Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Traductores e Interpretadores
Febrero 2021 – abril 2022

Anexo II - Etapa IV: Completando la Máquina Virtual de Stókhos

Al igual que el anexo anterior, este *no le agrega nada al proyecto*. Solo se provee como ayuda adicional. La sesión de REPL que sigue pone en evidencia como los ciclos de cómputo afectan la evaluación de arreglos. De hecho, no hay diferencia entre arreglos y variables escalares: los arreglos son una generalización de variables a colecciones de éstas.

En el ejemplo, pueden apreciar una diferencia *sintáctica* con Stókhos: el uso de inferencia de tipos. En vez declarar las variables ***a*** y ***b*** con el tipo [***num***], usamos un nuevo token **::** para pedirle a la VM "por favor infiere el tipo de la variable a partir del inicializador". Por si acaso, don't panic: *no deben implementar inferencia de tipos*. Opcionalmente, para que desarrollen su sentido computista, piensen en cómo lo implementarían, ¡para darse cuenta de que es trivial en Stókhos si ya tienen (como la etapa 3 requiere) el cálculo de tipos de expresiones!

```
<1: 1/1 => 1> :: a := [ 'uniform()', 'uniform()', 'uniform()', 'uniform()' ];
ACK: :: a := [ 'uniform()', 'uniform()', 'uniform()', 'uniform()' ];
<4: 2/2 => 2> a
OK: a ==> [0.8936350159453725, 0.01503559916383268, 0.5223552447959996, 0.7867298494223731]
<5: 3/3 => 3> a
OK: a ==> [0.8936350159453725, 0.01503559916383268, 0.5223552447959996, 0.7867298494223731]
<6: 4/4 => 4> a
OK: a ==> [0.8936350159453725, 0.01503559916383268, 0.5223552447959996, 0.7867298494223731]
<7: 5/5 => 5> sys.tick()
ACK: sys.tick()
<8: 6/6 => 6> a
OK: a ==> [0.21035547586657333, 0.42860710827289616, 0.9619688606532166, 0.9521877058877681]
<9: 7/7 => 7> a
OK: a ==> [0.21035547586657333, 0.42860710827289616, 0.9619688606532166, 0.9521877058877681]
<10: 8/8 => 8> sys.tick()
ACK: sys.tick()
<11: 9/9 => 9> a
OK: a ==> [0.128601606176858, 0.8223919081895996, 0.3561298443517278, 0.6957776824659538]
```

Dos programas de prueba sugeridos para histogramas

Para poder realizar la primera prueba tienen que agregar la función ***sqrt*** (raíz cuadrada) a su librería. Siento que olvidé agregarla en la etapa 3, pero si ya han implementado la función ***ln*** (logaritmo natural) ***sqrt*** sigue el mismo patrón: la única diferencia, aparte de llamar a la función Math.sqrt de JS o Python, es que el cero *pertenece* al dominio de la función.

a) Considere el siguiente modelo (programa) en Stókhos:

```
num u1 := 'uniform()';

num u2 := 'uniform()';

num x := 'sqrt(-2 * ln(u1)) * cos(2 * pi() * u2)';

num y := 'sqrt(-2 * ln(u1)) * sin(2 * pi() * u2)';


[num] hx := histogram('x', 1000, 30, -3.0, 3.0);

[num] hy := histogram('y', 1000, 30, -3.0, 3.0);
```

Comparen los histogramas (distribuciones empíricas) de las dos variables. Este ejemplo está inspirado en la transformación de Box–Muller para generar distribuciones en dos variables. La última página de este documento muestra el histograma de ***x*** gráficamente.

b) Usando la expresión '2 * ***uniform***() – 1' de manera adecuada, y aprovechando las funciones ***array*** (constructor de arreglos) e ***histogram***, generen N veces un arreglo, de tamaño K de números aleatorios entre -1 y 1, para obtener la distribución del valor promedio del arreglo. El siguiente ejemplo muestra un modelo "cableado" para K = 4:

```
num N := 1000;

[num] a := ['2 * uniform() – 1', '2 * uniform() – 1', '2 * uniform() – 1', '2 * uniform() – 1'];

num promedio := 'avg(a);'

[num] resultado := histogram('promedio', N, 40, -1, 1);
```

Probando valores de K = 1, 4, 16, 49, verifiquen empíricamente el teorema del límite central y la ley de los grandes números. Usen valores de N razonables para correr su código.

Y siempre recuerden: para lograr excelencia **usen su producto lo más posible**:

"Dog food manufacturers should eat their own dogfood!"

Aquí pueden apreciar, en ZenSheet ™, la distribución de la variable ***x***, evaluando el modelo de cómputo 100.000 veces, usando 30 + 2 barras en el intervalo [-3, 3).

Este proyecto fue realizado con la contribución de estudiantes y egresados de la USB: Mónica Figuera, Richard Lares Mejías, Javier López Lombano, y Pablo Maldonado.

ZenSheet sigue evolucionando: un futuro egresado de la USB, Jean Paul Yazbek, acaba de completar una pasantía, avanzando la nueva versión del producto.

https://2017.splashcon.org/details/live-2017/5/ZenSheet-a-live-programming-environment-for-reactive-computing

https://ieeexplore.ieee.org/document/9476942