Facultad
de
Ciencias

Desarrollo de una aplicación para la simulación de máquinas de Turing
(Development of an Application to Simulate Turing Machines)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Pablo Fernández Ruíz

Director: Domingo Gómez Pérez

Co-Director: Inés González Rodríguez

Junio - 2016

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Con la realización de este proyecto termina una etapa muy importante de mi vida. En primer lugar quiero dar las gracias a mis padres, Juan Manuel y Mº del Carmen y a mi hermano Álvaro por el apoyo que me han brindado a lo largo de todos estos años. Sin ellos y sin los valores y principios que me han transmitido mis padres a lo largo de todos estos años no hubiera sido posible llegar a donde estoy ahora.

En segundo lugar quiero dar las gracias a Domingo Gómez y a Inés González por ayudarme a realizar este proyecto. Quiero agradecer también sus enseñanzas durante todos estos años junto a otros profesores como Cristina Tirnauca o Diego García que siempre me han tendido una mano cuando lo he necesitado.

Y por último a todos mis amigos y compañeros de carrera por tantos años de historias y anécdotas, en especial a Ignacio (Tete), Chema y Jairo.

Resumen

Resumen

Las máquinas de Turing son un tema central en la mención de Computación del grado en Ingeniería Informática, estudiándose a fondo en la asignatura: Modelos de Cálculo. Se trata de un concepto abstracto, que en general resulta difícil de entender, lo que motiva la elaboración de una herramienta docente que ayude a los alumnos en el aprendizaje de las máquinas de Turing.

En este trabajo fin de grado se ha desarrollado una herramienta docente que permite a los alumnos visualizar y manipular máquinas de Turing de manera gráfica, mejorando la interfaz y aumentando las funcionalidades del script actualmente en uso en la asignatura, que sólo permite visualizar el resultado final de una máquina de Turing a través de una terminal.

La herramienta se ha desarrollado en *Python*, haciendo uso de la librería gráfica *Kivy*, y permite guardar o cargar ejemplos en formato *XML* siendo compatible con la herramienta *JFLAP*.

Palabras clave: Herramienta Docente, Máquina de Turing, Python, Kivy, XML.

Abstract

Turing machines are a main topic in the Computing track of the Computer Science degree and they are studied in depth in the course: Modelos de Cálculo. They are an abstract concept, difficult to understand, which motivates the design and implementation of a teaching tool with the purpose of helping the students in the task of learning Turing machines.

In this final degree project we have developed a teaching tool that allows students visualizing and handling Turing machines in a graphic way, improving the interface and increasing the features of the current script in use in the course, that only allows visualizing Turing machine's final result on a terminal.

The tool has been developed in *Python* using the graphics library *Kivy*, and allows saving or loading examples in *XML* format being compatible with the tool *JFLAP*.

Keywords: Teaching Tool, Turing Machine, Python, Kivy, XML.

1 Introducción

Este trabajo fin de grado parte de la propuesta del profesor Domingo Gómez de elaborar una herramienta gráfica que ayude a la docencia de máquinas de Turing dentro de la asignatura Modelos de Cálculo, del Grado de Ingeniería Informática. A continuación motivamos el interés de esta herramienta, hacemos un breve repaso de las alternativas existentes en la actualidad, y planteamos los objetivos concretos a alcanzar en el trabajo.

1.1. Motivación

Las máquinas de Turing (Turing (1936)) fueron esenciales en el desarrollo de la informática tal y como la conocemos ahora y son un concepto central en Teoría de la Computabilidad, que se imparte en la asignatura Modelos de Cálculo en el grado de Ingeniería Informática de la Universidad de Cantabria.

La relevancia de las máquinas de Turing en la Informática reside en su capacidad para modelar cualquier computador de propósito general. Podría decirse que una máquina de Turing puede hacer todo lo que un puede hacer ordenador real (Sipser (2006)). Dicho de otro modo, todo cálculo que pueda hacerse con una máquina existente más o menos sofisticada también podría hacerse con una máquina de Turing estándar.

La tabla 1.1 recopila una serie de paradigmas de computación que pueden ser emulados por una máquina de Turing.

Nombre	Descripción
Máquinas de Turing Mejoradas	Con multiples cintas, más símbolos en la entrada...
λ-cálculo	Método para definir y manipular funciones.
Funciones recursivas	Funciones en los número naturales.
Máquinas de registros	Número finito de registros que almacenan números naturales.
Lenguajes de programación	Java, Python, C++...

Tabla 1.1: Modelos de cálculo que son equivalentes a la máquina de Turing

La figura 1.1 muestra diferentes tipos de autómatas con su potencia de cálculo. Así, se puede ver que la simulación de un autómata finito se puede realizar utilizando un autómata con pila y éste a su vez se puede simular utilizando una máquina de Turing.

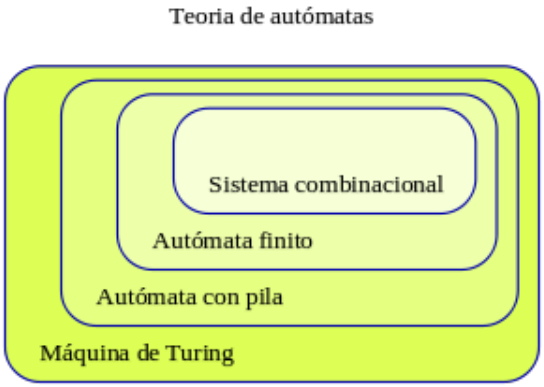

Figura 1.1: Teoría de autómatas.[a]

[a]Fuente: Wikipedia - Máquina de Turing

En el Grado de Ingeniería Informática de la Universidad de Cantabria, las máquinas de Turing se estudian en la asignatura Modelos de Cálculo, de 4º curso. Hasta ahora, en esta asignatura los ejercicios sobre las máquinas de Turing se han realizado a mano, utilizando lápiz y papel. Esto supone una tarea incómoda para el alumno dificultando el proceso de aprendizaje ya que para la realización de un problema sencillo, como una máquina con ocho estados, el proceso de depuración a mano por parte del alumno es bastante tedioso y no siempre resulta formativo.

Al tratarse de un concepto abstracto, a los alumnos no siempre les resulta fácil comprender y trabajar con máquinas de Turing. Se ha demostrado que el uso activo de herramientas de visualización puede ayudar a los alumnos en el proceso de aprendizaje de conceptos abstractos de informática (Naps u. a. (2002)). Es por ello que, aun considerando que el uso de lápiz y papel constituye una buena práctica para el aprendizaje, parece recomendable complementarla con una herramienta que facilite el proceso de depuración además de permitir una visualización gráfica de todo el proceso y de los elementos de la máquina. Actualmente en la asignatura de Modelos de Cálculo, la única forma en la que se puede depurar finalmente una máquina de Turing es con un script escrito en Python, desarrollado por el profesor. Este script tiene una serie de limitaciones:

- No existe una posibilidad de depurar.
- No hay una interfaz gráfica.
- Es necesario modificar la entrada de una forma específica para que funcione el script.

Todas estas restricciones complican el aprendizaje del funcionamiento de las máquinas de Turing por parte de los alumnos, por lo que se decidió que era importante el desarrollo de una herramienta para solventar dichas carencias.

1.2. Herramientas existentes

Ya existen algunas herramientas previas al desarrollo de nuestra aplicación. Por ejemplo, la herramienta en (Barwise u. Etchemendy (1993)) que permite crear y experimentar con máquinas de Turing y autómatas. (Taylor (1998)) utiliza el software *Deus Ex Machina* permitiendo a los usuarios experimentar con máquinas de Turing, y distintos tipos de autómatas. JFLAP (Rodger (2006)) permite la simulación únicamente de forma gráfica de varios tipos de autómatas incluyendo las máquinas de Turing. Todas estas herramientas no se ajustan a las necesidades de la asignatura por diversos motivos. La primera herramienta solo está disponible para una versión obsoleta del sistema operativo MAC, y las funcionalidades de la herramienta están relativamente anticuadas debido a su entorno gráfico. La segunda permite visualizar distintos tipos de autómatas de manera gráfica, pero no resulta nada intuitiva y resulta bastante tediosa además de presentar un entorno gráfico bastante antiguo y mejorable. La última herramienta, JFLAP, permite visualizar distintos tipos de autómatas de forma gráfica y presenta una interfaz más intuitiva y sencilla que la anterior pero no permite programar las máquinas manualmente ni la posibilidad de depuración a la hora de dicha tarea. Existe otra herramienta llamada SELFA-pro, usada en la asignatura de Teoría de Autómatas y Lenguajes Formales de la universidad de Extremadura, pero que no contempla la manipulación de máquinas de Turing (A. Rodríguez (2008)).

1.3. Objetivos

Por todo lo anterior, en este trabajo de fin de grado proponemos elaborar una herramienta específica para la simulación de las máquinas de Turing, mejorando en este campo las herramientas existentes.

Este proyecto tiene un triple objetivo.

- Ayudar a explicar el concepto de una máquina de Turing y facilitar a los alumnos el proceso de aprendizaje.
- Mejorar la versión existente en la actualidad con la que se estudian las máquinas de Turing. Dicha versión únicamente permite ver el resultado final de la máquina programada por el alumno en una terminal, ahora se ha aumentado la funcionalidad permitiendo además, la posibilidad de depurar ejecutando paso a paso y visualizar toda la información pertinente de manera gráfica.
- Compatibilidad con la aplicación JFLAP (Rodger (2006)) para poder visualizar las máquinas de Turing programadas con nuestra aplicación de una manera diferente.

2
Máquinas de Turing

En este capítulo se explicarán los conceptos básicos de una máquina de Turing. Para más información, se recomienda consultar el libro (Sipser (2006))

2.1. Módelo teórico de una máquina de Turing

Propuesta por Alan Turing en 1936, una máquina de Turing funciona de manera similar a un autómata finito, la máquina dispone de una cinta con una memoria ilimitada, es un módelo de un computador de propósito general mucho más preciso que un autómata, pudiendo reconocer lenguajes formales.

Según (Copeland (2002)) una máquina de Turing puede resolver cualquier problema que un computador pueda resolver, dentro de los limites teóricos de la computación. Un problema en su visión más abstracta es, dada una entrada hallar una salida.

Utiliza una cinta infinita puesto que no hay restricciones de memoria al tener una capacidad de memoria ilimitada. La cinta se divide en celdas, cada celda contiene un símbolo. La máquina dispone de un cabezal que permite para cada celda de la cinta:

- Leer y escribir un símbolo.
- Mover el cabezal hacia su izquierda o derecha.

Las decisiones se toman en base a una tabla de reglas.

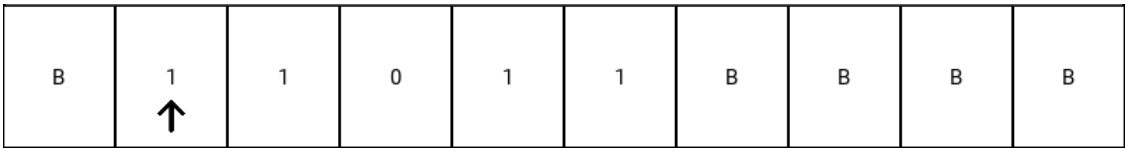

Figura 2.1: Cinta de una máquina de Turing

La cinta inicialmente, antes de ser computada, solo contiene símbolos blancos (*blank symbol*). En la Figura anterior, se ha introducido una entrada en la cinta, el cabezal está leyendo el símbolo 1 y según el estado actual de la máquina y su tabla de reglas, realizará las operaciones pertinentes.

2.2. Definición formal de una máquina de Turing

Definición 1 *Una máquina de Turing es una séptupla* $(Q, \Sigma, \Gamma, \delta, q0, B, F)$*, en donde* Q, Σ, Γ*, son conjuntos finitos y:*

1. Q es el conjunto de estados.
2. Σ es el alfabeto de entrada sin contener símbolos blancos.
3. Γ es el alfabeto de la cinta, donde $B \in \Gamma$ y $\Sigma \subseteq \Gamma$.
4. δ es la función de transición: $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$.
5. B es el símbolo blanco (blank symbol).
6. F es el conjunto de los estados finales ($F \subseteq Q$).

Se muestra a continuación un ejemplo de una máquina de Turing, en este caso realiza la operación suma. La codificación utilizada en el alfabeto de entrada es unaria, utilizando el símbolo 0 para delimitar el final de un número y el inicio de otro.

$$\begin{array}{rcc} Q & = & \{q0, q1, q2\} \\ \Sigma & = & \{0, 1\} \\ \Gamma & = & \{0, 1, B\} \\ \delta(q0, 0) & = & (q0, 1, R) \\ \delta(q0, 1) & = & (q0, 1, R) \\ \delta(q0, B) & = & (q1, B, L) \\ \delta(q1, 1) & = & (q2, B, L) \\ F & = & \{q2\} \end{array} \tag{2.1}$$

En la figura 2.1, la máquina presentaría esta configuración. $q_0 11011$, esto significa, que estaría en el estado q_0 leería un 1, se mantendría en el mismo estado, escribiría un 1 y finalmente movería el cabezal una posición a la derecha, acorde a su tabla de reglas. Alcanzaría entonces la siguiente configuración: $1q_01011$ finalizando la primera computación de esta máquina.

2.3. Lenguajes enumerables

No todas las entradas a una máquina de Turing hacen que la computación llegue a un estado final. Por ejemplo, para la entrada 101, la máquina de Turing definida en (2.1) llega a un estado final y es aceptada. En cambio, para la entrada 000, la máquina de Turing no llega a un estado final y, aunque para, la entrada no es aceptada. Resumiendo, la máquina para cuando llega a un estado aceptador o se queda sin reglas para leer y una entrada es aceptada solo si la máquina llega a un estado aceptador.

Definición 2 *Dada una máquina de Turing M, el lenguaje sobre el alfabeto Σ de las palabras que son aceptadas por M se llama el lenguaje aceptado por la máquina de Turing y se denotará como $\mathcal{L}(M)$.*

Definición 3 *Un lenguaje es recursivamente enumerable si existe una máquina de Turing M tal que $\mathcal{L}(M) = L$.*

Definición 4 *Un lenguaje es recursivo si existe una máquina de Turing M que siempre para ante cualquier entrada, siendo aceptadora por estado final, tal que $\mathcal{L}(M) = L$*

Siendo w una entrada,

- Lenguaje recursivamente enumerable:
  si $w \in \mathcal{L}(M)$ se para en un estado final.
  si $w \notin \mathcal{L}(M)$ se para en un estado no final o no se para.

- Lenguaje recursivo:
  si $w \in \mathcal{L}(M)$ se para en un estado final.
  si $w \notin \mathcal{L}(M)$ se para en un estado no final.

2.4. Finalidad de las máquinas de Turing

La finalidad de las máquinas de Turing estudiadas en la asignatura Módelos de Cálculo es doble.

- Simulación de algoritmos
- Problemas de decisión

Estos son los dos tipos de ejercicios que se realizan en dicha asignatura aplicando máquinas de Turing. La máquina de Turing definida en (2.1) realiza la suma de números enteros si la entrada tiene una forma específica. Con una entrada como 1101 la máquina se pararía en un estado final, si la entrada no es aceptada simplemente se pararía en un estado no final. Estamos entonces, frente a un lenguaje recursivo, realizando una simulación de algoritmos uno de los dos tipos de ejercicios comentados previamente.

El otro tipo de problemas serían los de decisión ¿$w \in \mathcal{L}(M)$?. Podría existir una máquina de Turing que para cierto lenguaje la máquina no se pare, (por ejemplo quedarse estancado en un bucle infinito) en este caso estaríamos frente a un lenguaje recursivamente enumerable.

3

Metodología, Análisis de Requisitos, Herramientas y Tecnologías

En este capítulo se describirá la metodología utilizada así como un análisis detallado de los requisitos del sistema y las herramientas y tecnologías a utilizar durante el desarrollo del proyecto.

3.1. Metodología

Para desarrollar la aplicación se ha utilizado una **metodología iterativa e incremental** (Sommerville (2005)) que combina elementos del modelo en cascada aplicado de forma iterativa. Se ha elegido esta metodología debido a la necesidad de proporcionar de manera rápida un conjunto limitado de funcionalidades para que puedan ser evaluadas por el cliente, generando como resultado de estas evaluaciones nuevos requisitos y un nuevo plan para incorporar más funcionalidades en la siguiente entrega. En el apartado 3.3 se documentan las iteraciones realizadas.

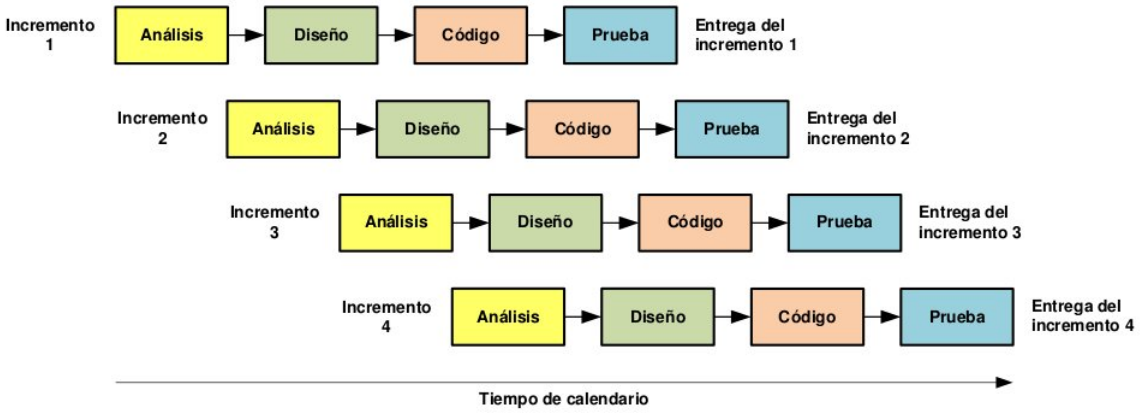

Figura 3.1: Fases del modelo iterativo e incremental.[a]

[a]Fuente: http://kvcapote.blogspot.com.es/

Como se muestra en la Figura 3.1 el modelo incremental aplica secuencias lineales de manera escalonada conforme avanza el tiempo en el calendario. Cada secuencia lineal produce incrementos del software (Sommerville).

3.2. Análisis de Requisitos

Para el desarrollo de la herramienta, se propusieron una serie de requisitos funcionales y no funcionales de una manera inicial. Dichos requisitos se han ido aumentando a lo largo del proceso iterativo e incremental según surgían nuevas necesidades o ideas para mejorar el software conforme se han ido evaluando los entregables.

3.2.1. Requisitos funcionales

En la Tabla 3.1 se muestran todos los requisitos funcionales del sistema determinados durante el proceso de desarrollo del proyecto.

ID	Requisito	Iteración
RF01	La aplicación permitirá al usuario cargar un fichero de texto con una máquina de Turing.	1
RF02	La aplicación permitirá al usuario modificar una máquina de Turing cargada previamente o crearla desde cero utilizando su editor de texto.	1
RF03	El editor de texto de la aplicación permitirá las opciones de edición: cortar,copiar y pegar.	1
RF04	La aplicación permitirá guardar la máquina de Turing que contenga el editor de texto de la aplicación en un fichero de texto.	1
RF05	El usuario podrá introducir manualmente el valor de los operandos deseados.	2
RF06	La aplicación permitirá al usuario indicar si desea utilizar un operando o dos.	2
RF07	La aplicación dispone de una cinta en la que se visualizarán 10 símbolos inicializada inicialmente con símbolos blancos.	2
RF08	La aplicación permitirá procesar la máquina de Turing que contenga el editor de texto de la aplicación inicializando así todas las variables necesarias en el sistema y la cinta de la aplicación en función de los operandos introducidos previamente.	3
RF09	La aplicación notificará al usuario si se ha producido un error en el procesamiento de la máquina de Turing con una ventana emergente(pop-up) indicando la naturaleza del fallo.	3
RF10	El usuario podrá ejecutar la máquina de Turing una vez procesada correctamente paso a paso.	3
RF11	El usuario podrá ejecutar la máquina de Turing una vez procesada correctamente en un solo paso.	3
RF12	El usuario podrá alternar de la ejecución paso a paso a la ejecución en un solo paso en el momento que desee.	3
RF13	La aplicación mostrará información en cada paso la regla leída y el estado actual.	4
RF14	La aplicación notificará al usuario con una ventana emergente(pop-up) cuando la ejecución de la máquina haya concluido.	4
RF15	La aplicación mostrará el resultado al finalizar la ejecución de la máquina de Turing.	3
RF16	El usuario podrá reiniciar el estado de la máquina al inicial volviendo a procesar la máquina de Turing.	3
RF17	La aplicación permitirá importar ficheros XML creados con la herramienta JFLAP7 y parsearlos.	5
RF18	La aplicación permitirá exportar la máquina de Turing del editor en formato XML para poder importarlo con la herramienta JFLAP7.	5
RF19	La aplicación mostrará un log de las reglas leídas en orden y el usuario tendrá la opción de exportar dicho log.	6
RF20	La aplicación mostrará para cada paso las reglas que puede ejecutar.	6

Tabla 3.1: Requisitos funcionales

3.3. Casos de Uso

Junto a los requisitos funcionales del sistema en la Figura 3.2 se expone el diagrama general de casos de uso.

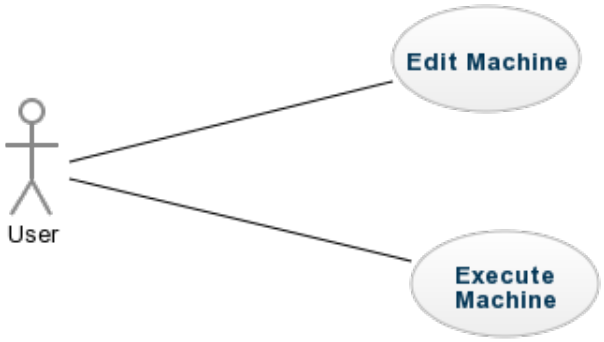

Figura 3.2: Diagrama general de casos de uso

El caso de uso Edit Machine, en la figura 3.3, atañe a las acciones que puede realizar el usuario en la pantalla de edición, la primera pantalla de la aplicación.

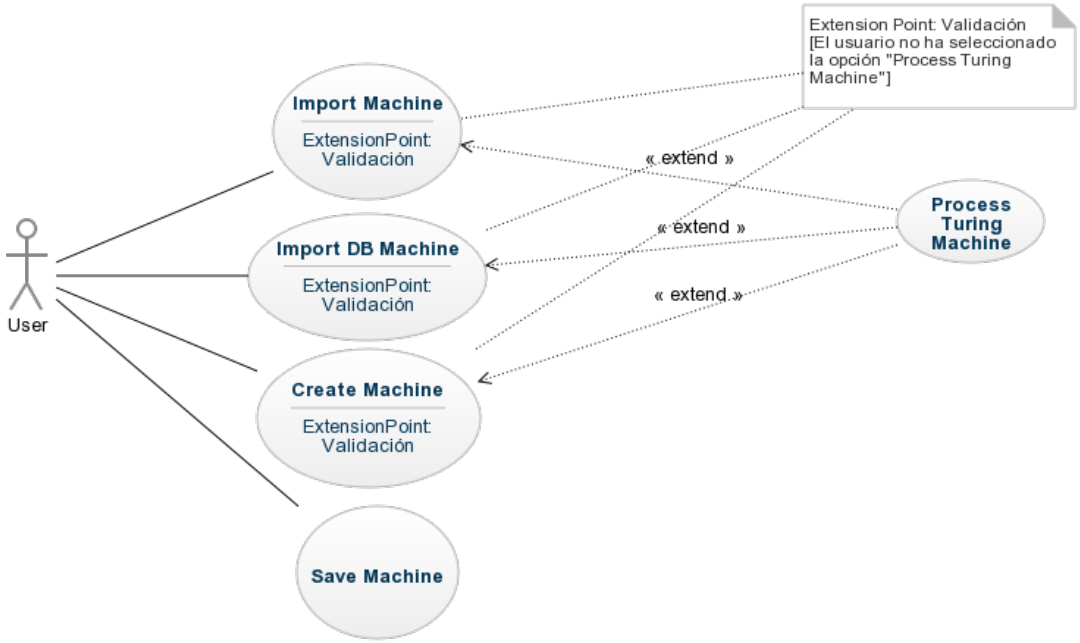

Figura 3.3: Diagrama correspondiente a Edit Machine

En la figura 3.4, podemos observar el caso de uso Execute Machine. El diagrama muestra las acciones que puede realizar el usuario en la pantalla de simulación es decir, la otra pantalla de la aplicación.

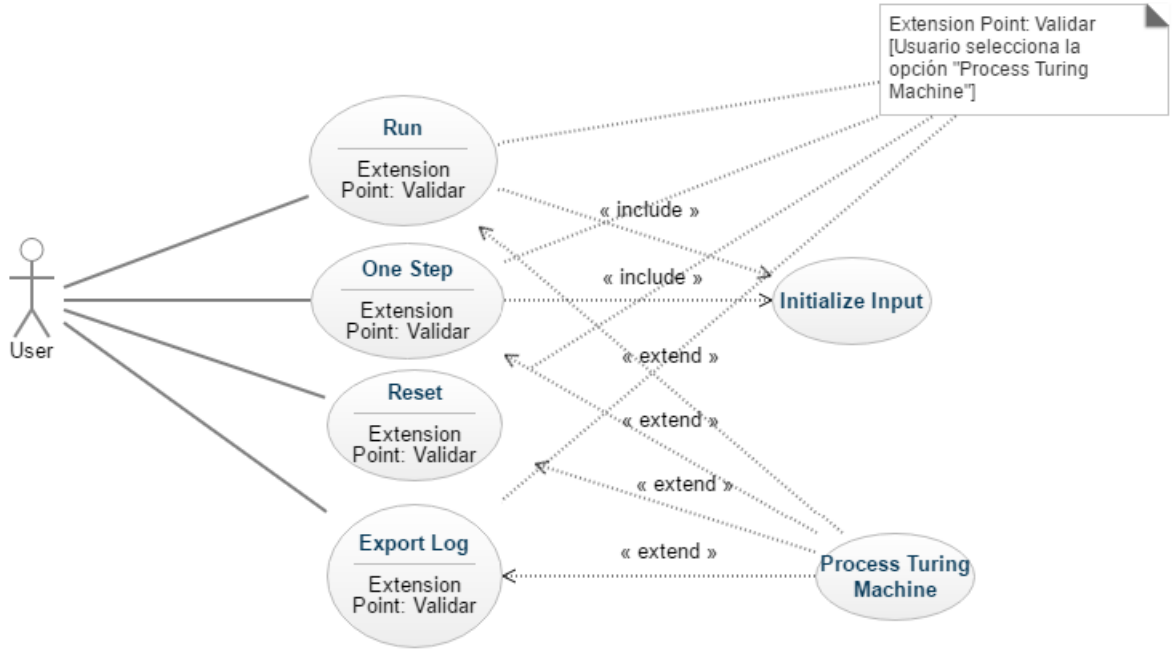

Figura 3.4: Diagrama correspondiente a Execute Machine

3.3.1. Requisitos no funcionales

Junto con los requisitos funcionales, que sirven para definir qué debe hacer la herramienta, se consideran otros requisitos relativos a características importantes de la herramienta. Estos requisitos no funcionales, en la Tabla 3.2, engloban consideraciones relativas, entre otros, a la seguridad, portabilidad, escalabilidad, usabilidad y rendimiento de la herramienta.

ID	**Requisito**	**Importancia**
RNF01	Seguridad: La máquina de Turing debe tener un formato determinado y tendrá que ser validado antes de ser procesado por el sistema.	Alta
RNF02	Portabilidad: La aplicación podrá ser utilizada en sistemas operativos que soporten Python y en Android.	Alta
RNF03	Seguridad: El tipo de dato de los operandos deberá ser de tipo Integer.	Alta
RNF04	Escalabilidad:La ventana de la aplicación se generará con el tamaño mínimo y podrá ser reescalada a un tamaño mayor.	Media
RNF05	Usabilidad y Accesibilidad: Los elementos gráficos de la aplicación presentarán un tamaño adecuado y están claramente definidos para facilitar la utilización de la aplicación por parte del usuario.	Media
RNF06	Integridad: Las variables almacenadas en el sistema una vez se haya procesado la máquina de Turing deben contener los datos correctos en la inicialización y en cada paso de la ejecución	Alta
RNF07	Rendimiento: El número de iteraciones de la máquina de Turing no superará las 999.	Alta
RNF08	Tecnológico: El lenguaje utilizado en la codificación será Python	Alta
RNF09	Idiomático: La aplicación deberá estar en inglés.	Baja

Tabla 3.2: Requisitos no funcionales

3.4. Iteraciones realizadas

De acuerdo a la metodología utilizada, se procede a detallar las funcionalidades que ha incorporado cada entregable.

1. Creación de la interfaz gráfica básica, satisfaciendo únicamente los requisitos (**RF01**, **RF02**, **RF03**, **RF04**).

2. Creación de la lógica de la cinta y de los operandos, satisfaciendo los requisitos (**RF05**, **RF06**, **RF07**).

3. Creación de la lógica de procesamiento y de ejecución, satisfaciendo los requisitos (**RF08**, **RF09**, **RF10**, **RF11**, **RF12**, **RF15**, **RF16**).

4. Implementación de mejoras, satisfaciendo los requisitos(**RF13**, **RF14**).

5. Creación del analizador sintáctico para ficheros XML y elaboración de un XSD para este, satisfaciendo los requisitos(**RF17**, **RF18**).

6. Creación del log y modificaciones visuales en la interfaz gráfica, satisfaciendo los requisitos(**RF19**, **RF20**).

3.5. Tecnologías

3.5.1. Python

Es un lenguaje de programación interpretado en el que se ha desarrollado este proyecto (Python (2016)). Se ha desarrollado el código en este lenguaje por imposición del cliente **RNF08**, debido a que el profesor conoce este lenguaje y le es más cómodo si debe interactuar en un futuro con el código.

3.5.2. Kivy

Se trata de una librería para Python de código abierto pensada para el desarrollo de aplicaciones y es la que se ha utilizado para desarrollar la interfaz gráfica (Kivy (2016a)). Se barajaron varias opciones a la hora de elegir una librería gráfica como *Pygame*, *PyQt* o *PyGTK*. Realizando un estudio de alternativas nos decantamos por Kivy, debido al soporte para Android que dicha librería sostiene y a que permite generar la estructura de la aplicación modificando una serie de widgets de una manera relativamente sencilla para obtener exactamente el diseño que se había pensado. Otro punto importante fue la documentación de cada librería siendo Kivy la más completa en este sentido.

3.5.3. XML y XSD

XML, e**X**tensible **M**arkup **L**anguage (w3schools (2016)), es un lenguaje de marcas diseñado para almacenar datos de forma legible, describe los datos de manera estructurada. En nuestro caso lo utilizaremos para almacenar máquinas de Turing.

También se ha utilizado un esquema XSD, **X**ML **S**chema **D**efinition (w3schools (2016)), para describir la estructura de nuestro documento XML relativo a una máquina de Turing, permitiendo así una validación del formato de dicho fichero. Las estructuras XML y XSD serán detalladas en el capítulo 4.

3.6. Herramientas

3.6.1. IDLE de Python

IDLE de (**I**ntegrated **D**evelopment and **L**earning **E**nvironment) es el entorno de desarrollo integrado en la librería estándar de Python (Python (2016)). Se ha utilizado para desarrollar todo el código de Python por las facilidades que presenta, como un buen soporte a la edición (incluyendo autocompletado o coloreado de sintaxis e indentación) y facilidades para la depuración (entre las que se incluye un registro de logs) y, además, forma parte de la distribución estándar de Python. La versión concreta utilizada es la 2.7.11.

3.6.2. Kivy Launcher

Es una aplicación disponible en el *Google Play Store* que permite ejecutar una aplicación desarrollada para Python con Kivy (Kivy (2016b)) en un dispositivo Android. Se ha utilizado esta alternativa frente a otras como Buildozer, por el siguiente motivo. Presenta una mayor facilidad de uso a la hora de realizar la aplicación para Android, sin necesidad de todo el proceso de compilado para generar una .apk innecesaria para una aplicación como la nuestra que no esta orientada al Google Play Store, tratándose de una herramienta docente para un público muy concreto.

3.6.3. LXML

Para el tratamiento de los ficheros XML y XSD se ha utilizado la librería *lxml* (LXML (2016)). Se planteó la posibilidad de utilizar JSON, pero finalmente se terminó descartando por mantener una compatibilidad con la aplicación JFLAP **RNF17**, **RNF18**.

3.6.4. JFLAP

JFLAP es un paquete de herramientas gráficas para el aprendizaje de conceptos relacionados con la Teoría de Autómatas y Lenguajes Formales, incluyendo máquinas de Turing (A. Rodríguez (2008)). Nuestra herramienta se ha desarrollado teniendo en cuenta ciertos aspectos de compatibilidad con JFLAP detallados en **RNF17** y **RNF18**.

4

Diseño y Desarrollo de la Aplicación

4.1. Diseño arquitectónico

A la hora de diseñar una estructura general del software ofreciendo una integridad conceptual del sistema, se ha decidido emplear una arquitectura en tres capas, separando el sistema en tres componentes: capa de presentación, capa de negocio y capa de datos. En concreto:

- **Capa de presentación:** Se corresponde con la interfaz gráfica de la aplicación, la cual realiza las llamadas (callbacks) necesarias a la capa de negocio. Dichos callbacks han sido programados con la librería Kivy.
- **Capa de negocio:** Se corresponde con toda la lógica implementada en Python que se activa cuando recibe un callback de la interfaz y se encarga de obtener los datos pertinentes de la capa de datos.
- **Capa de datos:** Se corresponde con una serie de ficheros XML en los cuales se almacenan las máquinas de Turing.

A continuación se explicará en profundidad cada una de las tres capas.

4.1.1. Capa de presentación (Interfaz)

La interfaz recoge información e interactúa con el usuario, por lo que se ha diseñado la manera más intuitiva y atractiva posible. Se ha optado por un diseño en dos pantallas. La primera proporciona un editor de texto para las máquinas de Turing. Con el objetivo de facilitar la tarea de edición se eliminan de esta pantalla el resto de aspectos que no son necesarios para esta tarea, máximizando la claridad y sencillez. La segunda pantalla se muestra una vez se ha procesado una máquina de Turing de manera correcta, presentando únicamente los aspectos necesarios para la simulación de la máquina programada anteriormente, eliminando el resto de distracciones.

Para diseñar la aplicación se han realizado entrevistas informales a un grupo de alumnos voluntarios de la asignatura en este curso, como potenciales usuarios de la aplicación y también se ha consultado con el profesor, como cliente del producto software y dada su experiencia en la docencia de máquinas de Turing. No ha sido posible observar a los usuarios utilizando la aplicación en su versión final, tal y como se recomienda en (Stone u. a. (2005)) porque la aplicación se ha desarrollado posteriormente a la asignatura para la que está pensada.

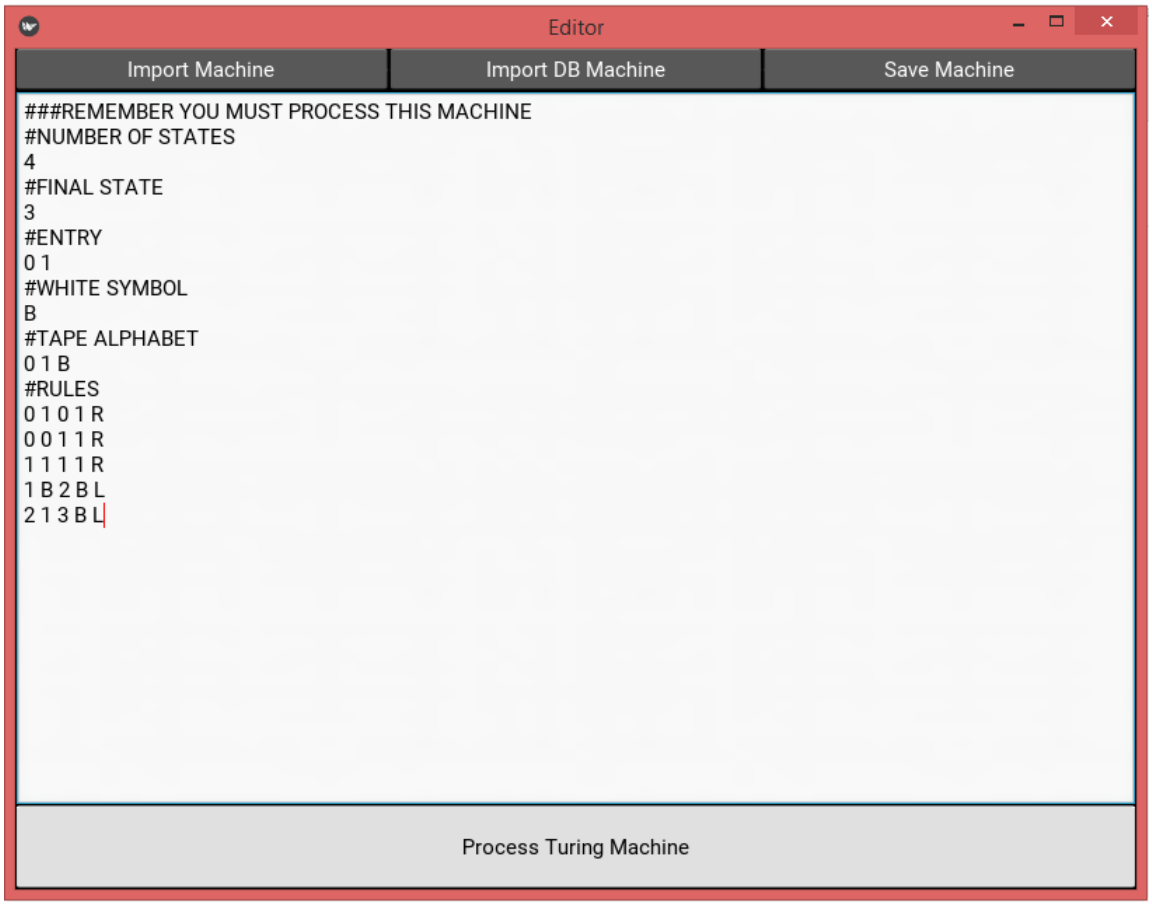

Figura 4.1: Pantalla del editor de texto de la interfaz

En la Figura 4.1 se ilustra la primera pantalla, correspondiente al editor la cuál contiene una máquina de Turing en el formato determinado por el responsable de la asignatura. Se procede a documentar las funcionalidades correspondientes a esta pantalla.

1. **Import Machine:** Permite importar una máquina de Turing en formato de texto o XML del sistema y mostrarla en el editor de texto.

2. **Import DB Machine:** Permite importar una máquina de Turing de nuestra base de datos en formato XML y mostrarla en el editor de texto.

3. **Save Machine:** Permite guardar la máquina de Turing que se encuentre en el editor de texto en un fichero de texto o en XML.

4. **Editor de texto:** En esta parte se muestra el código importado o programado por el alumno.

5. **Process Turing Machine:** Permite procesar la máquina de Turing que se encuentra en el editor y notifica si se ha producido algún tipo fallo de formato en la máquina.

Figura 4.2: Pantalla de simulación de la interfaz

En la Figura 4.2 se observa la pantalla de la aplicación en la cual se realiza la simulación (en esta figura no se encuentra inicializada la máquina) de la máquina de Turing que se encuentra en el editor de texto. Esta pantalla esta compuesta por:

1. **Operandos:** El usuario podrá especificar los operandos con los que desea inicializar la cinta.

2. **Resultado:** Al llegar a un estado aceptador la máquina notificará que se ha terminado la ejecución y mostrará el resultado.

3. **Initialize Input:** Permite inicializar la entrada con los operandos especificados anteriormente.

4. **Cinta:** Inicializada con símbolos blancos (blank symbols).

5. **Run:** Permite la ejecución en un solo paso de la máquina de Turing cargada.

6. **One Step:** Permite la ejecución paso a paso de la máquina de Turing cargada.

7. **Reset:** Permite reiniciar la ejecución al paso 0.

8. **Rules Info:** Muestra información de las reglas disponibles en cada paso de la ejecución teniendo en cuenta el estado en el que se encuentra la máquina y de la última regla leída.
9. **Machine Info & Log:** Muestra información de los estados inicial, final y actual además de mostrar un log de ejecución de las reglas.
10. **Export Rules Log:** Permite exportar el log de ejecución para una visión completa de éste.
11. **Edit Turing Machine:** Permite volver a la pantalla de edición para modificar o añadir reglas nuevas.

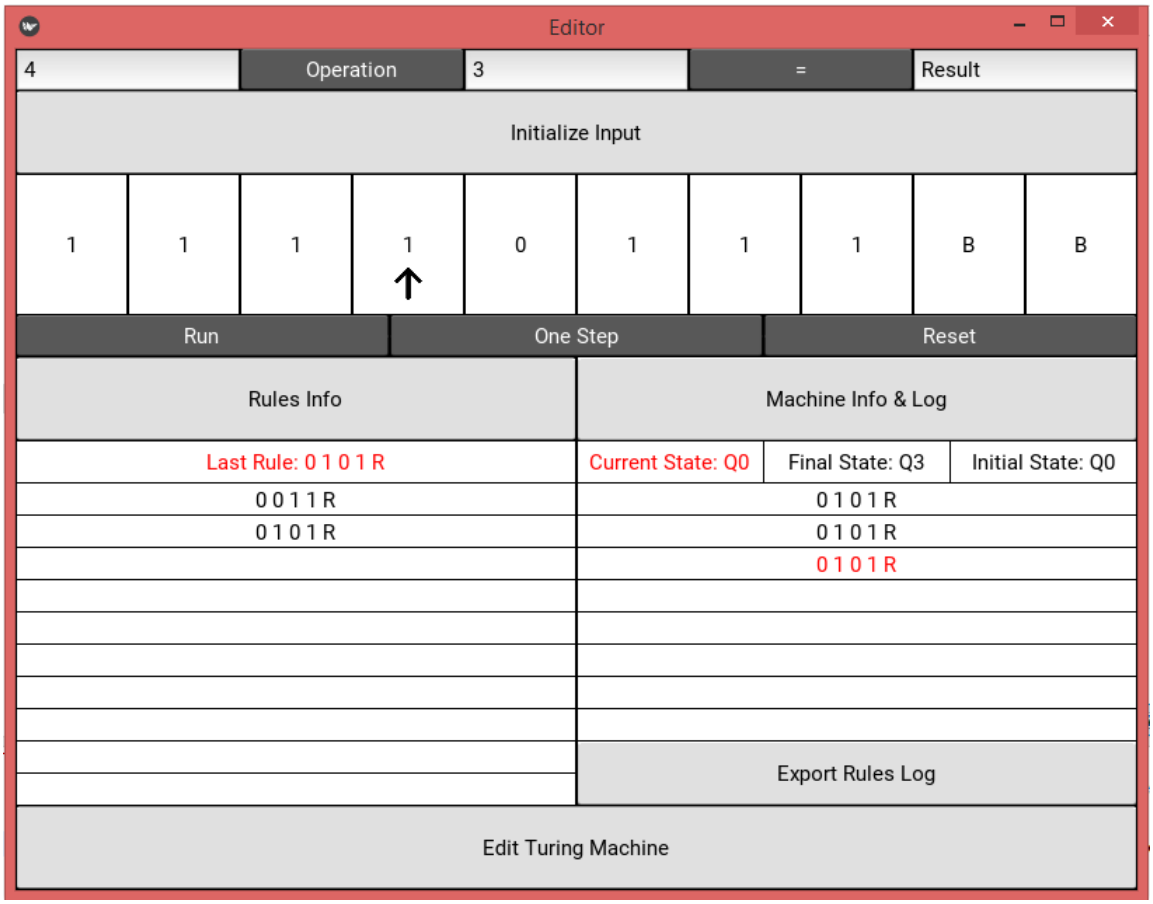

Figura 4.3: Pantalla de simulación de la interfaz

En la Figura 4.3 se puede observar la interfaz en un paso concreto de la ejecución de una máquina de Turing. Se destaca en color rojo la información más relevante para cada paso de la ejecución, concretamente la última regla leída y el estado actual en el que se encuentra la máquina. Es importante destacar que todos los posibles errores a la hora de declarar algún aspecto como por ejemplo, utilizar el operando M si N no ha sido declarado, han sido tratados y serán notificados al alumno para permitirle centrarse exclusivamente en la programación de las máquinas de Turing y abstraerse de otro tipo de tareas que no deberían de ser de su atribución.

4.1.2. Capa de negocio (Lógica)

Se procede a detallar algunas de las funciones más importantes de las dos clases principales que componen la aplicación.

Clase MainScreen

Esta clase es la encargada de gestionar toda la lógica referente a la pantalla de edición de texto de la interfaz.

El método **process** se encarga de implementar toda la lógica para inicializar y obtener todos los valores necesarios de la máquina de Turing e indicar si se ha producido algún error en el formato.

El método **parsear_xml** es la responsable de la verificación mediante XSD de la máquina de Turing importada en formato XML y del parseo de esta, mientras que la función **escribir_xml** se encarga de parsear la máquina de Turing del editor a formato XML.

Se ha utilizado la librería de python lxml tanto para parsear el fichero XML como para generar el fichero de salida en este mismo formato.

Clase Execute

Esta clase es la encargada de implementar toda la lógica de simulación de la máquina de Turing.

El método **initialize** tiene como cometido inicializar todo el sistema de simulación de la máquina en función de los operandos especificados, también notifica si se ha producido algún error.

El método **oneStep** es el encargado de simular la máquina de Turing paso a paso y de invocar los métodos necesarios para mostrar la información pertinente a cada paso de la ejecución.

Del mismo modo el método **run** se encarga de simular la máquina en un solo paso, se ha optado por una programación recursiva de este método, frente a su versión iterativa, para minimizar el coste en tiempo computacional de la aplicación.

En la figura 4.4 se muestra un extracto de código que pertenece al método escribir_xml, concretamente donde se generan las reglas de la máquina en formato XML para guardarlas en un fichero de estas características.

```
        for regla in reglas:
            inicio, lee_s, destino, escribe, move = regla.split(" ")
            transicion = etree.Element('transition')
            hijo_from = etree.Element('from')
            hijo_from.text = inicio
            hijo_to = etree.Element('to')
            hijo_to.text = destino
            hijo_lee = etree.Element('read')
            if lee_s != 'B':
                hijo_lee.text = lee_s
            hijo_escribe = etree.Element('write')
            if escribe != 'B':
                hijo_escribe.text = escribe
            hijo_move = etree.Element('move')
            hijo_move.text = move
            transicion.append(hijo_from)
            transicion.append(hijo_to)
            transicion.append(hijo_lee)
            transicion.append(hijo_escribe)
            transicion.append(hijo_move)
            automata.append(transicion)
```

Figura 4.4: escribir_xml

En la figura 4.5 podemos ver un extracto de código del método one_step, este extracto implementa la lógica de movimiento de la cinta para una computación.

```
                    if(line[r]!='R' and line[l]!='L'):
                        self.popup_RL()
                    if(line[r]=='R'):
                        self.valoresPos=self.valoresPos+1
                        if(self.tapePos==9):
                            self.tapePos=9
                        else:
                            self.tapePos=self.tapePos+1
                        if(self.valoresPos>=len(self.valores)):
                            self.valores.append('B')
                        self.imagenes[self.tapePos].source='Up-Arrow.jpg
                            '
                        if(self.tapePos%10==9):
                            i=self.tapePos
                            j=self.valoresPos
                            for i in range(9,-1,-1):
                                if(i>=len(self.valores)):
                                    self.cinta[i].text='B'
                                else:
                                    self.cinta[i].text=self.valores[j]
                                    j=j-1


                        self.information()
                        self.rules_info()
                        break
                    if(line[l]=='L'):
                        if(self.tapePos==0):
                            self.tapePos=0
                        else:
                            self.tapePos=self.tapePos-1
                        if(self.valoresPos==0):
                            self.valores.insert(0,'B')
                        if(self.valoresPos>0):
                            self.valoresPos=self.valoresPos-1
                        self.imagenes[self.tapePos].source='Up-Arrow.jpg
                            '
                        if(self.tapePos%10==0):
                            i=self.tapePos
                            j=self.valoresPos
                            for i in range(10):
                                if(i>=len(self.valores)):
                                    self.cinta[i].text='B'
                                else:
                                    self.cinta[i].text=self.valores[j]
                                    j=j+1
                        self.information()
                        self.rules_info()
                        break
```

Figura 4.5: one_step

4.1.3. Capa datos (Ficheros XML)

Las máquinas de Turing pueden ser almacenadas en formato XML, por un motivo doble: tener la información de forma estructurada y poder realizar una validación de formato a la hora de importar las máquinas y tener compatibilidad con la herramienta JFLAP. Se ha tomado parte de la estructura del XML generado por la herramienta JFLAP para que tenga compatibilidad con dicha herramienta, con ligeras modificaciones. En la figura 4.6 se observa la estructura de una máquina de Turing muy sencilla.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><structure>
        <type>turing</type>
        <automaton>
                <!--The list of states.-->
                <block id="0" name="q0">
                        <tag>Machine0</tag>
                        <initial/>
                </block>
                <block id="1" name="q1">
                        <tag>Machine1</tag>
            <final/>
                </block>
                <!--The list of transitions.-->
                <transition>
                        <from>0</from>
                        <to>0</to>
                        <read>1</read>
                        <write>1</write>
                        <move>R</move>
                </transition>
                <transition>
                        <from>0</from>
                        <to>1</to>
                        <read>0</read>
                        <write>1</write>
                        <move>R</move>
                </transition>
                <transition>
                        <from>1</from>
                        <to>1</to>
                        <read>1</read>
                        <write>1</write>
                        <move>R</move>
                </transition>
                <!--The list of automata-->
        </automaton>
</structure>
```

Figura 4.6: Estructura XML

En concreto, las características de la estructura XML original de JFLAP que se han optado por modificar son las siguientes. Por un lado, se codifica la posición de una máquina como parte de su nombre, siendo mejor separarlo al ser un diseño muy pobre. Además, a la hora de validar la estructura con un XSD, es necesario utilizar la versión 1.1 debido a las expresiones regulares que el diseño original necesita y los únicos procesadores de esquema que implementan la versión 1.1 son *Xerces* y *Saxon*, ambos basados en Java y no integrados con la librería lxml de Python.

En la figura 4.7 se observa la estructura del esquema XML.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified
    " xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="structure">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="type"/>
        <xs:element name="automaton">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="block" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="tag"/>
                    <xs:element type="xs:float" name="x" minOccurs="0"/>
                    <xs:element type="xs:float" name="y" minOccurs="0"/>
                    <xs:element type="xs:string" name="initial" minOccurs="
                        0"/>
                    <xs:element type="xs:string" name="final" minOccurs="0"
                        />
                  </xs:sequence>
                  <xs:attribute type="xs:byte" name="id" use="optional"/>
                  <xs:attribute type="xs:string" name="name" use="optional"
                      />
                </xs:complexType>
              </xs:element>
              <xs:element name="transition" maxOccurs="unbounded" minOccurs
                  ="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:byte" name="from"/>
                    <xs:element type="xs:byte" name="to"/>
                    <xs:element type="xs:string" name="read"/>
                    <xs:element type="xs:string" name="write"/>
                    <xs:element type="xs:string" name="move"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 4.7: Estructura XSD

4.2. Versión Android

Como se ha comentado en capítulos anteriores nuestra aplicación también tiene compatibilidad para dispositivos Android, facilitando así el acceso a los alumnos a la herramienta y haciéndola más atractiva. Al haber desarrollado nuestra aplicación en Python utilizando la librería Kivy, tener soporte para Android fue relativamente sencillo.

Se ha utilizado *Kivy Launcher* (Kivy (2016b)). para poder ejecutar nuestra aplicación en dispositivos Android. Es una aplicación disponible en el *Play Store* que nos permite ejecutar una aplicación desarrollada con Kivy abstrayéndonos del proceso de compilación para Android.

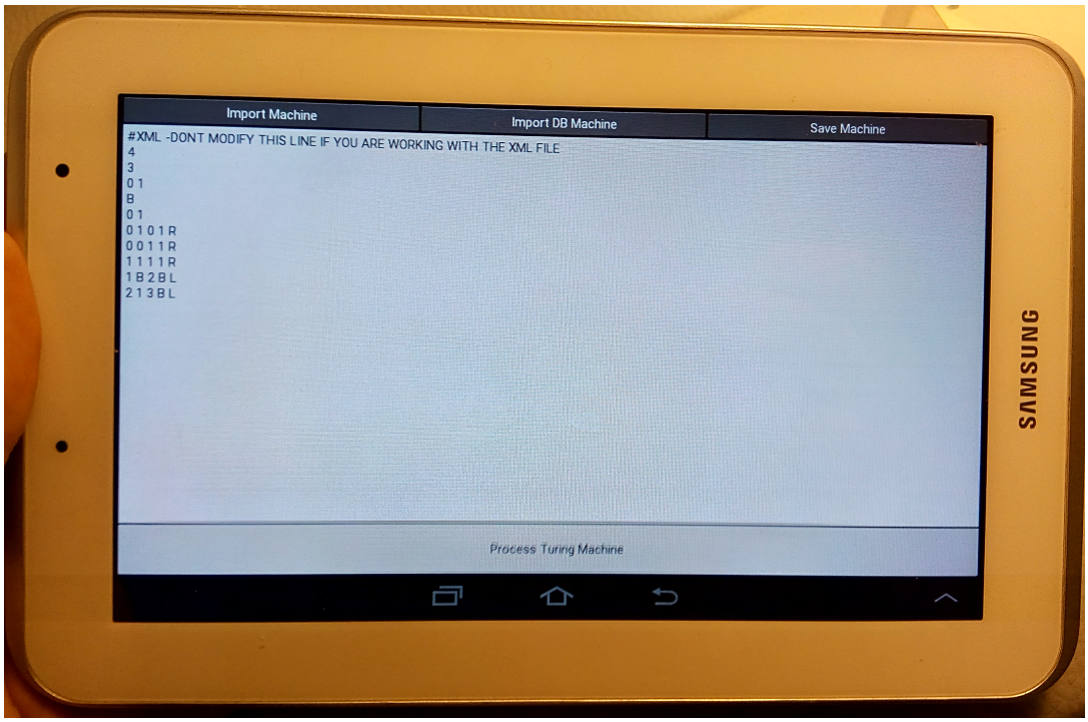

Figura 4.8: Aplicación en un dispositivo Android

En la Figura 4.8 se puede observar cómo se visualiza la aplicación en un dispositivo Android, concretamente en una tableta. Para configurar la aplicación para Android se ha necesitado especificar una serie de pre-requisitos en un manifiesto, como por ejemplo en nuestro caso, que la aplicación se visualice únicamente de forma apaisada, para aprovechar mejor el espacio en pantalla, facilitando así la labor de edición.

5 Validación y Pruebas

En cualquier proyecto software es fundamental una fase de validación, verificación y pruebas. Esta fase se ha realizado de manera iterativa e incremental a lo largo de todo el proyecto.

5.1. Pruebas unitarias

Se han realizado pruebas unitarias en cada módulo del sistema para garantizar el correcto funcionamiento de cada una de las partes de este de manera independiente. Se han desarrollado casos de prueba para cada módulo, verificando su correcto funcionamiento en las partes más críticas.

5.2. Pruebas de integración

Una vez superadas las pruebas unitarias, se realizaron las pruebas de integración, verificando así el correcto funcionamiento de cada uno de los elementos unitarios que componen cada proceso de manera común. Un ejemplo de una de las pruebas de integración realizadas fue verificar el correcto funcionamiento de las computaciones realizadas por la aplicación, dado que cada computación se podría considerar como una prueba de integración al depender de bastantes elementos.

5.3. Pruebas de rendimiento

Se realizaron pruebas de rendimiento para evaluar el tiempo máximo de ejecución al realizar la computación de las máquinas con el metódo run(). Durante el desarrollo del código se primó reducir lo máximo posible el tiempo computacional de los algoritmos aplicando los conceptos aprendidos a lo largo de la titulación como el uso de estructuras de datos eficiente.

5.4. Pruebas de aceptación

Las pruebas de aceptación verifican que los requisitos impuestos por el cliente han sido satisfechos.

Éstas se realizaron con el visto bueno del cliente en cada una de las iteraciones realizadas hasta validar el correcto funcionamiento del sistema final. Hubiese sido interesante haber podido mostrar el producto final y realizar encuestas a los alumnos de la asignatura Modelos de Cálculo mientras esta asignatura se estaba impartiendo, pero como la fecha de realización de este proyecto no coincide en calendario con la de la asignatura no ha sido posible. Lo que sí se ha hecho es enseñar el producto a lo largo de varias fases del desarrollo a algunos ex-alumnos de la asignatura para observar cuáles son sus opiniones del producto, las cuales se han tenido en cuenta a lo largo del desarrollo de la parte gráfica de la aplicación.

6
Conclusiones

El desarrollo de este proyecto tenía un objetivo claro, desarrollar una aplicación que ayude a la hora de enseñar el concepto de máquinas de Turing y facilite el proceso de aprendizaje. El proyecto se ha realizado aplicando una metodología de desarrollo iterativa e incremental, realizando de forma satisfactoria cada de una de las fases del ciclo de vida del software hasta alcanzar el resultado final con el cumplimiento de los objetivos base marcados para este proyecto, los cuales se definieron en el apartado de Objetivos en el capítulo 1.

Por lo tanto las metas alcanzadas a lo largo de este proyecto han sido:

- Creación de una interfaz gráfica para la creación y simulación de máquinas de Turing.
- Visualización de computaciones de máquinas de Turing concretas, visualizando para cada computación toda la información pertinente a ella y el estado de la máquina, permitiendo exportar un log de ejecuciones en cualquier momento.
- Ejecución paso a paso, permitiendo la posibilidad de depurar la máquina, visualizando exactamente dónde está el error y permitiendo eliminar, modificar o añadir nuevas reglas en cualquier momento de la simulación, sin perder la funcionalidad de ejecutar una máquina de forma directa para comprobar el resultado en cualquier momento de la simulación.
- Introducción de los operandos deseados para simular las máquinas y la opción de reiniciar la simulación en cualquier momento.
- Se ha eliminado la necesidad de modificar el input de una manera determinada como había que hacer anteriormente con el script.
- Integración de un editor de texto para las máquinas y la parte de simulación en una misma aplicación.

- Posibilidad de salvar y recuperar máquinas de Turing en diferentes formatos, como .XML guardándolas como información estructurada.
- Validación de formato de las máquinas de Turing programadas antes de ser simuladas.
- Soporte para Android.
- Compatibilidad con la aplicación JFLAP.

Las Tecnologías utilizadas durante el proyecto para alcanzar estas metas han sido principalmente *Python* y *Kivy*, tecnología no impartida durante la titulación, para desarrollar todo el código y la interfaz gráfica y *XML* para tener las máquinas guardadas de forma estructurada. Entre las herramientas cabe destacar *Kivy Launcher* con la cual se ha podido ejecutar la aplicación en un dispositivo Android.

Bibliografía

[A. Rodríguez 2008] A. Rodríguez, E. del C. J.J. Castro-Schez: *SELFA-Pro: Añadiendo nuevas entidades y funcionalidades para mejorar la enseñanza y el aprendizaje de TALF. Actas de las XIV Jornadas de Enseñanza Universitaria de la Informática (JENUI2008).* http://bioinfo.uib.es/~joemiro/aenui/procJenui/Jen2008/p541_ARodriguez.pdf, 2008. – [Online; accessed 3-March-2016]

[Barwise u. Etchemendy 1993] Barwise, Jon ; Etchemendy, John: Turing's World 3.0 for the Macintosh. (1993)

[Copeland 2002] Copeland, B J.: The Church-Turing thesis. In: *Stanford encyclopedia of philosophy* (2002)

[Kivy 2016a] Kivy: *Kivy*. https://kivy.org/, 2016. – [Online; accessed 3-March-2016]

[Kivy 2016b] Kivy: *Kivy Launcher*. https://kivy.org/docs/guide/packaging-android.html#packaging-your-application-for-kivy-launcher, 2016. – [Online; accessed 3-March-2016]

[LXML 2016] LXML: *LXML*. http://lxml.de/, 2016. – [Online; accessed 3-March-2016]

[Naps u. a. 2002] Naps, Thomas L. ; Rössling, Guido ; Almstrum, Vicki ; Dann, Wanda ; Fleischer, Rudolf ; Hundhausen, Chris ; Korhonen, Ari ; Malmi, Lauri ; McNally, Myles ; Rodger, Susan: Exploring the role of visualization and engagement in computer science education. In: *ACM Sigcse Bulletin* Bd. 35 ACM, 2002, S. 131–152

[Python 2016] Python: *Python*. https://www.python.org/, 2016. – [Online; accessed 3-March-2016]

[Rodger 2006] Rodger, Susan: Learning automata and formal languages interactively with JFLAP. In: *ACM SIGCSE Bulletin* 38 (2006), Nr. 3, S. 360–360

[Sipser 2006] Sipser, Michael: *Introduction to the Theory of Computation*. Bd. 2. Thomson Course Technology Boston, 2006

[Sommerville 2005] Sommerville, Ian: *Ingeniería del software*. Pearson Educación, 2005

[Stone u. a. 2005] Stone, Debbie ; Jarrett, Caroline ; Woodroffe, Mark ; Minocha, Shailey: *User interface design and evaluation*. Morgan Kaufmann, 2005

[Taylor 1998] Taylor, Ralph G.: Models of computation and formal languages. (1998)

[Turing 1936] Turing, Alan M.: On computable numbers, with an application to the Entscheidungsproblem. In: *J. of Math* 58 (1936), Nr. 345-363, S. 5

[w3schools 2016] w3schools: *XML Introduction*. http://www.w3schools.com/xml/xml_whatis.asp, 2016. – [Online; accessed 3-March-2016]

A

Manual de Usuario

Este apéndice se presenta como un manual de usuario para la aplicación que se ha desarrollado, en el cual se ilustrará la simulación de una máquina de Turing, concretamente la operación suma, importándola como un fichero .XML.

- **Paso 1:** Ejecutar la aplicación.

  Al ejecutar la aplicación lo primero que aparecerá será la ventana de edición (Figura A.1) donde se nos presentan varias opciones. Programar la máquina de Turing manualmente, importarla como un fichero de texto plano o como un fichero .XML. También se nos presenta la opción de guardar la máquina de Turing presente en el editor en el formato deseado.

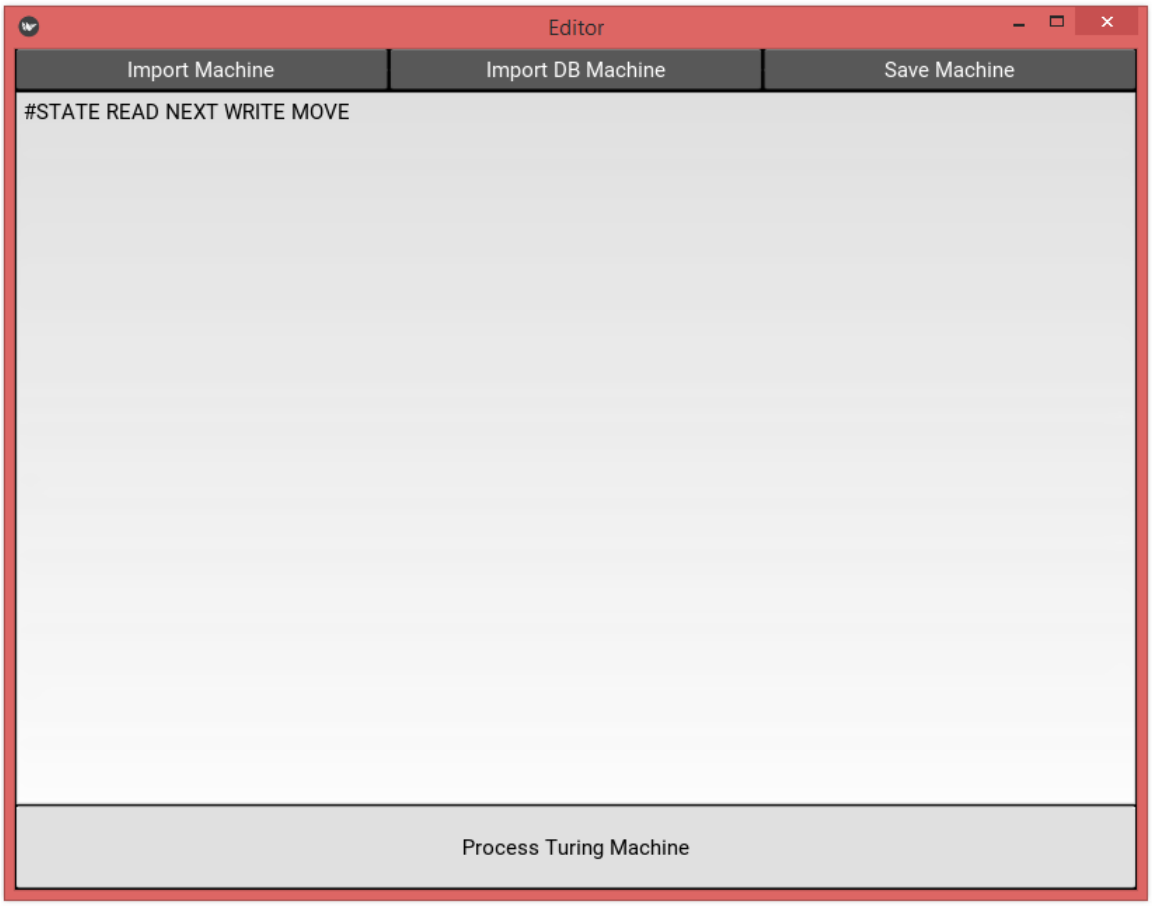

Figura A.1: Pantalla inicial

- **Paso 2:** Importar una máquina de Turing.

  Seleccionamos la opción Import DB Machine, para importar las máquinas de Turing que hemos guardado en formato .XML, en este caso solo tenemos la máquina que realiza la operación suma. En la Figura A.2 observamos la aplicación en este punto.

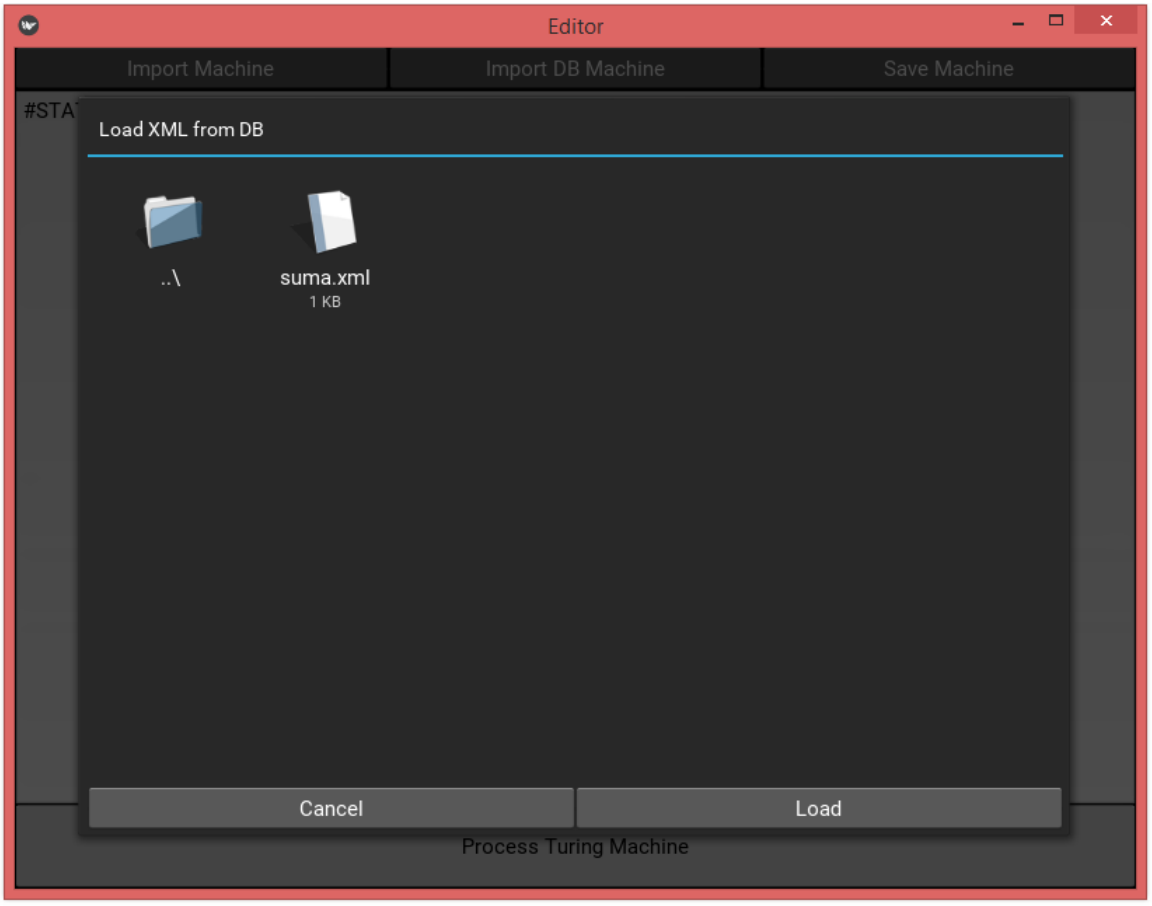

Figura A.2: Importar máquina de Turing

- **Paso 3:** Visualización de la máquina importada

  Una vez importada la máquina aparece automáticamente parseada en el editor de texto en el formato demandado por el responsable de la asignatura como observamos en la Figura A.4. Si el alumno ha modificado la estructura .XML y no es validada frente al schema, se notificará al alumno y la máquina no se importará.

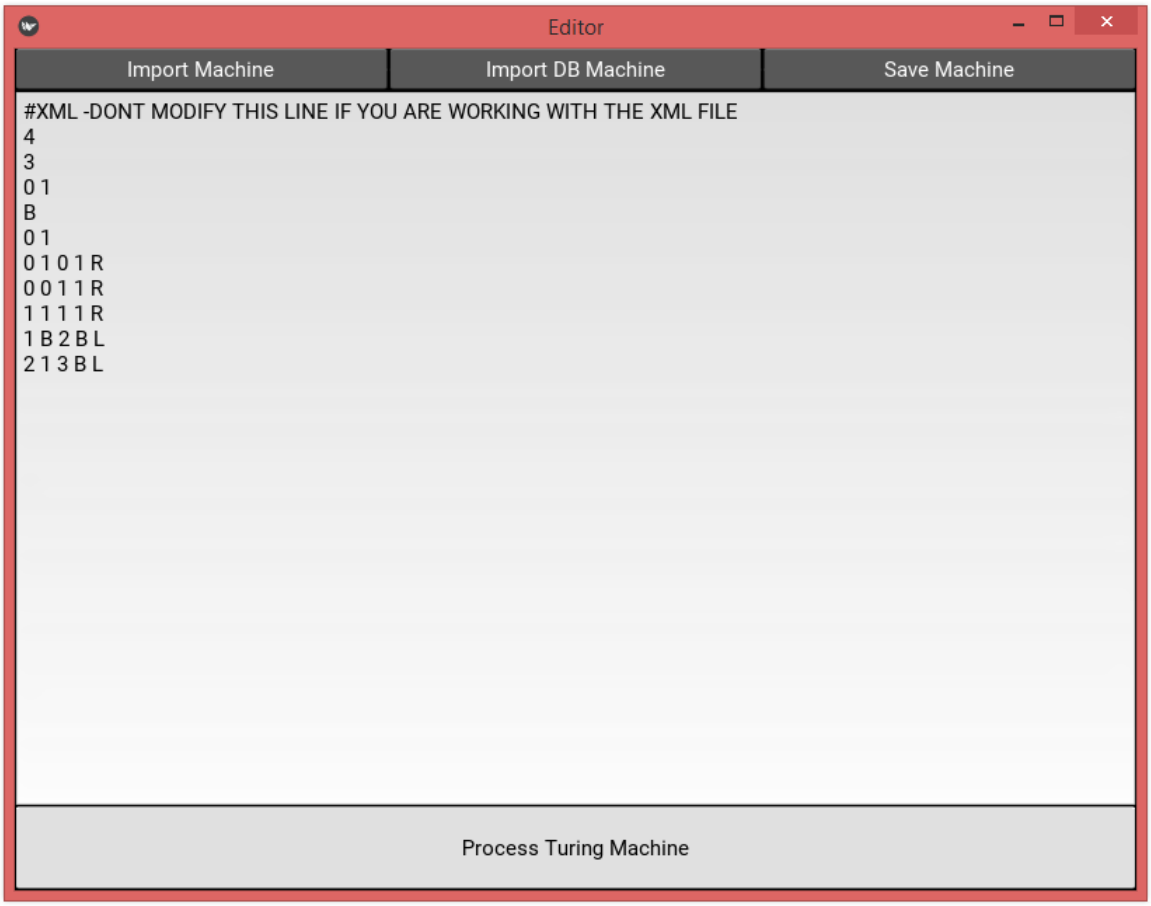

Figura A.3: Visualización de la máquina importada

- **Paso 4:** Procesamiento de la máquina

  Si presionamos la opción de Process Turing Machine, el sistema realizará una validación de formato por si el alumno ha cometido algún error modificando la máquina si el formato es el adecuado como observamos en la Figura A.4

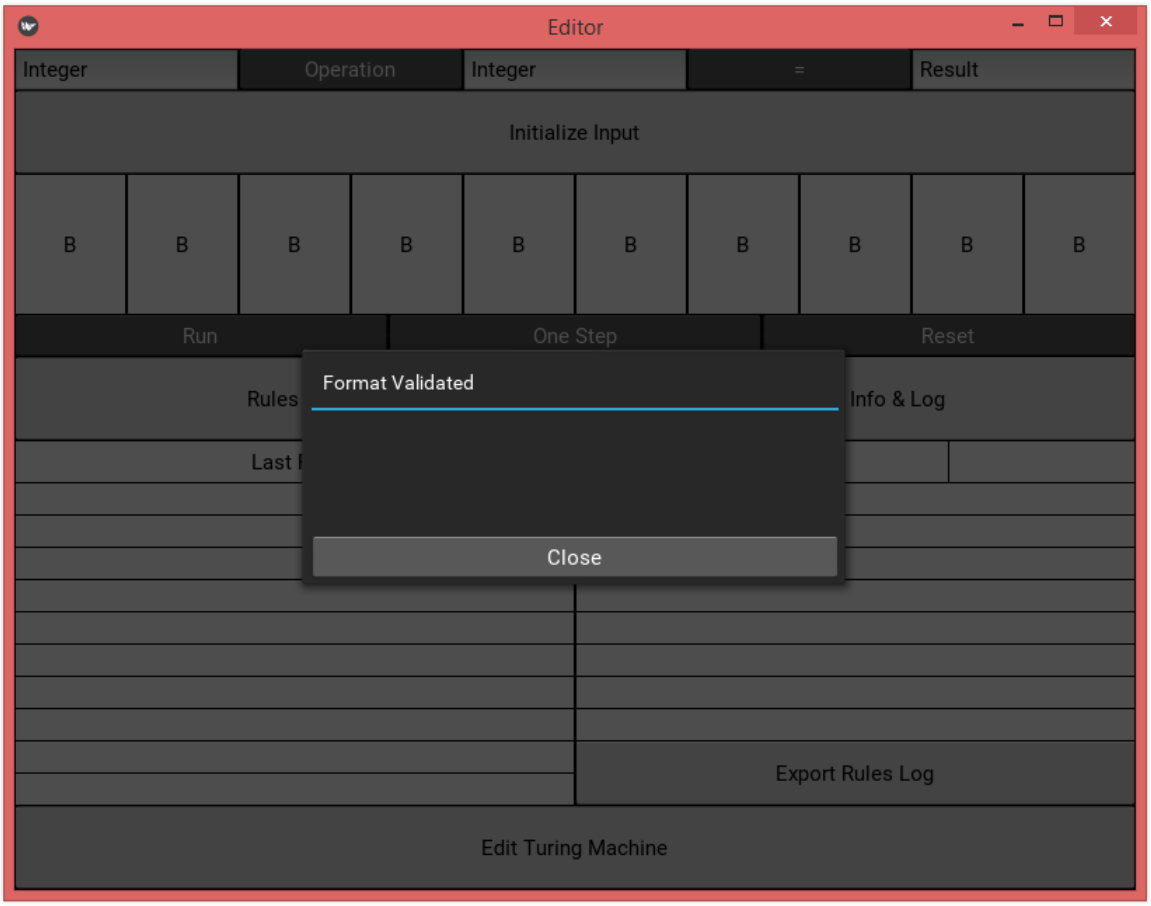

Figura A.4: Procesamiento y validación

- **Paso 5:** Inicializar entrada

  El siguiente paso es introducir una entrada, en este ejemplo hemos utilizado los operandos 3 y 2 y hemos presionado la opción Initialize Input. El sistema introducirá en la cinta la entrada utilizando una codificación unaria, como observamos en la Figura A.5. Una vez inicializada podemos observar en la parte izquierda la información de las reglas que se pueden utilizar en función del estado en el que nos encontremos ($Q0$ en este caso) y la información necesaria de la máquina en cada paso de la ejecución.

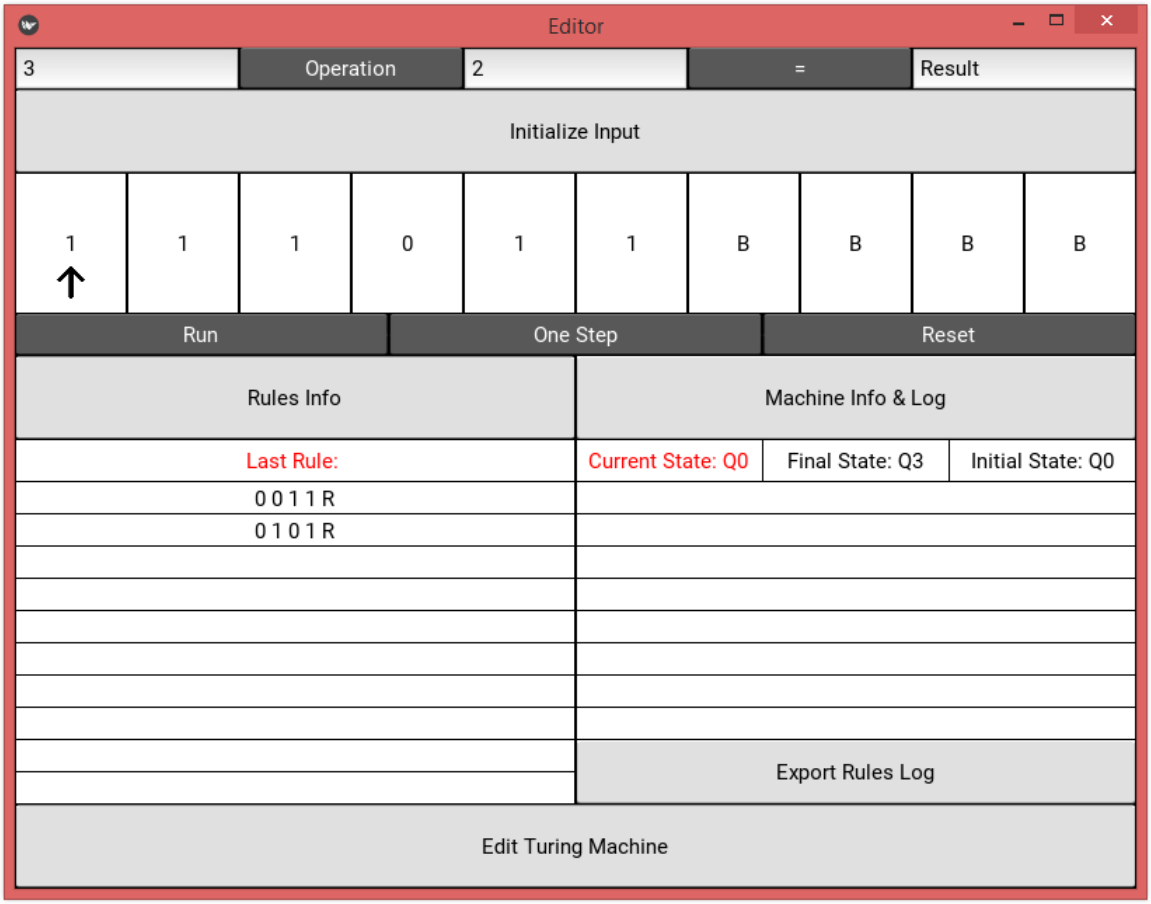

Figura A.5: Inicializar entrada

- **Paso 6:** Simulación

  En este punto tenemos varias opciones, ejecutar directamente la máquina con la opción Run, ejecutarla paso a paso con la opción One Step para ver detenidamente el funcionamiento de la máquina y localizar errores en la creación de las reglas por parte del alumno, reiniciar la máquina al estado inicial con la opción Reset o volver a editar la máquina con la opción Edit Turing Machine para añadir, eliminar o modificar reglas si es necesario. Si elegimos esta última opción puede que no sea necesario reiniciar el estado de la máquina si el alumno se ha dado cuenta del error antes de llegar a este, debido a que cuando se vuelva a la pantalla de simulación la máquina sigue almacenando el estado en el que se encontraba, algo muy útil para no repetir los mismos pasos de la simulación que han sido correctos de nuevo.

  En la Figura A.6 se observa como han cambiado los valores en la cinta, el cabezal, las reglas disponibles en la parte izquierda al haber cambiado de estado (el cambio de estado se observa en la parte de la derecha). También podemos observar el log de ejecución de las reglas, y la posibilidad de exportar este log en cualquier momento de la simulación. La información pertinente de cada paso se resalta en rojo.

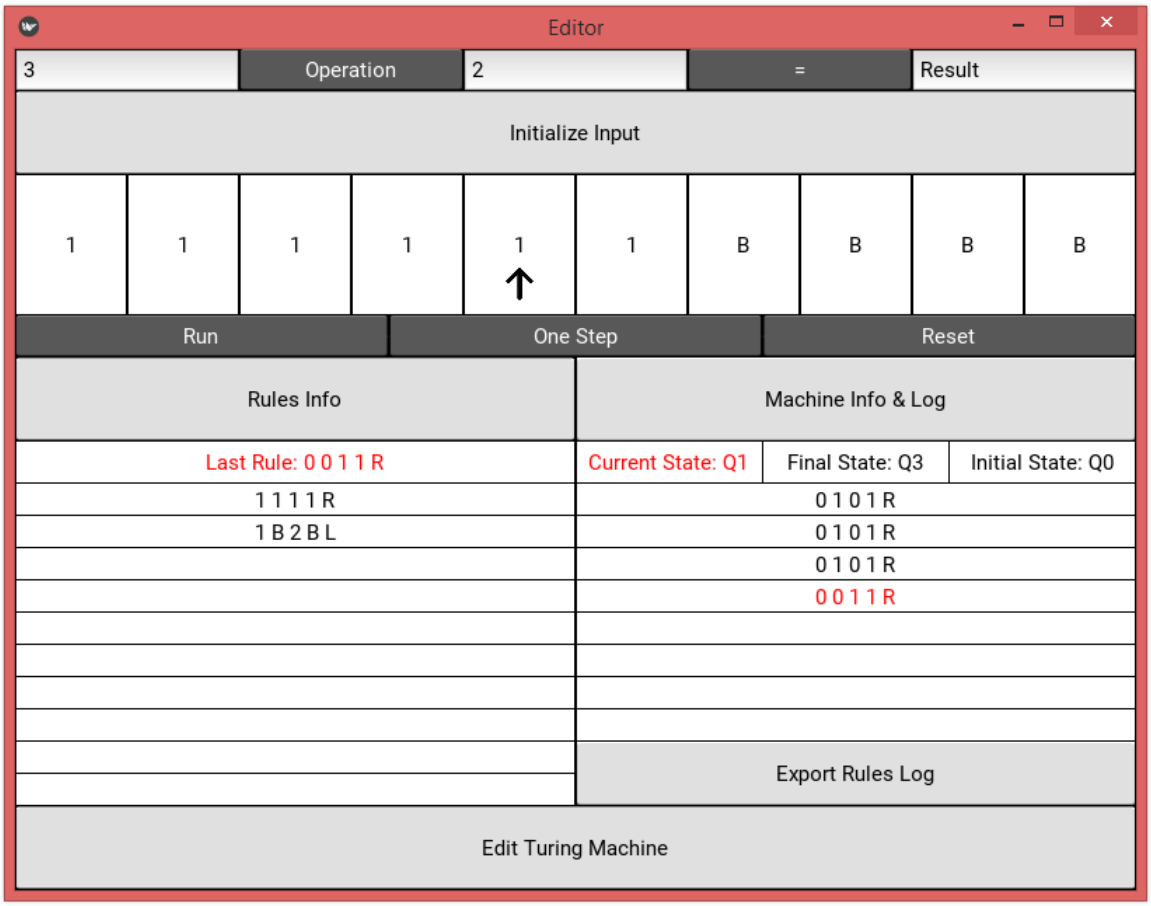

Figura A.6: Captura de la simulación en un momento dado

- **Paso 7:** Ejecución finalizada

  Una vez se ha llegado al estado final, la simulación de la máquina termina notificándole al alumno el fin de la simulación como se observa en la Figura A.7 e indicando el resultado en la parte superior derecha de la herramienta.

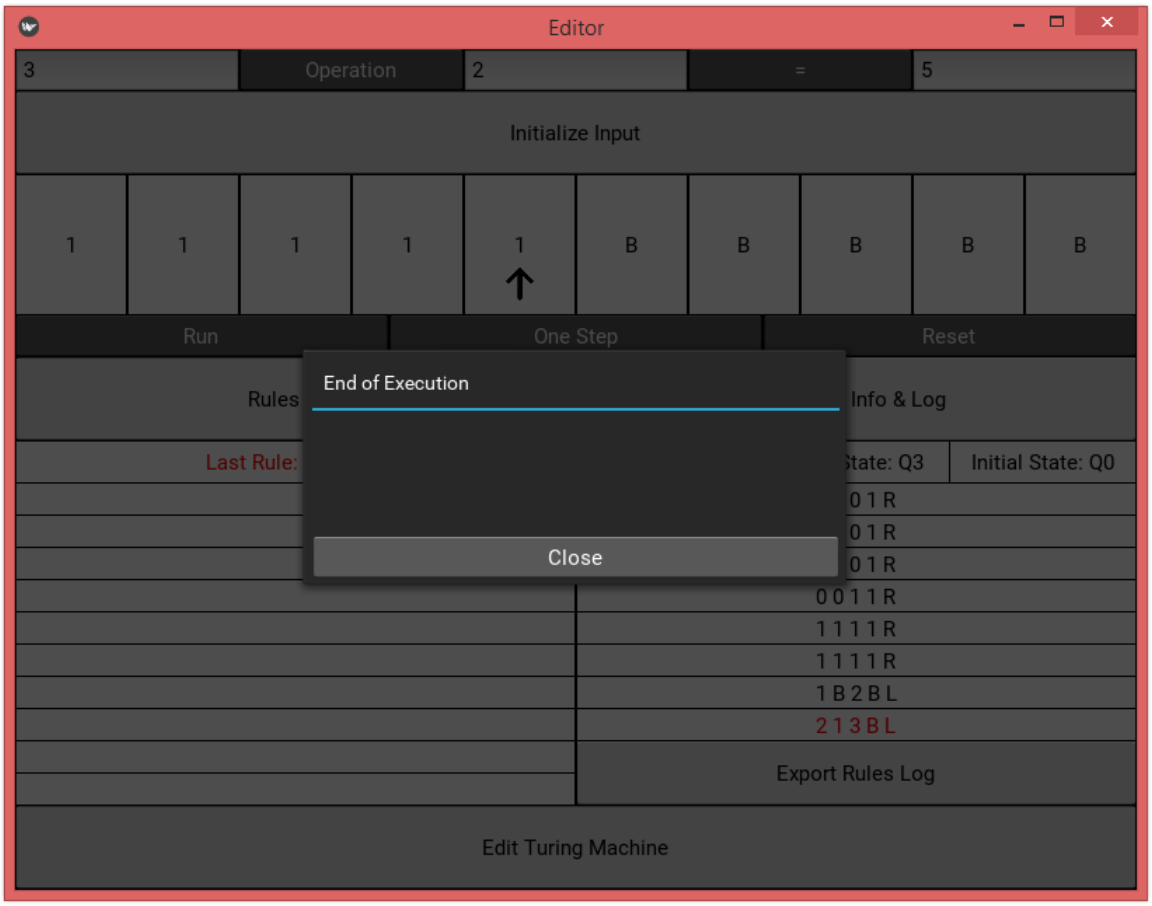

Figura A.7: Ejecución finalizada