Informe sobre Intérprete *She-HULK : Sheila's Havana University Language for Kompilers*

Sheila Roque Alemán

Introducción

She-HULK es un intérprete sencillo que implementa ciertas características del lenguaje de programación *HULK: Havana University Language for Kompilers*. *HULK* es un lenguaje de programación imperativo, funcional, estática y fuertemente tipado, creado y diseñado por la Facultad de Matemática y Computación de la Universidad de La Habana, *MatCom*.

She-HULK

El intérprete de *She-HULK* es una aplicación de consola, donde el usuario puede introducir una expresión de *HULK* , presionar [ENTER], e immediatamente se verá el resultado de evaluar expresión (si lo hubiere). Cada línea que comienza con '>' representa una entrada del usuario, e immediatamente después se imprime el resultado de evaluar esa expresión, si lo hubiere. Si el usuario presiona [ENTER] sin haber escrito una instrucción se tomará como una señal de que desea cerrar el programa

En este intérprete solo se aceptan expresiones de una línea. Para ello se han implementado diversas expresiones *inline* , como lo son la declaración de funciones de este tipo y ciertas expresiones como las *let-in* e *if-else* .

A pesar de esta restricción, el intérprete posee la capacidad y optimización necesarias para poder crear un programa completamente funcional y veloz habiéndose creado las funciones necesarias línea por línea.

Expresiones básicas

- Todas las instrucciones en *She-HULK* terminan en "*;*".
- La instrucción más simple en *She-HULK* que hace algo es:

  > print("Hello, World!");
  Hello, World!
- *She-HULK* además tiene expresiones y funciones aritméticas básicas:

  *> print((((1 + 2) ^ 3) * 4) / 5);*
  21.6
  *> print(sin(2 * PI) ^ 2 + cos(3 * PI / log(4, 64)));*
  -1
- *She-HULK* tiene tres tipos básicos: *String*, *Number*, y *Boolean*.

Funciones

- En *HULK* existen dos tipos de funciones, *inline* y regulares. En *She-HULK* solo se implementan las funciones *inline* . Tienen el formato siguiente:

  > function tan(x) => sin(x) / cos(x);

- Una vez definida una función, puede usarse en una expresión cualquiera:

  > print(tan(PI/2));
  ∞

- Todas las funciones declaradas anteriormente son visibles en cualquier expresión subsiguiente.

  * *Las funciones no pueden redefinirse.*

Variables

- En *She-HULK* es posible declarar variables usando la expresión *let-in* , que posee la siguiente sintaxis:

  > let x = PI/2 in print(tan(x));
  ∞

- Una expresión *let-in* consta de una o más declaraciones de variables, y un cuerpo, que puede ser cualquier expresión donde además se pueden utilizar las variables declaradas en el *"let"*.

  > let number = 42, text = "The meaning of life is" in print(text @ number);
  The meaning of life is 42
  > let number = 42 in (let text = "The meaning of life is" in (print(text @ number)));
  The meaning of life is 42

- Fuera de una expresión *let-in* las variables dejan de existir.

- El valor de retorno de una expresión *let-in* es el valor de retorno del cuerpo, por lo que es posible hacer:

  *> print(7 + (let x = 2 in x * x));*
  11

Condicionales

- Las condiciones en *She-HULK* se implementan con la expresión *if-else* , que recibe una expresión booleana entre paréntesis, y dos expresiones para el cuerpo del *if* y el *else* respectivamente.

  * *Siempre deben incluirse ambas partes*

  > let a = 42 in if (a % 2 == 0) print ("even") else print ("odd");
  even

- Como *if-else* es una expresión, se puede usar dentro de otra expresión

  > let a = 42 in print (if (a % 2 == 0) "even" else "odd");
  even

Funciones recursivas

- Una función recursiva en *She-HULK* es la siguiente:

  > function fib(n) => if (n > 2) fib(n-1) + fib(n-2) else 1;
  > fib(5);
  5
  > let x = 3 in fib(x+1);
  3
  > print(fib(6));
  8

Errores

- En *She-HULK* hay 3 tipos de errores que pueden ser detectados: *Léxico*, *Sintáctico* y *Semántico*.
- En caso de detectarse un error, el intérprete imprime una línea lo más informativa posible indicando el error.
- Al detectar un error, el intérprete detiene la compilación el código y lanza un error, sin causar que el programa caiga, dándole la oportunidad al usuario de corregir su error y continuando con su ejecución.
- En caso de ocurrir más de un error, solo se lanza el primero que aparece.

Errores Léxicos

Errores que se producen por la presencia de tokens inválidos o falta de comillas de cierre en los strings:

> let 14a = 5 in print(14a);
! LEXICAL ERROR: '14a' is not a valid token.
> print("Hello, World!);
! LEXICAL ERROR: Missing closing ' " ' in string expression.

Errores Sintácticos

Errores que se producen por expresiones mal formadas como paréntesis no balanceados o expresiones incompletas:

> let a = 5 in print(a;
! SYNTAX ERROR: Missing closing parenthesis after 'a'.
> let a = 5 inn print(a);
! SYNTAX ERROR: Missing 'in' keyword in 'let-in' expression.
> let a = in print(a);
! SYNTAX ERROR: Invalid expression after variable 'a' in 'let-in' expression.

Errores Semánticos

Errores que se producen por el uso incorrecto de los tipos y argumentos:

> let a = "Hello, World!" in print(a + 5);
! SEMANTIC ERROR: Operator '+' cannot be used between 'String' and 'Number'.
> print(fib("Hello, World!"));
! SEMANTIC ERROR: Function 'fib' receives 'Number', not 'String'.
> print(fib(4, 3));
! SEMANTIC ERROR: Function 'fib' receives 1 argument(s), but 2 were given.