que tales variables hayan sido declaradas y previamente inicializadas. Por ejemplo, la evaluación de `x \/ false` resulta en `true` si `x` fue declarada y en su última asignación tomó valor `true`. Si `x` no fue declarada o es de un tipo no compatible para la operación, se da un mensaje de error; a este tipo de errores se les llama *estáticos*, pues pueden ser detectados antes de la ejecución del programa. Si `x` fue declarada pero no ha sido inicializada previamente, también debe darse un mensaje de error; este error sería *dinámico*, pues sólo puede ser detectado durante la ejecución del programa.

Además `BOT` también contará con operadores relacionales que comparan expresiones entre sí. Éstas serán de la forma "⟨*Aritm*⟩ ⟨*Rel*⟩ ⟨*Aritm*⟩", donde ambas ⟨*Aritm*⟩ son expresiones aritméticas y ⟨*Rel*⟩ es un operador relacional. Los operadores relacionales a considerar son: menor (`<`), menor o igual (`<=`), mayor (`>`), mayor o igual (`>=`), igualdad (`=`) y desigualdad (`/=`). También será posible comparar expresiones booleanas bajo la forma "⟨*Bool*⟩ ⟨*Rel*⟩ ⟨*Bool*⟩", con ⟨*Rel*⟩ pudiendo ser únicamente igualdad (`=`) y desigualdad (`/=`).

4 Comentarios

En `BOT` es posible comentar secciones completas del programa, para que sean ignorados por el interpretador del lenguaje, encerrando dicho código entre los simbolos **"$-"** y **"-$"**. Estos comentarios no permiten anidamiento (comentarios dentro de comentarios) por lo que dentro de una sección comentada debe prohibirse el símbolo que cierra comentarios (**"-$"**). El símbolo que los abre (**"$-"**) puede reaparecer en el comentario, sin embargo no tendrá efecto alguno (será ignorado como el resto de la sección comentada). Además, es posible comentar una línea completa utilizando el símbolo `$$`.