Trabajo Práctico Especial

72.39 - Autómatas, Teoría de Lenguajes y Compiladores

Brizi et al

Ail, Brian

Baader, Juan Martín

Bergagna, Federico Manuel

Rodríguez Brizi, Manuel

Índice

Idea del lenguaje

Desde el comienzo de la materia el equipo tenía ganas de enfocar este trabajo al campo de la física. Planteamos la idea de hacer, en primer lugar, un lenguaje relacionado con los movimientos de cuerpos que se ven en las primeras físicas al entrar a la universidad: movimiento rectilíneo uniforme, movimiento rectilineo uniformemente variado, tiro oblicuo, tiro vertical, caída libre. Pero, dado que ya la mayoría de nosotros vivimos (y sufrimos) dichos campos de estudio, decidimos enfocarlo más a algo de lo que no tenemos mucho conocimiento pero nos resulta más interesante, como es la atracción entre los planetas y las interacciones entre ellos. Sabemos que este campo de la física es algo que depende de cientas de variables, que todas son muy importantes y que en mayor o menor medida, todas afectan al sistema. Pero, si tenemos en cuenta algunas de estas variables con las que estamos familiarizados como la masa, la velocidad, la aceleración y vemos de qué forma interactúa un sistema dado un “sistema inicial”, se obtienen resultados realmente geniales.

Objetivo del lenguaje

Se busca poder construir a través del lenguaje un sistema planetario inicial y ver cómo interactúa el mismo a través del tiempo, pudiendo configurar velocidades iniciales, dirección y sentido a los planetas. También su masa y el radio, que influirá en cuanto atraerá a otros cuerpos hacia él.

Consideraciones realizadas

Si bien intentamos generar el AST correspondiente al input, por cuestiones de tiempo no llegamos a terminarlo, por lo que decidimos hacer que, al terminar de reconocer un terminal, se imprima directamente. Sabemos que no es lo mismo y que lo correcto sería generar el AST, pero pudimos obtener el resultado que deseábamos y necesitábamos.

No logramos que LEX reconozca los espacios en las expresiones regulares por lo tanto, los strings no pueden contenerlos, solamente letras. (y números si no son nombres de variables)

Las distancia entre las entidades se miden en unidades astronómicas.

Si bien no hay un límite estipulado de la cantidad de entidades simultáneas, por cuestiones de salud del procesador es recomendable probarlo con pocas puesto que la librería gráfica y los cálculos físicos resultan un poco pesados. También la librería gráfica genera problemas cuando corre, dando errores que no sabemos cómo manejar. Por ejemplo, algunas veces imprime a salida estándar, a veces freezea el programa, o no se puede cerrar (en estos casos hay que usar *pkill -9 planets*). La mayoría de los problemas que pueden llegar a aparecer son causadas por no saber utilizar correctamente/limitaciones de la librería gráfica.

Los colores están representados a través de números enteros, que van desde el 0 al 15, siendo cada uno:

- Negro = 0
- Azul = 1
- Verde = 2
- Cian = 3
- Rojo = 4
- Magenta = 5
- Marrón = 6
- Gris claro = 7
- Gris oscuro = 8
- Celeste = 9
- Verde claro = 10
- Cian claro = 11
- Rojo claro = 12
- Magenta claro = 13

- Amarillo = 14
- Blanco = 15

A través de la función *print()* no se pueden imprimir espacios, y la función *print_string()* sólo puede recibir como parámetro el atributo “name” de un planeta.

Al final de cada programa de prueba, debe haber un enter. Esto se debe a que la gramática es muy restrictiva y poco flexible.

Descripción del desarrollo del Trabajo Práctico

Podemos decir que nuestro trabajo se redujo considerablemente al disponer del código que se encarga de hacer los cálculos físicos que determinan la fuerza de interacción entre las entidades. Esto nos permite enfocarnos en YACC y en LEX, que es la idea de hacer este trabajo, y además poder hacerlo de la temática que nos gusta.

En tanto al trabajo, se parsea y se da significado semántico al input en el lenguaje que desarrollamos (como es esperado), y desde YACC se genera código C intermedio. Desde dicho código intermedio se escribe un archivo de texto plano adicional, que contiene la información recolectada desde el input en el formato necesario para que el motor que calcula las fuerzas de interacción pueda operar correctamente. Una vez generado el archivo intermedio, mediante una librería gráfica y el output del programa *planets* se representan visualmente las interacciones entre las entidades.

Descripción de la Gramática

Intentamos hacer la gramática lo más sencilla posible, pero nos vimos obligados a ir extendiendola a medida que avanzabamos con los requisitos del trabajo. Dicha gramática permite definir planetas a través del tipo Planet de la siguiente forma:

```
Planet p;
```

Por supuesto, antes de agregarlo, deberíamos definir los valores para la masa, el radio, la velocidad inicial y la posición inicial:

```
p.mass  = 3;
p.radius = 2;
p.xvel = 1;
p.yvel = 0;
p.xpos = 1;
p.ypos = 1;
```

También, para agregar personalización a cada entidad, podemos asignarle un nombre y un color:

```
p.color = 4;
p.name = "earth";
```

Finalmente, cuando la entidad ya esta lista, podemos agregarla al sistema mediante:

```
addplanet(p);
```

Por otra parte, soportamos variables de tipo cadena de caracteres y también de tipo entero:

```
int a = 10;
string b = "houston_we_have_a_problem";
```

Así como también se pueden definir operaciones para las variables (de tipo entero):

```
int b = 10 * 10;
int c = b / 2;
```

Para imprimir por salida estándar:

```
Planet p;
p.name = "Earth";
print("El_planeta_se_llama:");
print_string(p.name);
printf("Y_tiene_una_masa_de:");
print_digit(p.mass);
```

También se pueden utilizar estructuras de repetición y de decisión:

```
int a = 3;
while(a > 0){
    print("Hola_mundo!")
    a--;
}

int b = 5;
if(b == 5){
    print("b_es_5!");
}
```

Dificultades encontradas

En un comienzo, la dificultad misma de este trabajo fue comprender cómo funcionan LEX y YACC. Conocer y aprender a usar una herramienta desde cero nunca es fácil y mucho menos cuando es de bajo nivel. Fuera de estas obvias dificultades, tuvimos la pésima idea de dejar las estructuras de repetición y condicionales para el final del trabajo. Teniendo casi todo el resto terminado, tuvimos que realizar cambios a lo largo y ancho del proyecto para poder adaptarlo a estas estructuras.

Una dificultad extra que tuvimos al utilizar el código anteriormente mencionado es que tuvimos que aprender lo básico del uso de la librería gráfica ‘libgraph’ para poder representar los planetas y su translación de

forma visual, puesto que el código solo devuelve los valores de la posición actual de las entidades.

Un problema que no pudimos solucionar es el flickering de la pantalla. Dado el tiempo que tarda en renderizar los cuerpos y limpiar la pantalla, sobretodo cuando son muchos, se pierde calidad en la simulación, por momentos, algún cuerpo puede desaparecer. Para intentar solventar esto se utilizaron threads para separar la parte de cálculo físico de la parte de render y si bien ayudo bastante no lo soluciona completamente. En un futuro se podría usar una librería gráfica más avanzada y lidiar con estos problemas con mayor facilidad.

Otro problema que no pudimos solucionar es intentar hacer una órbita perfecta alrededor de un cuerpo, aun poniendo magnitudes siguiendo fórmulas físicas para calcular una órbita perfecta de un planeta alrededor de un sol, se nos iba alejando cada vez más, no pudimos encontrar si era un problema de redondeo o del engine de física.

Futuras extensiones

Al ser un campo en continuo crecimiento de la física, hay muchas posibles extensiones para este trabajo serían muy interesantes de implementar en nuestro compilador. A continuación, enumeramos algunas de las más interesantes:

- Soportar cuerpos especiales, como estrellas o asteroides. Esta extensión, en principio, no sería muy difícil de implementar, puesto que implicaría cambios en la gramática y algunas configuraciones adicionales en el motor de cálculo de las fuerzas de interacción.
- Simular sistemas planetarios a partir de un conjunto de sistemas planetarios (varios sistemas interactuando entre sí). Esta extensión si significa una dificultad puesto que deberíamos adaptar el código que utilizamos para soportar varios sistemas en conjunto y que los mismo interactúen como uno.
- Aumentar la capacidad expresiva de la simulación, dando un tamaño a las entidades, añadiendo también la rotación sobre su

eje a las mismas. Esta extensión, si tuviéramos los conocimientos de cómo utilizar herramientas gráficas, estamos convencidos de que no sería una gran dificultad. Puesto que ningún integrante del equipo sabe utilizar herramientas gráficas, sí se convierte en una dificultad.

- Agregar configuraciones al programa de simulación, como la cantidad de iteraciones, la constante gravitacional o cuestiones gráficas como el plotteo de las órbitas.

Construcción del programa

Para construir el programa se deben instalar librerías gráficas necesarias para el archivo *planets*, y para ello se deben seguir los siguientes pasos[1]:

1) Descargar libgraph: https://download.savannah.gnu.org/releases/libgraph/libgraph-1.0.2.tar.gz
2) Instalar librerias requeridas:

```
sudo apt-get install build-essential
```

```
sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev guile-1.8 guile-1.8-dev libsdl1.2debian libart-2.0-dev libaudiofile-dev libesd0-dev libdirectfb-dev libdirectfb-extra libfreetype6-dev libxext-dev x11proto-xext-dev libfreetype6 libaa1 libaa1-dev libslang2-dev libasound2 libasound2-dev
```

3) Descomprimir libgraph-1.0.2.tar.gz
4) Dentro de la carpeta recién descomprimida ejecutar:
    a) ./configure --disable-guile
    b) make
    c) sudo make install
    d) sudo cp /usr/local/lib/libgraph.* /usr/lib
5) Ya se puede compilar el *planets* con el siguiente comando: *gcc -o planets planets.c -lgraph -lm -lpthread*

[1]Instalacion de libgraph: https://tutorialspoint.dev/computer-science/computer-graphics/add-graphics-h-c-library-gcc-compiler-linux

Ejecución del programa

Los pasos para realizar una simulación son:

1) Generar el archivo intermedio corriendo la línea:
   $> ./generate_planet_info.sh path_al_archivo

2) Correr el archivo intermedio llamado “planet_info” con la siguiente línea:
   $> ./planets planet_info

3) Enjoy the show!

A modo de ejemplo, para correr el Test N°1, deberíamos pararnos en el directorio raíz del proyecto y correr las siguientes líneas:

```
$> ./generate_planet_info.sh tests/test1.txt
$> ./planets planet_info
```

Explicación casos de prueba

- Test 1: la Tierra orbita alrededor del sol.
- Test 2: un meteorito pasa cerca del sol, afectando su dirección.
- Test 3: dos entidades de poca masa pasan cerca de un sol (menos masivo que el de nuestro sistema solar), donde redireccionan su rumbo debido al campo gravitatorio del sol y también por la interacción entre ellos.}
- Test 4: nuestro sistema solar. Funciona un poco lento debido a que la distancia entre planetas y el sol es bastante grande.
- Test 5: la Luna orbita la Tierra, y la Tierra orbita el sol. Igual que en el test anterior, hay una distancia grande entre el sol y la Tierra y una distancia corta entre la Tierra y la Luna, por lo que quedan un poco chicos los gráficos.

Referencias

El código que calcula la fuerza de interacción entre las entidades:
https://rosettacode.org/wiki/N-body_problem#C

Información utilizada sobre la Fuerza de Interacción Gravitatoria:
https://www.fisicalab.com/apartado/interaccion-gravitatoria#contenidos