# SVGAnimation

Alex Osés Laza, Gonzalo Diez Garrido

Compiladors

Group 32-02

Semester 2015/16-2

FIB - UPC

June 1, 2016

# Contents

# Chapter 1

# Introduction

En este documento explicamos los puntos clave de nuestra practica de compiladores: Hacer un iterprete con una gramática propia, que traduce una serie de instrucciones al lenguaje SVG, escribiendolo en un archivo .html que se puede abrir posteriormente para su visualización.

# Chapter 2

# Descripción del lenguaje

## 2.1 Objetivos

## 2.2 funcionalidades Principales

### 2.2.1 Create

La funcionalidad `Create` nos permite crear objetos svg de tipo `Circle`, de tipo `Rect` (rectangulo) o de tipo `text`.

### 2.2.2 Move

Esta funcionalidad tiene dos variantes:

- **Move**, que nos permite mover el `objeto` o `conjunto de objetos` a una posición dada instantaneamente.
- **Move Time** nos permite mover el objeto o `objetos` a una posición dada en un tiempo determinado por el usuario de forma animada.

### 2.2.3 Modify

Aquí también contamos con dos posibilidades:

- **Modify** nos permite modificar el `atributo svg` o `atributos` de forma instantánea.
- **Modify time** nos da la posibilidad de modificar `atributo svg` o `atributos` con una animación dependiendo del tiempo provisto por el usuario.

### 2.2.4 Show

Una vez más contamos con dos variantes de esta funcionalidad:

- **Show** permite mostrar por pantalla el `objeto` o `bloque de objetos` de manera instantánea.
- **Show time** nos da la oportunidad de mostrar por pantalla el `objeto` o `bloque de objetos` de forma progresiva en función del tiempo.

### 2.2.5 Hide

esta funcionalidad es el caso análogo a la funcionalidad `Show`, pero en vez de mostar el `objeto u objetos` los hace desaparecer.

### 2.2.6 Parallel

Con esta instrucción podemos hacer que varias instrucciones se ejecuten de forma paralela, permitiendo al programador, por ejemplo, mover dos `objetos` al mismo tiempo por pantalla. Tenemos dos variantes:

- **Parallel** El tiempo puede ser diferente para cada instrucción.
- **Parallel time** El tiempo para todas las instrucciones es el mismo.

### 2.2.7 Block

Esta funcionalidad nos permite crear `bloques` de `objetos` y la oportunidad de procesarlos como tales. Esto es util para hacer figuras, por ejemplo: una persona y hacer que se muevan todas las partes juntas de forma sencilla

### 2.2.8 Destroy

Esta instrucción se encarga de destruir el `objeto` en cuestión y hacer que no aparezca más por pantalla.

### 2.2.9 Delay

La funcionalidad `Delay`, produce que el programa espere durante un tiempo determinado, evitando que se produzcan animaciones.

# Chapter 3

# Gramatica

### 3.0.1 Create

```
create  :   CREATE^ ID '('! obj_type ','! expr ','! expr (','! list_attributes )?')'!
        ;
        obj_type : CIRCLE | RECTANGLE | TEXT
         ;
```

### 3.0.2 Move

```
move    :    move_time | move_no_time
        ;
// "move" id x y time
move_time : MOVE_T^ ID  time '('! expr ','! expr  ')'!
        ;
// "move" id x y
move_no_time : MOVE^ ID '('! expr ','! expr ')'!
        ;
```

### 3.0.3 Modify

```
modify    :    modify_time | modify_no_time
        ;
// "modify" id x y time
modify_time : MODIFY_T^ ID time list_attributes
        ;
// "modify" id x y
modify_no_time : MODIFY^ ID list_attributes
        ;
```

### 3.0.4 Show

```
show : show_time | show_no_time;

show_time : SHOW_T^ ID time;

show_no_time :  SHOW^ ID;
```

### 3.0.5 Hide

```
hide : hide_time | hide_no_time;

hide_time : HIDE_T^ ID time;

hide_no_time :  HIDE^ ID;
```

### 3.0.6 Parallel

```
parallel : parallel_time | parallel_no_time;

instructions_notime : move_no_time | modify_no_time | show_no_time | hide_no_time;

block_instructions_notime: instructions_notime';' ( instructions_notime ';')*
                          -> ^(LIST_INSTR instructions_notime+);

parallel_time : PARALLEL^ time '{'! block_instructions_notime '}'!;

instructions_time : move_time | modify_time | show_time | hide_time;

block_instructions_time: instructions_time ';' ( instructions_time';')*
                          -> ^(LIST_INSTR instructions_time+);

parallel_no_time : PARALLEL^ '{'! block_instructions_time '}'!;
```

### 3.0.7 Block

```
block : BLOCK^ ID '{'! (create ';'!)+ '}'!
        ;
```

### 3.0.8 Destroy

```
destroy :   DESTROY^ '('! ID ')'!
        ;
```

### 3.0.9 Delay

```
delay: DELAY^ time;
```

### 3.0.10 Attributes

```
list_attributes : '(' attribute (',' attribute)* ')' -> ^(LIST_ATTR attribute+)
        ;

attribute   :   attribute_name_color^ ':'! obj_attribute
            |   attribute_name_expr^ ':'! expr
            ;
obj_attribute   :   color
                | ID
                |   ID '.' ( attribute_name_color) -> ^(ATTR ID attribute_name_color)
                ;
                attribute_name_color    : COLOR
                                        | STROKE
                        ;
```

```
attribute_name_expr : POSX
                    | POSY
                    | WIDTH
                    | HEIGHT
                    | 'r'
                    ;

color    :   COLORHEX -> ^(COLORHEXA COLORHEX)
         |   RGB '(' expr ',' expr ',' expr ')' -> ^(COLORINT expr expr expr)
         |   RGBPRCTJ '(' expr ',' expr ',' expr ')' -> ^(COLORPRCTJ expr expr expr)
         |   color_keyword -> ^(COLORKEYWORD color_keyword)
         ;
```

tambien podemos acceder a cualquier atributo de cualquier `objeto` o `bloque` con la id.attribute. gracias a la extensión en la regla atomic

```
         |   ID '.' (attribute_name_expr) -> ^(ATTR ID  attribute_name_expr)
    |   ID '.' ( attribute_name_color) -> ^(ATTR ID attribute_name_color)
```

### 3.0.11 Tokens

```
HASHTAG : '#';

CREATE  : 'create';
DESTROY : 'destroy';
MOVE_T  : 'movet';
MOVE    : 'move';
MODIFY_T: 'modifyt';
MODIFY  : 'modify';
RGBPRCTJ: 'rgbp';
RGB     : 'rgb';

RECALC  : 'reComputeCenter';

SHOW_T   : 'showt';
SHOW    : 'show';
HIDE_T   : 'hidet';
HIDE    : 'hide';
DELAY   : 'delay';

BLOCK : 'block';
PARALLEL: 'parallel';

COLOR   : 'color';
WHITE   : 'white';
BLACK   : 'black';
BLUE    : 'blue';
RED     : 'red';
GREEN   : 'green';
POSX    : 'x';
POSY    : 'y';
CIRCLE  : 'circle';
STROKE : 'stroke';
WIDTH   : 'width';
```

```
HEIGHT   : 'height';
RECTANGLE : 'rect';
TEXT : 'text';
```

# Chapter 4

# Semántica

En esta seccion explicaremos que cambios hemos hecho en el iterprete, como y porque:

### 4.0.1 Data

Ya que nuestro interprete iba a tratar con estructuras de datos diferentes a integers o booleans, modificamos el `Data` añadiendo tres tipos diferentes: `String`, `Object` y `Block`.

- **String** Ya que tendríamos colores y podríamos acceder a atributos de los objetos a traves de objeto.atributo, necesitabamos para uso interno del iterprete un tipo de variable string.
- **Object** Un `objeto` se compone por una lista de atributos que describen un figura SVG, a las que podemos acceder a su valor a partir de su nombre de atributo.
- **Block** Para los `bloques` guardaremos los nombres de las variables que estan dentro del `bloque` en un hashset de strings y, además guardaremos la posicion x e y del centro del conjunto de `objetos` como atributo de la estructura de datos de los `objetos`.

### 4.0.2 States

El primer problema que nos encontramos fue la idea del destruir un objeto. Al ser nuestro interpre un traductor de nuestro lenguaje a puro svg, la unica manera de eliminar un objeto era hacerlo invisible. (Con javascript podríamos eliminar el código que mostramos en el navegador, pero preferimos complicarnos menos, ya que ninguno de los dos sabe javascript, y usar puro svg.)

Sabiendo que tendríamos que eliminar objetos y que en el Stack de Asl no es posible eliminar variables, decidimos hacer una clase nueva `States.java` en la cual almacenariamos los diferentes estados a medida que vamos interpretando el programa. Gracias a esta nueva estructura, solo cuando ha acabado el programa se saca el código SVG.

En esta nueva estructura se compone por un array donde cada posicion del array sería los `estados` que ha tenido ese `objeto` durante todo el programa. Los estados de un objeto se guardan como un array de strings, donde cada string sería una animación SVG (un transformación de un `atributo` del objeto que empieza en cierto momento y dura cierto tiempo).

Además, teniamos un mapa para saber que posicion del anterior array describe los estados de que id. Esta clase también es la encargada de guardar el tiempo en el que se encuentra la aplicación, para que cuando se hace cierta modificación del estado, saber en que momento se tiene que hacer.

A continuacion se explica que se hace exactamente con las tres operaciones de esta clase.

#### Create

En el caso de crear un nuevo objeto, lo que necesitamos hacer es añadir una nueva posicion al array que guarda los estados, y asignarle a la id del nuevo objeto, la posicion de esta nueva posicion del array. Una vez que sabemos donde vamos a ir guardando todas las modificaciones del estado, se inicializa el estado con el tipo del objeto y se crea el Data que va a tener todos los atributos del objeto.

#### Modify

Cuando queremos modificar, tenemos que tener en cuenta que no solo tenemos que modificar el estado del objeto que queremos modificar, sino que tambien tendremos que modificar el data asociado a ese objeto, además de que esas modificaciones se puden hacer en cierto tiempo, con lo que tendremos que sumarle ese tiempo al tiempo global de la aplicación.

#### Destroy

En cuanto a destruir, como lo que tenemos que hacer es poner su opacidad a 0, se llama al modify anterior con el atributo `opacity` y asignandole un valor de 0 en tiempo 0. De esta manera al destruir un objeto desapareceria instantaneamente de la animación. Para que ya no se puede volver a acceder a este objeto hasta que se vuelva a crear, se modifica la posición de su posicion del states a -1, de esta manera si tratamos usar alguna instruccion que use un bloque con esa id, sabremos que esa id en realidad no está asignada a ningun bloque.

### 4.0.3 Stack

Para poder declarar objetos dentro de funciones y que estos se eliminaran cuando se saliera del scope de esa funcion, hemos añadido la funcionalidad de que cuando se elimina un ActivationRecord, se mira si hay algun objeto y si lo hay se llama al destro del States.

### 4.0.4 Instrucciones

#### Create

Al crear un `objeto`, se usa el `create` del `States` para definirla y a continuacion se llama al `modify` para definir su `posicion`, ademas de sus `atributos`.

En el caso de los `bloques de objetos`, se crea un data , inicializando sus variables i se calcula el centro de la caja englobante del bloque, almacenandolo posteriormente en el Data como un atributo, para acabar tambien se crea una nueva variable con el id provisto.

#### Move, Modify, Show, Hide

Estas cuatro instrucciones internamente hacen lo mismo: Llaman al `modify` de `States`. La única diferencia es que el move solo modifica la posicion del objeto, show y hide modifican la `opacidad` y el modify modifica lo que el usuario quiera.

#### Parallel

Esta instruccion se compone por dos variantes: con tiempo o sin tiempo, como hemos explicado anteriormente.

El paralel sin tiempo simplemente ejecuta cada instruccion del bloque de instrucciones y seguidamente resetea el tiempo que ha añadido esa instuccion.

El paralel con tiempo tiene un significado semántico mas complejo. Esta instruccion, cuando tiene un tiempo asociado, modifica el significado de todas las instrucciones de su bloque de instrucciones. De esta manera nos vemos obligados a modificar cada instruccion pasando a su variante que recibe un tiempo.

### Block

Para la semántica de esta instrucción hemos optado por transformar el arbol y generar instrucciones básicas para cada `objeto` del bloque, haciendo las modificaciones de forma paralela en caso de las instrucciones con tiempo

### Asign

La instrucción ya estaba definida por el lenguaje Asl, pero en nuestro caso hemos decidido que cuando se asignara un bloque a otro, el objeto al que se le está asignando, si existe es borrado, y a continuacion se le copian todos los atributos del objeto asignado.

# Chapter 5

# Juegos de pruebas

## 5.1 Prueba 1

demostración de funcionamiento de create,modify,parallel, hide y como podemos acceder y obtener valores de atributo de otros `objetos`.

## 5.2 Prueba2

demostración del funcionamiento de move, y delay.

## 5.3 Prueba3

demostración de funcionamiento de los `Bloques de objetos`

## 5.4 Prueba4

demostración de parallels anidados en un while.

# Chapter 6

# Posibles extensiones no implementadas

### 6.0.1 Transformacion de un objeto a otro

Una de las cosas que nos hubiera gustado implementar hubiera sido poder modificar los tipos de los objetos con una instruccion concreta, de tal modo que puedieras tener un circulo con cierto color de fondo y cierto stroke, y aplicando una instruccion, transformarlo en un cuadrado con el mismo tamaño, mismo color y mismo stroke.

Para hacer esto ahora mismo tendríamos que crear el cuadrado con los mismos atributos que el circulo, asignarselo al circulo y eliminar el cuadrado origianal.

### 6.0.2 Copia en asignación de bloques

Una forma distinta de hacer las asignaciones de bloques, podria haber sido que en vez de asignar por referencia (propio de java) hacer una copia del Data y asignarlo a la nueva variable.

### 6.0.3 Funciones dentro de parallel

En nuestro programa actual, no se pueden hacer llamadas a funciones dentro de bloques `parallel`, ya que tendríamos que mirar que esas funciones no tuvieran paralels dentro.

### 6.0.4 Bloques y Objetos como parametro de funciones

Para las llamadas a funciones utilizamos la base aportada por la asignatura por tanto no contemplamos el caso en que nos pudiesen pasar `objetos` como parametro en nuestras funciones.

### 6.0.5 Extensión de la gramatica

una de las extensiones más faciles es la de extender la gramatica ya que, tal como hemos programado nuestro compilador, para añadir nuevos atributos, lo unico que tendriamos que hacer, es añadirlos en la gramatica en la regla, attribute_name_expr que se puede encontrar en 3.0.10