



Universidad  
de La Laguna

Escuela Superior de  
Ingeniería y Tecnología  
Sección de Ingeniería Informática

# Trabajo de Fin de Grado

---

Calculadora para Lenguajes Formales  
*Formal Language Calculator*

Germán Paz Méndez

---

La Laguna, 5 de septiembre de 2016

D<sup>a</sup>. **Gara Miranda Valladares**, con N.I.F. 78.563.584-T profesora Ayudante Doctor adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

## C E R T I F I C A

Que la presente memoria titulada:

*“Calculadora para Lenguajes Formales”*

ha sido realizada bajo su dirección por D. **Germán Paz Méndez**, con N.I.F. 45.850.420-N.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de septiembre de 2016

# Agradecimientos

Este Trabajo Fin de Grado se lo quiero dedicar a mi familia, la cual siempre ha estado ahí en todo momento aportándome apoyo y confianza para hacerlo posible.

También quería agradecer a mi tutora, Gara Miranda Valladares, su atención y dedicación a lo largo de este Trabajo Fin de Grado.

Finalmente quiero dar un agradecimiento especial a mis amigos que siempre me han dado todo el apoyo moral que necesitaba, tanto en los malos como en los buenos momentos.

# Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.

## Resumen

*El objetivo de este trabajo ha sido diseñar e implementar una aplicación web que funcione como Calculadora para Lenguajes Formales. Dicha aplicación permite la especificación de cadenas y lenguajes formales para, posteriormente, poder realizar sobre ellos algunas de las principales operaciones básicas sobre lenguajes. Para ello serán fundamentales los conocimientos en “lenguajes formales” y “teoría de la computación” obtenidos en la asignatura: ‘Computabilidad y Algoritmia’ y desarrollados en otras asignaturas como: ‘Complejidad Computacional’, ‘Diseño y Análisis de Algoritmos’ impartidas, la mayor parte de las mismas, en el itinerario de computación del Grado en Ingeniería Informática de la Universidad de La Laguna.*

*La herramienta desarrollada permite al usuario definir lenguajes y realizar operaciones básicas sobre los mismos, tales a: unión, intersección, concatenación, diferencia, igualdad, sublenguaje, potencia, inversa, cierre de kleene y cierre positivo. La herramienta no se limita a mostrar el resultado de las operaciones sino que ofrece una visualización sencilla e intuitiva que ayuda a entender la forma en que se aplican las distintas operaciones. Es por ello que emplear este tipo de herramienta educativa fomentará una menor intervención por parte del profesorado y una mayor facilidad por parte del alumnado para interpretar los conocimientos que deberá adquirir.*

*La Calculadora para Lenguajes Formales (FLCalc) queda disponible online para que pueda servir en un futuro como herramienta fundamental para la docencia de lenguajes formales.*

**Palabras clave:** Aplicación web, aplicación híbrida, diseño adaptativo, herramienta educativa

## Abstract

The aim of this study was to design and implement a web application that works as a Formal Languages Calculator. Such application allows the specification of strings and formal languages to then be able to perform on them some of the main basic operations on languages. It will be fundamental the knowledge in “formal languages” and “theory of computation” obtained in the subject: ‘Computability and Algorithmics’ and developed in other subjects such as: ‘Computational Complexity’, ‘Design and Analysis of Algorithms’ given, most of the same ones, in the itinerary of computation of the Degree in Computer Engineering of the University of La Laguna.

The developed tool allows to the user to define languages and to perform basic operations such as: union, intersection, concatenation, difference, equality, sublanguage, power, inverse, kleene closure and positive closure. The tool does not limit itself to showing the result of the operations but it offers a simple and intuitive visualization that helps to understand the form in which the different operations are applied. Therefore this type of educational applications foster a less intervention by teachers and greater ease by students to interpret the knowledge they have to acquire.

The Calculator for Formal Languages (FLCalc) is available online in order that it might serve in the future as a fundamental tool for teaching formal languages.

**Keywords:** *Web application, hybrid application, adaptive design, tool educational*

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Conceptos básicos . . . . .	3
1.2. Operaciones con cadenas . . . . .	4
1.2.1. Concatenación y potencia . . . . .	4
1.2.2. Igualdad . . . . .	4
1.2.3. Prefijos, sufijos, subcadenas y subsecuencias . . . . .	4
1.2.4. Inversa . . . . .	5
1.3. Operaciones con lenguajes . . . . .	5
1.3.1. Concatenación y potencia . . . . .	5
1.3.2. Unión e intersección . . . . .	6
1.3.3. Sublenguajes e igualdad de lenguajes . . . . .	7
1.3.4. Cierre de Kleene y cierre positivo . . . . .	7
1.3.5. Diferencia, complemento e inversa . . . . .	8
1.4. Actividades y tareas . . . . .	9
1.5. Plan de trabajo . . . . .	9
<b>2. Estado del Arte</b>	<b>11</b>
2.1. Herramientas para lenguajes formales . . . . .	11
2.1.1. SELFA pro . . . . .	11
2.1.2. JFLAP . . . . .	12
2.1.3. Noam . . . . .	13
2.2. Herramientas para conjuntos . . . . .	15
2.2.1. Wolframalpha . . . . .	15
2.3. Herramientas de decodificación . . . . .	16
2.3.1. Regexper . . . . .	16
2.4. Comparativa de las herramientas . . . . .	17
<b>3. Diseño e implementación de la herramienta</b>	<b>18</b>
3.1. Requisitos . . . . .	18
3.1.1. Aplicaciones nativas . . . . .	18
3.1.2. Aplicaciones web . . . . .	19
3.1.3. Aplicaciones híbridas . . . . .	20
3.2. Tecnologías . . . . .	21
3.2.1. Navegadores . . . . .	21

3.2.2.	HTML5	23
3.2.3.	CSS3	23
3.2.4.	JavaScript	23
3.2.5.	JQuery	24
3.2.6.	Twitter Bootstrap	25
3.2.7.	Git y Github	25
3.2.8.	Latex y ShareLatex	26
3.2.9.	PhoneGapp	27
3.2.10.	Atom	27
3.2.11.	Cloud9	28
<b>4.</b>	<b>FLCalc</b>	<b>29</b>
4.1.	¿Qué es FLLCalc?	29
4.2.	Requisitos	30
4.3.	Completando los pasos	30
4.3.1.	Paso 1	30
4.3.2.	Paso 2	31
4.3.3.	Paso 3	33
4.3.4.	Paso 4	34
4.4.	Avisos	35
4.5.	Mensajes de error	35
4.6.	Operaciones	36
4.6.1.	Potencia del lenguaje	37
4.6.2.	Inversa del lenguaje	39
4.6.3.	Cierre de Kleene del lenguaje	41
4.6.4.	Cierre Positivo del lenguaje	43
4.6.5.	Concatenación entre lenguajes	45
4.6.6.	Unión entre lenguajes	47
4.6.7.	Intersección entre lenguajes	49
4.6.8.	Diferencia entre lenguajes	51
4.6.9.	Sublenguaje entre lenguajes	53
4.6.10.	Igualdad entre lenguajes	55
<b>5.</b>	<b>Conclusiones y trabajos futuros</b>	<b>57</b>
5.1.	Conclusiones	57
5.2.	Trabajos futuros	58
<b>6.</b>	<b>Conclusions and future work</b>	<b>60</b>
6.1.	Conclusions	60
6.2.	Future work	61
<b>7.</b>	<b>Presupuesto</b>	<b>63</b>
7.1.	Presupuesto	63



**Bibliografía**

# Índice de figuras

1.1. Actividades del TFG . . . . .	10
2.1. Aplicación web SELFA pro . . . . .	11
2.2. Ventana inicial de JFLAP . . . . .	12
2.3. Herramienta web FSM simulator . . . . .	14
2.4. Herramienta web FSM2regex . . . . .	15
2.5. Página web de Regexper . . . . .	16
3.1. Navegadores . . . . .	22
3.2. HTML 5 . . . . .	23
3.3. CSS 3 . . . . .	23
3.4. JavaScript . . . . .	24
3.5. JQuery . . . . .	24
3.6. Bootstrap 3 . . . . .	25
3.7. GitHub . . . . .	26
3.8. Latex . . . . .	26
3.9. PhoneGap . . . . .	27
3.10. Atom . . . . .	27
3.11. Cloud9 . . . . .	28
4.1. Paso 1 - Símbolo . . . . .	30
4.2. Paso 1 - Alfabeto . . . . .	31
4.3. Paso 1 - Introducción del alfabeto . . . . .	31
4.4. Paso 2 - Cadena . . . . .	32
4.5. Paso 2 - Lenguaje . . . . .	32
4.6. Paso 2 - Introducción del lenguaje . . . . .	33
4.7. Paso 3 - Operaciones . . . . .	33
4.8. Paso 4 - Resultado . . . . .	35
4.9. Avisos . . . . .	35
4.10. Mensajes de error . . . . .	36
4.11. Operación - Potencia . . . . .	37
4.12. Resultado - Potencia . . . . .	38
4.13. Operación - Inversa . . . . .	39
4.14. Resultado - Inversa . . . . .	40
4.15. Operación - Cierre de Kleene . . . . .	41

4.16. Resultado - Cierre de Kleene . . . . .	42
4.17. Operación - Cierre Positivo . . . . .	43
4.18. Resultado - Cierre positivo . . . . .	44
4.19. Operación - Concatenación . . . . .	45
4.20. Resultado - Concatenación . . . . .	46
4.21. Operación - Unión . . . . .	47
4.22. Resultado - Unión . . . . .	48
4.23. Operación - Intersección . . . . .	49
4.24. Resultado - Intersección . . . . .	50
4.25. Operación - Diferencia . . . . .	51
4.26. Resultado - Diferencia . . . . .	52
4.27. Operación - Sublenguaje . . . . .	53
4.28. Resultado - Sublenguaje . . . . .	54
4.29. Operación - Igualdad . . . . .	55
4.30. Resultado - Igualdad . . . . .	56

# Índice de tablas

1.1. Actividades del TFG . . . . .	9
2.1. Comparativa de herramientas . . . . .	17
3.1. Aplicaciones nativas . . . . .	19
3.2. Aplicaciones web . . . . .	20
3.3. Aplicaciones híbridas . . . . .	21
3.4. Compatibilidad en distintos navegadores . . . . .	22
7.1. Tiempo dedicado y presupuesto . . . . .	63
7.2. Hardware utilizado . . . . .	64
7.3. Software utilizado . . . . .	64

# Capítulo 1

## Introducción

El presente Trabajo Fin de Grado (TFG) se enmarca en el ámbito de la asignatura “*Computabilidad y Algoritmia*”. Esta asignatura pertenece al bloque de formación básica y se imparte en el primer cuatrimestre del segundo curso. El módulo principal de esta asignatura se dedica al estudio de la teoría de autómatas y de los lenguajes formales. La teoría de autómatas y lenguajes formales se puede ubicar en el campo científico de la Informática Teórica, un campo clásico y multidisciplinar dentro de los estudios universitarios de Informática. Es un campo clásico debido no sólo a su antigüedad (anterior a la construcción de los primeros ordenadores) sino, sobre todo, a que sus contenidos principales no dependen de los rápidos avances tecnológicos que han hecho que otras ramas de la Informática deban adaptarse a los nuevos tiempos a un ritmo vertiginoso. Es multidisciplinar porque en sus cimientos encontramos campos tan variados como la lingüística, las matemáticas o la electrónica.

El hecho de que esta materia no haya sufrido grandes cambios en las últimas décadas no le resta interés dentro de un plan de estudios de una Ingeniería Informática puesto que el estudio de las máquinas secuenciales que abarca la teoría de autómatas, por una parte, sienta las bases de la algoritmia y permite modelar y diseñar soluciones para un gran número de problemas. Por otra parte, permite abordar cuestiones de gran interés en el campo de la informática como qué tipo de problemas pueden ser resueltos por un computador o incluso, en caso de existir una solución computable para un problema, cómo podemos medir la calidad (en términos de eficacia) de dicha solución. Es decir, el estudio de la teoría de autómatas que se lleva a cabo en el marco de la asignatura “*Computabilidad y Algoritmia*” es la puerta que nos permite la entrada hacia campos tan interesantes como la computabilidad y la complejidad algorítmica. Además, una de las principales aportaciones del estudio de los lenguajes formales, sobre todo desde un punto de vista práctico, es su contribución al diseño de lenguajes de programación y a la construcción de sus correspondientes traductores, cuestión de especial importancia para estudiantes de Ingeniería Informática.

Teniendo en cuenta entonces el ámbito de la asignatura “*Computabilidad y Algoritmia*” nos centraremos en el módulo dedicado al estudio de los autómatas y los lenguajes formales. De hecho, y tal y como su nombre indica, la asignatura “*Computabilidad y Algoritmia*” está compuesta por dos módulos principales: uno de “Computabilidad” y otro dedicado a la “Algoritmia”. Al primero de estos módulos se le dedican las primeras 10 semanas del cuatrimestre y en él se tratan los contenidos relacionados con los autómatas y los lenguajes formales. Durante la impartición de este módulo se realizan un total de 8 prácticas cuya finalidad principal es la de diseñar, testear y visualizar el comportamiento de expresiones regulares, autómatas finitos deterministas, autómatas finitos no deterministas, gramáticas, y máquinas de Turing. Para ello, el alumnado crea y diseña sus propios programas para simular y testear el comportamiento de estos mecanismos, o bien, se utilizan herramientas específicas de simulación que se utilizan en este ámbito. En la parte introductoria del módulo de “Computabilidad” se presentan los conceptos básicos sobre alfabetos, cadenas y lenguajes formales. También se introducen algunas de las operaciones básicas sobre cadenas y/o lenguajes:

- Inversa
- Concatenación
- Unión
- Intersección
- Diferencia
- Sublenguajes
- Igualdad de lenguajes
- Potencia
- Cierre de Kleene y Cierre Positivo

Para ilustrar el funcionamiento de estas operaciones sería muy útil el poder disponer de alguna herramienta gráfica que representara paso a paso cada una de las operaciones realizadas sobre cadenas y/o lenguajes. La herramienta podría entenderse como una “calculadora online” para hacer operaciones básicas sobre cadenas y lenguajes formales. Esto es, la herramienta deberá presentar una interfaz visual muy sencilla e intuitiva que, a modo de calculadora, permita al usuario introducir las cadenas y/o lenguajes con los que operar y luego seleccionar las operaciones que realizar sobre los mismos. En relación a la especificación y manejo de lenguajes infinitos, habrá que hacer un análisis riguroso

sobre cómo representar adecuadamente estos lenguajes para luego poder operar con ellos de manera sencilla y eficaz. La herramienta tendría además un especial interés educativo si, además de mostrar los resultados de las operaciones, incluyera también algún mecanismo visual para la representación del proceso seguido para realizar el cálculo correspondiente.

Teniendo en cuenta lo anterior, el objetivo de este Trabajo Fin de Grado será el **diseño e implementación de una aplicación web o una aplicación móvil que funcione como calculadora para lenguajes formales**. La aplicación deberá permitir la especificación de cadenas y lenguajes formales para, posteriormente, poder realizar sobre ellos algunas de las principales operaciones básicas. La interfaz debería ser muy funcional, ilustrativa y atractiva a nivel visual, para que pueda servir en un futuro como herramienta fundamental para la docencia de lenguajes formales.

A continuación presentare los conceptos básicos sobre autómatas y lenguajes formales siguiendo las siguientes referencias [1–3].

## 1.1. Conceptos básicos

Un *símbolo* es un ente abstracto que no se puede definir formalmente. Letras o dígitos son ejemplos que frecuentemente usan símbolos.

Una *cadena* (o *palabra*) es una secuencia finita de símbolos. Por ejemplo:  $a$ ,  $b$  y  $c$  son símbolos y  $abcb$  es una cadena. La *longitud* de una cadena  $w$ , denotada por  $|w|$ , es el número de símbolos que componen la cadena. Por ejemplo:  $abcb$  tiene longitud 4. La cadena vacía, denotada por  $\epsilon$ , es la cadena que contiene cero símbolos. Así  $|\epsilon| = 0$ .

Un *alfabeto*  $\Sigma$  es un conjunto no vacío y finito de símbolos.

Un *lenguaje* (formal) es un conjunto de cadenas. El lenguaje vacío  $\emptyset$  es el lenguaje compuesto por ninguna cadena. Decimos que el lenguaje vacío no es el mismo que el que consta de la cadena vacía:  $\emptyset \neq \{\epsilon\}$ . El conjunto de *palíndromos* (cadenas que se leen igual de izquierda a derecha que de derecha a izquierda) sobre el alfabeto  $\{0, 1\}$  es un lenguaje infinito. Otro lenguaje es el conjunto de todas las cadenas sobre un alfabeto fijo  $\Sigma$ . Denotamos a este lenguaje por  $\Sigma^*$ . Por ejemplo, si  $\Sigma = \{a\}$ , entonces  $\Sigma^* = \{\epsilon, a, aa, aaa, \dots\}$ .

El *lenguaje universal* de  $\Sigma$  es el lenguaje compuesto por todas las cadenas sobre el alfabeto  $\Sigma$ . Denotamos al lenguaje universal por  $\Sigma^*$ . Dado  $\Sigma = \{a\}$ , entonces  $\Sigma^* = \{\epsilon, a, aa, aaa, \dots\}$ .

## 1.2. Operaciones con cadenas

### 1.2.1. Concatenación y potencia

Sean  $w$  y  $z$  cadenas sobre cualquier  $\Sigma$ . La *concatenación* de  $w$  con  $z$  es la cadena que se obtiene al añadir a la cadena  $w$  la cadena  $z$ . Se denota como  $wz$  o  $w \cdot z$ .

**Ejemplo 1.2.1.1.** Sea  $w = abra$  y  $z = cadabra$ . Si aplicamos la concatenación obtenemos  $wz = abracadabra$ .

#### Definición 1.2.1.2.

- $|wz| = |w| + |z|$
- $\epsilon$  es la identidad para la concatenación:  $\epsilon \cdot w = w \cdot \epsilon = w$

Sean  $w$  una cadena y  $n \in \mathbb{N}$ . La *potencia* de una cadena sobre un alfabeto se define como:

$$w^n = \begin{cases} \epsilon & \text{si } n = 0 \\ ww^{n-1} & \text{si } n > 0 \end{cases}$$

**Ejemplo 1.2.1.3.** Si  $w = aba$  es una cadena sobre el alfabeto  $\Sigma = \{a, b\}$ :

$$w^0 = \epsilon$$

$$w^1 = aba$$

$$w^2 = abaaba$$

$$w^3 = abaabaaba$$

### 1.2.2. Igualdad

Si  $w$  y  $z$  son cadenas, se dice que  $w = z$  si tienen la misma longitud ( $|w| = |z|$ ) y los mismos símbolos en la misma posición.

**Ejemplo 1.2.2.1.** Sea  $\Sigma = \{\alpha, \beta\}$ :

$$\alpha\beta\beta = \alpha\beta\beta$$

$$\alpha\beta\beta \neq \beta\beta\alpha$$

### 1.2.3. Prefijos, sufijos, subcadenas y subsecuencias

Un *prefijo* de una cadena  $\alpha$  es cualquier secuencia con los primeros  $n$  símbolos de la cadena (donde  $0 \leq n \leq |\alpha|$ ), y un *sufijo* es cualquier secuencia con los últimos  $n$  símbolos de la cadena.



**Ejemplo 1.2.3.1.** la cadena  $abc$  tiene prefijos  $\epsilon, a, ab$  y  $abc$ ; y los sufijos son  $\epsilon, c, bc$  y  $abc$ .

**Definición 1.2.3.2.** Los **prefijos propios** son aquellos que no son iguales a la cadena.

Sea  $w, x, y, z \in \Sigma^*$ . Se dice que  $w$  es *subcadena* de  $z$  si existen las cadenas  $x$  e  $y$  para las cuales  $z = xwy$ .

**Ejemplo 1.2.3.3.** Sea  $z = abc$ . Subcadenas de  $z$  son:  $\epsilon, a, b, c, ab, bc$  y  $abc$ .

Una *subsecuencia* es cualquier cadena formada mediante la eliminación de cero o más símbolos no necesariamente contiguos a una cadena.

**Ejemplo 1.2.3.4.** La cadena  $bada$  es una subsecuencia de  $bandera$ .

## 1.2.4. Inversa

Dada una cadena  $w \in \Sigma^*$ , su inversa o traspuesta se define como:

$$w^I = \begin{cases} w & \text{si } w = \epsilon \\ y^I a & \text{si } w = ay, \text{ con } a \in \Sigma \wedge y \in \Sigma^* \end{cases}$$

**Ejemplo 1.2.4.1.** Sea  $w = atar$ , entonces la inversa de  $w$  es:

$$\begin{aligned} w^I &= (atar)^I \\ &= (tar)^I a \\ &= (ar)^I ta \\ &= (r)^I ata \\ &= (\epsilon)^I rata \\ &= \epsilon \cdot rata \\ &= rata \end{aligned}$$

## 1.3. Operaciones con lenguajes

### 1.3.1. Concatenación y potencia

Sean  $L_1$  y  $L_2$  lenguajes sobre un alfabeto  $\Sigma$ , la *concatenación o producto cartesiano* se define como:

$$L_1 \cdot L_2 = L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$$

**Ejemplo 1.3.1.1.** Si  $L_1 = \{\text{ojos}\}$  y  $L_2 = \{\text{azules, negros}\}$ , entonces:  
 $L_1L_2 = \{\text{ojosazules, ojosnegros}\}$

### Definición 1.3.1.2.

- Si  $L_1$  es un lenguaje sobre  $\Sigma_1$  y  $L_2$  es un lenguaje sobre  $\Sigma_2$ , entonces  $L_1L_2$  es un lenguaje sobre  $\Sigma_1 \cup \Sigma_2$
- Para cualquier lenguaje  $L$ , se tiene que:  $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$

Sean  $L$  un lenguaje sobre  $\Sigma$ , se define la *potencia* del lenguaje como:

$$L^n = \begin{cases} \{\epsilon\} & \text{si } n = 0 \\ L \cdot L^{n-1} & \text{si } n > 0 \end{cases}$$

**Ejemplo 1.3.1.3.** Si  $L = \{0, 1\}$ :

$$L^0 = \{\epsilon\}$$

$$L^1 = L = \{0, 1\}$$

$$L^2 = L \cdot L = \{00, 01, 10, 11\}$$

$$L^3 = L \cdot L^2 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

...

## 1.3.2. Unión e intersección

Sean  $L_1$  y  $L_2$  lenguajes sobre un alfabeto  $\Sigma$ , definimos la *unión* como:

$$L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$$

Y la *intersección* como:

$$L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$$

**Ejemplo 1.3.2.1.** Si  $\Sigma = \{0, 1\}$ ,  $L_1 = \{\epsilon, 0, 1, 10, 11\}$  y  $L_2 = \{\epsilon, 1, 0110, 11010\}$ , entonces:

$$L_1 \cup L_2 = \{\epsilon, 0, 1, 10, 11, 0110, 11010\}$$

$$L_1 \cap L_2 = \{\epsilon, 1\}$$

### 1.3.3. Sublenguajes e igualdad de lenguajes

Sean  $L_1$  y  $L_2$  lenguajes sobre un alfabeto  $\Sigma$ . Si todas las cadenas de  $L_1$  son también cadenas de  $L_2$  se dice que  $L_1$  es un *sublenguaje* de  $L_2$  ( $L_1 \subseteq L_2$ ).

**Ejemplo 1.3.3.1.** Para los lenguajes  $L_1 = \{a, aa, aaa, aaaa, aaaaa\}$  y  $L_2 = \{a^n \mid n = 0, 1, 2, \dots\}$ , se tiene que  $L_1 \subseteq L_2$

#### Definición 1.3.3.2.

- Cualquier lenguaje  $L$  sobre el alfabeto  $\Sigma$  es un sublenguaje de  $\Sigma^*$ :  $L \subseteq \Sigma^*$
- Se dice que dos lenguajes  $L_1$  y  $L_2$  son **iguales** si contienen exactamente las mismas cadenas, por tanto,  $L_1 = L_2$  si y sólo si  $L_1 \subseteq L_2$  y  $L_2 \subseteq L_1$

### 1.3.4. Cierre de Kleene y cierre positivo

Sea  $L$  un lenguaje sobre un alfabeto  $\Sigma$ . Definimos el **Cierre de Kleene** (o cierre estrella) de  $L$  como:

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

Y el **Cierre positivo** de  $L$  como:

$$L^+ = \bigcup_{n=1}^{\infty} L^n$$

**Ejemplo 1.3.4.1.** Supongamos que  $L = \{a\}$ , entonces:

$$\begin{aligned} L^0 &= \{\epsilon\} \\ L^1 &= \{a\} \\ L^2 &= \{aa\} \\ &\dots \end{aligned}$$

Y, por tanto:

$$\begin{aligned} L^* &= \{\epsilon, a, aa, aaa, \dots\} \\ L^+ &= \{a, aa, aaa, \dots\} \end{aligned}$$

#### Definición 1.3.4.2.

- Las cadenas de  $L^*$  se forman al realizar **cero o más** concatenaciones de las cadenas del lenguaje.
- Las cadenas de  $L^+$  se forman realizando **una o más** concatenaciones de las cadenas del lenguaje.

**Recordatorio 1.3.4.3.** Un lenguaje universal sobre algún alfabeto  $\Sigma$ , o cierre de  $\Sigma$ , es el lenguaje que contiene todas las cadenas que es posible formar con los símbolos de  $\Sigma$  y se denota como  $\Sigma^*$ .

**Ejemplo 1.3.4.4.**

Sea  $\Sigma = \{a\}$ , entonces  $\Sigma^* = \{\varepsilon, a, aa, aaa, \dots\}$ .

**Propiedades:**

- Si  $L$  es un lenguaje sobre  $\Sigma$ ,  $L \subseteq \Sigma^*$
- Si  $L$  es un lenguaje sobre  $\Sigma$ ,  $L^n \subseteq \Sigma^*$ ,  $\forall n \in \mathbb{N}$ .
- $L^* \subseteq \Sigma^*$ .
- $L^+ \subseteq \Sigma^*$ .
- Puesto que  $\emptyset^0 = \{\varepsilon\}$  y  $\emptyset^n = \emptyset$ ,  $\forall n \geq 1$ , entonces:
  - $\emptyset^* = \{\varepsilon\}$
  - $\emptyset^+ = \emptyset$

### 1.3.5. Diferencia, complemento e inversa

Sea  $L_1$  y  $L_2$  lenguajes sobre un alfabeto  $\Sigma$ , definimos la **diferencia** como:

$$L_1 - L_2 = \{w \mid w \in L_1 \wedge w \notin L_2\}$$

**Ejemplo 1.3.5.1.** Consideremos  $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Definamos  $L_1$  como el lenguaje formado por las cadenas que no contienen ninguno de los dígitos 2, 3, ..., 9.

Definamos  $L_2$  como el lenguaje formado por las cadenas de ceros.

Entonces  $L_1 - L_2$  es el lenguaje formado por las cadenas de ceros y unos que tienen al menos un uno

Sea  $L$  un lenguaje sobre un alfabeto  $\Sigma$ , definimos el **complementario o complemento del lenguaje** sobre el alfabeto como:

$$\overline{L} = \Sigma^* - L$$

**Ejemplo 1.3.5.2.** Consideremos  $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Definamos  $L$  como el lenguaje formado por las cadenas que no contienen ninguno de los dígitos 2, 3, ..., 9.

Entonces  $\overline{L}$  es el lenguaje formado por las cadenas que contienen al menos uno de los dígitos 2, 3, ..., 9

Sea  $L$  un lenguaje sobre un alfabeto  $\Sigma$ , definimos la **inversa del lenguaje** como:

$$L^I = L^{-1} = \{w^I \mid w \in L\}$$

**Ejemplo 1.3.5.3.** Si  $L = \{sala, eva\}$  entonces:

$$L^I = L^{-1} = \{alas, ave\}$$

Obsérvese que:

$$(L^I)^I = L$$

$$(L_1 \cdot L_2)^I = L_2^I \cdot L_1^I$$

## 1.4. Actividades y tareas

Con el fin de diseñar una herramienta que permita realizar un conjunto básico de operaciones sobre cadenas y/o lenguajes formales, será necesario abordar las tareas o actividades siguientes:

	Objetivo	Descripción
ACT1	Revisión bibliográfica	Análisis de herramientas similares que existan para la docencia de lenguajes formales, o en su defecto, para la docencia o para el manejo de conjuntos en general.
ACT2	Diseño de la herramienta	Análisis de los mecanismos más adecuados para la especificación y representación de lenguajes. En esta fase también se realizará un análisis de las tecnologías existentes para realizar el desarrollo de la herramienta.
ACT3	Implementación de la herramienta	Implementación de la calculadora para lenguajes formales, haciendo especial énfasis en el aspecto visual de la misma, la sencillez de uso y la forma en la que se ilustran las operaciones realizadas.
ACT4	Generación de ejemplos	Diseñar y definir una serie de ejemplos de alfabetos, cadenas y lenguajes, así como, la realización de operaciones básicas sobre los mismos.
ACT5	Implementación de la aplicación móvil	Implementar una versión móvil para la herramienta diseñada.
ACT6	Documentación	Elaboración del material de soporte para el uso de la herramienta así como la memoria del TFG y demás documentación requerida durante la realización de la asignatura de TFG.

Tabla 1.1: Actividades del TFG

## 1.5. Plan de trabajo

Atendiendo al conjunto de actividades principales que conformarán este TFG, la temporalización estimada de las mismas se recoge en el siguiente plan de trabajo:

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16 S18
ACT1																
ACT2																
ACT3																
ACT4																
ACT5																
ACT6																

Figura 1.1: Actividades del TFG

El plan de trabajo comprende las 15 semanas del segundo cuatrimestre (del 3 de febrero de 2016 al 23 de mayo de 2016), reflejadas en la Figura 1.1 como S1, S2, ... S15; más las semanas que comprenden la convocatoria de exámenes correspondiente, representadas en la Figura 1.1 como S16-S18.

# Capítulo 2

## Estado del Arte

El presente capítulo introduce el panorama de herramientas existentes que proporcionan una visión semejante a la Calculadora para Lenguajes Formales.

Toda aplicación desarrollada en este sentido sirve principalmente como ayuda para el aprendizaje de los conceptos básicos en la materia de “Computabilidad y Algoritmia” del Grado en Ingeniería Informática de la Universidad de La Laguna.

### 2.1. Herramientas para lenguajes formales

#### 2.1.1. SELFA pro

SELFA pro [4] es una herramienta online que resuelve ejercicios sobre autómatas finitos, expresiones regulares, gramáticas regulares, autómatas con pila y gramáticas libres de contexto. Además se pueden realizar simulaciones sobre los autómatas finitos y los autómatas con pila.

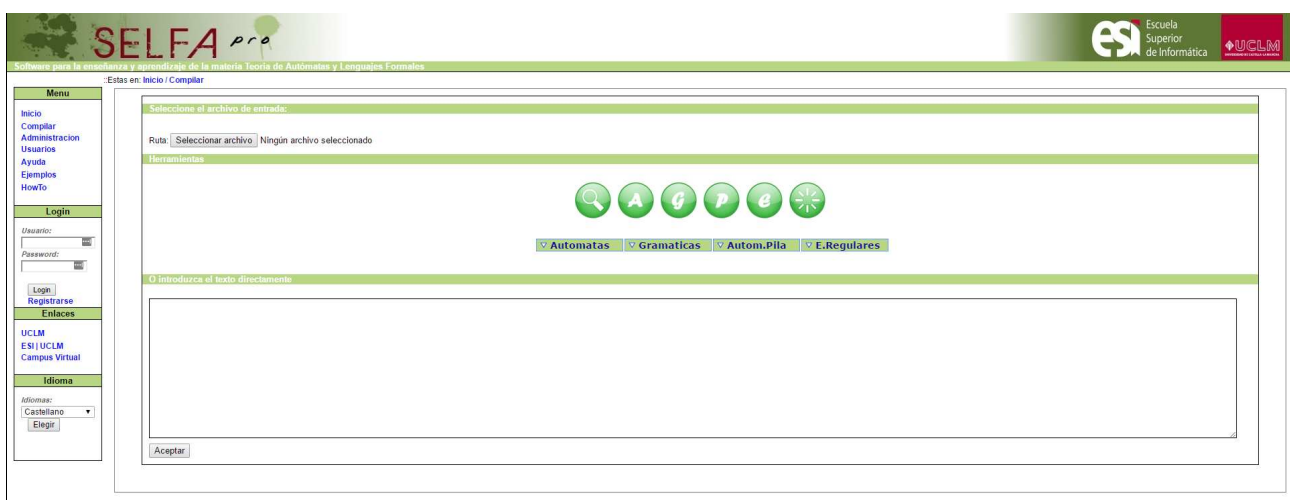


Figura 2.1: Aplicación web SELFA pro

Permite que los alumnos se entrenen en todos los contenidos exclusivos de este tipo de materias, a excepción de las máquinas de Turing, mejorando la calidad tanto en el aprendizaje como en la enseñanza.

Esta herramienta ha sido realizada por D. Julian Hortolano Villarejo (Selfa), D. Alfredo Rodríguez Sánchez (Selfa Pro) y D. Raúl Miguel Sabariego (resolución de bugs sobre Selfa Pro), todos ellos alumnos de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha.

Además su diseño está basado en un procesador de lenguajes y la interacción con la misma se realiza a través de un lenguaje formal.

Esta herramienta permite seguir la línea de enseñanza del tipo de materias que se ocupan de los “lenguajes formales” y la “teoría de la computación”. No permite definir lenguajes ni realizar ningún tipo de operación entre lenguajes.

### 2.1.2. JFLAP

JFLAP [5] es un paquete de herramientas gráficas diseñado principalmente como ayuda en el aprendizaje de los conceptos básicos en “Computabilidad y Algoritmia”. Se trata posiblemente de la herramienta más sofisticada y evolucionada de este tipo.



Figura 2.2: Ventana inicial de JFLAP

Inició su andadura como una serie de herramientas en el Instituto Politécnico de Rensselaer en torno a 1990, con los estudiantes que trabajan bajo la dirección



de Susan Rodger. Más tarde el proyecto JFLAP se trasladó a la Universidad de Duke en 1994, cuando se trasladó allí Rodger. Actualmente esta herramienta se distribuye bajo un tipo de licencia Creative Commons que no admite la libertad de uso con fines comerciales. Su característica más notable es que dispone de una interfaz gráfica que permite al usuario introducir autómatas y visualizar la salida de los diferentes algoritmos.

Esta herramienta permite crear y operar sobre autómatas (finitos, máquinas de Moore y Mealy, Turing, . . .), gramáticas, expresiones regulares y L-systems. Además, pretende ser una herramienta fácil de usar, intuitiva y práctica en todo momento. No permite definir lenguajes ni realizar ningún tipo de operación entre lenguajes.

### 2.1.3. Noam

Noam [6] es una biblioteca JavaScript para trabajar con autómatas y gramáticas formales para lenguajes regulares y libres de contexto. El nombre de Noam proviene de Noam Chomsky y su jerarquía de lenguajes formales y gramáticas.

Un conjunto de herramientas web desarrolladas con Noam son FSM2REGEX [7] y FSM simulator [8]. Básicamente FSM simulator trata de ser una herramienta que simula visualmente el funcionamiento de una máquina de estados finitos dada una cadena de una entrada. Mientras que FSM2REGEX permite la conversión de una expresión regular a un autómata finito no determinista y viceversa.

# FSM simulator

Visually simulate your DFAs, NFAs and  $\epsilon$ -NFAs one input symbol at a time!

### ① Create automaton

Input regex

Enter a regular expression into the input field below or click **Generate random regex** to have the app generate a simple regex randomly for you. Next, click **Create automaton** to create a FSM for the defined regex and display its transition graph.

A valid regex consists of alphanumeric characters representing the set of input symbols (e.g. `a`, `B`, `9`), the `$` character representing the empty string, the choice operator `|`, the Kleene operator `*`, and parentheses `(` and `)`. An example of a valid regex is: `(a+B)*(c9+$)+$`.

or write your own

### ② Simulate automaton

Enter a sequence of input symbols into the input field below or click **Random string**, **Acceptable string** and **Unacceptable string** to have the app generate random acceptable and unacceptable sequences for you.

Click **Read next** to have the FSM consume the next input symbol in the sequence and **Read all** to consume all remaining input symbols. Click **Step backward** to go back one symbol and **Reset** to reset the FSM and go back to the beginning of the input sequence.

The input field highlights the input symbol that will be read next.

---

### ③ Transition graph

The FSM being simulated is displayed in the form of a transition graph. The nodes representing the current states of the FSM are colored in ■.

### ④ What's next?

This is just the beginning! If you like learning about and playing with FSMs and regexes, check out these other Web apps:

- [Regular Expressions Gym](#) - A Web app that simplifies your regular expressions by detecting and removing sub-expressions that generate the same strings.
- [FSM2Regex](#) - A Web app that converts finite-state machines to regular expressions and regular expressions to finite-state machines.
- [Regexper](#) - A Web app that displays regular expressions as railroad diagrams.
- [Grammophone](#) - A Web app for analyzing and transforming context-free grammars, e.g. for generating strings from a grammar and computing SLR/LR/LALR parsing tables.
- [Debuggex](#) - A Web app that is a visual regular expression debugger, tester, and helper.

### ⑤ Feedback

Love the application, hate it, found a bug, or have a feature idea? I'd love to hear about it! Please send your feedback via the [noam project issues page on GitHub](#).

FSM simulator is a demo of using [noam](#), a JavaScript library for working with finite-state machines, grammars and regular expressions.

Created by [Ivan Zuzak](#) and [Vedrana Jankovic](#). Built with [Noam](#), [Bootstrap](#), [Viz.js](#), and [jQuery](#).

Code available on [GitHub](#) and licensed under [Apache License v2.0](#).

Figura 2.3: Herramienta web FSM simulator

## FSM2Regex

Convert your FSMs to regexes  
and your regexes to FSMs!

**① Create automaton**

Enter a **FSM** below and the application will convert and show the equivalent regular expression. Alternately, enter a **regular expression** and the application will convert and show the equivalent FSM.

**Input automaton**

Enter a FSM into the input field below or click **Generate random DFA/NFA/eNFA** to have the app generate a simple FSM randomly for you. Next, click **Create automaton** to display the FSM's transition graph.

A valid FSM definition contains a list of states, symbols and transitions, the initial state and the accepting states. States and symbols are alphanumeric character strings and can not overlap. Transitions have the format: `stateA: symbol: stateB, stateC`. The `$` character is used to represent the empty string symbol (epsilon) but should not be listed in the alphabet. Generate a FSM to see a valid example.

or write your own

**Input regex**

Enter a regular expression into the input field below or click **Generate random regex** to have the app generate a simple regex randomly for you. Next, click **Create automaton** to create a FSM for the defined regex and display its transition graph.

A valid regex consists of alphanumeric characters representing the set of input symbols (e.g. `a`, `B`, `0`), the `$` character representing the empty string, the choice operator `|`, the Kleene operator `*`, and parentheses `(` and `)`. An example of a valid regex is: `(a+b)*(c|d)+$`.

or write your own

---

**② Transition graph**

The FSM being converted is displayed in the form of a transition graph.

**③ What's next?**

This is just the beginning! If you like learning about and playing with FSMs and regexes, check out these other Web apps:

- [FSM simulator](#) - A Web app that visually simulates the step-by-step execution of finite-state machines
- [Regular Expressions Gym](#) - A Web app that simplifies your regular expressions by detecting and removing sub-expressions that generate the same strings.
- [Regexper](#) - A Web app that displays regular expressions as railroad diagrams.
- [Grammophone](#) - A Web app for analyzing and transforming context-free grammars, e.g. for generating strings from a grammar and computing SLR/LR/LALR parsing tables.
- [Debugex](#) - A Web app that is a visual regular expression debugger, tester, and helper.

**④ Feedback**

Love the application, hate it, found a bug, or have a feature idea? I'd love to hear about it! Please send your feedback via the [noam project issues page on GitHub](#).

FSM2Regex is a demo of using [noam](#), a JavaScript library for working with finite-state machines, grammars and regular expressions.

Created by Ivan Zuzak. Built with [Noam](#), [Bootstrap](#), [Vz.js](#), and [jQuery](#).

Code available on [GitHub](#) and licensed under [Apache License v2.0](#).

Figura 2.4: Herramienta web FSM2regex

Al igual que en nuestra Calculadora para Lenguajes Formales y en la senda de las herramientas anteriormente nombradas estas herramientas creadas con Noam pretenden ser herramientas de uso específico, atractivas, fáciles de usar, intuitivas y gratuitas. Noam no permite definir lenguajes ni realizar ningún tipo de operación entre lenguajes.

## 2.2. Herramientas para conjuntos

### 2.2.1. Wolframalpha

Wolframalpha [9] presenta una nueva forma de obtener conocimiento y respuestas haciendo cálculos dinámicos sobre una gran base de colección de datos, algoritmos y métodos incorporados.

La teoría de conjuntos está estrechamente asociada con la rama de las matemáticas conocida como la lógica. Wolframalpha proporciona una herramienta web con la que probar el conjunto de operaciones sobre conjuntos.

Con respecto a nuestra Calculadora para Lenguajes Formales podemos encontrar varias similitudes en el conjunto de operaciones que se pueden realizar,

entre las que destacamos:

- Unión de conjuntos.
- Diferencia de conjuntos.
- Intersección de conjuntos.
- Complemento de conjuntos.
- Subconjuntos de un conjunto.

## 2.3. Herramientas de decodificación

### 2.3.1. Regexper

Regexper [10] es una sencilla y simple herramienta web que permite introducir expresiones regulares y decodificarlas, o visualizarlas, en un formato de flujo de trabajo. El objetivo final es eliminar el procesamiento complicado de una expresión regular y exponerlo de una forma visualmente atractiva y fácil de ver.

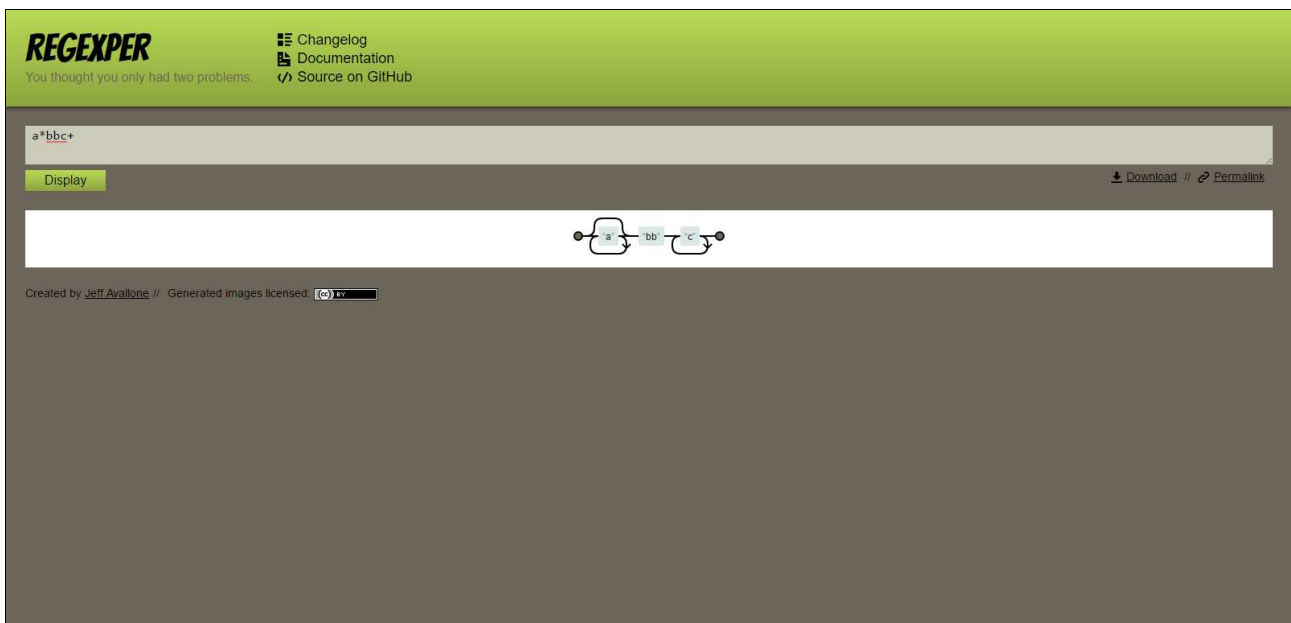


Figura 2.5: Página web de Regexper

Esta herramienta sirve de inspiración para el desarrollo de nuestra Calculadora para Lenguajes Formales ya que resulta muy cómoda de utilizar y otorga al que la utilice una mejor interpretación de las expresiones regulares.

## 2.4. Comparativa de las herramientas

A continuación, en la Tabla 2.1, ofrecemos una comparativa de las herramientas analizadas.

Herramientas	Operaciones con conjuntos	Operaciones con lenguajes	Simulación de autómatas	Simulación de gramáticas	Simulación de expresiones regulares
SELFA pro	-	-	✓	-	-
JFLAP	-	-	✓	✓	✓
Noam	-	-	✓	-	-
Wolframalpha	✓	-	-	-	-
Regexper	-	-	-	-	✓
FLCalc	-	✓	-	-	-

Tabla 2.1: Comparativa de herramientas

# Capítulo 3

## Diseño e implementación de la herramienta

Hay que tener muy presente a la hora de realizar un Trabajo Fin de Grado cuál es la finalidad de éste. Este capítulo habla en general de los requisitos y las tecnologías usadas, los cuales son el punto de referencia que nos permitirá alcanzar, de la forma más satisfactoria posible, el resultado deseado.

### 3.1. Requisitos

Como requisito general se plantea que la aplicación web a parte de ser utilizada en un navegador web, debe poder ser utilizada desde cualquier dispositivo móvil y mostrar los resultados de las operaciones entre lenguajes, permitiendo al usuario visualizar el resultado de las mismas.

Además, esta aplicación será capaz de mostrar al usuario cómo va a ser la evolución temporal de las principales operaciones entre lenguajes.

Dado que se ha de realizar una aplicación móvil, además de la aplicación web principal, se han de evaluar las distintas opciones de desarrollo móvil. Entre las opciones que se han evaluado se presentan las siguientes, cada una con sus ventajas e inconvenientes:

#### 3.1.1. Aplicaciones nativas

Las aplicaciones nativas son aquellas que se desarrollan de manera específica para un determinado sistema operativo utilizando un software de desarrollo específico comúnmente conocido como kit de desarrollo de software (SDK). Cada plataforma Android, IOS o Windows Phone, tiene su propio sistema, por lo que si quieres que la aplicación esté disponible en cada una de estas plataformas, deberás desarrollar una aplicación diferente para cada sistema operativo.

- Las aplicaciones para iOS se desarrollan con el lenguaje Objective-C.
- Las aplicaciones para Android se desarrollan con el lenguaje Java.
- Las aplicaciones en Windows Phone se desarrollan con el lenguaje .Net.

Cuando hablamos de desarrollo móvil casi siempre nos estamos refiriendo a aplicaciones nativas. La principal ventaja con respecto a los otros dos modelos o tipos, es la posibilidad de acceder a todas las características del hardware del móvil: cámara, GPS, agenda, dispositivos de almacenamiento y otras muchas. Esto hace que la experiencia del usuario sea mucho más positiva que con otro tipo de apps.

La descarga e instalación se realiza a través de las tiendas de aplicaciones, disponibles en los sistemas operativos de los dispositivos, facilitando la búsqueda e instalación a usuarios. Con esto el proceso de marketing y promoción se agiliza.

Si el coste no es un problema y disponemos del suficiente tiempo para desarrollarla, la mejor opción es la de desarrollar una aplicación nativa, pero si por el contrario el presupuesto y el tiempo de desarrollo es limitado, las aplicaciones web pueden proporcionar grandes ventajas.

A continuación, en la Tabla 3.1, presentamos sus ventajas e inconvenientes:

Ventajas	Inconvenientes
Acceso completo al dispositivo.	Diferentes habilidades / idiomas / herramientas para cada plataforma de destino.
Mejor experiencia del usuario.	Tienden a ser más caras de desarrollar.
Visibilidad en App Store.	El código del cliente no es reutilizable entre las diferentes plataformas.
Envío de notificaciones o “avisos” a los usuarios.	
La actualización de la App es constante.	

Tabla 3.1: Aplicaciones nativas

### 3.1.2. Aplicaciones web

Una aplicación web suele desarrollarse en lenguajes como HTML, JavaScript y CSS, muy conocidos y utilizados en el diseño de páginas web. La principal ventaja es que podemos programar la aplicación en un único lenguaje independientemente del sistema operativo que vayamos a utilizar, permitiéndonos ejecutar una misma aplicación en diferentes dispositivos sin tener que crear varias aplicaciones.

Las aplicaciones web se ejecutan dentro del propio navegador web del dispositivo a través de una URL. El contenido de éstas se adapta a la pantalla,

adquiriendo el aspecto de la navegación de una aplicación nativa gracias a la utilización de plantillas adaptativas.

Estas aplicaciones no requieren instalación y, aunque también es una ventaja, genera un inconveniente al no estar visible en los mercados de aplicaciones. La promoción y comercialización se realiza de manera independiente.

Las aplicaciones web móviles son siempre una buena opción si nuestro objetivo es adaptar la web a formato móvil

A continuación, en la Tabla 3.2, presentamos sus ventajas e inconvenientes:

Ventajas	Inconvenientes
El mismo código base es reutilizable en múltiples plataformas.	Requiere conexión a internet.
Proceso de desarrollo más sencillo y económico.	Acceso muy limitado a los elementos y características del hardware del dispositivo.
No necesitan ninguna aprobación externa para publicarse (a diferencia de las nativas para estar visibles en app store).	La experiencia del usuario (navegación, interacción, ...) y el tiempo de respuesta es menor que en una app nativa.
El usuario siempre dispone de la última versión.	Requiere de mayor esfuerzo en promoción y visibilidad.
Pueden reutilizarse sitios "responsive" ya diseñados.	

Tabla 3.2: Aplicaciones web

### 3.1.3. Aplicaciones híbridas

Una aplicación híbrida es una combinación de las dos anteriores, se podría decir que recoge lo mejor de cada una de ellas. Las apps híbridas se desarrollan con lenguajes propios de las web app, es decir, HTML, Javascript y CSS por lo que permite su uso en diferentes plataformas, pero también dan la posibilidad de acceder a gran parte de las características del hardware del dispositivo. La principal ventaja es que a pesar de estar desarrollada con HTML, Java o CSS, es posible agrupar los códigos y distribuirla en app store.

PhoneGap [11, 12] es uno de los frameworks más utilizados por los programadores para el desarrollo multi-plataforma de aplicaciones híbridas. Otro ejemplo de herramienta para desarrollar apps híbridas es Cordova [13].

A continuación, en la Tabla 3.3, presentamos sus ventajas e inconvenientes:



Ventajas	Inconvenientes
Es posible distribuirla en las tiendas de aplicaciones, como iOS y Android.	Experiencia del usuario más propia de la aplicación web que de la aplicación nativa
Instalación nativa pero construida con JavaScript, HTML y CSS.	Diseño no siempre relacionado con el sistema operativo en el que se muestre.
Acceso a parte del hardware del dispositivo.	La experiencia del usuario (navegación, interacción, ...) y el tiempo de respuesta es mayor que en una aplicación nativa.
El usuario siempre dispone de la última versión.	

Tabla 3.3: Aplicaciones híbridas

Una vez evaluadas las mismas seleccionamos las aplicaciones web e híbridas para el desarrollo de este proyecto por las razones que se explican a continuación:

Dado que una aplicación nativa supone el desarrollo de una aplicación en el lenguaje propio de cada sistema operativo, con los costes que esto supone, descartamos esta propuesta para evitar tener que diseñar, desarrollar y probar cada aplicación, ya que no contamos con el tiempo ni los medios necesarios para ello.

La aplicación web solucionaría los problemas presentados en la aplicación nativa, ya que permite que se pueda reutilizar el código y acceder a ella desde cualquier navegador disponible en los dispositivos móviles. Además comparte junto a las aplicación híbridas el ideal de código reutilizable. Esto hará finalmente que el desarrollo sea único para ambas plataformas.

## 3.2. Tecnologías

Detallaremos a continuación el conjunto de tecnologías usadas durante el transcurso de este trabajo fin de grado.

### 3.2.1. Navegadores

El navegador [14] es un programa que permite acceso a internet y es fundamental para poder acceder a la aplicación web. Éste se encarga de interpretar la información y de visualizarla. A través del navegador se realizarán las peticiones al controlador que se encuentra en el servidor y éste último nos devuelve el resultado para que lo muestre en el navegador.



Figura 3.1: Navegadores

Es importante destacar que nuestra aplicación web se ha desarrollado y testeado en el navegador web Google Chrome como primera elección dado que es el navegador más usado a día de hoy.

El Consorcio World Wide Web (W3C) [15] es una comunidad internacional donde las organizaciones Miembro, el personal a tiempo completo y el público en general trabajan conjuntamente para desarrollar estándares Web. Liderado por el inventor de la Web Tim Berners-Lee y el Director Ejecutivo (CEO) Jeffrey Jaffe, la misión del W3C es guiar la Web hacia su máximo potencial.

Conforme ha avanzado el proyecto he añadido Firefox al conjunto de navegadores web donde testear nuestra aplicación ya que se presenta como el navegador web que más se preocupa por seguir al máximo los estándares del W3C.

Cabe señalar que al comprobar el funcionamiento de nuestra aplicación web en ambos navegadores web estamos minimizando el uso de reglas indebidas o malinterpretadas, ya que lo que nos devuelve el navegador de lo que programamos se acerca mucho más a la regla general.

A continuación, en la Tabla 3.4, presentamos la compatibilidad de la Calculadora para Lenguajes Formales en los distintos navegadores:

Navegadores	Versión	Compatibilidad
Google Chrome	52.0.2743.116 m	Compatible
Firefox	48.0.2	Compatible
Safari	9.1.2	Compatible
Microsoft Edge	38.14393.0.0	No compatible
Opera	39.0.2256.48	Compatible

Tabla 3.4: Compatibilidad en distintos navegadores

### 3.2.2. HTML5

HTML5 (HyperText Markup Language, versión 5) [16,17] es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML.



Figura 3.2: HTML 5

Este lenguaje de marcas de hipertexto es un estándar a cargo de la W3C. Sirve para la elaboración de páginas web. Define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes, vídeos, entre otros.

### 3.2.3. CSS3

Hoja de estilo en cascada o CSS [18,19] es un lenguaje formulado por el W3C, utilizado para definir y crear la presentación de un documento estructurado escrito en HTML. CSS3 es la última versión y más actual para CSS.



Figura 3.3: CSS 3

### 3.2.4. JavaScript

JavaScript [20, 21] es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.



Figura 3.4: JavaScript

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

### 3.2.5. JQuery

jQuery [22, 23] es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.



Figura 3.5: JQuery

Es un software libre y de código abierto. Al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código.

### 3.2.6. Twitter Bootstrap

Twitter Bootstrap [24] es un framework o conjunto de herramientas de Código abierto para el diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales.



Figura 3.6: Bootstrap 3

Es el proyecto más popular en GitHub y es usado por la NASA y la MSNBC junto a demás organizaciones.

### 3.2.7. Git y Github

Git [25] es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

GitHub [26] es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc. (anteriormente conocida como Logical Awesome).

Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.



Figura 3.7: GitHub

### 3.2.8. Latex y ShareLatex

LaTeX [27] es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas.



Figura 3.8: Latex

Está formado por un gran conjunto de macros de TeX, escrito por Leslie Lamport en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica, LaTeX, creado por Donald Knuth. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea.

Esta disponible bajo una licencia LPPL.

Esta memoria se ha generado usando LaTeX.

ShareLaTeX [28] es un editor de texto basado en LaTeX diseñado específicamente para permitir la fácil y rápida colaboración de varios usuarios a la vez en un mismo proyecto en tiempo real. Este programa es totalmente gratuito y ofrece su código fuente completo a través de GitHub de manera que cualquiera podrá revisarlo y reportar posibles fallos y vulnerabilidades.

### 3.2.9. PhoneGap

PhoneGap [11, 12] es un framework para el desarrollo de aplicaciones móviles desarrollado por Nitobi, y posteriormente comprado por Adobe Systems. Permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3. El resultado no es una aplicación nativa al dispositivo, pero tampoco se trata de una aplicación web. En este caso hablamos de aplicaciones híbridas.



Figura 3.9: PhoneGap

Maneja la API que permite tener acceso a elementos como el acelerómetro, la cámara, los contactos en el dispositivo, la red, el almacenamiento, las notificaciones, etc. Estas API se conectan al sistema operativo usando el código nativo del sistema huésped a través de una Interfaz de funciones en Javascript.

No necesita de un simulador dedicado, permitiendo su desarrollo ejecutando las aplicaciones en nuestro navegador web y nos da la posibilidad de utilizar frameworks como Sencha Touch o JQuery Mobile.

### 3.2.10. Atom

Atom [29] es un editor de texto de código abierto con soporte para plugins escritos en Node.js, y bajo el control de cambios de Git, desarrollado por GitHub.

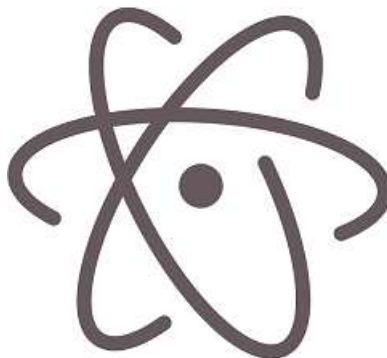


Figura 3.10: Atom

Es una aplicación de escritorio construida utilizando tecnologías web. La mayoría de los paquetes de ampliación tienen licencias de software libre y son construido por la comunidad y mantenidos.

Se basa en electrones (anteriormente conocido como átomo de Shell), una librería que permite a las aplicaciones de escritorio multi-plataforma usar chrome y Node.js. Está escrito en CoffeeScript y Less. Puede ser utilizado como un IDE.

Fue liberado como beta, en su versión 1.0, el 25 de junio de 2015.

### 3.2.11. Cloud9

Cloud9 [30] ofrece un entorno de desarrollo en la nube que permite a los desarrolladores empezar con la codificación de inmediato y colaborar con sus compañeros.



Figura 3.11: Cloud9

Con 3 años de experiencia en desarrollo IDE en la nube y cerca de medio millón de usuarios registrados, la misión de Cloud9 es desbloquear los beneficios de la escritura de software en la nube.

Fundada en 2010 y con sede en San Francisco y Ámsterdam, Cloud9 IDE es una compañía privada respaldada por Accel y Atlassian.



# Capítulo 4

## FLCalc

En este capítulo se presenta un pequeño manual de nuestra Calculadora para Lenguajes Formales.

### 4.1. ¿Qué es FLCalc?

FLCalc [31] es el nombre simbólico de nuestra Calculadora para Lenguajes Formales. Es una aplicación web destinada a presentar los conceptos básicos que se introducen a menudo en asignaturas dedicadas al estudio de los autómatas y los lenguajes formales.

La aplicación posee una interfaz sencilla que guía al usuario con una serie de pasos a completar para cada una de las operaciones entre lenguajes. A modo de síntesis, permite ver la evolución en el cálculo de las siguientes operaciones:

- Potencia del Lenguaje.
- Inversa del Lenguaje.
- Cierre de Kleene del Lenguaje.
- Cierre Positivo del Lenguaje.
- Concatenación entre Lenguajes.
- Unión entre Lenguajes.
- Intersección entre Lenguajes.
- Diferencia entre Lenguajes.
- Sublenguaje entre Lenguajes.
- Igualdad entre Lenguajes.

Para la interacción del usuario con las operaciones, la aplicación dispone de una serie de características que hacen más sencillo su uso.

- Generación del alfabeto.
- Generación del lenguaje.
- Señalización de avisos.
- Señalización de errores.
- Señalización de que un paso se ha completado.
- Señalización de que un paso no se ha completado.

## 4.2. Requisitos

El único requisito será disponer de un navegador web compatible con FLCalc y de una conexión a internet para tener acceso.

## 4.3. Completando los pasos

### 4.3.1. Paso 1

El primer paso trata de introducirnos en la definición de los alfabetos, los cuales serán utilizados más adelante. Para ello, se define en un primer momento los conceptos de símbolo y de alfabeto.

Símbolo/s

Definición: Un **símbolo** es un ente abstracto que no se puede definir formalmente. Letras o dígitos son ejemplos que frecuentemente usan símbolos.

Ejemplos:  
0, 1, a, b, hello, world

Dígitos binarios: 0 1

Letras del alfabeto español: A B

Palabras del idioma inglés: HELLO WORLD

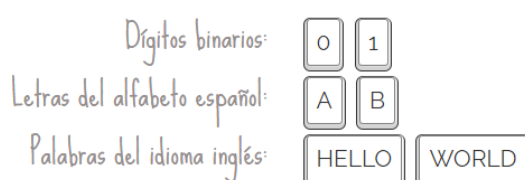


Figura 4.1: Paso 1 - Símbolo

Alfabeto/s

Definición: Un alfabeto  $\Sigma$  es un conjunto no vacío y finito de símbolos.

Ejemplos:

- $\Sigma = \{0, 1\}$
- $\Sigma = \{a, b\}$
- $\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$
- $\Sigma = \{\text{hello, world}\}$

Recuerda el uso de comas para separar cada símbolo

$\Sigma_1 = \{$    $\}$





   

Figura 4.2: Paso 1 - Alfabeto

El objetivo de este primer paso es que el usuario introduzca el alfabeto o alfabetos con los que va a trabajar. Debe de tener en cuenta que el lenguaje que creará en el siguiente paso deberá estar asociado al alfabeto que cree.

Recuerda el uso de comas para separar cada símbolo

$\Sigma_1 = \{$    $\}$


   

Figura 4.3: Paso 1 - Introducción del alfabeto

Los alfabetos de FLCalc sólo admiten símbolos formados por letras o números separados por comas. Además el alfabeto no puede contener el símbolo  $\epsilon$  representado en FLCalc con el símbolo &.

### 4.3.2. Paso 2

El segundo paso trata de introducirnos en la definición de los lenguajes, los cuales serán utilizados más adelante para realizar alguna operación. Para ello, se define en un primer momento los conceptos de cadena y de lenguaje.

Cadena/s

Definición: Una **cadena** (o palabra) es una secuencia finita de símbolos.

Ejemplo:  
 'a', 'b' y 'c' son **símbolos** y "abcb" es una **cadena**.  
 Una cadena sobre el alfabeto  $\Sigma = \{0,1\}$  podría ser "0100010".  
 Una cadena sobre el alfabeto  $\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$  podría ser "cadena".

Propiedad: La **longitud** de una cadena  $w$ , denotada por  $|w|$ , es el número de símbolos que componen la cadena.

Ejemplo:  
 "abcb" tiene longitud 4. Así  $|abcb| = 4$ .  
 La cadena vacía, denotada por  $\epsilon$ , es la cadena que contiene cero símbolos. Así  $|\epsilon| = 0$ .

Figura 4.4: Paso 2 - Cadena

Lenguaje/s

Definición: Un **lenguaje** (formal) es un conjunto de cadenas. El lenguaje vacío  $\emptyset$  es el lenguaje compuesto por ninguna cadena. Decimos que el lenguaje vacío no es el mismo que el que consta de la cadena vacía:  $\emptyset \neq \{\epsilon\}$ .

Para especificar el conjunto de cadenas que conforman un lenguaje (finito), las separaremos por comas y las delimitaremos por llaves.

Ejemplo:  
 $L = \{abc, bbcc, a, cab\}$

Para especificar el **lenguaje vacío** se podría hacer algo como lo siguiente:

Ejemplo:  
 $L = \{\}$

Y para especificar la **cadena vacía** se podría utilizar, por ejemplo, el símbolo  $\&$ . Teniendo esto en cuenta, y para evitar confusiones, el símbolo  $\&$  no podrá utilizarse como elemento de los alfabetos que se utilizarán en nuestra calculadora.

Ejemplo:  
 $L = \{\&\}$

Recuerda el uso de los tres puntos para hacer el lenguaje infinito

$L_1 = \{ \quad \} \Sigma_1$

Figura 4.5: Paso 2 - Lenguaje

El objetivo de este segundo paso es que el usuario introduzca el lenguaje o lenguajes con los que va a trabajar. Debe de tener en cuenta que según el número

de lenguajes podrá realizar una operación u otra. Además deberá asociar el lenguaje al alfabeto anteriormente creado.

Recuerda el uso de los tres puntos para hacer el lenguaje infinito



Figura 4.6: Paso 2 - Introducción del lenguaje

Se ha de tener en cuenta que los lenguajes de FLCalc sólo admiten símbolos asociados al alfabeto de este. Además un lenguaje podrá contener el símbolo épsilon (“&”) admitido por FLCalc. En FLCalc no será válido un lenguaje con cadenas similares.

Para cambiar el alfabeto asociado del lenguaje deberemos pulsar sobre este y cambiará automáticamente entre los creados anteriormente.

Además FLCalc soporta una aproximación a que los lenguajes creados sean infinitos. Para ello al final del lenguaje añadiremos tres puntos (“...”).

### 4.3.3. Paso 3

Una vez completados los pasos anteriores podremos elegir la operación a realizar. Para ello, elegiremos una de las siguientes, mostradas en la Figura 4.7, según el número de lenguajes creados.

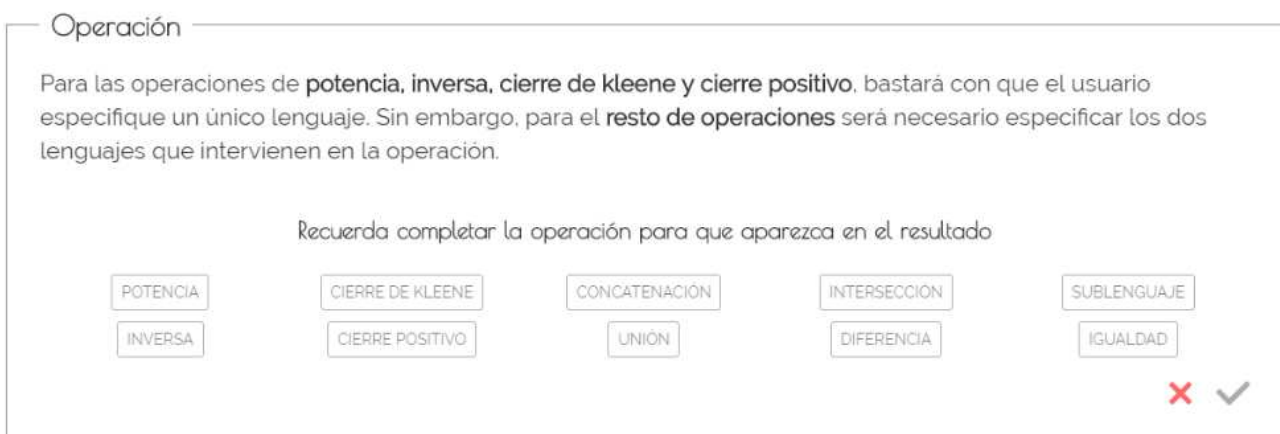


Figura 4.7: Paso 3 - Operaciones

Se ha de tener en cuenta que las operaciones estarán disponible automáticamente según los lenguajes creados anteriormente y el tamaño de estos.

En el caso de tener definido un único lenguaje podremos realizar las siguientes operaciones:

- Potencia del Lenguaje.
- Inversa del Lenguaje.
- Cierre de Kleene del Lenguaje.
- Cierre Positivo del Lenguaje.

En el caso de tener definido dos lenguajes podremos realizar las siguientes operaciones:

- Concatenación entre Lenguajes.
- Unión entre Lenguajes.
- Intersección entre Lenguajes.
- Diferencia entre Lenguajes.
- Sublenguaje entre Lenguajes.
- Igualdad entre Lenguajes.

#### **4.3.4. Paso 4**

El objetivo de este cuarto paso es que el usuario vea el resultado final de la operación ejecutada anteriormente. Básicamente se mostrará un resumen de los lenguajes con los que se ha operado, la operación realizada y el resultado de la misma.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ \text{ } \}$

Lenguaje/s:

$L_1 = \{ \text{ } \}$   $\Sigma_1$

Operación:

---

Resultado:

$L = \{ \text{ } \}$

✖ ✔

Figura 4.8: Paso 4 - Resultado

## 4.4. Avisos

FLCalc dispone de una serie de avisos que ayudarán a el usuario a interactuar con la herramienta.

Debe existir **mínimo** un alfabeto para asociarlo al lenguaje.

Ha llegado al **máximo número de alfabetos** permitidos.

Debe existir **mínimo** un lenguaje para realizar una operación.

Ha llegado al **máximo** número de lenguajes permitidos.

Figura 4.9: Avisos

## 4.5. Mensajes de error

FLCalc dispone de una serie de mensajes de error que ayudarán a el usuario a interactuar con la herramienta.



Figura 4.10: Mensajes de error

## 4.6. Operaciones

Definidos los lenguajes formales podemos realizar sobre ellos las siguientes operaciones básicas. En cada operación se muestra una interfaz funcional, ilustrativa y atractiva a nivel visual, en la que vemos la evolución temporal de la operación elegida.



### 4.6.1. Potencia del lenguaje

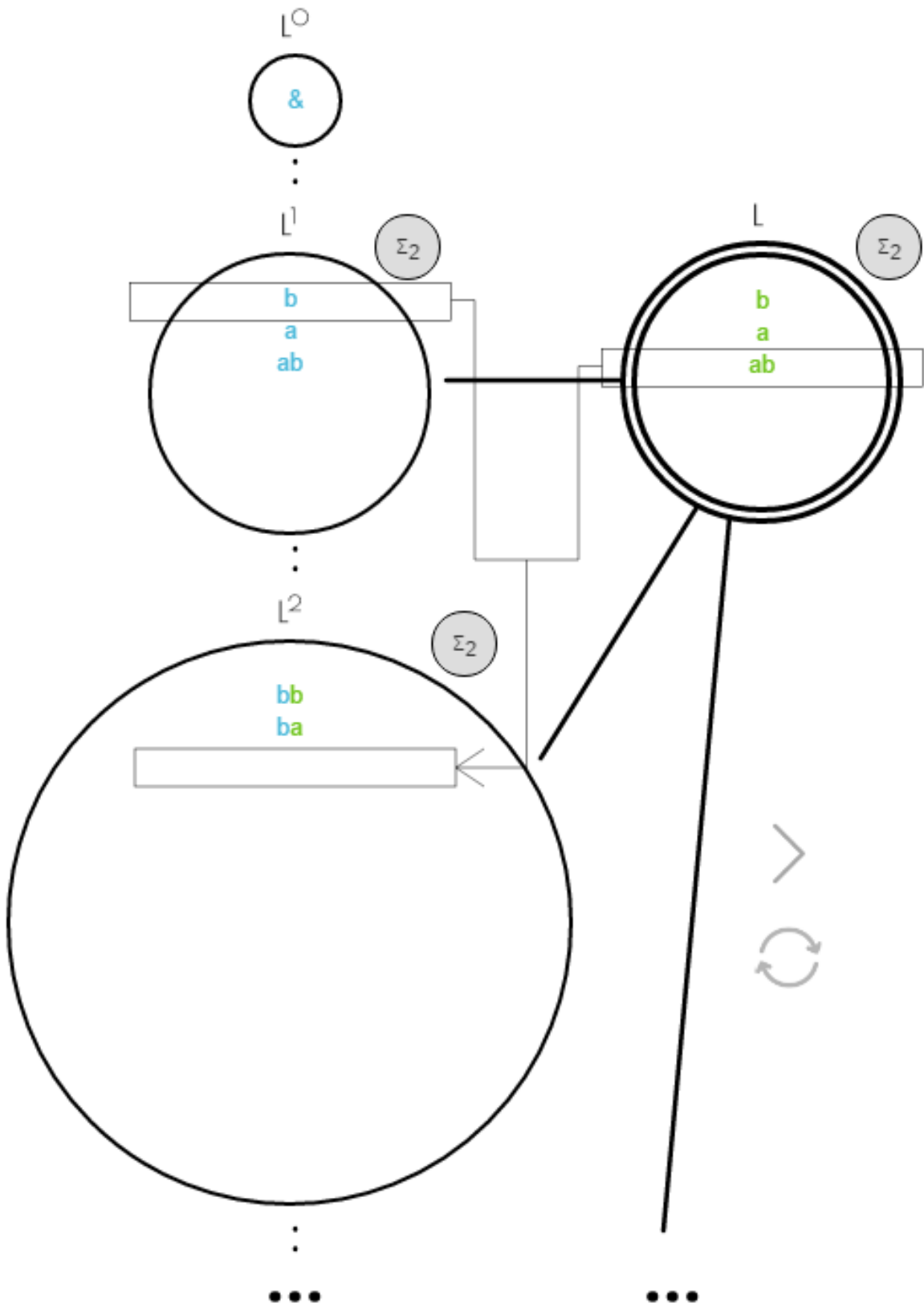


Figura 4.11: Operación - Potencia

Esta operación requiere la definición de un único lenguaje.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

$\Sigma_2 = \{ a, b \}$

Lenguaje/s:

$L_1 = \{ b, a, ab \} \mid \Sigma_2$

Operación:

$L^0, L^1, L^2, \dots$

Resultado:

$L^0 = \{ \& \}$

$L^1 = \{ b, a, ab \}$

$L^2 = \{ bb, ba, bab, ab, aa, aab, abb, aba, abab \}$

...

✕ ✓

Figura 4.12: Resultado - Potencia

### 4.6.2. Inversa del lenguaje

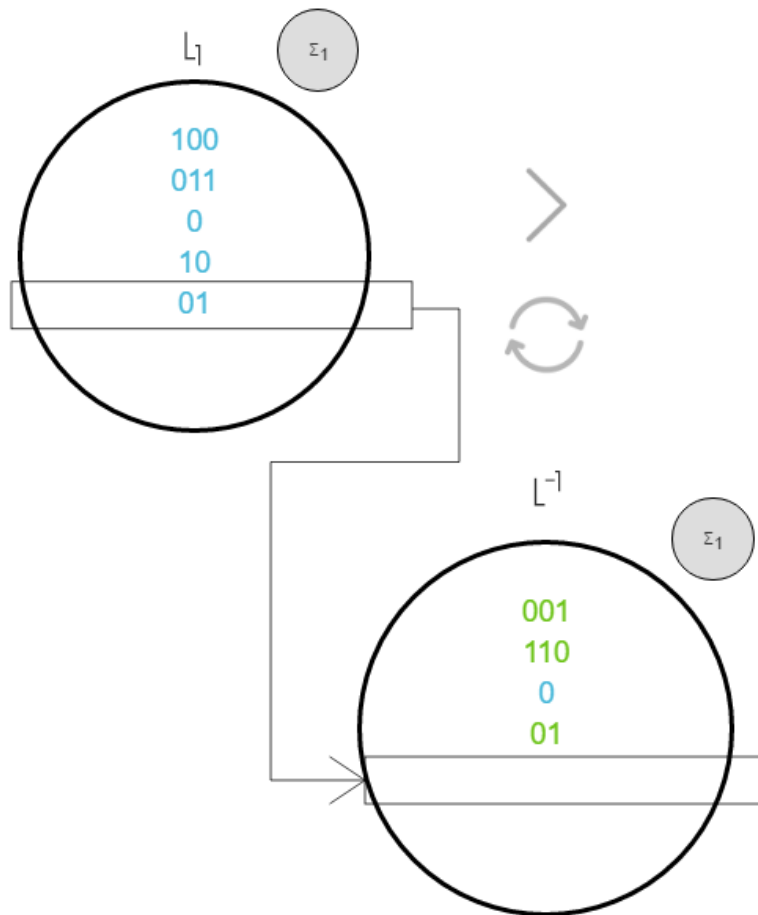


Figura 4.13: Operación - Inversa

Esta operación requiere la definición de un único lenguaje.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{$  0, 1  $\}$

$\Sigma_2 = \{$  a, b  $\}$

Lenguaje/s:

$L_1 = \{$  100, 011, 0, 10, 01  $\}$   $\Sigma_1$

Operación:

$L^{-1}$

Resultado:

$L^{-1} L_1 = \{$  001, 110, 0, 01, 10  $\}$

✕ ✓

Figura 4.14: Resultado - Inversa

### 4.6.3. Cierre de Kleene del lenguaje

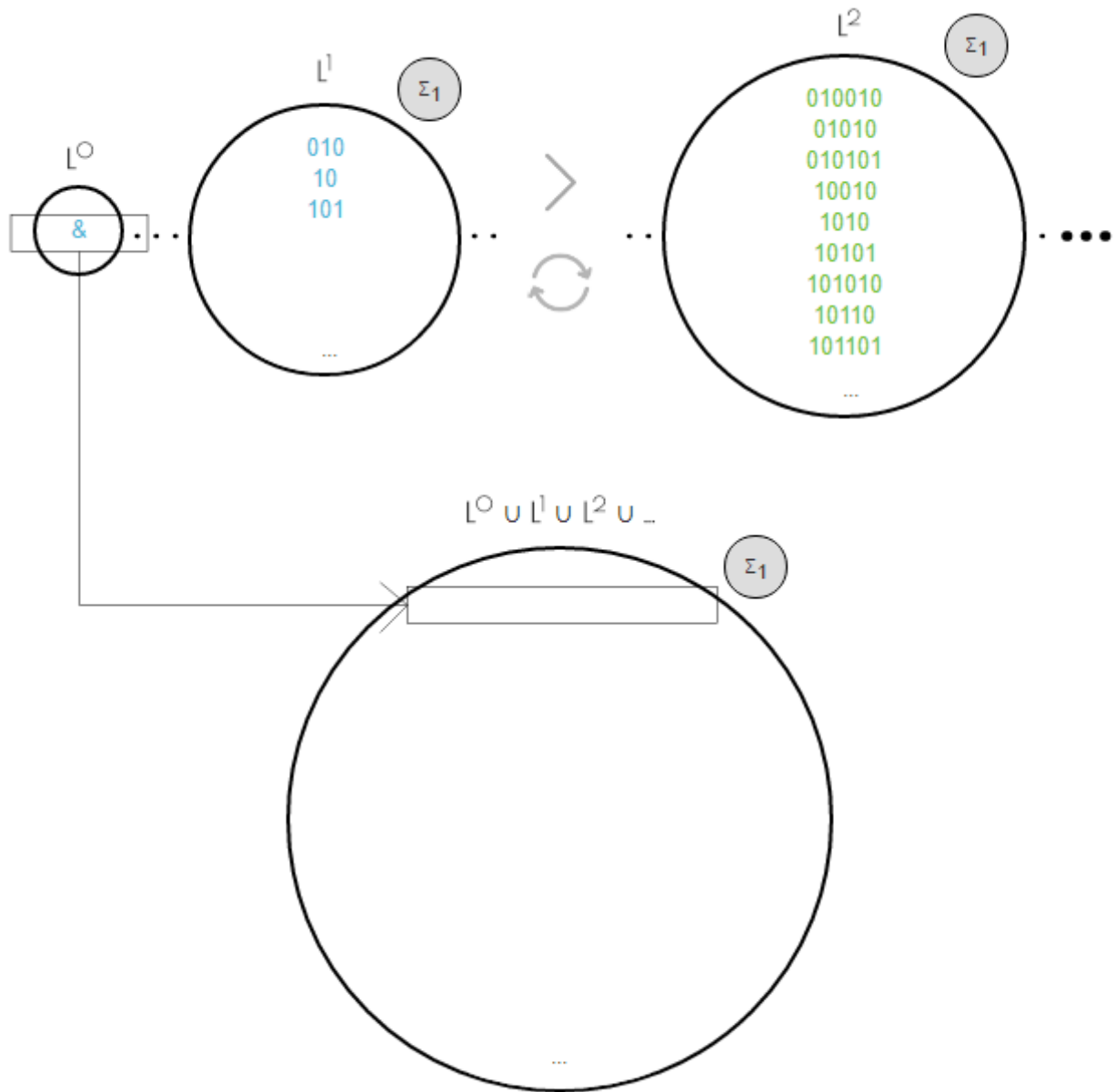


Figura 4.15: Operación - Cierre de Kleene

Esta operación requiere la definición de un único lenguaje.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 010, 10, 101, \dots \}$

Operación:

$L^*$

Resultado:

$L^*_{L_1} = \{ \epsilon, 010, 10, 101, 010010, 01010, 010101, 10010, 1010, 10101, 101010, 10110, 101101, \dots \}$

✕ ✓

Figura 4.16: Resultado - Cierre de Kleene

### 4.6.4. Cierre Positivo del lenguaje

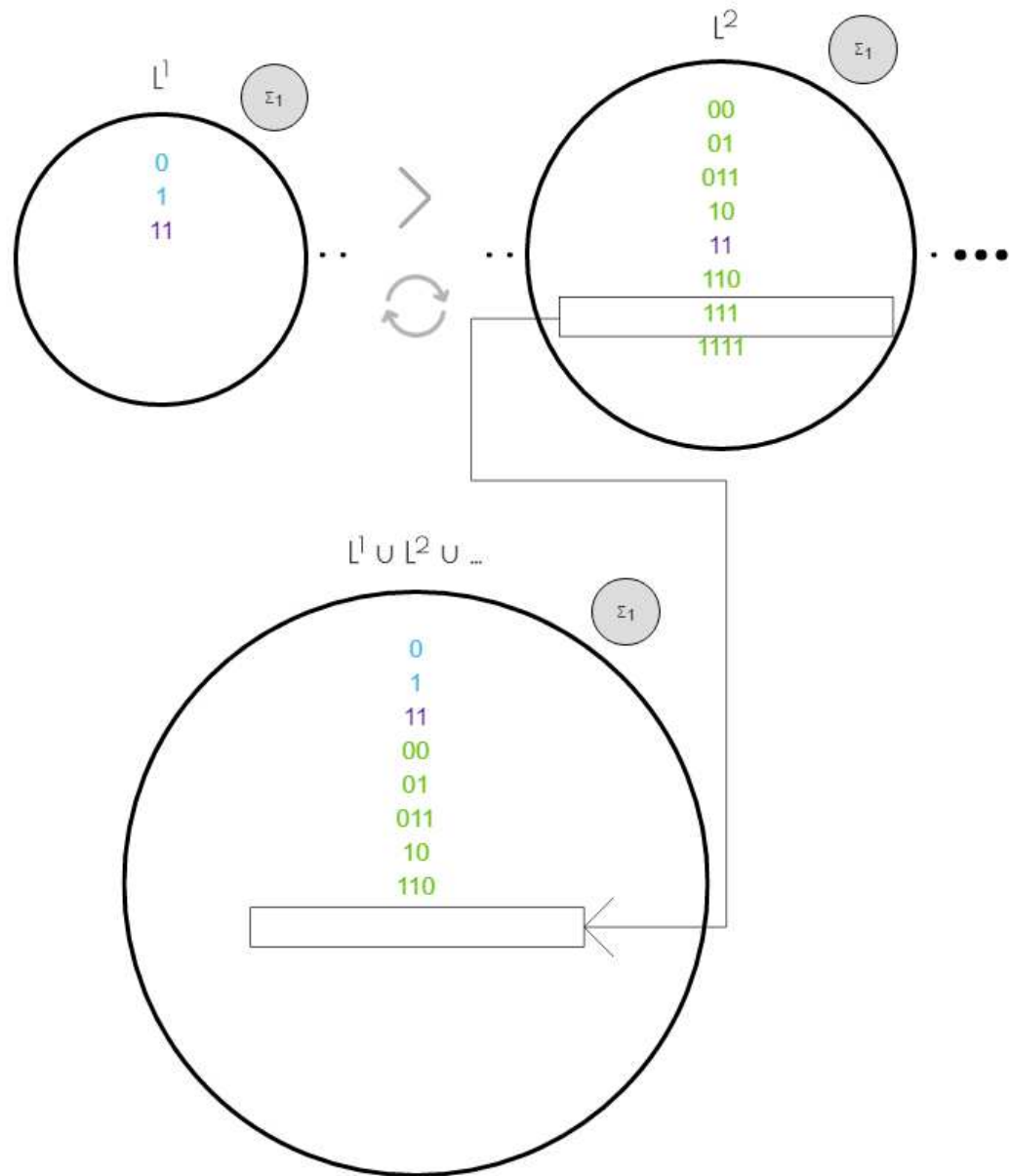


Figura 4.17: Operación - Cierre Positivo

Esta operación requiere la definición de un único lenguaje.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 0, 1, 11 \}$

Operación:

$L^+$

Resultado:

$L^+_{L_1} = \{ 0, 1, 11, 00, 01, 011, 10, 110, 111, 1111, \dots \}$

✕ ✓

Figura 4.18: Resultado - Cierre positivo



### 4.6.5. Concatenación entre lenguajes

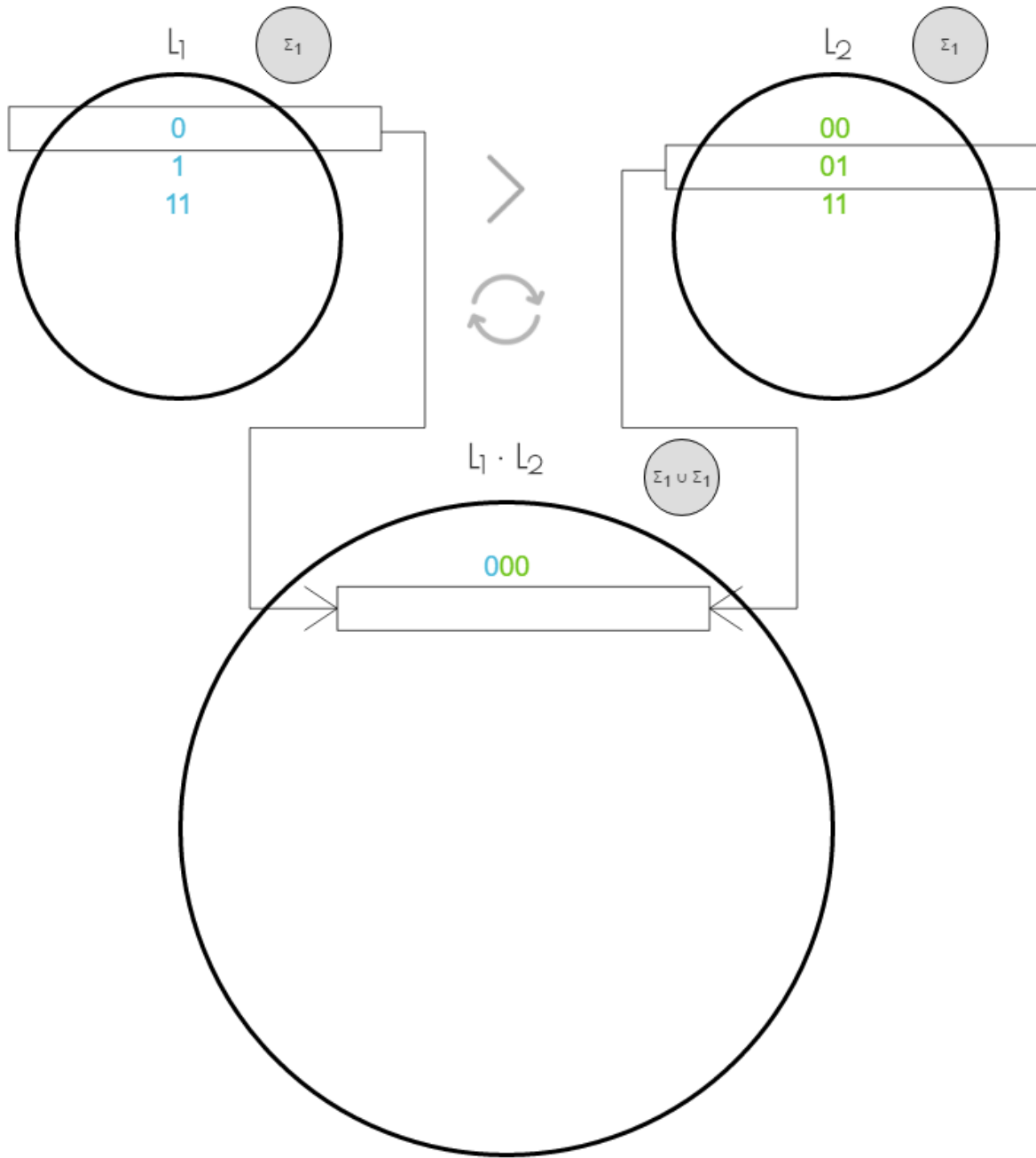


Figura 4.19: Operación - Concatenación

Esta operación requiere la definición de dos lenguajes.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 0, 1, 11 \}$   $\Sigma_1$

$L_2 = \{ 00, 01, 11 \}$   $\Sigma_1$

Operación:

$L_1 \cdot L_2$

Resultado:

$L_{L_1 \cdot L_2} = \{ 000, 001, 011, 100, 101, 111, 1100, 1101, 1111 \}$

✕ ✓

Figura 4.20: Resultado - Concatenación

### 4.6.6. Unión entre lenguajes

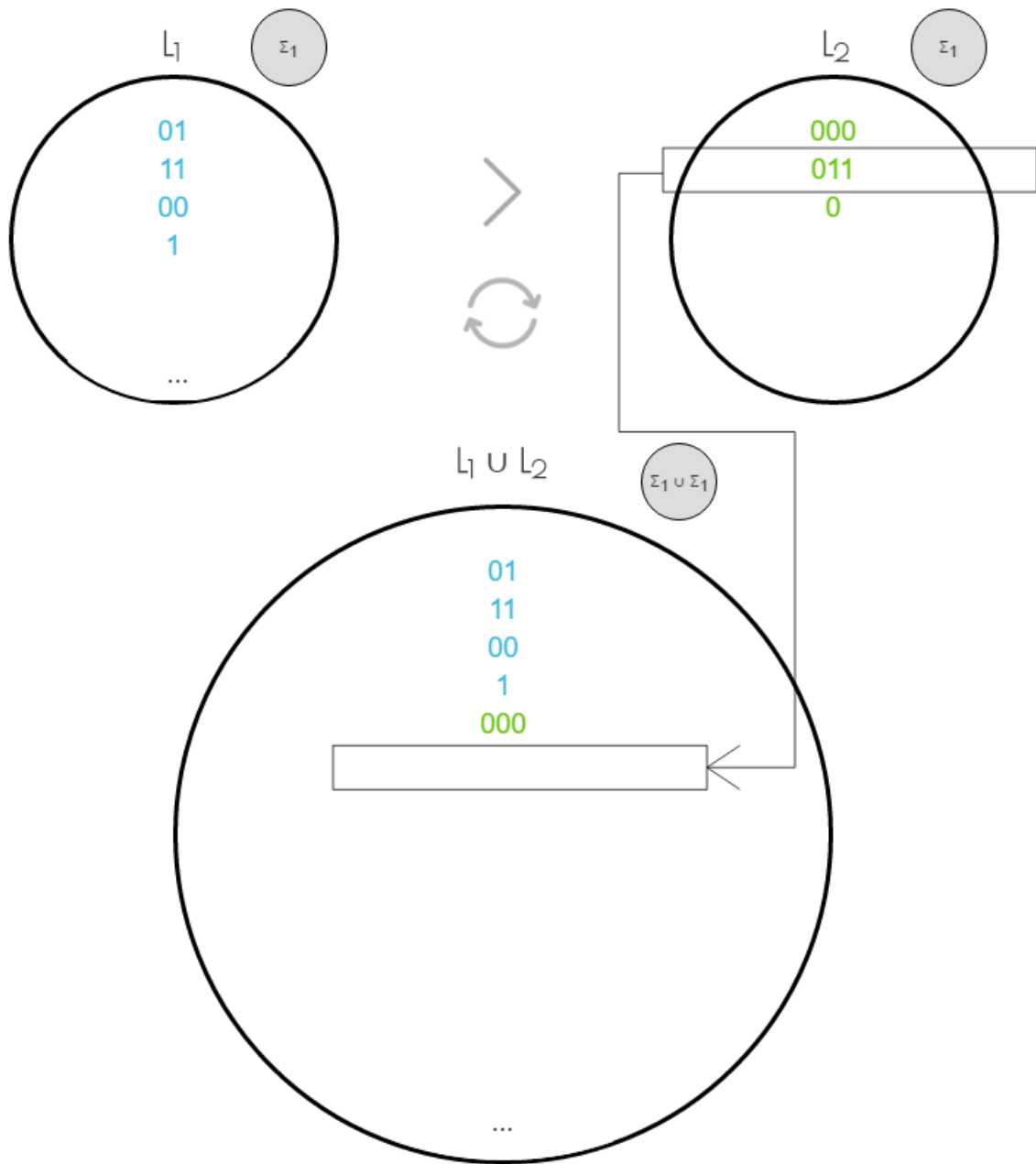


Figura 4.21: Operación - Unión

Esta operación requiere la definición de dos lenguajes.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 01, 11, 00, 1, \dots \}$   $\Sigma_1$

$L_2 = \{ 000, 011, 0 \}$   $\Sigma_1$

Operación:

$L_1 \cup L_2$

Resultado:

$L_{L_1 \cup L_2} = \{ 01, 11, 00, 1, 000, 011, 0, \dots \}$

✕ ✓

Figura 4.22: Resultado - Unión

### 4.6.7. Intersección entre lenguajes

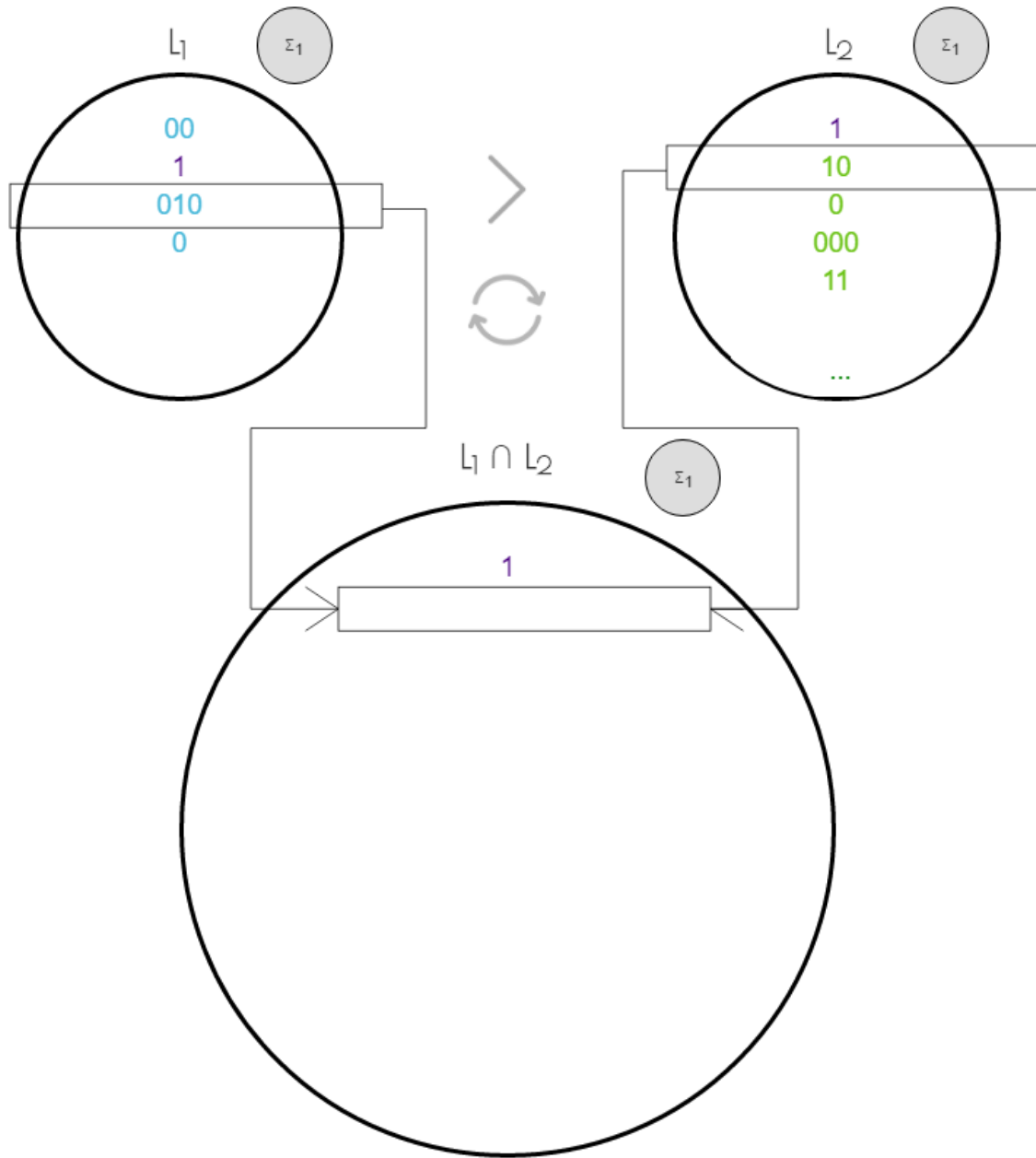


Figura 4.23: Operación - Intersección

Esta operación requiere la definición de dos lenguajes.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 00, 1, 010, 0 \}$   $\Sigma_1$

$L_2 = \{ 1, 10, 0, 000, 11, \dots \}$   $\Sigma_1$

Operación:

$L_1 \cap L_2$

Resultado:

$L_{L_1 \cap L_2} = \{ 1, 0 \}$

✕ ✓

Figura 4.24: Resultado - Intersección

### 4.6.8. Diferencia entre lenguajes

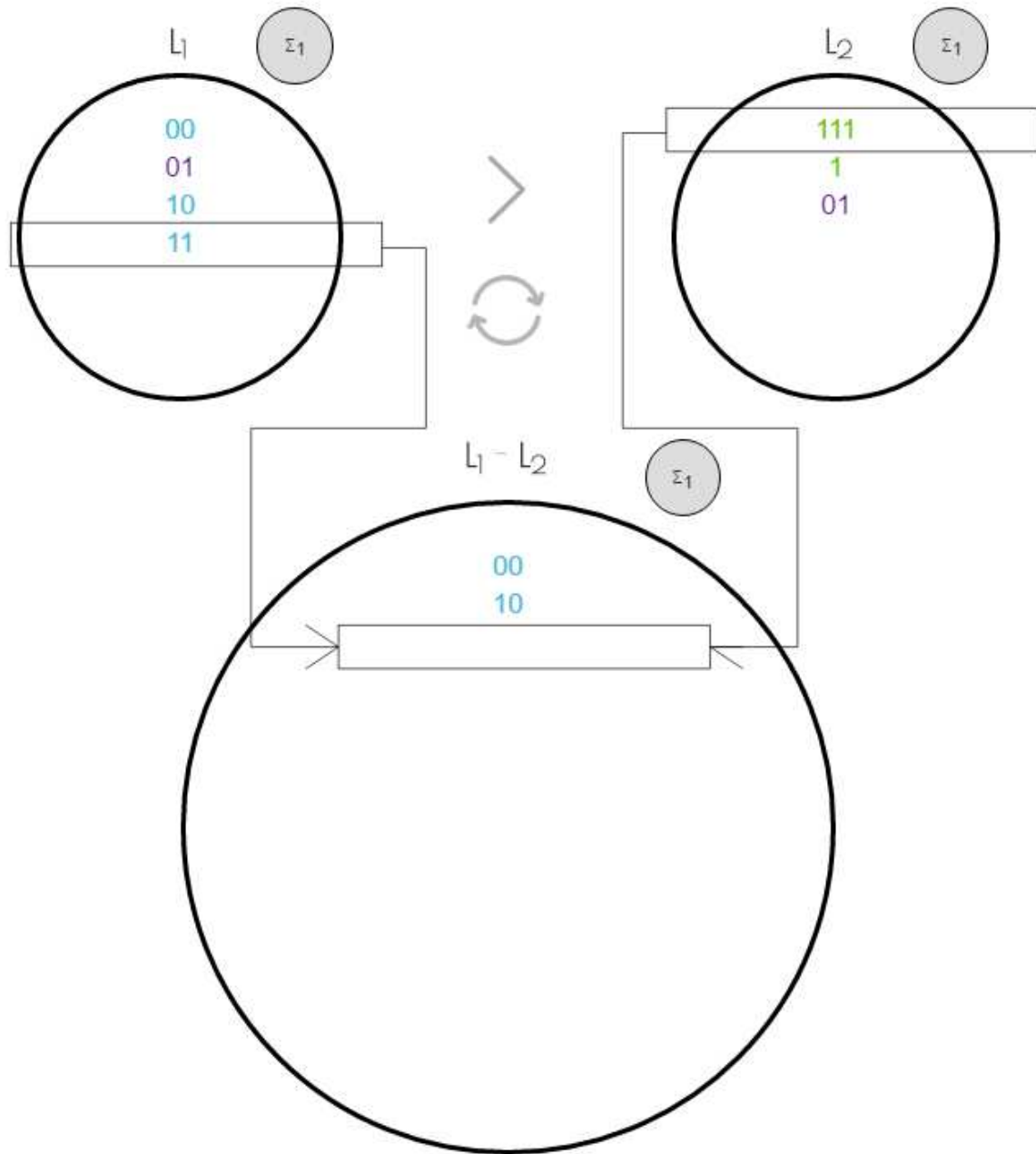


Figura 4.25: Operación - Diferencia

Esta operación requiere la definición de dos lenguajes.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 00, 01, 10, 11 \}$

$L_2 = \{ 111, 1, 01 \}$

Operación:

$L_1 - L_2$

Resultado:

$L_{L_1 - L_2} = \{ 00, 10, 11 \}$

Figura 4.26: Resultado - Diferencia



### 4.6.9. Sublenguaje entre lenguajes

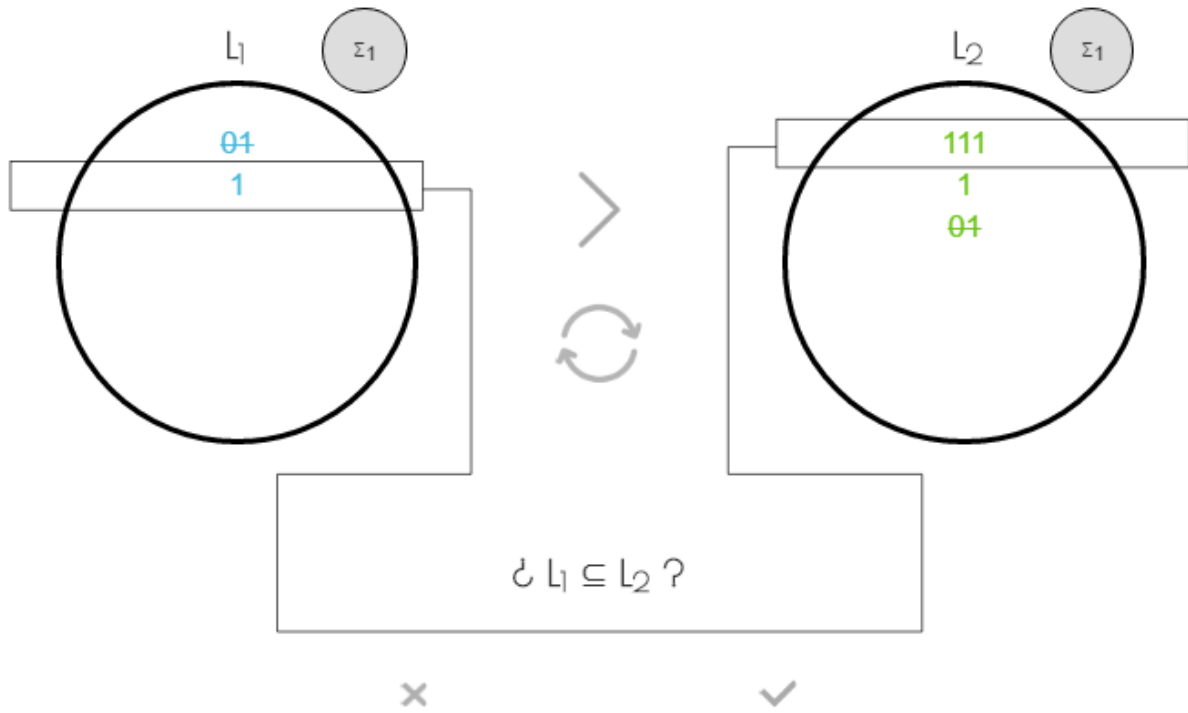


Figura 4.27: Operación - Sublenguaje

Esta operación requiere la definición de dos lenguajes.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = \{ 0, 1 \}$

Lenguaje/s:

$L_1 = \{ 01, 1 \}$   $\Sigma_1$

$L_2 = \{ 111, 1, 01 \}$   $\Sigma_1$

Operación:

$\zeta L_1 \subseteq L_2 ?$

Resultado:

$L_1$  es un sublenguaje de  $L_2$

Figura 4.28: Resultado - Sublenguaje

### 4.6.10. Igualdad entre lenguajes

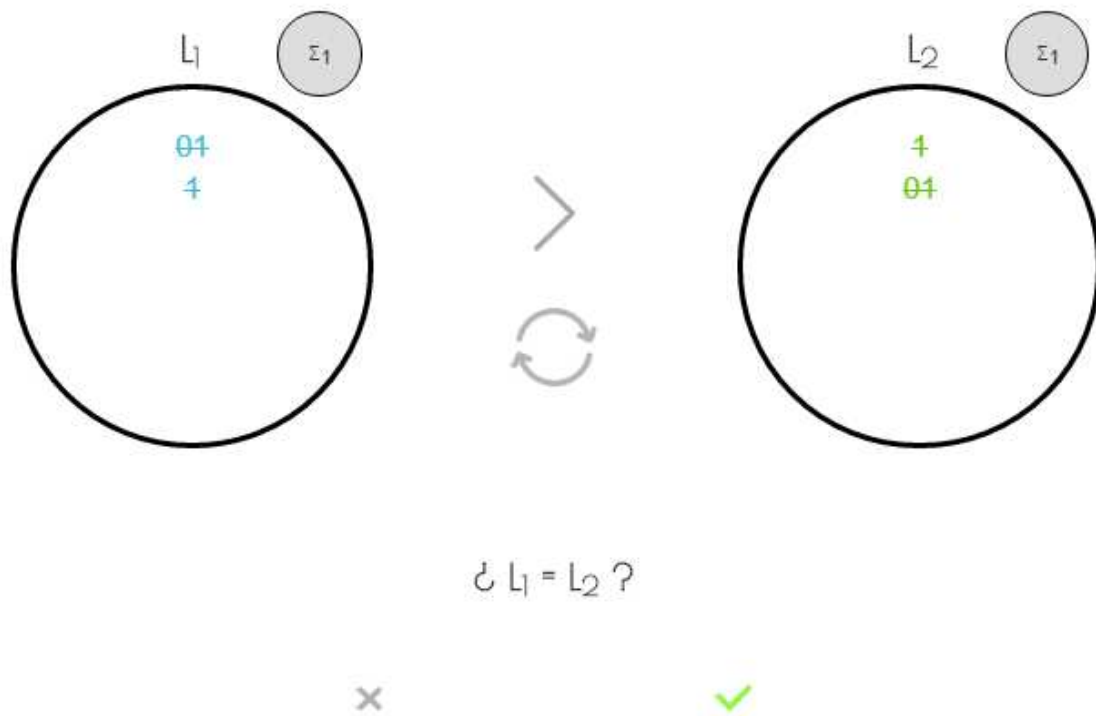


Figura 4.29: Operación - Igualdad

Esta operación requiere la definición de dos lenguajes.

Resultado

Resultado de nuestra Calculadora de Lenguajes Formales.

Alfabeto/s:

$\Sigma_1 = [ 0, 1 ]$

Lenguaje/s:

$L_1 = [ 0^*1, 1 ]$   $\Sigma_1$

$L_2 = [ 1, 0^*1 ]$   $\Sigma_1$

Operación:

$\zeta L_1 = L_2 ?$

Resultado:

$L_1$  y  $L_2$  son iguales

✕ ✓

Figura 4.30: Resultado - Igualdad

# Capítulo 5

## Conclusiones y trabajos futuros

En este capítulo se describen las conclusiones posteriores a la realización del Trabajo Fin de Grado, a fin de revisar los objetivos que fueron planteados en los distintos apartados durante su realización y para comprobar el grado de satisfacción de cada uno de ellos. Tras este análisis se introducen algunas líneas futuras sobre un posible Trabajo Fin de Grado donde se podrían seguir avanzando funcionalidades del proyecto.

El presente Trabajo Final de Grado ha tenido como resultado la creación de una aplicación web amigable y sencilla disponible para diversos tipos de dispositivos. Ésta, conocida como Calculadora para Lenguajes Formales, permite operar con lenguajes formales, permitiendo visualizar el proceso y mostrar la evolución temporal de cada operación.

### 5.1. Conclusiones

En este apartado se describen una serie de objetivos que se plantearon para la realización del Trabajo Fin de Grado y se exponen las conclusiones sobre su cumplimiento:

- Se ha llevado a cabo una evaluación de soluciones existentes, donde hemos analizado el desarrollo de aplicaciones (nativas, web e híbridas) y la selección de la tecnología a utilizar.
- Se ha conseguido la creación de una interfaz con capacidad de mostrar información útil de cara al alumnado.
- Se ha conseguido que el usuario interactúe con nuestra Calculadora de Lenguajes Formales.

- Se ha conseguido que la aplicación muestre la evolución temporal de cada operación.
- Se ha conseguido llevar FLCalc a una versión para el móvil gracias al uso de PhoneGap.
- Y por último, gracias a todos los objetivos planteados, se ha logrado el desarrollo de una aplicación web e híbrida, robusta y eficiente, disponible para su uso.

Tras haber cumplido los objetivos principales planteados en un principio, se puede llegar a la conclusión de que se ha alcanzado satisfactoriamente los objetivos del Trabajo Fin de Grado.

La realización de este Trabajo Fin de Grado ha conllevado un amplio estudio sobre herramientas tales a JQuery. Además de ampliar los conocimientos en herramientas ya conocidas como, HTML 5, CSS 3, JavaScript y PhoneGap.

Durante el desarrollo de la aplicación se han presentado diversas dificultades a las que nos hemos tenido que enfrentar, como el desconocimiento en parte de algunas tecnologías usadas como JQuery y como las dificultades encontradas en la creación de los gráficos de cada operación.

Cabe decir que el presente Trabajo Fin de Grado ha sido una labor enriquecedora, proporcionando y reforzando conocimientos. Gracias a éste hemos adquirido experiencia de cómo enfrentarnos a un problema real.

## 5.2. Trabajos futuros

Para finalizar este capítulo se presentarán posibles líneas de futuro para esta Calculadora para Lenguajes Formales. Éstas pueden ser estudiadas y desarrolladas con una gran probabilidad de éxito, continuando así con el trabajo ya realizado:

- **TypeScript:** es un lenguaje basado en Javascript para, en un principio, ayudar a desarrollos Javascript a gran escala y hacerlos más sencillos y con un mejor rendimiento y una mejor escalabilidad. En nuestra Calculadora para Lenguajes Formales aportará un mayor control del código para que este sea más legible, además será más sencillo mejorar el rendimiento y la funcionalidad de nuestra Calculadora para Lenguajes Formales.
- **API (Interfaz de programación de aplicaciones):** es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por

otro software como una capa de abstracción. En nuestra Calculadora para Lenguajes Formales aportará el tener toda su funcionalidad guardada para ser utilizada fuera de la aplicación web en sí misma.

- **Manejo de lenguajes finitos e infinitos mediante expresiones regulares:** mediante el uso de expresiones regulares podremos formar lenguajes finitos e infinitos en nuestra Calculadora para Lenguajes Formales. Estas aportarán una mayor potencia y un mayor rendimiento a la hora de operar con los lenguajes formales.
- **Mejorar la parte gráfica para lenguajes con muchas cadenas:** es importante encontrar alguna solución auto adaptable para los círculos que contienen muchas cadenas. Resulta necesario que las cadenas no se oculten en ningún momento ya que es importante que estén siempre visibles para el usuario.

# Capítulo 6

## Conclusions and future work

This chapter describes the subsequent conclusions to the realization of the Final Degree Project, to revise the objectives raised in the different sections during implementation and check the overall satisfaction. After such an analysis, we will introduce some future lines of work which would help to improve the project features.

This Final Degree Project has resulted in the design and development of a friendly and simple web application available for different types of devices. Such application, known as Formal Languages Calculator, can operate with formal languages, allowing the users to visualize the process and show the time evolution of each operation.

### 6.1. Conclusions

This section describes a number of proposed objectives for the realization of Final Degree Project and conclusions are presented on their fulfillment:

- It has carried out an assessment of existing solutions, where we have analyzed the development of applications (native, web and hybrid) and the selection of the technology to use.
- It has been achieved creating an interface with ability to display useful information for the students.
- It has gotten the user to interact with our calculator Formal Languages.
- It was determined that the application to display the time evolution of each operation.



- It has managed to bring FLCalc to a version for mobile by using PhoneGap.
- And finally, thanks to all those objectives, it has achieved development of a web application and hybrid, robust and efficient, available for use.

Having fulfilled the main objectives in principle, we can conclude that can reach the conclusion that it has successfully reached the pre-established objectives of the Final Degree Project.

The realization of this Final Degree Project has implied a wide study of tools like JQuery. In addition to expanding knowledge in tools already known as HTML 5, CSS 3, JavaScript and PhoneGap.

During the development of the application have been submitted several difficulties have been faced: the lack of knowledge of some technologies used as JQuery and difficulties in creating graphs of each operation.

We can say that this Final Degree Project has been a labor enriching, providing and reinforcing knowledge. Thanks to this we have gained experience of how to deal with a real problem.

## 6.2. Future work

To end this chapter possible future lines will be presented for this calculator for Formal Languages. These can be studied and developed with a high probability of success, continuing the work already done:

- **TypeScript:** It is based on Javascript for, initially, help Javascript large-scale developments and make them simpler and better performance and better scalability language. In our Formal Language Calculator will provide greater control over the code to make this more readable, also it will be easier to improve performance and functionality of our Formal Language Calculator.
- **API (application programming interface):** is the set of subroutines, functions, and procedures (or methods in object-oriented programming) that offers certain library for use by other software as an abstraction layer. In our Formal Language Calculator provide the full functionality have saved for use outside the web application itself.
- **Management finite and infinite languages using regular expressions:** using regular expressions we can form finite and infinite in our Formal Language Calculator. These provide more power and increased performance when operating with formal languages.

- **Improve the visualization part for languages with many chains:** is important to find a solution for adaptive circles containing many chains . It is necessary that the strings are not hidden at any time as it is important that they are always visible to the user.

# Capítulo 7

## Presupuesto

En este capítulo se va a detallar el presupuesto para la realización de este Trabajo Fin de Grado.

### 7.1. Presupuesto

A continuación, en la Tabla 7.1, se puede observar las distintas actividades realizadas durante el presente Trabajo Fin de Grado indicando la duración en horas de cada una de ellas, con un total de 300 horas.

Para la estimación del cálculo se ha establecido un precio total de 20€ la hora. El resultado estimado del coste total de la aplicación es de 6000€.

Actividad	Duración (horas)	Coste (€)
Evaluación de las soluciones existentes	20	400
Selección de la tecnología a utilizar	20	400
Análisis de requisitos	40	800
Diseño	40	800
Implementación	70	1400
Evaluación y pruebas	20	400
Elaboración de la memoria	70	1400
Revisión	20	400

<b>TOTAL</b>	<b>300</b>	<b>6000</b>
--------------	------------	-------------

Tabla 7.1: Tiempo dedicado y presupuesto

Además, en la Tablas 7.2 y 7.3, se pueden observar el hardware y software utilizado durante el presente Trabajo Fin de Grado con su coste asociado.

<b>Hardware</b>	<b>Coste (€)</b>
MSI pe60 6QD	770,00 €

<b>TOTAL</b>	<b>770,00 €</b>
--------------	-----------------

Tabla 7.2: Hardware utilizado

<b>Software</b>	<b>Coste (€)</b>
Atom	0
ShareLatex	0 (limitado)
Cloud9	0
Git y Github	0
Google chrome, Firefox, safari, edge, opera	0
Windows 10	279,00 €

<b>TOTAL</b>	<b>279,00 €</b>
--------------	-----------------

Tabla 7.3: Software utilizado

Debido a que se ya se disponía de el hardware y el software utilizado, el presupuesto real invertido no ha ascendido a la totalidad de dicha cantidad.

# Bibliografía

- [1] Kelley D. *Teoría de Autómatas y Lenguajes Formales*. Prentice-Hall, 1995.
- [2] Brookshear J. G. *Teoría de la Computación: Lenguajes Formales, Autómatas y Complejidad*. Addison-Wesley Iberoamericana, 1993.
- [3] Motwani R. Hopcroft J. and Ullman J. *Introducción a la teoría de Autómatas, Lenguajes y Computación*. Addison-Wesley, 2002.
- [4] D. Alfredo Rodríguez Sánchez (Selfa Pro) y D. Raúl Miguel Sabariego (resolución de bugs sobre Selfa Pro) D. Julian Hortolano Villarejo (Selfa). **SELF A Pro** - Herramienta Web que resuelve ejercicios sobre autómatas finitos, expresiones regulares, gramáticas regulares, autómatas con pila y gramáticas libres de contexto, 2007. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [5] Susan H. Rodger. **JFlap** - Aplicación escrita en Java sobre lenguajes formales y autómatas, 1990. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [6] Ivan Zuzak. **Noam** - Librería JavaScript para trabajar con autómatas y lenguajes formales, 2012. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [7] Ivan Zuzak. **FSM2Regex** - herramienta web que convierte una expresión regular en un autómata finito determinista, 2012. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [8] Ivan Zuzak. **FSM Simulator** - herramienta web que simula el funcionamiento de un autómata finito dada una cadena de entrada, 2012. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [9] Wolfram Research. **Wolframalpha** - herramienta web para el calculo de operaciones con conjuntos, 2009. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].

- [10] Jeff Avallone. **Regexper** - herramienta web para decodificar expresiones regulares, 2014. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [11] **PhoneGap** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [12] Andrey Kovalenko. *PhoneGap by Example*. Packt publishing, 2015.
- [13] **Cordova** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [14] **Navegador Web** - wikipedia. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [15] **W3C** - consorcio world wide web, españa. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [16] **HTML5** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [17] Brian Albers Peter Lubbers and Frank Salim. *Pro HTML5 Programming*. Apress, 2011.
- [18] **CSS3** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [19] Peter Gasston. *The Book of CSS3*. No Starch Press, 2011.
- [20] **JavaScript** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [21] Marijn Haverbeke. *Eloquent JavaScript*. No Starch Press, 2011.
- [22] **JQuery** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [23] Ronan Cranley Ryan Benedetti. *Head First jQuery*. O'Reilly Media, 2011.
- [24] **Bootstrap** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [25] **Git** - wikipedia. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [26] **GitHub** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].

- [27] **LaTeX** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [28] **ShareLatex** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [29] **Atom** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [30] **Cloud9** - página web oficial. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].
- [31] **Calculadora para Lenguajes Formales**. [Enlace](#) [Disponible electrónicamente; Último acceso julio de 2016].