Graphical Animation of Transition Systems Behaviour

Autor:
Albert Cortada Ballester

Data de la defensa:
25 d'Abril de 2014

Director:
Albert Rubio Gimeno

Departament:
LSI

Titulació:
Grau en Informàtica

Especialitat:
Computació

Centre:
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Universitat:
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BarcelonaTech

Índex

1 Resum del projecte

1.1 Català

Aquest document especifica i justifica l'existència del Treball Final de Grau del Grau en Informàtica, defensat per Albert Cortada Ballester i que té com a director a Albert Rubio Gimeno. Aquest projecte es presenta el dia 25 d'Abril de 2014.

Aquest treball presenta una aplicació gràfica d'escriptori, que s'encarrega de dues tasques, sobre les quals gira el seu desenvolupament. Per una banda, permet representar sistemes de transicions en forma de graf. D'altra banda, permet simular una execució d'aquests sistemes, tot mostrant de forma visual i intuitiva el camí d'execució escollit en cada moment.

Aquesta execució s'ha de poder realitzar tant des d'un punt que considerem com a inicial, com des de qualsevol altre punt, sempre que es disposi de les dades d'entrada necessàries.

El fet de poder executar un programa des de diversos punts permet provar el seu comportament en parts concretes del codi, i també permet provar diferents combinacions de paràmetres, per observar quin resultat obtenim, o si hi ha casos límits que n'impedeixen el correcte funcionament.

El que es vol aconseguir amb la combinació d'aquests dos objectius és complementar les eines existents d'anàlisi de programes. Aquest projecte pretén crear una eina intuitiva que sigui fàcil d'usar i permeti visualitzar errors detectats mitjançant mètodes formals.

Per això, aquest projecte pot servir com a eina de recerca, ja que es pot integrar dins d'eines que comprovin problemes com els descrits a continuació:

- Problema de terminació (3): Donat un programa P, i una entrada legal X d'aquest programa, saber si P acaba la seva execució amb X com a entrada. Aquest va ser el primer problema que es va descobrir que era indecidible, és a dir, que no existeix cap algorisme que sigui capaç de respondre en temps finit la pregunta de si un programa acabarà la seva execució o no.
- Problema d'accessibilitat: Donat un punt del codi, descobrir si aquest

és abastable durant l'execució del programa. Aquest problema també és indecidible, ja que entre altres components, trobar una resposta d'accessibilitat necessita resoldre el problema de terminació.

Per últim, una altra finalitat que pot assumir aquest projecte és la d'eina didàctica, ja que es pot fer servir per professors que hagin d'ensenyar programació als seus alumnes. Amb aquesta eina els alumnes podrien entendre millor com funcionen certs algorismes, ja que poden observar en cada moment quin és el camí d'execució triat i l'estat del sistema posterior a cada transició. Aquest mètode creiem que és un bon complement a una possible demostració més formal del seu funcionament.

1.2 Castellà

Este documento especifica y justifica la existencia del Trabajo Final de Grado del Grado en Informática, defendido por Albert Cortada Ballester i que tiene como director a Albert Rubio Gimeno. Este proyecto se presenta el dia 25 de Abril de 2014.

Este trabajo presenta una aplicación gráfica de escritorio, que se encarga de dos tareas, sobre las cuales gira su desarrollo. Por una parte, permite representar sistemas de transiciones en forma de grafo. Por otra, permite simular una ejecución de estos sistemas, mostrando de forma visual e intuitiva el camino de ejecución escogido en cada momento.

Esta ejecución se debe poder realizar tanto desde un punt que consideremos como inicial, como desde cualquier otro punto, siempre que se dispongan de los datos de entrada precisos.

El hecho de poder ejecutar un programa desde diversos puntos permite probar su funcionamiento en partes concretas del código, i también permite probar diferentes combinaciones de parámetros, para observar qué resultado obtenemos, o si hay casos límite que impiden su correcto funcionamiento.

Lo que se quiere conseguir con la combinación de estos dos objetivos es complementar las herramientas existentes que analizan programas. Este proyecto pretende crear una herramienta que sea intuitiva y fácil de usar y permita visualizar errores detectados mediante métodos formales.

Por eso, este proyecto puede servir como herramienta de investigación, ya que se puede integrar dentro de herramientas que comprueben problemas como los descritos a continuación:

- Problema de terminación(3): Dado un programa P, y una entrada legal X de este programa, saber si P acaba su ejecución con X como entrada. Este fue el primer problema que se descubrió que era indecidible, es decir, que no existe ningún algoritmo que sea capaz de responder en tiempo finito la pregunta de si un programa acabará su ejecución o no.
- Problema de accesibilidad: Dado un punto del código, descubrir si este resulta alcanzable durante la ejecución del programa. Este problema también es indecidible, ya que entre otros componentes, encontrar una

respuesta de accesibilidad necesita resolver el problema de la terminación.

Por último, otra finalidad que puede asumir este proyecto es la de herramienta didáctica, ya que puede ser usada por profesores que tengan que enseñar programación a sus alumnos. Con esta herramienta los alumnos podrían entender mejor cómo funcionan ciertos algoritmos, ya que pueden observar en cada momento cuál es el camino elegido i el estado del sistema posterior a cada transición. Creemos que este método es un buen complemento a una posible demostración más formal de su funcionamiento.

1.3 Anglès

This document specifies and justifies the existence of this "Treball Final de Grau" from the "Grau en Informàtica" titulation, argued by Albert Cortada Ballester, which has as a director Albert Rubio Gimeno. This project is presented on April 25th, 2014.

This paper presents a graphic desktop application, which is responsible for two tasks, which are the center of its development. On one hand, it allows to represent transition systems in the form of graphs. On the other hand, it allows the simulation of these systems, showing visually and intuitively the chosen path of execution at any time.

This execution has to allow the possibility of beginning from a predetermined starting point, or from any other point of the program, whenever there is the required initial data.

By allowing this option, it allows to prove a program behaviour in specific parts of the code, and also allows to try different parameter combinations in order to analyze the results, or to detect some strange cases where some parameters cause the program to malfunction.

What we want to achieve with the combination of these two tasks is to complement some existing tools responsible of program analysis. This project aims to create an intuitive tool which must be easy to use and allows to visualize errors detected by formal methods.

For this reason, this project can be used as a research tool. It can be integrated into other tools which analyze these problems:

- Halting problem(3): Given a program P, and a legal input X of this program, we want to know if P finishes its execution having X as an input. This was the first undecidable problem discovered ever. It means that there can't be any algorithm able to solve this problem in a finite amount of time.
- Reachability problem: Given a point in some code, we want to know if we can reach this point during an execution. This problem is also undecidable, because among other components, finding an answer to this problem would also need to solve the halting problem.

Finally, there is another use we can find for this project. That is as a teaching tool, because it can be used by professors who have to teach programming skills to their students. With the aid of this tool, students could understand better how certain algorithms work, as they can see at every moment which is the chosen execution path and the state of the system after every transition. We believe this method to be a good complement to a possible formal demostration of the algorithm behaviour.

2 Visió general del projecte

2.1 Motivacions

Com s'ha comentat abans, aquest projecte treballa amb sistemes de transicions com a dades d'entrada. Els sistemes de transicions són abstraccions de llenguatges de programació àmpliament utilitzats en l'àmbit de l'anàlisi de programes.

L'anàlisi de programes (2) és el procés d'analitzar automàticament el comportament dels programes informàtics. En concret, els dos problemes especificats anteriorment són motiu d'estudi en l'anàlisi de programes, principalment amb el propòsit de determinar-ne la correctesa.

Com s'ha comentat anteriorment, els problemes de terminació i accessibilitat pertanyen a la classe de problemes indecidibles,

Tot i que els problemes de terminació i accessibilitat pertanyen a la classe de problemes indecidibles, existeixen heurístics que ens permeten obtenir aproximacions quan es compleixen una sèrie de condicions.

Per tractar aquest tipus de problemes, el que es sol fer és abstreure el llenguatge de programació concret amb què es treballa, i substituir-lo per un sistema de transicions.

Un sistema de transicions $S = (\bar{v}, \mathcal{L}, \Theta, \mathcal{T})$ consisteix en una tupla de variables $\bar{v}$, un conjunt de localitats $\mathcal{L}$, un diccionari Θ que relaciona localitats amb fòrmules que caracteritzen els valors inicials de les variables, i un conjunt de transicions $\mathcal{T}$.

Una localitat $l \in \mathcal{L}$ és un punt del codi a partir del qual existeixen múltiples camins d'execució. En l'exemple de la figura 1 cada localitat ve representada en forma de node en el graf. Cadascuna d'aquestes localitats es corresponen a un dels bucles niats del programa escrit en llenguatge imperatiu que teniem al principi.

Una transició $\tau \in \mathcal{T}$ és una tripleta (l, l', ρ), on $l, l' \in \mathcal{L}$ i representen:

- Localitat de sortida l
- Localitat d'arribada l'

- Condició de transició ρ: és una fórmula sobre les variables del programa $\bar{v}$ i les seves versions primades $\bar{v}'$.

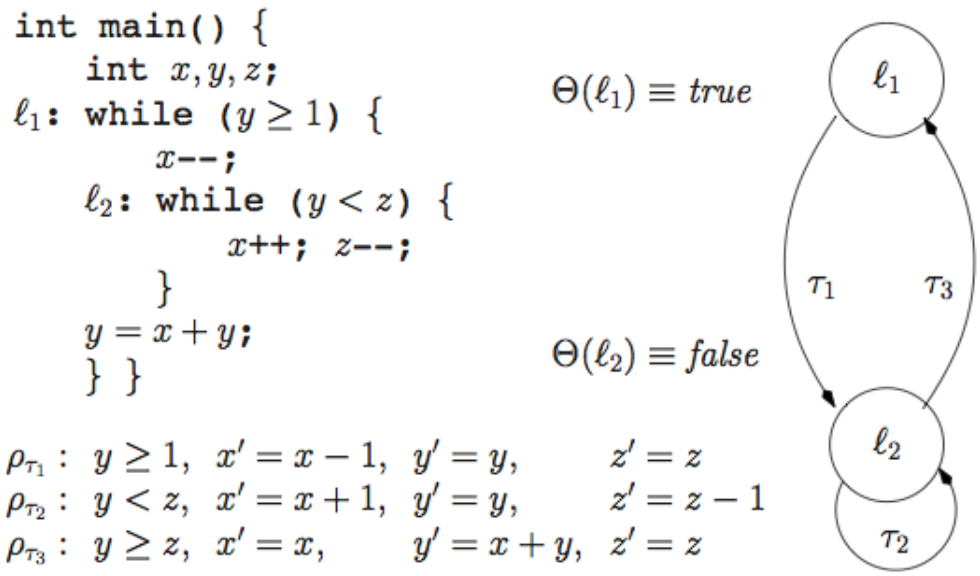

Figura 1: Exemple de programa escrit en llenguatge imperatiu i el seu corresponent sistema de transicions

Aquestes eines es poden beneficiar molt d'un programa que permeti visualitzar els seus resultats, funcionant amb el sistema de transicions com a llenguatge d'entrada.

En concret, en aquest treball es fa servir el sistema de transicions descrit pel projecte CppInv(1).

CppInv es diferencia d'altres eines similars en la seva capacitat de tractar amb estructures de control arbitràries. En l'exemple de la figura 1, podem observar dos bucles niats.

2.2 Objectius

Per tal de crear una aplicació que pugui ser útil en l'àmbit de l'anàlisi de programes, s'han definit els següents objectius:

- Representar gràficament un sistema de transicions. Consisteix en transformar un sistema de transicions en un graf que es pugui visualitzar per pantalla. D'aquesta manera, és molt més fàcil veure com es relacionen entre si diferents localitats. És important poder visualitzar sistemes de transicions amb moltes localitats o moltes transicions, ja que alguns dels programes amb els que es treballen poden tenir una mida molt gran.
- Executar un sistema de transicions des de l'inici. Consisteix en crear un intèrpret del sistema de transicions, que sigui capaç d'evaluar les condicions com a certes o falses i actualitzar l'estat del sistema amb una combinació de valors que compleixi la condició de la transició seleccionada. En alguns casos s'haurà de seleccionar una transició, ja que hi poden haver múltiples entre dues localitats que s'avaluin com a certes per a la mateixa combinació d'assignacions de variables.
- Executar un sistema de transicions a partir d'una localitat qualsevol, amb les dades d'entrada que siguin necessàries. Consisteix en ampliar l'objectiu anterior, per tal que no només es pugui interpretar el sistema de transicions des d'una localitat que considerem com a inicial, sino que l'usuari d'aquesta eina pugui escollir des de quina vol començar l'execució, i que pugui fixar valors per algunes variables, substituint aquells valors que resulten d'una execució normal del sistema. D'aquesta manera, es permet a l'usuari verificar aquella part del sistema que l'interessa.
- Mostrar animacions sobre l'execució del sistema de transicions. No només volem escollir un camí d'execució i calcular l'estat del sistema en finalitzar una transició, sino que també ens interessa mostrar això no com un canvi instantàni, sino com un procès que consti d'una petita animació, per tal que l'usuari pugui realitzar un seguiment de l'execució del sistema.
- Realitzar instantànies del sistema de transicions en qualsevol moment. Consisteix en oferir a l'usuari una opció que li permeti realitzar fotografies del graf durant l'execució. D'aquesta manera, obté una font de material que pot incloure en demostracions sobre algun anàlisi que realitzi sobre el programa.

2.3 Obstacles

El major obstacle que s'ha trobat durant la realització d'aquest projecte té a veure amb l'ús de la llibreria gràfica escollida.

Per una banda, s'han subestimat les dificultats que pot arribar a comportar la creació d'una interfície gràfica, sobretot quan es fa servir una eina que no havíem emprat mai abans.

Per altra banda, la documentació disponible d'aquesta llibreria no és suficient, ja que si bé s'expliquen la majoria de funcions principals, hi ha pocs exemples, i també ha estat difícil trobar la solució a certs problemes degut a la poca quantitat d'usuaris que treballen amb ella.

Aquests han estat problemes no previstos a la planificació inicial.

2.4 Context

Aquest projecte s'enmarca dins el context de l'anàlisi de programes. No intenta implementar algorismes heurístics per resoldre problemes com la terminació o l'accessibilitat, sino que pretèn ser una eina complementària, que permeti mostrar els resultats obtinguts amb eines d'aquest tipus.

Com es veurà a continuació, existeixen poques alternatives que presentin el tipus de funcionalitat que implementa aquest projecte.

2.5 Impacte mediambiental

L'impacte mediambiental d'aquest projecte és limitat, ja que es tracta d'un projecte de desenvolupament de software, i l'única contaminació produïda és la relativa al consum energètic dels ordinadors amb els quals s'ha treballat.

2.6 Alternatives

Per verificar la utilitat real d'un treball com el proposat, es van investigar una sèrie de projectes semblants. A continuació es detallen els resultats que s'han trobat.

2.6.1 LTSA(4)

LTSA és una eina de verificació per sistemes concurrents. Mecànicament, comprova que l'especificació d'un sistema concurrent satisfà les propietats requerides del seu comportament. A més, LTSA suporta l'especificació d'animacions per tal de facilitar l'exploració interactiva del comportament del sistema.

Aquest projecte té en comú amb LTSA la creació d'una eina de visualització del comportament d'un sistema de transicions, encara que el plantejament de LTSA és diferent: transforma un sistema de transicions en una màquina d'estats, on cada node representa un estat i on les transicions són molt més simples que les que ha de tractar aquest projecte, ja que només treballa amb condicions booleanes.

Trobem un exemple del programa d'animació a les figures 2 i 3

La figura 2 mostra un editor de variables. Com que aquestes són de tipus booleà, deixa seleccionar el seu valor en cada moment de l'execució mitjançant un element "checkbox". En el nostre projecte, donat que treballem amb nombres enters, necessitem una caixa de text per indicar el valor que volem que adquireixi una variable.

La figura 3 es troba associada a 2, i és la representació visual d'un sistema de transicions. Com es pot veure, les condicions de pas per les arestes són molt simples, ja que es basen en les variables booleanes que acabem de comentar, i només apareix una a cada condició, sense l'intervenció de cap més operador.

2.6.2 LTSView(5)

LTSView és una eina de visualització de sistemes de transicions. Modela els sistemes en 3D fent servir estructures anomenades arbres cònics, i ofereix eines per desplaçar-se sobre la visualització, com zoom i panning. A més, també ofereix com a opció realitzar una simulació d'execució tant des de l'estat inicial com des de qualsevol altre node.

Aquest projecte s'assembla molt al que es desenvolupa en el nostre treball. Hi ha dues diferències, importants, una en la visualització i una segona que afecta a la simulació de l'execució.

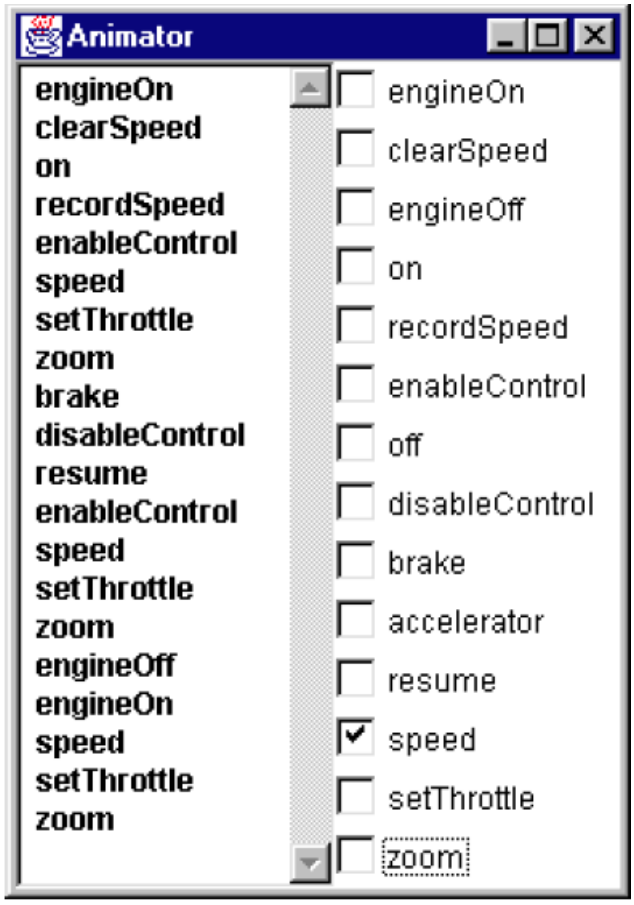

Figura 2: Editor de variables de LTSA

Pel que fa a la visualització, en el nostre cas, aquesta és en dues dimensions en comptes de tres.

En el nostre projecte no cal una visualització 3D. El motiu amb el que es defensa aquest tipus de vista en LTSView és la cerca de la propietat de simetria. Aquesta propietat vol que un conjunt d'estats que es comportin de manera similar també tinguin una estructura similar a la visualització. A la figura 4 es pot veure un exemple d'aquesta propietat.

Per treure profit d'aquesta propietat, el sistema de transicions hauria de ser modelat en forma de mòduls o s'hauria de basar en un sistema distribuït. No és el nostre cas, ja que treballem amb un únic àmbit, sense funcions ni

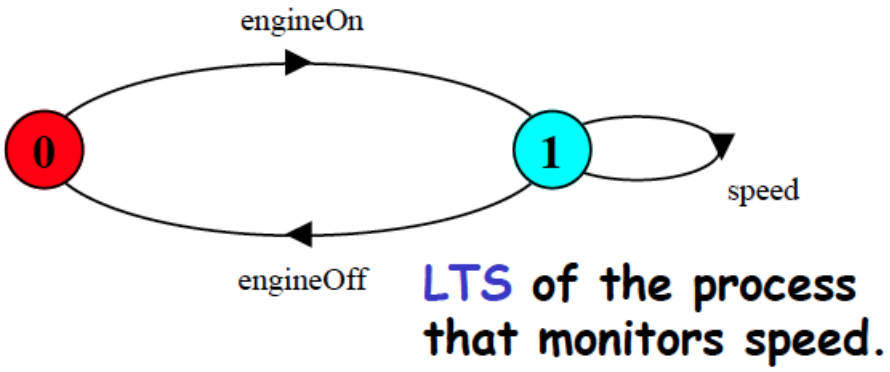

Figura 3: Representació gràfica generada per LTSA

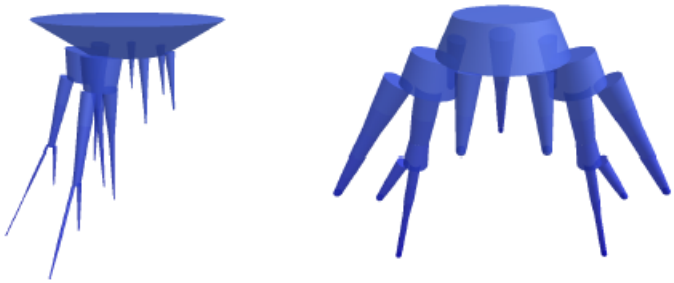

Figura 4: Exemple de sistema de transicions modelat sense i amb simetria

mòduls, i tampoc treballem amb sistemes distribuïts.

Pel que fa a la simulació de l'execució, LTSView treballa amb sistemes de transicions que anomena "bàsics". Nosaltres treballem amb sistemes de transicions etiquetats, que són una extensió dels primers, i permeten l'expressió de condicions més complexes.

2.6.3 Conclussions sobre les alternatives

Encara que existeixen moltes eines que es dediquen a l'anàlisi de programes, no existeixen prous alternatives de cara a un programa que permeti visualitzar un sistema de transicions. Tot i existir-ne algunes, el fet que es concentrin

en un tipus de problema, com és el de sistemes concurrents o distribuïts, o en altres, com terminació i accessibilitat, fa que calgui dissenyar diferents llenguatges que descriguin sistemes de transicions, i per tant, és poc probable que existeixi una aplicació gràfica capaç de representar tots aquests formats.

Per tant, creiem que hi ha una necessitat de crear una eina com la que proposa aquest projecte.

3 Planificació temporal del projecte

En aquesta secció discutim la planificació temporal del projecte, quines tasques s'han identificat, una llista de les accions que s'han pres en cas d'haver-hi retards i com ha canviat aquesta planificació respecte la versió inicial.

3.1 Descripció de les tasques

Es van identificar les tasques següents:

- Investigació: calia estudiar com funciona la llibreria gràfica escollida per tal de conèixer quin tipus d'animacions es poden fer. En particular, volíem conèixer si es poden realitzar ampliacions d'imatge per treballar amb grafs de gran mida. També va caldre investigar sobre l'ús de SMT-Solvers per tal de simular una execució del programa original.
- Crear la representació en forma de graf: necessitem traduir un codi escrit en un llenguatge que defineix "locations" i transicions a un que consta de nodes i arcs.
- Crear la interacció amb l'usuari: Volem permetre a l'usuari modificar el valor de variables i seleccionar el node des d'on vol començar l'execució. A més, volem que pugui desplaçar-se per grafs de mida gran.
- Executar el graf des de qualsevol "location": volem ser capaços d'executar el graf no només des del principi, sinó també des de qualsevol altre punt, donades les dades necessàries.
- Testeig: provar que no hi hagi errors en cap d'aquests processos.
- Produïr imatges: volem mostrar les transformacions i execucions del graf en forma de seqüències d'imatges.

3.2 Retards i solucions

La planificació original es va fer amb l'objectiu d'acabar el projecte entre el gener o el febrer. Això no ha estat possible degut a retards en algunes de les tasques.

Una mesura que es va adoptar va ser allargar la duració del projecte fins a l'abril, afegint així dos mesos addicionals que han servit per acabar el projecte i redactar la memòria.

La part de visualització del graf es va allargar més del que ens hauria agradat: ens interessava més aquest projecte per interpretar un programa, però la part de visualització era necessària, i va requerir moltes tasques més de les previstes.

Donat que volíem treballar amb grafs de mida gran, vam haver d'implementar les funcions de zoom-in i zoom-out, a més d'una càmara que anès seguint el punt actual de l'execució.

Com que aquestes tasques van resultar més llargues del que havíem previst, vam descartar algunes de les tasques que haviem acordat inicialment. Concretament, les de detectar accessibilitat i terminació de programes. Aquestes tasques no eren del tot imprescindibles, perquè els mecanismes per detectar aquests problemes ja estaven implementats, i l'abast d'aquest projecte hauria estat només el d'establir un pont per comunicar-nos amb les eines que avaluen aquestes propietats.

També s'han modificat algunes tasques: ara, en comptes de produir videos sobre l'aplicació, permetem a l'usuari realitzar fotografies. Aquest canvi no va ser tant degut a la planificació temporal com a descobrir que la llibreria escollida tampoc permetia de forma fàcil realitzar vídeos sobre l'aplicació, però si que donava facilitats per realitzar fotografies en qualsevol moment de l'execució.

Per últim, es va modificar la tasca de crear la interacció amb l'usuari: ara s'ha unit amb una tasca que teníem anteriorment, que era permetre a l'usuari aplicar transformacions sobre el graf, ja que en comptes de modificar nodes i arestes, vam considerar més important que l'usuari pogués modificar el node inicial de l'execució, entre altres tasques.

3.3 Diagrama de Gantt

La figures 5 i 6 mostren el diagrama de Gantt que representa la planificació final del projecte.

Nombre	Duración	Inicio	Fin	Predecesora
Investigació	2m	01/07/2013	23/08/2013	
Crear la representació en forma de graf	2m	26/08/2013	18/10/2013	1
Crear la interacció amb l'usuari	3m	21/10/2013	10/01/2014	2
Executar el graf des de qualsevol "location"	1.85m	13/01/2014	04/03/2014	1,2,3
Testeig	5m	30/10/2013	18/03/2014	1,2
Produïr imatges	2s	05/03/2014	18/03/2014	1,2,3,4
Documentació	1.15m	19/03/2014	18/04/2014	1,2,3,4,5

Figura 5: Tasques del diagrama de Gantt al final del projecte

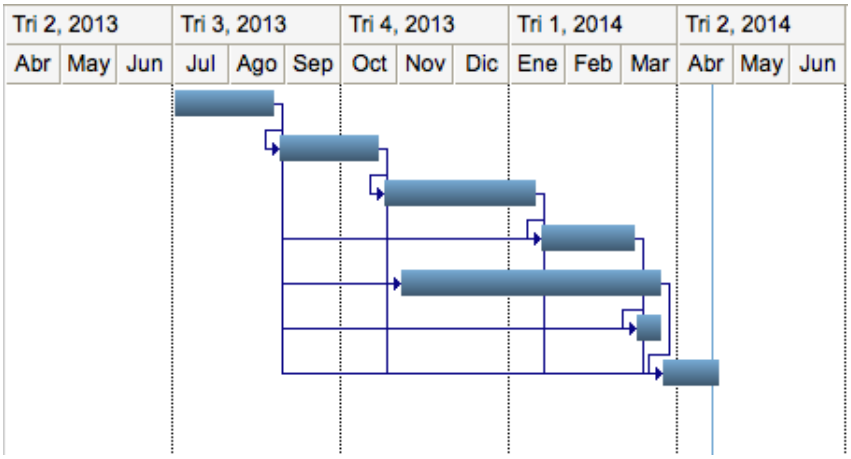

Figura 6: Taula de temps del diagrama de Gantt al final del projecte

3.4 Variació respecte a la planificació original

La figura 7 mostra el diagrama de Gantt que es va crear a l'inici del projecte.

El canvi més important és la durada del projecte. Inicialment, el projecte havia de durar fins al desembre, amb la posterior confecció de la memòria. Malhauradament, aquesta planificació va ser massa optimista respecte a la durada d'algunes de les tasques, en especial aquelles que tractaven de la part de visualització del projecte.

A més, a la planificació final hi ha algunes tasques menys, que es van eliminar

		Nombre	Duraci□n	Inicio	Fin	Predecesoras
1		Investigation	2m	01/07/2013	23/08/2013	
2		Create graph representation	2s	26/08/2013	06/09/2013	1
3		Create user interaction with the graph	3m	09/09/2013	29/11/2013	2
4		Detect code reachability	1m	23/09/2013	18/10/2013	1,5
5		Detect non-termination	2s	09/09/2013	20/09/2013	2
6		Execute a graph from any location	1m	21/10/2013	15/11/2013	1,4
7		Allow the user to apply some correct transformations to the transition s	3s	18/11/2013	06/12/2013	4,5,6
8		Testing	4m	26/08/2013	13/12/2013	1
9		Produce videos	1s	16/12/2013	20/12/2013	1,3,8

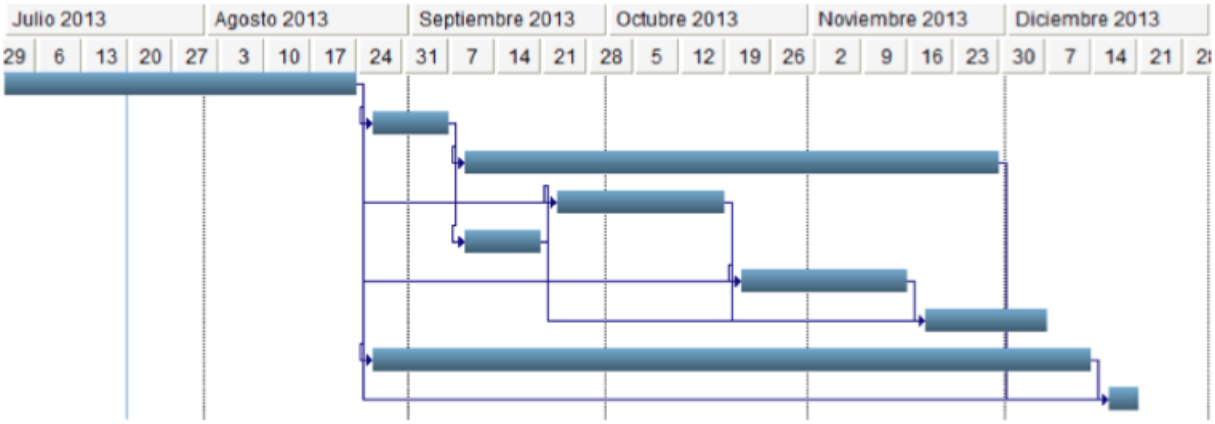

Figura 7: Diagrama de Gantt a l'inici del projecte

ja que no eren essencials pels objectius principals del projecte.

Finalment, s'ha inclòs com a tasca l'elaboració de la memòria, ja que probablement s'hauria d'haver inclòs també a la planificació inicial.

4 Planificació econòmica del projecte

En aquesta secció, detallarem els recursos econòmics que s'han identificat com a necessaris per realitzar el projecte, així com una estimació dels costos i una argumentació sobre la viabilitat del projecte.

4.1 Identificació dels recursos

Per realitzar aquest projecte, es necessiten els següents recursos:

- Primer, ens calen recursos humans. És essencial tenir algun programador, i també una persona ha de crear jocs de prova que ens permetin comprovar si hi ha errors o alguna funció no es comporta segons la seva especificació. Per tal de realitzar una aplicació complerta, ens cal algú que s'encarregui d'elaborar la interfície gràfica, de forma que l'usuari pugui interactuar amb el programa.
- També necessitem recursos materials. Per cada persona treballant en el projecte, cal un ordinador.

4.2 Estimació dels costos

Per tal d'estimar els costos totals, assumirem que treballem en aquest projecte unes 500 hores. Aquesta és una estimació basada en el nombre de crèdits ECTS requerits.

A la taula 1 es pot veure detallat el cost en recursos humans.

Taula 1: Costos dels recursos humans

Recurs	Cost/hora	nº hores de treball	Cost total
Programador	30€	200	6000€
Dissenyador de la interfície	20€	150	3000€
Provador	30€	50	1500€
Investigador	40€	100	4000€
			Total = 14500€

El programador va ser responsable de programar les funcionalitats bàsiques de l'aplicació, com la traducció del llenguatge del sistema de transicions en un graf.

El dissenyador de la interfície va ser el responsable de definir els diferents tipus d'interacció de l'usuari amb la interfície, així com el disseny d'aquesta.

El provador va ser responsable de la tasca de testeig. Es va assegurar que el programa funcionava sense errors.

Finalment, l'investigador és responsable de la primera tasca definida a la planificació, que és la d'investigació. Va haver de trobar altres solucions similars a les que proposem, a més de determinar quines llibreries externes fariem servir.

Taula 2: Costos dels recursos materials

Recurs	Cost base	Cost/hora	nº hores de treball	Cost total
Ordinadors	4x1000€	0.10€	500	4050€
				Total = 4050€

A la taula 2, el cost per hora és una estimació basada en l'electricitat consumida i l'amortització dels materials.

4.3 Viabilitat del projecte

La viabilitat d'aquest projecte és elevada, ja que confiem en que sigui útil com a eina de recerca i de docència. Cal destacar el fet que mentre que existeixen moltes eines que es dediquen a diferents aspectes de l'anàlisi de programes, i moltes d'elles fan servir en major o menor mesura sistemes de transicions, no existeixen prous solucions per visualitzar els resultats obtinguts per aquestes eines, i per tant, creiem que un projecte com aquest és necessari.

4.4 Variació respecte a la planificació original

Encara que la planificació temporal ha variat, la planificació econòmica no s'ha hagut de modificar, ja que l'estimació inicial es va fer prenent com a base el màxim d'hores de què podia constar un projecte d'aquest tipus, i no

tant en una estimació d'hores de dedicació segons la durada en mesos del projecte.

5 Eines externes

En aquesta secció descriurem les eines externes que s'han utilitzat al llarg del projecte i per què van ser escollides.

5.1 Llibreria de gràfics

Durant el procès de cerca d'una llibreria gràfica per aquest projecte, es van comparar aquestes propostes: Prefuse(6), D3(7) i GraphStream(8).

Una de les llibreries gràfiques més conegudes per treballar en Java és Prefuse. Aquesta eina té una gran versatilitat ja que permet altres tipus i estructures de visualització a més de grafs. En el nostre projecte, només ens cal treballar amb grafs, així que tot i valorar aquesta propietat sobre Prefuse, no va servir per inclinar la balança a favor o en contra d'aquesta eina.

GraphStream, per contra, és una llibreria de creació recent, i només es centra en la visualització de grafs. De cara a aquest projecte, es van detectar diversos avantatges d'aquesta eina respecte Prefuse per algunes de les funcionalitats que es volien implementar:

- Modificació dinàmica del graf: GraphStream fa molt fàcil la tasca de generar algorismes que modifiquin la morfologia del graf o en variin algunes de les característiques de la visualització durant un recorregut del graf. En treballs anteriors que he realitzat amb Prefuse, aquest apartat ha resultat més difícil.
- Emmagatzemar atributs de tot tipus en els nodes: un apartat on destaca GraphStream respecte a Prefuse és la facilitat amb què es pot crear un graf amb nodes que contenen atributs de tipus molt diferents, o fins i tot atributs que només uns nodes tenen o altres no. Això ha servit, per exemple, per marcar el node inicial del sistema de transicions o emmagatzemar, per cada node, l'estat del sistema en aquell punt de l'execució. En un treball previ realitzat amb Prefuse, em va costar unir amb arestes nodes d'un graf quan els nodes tenien un atribut tipus diferent.
- Opcions de visualització: encara que Prefuse és molt extens en quant a opcions en la forma de visualitzar estructures de dades, em va semblar

que el que oferia per a grafs era poc i no massa agradable a la vista. En canvi, amb GraphStream es poden definir tota una sèrie d'atributs de visualització per nodes o arestes, ja sigui diferenciant els tipus per classe o descrivint la visualització per nodes o arestes concretes. A més, aquesta descripció resulta fàcil de realitzar, ja que GraphStream admet un subconjunt de CSS per especificar regles d'estil. Per exemple:

```
private String styleSheet =
    "node {" +
    " size: 20px;" +
    "       fill-color: white;" +
    " stroke-mode: plain;"+
    " stroke-color: black;"+
    "}" +
    "node.first {" +
    " stroke-color: green;" +
    " stroke-width: 2;" +
    "}"
```

En l'exemple anterior, definim un estil per tots els nodes, i a més, un estil complementari pels nodes que tingui com a classe "first". És a dir, el node inicial del sistema de transicions.

Quan diem que el segon estil és complementari del primer, significa que les regles de "node" també s'apliquen a "node.first", excepte quan aquest sobrescriu les del primer.

En aquest cas, tots dos estils defineixen la regla "stroke-color", per tant, quan node té la classe "first", el seu color de contorn serà verd en comptes de negre.

Per altra banda, amb Prefuse vam tenir problemes en una característica important dels sistemes de transicions generats: entre dos nodes es pot definir més d'una aresta. GraphStream genera automàticament una separació entre les arestes quan hi ha més d'una. Amb Prefuse es van obtenir alguns problemes de visualització, de manera que només es veia una aresta en tot moment.

En aquesta comparació no s'ha parlat encara de D3. D3 significa Data-Driven Documents. Aquesta eina pot considerar en Prefuse un antecessor, tot i que està orientat a la web i escrit en JavaScript. U

n avantatge de D3 és que, en ser basat en tecnologies web, permet fer servir la potència de CSS per definir regles d'estil per nodes o arestes, de manera similar a com ho fa GraphStream. Ja que en aquest cas no es tracta d'un subllenguatge, sinó que admet tota la sintaxi de CSS, permet més varietat que GraphStream per definir la visualització d'un graf.

Degut a que aquest projecte ja requereix d'altres eines no enfocades a la web, el llenguatge que s'ha triat per implementar-lo és Java. Per tant, D3 no ha resultat una opció viable.

Per les raons esmentades anteriorment, es va considerar GraphStream com una eina més adequada a la realització de les tasques d'aquest projecte.

5.2 Eina per generar descripcions de llenguatges

En aquest projecte, un dels objectius principals és poder executar un sistema de transicions. Per tal d'aconseguir això, ens cal poder entendre el llenguatge generat per CppInv(1), per tal de, per una banda, crear els nodes i arestes que calguin, i per l'altra, interpretar cadascuna de les condicions de les transicions.

ANTLR(9), ANother Tool for Language Recognition, és una eina que ofereix un entorn per desenvolupar reconeixedors, intèrprets, compiladors i traductors de llenguatges a partir de descripcions de gramàtiques. Es fa servir àmpliament en entorns acadèmics i industrials per construir tot tipus de llenguatges, eines, i entorns de desenvolupament.

Per exemple, la cerca de Twitter fa servir ANTLR per tractar amb l'entrada dels usuaris, i així poder determinar si s'està buscant un altre usuari, un tema del moment o una altra cosa. En total, es realitzen més de 2 bilions de consultes al dia.

Altres exemples inclouen els llenguatges Hive i Pig, que són els sistemes de data warehousing i anàlisi de Hadoop. Ambdòs fan servir ANTLR.També hi ha entorns de desenvolupament, com SQL Developer d'Oracle i Netbeans, que fan servir la versió en C++.

La raó principal per escollir ANTLR és que durant la carrera jo ja havia emprat aquesta eina, per tant ja hi estava familiaritzat. A més, el llenguatge del projecte és Java, un dels llenguatges suportats per ANTLR. El gran

avantatge que té és que autogenera les classes necessàries per integrar un llenguatge descrit en forma de gramàtica en un entorn de desenvolupament en Java.

Existeixen alternatives a ANTLR, però donat que la funcionalitat que calia extreure d'aquesta eina és limitada, a l'hora de decidir quina eina calia pel projecte, va pesar molt l'argument de la familiaritat amb el llenguatge.

Per llenguatges més sencills dels que es poden descriure amb ANTLR existeix la classe StringTokenizer de Java, que permet extreure fragments de text mitjançant expressions regulars. Aquesta alternativa no va ser viable, ja que el nostre llenguatge no es pot descriure amb aquestes expressions. Necessitàvem una eina que poguès definir llenguatges amb gramàtiques incontextuals, que permeten definir un conjunt més ampli de llenguatges, els llenguatges incontextuals, que inclouen tots els que es poden descriure amb expressions regulars, els llenguatges regulars.

Per exemple, en el cas dels operadors aritmètics, s'ha de preservar una prioritat en els operadors: no podem mantenir al mateix nivell el producte i la suma, ja que el primer s'ha d'aplicar abans. Això no es possible en el context dels llenguatges regulars.

5.3 Eina per interpretar llenguatges

En aquest projecte, per tal d'executar un sistema de transicions, necessitem una manera d'interpretar el significat de les condicions que imposen les transicions. Donada una transició i un estat del sistema previ a la transició, volem saber si es pot generar un estat del sistema posterior al pas per aquesta que compleixi les condicions que ens imposa.

Una primera manera d'atacar el problema seria crear un intèrpret dins d'una classe de Java, on es tradueixi cada condició en una instrucció de llenguatge imperatiu. Aquesta solució té un problema: nosaltres sabem tractar instruccions, però ens arriben condicions en forma de predicats lògics, que no podem tractar amb una traducció literal. Per tant, fem servir un tipus d'eina anomenada SMT-Solver.

Com s'explica a (1), els SMT-Solver són una extensió dels SAT-Solver.

El problema de SAT és el d'esbrinar si, donada una fórmula proposicional, aquesta és satisfactible, és a dir, existeix una combinació de valors que poden prendre les variables que fa que la fórmula s'avalui com a certa. Un SAT-Solver és un programa que resol el problema de SAT. El problema de SMT és una extensió de SAT. Es diferencia en que es treballa amb lògica de primer ordre sense quantificadors en comptes de lògica proposicional, i a més de trobar si aquesta fórmula és satisfactible, el resultat obtingut ha de verificar una teoria de fons. D'aquest problema s'encarreguen els SMT-Solvers, entre els quals podem trobar Barcelogic(10) i z3(11).

En aquest projecte, s'ha escollit el SMT-Solver anomenat z3(11). Aquest programa s'ha creat i es fa servir en molts projectes de Microsoft. El seu ús és lliure si es vol fer servir en l'àmbit de la recerca acadèmica. Rep com a entrada fitxers en format "smt", que es caracteritzen per fer servir una notació prefixa, com per exemple:

```
(= V1 (+ 1 (* 2 3)))
```

Aquest exemple es traduiria a una instrucció de Java de la següent manera:

```
V1 = 1 + 2*3;
```

Això no representa cap problema, ja que com veurem més endavant, ANTLR(9) ens ajuda a crear un arbre, amb el qual podem transformar una condició expressada en notació infixa, en una expressada en notació prefixa, depenent de l'ordre en el que es llegeixen els nodes de l'arbre.

6 Implementació del sistema

El projecte està dividit en dues parts: la visualització del sistema de transicions, i la simulació de l'execució.

6.1 Visualització del sistema de transicions

En aquesta secció descriurem el funcionament de la part corresponent a la visualització del sistema de transicions.

L'eina emprada en aquesta part és GraphStream, una llibreria escrita en Java que ens permet crear estructures de dades en forma de graf i visualitzar-les.

6.1.1 Disseny de la interfície gràfica

La interfície gràfica dissenyada per aquest projecte consta de dues finestres: una de principal (figura 8) i una segona de secundària (figura 9).

La finestra principal es troba dividida en tres parts: la finestra de visualització del graf, la barra del reproductor i la barra d'eines.

La finestra de visualització del graf és la part principal de l'aplicació. Aquí se'ns mostra el sistema de transicions que hem carregat, ja en forma de graf.

La barra del reproductor conté les utilitats bàsiques que necessitem per realitzar una execució del graf:

- Obre

  En prèmer aquest botó, se'ns ofereix l'opció de carregar un fitxer d'entrada. A la figura 8 encara no s'ha carregat cap fitxer, per aquest motiu es veu en blanc. Un exemple de finestra principal amb un sistema de transicions creat es pot veure a la figura 11.

  Quan es carrega un fitxer, primer apareix una finestra addicional, que ens permet seleccionar-ne un (fig. 12). Podem seleccionar entre dos formats: cfg i pth. Aquests formats representen el mateix programa, però cfg té més nodes i arestes, i pth intenta obtenir el mínim nombre de nodes i arestes. Per contra, les transicions del segon format són

Figura 8: Finestra principal

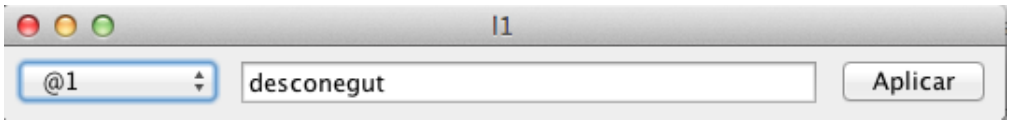

Figura 9: Finestra secundària

Figura 10: Barra del reproductor

molt més complexes. Un exemple de sistema de transicions en format cfg el podem veure a la figura 11, i la figura 13 representa el mateix programa en format pth.

- Fer una fotografia

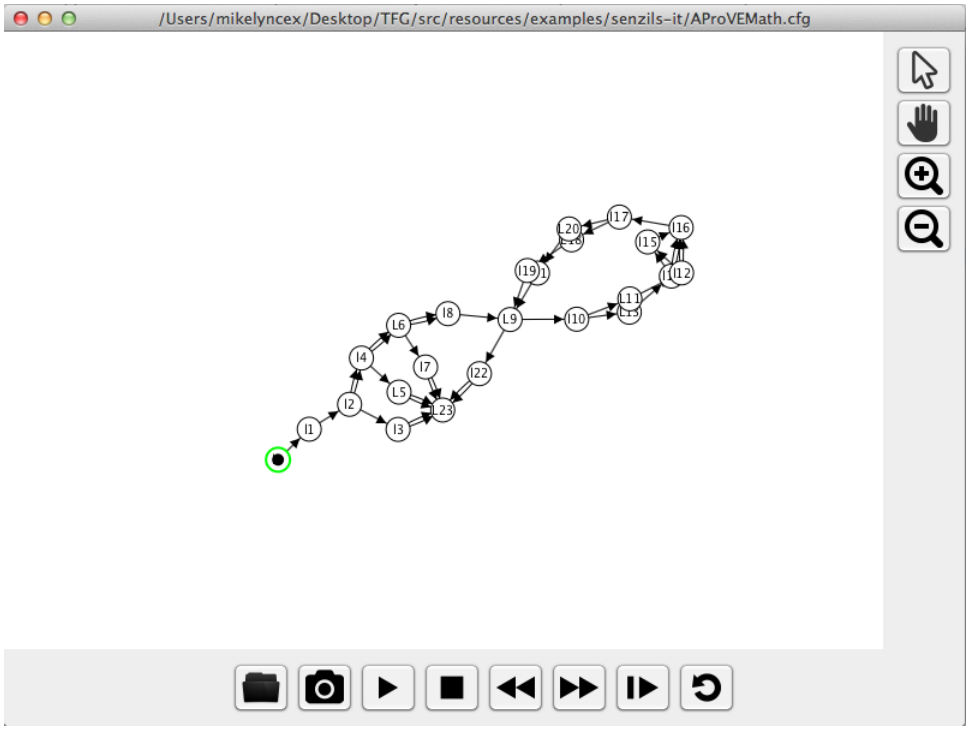

Figura 11: Finestra principal amb un sistema de transicions carregat

En qualsevol moment posterior a la càrrega d'un fitxer, es pot realitzar una fotografia de la finestra de visualització prement aquest botó (figura 14)

- Reprodueix/pausa

  Inicia o atura temporalment l'execució del programa. En prèmer aquest botó, es canvia l'icona (figs. 15 i 16) per indicar quina acció es troba disponible.

- Atura

  Atura completament l'execució actual, retornant a l'estat inicial. No modifica la visualització que es tenia del graf.

- Més velocitat

  Augmenta la velocitat d'execució de la simulació.

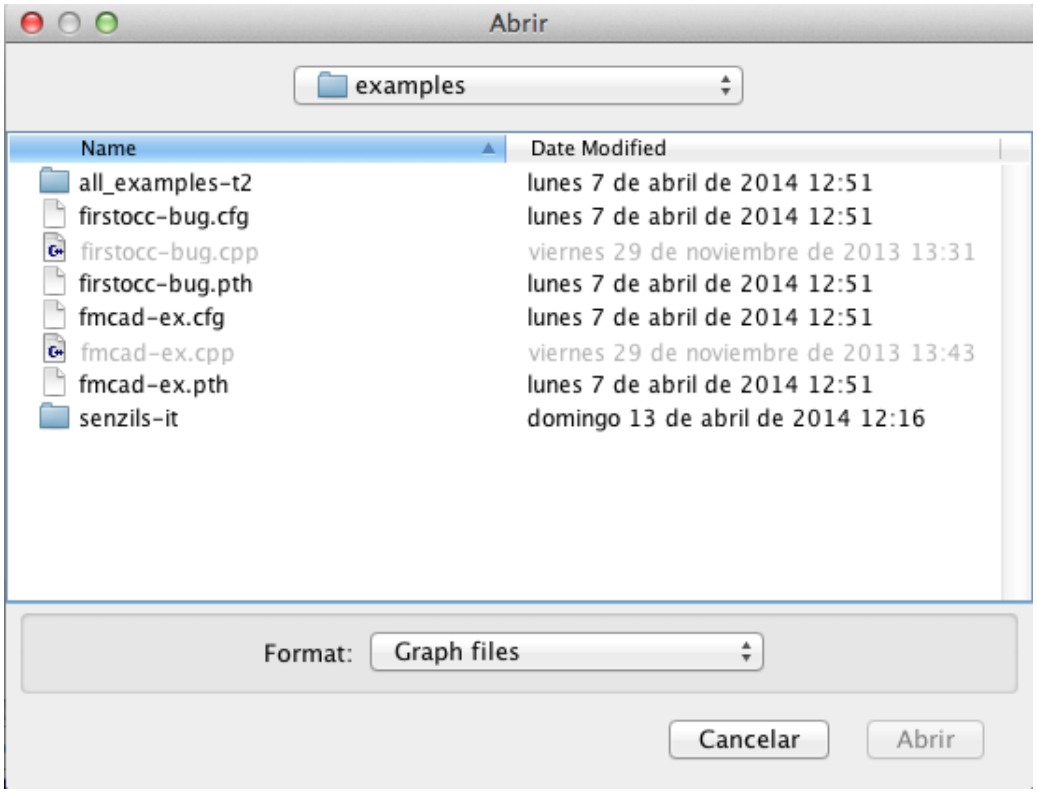

Figura 12: Finestra de càrrega de fitxers

- Menys velocitat

  Disminueix la velocitat d'execució de la simulació.

- Executa un sol pas

  Amb el programa pausat, aquesta opció permet executar una única transició del graf. Vam considerar que aquesta era una funció útil, ja que d'aquesta manera podem observar amb deteniment el que succeeix al voltant d'una determinada transició.

- Reinicia

  En prèmer aquest botó, s'atura completament l'execució actual, i es torna a carregar de nou el graf. S'assembla a la funció d'aturada, amb la diferència que aquesta comporta que no es conservin les propietats

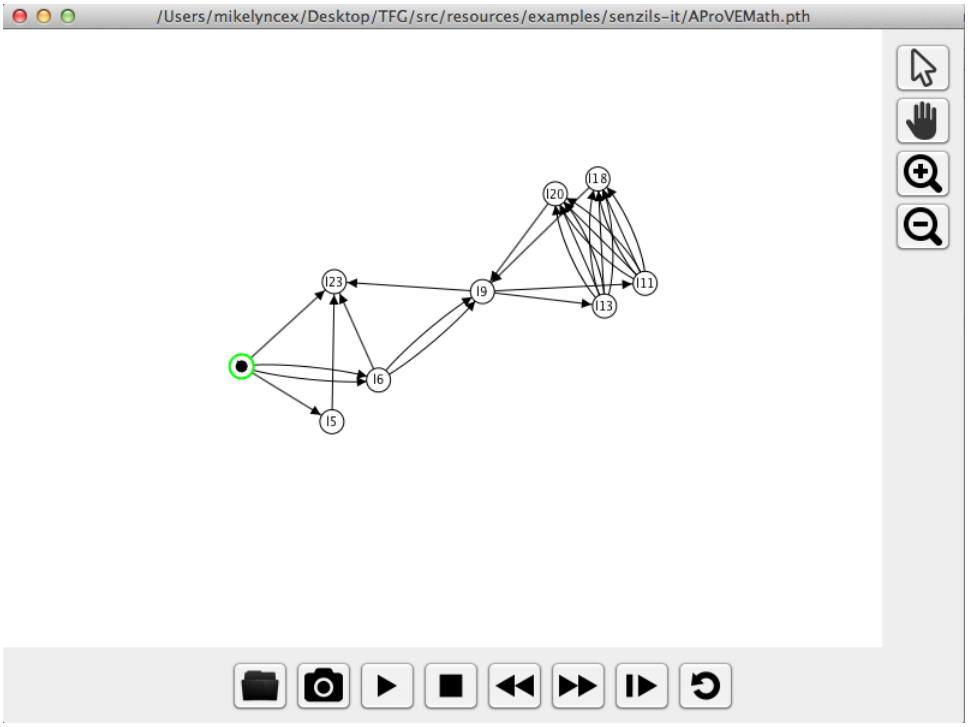

Figura 13: Sistema de transicions en format pth

de visualització de l'execució anterior.

A vegades pot ser útil disposar d'aquesta funció, ja que l'algorisme de generació de nodes no els genera i col·loca sempre en el mateix ordre, i per tant, a vegades pot quedar millor repartit. És una de les maneres que hem trobat de fer menys greu una de les limitacions de GraphStream: la forma que té de col·locar els nodes no és perfecta, i a vegades es produeixen solapaments.

En un principi, la funció que realitza el botó de reinici era la que feia el botó d'aturada, però vam considerar que era poc pràctic, durant una execució normal d'aquesta eina, tornar a carregar el graf, ja que després s'hauria de tornar a aplicar zoom o moure la càmera per aconseguir la visualització desitjada.

Així, es va derivar aquesta funcionalitat a un botó de reinici, i es va modificar el comportament del botó d'aturada.

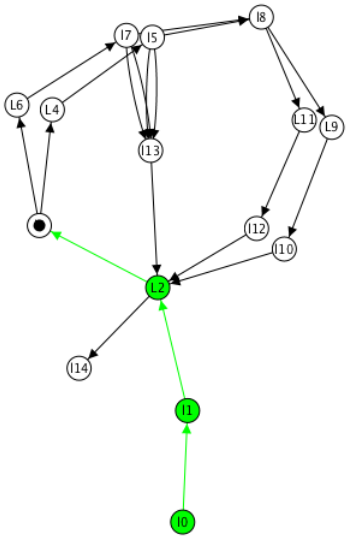

Figura 14: Fotografia del graf feta amb l'eina del programa

Figura 15: Botó reprodueix

Figura 16: Botó pausa

La barra d'eines conté elements els quals modifiquen el comportament i les accions que es poden fer amb el ratolí sobre la finestra de visualització.Consta dels següents elements:

- Cursor: Eina per defecte. Permet obtenir informació sobre les arestes que entren i surten d'un node en fer click sobre aquest. També obre l'editor de variables (fig. 9) per aquest node.
- Mà: Realitza una funció similar a la del ratolí. La diferència és que no obre l'editor de variables.

Figura 17: Barra d'eines

En un principi la funcionalitat que es volia realitzar amb aquesta eina va ser la de poder moure el graf fent ús de la tècnica "drag and drop".

Aquesta funcionalitat no va ser possible de realitzar. Donat que aquest projecte consta també d'una part d'anàlisi, s'han hagut de deixar de banda alguns aspectes de la visualització, com aquest, que haurien requerit una mica més de temps i no haurien aportat massa a l'objectiu principal del projecte.

- Zoom in: Permet apropar la visualització a un conjunt de nodes concret. Més endavant es donen més detalls del funcionament exacte d'aquesta eina.
- Zoom out: Com l'eina anterior, però allunya la vista en comptes d'apropar-la.

Els tres elements descrits anteriorment conformen la finestra principal de l'aplicació (fig. 8). A més, tenim una segona finestra (fig. 9), de caràcter més senzill, que permet a l'usuari assignar valors a les variables, i determinar el punt des d'on s'executarà el programa. Aquesta finestra és l'editor de variables.

La forma d'accedir a l'editor de variables es fa prement sobre un node amb el programa pausat, tenint l'eina del cursor seleccionada. En obrir-se la finestra, porta per títol l'identificador del node seleccionat. Aquest identificador també determina que en cas d'acceptar els canvis de variables realitzats, el node inicial passarà a ser aquest.

L'editor de variables consta de tres elements: una llista on apareixen totes les variables del programa, un quadre de text on hi figura el valor de la variable seleccionada a la llista, i un botó que permet aplicar els canvis realitzats.

La llista de nodes amb els seus valors s'obté d'un atribut que conté cada node. Aquest atribut és una còpia local de l'estat del sistema. El fet de tenir una còpia en comptes d'una única referència per tots els nodes permet, per una banda, fer servir l'editor de variables no només per modificar-ne el valor, sino també per consultar-lo en qualsevol moment de l'execució. Per altra banda, tenir valors en cada node és important de cara al correcte funcionament de l'aplicació.

El quadre de text mostra el valor de la variable seleccionada a la llista. Si aquest valor és desconegut, es mostra el valor "desconegut" en aquest quadre. Aquest valor s'esborra en prèmer aquest element amb el ratolí, amb l'objectiu de facilitar la inserció d'un nou valor.

Cada vegada que es modifica el valor que hi ha al quadre de text i es canvia l'element seleccionat de la llista, el nou valor s'emmagatzema en una estructura auxiliar. En prèmer el botó d'aplicar els canvis, aquesta estructura es copia a l'estructura real, i es torna a l'estat inicial de l'execució del programa, amb la diferència que ara el node inicial és un altre, i els valors de les variables es corresponen amb els que ha seleccionat l'usuari.

6.1.2 Estil de nodes i arestes

Per tal de definir l'estil de nodes i arestes, el mecanisme que es fa servir quan es treballa amb GraphStream és fer ús d'un atribut que s'assigna al graph, "ui.stylesheet", que és un String que segueix la sintaxi de CSS. Per exemple:

```
private String styleSheet =
    "node {" +
    " size: 20px;" +
    "       fill-color: white;" +
    " stroke-mode: plain;"+
    " stroke-color: black;"+
    "}" +
    "node.first {" +
    " stroke-mode: plain;" +
```

```
" stroke-color: green;" +
" stroke-width: 2;" +
"}"+
"node.marked {" +
"       fill-color: green,yellow, red;" +
" fill-mode: dyn-plain;" +
"}" +
"edge {"+
"  text-mode:hidden;"+
" text-alignment: along;"+
" text-offset: -10, -20;"+
"}"+
"edge.marked {" +
"       fill-color: green,yellow, red;" +
" fill-mode: dyn-plain;" +
"}"+
"edge.clicked {"+
"  text-mode:normal;"+
"}";
```

GraphStream admet un subconjunt de CSS que permet definir regles d'estil per nodes, arestes, i grafs. Es poden definir regles per tots els elements d'un tipus, regles que només s'apliquen a elements d'una classe i regles que s'apliquen només a un element que tingui un cert nom.

Per assignar una classe a un node o a una aresta, es pot fer en qualsevol moment posterior a la creació del graf, assignant un valor a l'atribut "ui.class". D'aquesta manera, GraphStream ens permet modificar, de forma dinàmica en temps d'execució, l'aparença de nodes i arestes.

En aquest projecte, s'aprofita aquesta característica quan es recorren els nodes, de manera que hi ha una gradació entre els colors verd, groc i vermell que indiquen si un node o aresta s'ha recorregut moltes o poques vegades (fig. 18).

Per tal d'aconseguir aquesta gradació, només cal afegir a l'element corresponent, dins l'atribut "ui.stylesheet", un paràmetre d'aquest tipus:

```
fill-color: green,yellow, red;
```

També cal afegir el paràmetre

`fill-mode: dyn-plain;`

, que indica a GraphStream que efectivament volem fer la gradació. En cas de no afegir aquest últim paràmetre, la gradació no funciona ja que només ens reconeixeria el primer element de la llista.

Una vegada s'han afegit els paràmetres necessaris, en temps d'execució només cal afegir un atribut "ui.color" amb un valor entre 0 i 1, que determinarà en quin punt de la gradació ens trobem. Per exemple, en el nostre cas, 0.75 correspondria al color taronja, en trobar-se a mig camí entre el groc (0.5) i el vermell (1). El resultat d'aplicar aquest estil a nodes i arestes es pot veure a la figura 18.

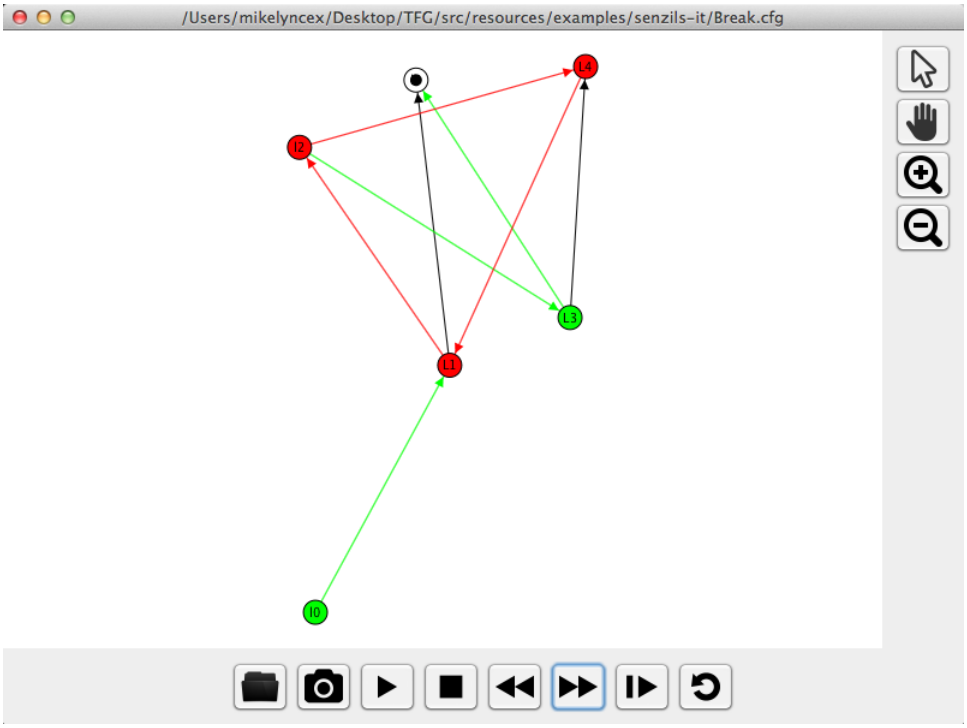

Figura 18: Els nodes marcats en vermell s'han visitat més vegades

Un altre ús que es fa de definir una classe per nodes i arestes és per tal de mostrar les etiquetes de les arestes. En grafs amb molt nodes, o on les transicions són extenses, resulta molest i no es veu bé si directament mostrem la informació de totes les transicions. Per tant, es va optar per crear una

classe especial per les arestes, de forma que quan es prem a sobre d'un node, les arestes que entren i surten d'aquest es marquen amb l'atribut

```
ui.class: clicked;
```

Per aquesta classe, diem que sí que volem que es mostrin les transicions. Si es torna a prèmer el mateix node, aquestes transicions desapareixen (fig. 19).

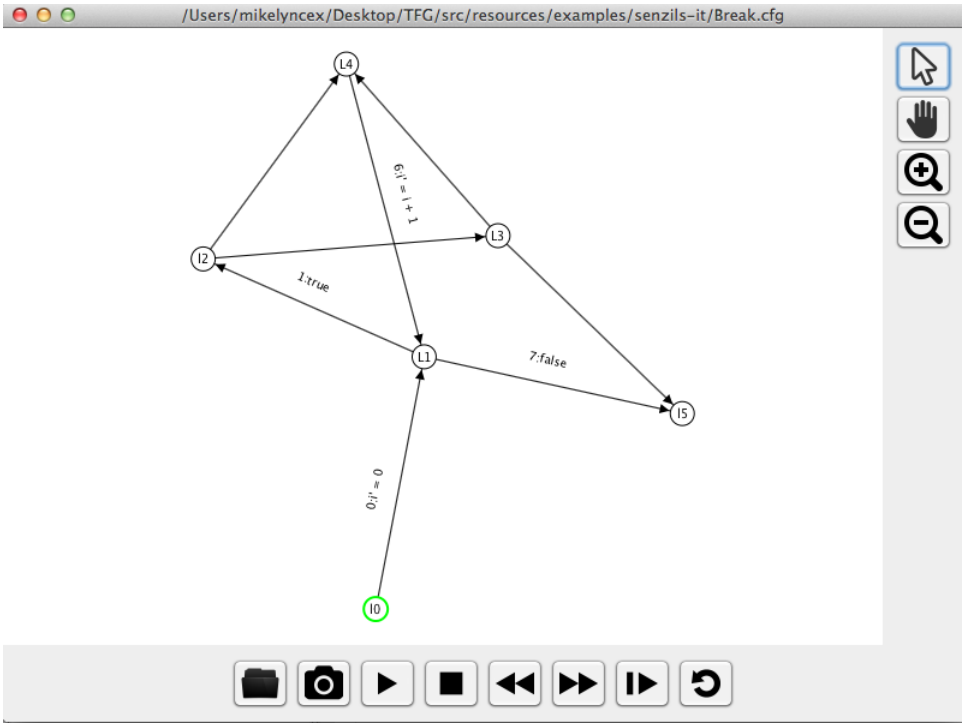

Figura 19: Es mostren les condicions en fer click sobre els nodes

Una alternativa que vam estudiar en un principi va ser que només es mostrés la transició si l'usuari premia a sobre de l'aresta, però és força més difícil, ja que GraphStream no ens proporciona un mètode directe per detectar les arestes quan son premudes, al contrari que els nodes, i a més vam tenir en compte factors d'usabilitat.

Les arestes són molt fines, per tant, si l'usuari vol fer click sobre elles, ha de tenir molta precissió. És molt més fàcil encertar en fer click a sobre d'un node, ja que la seva superfície és major. Per tant, per poder fer que aquesta funcionalitat fos útil, el que es podria haver hauria estat crear objectes de

tipus Sprite sobre les arestes, que fossin rectangulars i invissibles, i envoltessin l'aresta com una capsa. GraphStream permet detectar aquest tipus d'elements, però no hauria estat gens pràctic: les capses podrien solapar-se, tapant-se unes a altres en cas de tenir arestes properes entre sí.

Per tant, vam descartar aquesta idea, ja que vam considerar que el guany que rebia l'usuari no era tant gran en comparació amb els nous problemes que podia introduir l'ús d'aquests elements.

Per altra banda, també s'aprofita l'ús de classes en nodes per marcar de forma diferent el node inicial de l'execució, ja sigui el que considerem com a tal per defecte o el que indica l'usuari pel seu compte.

6.1.3 Zoom-in i Zoom-out

Un aspecte important de la visualització d'un graf en pantalla és la possibilitat de fer zoom d'una part concreta de l'estructura.

Moltes vegades, ens trobarem grafs amb un gran nombre de nodes o arestes. Aquests grafs és molt possible que es vegin massa compactes amb un nivell normal d'augments, arribant al cas que es solapin molts nodes i arestes (fig. 20).

En aquest cas, ens cal tenir la opció d'enfocar només una part concreta del graf i oblidar-nos de moment de la resta. Per defecte, quan es carrega un graf de mida gran, automàticament es realitzen augments i es centra la càmara sobre el node inicial (fig. 21).

En aquest projecte s'han definit dos tipus d'augment:

- Augment d'un punt
- Augment d'una àrea

L'augment d'un punt consisteix en que, en fer click, renderitzem de nou el graf, però augmentat al doble de la mida original i centrat en el punt on s'ha fet click.

L'augment d'una àrea consisteix en que, en fer click i arrossegar, renderitzem de nou el graf, però centrat en el punt mig de l'àrea seleccionada i l'augment és resultat de realitzar la mitjana aritmètica entre les proporcions de l'àrea

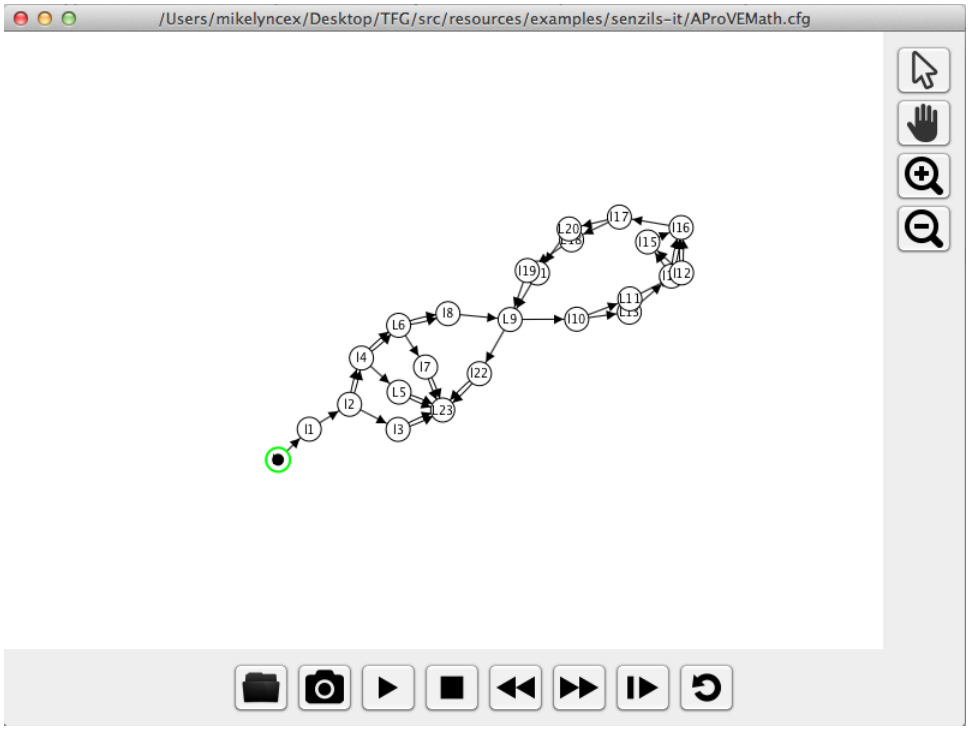

Figura 20: Exemple de graf amb un nombre considerable de nodes

seleccionada respecte la finestra original. Aquest tipus d'augment funciona força bé quan la relació d'aspecte entre l'àrea i la finestra és semblant, ja que l'usuari pot veure, abans de realitzar l'augment, quins nodes podrà veure en més detall. Per contra, quan aquesta relació és molt dispar, la imatge resultant pot no aportar informació útil.

En un principi, es va voler modificar el comportament del zoom, de manera que hi hagués un límit en quant a aquest nivell. El que es va fer va ser limitar el zoom quan el nombre de nodes visibles a la finestra de visualització del graf no era superior a 5. El problema de fer-ho d'aquesta manera és que segons el graf que carreguem, sobretot els que tenen format ".pth", ja en tenen menys d'inici, i per tant no permetíem a l'usuari fer zoom sobre aquest tipus de grafs. Ara s'ha modificat aquest límit, de manera que es poden realitzar augments a la finestra de visualització del graf fins que només es vegi un node.

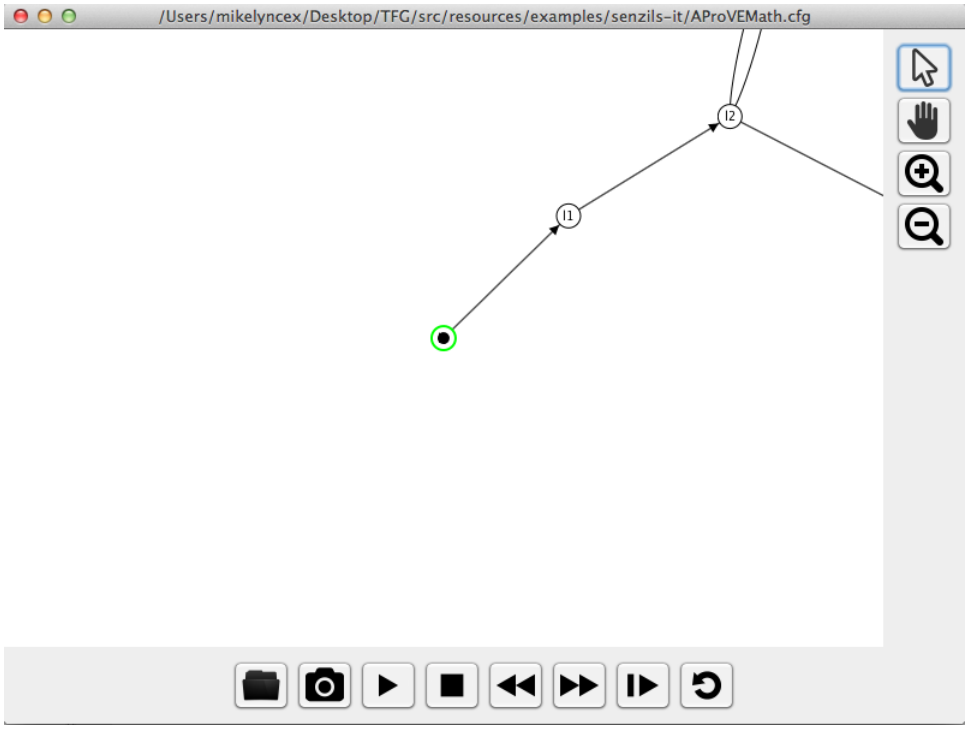

Figura 21: Exemple de graf amb molts nodes just després de ser carregat

6.1.4 Panning

Una vegada s'ha afegit l'opció de realitzar zoom sobre una part dels nodes del graf, un altre aspecte important de la visualització és la possibilitat de desplaçar-nos pel graf.

Aquest problema sorgeix precissament una vegada s'ha realitzat la funcionalitat del zoom, ja que per defecte, el "layout" de GraphStream procura posicionar els nodes de la millor manera possible, tenint en compte dues restriccions: hi ha d'haver el mínim nombre de solapaments entre nodes i arestes u tots els nodes del graf han d'aparèixer a la pantalla amb un nivell d'augments normal.

L'algorisme que fa servir GraphStream per determinar les posicions dels nodes és un basat en forces, el de Barnes-Hut, que repeleix els nodes que no es troben connectats entre si i atrau els que sí ho estan mitjançant una o més arestes. A més, procura que les arestes estiguin separades per una distància

de 1, en unitats pròpies de GraphStream, que anomena "Graph Units".

Però en fer zoom sobre la finestra, la segona restricció deixa d'existir, i hi ha nodes que no veiem en un moment donat. Com que pot interessar veure els nodes que es troben al costat dels que si es veuen, s'ha implementat la funcionalitat de panning.

El panning consisteix en moure la visualització en l'eix x o y, sense canviar el nivell d'augment o altres característiques de visualització. Això es realitza en aquest projecte mitjançant les tecles de direcció.

La principal dificultat d'implementar aquesta funcionalitat ha estat descobrir que les unitats que fa servir GraphStream són unes pròpies. Son les anomenades "Graph Units". Aquestes unitats s'han de transformar a píxels per poder establir una relació entre la mida de la finestra i la quantitat de desplaçament en els eixos x i y.

Una alternativa al mètode descrit anteriorment hauria estat realitzar aquesta funcionalitat no amb el teclat, sino amb el ratolí, fent ús de la tècnica "drag and drop". Això no ha estat possible degut a restriccions temporals del projecte, i no s'ha considerat tant important, ja que només pot aportar una petita comoditat a l'usuari i a canvi implementar aquesta funcionalitat interferia amb la realització d'altres processos del projecte.

6.1.5 Moviment de càmara

Un altre problema que introdueix el fet de tenir zoom és que durant una execució, si mirem el graf amb un nivell d'augments determinat, podem perdre de vista el node actual de l'execució, ja que pot quedar fora de l'abast actual de la finestra de visualització del graf.

Per tal d'evitar aquest problema, s'ha implementat un moviment de càmera, que aprofita el que hem vist a l'apartat anterior de panning, per moure la càmera vertical o horitzontalment, depenent de a quina distància es troba el node actual de l'execució respecte el centre de la pantalla.

El que es fa és traçar una recta imàginaria entre el punt actual i el centre de la pantalla, i quan s'arriba a una certa distància, es mou la càmera a través d'aquesta recta de forma gradual, fins arribar al punt actual. Durant aquest període de temps, l'execució es para, de forma que l'usuari no perd de vista

en cap moment on es troba. D'aquesta forma, al final del moviment, el punt on es trobava l'execució passa a situar-se al centre de la finestra d'execució, i es reanuda la simulació.

La càmara només es mou en cas que no es vegin tots els nodes del graf per pantalla, ja que s'ha considerat que pot arribar a ser molest i pot causar distraccions el fet de veure el graf en moviment a cada pas de l'execució, encara que es vegi sencer per pantalla.

6.1.6 Posicionament absolut

Les funcionalitats anteriors no haurien estat possibles sense saber la posició absoluta dels punts de la finestra de visualització.

GraphStream permet crear un graf de dues maneres diferents: o bé creant el graf en el mateix thread que el visualitzador, o creant-lo en un de diferent.

Quan es crea el graf en el mateix thread, no cal cap mena de sincronització d'events, ja que la compartició de dades entre el graf i el visualitzador és directa. Com a inconvenient, no podem crear un graf en aquest mode si volem mantenir a la vegada una visualització amb la simulació d'execució, ja que llavors la simulació impediria interactuar amb qualsevol altra part de l'aplicació fins que acabi.

Quan es crea el graf en un thread diferent, no es compleix el que veiem a la figura 22. Normalment, el graf i el visualitzador es troben connectats directament, formant un bucle on un és "sink" d'events de l'altre.

Però en aquest cas, no podem connectar aquests elements directament, ja que es troben en threads diferents. Fins ara aquesta implementació interna havia estat completament transparent, de forma que no ens havíem de preocupar de com estaven connectats aquests dos elements per compartir dades i events.

Ara, si només volguéssim visualitzar el graf, tindríem l'estructura que es pot veure a la figura 23. El graf i el visualitzador no es troben connectats directament, sinó que ho fan mitjançant una "pipe" especial que s'encarrega de la sincronització entre threads.

Però nosaltres també volem que el visualitzador ens retorni informació sobre els events, de forma que hem de connectar de manera similar el visualitzador

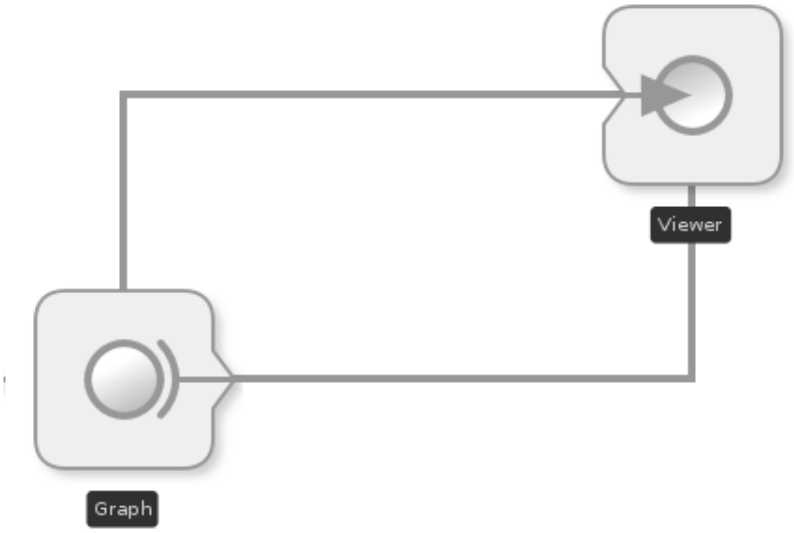

Figura 22: Graf i visualitzador en un mateix thread

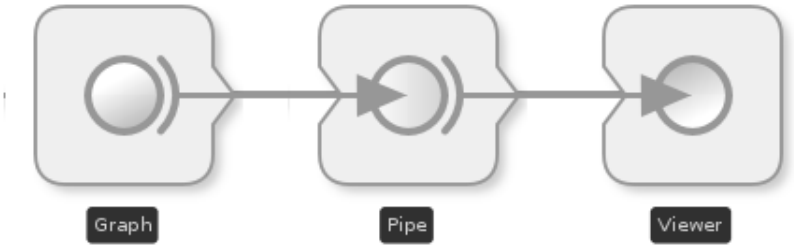

Figura 23: Graf i visualitzador en diferents threads

i el graf amb un objecte de la classe "ViewerPipe" de GraphStream. A més, hem de demanar a aquesta pipe que ens vagi enviant els events de forma regular, fent ús del mètode "pump()".

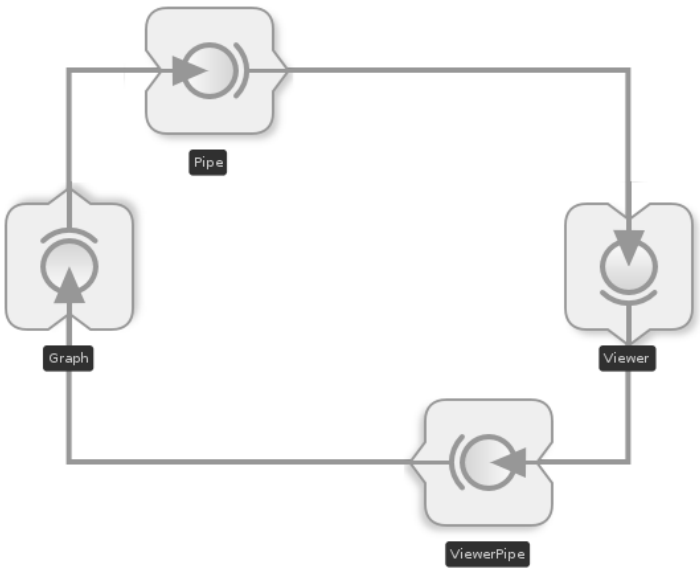

Figura 24: Graf i visualitzador connectats amb dues pipes

El resultat és el que es pot observar a la figura 24.

6.1.7 Pitjar en un node

Una altra funcionalitat de la visualització del graf és la possibilitat de realitzar diferents accions, segons l'eina que es té sel·leccionada en cada moment, quan es pitja en un node.

Fins ara, hem estat capaços de trobar la posició absoluta de qualsevol punt de la finestra de visualització, però no tenim manera de saber si en el punt on hem fet click hi ha algun node.

Per sort, GraphStream proporciona un mecanisme, en forma de "Listener", que permet recollir els events de fer click en un node.

El funcionament d'aquesta interfície, que s'anomena ViewerListener, és similar al de la resta de Listeners de Java. Proporciona una sèrie de mètodes dins dels quals tenim accès a un objecte que representa l'identificador del node sobre el que hem fet click. Mitjançant aquest identificador, podem obtenir

no només la posició del node, sinó també accés als seus atributs, fins i tot per modificar-los.

Això ens permet, per una banda, poder accedir a la finestra de l'editor de variables (fig. 9), i per l'altra, podem obtenir els noms de les arestes que entren i surten del node (fig. 19).

L'editor de variables és una finestra que mostra una llista de totes les variables que hi ha definides en el programa que s'executa en un moment donat, i per cada variable mostra el seu valor quan es fa click sobre ella. A més, permet modificar el valor d'aquestes variables, de forma que quan es pitja el botó "Aplicar", es guarden els nous valors i es reinicia el graf, però canviant el node d'inici pel node on s'ha fet click per accedir al graf.

Això permet realitzar un dels objectius del projecte, que és el de poder executar el graf des de qualsevol punt definits una sèrie de paràmetres.

La segona tasca que ens permet realitzar el fet de saber si estem pitjant un node és la possibilitat de mostrar, per les arestes que entren i surten del node, els seus noms, que hem guardat en forma d'atribut "ui.label". Aquest atribut es pot control·lar des del full d'estil CSS que hem definit anteriorment, de forma que per defecte el text no es mostra ja que hem definit que les arestes sense classe tenen l'atribut

```
text-mode:hidden;
```

En canvi, quan fem click sobre un node, canviem la classe de les arestes, de forma que ara s'apliquen les regles de la classe "edge.clicked", una de les quals és

```
text-mode:normal;
```

L'objectiu que es té en mostrar la informació d'aquesta manera és la de no saturar a l'usuari de l'aplicació quan es carrega un graf, ja que d'altra manera es mostrarien tots els noms de les arestes i es solaparien, i d'aquesta només es mostren per acció directa de l'usuari, i només per les arestes que hi ha al voltant d'un node, que normalment seran poques en comparació amb el total.

6.1.8 Poder arrossegar els nodes

Una de les funcionalitats que ofereix GraphStream és la possibilitat de definir o assignar un element "Layout" a un graf. Aquest element passa a tenir control absolut del posicionament dels nodes i arestes en el graf.

L'algorisme triat és un basat en forces, els nodes que no es troben units per arestes es repeleixen entre ells, i els que si ho estan s'atrauen. D'aquesta manera, es busca un equilibri en el qual es mostri el mínim nombre possible de solapaments entre nodes.

De totes maneres, aquest algorisme no és perfecte, ja que moltes vegades aquest requisit entra en conflicte amb el requisit de mantenir tots els nodes del graf en pantalla.

Una alternativa seria col·locar els nodes manualment, assignant de forma algorísmica posicions a aquests conforme es vagin creant, però no ens va semblar una solució viable.El problema de distribuir o posicionar nodes d'un graf en una superfície, sense que hi hagi solapaments entre nodes i arestes, està considerat com a difícil des del punt de vista algorísmic. No és objectiu d'aquest projecte trobar una millor solució als algorismes que ja ens proporciona la llibreria de GraphStream, i per tant es fan servir aquests últims.

De totes maneres, i encara que la aproximació que ens fa GraphStream acostuma a ser força bona, hem considerat que donar-li la opció a l'usuari a recol·locar alguns dels nodes era necessària. Per defecte, l'algorisme de forces del "Layout" es troba sempre en execució, i per tant, quan volem arrossegar un node per donar-li una nova posició, l'algorisme es posa en marxa, i torna a moure el graf fins que es troba en una posició d'equilibri.

Per sort, aquest comportament es pot desactivar, i la forma d'atacar aquest problema ha estat la següent: deixem en execució l'algorisme durant un temps determinat, el necessari per aconseguir un estat d'equilibri i on l'usuari no pot interactuar amb la interfície, ja que durant aquest temps el thread de visualització s'ha de paralitzar. Passat aquest temps, es desactiva l'algorisme i l'usuari pot arrossegar els nodes fins que troba una configuració que li satisfà (fig. 25).

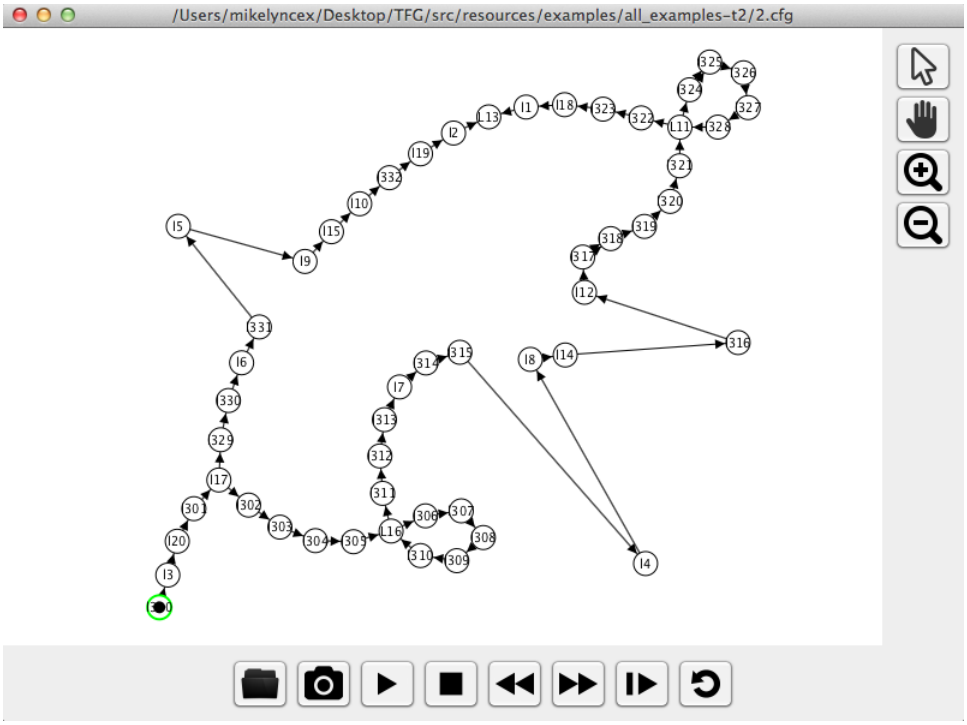

Figura 25: Exemple de graf on s'han arrossegat alguns nodes

6.1.9 Ús de Sprites

GraphStream, a més de nodes i arestes, ofereix un altre tipus d'element que anomena "sprites". Els sprites són elements gràfics semblants als nodes, en el sentit que es pot definir la seva forma i atributs de forma dinàmica, i també poden emmagatzemar qualsevol tipus de dades.

L'avantatge principal de fer servir sprites, però, és que aquests no es troben en una posició fixa respecte a nodes i arestes, sinó que poden circular lliurament pel graf en temps d'execució.

Aquesta particularitat permet fer-los servir per representar un recorregut del graf. Es poden enllaçar amb un node o aresta, de forma que al fixar la seva posició, en comptes de fer servir un posicionament absolut, es pot fer servir un de relatiu a nodes o arestes. Respecte als nodes, podem determinar si volem que els sprites es col·loquin just a sobre o a una certa distància, determinada en forma de offset en els eixos x i y. Respecte a les arestes, podem determinar

a quina distància del node origen de l'aresta el volem col·locar. En aquest cas, només tenim una magnitud, que es representa amb un nombre entre 0 i 1.

6.2 Simulació de l'execució

En aquesta secció descriurem el funcionament de la part corresponent a la simulació de l'execució de l'aplicació.

6.2.1 Descripció del llenguatge d'entrada

Primerament, el nostre projecte rep com a entrada fitxers que descriuen programes en forma de sistema de transicions.

En una condició, podem tenir equacions, que poden tenir solució o no, valors booleans (true o false), o desconeguts.

Els elements principals de les equacions són els operadors aritmètics i de comparació i les variables.

Els operadors que podem tenir són:

- Aritmètics: suma, producte de constant per una variable, resta binària i resta unària.
- De comparació: menor o igual, igual
- Assignació

Operadors que es poden veure en un llenguatge de programació imperatiu, com la divisió o el mòdul, s'han abstret en el lleguatge del sistema de transicions, ja que es tracta d'operadors que es poden obtenir mitjançant la combinació d'altres.

Per exemple, una condició que es pot trobar en un llenguatge de programació imperatiu és la de determinar si un nombre és parell. En C++ s'expressaria així:

$\boxed{x\%2 == 0}$

mentre que en aquest sistema de transicions s'expressa així:

$$\boxed{(2\$1' + \$2' - x = 0) \wedge (-\$2' <= 0) \wedge (\$2' - 1 <= 0)}$$

En aquest exemple es pot veure com es creen noves variables amb l'objectiu d'emmagatzemar els resultats intermitjos de l'operació.

En quant a les variables, distingim una mateixa variable quan ens referim al valor que pren a la sortida d'una transició i el que té abans d'executar-la. Les primeres reben el nom de variables primades i les segones, variables no primades. Excepte algunes variables de caràcter especial, tota variable té la seva versió primada i no primada.

Per exemple, si volem realitzar una assignació en un llenguatge imperatiu, ens trobarem amb una instrucció com aquesta:

$$\boxed{i = i + 5}$$

L'equivalent a aquesta instrucció en forma de sistema de transicions és aquesta altra:

$$\boxed{i' = i + 5}$$

Classifiquem les variables segons el seu significat dins el llenguatge del sistema de transicions:

- Variables normals
- Variables $
- Variables @
- Variables %

Les variables normals es corresponen amb una variable que podem trobar en qualsevol llenguatge imperatiu.

Les variables $ són un tipus especial de variables que s'utilitzen com a variables auxiliars per traduir càlculs que existeixen en llenguatges imperatius però no podem representar en format de sistema de transicions.

Les variables @ són un tipus especial de variables que representen valors aleatoris o variables que se'ls assigna un valor fruit d'un càlcul que no s'ha pogut traduir al llenguatge del sistema de transicions o poden representar una funció que demana dades d'entrada de l'usuari, cosa que no podem fer en aquesta

simulació. En el nostre cas, com tractem amb abstraccions del llenguatge del programa original, aquests valors es decideixen de forma aleatòria.

Les variables $ s'encarreguen d'emmagatzemar els resultats intermitjos, i tenen la particularitat que no es reaprofiten, si no que es creen noves cada vegada que hi ha un càlcul similar. Per tant, sempre se li assigna un valor a una variable d'aquest tipus i immediatament després es consulta.

Aquest mecanisme s'aprofita en el següent cas: no disposem de cap operador per realitzar operacions de divisió o de mòdul. Per tant, el que es fa és transformar aquestes operacions en una combinació de productes, sumes i restes.

Les variables % representen la mida d'un array. Encara que diferenciem aquest tipus de variables, el llenguatge del sistema de transicions no treballa amb arrays, sino que els abstreu.

Totes aquestes variables han ser de tipus enter.

El llenguatge descrit anteriorment és el que ens podem trobar en una condició pel que fa a les equacions. Cal notar que treballem únicament amb equacions lineals, per tant, no tenim potències ni arrels entre els operadors, per exemple. Això es deu a que és molt més fàcil algorísmicament treballar amb aquest tipus d'equacions.Però també tenim dos altres tipus de condicions:

- Valors booleans: són aquells que només poden ser "true" o "false".
- UNKNOWN: Aquests valors representen càlculs que o no ens importa el seu resultat o no em sapigut calcular sense més context. En aquest cas, una altra vegada, estem tractant una abstracció del problema original, per tant el comportament desitjat és que a vegades volem que aquestes condicions s'avaluin com a certes i d'altres, com a falses. D'aquesta manera, les solucions acceptades pel programa original són un subconjunt de les solucions que admetem nosaltres.

Una condició pot estar formada per una o més subcondicions, i poden ser qualsevol combinació dels diferents tipus descrits anteriorment. Aquestes subcondicions estàn relacionades mitjançant l'operador d'àlgebra booleana AND ($\wedge$). Això vol dir que per tal que una transició s'executi, s'han de complir totes les subcondicions de què consta.

Una vegada hem definit el llenguatge que representa les transicions, ens cal

una manera de poder tractar aquesta informació provinent dels fitxers. La manera que s'ha trobat és fent servir l'eina ANTLR.

6.2.2 Tractament del sistema de transicions

Un avantatge de fer servir ANTLR és que part de la seva sortida és el que s'anomena un AST (Abstract Syntax Tree).

Un AST és una estructura en forma d'arbre que conté a l'arrel un operador i als nodes fills els arguments d'aquest operador. Això ens permet definir una sintaxi pel nostre llenguatge de transicions. D'aquesta forma, més tard podrem saber com executar una transició que conté una suma. A més, també ens és molt útil tenir aquesta representació en arbre per saber quines variables intervenen en una transició.

Per generar un arbre AST amb ANTLR, necessitem crear una gramàtica. Aquesta gramàtica consta d'una sèrie de regles de la forma:

```
regla : símbols;
```

On els símbols poden ser de dos tipus: terminals o no terminals.

Els símbols terminals corresponen amb elements del llenguatge que estem tractant, com el símbol de sumar o un nom de variable.

Els símbols no terminals corresponen a regles de la nostra gramàtica. D'aquesta manera, podem tenir recursivitat, si a la part dreta d'una regla apareix el nom de la mateixa regla.

Per exemple, suposem el cas dels operadors aritmètics. Entre els operadors de suma i producte hi ha una prioritat, i és important respectar-la si més tard volem interpretar aquesta instrucció de manera unívoca. Veiem el següent exemple:

```
num_expr:   term ( (PLUS^ | MINUS^) term)*
        ;


term  :   atom ( (MUL^ | DIV^) atom)*
        ;
```

```
atom    :
            ID
        |   INT
        |   '('! num_expr ')'!
        ;
```

En aquest exemple, es defineixen 3 ordres de prioritat. De més a menys prioritari tenim: les expressions entre parèntesi, la multiplicació i divisió, i la suma i resta. El que hem fet és separar els operadors amb diferents prioritats en regles diferents, i hem encadenat les regles de forma que la primera fa referència a la segona, la segona a la tercera i la tercera a la primera. D'aquesta manera, més tard es crea un arbre sintàctic on els operadors més prioritaris es troben més a prop de les fulles, i els menys prioritaris més a prop de l'arrel. Això determinarà l'ordre de lectura d'aquests operadors i, per tant, l'ordre en el que es realitzen les operacions que defineixen.

En aquest exemple, també destaca l'ús dels elements ∧ i !. Aquests són indicadors que ens permet afegir GraphStream a les regles de la gramàtica que determinen l'aspecte final de l'arbre sintàctic generat. ∧ col·loca l'element anterior com a arrel dins una regla, i ! indica que no volem que aquest element aparegui a l'arbre generat. Ens serveix per eliminar símbols innecessaris, com els parèntesi.

Hi ha dos inconvenients principals a l'hora de fer servir una gramàtica incontextual en comptes d'expressions regulars.

El primer inconvenient s'anomena ambigüitat. La ambigüitat és un fenòmen que sovint succeeix en els llenguatges naturals, en forma de frases que es poden interpretar de diverses maneres. No és una qualitat desitjable en un llenguatge de programació, ja que volem que les instruccions s'interpretin d'una única manera possible.

És a l'hora de definir una gramàtica pel nostre llenguatge quan poden sorgir problemes: una mateixa instrucció es podria obtenir de dues maneres diferents segons l'ordre en el que s'apliquen les regles. Això no provoca cap error durant la generació de l'arbre, però com aquest arbre és incorrecte, quan volem interpretar el llenguatge generat podem realitzar malament els càlculs. Per exemple, es poden canviar l'ordre d'operacions amb prioritats diferents o aplicar la resta unària al resultat d'una operació quan el que es volia era aplicar-la només a una variable.

Més enllà del problema d'obtenir resultats incorrectes, es troba el fet que una gramàtica ambigua fa que una execució d'un programa escrit en el nostre llenguatge sigui no determinista, és a dir, el resultat que obtenim al final pot variar d'una execució a una altra sense que tinguem en el camí cap element aleatori.

Un altre problema que podem trobar en la generació d'una gramàtica és que no s'arribi mai a alguna de les regles. Això pot ser degut a tenir una gramàtica recursiva per l'esquerra.

ANTLR no admet aquest tipus de gramàtiques, ja que quan intenta substituir per una regla d'aquest tipus, el primer element de la substitució no és un terminal. Això és un problema perquè ANTLR controla el nombre de símbols terminals llegits, i cada vegada que es troba un, sap que està més a prop de generar la seqüència de regles que corresponen a una instrucció del llenguatge. Si es troba un símbol no terminal, no sap per quina regla ha de substituir-lo, ja que l'últim símbol terminal llegit és el mateix d'abans i per tant, no té una nova referència. D'acceptar aquest tipus de gramàtiques, ANTLR es podria quedar atrapat en un bucle infinit.

Per sort, com que ANTLR no admet les gramàtiques recursives per l'esquerra, aquest tipus de problema es detecta ràpid, en temps de compilació de la gramàtica, i només cal aplicar una sèrie de regles per transformar qualsevol gramàtica d'aquest tipus en una que, de ser recursiva, ho és per la dreta.

Per comunicar-se amb l'eina ANTLR, s'ha agafat com a base algunes classes del llenguatge Asl(13), dels materials de l'assignatura de Compiladors del Grau en Informàtica.

6.2.3 Creació del graf a partir del sistema de transicions

Un cop hem generat i compilat la gramàtica del nostre llenguatge, ens interessa realitzar diverses accions.

La primera és transformar el sistema de transicions, ja en forma d'arbre, en un graf.

La segona és interpretar les instruccions de forma que poguem simular l'execució del programa original, tot i que com hem dit abans, el nostre programa

és una abstracció de l'original, i per tant, la seva execució es comportarà diferent en alguns casos.

Per realitzar la primera tasca, farem ús de part de les eines que ens ofereix GraphStream.

Per la creació del graf, fem servir la classe especial "GeneradorGraf". Aquesta classe hereta de la classe "SourceBase" i implementa la interfície "Generator", ambdues de GraphStream.

"SourceBase" és una classe de GraphStream que ens permet afegir entrades de dades per generar un graf. Això ho fa permetent connectar un objecte de tipus graf com a "sink" del generador.

Un "sink" és un element que s'encarrega de rebre events. D'aquesta manera, podem crear el graf de forma algorísmica en una classe diferent, sense haver de passar l'objecte, i amb la possibilitat d'afegir o eliminar nodes de forma dinàmica.

La interfície "Generator" ens ofereix la descripció dels mètodes que fem servir des de fora de la classe per invocar la generació del graf, que són: begin(), end() i nextEvents();

El mètode begin() serveix per començar la generació del graf. El mètode end() serveix per acabar la generació del graf. El mètode nextEvents() serveix per evolucionar dinàmicament el graf. S'ha de cridar entre els mètodes begin() i end() i es pot fer de manera repetida. En el nostre cas no es fa servir, ja que el graf generat al principi es conserva fins al final de l'execució, sense afegir o treure arestes o nodes.

D'aquesta manera, podem crear un algorisme, que combinat amb la sortida que ens proporciona ANTLR, permet generar un objecte de tipus "Graph".

La classe "Graph" de GraphStream permet emmagatzemar dades de qualsevol tipus com a atributs de nodes o arestes. Això resulta molt útil, ja que per qüestions de visualització ens interessa, per una banda, guardar una còpia a cada node amb la llista de variables amb els valors que contenen l'última vegada que l'han travessat. Per altra banda, en una aresta ens interessa guardar un atribut de tipus "label" per poder mostrar més tard el conjunt de condicions que defineixen la "location" que representa.

La traducció entre el sistema de transicions i un graf és la següent: per cada

transició, es creen dos nodes, si no existeixen d'abans, que representen les localitats de sortida i d'arribada. Aquests nodes tenen com a identificador el mateix que figura com a nom de les localitats. A continuació es crea l'aresta que uneix aquests dos nodes, amb un identificador que és un número que només serveix per diferenciar-les, i dos atributs addicionals: representen tots dos la condició que determina si es travessa l'aresta, però un és en forma de String, per poder visualitzar-ho fàcilment a la pantalla, i l'altre és un arbre sintàctic, que més endavant ens permetrà navegar per la condició i evaluar-la.

6.2.4 Simulació de l'execució

En aquesta secció expliquem el procès que s'ha dut a terme per simular l'execució del graf de transicions.

Volem tenir dos tipus d'execucions: una on es detecti un node inicial i es comenci l'execució des d'aquell punt, i una altra on l'usuari pugui escollir un node inicial, i també pugui introduir valors per les variables que es defineixen al llarg del programa.

El que es va intentar al principi va ser generar un intèrpret del llenguatge de transicions. Com que a dins de cada transició guardem un atribut que conté l'arbre sintàctic que hem generat abans, som capaços de recòrrer aquest arbre i retornar solucions per les operacions aritmètiques bàsiques.

Aquesta forma de tractar el problema no va funcionar. El problema que vam trobar és que el llenguatge d'entrada no ve expressat com un conjunt d'instruccions que s'han d'executar en ordre. En aquest cas, la traducció hauria estat bastant literal.

Malhauradament, el llenguatge es troba expressat en forma de condicions, moltes d'elles amb més d'una clàusula. Va ser impossible tractar aquestes condicions, ja que alguns tipus de variables requereixen, per trobar el seu valor, examinar més d'una condició. Aquestes condicions formen sistemes d'inequacions lineals, que no podem resoldre fent un simple recorregut per l'arbre.

Per aquest motiu, es va optar per fer servir l'ajuda d'un SMT-Solver, en concret, z3(11).

Per tal de fer servir z3, necessitem transformar les condicions de les nostres

transicions en format smt, que és el que llegeix aquest programa. Un exemple d'aquest format es pot veure a continuació:

```
(benchmark paper
:logic QF_LIA
:extrafuns ((V20 Int))
:extrafuns ((V19 Int))
:extrafuns ((b3 Bool))
:extrafuns ((b4 Bool))
:formula
(and
(= V19 8)
(iff b3
(and
(<= (~ V19) 0)
)
)
    (iff b4
(and
(<= V19 0)
(<= V20 5)
)
)
(or b3 b4)
)
)
```

Primerament, cal, per cada variable que apareix a la part de ":formula", aquesta s'ha de declarar. Això es fa amb la sentència

```
:extrafuns ((V20 Int))
```

on Int indica que es tracta d'una variable entera. En aquest format, tractem amb dos tipus de variables, les enteres i les booleanes. Les variables booleanes són variables auxiliars que creem en aquest moment per distingir els diferents camins d'execució.

La major diferència amb el llenguatge que defineix els sistemes de transicions és que la notació dels operadors és prefixa en comptes d'infixa, és a dir,

```
(<= V19 0)
```

en comptes de

```
V19 <= 0
```

Els operadors amb què podem treballar són els mateixos, amb l'excepció que aquí es fa servir un símbol especial ($\sim$) per distingir la resta unària de la binària.

Per cada node, s'agrupen totes les transicions definides per les seves arestes de sortida. Cada camí es representa de la forma:

```
(iff b3
(and
(<= (~ V19) 0)
)
)
```

Finalment, volem que almenys un d'aquests camins sigui cert, per tant afegim la condició:

```
(or b3 b4)
```

Una vegada hem generat aquest format d'entrada per z3(11), ja podem realitzar una crida a aquest programa. Com a format de sortida, podem rebre un d'aquests dos:

```
unsat
```

o

```
sat
b3 -> true
b4 -> false
V19 -> 8
V20 -> 5
```

El primer representa la sortida de z3 quan no existeix cap solució que compleixi les condicions que ha rebut, i el segon mostra que la fórmula és satisfactible i dóna un conjunt d'assignacions per les variables que fa que algun dels camins sigui cert.

Per interpretar aquests resultats, i per actualitzar l'estat del sistema dins el nostre programa, ens cal tornar a generar una gramàtica amb ANTLR(9) que accepti aquest llenguatge. Aquesta gramàtica és més senzilla que la

corresponent al llenguatge del sistema de transicions, ja que en aquest cas només tenim una llista d'assignacions de variables.

7 Verificació

Per verificar el correcte funcionament de les funcionalitats d'aquest projecte, s'han fet servir com a jocs de proves una primer llista d'aproximadament 100 programes que corresponen a problemes de la plataforma Jutge.org(12), a més d'una segona llista amb prop de 400 problemes més complexos, que es tradueixen en sistemes de transicions amb més nodes o transicions més complicades. Tants els problemes de la primera llista com els de la segona ofereixen una versió en format cfg i una en format pth.

Partint d'aquests fitxers, s'ha generat un test que comprova, per cadascun dels fitxers, que són reconeguts per la gramàtica que hem creat per tractar amb el llenguatge del sistema de transicions.

8 Conclusions

L'anàlisi de programes és una branca important de la informàtica. Entre les tasques de què s'encarrega, s'inclouen la verificació de programes. Per verificar el correcte funcionament d'un programa, cal analitzar-ne la terminació i l'accessibilitat del codi.

Encara que existeixen eines que són capaces de trobar heurístics per aquests problemes, existeixen poques alternatives quan el que es vol és analitzar els resultats obtinguts durant un anàlisi d'aquest tipus, o es necessiten eines específiques per tractar problemes concrets.

És per això que neix la necessitat de crear un projecte com aquest, que pot servir com a eina complementària per visualitzar sistemes de transicions i simular-ne l'execució.

En un projecte d'aquest tipus cal la combinació de diversos coneixements: per una banda, ha calgut fer ús d'una llibreria gràfica per permetre la representació en forma de graf dels sistemes de transicions. Per altra banda, ha calgut interpretar aquest sistema mitjançant la generació d'una gramàtica que acceptés el llenguatge del sistema de transicions, i també s'ha fet servir un SMT-Solver per trobar una combinació de valors per les variables del sistema que fan que es compleixin les condicions d'una o més transicions, per tal de poder simular una execució del programa original. Per últim, per interpretar la sortida creada pel SMT-Solver, ha tornat a caldre la generació d'una gramàtica per aquest llenguatge.

8.1 Valoració personal del projecte

Des del meu punt de vista, aquest projecte ha estat una experiència enriquidora per a mi. Normalment els projectes que he realitzat a la universitat no han estat tant extensos, ni en durada ni en volum de treball, i he pogut comprovar de primera mà que planificar bé un projecte d'aquesta mida és molt important.

En el tema de la planificació és on he patit més, ja que amb un gran marge de temps i amb tasques que poden ser molt extenses costa molt més calcular aproximadament quant de temps trigaré en fer cadascuna, i al final he notat

molt la meva manca d'experiència.

Tot i això, confio en que aquest projecte m'hagi servit per aprendre una mica de planificació de projectes de cara al futur.

8.2 Dificultats

Una de les dificultats principals d'aquest projecte ha estat la sincronització entre threads. Donat que la llibreria GraphStream ja ofereix alguns mètodes que creen objectes en threads nous, com per exemple, generadors i algorismes de recorregut del graf, i permet crear en threads separats el graf i el visualitzador, s'ha fet una mica difícil tenir present quins threads s'havien creat en cada moment i si un mètode o dades calien alguna mena de sincronització.

A més, igualment he hagut de crear nous threads amb les llibreries de Java per crear la interfície gràfica, que no només consta del visor del graf, sinó també d'una finestra d'edició de variables, i barres d'eines per controlar l'execució o per canviar l'acció principal del ratolí.

8.3 Futures línies de desenvolupament

Per restriccions temporals del projecte, han sorgit una sèrie d'idees que no s'han pogut dur a terme, però considero que serien interessants desenvolupar:

- Crear un petit llenguatge per definir un camí d'execució. Aquesta idea consistiria en millorar la gestió que es té de l'objectiu d'executar el graf des de qualsevol node. Amb un llenguatge com els que s'han descrit durant el projecte, es podria definir una execució on no es pot travessar una o més arestes concretes.

  Una alternativa seria poder fer-ho visualment, fent click o seleccionant grups d'arestes que es volen prohibir.

- Realitzar vídeos de l'aplicació. Aquest era un dels objectius del projecte, però malhauradament no s'ha pogut dur a terme.

  GraphStream conté eines que permeten realitzar fotografies del graf en arribar certs events, però aquesta utilitat dona problemes quan el que

es vol grabar és una simulació d'execució i la interfície gràfica inclou més elements a més del visualitzador del graf.

Per altra banda, només permet extreure imatges quan hi ha un event del graf, i no munta les imatges (a la pàgina del projecte recomanen fer ús d'una eina externa).

Amb més temps, es podria investigar alguna altra manera d'obtenir vídeos a partir de les execucions del programa, amb la possibilitat de començar el vídeo i aturar-lo quan l'usuari ho decideixi.

El que s'ha aconseguit al final ha estat un compromís amb l'objectiu inicial: no es poden realitzar vídeos sobre l'aplicació, però si es poden fer fotografies en qualsevol moment de la finestra de visualització del graf.

9 Agraïments

Als meus amics i familia, per animar-me en els moments difícils del projecte.

Al director del projecte, Albert Rubio, a qui he pogut acudir sempre que tenia algun dubte sobre el projecte i per dedicar-me el seu temps.

També vull agrair als meus companys de feina a la UOC, que m'han donat alguns consells sobre la part final del projecte, respecte a l'elaboració de la memòria.

Referències

[1] Daniel Larraz; Albert Oliveras; Enric Rodríguez-Carbonell; Albert Rubio
"Proving Termination of Imperative Programs Using Max-SMT"
Universitat Politècnica de Catalunya

[2] Flemming Nielson, Hanne Riis Nielson, Chris Hankin (2005)
"Principles of Program Analysis"
Springer

[3] Dr. Denise Gorse
"Limits of computation: Undecidable Problems"
University College London (UCL)

[4] "LTSA - Labelled Transition System Analyser"
`http://www.doc.ic.ac.uk/ltsa/`

[5] Bas Ploeger; Carst Tankink
"Improving an Interactive Visualization of Transition Systems"
Eindhoven University of Technology - The Netherlands

[6] "prefuse — interactive information visualization toolkit"
`http://prefuse.org`

[7] "D3.js - Data-Driven Documents"
`http://d3js.org`

[8] "GraphStream - A Dynamic Graph Library"
`http://graphstream-project.org`

[9] "ANTLR Parser Generator"
`http://www.antlr3.org`

[10] Miquel Bofill; Robert Nieuwenhuis; Albert Oliveras; Enric Rodríguez-Carbonell; Albert Rubio
"The Barcelogic SMT Solver"
Proceedings 20th Int. Conf. on Computer Aided Verification (CAV)
Springer Lecture Notes in Computer Science 5123, pp.
294-298. 2008.

[11] Leonardo De Moura; Nikolaj Bjorner
"Z3: an efficient SMT solver"
Proceedings 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)
Springer Lecture Notes in Computer Science 4963, pp.\ 337-340. 2008.

[12] "Jutge.org - The Virtual Learning Environment for Computer Programming"
`https://www.jutge.org`

[13] Jordi Cortadella
"Asl - A Simple Language"
Universitat Politècnica de Catalunya
`http://www.lsi.upc.edu/~jordicf/Teaching/CL/Lab/`