COMPILADORS

Yoel Cabo

Jordi Piqué

Data: 01/06/2016

Compilador SVG

ÍNDEX

1. Introducció

En aquest treball ens hem proposat crear un nou llenguatge de programació per escriure animacions, tal com s'especifica en un dels enunciats proposats pel treball de l'assignatura compiladors.

La motivació principal per desenvolupar aquest nou llenguatge és la de facilitar la construcció de figures geomètriques en SVG i, sobretot, estructurar d'una manera diferent la conceptualització dels moviments aplicats a les figures geomètriques. Això vol dir que l'usuari veurà un llenguatge de programació de caràcter imperatiu, amb una sintaxi senzilla que li pot recordar, per exemple, Python, amb la conseqüent facilitat per aprendre'l que aquestes característiques comporten.

Permetrà utilitzar tota la potència de SVG però sense haver de conèixer aquest llenguatge. A més a més, se simplificaran i s'empaquetaran algunes de les seves caracterítiques per poder guanyar flexibilitat i facilitat en la seva programació.

En els següents apartats, viatjarem a través de la definició de les paraules clau (lèxic), la definició de la sintaxi, la interpretació de l'arbre AST resultant dels dos anàlisis anteriors i la conseqüent comprovació semàntica i la traducció de les sentències que ens permeten obtenir el codi SVG desitjat per simular les animacions descrites per l'usuari amb el nostre llenguatge.

2. Objectius

Si el lector ja ha donat un cop d'ull al primer apartat d'aquesta memòria, haurà intuït més o menys els objectius que s'han plantejat assolir amb el desenvolupament d'aquest nou llenguatge.

Seguint amb la línia que marca la motivació per fer aquest projecte, la idea principal és la de crear una plataforma que abstregui a l'usuari del llenguatge SVG però que pugui utilitzar tota la seva potència mitjançant un llenguatge de caràcter imperatiu amb una sintaxi basada en el llenguatges imperatius més famosos.

Per assolir aquest objectiu correctament, també es necessita que el llenguatge sigui de la forma més senzilla i fàcil d'aprendre possible, ja que un llenguatge críptic i difícil no aportaria cap valor afegit respecte SVG.

Un altre objectiu, però que està més encarat de cara al programador d'aquest llenguatge és el d'assolir els dos objectius anteriors modificant el mínim possible el llenguatge ASL per tal de reduir al màxim les hores de programació i augmentar així la productivitat.

Per resumir aquest apartat i deixar ben clars els objectius d'aquest treball, just a continuació s'enumeren en una llista:

- Creació d'un nou llenguatge que compili a SVG
- Donar-li el caràcter, l'estructura i la potència dels llenguatges imperatius
- Aportar bucles, if's, assignacions i funcions per poder fer càlculs complexos
- Crear una sintaxi senzilla i coneguda
- Facilitar la creació de moviments en animacions complexes

3. Descripció del llenguatge

3.1 Característiques

Per anar introduint al lector en aquest nou llenguatge, en aquest apartat s'explicaran les característiques principals que el defineixen (paradigma, sistema de tipus, detalls que el diferencia d'altres llenguatges…).

Paradigma

És un llenguatge imperatiu i estructurat.

Compilat o interpretat?

Aquesta és una pregunta amb resposta ambigua. El llenguatge que s'ha creat interpreta les línies de codi que ha escrit el programador, fa els càlculs pertinents que se li indiquen amb tals línies i retorna com a resultat la compilació d'un programa en SVG. És a dir, el llenguatge actua com un llenguatge típic de scripting (Python), tot i que l'objectiu de la

interpretació del programa és retornar com a sortida la traducció de certes sentències d'aquest a lleguatge SVG.

Amb aquesta explicació s'espera que el lector hagi pogut veure com aquest llenguatge es pot veure des del punt de vista de llenguatge interpretat i també com a llenguatge compilat.

Sistema de tipus

El sistema de tipus utilitzat és dinàmic i feble. Això vol dir que el tipus d'una expressió no s'indica literalment en el codi, sinó que s'infereix en temps d'execució.

Aquesta característica comporta que una mateixa variable es pugui reutilitzar per emmagatzemar valors de diferent tipus en parts diferents del programa. També comporta que un error de tipus pugui saltar en temps d'execució, tot i que si mirem el nostre llenguatge com a llenguatge interpretat, l'error saltarà en temps de compilació.

Altres característiques

Per tal de poder-lo utilitzar com un llenguatge senzill de scripting, a part de tenir la capacitat de realitzar càlculs, es necessita poder-los visualitzar d'alguna manera. Per això existeix la sentència "print", que permet imprimir per pantalla el valor que segueix a aquesta paraula clau.

Totes les assignacions, els paràmetres que es passen a una funció i les construccions de les estructures pròpies del llenguatge es fan per còpia. Això vol dir que internament es crea una nova còpia del valor d'una variable quan, per exemple, s'assigna a una altra. En Java, la instrucció x = y fa que x apunti al mateix objecte que y (si no són classes primitives). En canvi, en el llenguatge que presentem l'assignació x = y fa es creï una còpia de l'objecte al qual y es refereix i passi a ser designat mitjançant x.

No es permet que una línia de codi creï un objecte sense que aquest no s'assigni a una variable, ja que no té sentit crear un objecte que després no es pugui fer servir. Un exemple d'aquest cas seria: Circle [color=255:0:0, radi=20]. Aquesta sentència, sense assignar-la a una variable, no s'accepta.

3.2 Funcionalitats

Com ja s'ha explicat en els apartats anteriors, aquest llenguatge hereda les capacitats de dibuixar figures geomètriques i animar-les que té SVG i, a més a més, també hereda caracterítiques típiques dels llenguatges imperatius, per facilitar l'aprenentatge de l'usuari i adquirir molta més potència de càlcul que la que té SVG (que és pràcticament nul·la).

3.2.1 Part imperativa

Passarem primer a explicar la part imperativa de SVG que pot recordar a la d'altres llenguatges com Python o ASL, ja que, en el cas d'aquest últim, s'ha heredat part de la gramàtica i de les classes que permeten la seva compilació.

IF

Com en pràcticament tots els llenguatges imperatius, el nostre llenguatge, per controlar el flux d'execució i decidir en funció de l'estat del programa, hi ha la sentència "if". Es veu de forma intuïtiva com funciona. Després del "if" s'avalua una expressió i, si és certa, s'executa la llista d'instruccions que el segueixen. Si és falsa, o bé es passa a avaluar la condició del "elif" (igual que Python), si n'hi ha, o bé es passa a executar el cos d'instruccions del "else", si n'hi ha, o bé es continua amb l'execució de les instruccions que hi ha a continuació del bloc del "if".

Loops

El nostre llenguatge disposa de dos tipus diferents de bucles: el while i el for.

Pel que fa al "while", la manera d'escriure'l és similar a altres llenguatges. Després de la paraula clau "while" ve una expressió booleana que, si s'avalua a cert, fa que es torni a executar el cos d'instruccions del "while". Un cop finalitzada l'execució de les instruccions associades al "while", es torna a avaluar la seva expressió booleana i així successivament fins que aquesta s'avalua a fals i, llavors, es passa a executar la instrucció que hi ha just a continuació del bloc del "while".

Pel que fa al "for", la manera d'escriure'l pot resultar una mica estranya al principi, ja que és força diferent de com s'escriu en llenguatges com C++ o Java. Recorda més aviat al "for" de Python (per exemple, for i in range(10)). El "for" itera amb una variable sobre una llista de valors. La variable que itera s'indica just abans de la paraula clau "in". A continuació,

s'indica la llista de valors que prendrà tal variable. Aquesta llista de valors es pot escriure de tres maneres:

- [n]: genera una llista fictícia de 0 a n-1, ambdós inclosos.
- [n1..n2]: genera una llista fictícia de n1 a n2-1, ambdós inclosos.
- [n1..n2..n3]: genera una llista fictícia de n1 a n2-1 amb increments entre dos elements consecutius de la llista de valor n3.

Per exemple, la línia de codi "for i in [10..20..2]" faria que "i" prengués els valors 10, 12, 14, 16, 18.

Assignacions

Com en qualsevol llenguatge imperatiu conegut, el nostre llenguatge també permet assignar valors a variables mitjançant l'operador "=". A més a més, permet assignar valors a camps interns d'una variable. Això de "camps interns d'una variable" vol dir que podem modificar el valor de l'atribut color, per exemple, d'una figura geomètrica. És a dir, en una variable "C" podem tenir assignat una figura geomètrica (un cercle). Més endavant, volem assignar un color diferent a aquest cercle. Per fer-ho, podem assignar un nou valor a l'atribut "color" del cercle "C" mantenint constants tots els altres atributs.

Crides a funcions

Per poder tenir la potència i claredat de codi dels llenguatges estructurats, fa falta poder definir funcions (o subrutines). Això permet encapsular el codi per guanyar llegibilitat i poder-lo reutilitzar.

Per cridar una funció (suposant que està definida) el protocol que se segueix és el típic en la immensa majoria de llenguatges. S'escriu el nom de la funció i entre parèntesi s'escriuen els paràmetres que se li passen (poden ser 0 o molts). Llavors la funció s'executa i acaba retornant algun valor. Si la funció acaba sense arribar a una sentència "return", pot generar-se un error d'execució, ja que la crida a la funció normalment s'utilitzar en una assignació o en una expressió d'un if, while… i no té sentit que retorni un valor nul. Per això és responsabilitat del programador que s'asseguri que el flux d'execució d'una funció fa que sempre arribi a una sentència del tipus "return" si s'ha d'utilitzar el valor que retorna en algun lloc.

L'únic cas que té sentit cridar una funció que no retorni cap valor és perquè la funció cridada fa una sèrie de càlculs i fa "print" d'algun d'ells.

Print

Aquesta instrucció, tal com el seu nom indica, serveix per mostrar per pantalla el valor de l'objecte o la variable que el segueix.

Run

Aquesta instrucció, específica d'aquest llenguatge, serveix per traduir a SVG una animació determinada que s'hagi creat durant l'execució del programa. Aquesta sentència fa que s'acabi el programa, ja que significa que ja s'ha arribat a crear l'animació desitjada, que és l'objectiu principal del llenguatge.

3.2.2 Part SVG

En aquesta part s'explicaran les característiques del llenguatge que permeten que pugui generar codi d'animacions en SVG.

Objecte

El llenguatge permet crear diversos tipus d'objectes que es visualitzaran per pantalla quan s'executin amb el mètode "run". Els objectes permesos són: CIRCLE, POLYGON, POLYLINE, TRIANGLE, PATH. Cada objecte té definits per defecte tots els atributs necessaris per tal que es pugui traduir a SVG correctament. No obstant això, l'usuari, a l'hora de definir un nou objecte, pot indicar el valor dels seus atributs si el valor que tenen per defecte no el satisfà. Per exemple, circle1 = Circle [color="0:0:128"];. També es poden modificar els atributs d'un objecte un cop ja està definit. Per exemple, per al cercle anterior, si l'usuari no està content amb el valor que té l'atribut color, el pot canviar mitjançant la sentència circle1.color = "0:255:0";

Pack d'objectes

Per tal d'agrupar un conjunt d'objectes i fer que es comportin com un únic objecte s'ha creat l'opció de definir un pack d'objectes. Per exemple, si es vol definir una caseta, es pot crear un quadrat a sota i un triangle a dalt. Per tractar-ho com una única figura geomètrica es pot definir un pack d'objectes amb el quadrat i el triangle anteriors de la següent manera: caseta = {quadrat1, triangle1};

Moviment

Serveix per definir, com el seu nom indica, moviments. Els moviments poden ser de diveros tipus: TRANSLATE, ROTATE, SCALE, FOLLOWPATH. La manera per definir moviments és molt similar a la forma emprada pel cas dels objectes. Els moviments tenen una sèrie de paràmetres per defecte que es poden canviar al moment de definir-los o més endavant, mitjançant “nom_moviment.nom_atribut = nou_valor”.

Objecte en moviment

Per aplicar un moviment a un objecte o a un grup d’objectes, s’utilitza l’operador “->” de manera que a la dreta hi ha un moviment i a l’esquerra l’objecte o pack d’objectes als quals es vol aplicar.

Moviments seqüencials

Per encadenar una sèrie de moviments de forma seqüencial s’utilitza l’operador “&”. Si es vol inserir una pausa entre dos moviments, es pot posar com a operant un enter que indica el temps d’espera entre els dos moviment. Per exemple: movSeq = mov1 & 3 & mov2. L’exemple anterior crea un nou moviment encadenant de forma seqüencial mov1, una pausa de 3 segons i mov2.

Moviments paral·lels

Per encadenar una sèrie de moviments de forma paral·lela s’utilitza l’operador “|”. Per exemple: movPar = mov1 | mov2. L’exemple anterior crea un nou moviment a partir de mov1 i mov2 posant-los de forma paral·lela, és a dir, que s’executaran simultàniament.

3.3 Gramàtica

A continuació s’enumeren els canvis més signficatius respecte ASL de la gramàtica del llenguatge.

Addició del for

Aquesta gramàtica permet escriure bucles amb al paraula clau “for”.

Extensió dels if’s

A diferència de ASL, aquest llenguatge permet l'estructura del tipus if, elif, else.

Operadors específics

La definició d'una expressió és molt més complexa que en ASL, ja que s'han afegit una sèrie d'operadors específics per tal de permetre operacions entre objectes i moviments. S'ha hagut de redefinir la jerarquia d'operadors. Els operadors, de menys a més preferència, són: or, and, not, comparadors, "->", "|", "&", + -, * /, - (com a operador unari), parèntesi.

Constructors específics

Per poder construir les figures geomètriques i els moviments que se'ls aplicaran per traduir-ho a SVG, s'han hagut d'incorporar nous constructors específics per aquests.

3.4 Semàntica

No hi ha hagut canvis significatius respecte ASL pel que fa a la semàntica. Tots els operadors que hi havia en ASL tenen el mateix significat en el nou llenguatge. El sistema de tipus i la comprovació d'aquests també s'hereda de ASL.

4. Jocs de proves

Al primer joc de proves s'avalua només les característiques que no involucren la part SVG. Això serveix per poder valorar la correctesa de la part imperativa estàndard del lenguatge per, després, poder afegir de forma incremental les instruccions que permeten la generació de codi SVG.

```
def factorial (x)
	res = 1;
	while x > 1 do
		res = res * x;
		x = x - 1;
	endwhile
	return res;
endfunc

def factorial_rec (x)
	if x > 1 then
		return x * factorial_rec(x-1);
```

```
	else
		return 1;
	endif
endfunc

def main ()
	print "factorial_iteratiu";
	print factorial(5);
	print "factorial_recursiu";
	print factorial_rec(5);

endfunc
```

En segon primer joc de proves podem veure de forma senzilla un global de les funcionalitats que aquest llenguatge proporciona. Aquí es comprova com es composen dos moviments, com s'assignen a un objecte, com funciona un bucle for, com dins del bucle es creen altres objectes i com es composen les diferents animacions de cada objecte de forma seqüencial.

```
def main ()
	triangle = Triangle [centerX=100, centerY=50, colorFill="0:255:0", colorLine ="0:0:255", radius=50.0];
	escalat = Scale [factor=2, dur=1];
	rotacio = Rotate [w=30, dur=2];
	mov = escalat | rotacio;
	objEnMov = mov->triangle;
	for i in [2] do
		circle = Circle [centerX=i*70 + 500, centerY=800, radius=50.0];
		objEnMov = objEnMov & (escalat->circle);
	endfor
	run 2000,2000,objEnMov;
endfunc
```

El tercer joc de proves és una extensió del segon. Per tal de comprova el funcionament del llenguatge de forma incremental, s'ha decidit anar afegint complexitat en els diferents jocs de proves. Aquest incorpora, a més a més de la rotació i l'escalat, la translació. La composició de moviments és més complexa, ja que inclou moviments en sèrie i paral·lel en la definició de "mov". També s'afegeix una tercera esfera.

```
def main ()
	triangle = Triangle [centerX=300, centerY=300, colorFill="0:255:0", colorLine ="0:0:255", radius=50.0];
	escalat = Scale [factor=2, dur=1];
	rotacio = Rotate [w=30, dur=2];
	translacio = Translate [dur=2, x=-300, y=-300];
	mov = (escalat | rotacio) & escalat & rotacio & translacio & escalat;
	objEnMov = mov->triangle;
```

```
        circle = Circle [centerX=100 + 500, centerY=800, radius=50.0];
        objEnMov = objEnMov & (mov->circle);
        for i in [2] do
                circle = Circle [centerX=i*70 + 500, centerY=500, radius=50.0];
                objEnMov = objEnMov & (escalat->circle);
        endfor
        run 2000,2000,objEnMov;
endfunc
```

En aquest quart joc de proves simplement es comprova l'efecte d'aplicar un escalat amb un nombre positiu, primer, amb un altre escalat consecutiu amb un nombre negatiu. D'aquesta manera s'observa com el triangle col·lapsa en un punt i després passa a créixer altre cop però de forma inversa.

```
def main ()
  x = -4;
        triangle = Triangle [centerX=50, centerY=50, colorFill="0:255:0", colorLine ="0:0:255", radius=50.0];
        escalat1 = Scale [factor=5, dur=2];
        escalat2 = Scale [factor=-5, dur=2];
        escalat = escalat1 & escalat2;
        objEnMov = escalat->triangle;
        run 2000,2000,objEnMov;
endfunc
```

5. Possibles extensions

Per tal de facilitar la vida al programador que en un futur utilitzi el llenguatge descrit en aquest document, s'han pensat diverses millores que es podrien implementar. El motiu pel qual no s'han arribat a implementar les extensions que es detallen a continuació ha estat la complexitat inherent que tenen i el temps que es requereix per implementar-les.

Arrays

El llenguatge podria suportar arrays i així poder incrementar la seva capacitat i flexibilitat de càlcul al tenir aquestes estructures de dades. La millora es notaria tant pel que fa a la part més imperativa del llenguatge com a la que està més lligada a SVG, ja que permetria manejar dades de forma més còmoda.

Poder passar paràmetres per referència

Ja que, en principi, no ha de ser un llenguatge destinat a realitzar càlculs costosos amb grans estructures de dades, tots els paràmetres es passen per còpia. No obstant això, seria bo poder tenir l'opció de passar paràmetres a les funcions per referència, no tansols per

guanyar eficiència al no haver de copiar dades de gran magnitud, sinó també per poder modificar dins la funció tals variables d'entrada i poder notar el canvi fora de la funció.

Accedir als objectes d'un pack d'objectes

Tal com està ara el llenguatge, no es permet l'accés als objectes que conformen un pack d'objectes ni tampoc als moviments que conformen un moviment en paral·lel. Seria bo poder accedir a tals dades per poder-les modificar de la manera que l'usuari cregués més adient.

Eliminar objectes d'un pack d'objectes

Similar a l'apartat anterior. Tal i com està ara el llenguatge, no es poden eliminar els objectes que conformen un pack d'objectes, ni tampoc els moviments que conformen un moviment més complex.

Afegir més varietat de figures predefinides

Per facilitar la vida a l'usuari, es podrien predefinir les figures geomètriques més habituals (ja n'hi ha alguna, com triangle) perquè estiguessin llestes per utilitzar-les de forma ràpida. Amb la polilínia ja es poden definir totes les figures geomètriques que un vulgui, però algunes són laborioses de fer i, la millora mencionada en aquest apartat, estalviaria part d'aquest cost.

6. Bibliografia

- Mozilla Developer [en línia] [Consulta: 2 abril 2016]. Disponible a: <https://developer.mozilla.org/en-US/docs/Web/SVG/Element>
- Tutsplus tutorials [en línia] [Consulta: 5 abril 2016]. Disponible a: <http://webdesign.tutsplus.com/tutorials/how-to-use-animatetransform-for-inline-svg-animation--cms-22296>
- W3 tutorials [en línia] [Consulta: 5 abril 2016]. Disponible a: <https://www.w3.org/TR/SVG/interact.html>
- Sitepoint [en línia] [Consulta: 5 abril 2016]. Disponible a: <http://www.sitepoint.com/svg-path-element/>

7. Annex

En aquest annex es mostren tots els tipus d'objectes i moviments que accepta el programa amb els seus corresponents atributs. Pel que fa als atributs, es mostren els que el programa assigna per defecte. D'aquesta manera l'usuari pot saber el seu valor i conèixer el seu tipus.

Circle	Polygon*	Polyline*	Triangle	Path*
colorLine="0:0:0"	colorLine="0:0:0"	colorLine="0:0:0"	colorLine="0:0:0"	colorLine="0:0:0"
colorFill="256:0:0"	colorFill="256:0:0"	colorFill="256:0:0"	colorFill="256:0:0"	colorFill="256:0:0"
lineWidth=5	lineWidth=5	lineWidth=5	lineWidth=5	lineWidth=5
opacity=1.0f	opacity=1.0f	opacity=1.0f	opacity=1.0f	opacity=1.0f
centerX=0	centerX=0	centerX=0	centerX=0	centerX=0
centerY=0	centerY=0	centerY=0	centerY=0	centerY=0
radius=1.0f			radius=1.0f	

Translate	Rotate	Scale	Followpath*
begin=0	begin=0	begin=0	begin=0
dur=20	dur=20	dur=20	dur=20
fill="freeze"	fill="freeze"	fill="freeze"	fill="freeze"
calcMode="spline"	calcMode="spline"	calcMode="spline"	calcMode="spline"
x=0	w=1	factor=1f	
y=0			