Teoría de Lenguajes

Trabajo Práctico.
2º Cuatrimestre, 2014

1. Introducción

Los sistemas generativos o de generación de contenidos por procedimientos (procedural content generation) son técnicas utilizadas para crear imágenes, música o animaciones de forma automática o semi-automática. Estas técnicas apuntan a facilitar la creación de una composición compleja especificando solo unas pocas reglas y/o unos pocos parámetros en lugar de tener que definir individualmente cada parte de la composición.

Uno de los ejemplos más conocidos son los llamados Lindenmayer Systems, o simplemente L-systems. Este tipo de sistemas son en verdad una forma de gramática formal en donde los símbolos son interpretados como comandos gráficos similares a los del lenguaje LOGO y las reglas describen que tipo de figura va a dibujarse. Existen también alternativas que llevan esta idea aún más lejos y que permiten realizar composiciones más complejas [CFGD[1]] y aún composiciones en 3 dimensiones [Structure Synth[2]].

En este enunciado se describirá un lenguaje sencillo que permite realizar un tipo de generación de contenidos por procedimientos para composiciones visuales en tres dimensiones. El trabajo consistirá en escribir una gramática para este lenguaje, un analizador sintáctico que pueda procesar archivos escritos en el mismo y unos procedimientos para mostrar el resultado.

2. Sintaxis del lenguaje

El lenguaje posee en principio tres componentes sintácticos básicos: las primitivas, las transformaciones y las operaciones. Las primitivas son las que dibujan algo, las transformaciones modifican su aspecto y las operaciones las agrupan de varias formas. Denominaremos a las combinaciones de estos componentes como *elemento*s. Existe además un cuarto componente de la sintaxis llamado *reglas* que permite asignarle un nombre a un elemento. Los elementos a los que puede asignarsele un nombre mediante una regla no solo pueden estar formados por una combinación de primitivas, transformaciones, operaciones sino que también pueden incluir otros nombres de reglas. Esta posibilidad de utilizar nombres de reglas en las reglas es lo que hace interesante a este lenguaje. El resultado final, es decir, lo que se termina dibujando o mostrando lo llamaremos *composición visual*.

2.1. Primitivas

Las primitivas son los únicos *elementos* del lenguaje que en última instancia terminan mostrándose en la composición visual. Existen tres primitivas válidas.

[1] http://www.contextfreeart.org/
[2] http://structuresynth.sourceforge.net/

box Muestra un cubo blanco de dimensión 1 por lado centrado en el origen.

ball Muestra una esfera blanca de diámetro 1 centrada en el origen.

_ El guión bajo no muestra nada pero es útil en varias situaciones.

2.2. Transformaciones

Las transformaciones permiten alterar la forma en que se muestran las primitivas u otros *elementos* en la composición visual. Las transformaciones válidas son:

rx,ry,rz Rotación en grados sobre los ejes X, Y y Z respectivamente.

sx,sy,sz,s Escala sobre los ejes X, Y y Z respectivamente, y s para los 3 ejes simultáneamente.

tx,ty,tz Translación sobre los ejes X, Y y Z respectivamente.

cr,cg,cb Nivel de color rojo, verde y azul respectivamente.

d Máxima profundidad. (Se explica en detalle más adelante)

Cada transformación se indica mediante dos puntos, el nombre de la transformación y una expresión numérica. El resultado de aplicar una transformación a un *elemento* es un nuevo *elemento* transformado, de manera que es posible aplicar varias transformaciones sucesivamente.

Por ejemplo, para reducir a la mitad el nivel de azul de un cubo, rotarlo 45 grados sobre el eje Y y trasladarlo sobre el eje Z a la posición -1 podemos escribir:

```
box : cb 0.5 : ry 360/8 : tz -1
```

Se debe notar que el orden en que se aplican las transformaciones es importante.

```
box :ry 45 :tx 1        no es lo mismo que
box :tx 1  :ry 45
```

A partir de ahora notaremos como :T a una secuencia genérica de transformaciones.

2.3. Operaciones

Las operaciones permiten combinar varios elementos para incluirlos de distintas formas en la composición visual. El resultado de una operación es también un elemento. Las siguientes son operaciones válidas, siendo A y B elementos y N una expresión numérica:

A&B Conjunción de dos elementos. Ambos elementos, A y B, serán mostrados.

A|B Disyunción de dos elementos. Solo un elemento al azar, A o B, será mostrado.

[A] Agrupación. Las operaciones de A son agrupadas como un solo elemento.

A^N Potenciación. El elemento A será repetido N veces.

<**A**> Opcional. El elemento A puede o no ser mostrado.

La operación de conjunción tiene precedencia sobre la de disyunción y la asociación de ambos operadores es por izquierda. Es decir que A | B & C es equivalente a A | [B & C]. Sin embargo cabe notar que A | B | C no es exactamente equivalente a [[A | B] | C]. En el primer caso tanto A, B y C tienen iguales probabilidades de ser mostrados, pero en el segundo caso C tiene el doble de probabilidades que A o B.

Con el ejemplo anterior se puede ver que los corchetes no solo permiten afectar la precedencia y asociación de los operadores sino que además pueden tener otros efectos. Un motivo en especial para agrupar operaciones con corchetes es el poder aplicarle al resultado una transformación, por ejemplo [A & B]:T, ya que las transformaciones tienen mayor precedencia que los operadores. Otro uso especial de los corchetes junto a una transformación está en la operación de potenciación.

La operación de potenciación A^N es equivalente a realizar N conjunciones del elemento A. En el caso particular en el que A esté formado por una agrupación con corchetes B junto a una transformación T, esto es, [B] : T, la transformación también será aplicada sucesivamente a cada conjunción. Es decir:

```
 A^N      es equivalente a  [ ...[ [ A ] & A ] ...& A ]          pero...
[B]:T^N   es equivalente a  [ ...[ [ B ]:T & B ]:T ...& B ]:T
```

Ambas con N conjunciones. Notar que el segundo ejemplo, en donde la transformación T se aplica N veces, tiene este comportamiento por ser un caso particular en el que la transformación afecta a un elemento agrupado con corchetes. En cualquier otro caso el comportamiento debe ser como en el primer ejemplo.

La operación de opcional <A> es equivalente a hacer [A | _]. Esto significa que A tiene un 50 % de probabilidades de ser mostrado.

2.4. Reglas

Es posible asignarle un nombre a una combinación de elementos del lenguaje, a esta asignación la llamaremos *regla*. Las reglas estarán formadas por el nombre, un signo igual y una definición de elemento.

```
nombre = elemento
```

Es posible también definir más de una regla con el mismo nombre.

```
nombre = elemento1
nombre = elemento2
```

Esto es equivalente a tener una sola regla con la definición:

```
nombre = elemento1 | elemento2
```

El nombre puede estar formado exclusivamente por una combinación de letras mayúsculas y minúsculas. La definición de elemento puede estar formada por una combinación de primitivas, transformaciones, operaciones e inclusive, nombres de reglas. Mostrar un elemento que contiene un nombre de regla es equivalente a reemplazar la definición de elemento de esa regla en donde aparece el nombre. Cuando ocurre este tipo de reemplazos diremos que la profundidad aumenta en uno. Por ejemplo esta regla utiliza su propio nombre en la definición de elemento.

```
spiral = ball & spiral:tx 1:rz 15:s 0.9
```

Como este tipo de reemplazos pueden ocurrir recursivamente, y potencialmente de forma infinita, vamos a utilizar tres estrategias que nos permitan tener control sobre la situación. Primero vamos a limitar la profundidad total de la composición visual. Con una profundidad máxima de 100 es suficiente para obtener buenos resultados y prevenir potenciales recursiones infinitas.

Además vamos a utilizar las transformación *:d* para poder limitar la profundidad máxima desde el punto en donde es utilizada. El efecto será el de no exceder la cantidad de reemplazos de nombre que se hagan a partir de ese elemento. Por ejemplo las siguientes reglas dibujarán las 10 esferas correspondientes a la profundidad máxima indicada más una esfera de comienzo.

```
scene  = spiral:d 10
spiral = ball & spiral:tx 1:rz 15:s 0.9
```

Finalmente vamos a poder indicar que elemento se debe mostrar si se alcanza la profundidad máxima permitida utilizando un punto al final del nombre en la declaración de una regla. A este tipo de reglas las llamaremos *reglas finales*. Por ejemplo, agregando la siguiente regla final, la espiral terminará siempre en una bola roja.

```
scene   = spiral:d 10
spiral  = ball & spiral:tx 1:rz 15:s 0.9
spiral. = ball:cg 0:cb 0
```

Se debe notar que la regla final no se aplicará exclusivamente al alcanzar la profundidad máxima sino que tiene iguales probabilidades de aplicarse que cualquier otra regla. Es decir que el efecto que tiene es de limitar la profundidad a una *igual o menor* a la indicada con la transformación :d.

2.5. Definición de programa

Un programa estará formado por una serie de definiciones de reglas y su objetivo será el de especificar una composición visual completa. Para esto es necesario poder especificar por cual regla comenzar. Para esto incluiremos el nombre de regla $ como nombre distinguido de comienzo.

```
spiral = ball & spiral:tx 1:rz 15:s 0.9
$ = spiral:d 25
```

2.6. Otros aspectos de la sintaxis

Los espacios, tabulados y retornos de linea no tienen efecto. Es posible escribir las reglas utilizando cualquier estilo de espaciado. Por ejemplo todas las siguiente reglas son válidas:

```
one=box:tx1:ry1&ball:tz2:cr0
two = box  : tx 1 : ry 1
    & ball : tz 2 : cr 0
```

Los textos encerrados entre comillas dobles serán considerados comentarios y deben ser ignorados. No es necesario que los comentarios tengan un alcance de varias lineas pero si es deseable que se puedan incluir comillas dobles dentro del comentario.

```
rule = box : tx 10              "Este es un comentario"
```

Las expresiones numéricas que se utilizan con las transformaciones pueden estar formadas por números y operaciones aritméticas que incluyan la suma, resta, multiplicación, división y paréntesis. La precedencia y asociación de estos operadores debe ser la esperada comúnmente. No es necesario considerar números enteros y reales por separado pero en algunos casos el valor puede ser interpretado como un entero. Los números también pueden estar notados con su signo, positivo o negativo. De modo que expresiones como *-1 - -1* o *+1++1* deben ser válidas. Las operaciones aritméticas entre elementos no son parte del lenguaje, de modo que algo del tipo *box+ball* no es válido.

3. Detalles de la entrega

La entrega consistirá en un informe escrito y un programa. El informe deberá ser conciso y contener al menos los siguientes items además de los que consideren necesarios para que sea claro:

- Breve introducción al problema.
- Especificación de la gramática, incluyendo reglas y tokens.
- Descripción de cómo se implementó la solución.
- Ejemplos de arboles de derivación con entradas correctas.
- Ejemplos de programas válidos e inválidos con sus resultados.
- Detalle de las opciones que acepta el programa y su modo de uso.
- Detalle de los requerimientos para compilar, instalar y/o ejecutar el programa.
  (De ser necesario incluir las librerías requeridas para su funcionamiento; si no, especificar.)
- El código fuente impreso del programa.
  (Si se usaron herramientas generadoras de código, imprimir la fuente ingresada a la herramienta, no el código generado.)
- Decisiones tomadas y su justificación.
- Conclusiones si las hubiere.

El programa deberá tomar como parámetro por linea de comando un nombre de archivo. Si el archivo contiene un programa válido en el lenguaje aquí especificado deberá mostrar la composición visual resultante, si no deberá mostrar un mensaje de error apropiado.

Para la implementación del analizador léxico y el analizador sintáctico se recomienda el uso de la herramienta PLY (www.dabeaz.com/ply) que es una implementación en Python puro de las herramientas de parsing lex/yacc y permite construir tablas LALR(1) y SLR. En la implementación de las transformaciones se recomienda el uso de la librería NumPy (www.numpy.org) para poder realizar transformaciones lineales mediante el producto de matrices. Y para poder mostrar las composición final se recomienda el uso de la librería Visual Python (vpython.org) que permite mostrar objetos en tres dimensiones de forma muy sencilla. Todos estos paquetes son multiplataforma y están disponibles desde los enlaces correspondientes y desde los repositorios de Debian, Ubuntu y muchas otras distribuciones de GNU/Linux.

Se recomienda que una vez armada la entrega con todo lo necesario para ser usada sea probada en otra máquina y que se aclare sobre que plataformas fue probada.

El informe debe entregarse impreso y además en un archivo comprimido junto al código a la dirección:

tptleng@gmail.com

Fecha de entrega: 27 de Noviembre de 2014.