Universitat de Girona
Escola Politècnica Superior

Treball de final de grau

Estudi	Grau en enginyeria informàtica
Títol	Desenvolupament d'una aplicació web per visualitzar i executar els models de la teoria d'autòmats
Document	Memòria
Alumne	Pol Barrachina Hernandez (u1946623)
Tutor	Dr. Rigau Vilalta, Jaume
Departament	IMAE
Àrea	LPI
Convocatòria	Juny 2020

Índex

1 Introducció, motivacions, propòsit i objectius del projecte

La idea principal de realitzar aquest treball sorgeix a arrel del meu interès per l'àrea de computació. Veient les propostes per part del professorat em va cridar l'atenció perquè reunia un altre punt d'interès, la programació web.

La motivació per dur a terme aquest projecte és ajudar a donar a conèixer aquesta branca de la computació que és la teoria d'autòmats i oferir a tothom que s'hi interessi una eina de suport addicional que permeti treballar els conceptes més bàsics. S'espera desenvolupar una aplicació que de manera interactiva mostri quines son les diferències entre els tipus d'autòmats, quines son les seves característiques i propietats i quines són les seves capacitats de còmput.

Es tracta d'una eina pràctica que ajuda a introduir-se en aquest àmbit i que pot despertar l'interès d'alumnes i guiar-los en l'aprenentatge treballant aplicacions i exemples d'ús del temari estudiat.

L'objectiu és la programació d'una eina de suport web a la docència que permeti definir autòmats finits deterministes, autòmats de pila, i màquines de Turing d'una sola cinta i multicinta, així com les seves respectives versions indeterministes i la seva modificació, execució pas per pas, la representació de manera visual i la capacitat per exportar i importar-les.

Es busca donar una solució senzilla que permeti als alumnes testejar els dissenys d'autòmats i verificar el seu funcionament sense necessitat d'aprendre altres conceptes sobre l'aplicació. El llibre que s'utilitza com a base de l'assignatura corresponent és *Introduction to the Theory of Computation* [1] de Michael Sipser és per això que s'ha fet servir per facilitar encara més l'aprenentatge permetent simular i testejar alguns dels autòmats que apareixen al llibre com a exemples.

2 Estudi de viabilitat

Donat que l'aplicació s'executa només en l'ordinador client i no és necessari l'ús de servidor, no són necessaris recursos externs per dur-la a terme. La distribució de la aplicació consisteix en transferir un fitxer estàtic que podem esperar que ocupi 1Mb, per tant els costs d'emmagatzematge i transferència són molt baixos, gratuit en vàries plataformes per compatir fitxers.

Per tant els únics costs que s'han de tenir en compte a l'hora de fer el pressupost són els recursos humans i els equips necessaris pel desenvolupament del projecte. Es farà servir el cost complert del màquinari, tot i que es podria reutilizar per altres projectes un cop acabada la fase de desenvolupament.

2.1 Recursos equip

Recurs	Preu en Euros
Ordinador durant desenvolupament	1400 €

Figura 1: Recursos maquinari

El cost del programari és gratuit, incloent el sistema operatiu (Distribució de Linux), editor de text (vim), editor d'imatges (inkscape), navegador i llibreries emprades durant el desenvolupament que es detallen en l'apartat Estudis i decisions. S'ha fet servir un servei de repositori de codi *git* extern també gratuït.

En el cas dels recursos humans es fa un càlcul aproximat de les hores i el seu cost en funció de la tasca que es duu a terme.

2.2 Recursos humans

Recurs	€/h	Hores	Total €
Analista	40	60	2400
Programació	25	350	8750
Total	-	410	11150

Figura 2: Recursos humans

El cost total del desenvolupament de l'aplicació és de 12.550€ i aquí faltaria sumar el temps dedicat pel tutor i els impostos.

3 Metodologia

Abans de començar amb el projecte, es desenvolupen una sèrie de prototips molt senzills, pràcticament proves de funcionament, de les diferents llibreries, tant de manera separada com conjunta, per veure la viabilitat del projecte.

Un cop es conclou la possibilitat d'executar el projecte, els objectius principals en aquests primers prototips passen a ser detectar què és el necessari per millorar la compatibilitat entre eines i llibreries, quines son necessàries i sobretot quina estructura del projecte és més adient per

aprofitar al màxim les funcionalitats clau de cada llibreria, produir un codi que sigui senzill de mantenir i millorar i que admeti canvis, per tant afegint les capes d'abstracció necessàries que facilitin l'anterior a través d'interfícies.

El primer que s'ha fet és dividir l'aplicació en dues parts, en primer lloc una llibreria amb la lògica dels autòmats i en segon lloc la interfície visual i la interacció amb l'usuari. El motiu d'aquesta separació en una llibreria, és degut a que son un conjunt de funcionalitats prou autònom com per fer-se servir en altres aplicacions i que al tenir una interfície clara permet validar el funcionament de la llibreria de manera independent.

La metodologia global ha seguit un paradigma incremental, en cadascuna de les funcionalitats de la llibreria es programava la respectiva interfície generant una aplicació funcional. En cada cicle, s'afegia una nova funcionalitat i es verificava manualment que no s'havien produït regressions.

L'aplicació té tres models de computació, els autòmats finits, els autòmats de pila i les màquines de Turing que tot i tenir una part comuna s'ha decidit fer-los independents i d'aquesta manera evitar les regressions.

Degut a la diferent naturalesa de cadascuna de les parts de l'aplicació, tant la llibreria com la interfície, s'ha fet servir una metodologia diferent en cadascuna de les parts.

3.1 Funcionament dels autòmats

L'avantatge principal d'aquesta part és que els requeriments son rígids i la planificació no ha de tenir en compte canvis imprevists. En aquesta part s'inclou la lògica dels autòmats, és a dir, la creació, el parsing, la modificació, l'execució i les accions que podem aplicar, per exemple minimitzar un autòmat finit determinista.

La definició i les propietats dels autòmats estan clarament definides i per tant és senzill definir un conjunt respectiu de tests que en verifiquin el correcte funcionament abans d'escriure el codi, és per això que s'ha decidit emprar la metodologia TDD (*Test Driven Development*).

En la metodologia TDD un test es una funció que verifica que el comportament d'una altra coincideix amb el que s'espera, per fer-ho disposa d'una o més parelles de paràmetres i resultats que fa servir per cridar la funció i comparar el que retorna amb el resultat esperat.

Els passos principals a seguir d'aquesta metodologia son els següents [2]:

1. **Afegir un test** En aquest primer pas per exemple s'afegiria un test que inicialitzes un autòmat finit determinista i es verificaria que els estats inicials i finals son correctes així com les transicions entre estats.
2. **Executar tots els tests** En aquest pas observaríem que el nou test falla. En aquest projecte s'executa de manera implícita cada vegada que es modifica un fitxer. S'executen tots els tests i es mostra un resum dels que han fallat, el valor que han tornat i el que se'n esperava.
3. **Escriure el codi** Programar la funció i comprovar que passa el test que s'acaba d'afegir.

4. **Executar tots els tests** Verificar que tots els tests passen. En aquest projecte aquest pas és implícit perquè cada vegada que es modifica un fitxer s'executen tots els tests.
5. **Refactor** Eliminar i natejar codi duplicat.

3.2 Interacció amb l'usuari

En aquesta segona part no és senzill aplicar una metodologia orientada a tests perquè els requeriments de la interfície no son estàtics i estan subjectes a canviar o adaptar-se amb més facilitat.

La metodologia seguida ha consistit en el desenvolupament d'un prototip evolutiu. El prototip inicial contenia l'estructura de l'aplicació i la distribució dels apartats. No tenia cap requeriment implementat i només permetia navegar entre els components buits.

A partir d'aquí, en cada cicle de desenvolupament de la interfície d'usuari s'ha afegit una de les funcionalitats de la llibreria d'autòmats de l'apartat anterior. En cada iteració del cicle es disposava d'una versió funcional que donades les dimensions del projecte i l'aïllament entre components es verificava de manera manual.

Igual que en l'apartat anterior, les funcionalitats s'han afegit en ordre de proximitat i usabilitat. Garantint que es pogués verificar la nova funcionalitat afegida. S'ha començat pels autòmats finits i les diferents maneres en que es poden importar, exportar i definir. Després, la resta de requeriments garantint que era possible testejar-les, és a dir, s'ha implementat primer la conversió a GNFA i després l'eliminació d'un estat d'un autòmat GNFA.

4 Marc de treball i conceptes previs

4.1 Marc de treball

Eina de suport a la docència que ajuda a iniciar-se en els conceptes de la teoria de la computació permetent una interacció i visualització dels models de còmput i les seves propietats.

4.2 Conceptes previs

[3] La teoria de la computació comprèn les propietats matemàtiques fonamentals del maquinari, programari i certes aplicacions dels mateixos. Estudiant la teoria de la computació, busquem determinar què es pot i no es pot computar, amb quina rapidesa, amb quanta memòria i sota quin model de computació. El tema esta fortament relacionat amb enginyeria i, com en moltes ciències, també té aspectes purament filosòfics.

En aquest treball es tracten 3 tipus d'autòmats i algunes de les seves possibles variants.

4.2.1 Autòmats finits

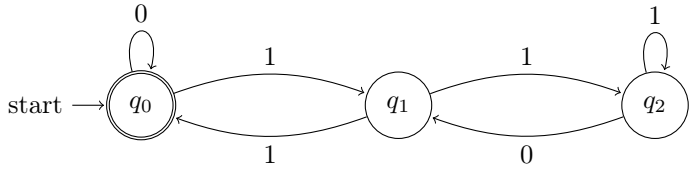


Figura 3: Diagrama d'estats d'un autòmat finit determinista que reconeix el llenguatge dels nombres múltiples de 3 en base binaria

Un autòmat finit consisteix en un conjunt d'estats Q, un inicial i cap o múltiples finals F i transicions δ entre estats que tenen associades un símbol w. El conjunt de tots els símbols d'un autòmat és l'alfabet Σ. Es fa servir la notació $\delta(q_i, w) = q_j$ per indicar una transició de l'estat q_i a l'estat q_j amb el símbol d'entrada w. Un mot és una combinació de símbols de l'alfabet $w_0w_1w_2 \ldots \in \Sigma^*$. Si un estat no té cap transició associada a un símbol, implícitament s'indica que rebutja el mot, a nivell teòric, es desplaça a un estat de rebuig d'on no se'n pot sortir.

Un autòmat finit llegeix un mot d'entrada, símbol a símbol, d'esquerra a dreta i es desplaça entre els estats seguint les transicions. La màquina s'atura quan ha llegit tot el mot i si es troba en un estat final el mot pertany al llenguatge i és acceptat, en cas contrari és rebutjat.

El llenguatge d'un autòmat és el conjunt dels mots que accepta. Tots els autòmats reconeixen un i només un llenguatge [4].

En l'exemple de la Fig. 3 s'observen 3 estats $\{q_0, q_1, q_2\}$, 6 transicions $\{\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = q_1$, $\delta(q_1, 1) = q_0$, $\delta(q_1, 1) = q_2$, $\delta(q_2, 1) = q_2$, $\delta(q_2, 0) = q_1\}$, l'estat inicial i final q_0.

La variant no determinista permet que les transicions siguin d'un estat a molts (one-to-many) i amb el símbol ϵ (epsilon) s'indica que es pot anar d'un estat a un altre sense llegir cap símbol d'entrada. La màquina s'atura en acabar de llegir el mot i com en el cas determinista només si ho fa en un estat final el mot pertany al llenguatge. El poder computacional d'aquesta variant és equivalent a la de l'autòmat finit determinista.

Determinizar un autòmat
Consisteix en trobar un autòmat que reconegui el mateix llenguatge que l'inicial i no utilitzi ϵ ni transicions d'un estat a molts. Tots els autòmats finits no deterministes tenen un autòmat equivalent determinista [5].

Minimitzar un autòmat finit determinista
Consisteix en trobar un autòmat amb el menor nombre d'estats possibles que reconegui el mateix llenguatge que l'autòmat inicial.

Expressions Regulars
Una expressió regular descriu un llenguatge regular, un llenguatge regular és aquell que pot ser reconegut per un autòmat finit. Tots els llenguatges regulars es poden descriure amb una expressió regular.

R és una expressió regular si R és un símbol d'un alfabet, un $\emptyset$ (que representa el llenguatge buit), ϵ (que representa el llenguatge que només accepta el mot buit); la concatenació o unió ($\cup$ o $|$) de dues expressions regulars; l'estrella * d'una expressió regular. [6]

Un exemple d'expressió regular per descriure el llenguatge la Fig. 3 és $(0|(1(01^*(00)^*0)^*1)^*)^*$.

GNFA
És una variació dels autòmats finits no deterministes que enlloc de transicions que tenen associades un símbol hi tenen una expressió regular.

Són útils per obtenir una expressió regular que descrigui el llenguatge d'un autòmat finit. La transformació més important que hi podem aplicar és l'eliminació d'un estat, que aplicada recursivament produeix un GNFA de dos estats, un inicial i un final. La transició de l'estat inicial al final té associada l'expressió regular que descriu el llenguatge.

4.2.2 Autòmats de pila

Els autòmats de pila són similars als autòmats finits però tenen una pila on poden llegir i escriure un símbol en cada transició. El tamany de la pila es infinit. La limitació d'aquesta pila és que només pot llegir el símbol de dalt de tot, l'últim afegit, i treure'l. Per llegir un símbol anterior s'ha de buidar tota la pila fins la posició on s'havia afegit el símbol.

La variant no determinista, és a dir, que pot desplaçar-se entre estats sense llegir cap símbol d'entrada (ϵ) o accedir a diversos estats amb un sol símbol d'entrada, és més potent que la versió determinista. El conjunt de llenguatges que pot reconèixer la versió no determinista és més gran al de la versió determinista.

4.2.3 Màquines de Turing

El model de màquina de Turing utilitza una cinta infinita com a memòria. A diferència de l'autòmat de pila, té un capçal que li permet moure's per la cinta i pot llegir i escriure en la posició on es troba. Les transicions entre estats de la màquina depenen de l'estat actual i el que es llegeix de la cinta, en funció d'això canvia d'estat, sobreescriu el símbol de la cinta de sota el capçal i el desplaça a la dreta o l'esquerra. En la màquina de Turing d'aquest treball es permet també mantenir la mateixa posició del capçal.

Inicialment la cinta conté el mot d'entrada i espais buits a la resta de posicions. La cinta és infinita per les dues bandes. Si la màquina necessita escriure informació, ho fa a la cinta. La màquina pot recuperar la informació que ha escrit anteriorment movent el capçal fins la posició anterior.

La màquina computa fins que decideix produir una resposta. Les respostes *accept* i *reject* només es poden obtenir entrant en estats acceptadors i de rebuig. Si la màquina no entra en un dels estats d'acceptació o de rebuig, segueix computant per sempre, sense parar.

Les dues variants que es tracten en aquest treball són les màquines multicinta i les no deterministes.

Una TM de k cintes té k capçals. En cada transició es té en compte l'estat actual i el conjunt

de k-símbols que llegeix de tots els capçals. En una transició es canvia d'estat, es sobreescriu els símbols de sota tots els capçals i es desplacen com a molt una posició.

Una TM no determinista o NDTM pot en cada transició prendre vàries rutes de còmput. El concepte de ruta de còmput correspon a que en una sola transició en comptes d'anar a un sol estat i escriure un símbol pot simultàniament anar a diversos estats i en cada un escriure un símbol a la cinta. Només cal que una ruta de còmput accepti el mot perquè la TM accepti el mot, encara que infinites rutes el rebutgin.

Es pot imaginar que la NDTM sempre té sort i que en cas que existeixi una ruta de còmput que accepta el mot, la selecciona. Com s'explica més endavant, en aquesta aplicació en cada indeterminació es fa una còpia sencera de la màquina i en cada pas totes les còpies avancen un pas.

També és possible utilitzar les dues variacions anteriors simultàniament i crear una màquina de Turing no determinista multicinta.

La tesis de Church-Turing indica l'equivalència entre la idea d'algoritme i màquina de Turing, per tant, totes les variacions de la màquina de Turing anteriors han de tenir un poder computacional equivalent a una màquina de Turing estàndar. És possible crear una màquina de Turing determinista d'una sola cinta que reconegui el mateix llenguatge que qualsevol de les variants anteriors.

5 Requisits del sistema

5.1 Requeriments funcionals

Àrea	Requeriment
Autòmats finits	El sistema permet crear, modificar, importar i exportar autòmats a partir de la definició d'estats i transicions
Autòmats finits	El sistema permet carregar un autòmat a partir de la definició d'una expressió regular
Autòmats finits	El sistema permet executar autòmats NFA i DFA i comprovar si un mot pertany al llenguatge
Autòmats finits	El sistema permet determinitzar l'autòmat en cas que no ho estigui
Autòmats finits	El sistema permet obtenir l'autòmat amb menor nombre d'estats que reconeix el mateix llenguatge que un altre DFA (minimitzar)
Autòmats finits	El sistema permet convertir un autòmat a GNFA
Autòmats finits	El sistema permet reduir un GNFA eliminant un estat però mantenint el llenguatge reconegut.
Autòmats finits	El sistema permet visualitzar i accedir a l'historial de canvis que s'han realitzat sobre l'autòmat.
Autòmats de pila	El sistema permet crear, modificar, importar i exportar autòmats a partir de la definició d'estats i transicions.
Autòmats de pila	El sistema permet executar pas per pas un autòmat de pila i comprovar si un mot pertany al llenguatge. En cas que sigui no determinista poder seleccionar quina instància es vol visualitzar i mostrar l'estat i el contingut de la pila.
Autòmats de pila	El sistema permet visualitzar i accedir a l'historial de canvis que s'han realitzat sobre l'autòmat.
Màquines de Turing	El sistema permet crear, modificar, importar i exportar màquines de Turing a partir de la definició d'estats i transicions.
Màquines de Turing	El sistema permet executar pas per pas una màquina de Turing i comprovar si un mot pertany al llenguatge

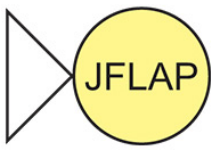
5.2 Requeriments no funcionals

Àrea	Requeriment
Usabilitat	Els símbols dels autòmats han de poder ser strings, números o caràcters.
Usabilitat	Les entrades de l'historial d'autòmats han de tenir un nom entenedor que indiqui la versió que representa.
Usabilitat	Es permet a l'usuari moure els estats i les transicions, canviar l'algoritme de layout, canviar el color dels estats i el tamany de la lletra
Usabilitat	La exportació i importació d'autòmats es fa en formats de text que l'usuari pot editar i entendre.
Usabilitat	Els formats de text per definir els estats i les transicions de les diferents variants d'autòmats tenen una gramàtica similar
Usabilitat	La interfície entre les diferents variants d'autòmats ha de ser similar.
Usabilitat	Al navegar per l'aplicació s'ha de conservar l'estat de tots els components.
Usabilitat	Poder fer *drag & drop* de fitxers amb els codis dels autòmats a l'aplicació i carregar-los automàticament.
Rendiment	Donar l'opció d'executar els autòmats sense mostrar-los per millorar el rendiment.
Software	Ha de ser compatible amb tots els navegadors amb una quota de mercat superior al 1%.

6 Estudi previ

Existeixen aplicacions que tenen un propòsit similar al d'aquest projecte. He centrat la recerca en la més utilizada i complerta, JFLAP.

6.1 JFLAP



JFLAP (*Java Formal Language and Automata Package*) permet a l'usuari crear i operar autòmats, gramàtiques, sistemes L i expressions regulars. A més, disposa de multitud d'utilitats que permeten explorar els llenguatges dels diversos models de computació, la conversió entre els models equivalents i altres operacions per visualitzar les seves propietats. [7]

JFLAP és un programa basat en Java, per tant compatible amb tots els sistemes operatius destinats a ordinadors personals. No és necessari instal·lar-lo, només tenir instal·lat l'entorn de Java.

És un projecte molt complert que abasta més continguts que els d'aquest treball. Aquest treball intenta centrar-se en portar la part dels autòmats a una aplicació web i on no sigui necessari instal·lar dependències externes, només un navegador.

JFLAP conté un editor i dissenyador d'autòmats visual, en l'aplicació del treball no es crearà un dissenyador, sinó un editor senzill que permeti modificar i adaptar els autòmats, però amb una manera senzilla d'exportar i importar els autòmats en un format de text llegible i formal que permeti utilitzar programes externs pel disseny.

7 Estudis i decisions

En aquest apartat es detallen les eines que s'han fet servir tant en la llibreria de simulació d'autòmats com en la interfície d'usuari.

Com s'indica als requeriments, aquesta aplicació ha de ser local i per navegador, per tal de ser compatible amb tots els dispositius sense importar el sistema operatiu. L'aplicació ha de ser dinàmica i per tant s'utilitzarà Javascript que és el llenguatge de programació que interpreten tots navegadors.

L'aplicació es divideix en dues parts, la part lògica del funcionament dels autòmats i la part de representació i interacció amb l'usuari (*frontend*). Les dues es trobaran al navegador del client, però a l'hora de desenvolupar es tractaran com dos projectes diferents.

7.1 Software necessari

Per desenvolupar una aplicació en Javascript són molt necessàries aquestes eines i ho són tant a la part de *frontend* com a la llibreria de teoria d'autòmats. Existeixen alternatives però s'han seleccionat aquestes degut al seu ús i el suport disponible.

	Nom	Llicència	Descripció
	Node.js	MIT license [8]	Permet l'execució de codi Javascript fora d'un navegador
	npm	Artistic License 2.0 [9]	Gestor de paquets per defecte de Node.js
	Babel	MIT-licensed [10]	Transpilador de Javascript. Converteix codi Javascript entre versions. Permet l'ús de nous estàndards del llenguatge però manté la compatibilitat amb navegadors desactualitzats.

7.2 Llibreria de simulació d'autòmats

7.2.1 Representació interna dels autòmats

Es decideix representar els autòmats com a estructures immutables, un cop creades no es poden modificar directament sinó que s'ha de fer a través de la creació d'una nova instància amb les modificacions desitjades.

Per fer-ho s'utilitza *Immutable.js* una llibreria que internament representarà aquestes estructures immutables de manera que les operacions de crear noves instàncies siguin eficients i segures.

7.2.2 Representació externa dels autòmats

Per permetre que l'usuari pugui visualitzar i modificar els autòmats generats, una primera solució seria exportar i importar directament la representació interna de l'autòmat però això es poc pràctic i a nivell teòric no identifica de manera clara els estats i les transicions.

El format en que s'exporten i s'importen els autòmats serà similar a la manera formal de definir-los, per tant per cada tipus d'autòmat es crearà una gramàtica, a excepció dels llenguatges regulars on addicionalment s'afegirà una gramàtica per parsejar les expressions regulars.

Aquestes gramàtiques es definiran amb el format de *Nearley* una llibreria en Javascript que donada una gramàtica extraurà l'arbre de parsing que posteriorment convertirem a una estructura immutable que representarà l'autòmat.

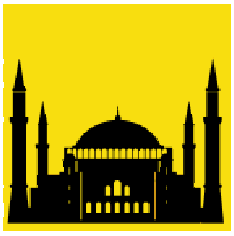
Com a lexer es farà servir *Moo* i es compartirà en totes les gramàtiques per reduir les possibles incoherències entre les definicions dels autòmats.

7.2.3 Testing

Totes les funcions d'aquest apartat estan testejades amb *Mocha* una eina que podem configurar perquè cada vegada que es modifica un fitxer de codi executi tots els tests i generi un resum i en cas que una funció falli indiqui el motiu.

La llibreria *istanbul* permet generar un resum del percentatge i les línies de codi que no s'executen durant els tests. Ajuda a detectar trossos del projecte que no estan testejades i codi obsolet que no es fa servir.

7.2.4 Taula utilitats per simulació autòmats

	Nom	Llicència	Descripció
	immutable.js	MIT-licensed [11]	Proporciona estructures immutables Llistes, Piles, Diccionaris, Conjunts etc.
	Mocha	MIT-licensed [12]	Framework de testing compatible amb Node.js
	Nearley	MIT-licensed [13]	Streaming parser que implementa l'algoritme Earley per parsejar gramàtiques. També permet crear fuzzers i diagrames de les gramàtiques.
	Moo	BSD 3-Clause License [14]	Moo és un analitzador lèxic. Converteix una cadena de text en un conjunt de tokens. És l'analitzador lèxic recomanat per utilitzar conjuntament amb Nearley.
	Istanbul	ISC License [15]	Genera un resum de quines línies i funcions no s'executen en els tests unitaris.

7.3 Front end

Actualment s'utilitzen frameworks que permeten treballar amb una DOM virtual per qüestions d'eficiència, els tres grans projectes que permeten dissenyar una aplicació seguint aquest paradigma són: React, Vue i Angular. S'ha decidit utilitzar React.js perquè actualment és la preferida i la més buscada [16] d'acord amb l'enquesta feta a la pàgina Stack Overflow el 2019.

React és una llibreria de construcció d'interfícies d'usuari. Els elements de la interfície s'afegeixen al DOM i es repinten quan és necessari fent els menors canvis possibles. Els components són funcions que reben paràmetres i retornen elements i són els blocs que de manera jeràrquica construeixen la interfície.

El concepte d'estat en React s'associa als components. Són el conjunt de propietats i atributs que quan es modifiquen és necessari tornar a representar el component. En general aquestes modificacions d'estat son produïdes per un event. Redux és una llibreria que dóna la possibilitat de crear estats 'globals' i mapejar atributs de l'estat global al d'un component. En cas que es modifiqui un atribut de l'estat global automàticament tots els components que depenen d'aquell atribut s'actualitzen.

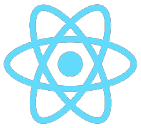
Els tres apartats principals de l'aplicació són cadascun dels autòmats NFA, PDA i TM. Per fer aquesta separació s'utilitza *React Router* una eina que permet accedir als paràmetres de la URL i mostrar un component o un altre en funció del *path*. Tot i que l'aplicació és una SPA (*Single Page Application*) de cares a l'usuari, es pot accedir a qualsevol de les rutes i automàticament es mostra el component corresponent.

Pel que fa a l'estil de l'aplicació s'ha decidit utilitzar *Material UI*, una llibreria de components de *React*, icones i una API de customització que permet organitzar codi *CSS* dins els fitxers de cada component i definir i utilitzar variables globals per fer servir els mateixos estils en qualsevol component i simplificar la cohesió estètica de la interfície.

Pel que fa a la representació dels autòmats, la representació del diagrama d'estats i les transicions es farà amb *mxGraph*. S'ha triat aquesta llibreria perquè inclou funcionalitats a nivell de layout, estil de representació, permet customitzar fonts, colors i s'utilitza en altres grans projectes com *diagrams.net* anteriorment *draw.io*.

Per poder distribuir l'aplicació i poder-la executar és necessari ajuntar tot el codi en un sol fitxer Javascript. Per dur a terme aquesta tasca és necessari un empaquetador, en aquest cas s'utilitza *Webpack*, aquesta eina detecta els fitxers que s'importen i a través del *loader* corresponent al tipus de fitxer l'afegeix a un únic fitxer final que contindrà tota la aplicació.

7.3.1 Principals

	Nom	Llicència	Descripció
	React	MIT-License [17]	Llibreria Javascript per construir interfícies gràfiques.
	React DOM	MIT-License [17]	Inicialitza un component de React a la DOM.
	React Dropzone	MIT-License [18]	Crear components amb suport per accions Drag & Drop.
	Redux	MIT-License [19]	Crear estats globals i accions per modificar-los.
	React Redux	MIT-License [20]	Llibreria per utilitzar els estats de Redux amb React.
	React Router	MIT-License [21]	Llibreria per poder accedir a components a través de la URL del navegador.
	Material UI	MIT-License [22]	Components per construir interfícies gràfiques.
	mxGraph	Apache License 2.0 [23]	Llibreria en Javascript per dibuixar i editar diagrames.

webpack	MIT-licensed [24]	Empaqueta els recursos de l'aplicació, principalment dependències Javascript però existeixen mòduls per *CSS*, *SVG* i altres tipus de fitxer.
Webpack CLI	MIT-licensed [25]	Conjunt de comandes per gestionar la configuració de Webpack.
Webpack Dev Server	MIT-licensed [26]	Servidor local de desenvolupament d'aplicacions amb Webpack que suporta actualització de components en calent.

7.3.2 Mòduls de Webpack

Webpack empaqueta tots els fitxers del projecte en un únic fitxer, per fer-ho detecta tots els *imports* que es fan a l'aplicació i en funció del tipus de fitxer importat farà servir el corresponent *loader* de la següent llista.

Babel Loader	MIT-licensed [27]	Mòdul de Webpack que permet transpilar codi amb babel.
Style Loader	MIT-licensed [28]	Mòdul per injectar *CSS* a la *DOM*
Css Loader	MIT-licensed [29]	Mòdul de Webpack que permet resoldre fitxers d'estil CSS.
Sass Loader	MIT-licensed [30]	Converteix Sass a CSS.
Script Loader	MIT-licensed [31]	Mòdul per carregar i executar llibreries Javascript.
React svg loader	MIT-licensed [32]	Permet afegir fitxers *SVG* al paquet de l'aplicació.

7.3.3 Altres

Altres llibreries que s'han utilitzat per fer el projecte i que la decisió d'utilitzar-les ve donada per les decisions que s'han pres anteriorment.

Node sass	MIT-licensed [33]	Llibreria que permet compilar de manera nativa Sass. Necessari per Sass Loader.
React hot loader	MIT-licensed [34]	Plugin per actualitzar els components de l'aplicació mantenint l'estat durant la fase de desenvolupament.

8 Anàlisi i disseny del sistema

8.1 Casos d'ús

L'aplicació només interacciona amb el navegador local, per tant, només hi ha un actor, l'usuari. En el següent diagrama de casos d'ús en UML es representen els requeriments del programa i els passos que ha de seguir l'actor principal per dur a terme les accions.

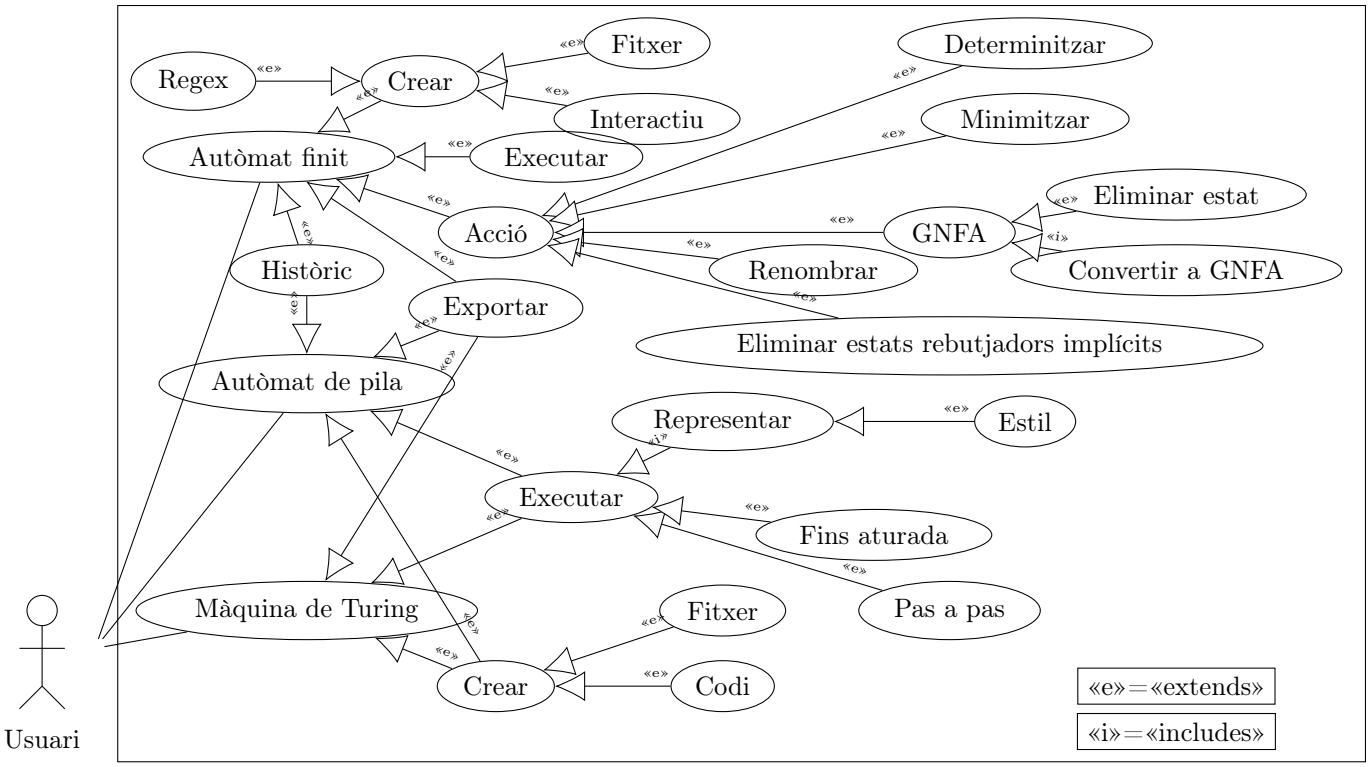


Figura 4: Diagrama de casos d'ús

A continuació es mostrarà en més detall els casos d'ús principals de l'aplicació.

Cas d'ús: #1	Crear un DFA/NFA a partir d'estats i transicions
Descripció	L'usuari vol crear un autòmat definit pel seus estats i transicions
Actor	Usuari
Àrea	Autòmats finits
Flux principal	1. Crear autòmat a partir d'estats inicial, final i transicions. 2. Afegir autòmat a la llista d'historial d'autòmats 3. Assignar nou autòmat actual 4. Executar l'autòmat amb el mot d'entrada o mot buit 5. Representar el diagrama d'estats de autòmat
Flux alternatiu	L'usuari pot carregar la definició de l'autòmat a partir d'un fitxer de text.
Postcondició	A partir de la primera transició introduida es representa el diagrama de l'autòmat i es pot començar a simular amb entrades
Requeriments no funcionals	Els símbols d'entrada poden ser strings, números o un caràcter
Prioritat	Alta

Cas d'ús: #2	Crear un NFA a partir d'una Regex
Descripció	L'usuari vol crear un autòmat
Actor	Usuari
Àrea	Autòmats finits
Flux principal	1. Crear arbre de parsing de l'expressió regular 2. Crear autòmat a partir de l'arbre de parsing 3. Afegir autòmat a la llista d'historial d'autòmats 4. Assignar nou autòmat actual 5. Executar l'autòmat amb el mot d'entrada o mot buit 6. Representar el diagrama d'estats de l'autòmat
Flux alternatiu	En cas que l'expressió regular no sigui vàlida s'indica en quin punt de la cadena hi ha l'error
Postcondició	L'autòmat finit es substitueix per l'actual
Requeriments no funcionals	Els símbols de l'expressió regular poden ser strings, números o un caràcter
Prioritat	Mitjana

Cas d'ús: #3	Executar autòmats NFA i DFA i comprovar si un mot pertany al llenguatge
Descripció	Un cop carregat l'autòmat l'usuari vol comprovar si s'accepta o no un mot
Actor	Usuari
Àrea	Autòmats finits
Precondició	Autòmat esta definit
Flux principal	1. Introduir el mot 2. Executar l'autòmat amb el mot 3. Crear un nou autòmat amb els nous estats i transicions actives 4. Assignar nou autòmat actual 5. Representar el diagrama d'estats de l'autòmat
Requeriments no funcionals	Els símbols de l'input poden ser strings, números o un caràcter
Prioritat	Alta

Cas d'ús: #4	Aplicar una acció {minimitzar, renombrar, determinitzar, convertir a gnfa} a l'autòmat
Descripció	Un cop l'usuari ha carregat un autòmat pot aplicar accions per simplificar-lo, extreure'n una expressió regular equivalent, renombrar els estats o determinitzar-lo.
Actor	Usuari
Àrea	Autòmats finits
Precondició	Autòmat esta definit
Flux principal	1. Obtenir l'autòmat actual 2. Crear un nou autòmat modificat segons la funció corresponent a l'acció 3. Afegir el nou autòmat a l'historial d'autòmats 4. Assignar el nou autòmat actual 5. Executar l'autòmat amb el mot d'entrada o mot buit 6. Representar el diagrama d'estats de l'autòmat
Postcondició	L'autòmat es substitueix per l'actual
Requeriments no funcionals	El nom de l'acció i l'autòmat vell queden guardats a l'historial per la seva posterior identificació i recuperació
Prioritat	Baixa

Cas d'ús: #5	Accedir a un autòmat anterior
Descripció	Si l'usuari ha aplicat una funció o ha modificat l'autòmat, pot tornar a qualsevol versió anterior accedint a l'apartat d'històric
Actor	Usuari
Àrea	Autòmats finits
Precondició	Autòmat esta definit
Flux principal	1. Obtenir llista d'autòmats anteriors 2. Mostrar llista 3. Escollir un autòmat 4. Substituir l'autòmat actual pel seleccionat 5. Executar l'autòmat amb el mot d'entrada o mot buit 6. Representar el diagrama d'estats de l'autòmat
Postcondició	L'autòmat es substitueix per l'actual
Prioritat	Baixa

Cas d'ús: #6	Exportar un autòmat finit
Descripció	L'usuari pot guardar en un format formal la definició de l'autòmat actual, per importar-lo de nou veure Cas d'ús #1
Actor	Usuari
Àrea	Autòmats finits
Precondició	Autòmat esta definit
Flux principal	1. Obtenir autòmat actual 2. Convertir autòmat a format formal 3. Crear un fitxer amb el contingut
Requeriments no funcionals	La exportació i importació d'àutomats es fa en formats de text que l'usuari pot editar i entendre
Prioritat	Mitjana

Cas d'ús: #7	Importar un PDA/NPDA d'un fitxer
Descripció	L'usuari vol carregar un autòmat a partir de la definició dels seus estats i transicions
Actor	Usuari
Àrea	Autòmats de pila
Flux principal	1. Introduir fitxer 2. Crear arbre de parsing del fitxer importat 3. Crear autòmat a partir de l'arbre de parsing 4. Afegir autòmat a la llista d'historial d'autòmats 5. Assignar nou autòmat actual 6. Executar l'autòmat amb el mot d'entrada o mot buit 7. Representar el diagrama d'estats de l'autòmat
Flux alternatiu	En cas que el codi no sigui vàlid s'indica en quin punt de la cadena hi ha l'error
Postcondició	L'autòmat es subtitueix per l'autòmat actual
Requeriments no funcionals	Els formats de text per definir els estats i les transicions de les diferents variants d'autòmats tenen una gramàtica similar
Prioritat	Alta

Cas d'ús: #8	Executar autòmats NPDA i PDA i comprovar si un mot pertany al llenguatge
Descripció	Un cop carregat l'autòmat, l'usuari pot comprovar si accepta o no un mot
Actor	Usuari
Àrea	Autòmats de pila
Precondició	Autòmat esta definit
Flux principal	1. Introduir el mot 2. Executar l'autòmat amb el mot 3. Crear nou/s autòmat/s amb l'estat actiu i la pila modificada com s'indica a la transició 4. Assignar nou/s autòmat/s actual/s 5. Representar el diagrama d'estats d'un autòmat
Requeriments no funcionals	Els símbols de l'input poden ser strings, números o un caràcter
Prioritat	Alta

Cas d'ús: #9	Accedir a un PDA/NPDA anterior
Descripció	Si l'usuari ha modificat l'autòmat pot tornar a qualsevol versió anterior accedint a l'apartat d'històric
Actor	Usuari
Àrea	Autòmats de pila
Precondició	Autòmat esta definit
Flux principal	1. Obtenir llista d'autòmats anteriors 2. Mostrar llista 3. Escollir un autòmat 4. Substituir l'autòmat actual pel seleccionat 5. Executar l'autòmat amb el mot d'entrada o mot buit 6. Representar el diagrama d'estats de l'autòmat
Postcondició	L'autòmat es substitueix per l'actual
Prioritat	Baixa

Cas d'ús: #10	Exportar un autòmat de pila
Descripció	L'usuari pot guardar en un format formal la definició de l'autòmat actual, per importar-lo de nou veure Cas d'ús 'Crear un PDA/NPDA a partir d'estats i transicions'
Actor	Usuari
Àrea	Autòmats de pila
Precondició	Autòmat esta definit
Flux principal	1. Obtenir autòmat actual 2. Convertir autòmat a format formal 3. Crear un fitxer amb el contingut
Requeriments no funcionals	La exportació i importació d'àutomates s'ha de fer en formats de text que l'usuari pugui editar i entendre
Prioritat	Mitjana

Cas d'ús: #11	Importar TM d'un fitxer
Descripció	L'usuari vol carregar una màquina de Turing d'una cinta o multicinta
Actor	Usuari
Àrea	Màquines de Turing
Flux principal	1. Introduir fitxer 2. Crear arbre de parsing del codi de la màquina de Turing 3. Crear TM a partir de l'arbre de parsing 4. Introduir el mot d'entrada a la TM 5. Representar la/les cinta/es i l'estat inicial de la màquina
Flux alternatiu	En cas que el codi no sigui vàlid s'indica en quin punt de la cadena hi ha l'error
Postcondició	S'assigna la TM actual
Requeriments no funcionals	Els formats de text per definir els estats i les transicions de les diferents variants d'autòmats tenen una gramàtica similar
Prioritat	Alta

Cas d'ús: #12	Executar TM/NDTM i comprovar si un mot pertany al llenguatge
Descripció	Un cop carregada la TM l'usuari vol comprovar si s'accepta o no un mot
Actor	Usuari
Àrea	Màquines de Turing
Precondició	TM esta definida
Flux principal	1. Obtenir estat actual de la TM 2. Crear una nova TM on totes les instàncies han avançat un pas 3. Assignar nova TM 4. Representar instàncies i cintes de la TM
Flux alternatiu	Si es modifica l'input de la màquina s'assigna el nou input, es reseteja l'estat de la màquina i s'entra al flux principal d'aquest cas d'ús.
Prioritat	Alta

8.2 Interfícies d'usuari

S'utilitzarà una interfície similar entre els autòmats finits i els de pila, exceptuant que en el primers es poden representar totes les instàncies, o conjunts d'estats que poden estar simultàniament actius en la versió no determinista, en una sola pàgina. En el cas dels autòmats de pila, on cada instància pot tenir un contingut de la pila diferent, es mostrarà una llista de totes les instàncies i un resum on s'indiqui l'estat actual, el tamany de la pila i si accepten o rebutgen el mot.

Les màquines de Turing no es representaran amb un diagrama d'estats, es mostraran les cintes i l'estat actual. Es paginaran les indeterminacions i en cas que una accepti el mot, es permetrà accedir directament a la instància que ho fa.

En tots els casos es permetrà amagar la representació per donar la possibilitat d'executar autòmats sense malgastar temps de còmput renderitzant les animacions o els diagrames.

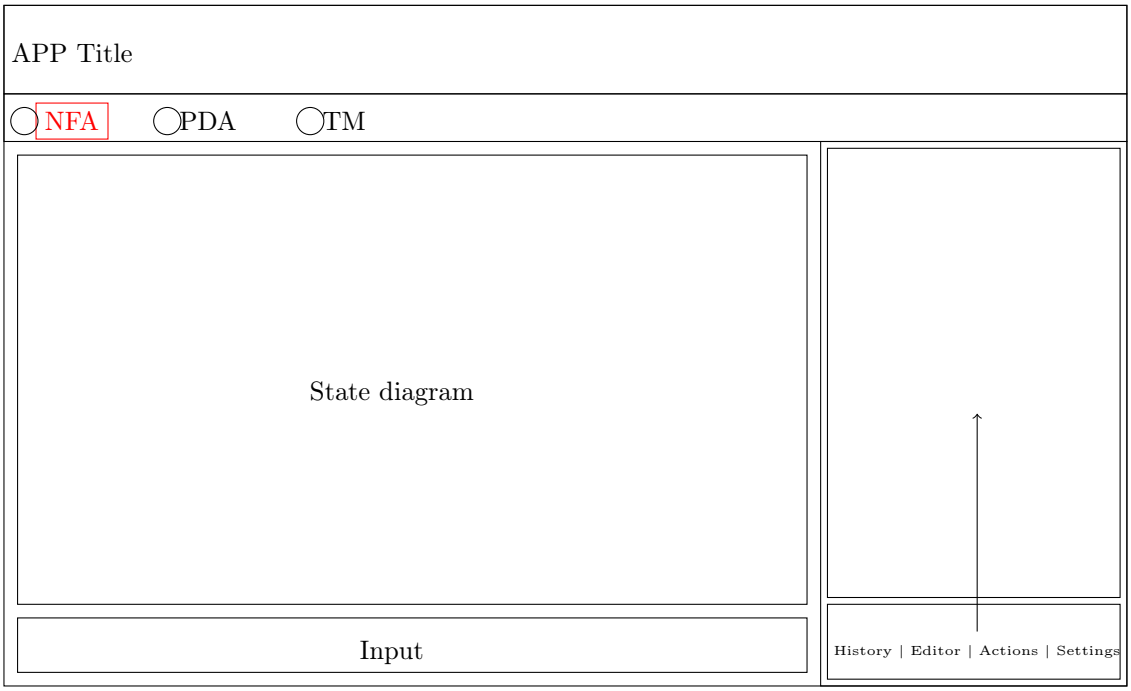


Figura 5: Disseny Interfície NFA

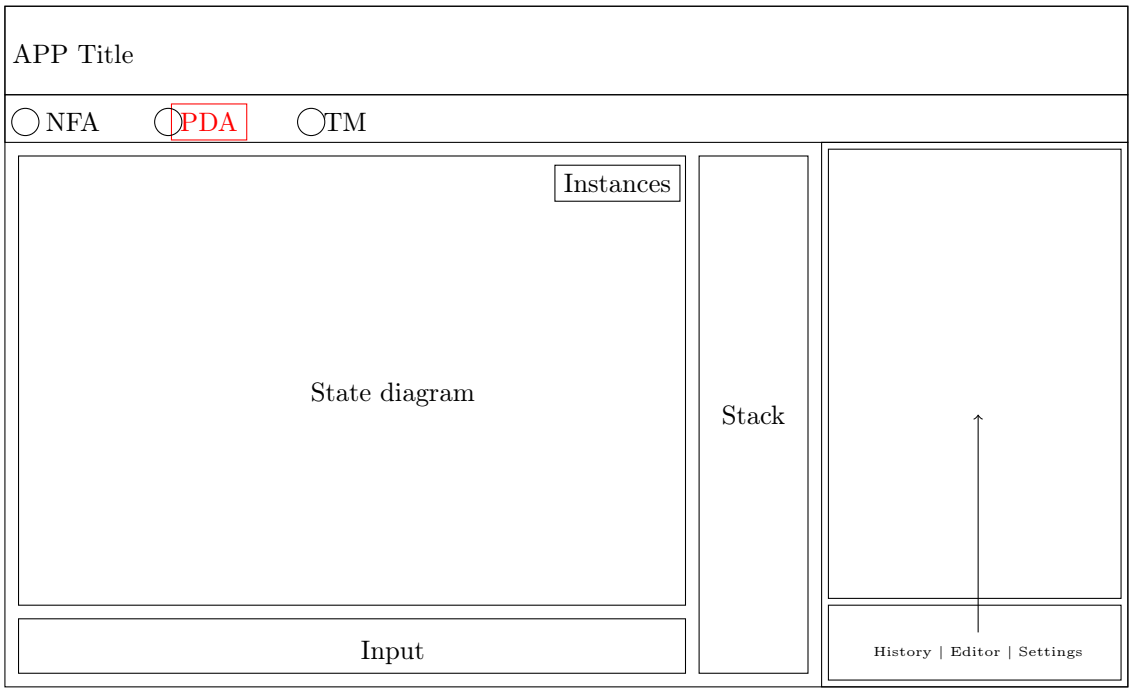


Figura 6: Disseny Interfície PDA

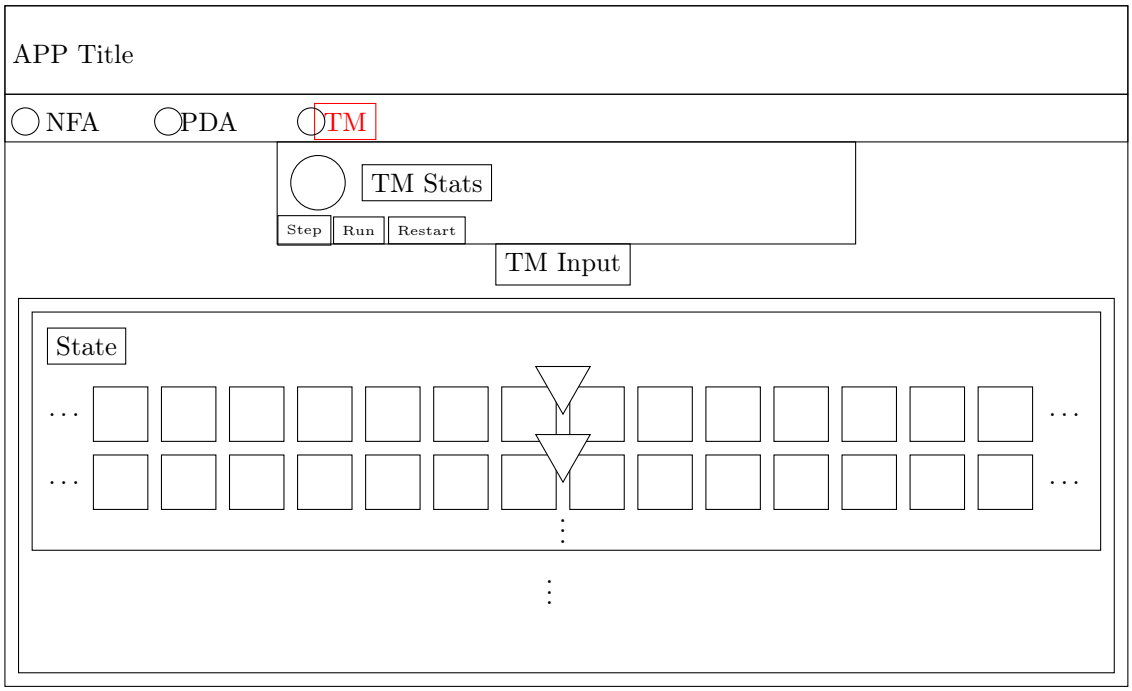


Figura 7: Disseny Interfície TM

8.3 Models de dades dels autòmats

Com s'explica en la següent secció la lògica de l'aplicació en els tres models d'autòmat es tracta la versió determinista com un cas particular de la indeterminista. Les estructures son immutables i per modificar-les cal crear-ne una nova amb les modificacions desitjades, això es fa de manera eficient amb la llibreria *immutable.js*.

```
name         :: String
description  :: String
matrix       :: Map (State, Map (State, Set (State)))
states       :: Map (State, String)
startState   :: State
acceptStates :: Set (State)
activeStates :: Set (State)
activeEdges  :: Map (State, Set (State))
```

Figura 8: Atributs d'un autòmat

```
name         :: String
description  :: String
states       :: Map(State, String)
matrix       :: Map(State,
                    Map(SimbolEntrada,
                        Map(SimbolStack,
                          List(Map(State, List(SimbolStack)
                        )
                    )
                )
startState   :: State
acceptStates :: Set(State)
instances    :: List(Map(State, Stack(Simbol)))
initialStack :: Stack(Simbol)
```

Figura 9: Atributs d'un autòmat de pila

```
name         :: String
description  :: String
states       :: Map(State, String)
matrix       :: Map(State,
                  Map(List(SimbolCinta),
                    List(
                      Map(State,
                        Map(List(SimbolCinta), List(Moviment))
                      )
                    )
                  )
                )
steps        :: Int
tapeCount    :: Int
active       :: List(Map(State, List(Map(IndexCinta, List(SimbolCinta)))))
```

Figura 10: Atributs de la màquina de Turing

8.4 Disseny de la gramàtica dels autòmats

S'ha dissenyat una gramàtica per cada un dels tres models d'autòmats i una per les expressions regulars. Els autòmats tenen una descripció molt similar i s'ha intentat que les gramàtiques ho fossin. Es permet afegir un sobrenom i una descripció als estats. També es permet anomenar l'autòmat i afegir una descripció del funcionament.

Els llenguatges d'exemple que indiquen en la descripció una pàgina s'han tret del llibre *Introduction to the Theory of Computation* de Michael Sipser [1].

A continuació es mostra un exemple de cada una de les gramàtiques i la representació en diagrama d'estats de l'autòmat que descriuen.

```
name: "Sum modulo 3 + RESET"
description: "Example 1.13, pàg. 39"

states:
q0*- = q0 "0 Mod 3"
q1   = q1 "1 Mod 3"
q2   = q2 "2 Mod 3"

instructions:
(q0, 0    ) -> [q0]
(q0, 1    ) -> [q1]
(q0, 2    ) -> [q2]
(q0, RESET) -> [q0]
(q1, 0    ) -> [q1]
(q1, 1    ) -> [q2]
(q1, 2    ) -> [q0]
(q1, RESET) -> [q0]
(q2, 0    ) -> [q2]
(q2, 1    ) -> [q0]
(q2, 2    ) -> [q1]
(q2, RESET) -> [q0]
```

Figura 11: Fitxer de definició de l'autòmat

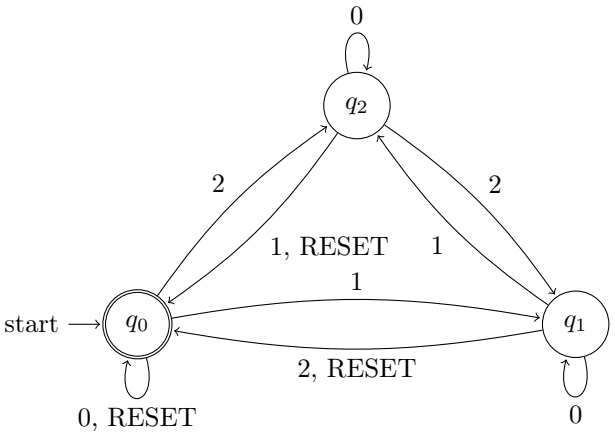


Figura 12: Representació en diagrama d'estats

Figura 13: Example 1.13, pàg. 39

Els apartats que descriuen l'autòmat son 'states' i 'instructions'.

En l'apartat d'estats s'indiquen quins son, un sobrenom per referir-s'hi, una descripció i si són acceptadors '-' o '*' finals o les dues coses, com en cas de la figura anterior. L'estructura és `<alias><modificadors> _ = _ <nom> _ "<descripció>"`.

En l'apartat d'instruccions s'indiquen totes les transicions entre estats, en el cas dels autòmats finits, la funció de transició és $\delta(q_i, w) = q_j$ i es representaria (q_i, w) `->` $[q_j]$, el motiu dels [] és perquè en el cas no determinista es permet accedir a varis estats per un sol símbol d'entrada. En aquest cas es posarien els diferents q_j separats per comes $[q_{j_0}, q_{j_1}, \ldots]$.

```
name: "{a^{i}b^{j}c^{k}| i,j,k >= 0 and i = j or j = k}"
description: "Example 2.17 pàg. 114"

states:
q1* = q1
q2   = q2
q3   = q3
q4-  = q4
q5  = q5
q6  = q6
q7-  = q7

instructions:
(q1, ϵ, ϵ) -> [(q2, $)]
(q2, a, ϵ) -> [(q2, a)]
(q2, ϵ, ϵ) -> [(q3, ϵ), (q5, ϵ)]
(q3, b, a) -> [(q3, ϵ)]
(q3, ϵ, $) -> [(q4, ϵ)]
(q4, c, ϵ) -> [(q4, ϵ)]
(q5, b, ϵ) -> [(q5, ϵ)]
(q5, ϵ, ϵ) -> [(q6, ϵ)]
(q6, c, a) -> [(q6, ϵ)]
(q6, ϵ, $) -> [(q7, ϵ)]
```

Figura 14: Fitxer de definició de l'autòmat

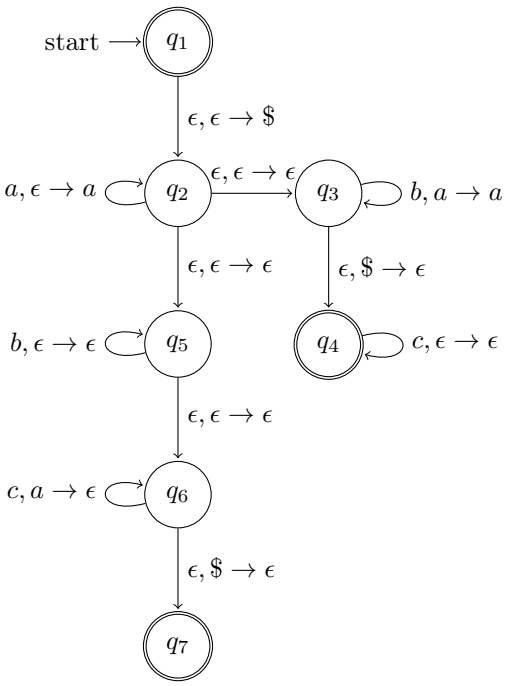


Figura 15: Representació en diagrama d'estats

Figura 16: Example 2.17, pàg. 114

La gramàtica dels autòmats de pila és molt similar a la de dels autòmats finits. En el cas dels estats es fa servir el mateix format per definir-los. Tant l'alfabet de l'entrada com el de la pila es dedueixen a través de les transicions i no cal identificar-los. En el cas de les transicions del PDA $\delta(q_i, w, a) = (q_j, b)$ on a,b són símbols de la cinta i w de l'alfabet d'entrada, es representen en (q_i, w, a) `->` $[(q_j, b)]$.

En la tercera línia de les instruccions de la figura anterior es pot veure un exemple d'indeterminació d'un estat a molts.

```
name: "{0^{2^{n}} | n >= 0}"
description: "Fig. 3.8, pàg 144"

states:
q1* = q1
q2 = q2
q3 = q3
q4 = q4
q5 = q5
q6- = qaccept

instructions:
(q1, 0) -> [(q2, E, r)]
(q2, x) -> [(q2, x, r)]
(q2, 0) -> [(q3, x, r)]
(q2, E) -> [(q6, E, r)]
(q3, x) -> [(q3, x, r)]
(q3, 0) -> [(q4, 0, r)]
(q3, E) -> [(q5, E, l)]
(q5, 0) -> [(q5, 0, l)]
(q5, x) -> [(q5, x, l)]
(q5, E) -> [(q2, E, r)]
(q4, x) -> [(q4, x, r)]
(q4, 0) -> [(q3, x, r)]
```

Figura 17: Fitxer de definició de l'autòmat

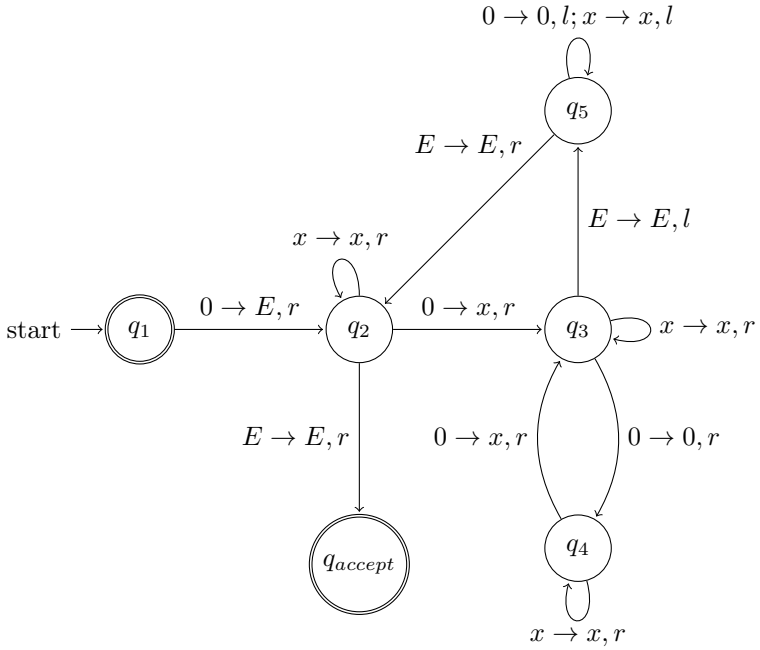


Figura 18: Representació en diagrama d'estats

Figura 19: Fig. 3.8, pàg 144

En la gramàtica de les TM també es manté la mateixa estructura de definició d'estats. En el cas de les TM es llegeixen els símbols de la cinta que inicialment conté el mot. Es fa servir la 'E' per indicar un espai buit. La funció de transició de les TM és $\delta(q_i, a) = (q_j, b, M)$, on a i b son són símbols de lectura i escriptura i M és un moviment de capçal, es representa en (q_i, a) -> $[(q_j, b, M)]$. Com en els casos anteriors es poden fer transicions a varis estats simultàniament.

Per fer servir vàries cintes es farà amb el separador vertical '|'. Per exemple$(q_i, a|a')$ -> $[(q_j, b|b', M|M')]$ on a', b' i M' serien els símbols llegits i escrits a la segona cinta i M' el moviment. No hi ha límit en el nombre de cintes que es poden fer servir. Si en una transició es fan servir k cintes, en la resta de transicions també hi ha d'haver k cintes.

La gramàtica de les expressions regulars és molt senzilla però té algunes opcions que faciliten la representació de llenguatges regulars. En aquest exemple es poden veure totes les característiques implementades: $a^*b\{3\}(a?(a|b)a)$. Apart de la concatenació, la unió i l'estrella s'han afegit dos modificadors extra comuns en la majoria de Software on s'utilizen les expressions regulars. El modificador ? s'aplica sobre l'expressió regular anterior R_1 i equival a l'expressió $(R_1|\epsilon)$. El modificador {k} on k és un natural, modifica l'expressió regular anterior R_2 i equival a l'expressió $(R_{2_1} R_{2_2} \dots R_{2_k})$.

9 Implementació i proves

En tots els autòmats es pot considerar la versió determinista com un cas particular de la no determinista, de cares a l'aplicació es tractaran tots els autòmats com a no deterministes i es deixarà en mans de l'usuari que ho sigui o no.

En el cas dels autòmats finits això podria comportar problemes, no és el mateix minimitzar un NFA que un DFA. En aquest cas s'ha optat per intentar aplicar els algoritmes i en cas que es detecti que l'autòmat no és determinista o no compleix les restriccions, indicar-ho a l'usuari. En el cas dels NFA existeix la opció de determinitzar-los.

Pel que fa als autòmats de pila i les màquines de Turing en les seves versions no deterministes el que es fa és clonar la màquina per cada indeterminació i en cada pas de la simulació avançar un pas la simulació de cada una d'elles. Aquesta exploració correspon a una cerca *Breadth-first* o per nivells. Aquesta cerca garanteix que encara que una màquina de Turing no s'aturi, el programa no quedi penjat i segueixi buscant una acceptació del mot en la resta d'instàncies.

9.1 Problemes

Un problema que no s'ha pogut solucionar completament, sorgeix quan es vol representar un autòmat amb la llibreria mxGraph. Cada vegada que es fa una operació sobre un autòmat, per exemple canviar el mot d'entrada, se'n crea un de nou. Al repintar aquest autòmat es recreen tots els vèrtex i arestes, és un procés molt més lent del que es podria fer, ja que normalment només és necessari canviar el color dels vèrtexs actius. La solució seria utilitzar un algoritme similar al que es fa amb la DOM virtual per poder comparar el nou autòmat amb l'antic i actualitzar només els nodes necessaris del graf representat. Per pal·liar aquest problema de rendiment s'han fet esforços per evitar repintats innecessaris.

9.2 Algoritmes

Els algoritmes més interessants es troben en l'apartat dels autòmats finits. A continuació es descriuen les transformacions i els processos amb els que s'han dut a terme.

Determinitzar NFA	DFA que reconeix el mateix llenguatge que un NFA [5]
Minimitzar DFA	Eliminar estats indistingibles amb Myhill-Nerode Theorem [35]
Eliminar un estat d'un GNFA	Construint un GNFA amb un estat menys [36]

Figura 20: Algoritmes utilitzats en autòmats finits

9.2.1 Determinitzar NFA

L'objectiu de l'algoritme és generar un autòmat finit determinista equivalent a un no determinista. La idea general és crear un estat al DFA per cada una de les combinacions possibles d'estats actius al NFA.

Es parteix de l'estat inicial i es marquen tots els estats actius incloent als que s'hi accedeix amb ϵ i s'afegeix un únic estat al DFA que els representi. Si l'NFA té actius $\{q_0, q_1, q_2\}$ en l'estat inicial, al DFA s'afegeix l'estat $q_0 \# q_1 \# q_2$. En cas que com a mínim un d'aquests estats sigui acceptador, l'estat afegit també ho serà.

Recursivament i per tots els símbols de l'alfabet del llenguatge s'avança el conjunt d'estats actius, en l'exemple anterior $\{q_0, q_1, q_2\}$, i s'afegeixen les noves combinacions d'estats al DFA i la respectiva transició amb el símbol. Aquest procés es repeteix per tots els símbols assolint totes les combinacions possibles d'estats de l'NFA. Els estats del DFA equivalent $\subset$ Powerset(Estats de NFA original). En aquest procediment no s'afegeixen al DFA combinacions d'estats actius impossibles de l'NFA.

En cas que no hi hagi cap estat actiu a l'NFA, es crea una transició al DFA a un estat de rebuig del que no se'n pot sortir.

```
getDFA(nfa) :: NFA -> DFA
  dfa := inicialitzarDFA()

  explore(xs) :: [State] -> State
    if xs.isEmpty: // Afegir estat de rebuig. Ignorar si ja hi era.
      dfa = addState(dfa, "∅")
      dfa = addTransition(dfa, from: "∅", to: "∅", symbol: forall a in alphabet(nfa))
      return "∅"

    // Unificar tots els estats actius en un sol
    nfa = setActiveStates(nfa, xs)
    active := getActiveStates(nfa)  // Inclou rutes amb epsilon
    stateName := active.join("#")

    if stateName in states(dfa):
      return stateName

    addState(dfa, stateName)
    if isAccepted(nfa):
      dfa = addFinalState(dfa, stateName)

    for s in alphabet(nfa):
       nfa' := step(nfa, active_states: active, symbol: s)
       xs' :=  getActiveStates(nfa')
       stateName' := explore(xs')
       dfa = addTransition(dfa, from: stateName, to: stateName', symbol: simbol)

    return stateName;
```

```
  startState := explore([getStartState(nfa)]);
  dfa = setStartState(dfa, startState);
  return dfa
```

9.2.2 Minimització

En la minimització es fa servir el teorema Myhill-Nerode [35]. Dos estats $\{p, q\}$ son indistingibles si $\delta^*(p, w) \in F \iff \delta^*(q, w) \in F$, $w \in \Sigma^*$. Fusionar dos estats $\{p, q\}$ equival a canviar totes les transicions d'entrada $\delta(x, w) = p$ a $\delta(x, w) = q$ i les transicions de sortida $\delta(p, w) = x$ a $\delta(q, w) = x$.

La idea principal d'aquest algoritme consisteix en trobar totes les parelles d'estats distingibles. Quan aquest procés acaba ha trobat totes les parelles distingibles. Les que quedin sense marcar son indistingibles i es poden fusionar sense modificar el llenguatge.

El procés comença amb dos conjunts d'estats clarament distingibles, els que accepten i els que no. Es marquen totes les parelles possibles entre aquests dos conjunts com distingibles.

```
A := F
B := Q \ F
forall a in A, b in B:
  mark(a, b)

end := false
while not end:
  end = true
  forall p in Q, q in Q, p < q, not marked(p, q):
    forall a in alphabet(DFA):
      if marked(delta(p, a), delta(q, a)):
        mark(p, q)
        end = false
```

En la segona part de l'algoritme es fa una cerca per força bruta d'un conjunt $\{w, p, q\}$ on $w \in \Sigma$, p, q $\in Q$ tal que la parella $\{\delta(p, w), \delta(q, w)\}$ estigui marcada. Si es troba, també es marca la parella {p, q} doncs és distingible amb el símbol w. S'atura quan ja no queden parelles d'estats $\{p, q\}$ que siguin distingibles.

Totes les parelles que no s'han marcat són indistingibles i es poden unificar entre elles. El Teorema garanteix que el resultat és l'autòmat amb menys estats necessaris per decidir el llenguatge.

9.2.3 Eliminar un estat d'un GNFA

El primer pas per obtenir l'expressió regular d'un NFA, és convertir-lo a GNFA. Un GNFA ha de complir: [37]

1. L'estat inicial només té fletxes de sortida, no hi pot haver cap estat que torni a l'estat inicial.
2. L'estat final no pot tenir fletxes de sortida, només d'entrada. L'estat final es diferent a l'inicial.
3. Exceptuant l'estat inicial i el final, tots els estats tenen una fletxa a tots els estats i a ells mateixos.

Per complir aquests tres requisits en un NFA qualsevol, s'afegeixen dos estats, un nou estat inicial amb una transició epsilon a l'antic estat inicial, i un estat d'acceptació on tots els antics estats d'acceptació hi arriben amb epsilon. Si hi ha múltiples fletxes entre els mateixos dos estats amb el mateix sentit s'uneixen en una sola. Si entre dos estats no hi ha transició afegir-la amb un $\emptyset$. [37]

El procés per obtenir la expressió regular equivalent al llenguatge consisteix en eliminar estats fins que només queda l'inicial i el final. Per eliminar un estat i mantenir el llenguatge es modifiquen totes les transicions $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$ on $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, $R_4 = \delta(q_i, q_j)$. On q_{rip} és l'estat a eliminar. [37]

Exemple eliminació q_1:

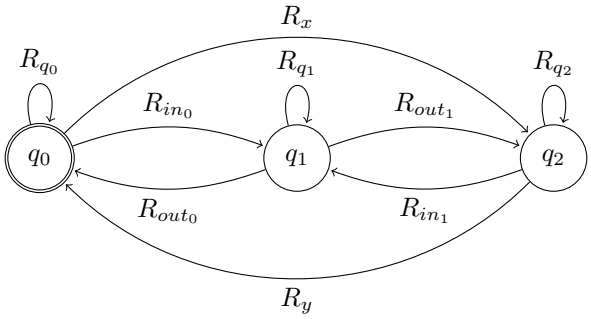


Figura 21: Algoritme eliminació d'estat de GNFA. Estats que porten o surten de q_1

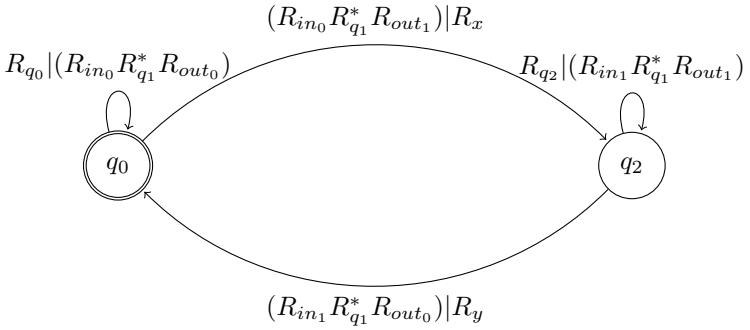


Figura 22: Algoritme eliminació d'estat de GNFA. Estat eliminat

9.3 Tests unitaris i coverage

La llibreria de simulació d'autòmats s'ha desenvolupat seguint la metodologia TDD, en la interfície gràfica les proves de funcionament s'han fet de manera manual. S'inclouen a continuació els tests

unitaris de la llibreria d'autòmats.

```
DFA, NFA
  Crear NFA
     #createNFA
     #getActiveStates
     #getAcceptStates
     #getTransitions
  Llenguatge reconegut és correcte
     Computar mots del llenguatge
     Computar mots que no formen part del llenguatge
  Unió d'autòmats
     #union Amb el mateix autòmat (mateix llenguatge)
     Computar mots del llenguatge
     Computar mots que no formen part del llenguatge
     #union Unir amb un NFA de llenguatge diferent (A*)
     Computar mots del llenguatge
     Computar mots que no formen part del llenguatge
  Contatenar autòmats
     #concat (automat amb ell mateix)
     Computar mots del llenguatge
     Computar mots que no formen part del llenguatge
  Klenee star
     #star (AA)*
     Computar mots del llenguatge
     Computar mots que no formen part del llenguatge
  Determinitzar NFA -> DFA
     #getDFA
     Computar mots del llenguatge
     Computar mots que no forma part del llenguatge
  Minimize
     Minimitzar DFA
     Computar mots del llenguatge
     Computar mots que no formen part del llenguatge
  Delete implicit rejects
     Delete implicit reject states
  Rename states
     Rename states
  GNFA
     Convertir a GNFA
     Eliminar tots estats (excepte inicial i final)
  Exportar NFA
     Exportar NFA
     Importar NFA
PDA
  #parsePDA
     {0^{n}1^{n} | n >= 0} from file
     Computar una paraula del llenguatge
```

```
      Computar una paraula que no pertany al llenguatge
      {a^{i}b^{j}c^{k}| i,j,k >= 0 and i = j or j = k}
      Computar una paraula del llenguatge
      Computar una paraula que no pertany al llenguatge
      {ww^{R} | w ∈ {0, 1}*}
      Computar una paraula del llenguatge
      Computar una paraula que no pertany al llenguatge
    Exportar PDA
      Exportar PDA
      Importar PDA
  REGEX
    Convertir Regex a NFA
      Crear un NFA a partir d'una Regex
      Computar mots del llenguatge
      Computar mots que no formen part del llenguatge
    Testejar Regex i operacions amb NFA
      Minimize Regex (binary multiple of 3) (174ms)
      Regex 111(0|1)?
      Regex 1+2+
      Regex 1{3}
  TM/NDTM
    Turing Machine
      Crear turing machine
      Obtenir els símbols de la primera instància
      Avançar un step la primera instància
      Step en màquina multitape determinista
      Parser de turing machines
      Maquina multitape no determinista i multicinta
      Parser de turing machines
      {0^{2^{n}} | n >= 0}
```

És interessant conèixer quina o quines parts del codi de la llibreria no s'està testejant. S'ha utilitzat la llibreria especialitzada *istanbul* que indica quines línies no s'han executat i quin percentatge de cada fitxer de codi s'ha executat. L'execució d'aquest programa dóna la sortida següent.

```
-------------------------|---------|----------|---------|---------|-------------------
File                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------|---------|----------|---------|---------|-------------------
All files                |   98.07 |    73.68 |   98.13 |   97.85 |
 src                     |   98.73 |    95.76 |   98.54 |   98.63 |
  NFA.js                 |   99.42 |    97.26 |     100 |   99.37 | 571,588
  PDA.js                 |    94.9 |    81.25 |   93.33 |   94.57 | 18-39,172
  REGEX.js               |     100 |      100 |     100 |     100 |
  TM.js                  |     100 |      100 |     100 |     100 |
 src/grammars            |     100 |      100 |     100 |     100 |
  helpers.js             |     100 |      100 |     100 |     100 |
  lexer.js               |     100 |      100 |     100 |     100 |
```

S'observa que durant els tests s'executen la majoria de línies de codi, no és garantia de la inexistència d'errors, però és útil durant el desenvolupament per millorar la qualitat del codi.

10 Implantació i resultats

En iniciar l'aplicació es visualitzarà la pàgina de la Fig. 23, aquí es mostren els tres models d'autòmats i un llistat d'exemples de cada un d'ells. Es farà un recorregut visual de l'aplicació començant pels autòmats finits seguit dels autòmats de pila i per acabar les màquines de Turing.

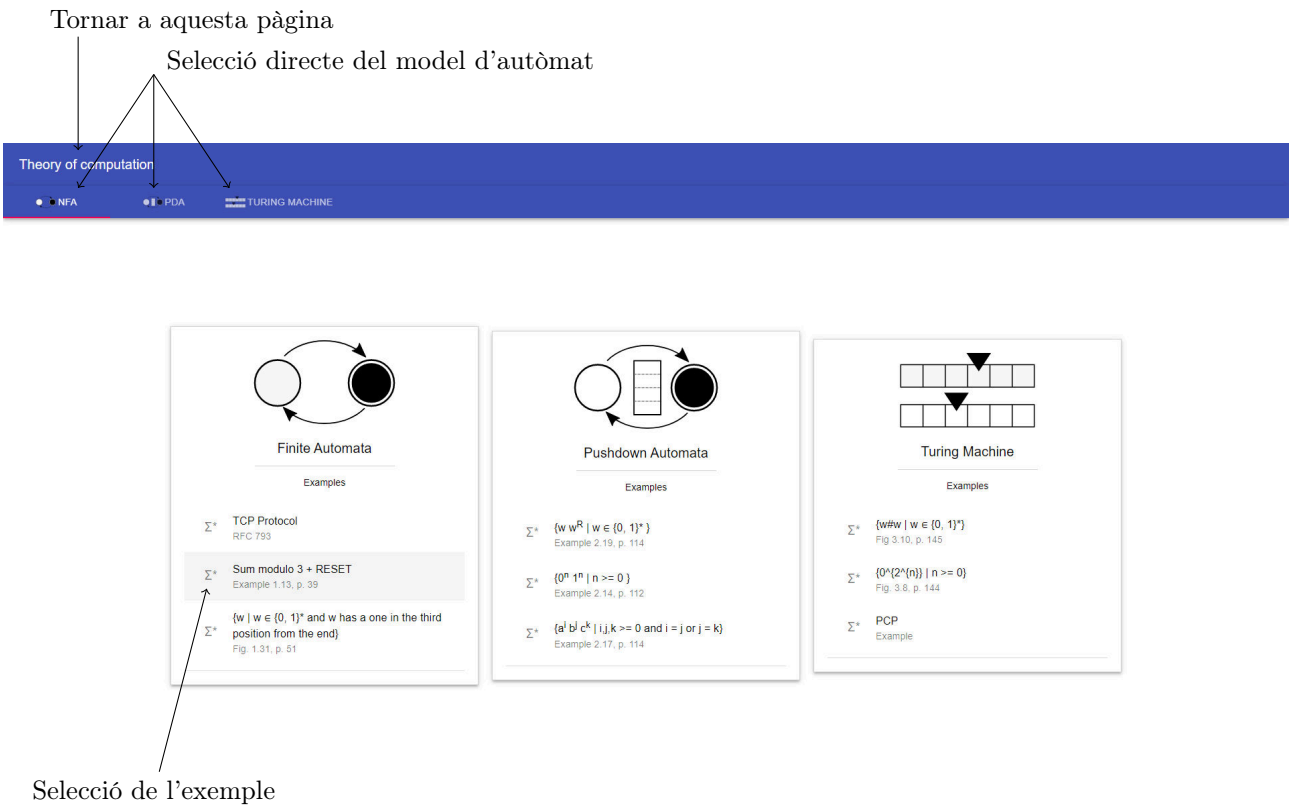


Figura 23: Pantalla d'inici

En aquesta pàgina inicial Fig. 23 es resumeixen els continguts que comprèn aquesta aplicació i es donen uns exemples característics de cada un dels models de computació. En qualsevol pàgina de l'aplicació es podrà tornar a aquest punt prement 'Theory of computation' a la part superior esquerra de la finestra. Sota el botó anterior hi ha un accés directe a cadascun dels models, també estarà sempre disponible. El fet de canviar de model o pàgina, mai fa perdre els continguts en memòria de l'aplicació i es pot reprendre l'activitat al punt on es deixa tornant a la pàgina corresponent.

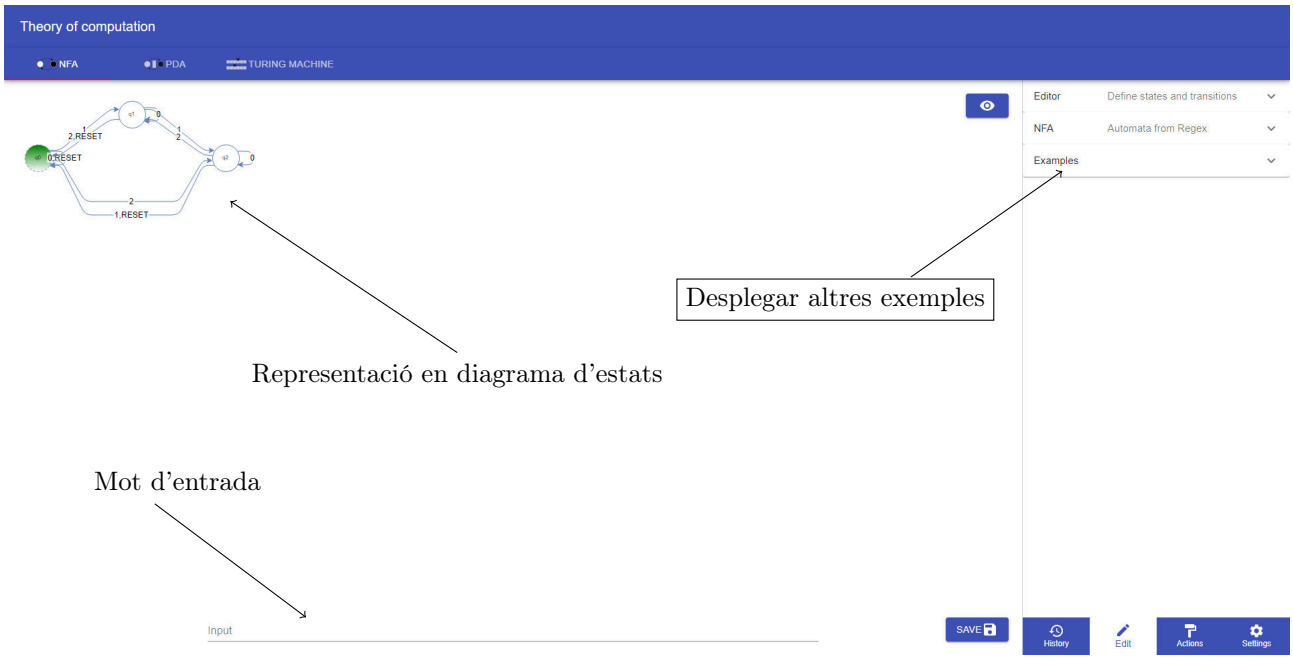


Figura 24: Exemple d'autòmat finit determinista carregat en el pas anterior

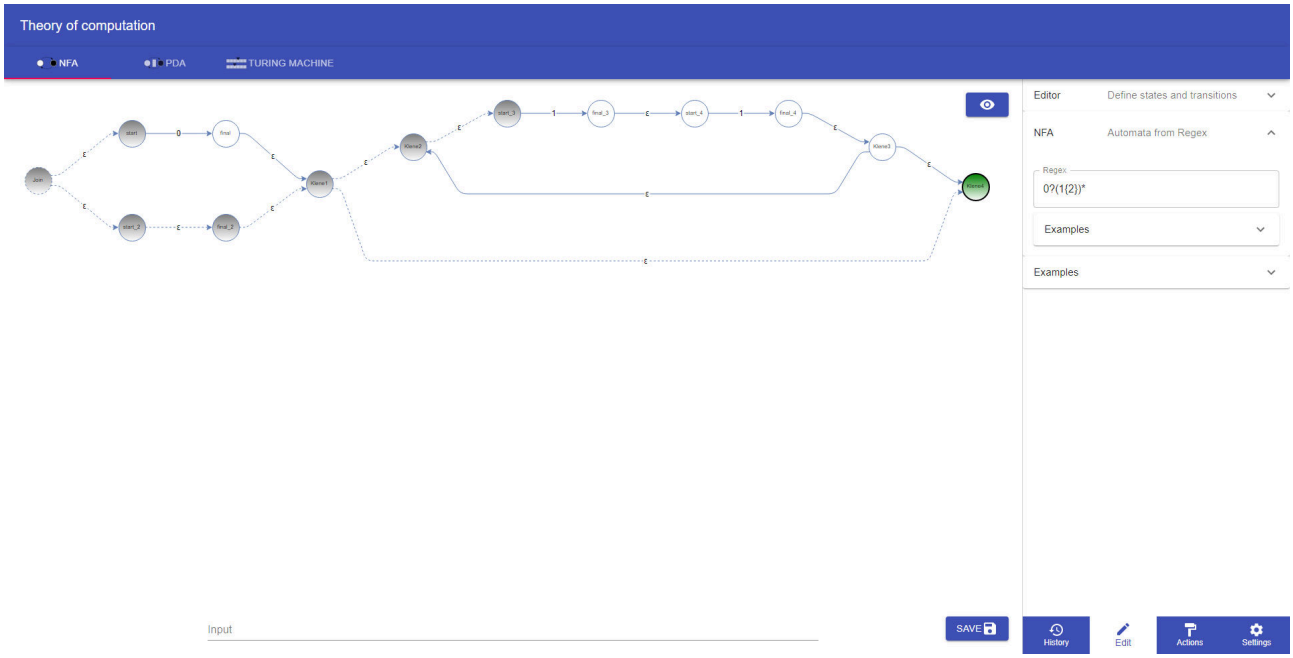


Figura 25: Carregar un NFA a partir de la Regex $0?(1\{2\})^*$

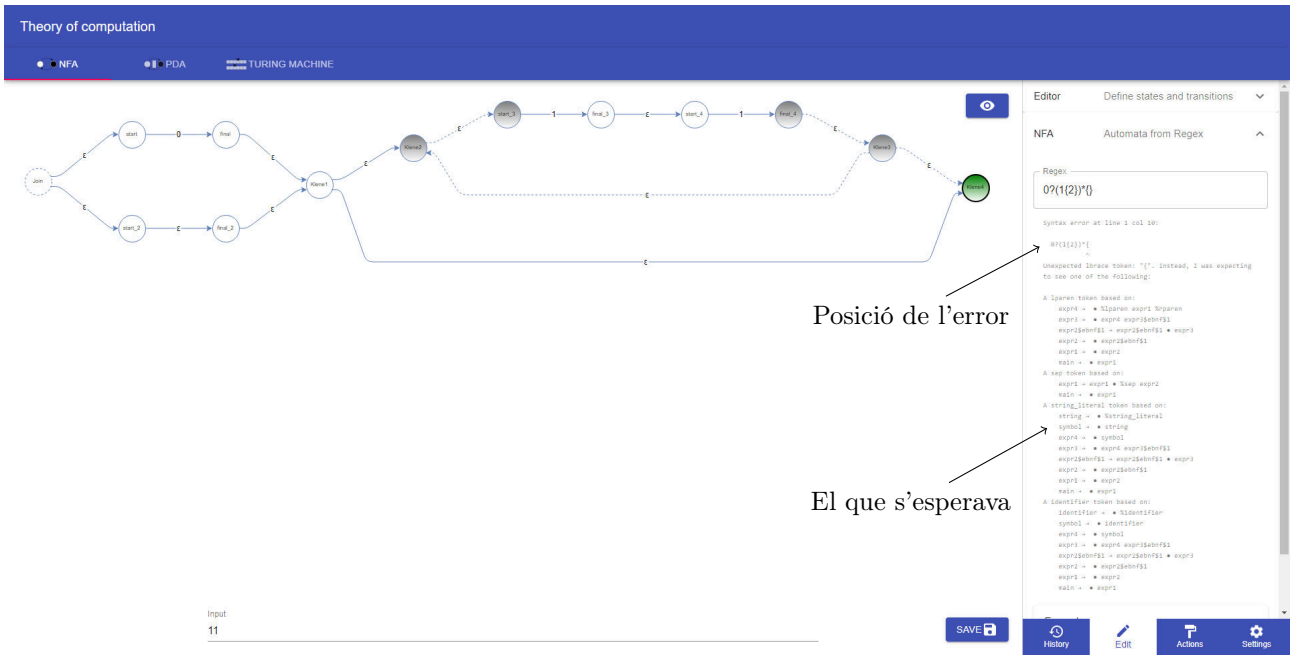


Figura 26: Indicació de la posició dels errors a la Regex 0?(1{2})*{} i el que s'esperaria perquè fos correcte

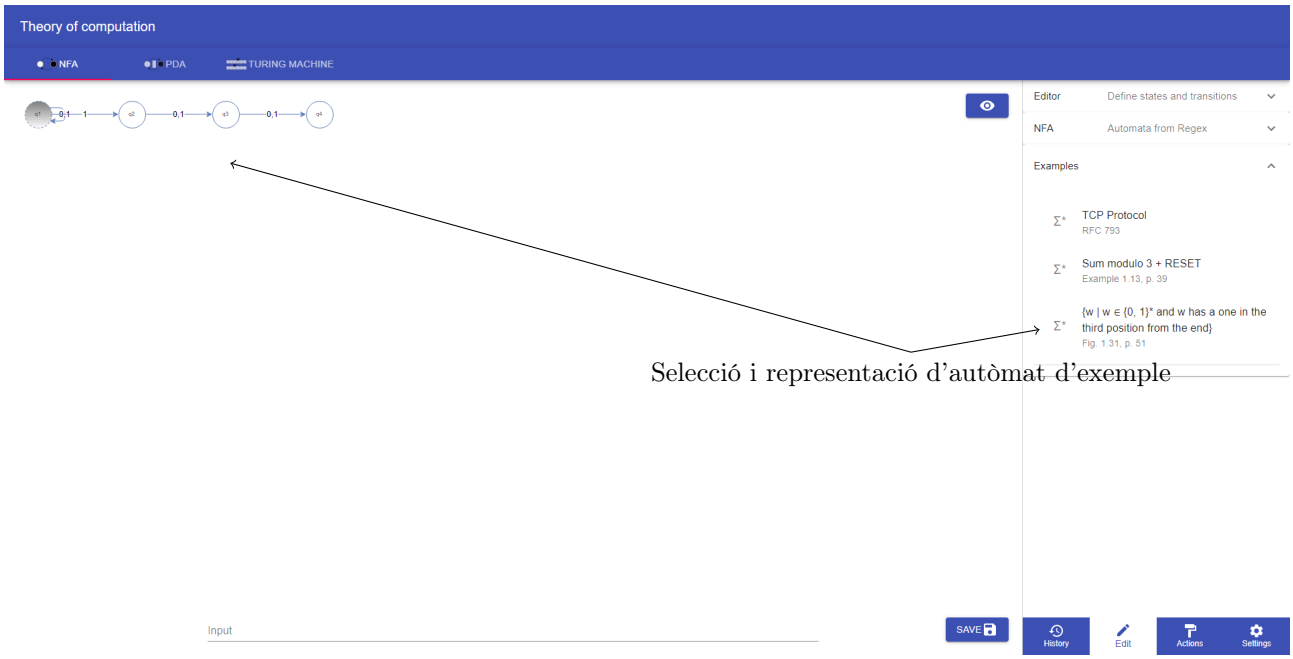


Figura 27: Exemple d'autòmat finit no determinista

S'ha mostrat la representació i la càrrega d'autòmats. A continuació es mostraran les accions i transformacions que es poden aplicar sobre aquests. Seguint l'exemple de la Fig. 27 es mostrà com determinitzar-lo.

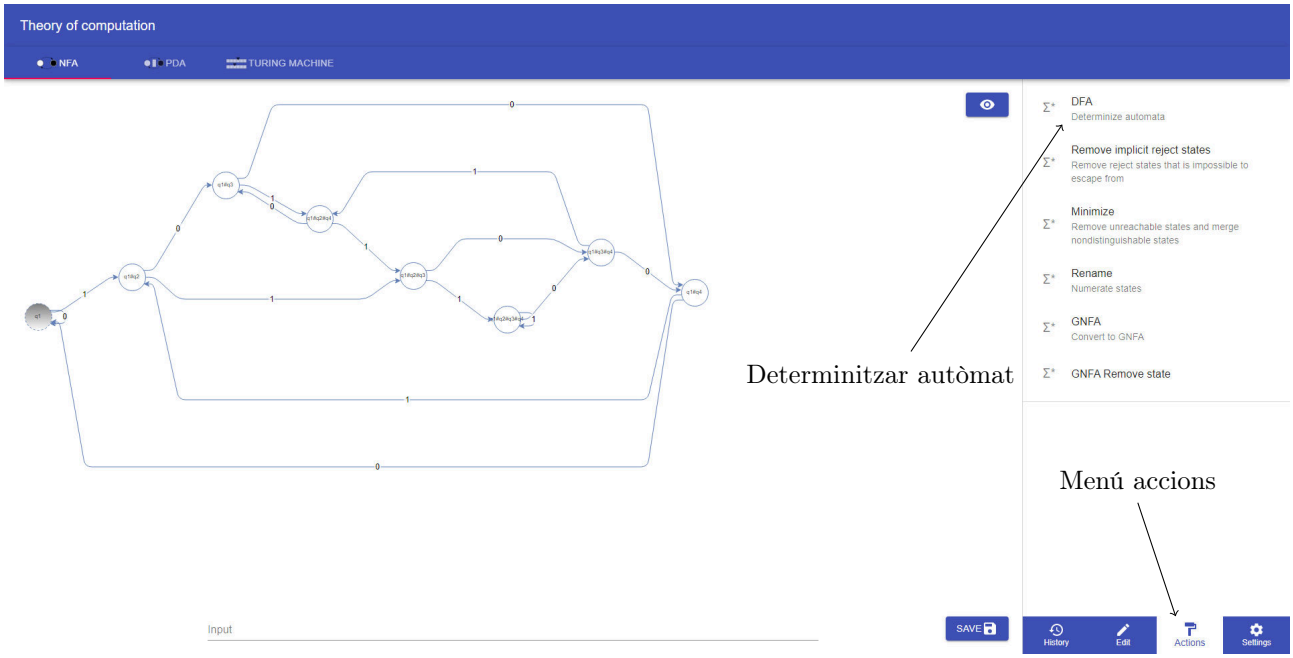


Figura 28: Autòmat determinista equivalent al de la figura 27

A continuació es farà un recorregut per totes les accions disponibles. Es partirà d'una expressió regular, s'obtindrà un autòmat no determinista amb el mateix llenguatge, es determinitzarà i es minimitzarà, és a dir, Regex $\rightarrow$ NFA $\rightarrow$ DFA $\rightarrow$ DFA$_{min}$. Amb l'objectiu de mostrar la resta de funcionalitats, un cop s'obtingui DFA$_{min}$ es generarà un GNFA i s'eliminaran estats fins a obtenir l'expressió regular del llenguatge fent el camí invers, DFA$_{min}$ $\rightarrow$ GNFA$_k$ $\rightarrow$ GNFA$_2$. En aquest punt es disposa de dues expressions regulars pel mateix llenguatge, per verificar que son equivalents, es carrega aquesta nova Regex al programa i tornarem a fer Regex $\rightarrow$ NFA $\rightarrow$ DFA $\rightarrow$ DFA$_{min}$ i es verifica que els dos DFA$_{min}$ son equivalents (només canvien els noms dels estats).

L'expressió regular que es farà servir per dur a terme el procés descrit anteriorment serà la següent $(0|(1(01^*(00)^*0)^*1)^*)^*$ que descriu el llenguatge dels nombres binaris múltiples de tres. Se'n pot veure el DFA$_{min}$ en la Fig 3.

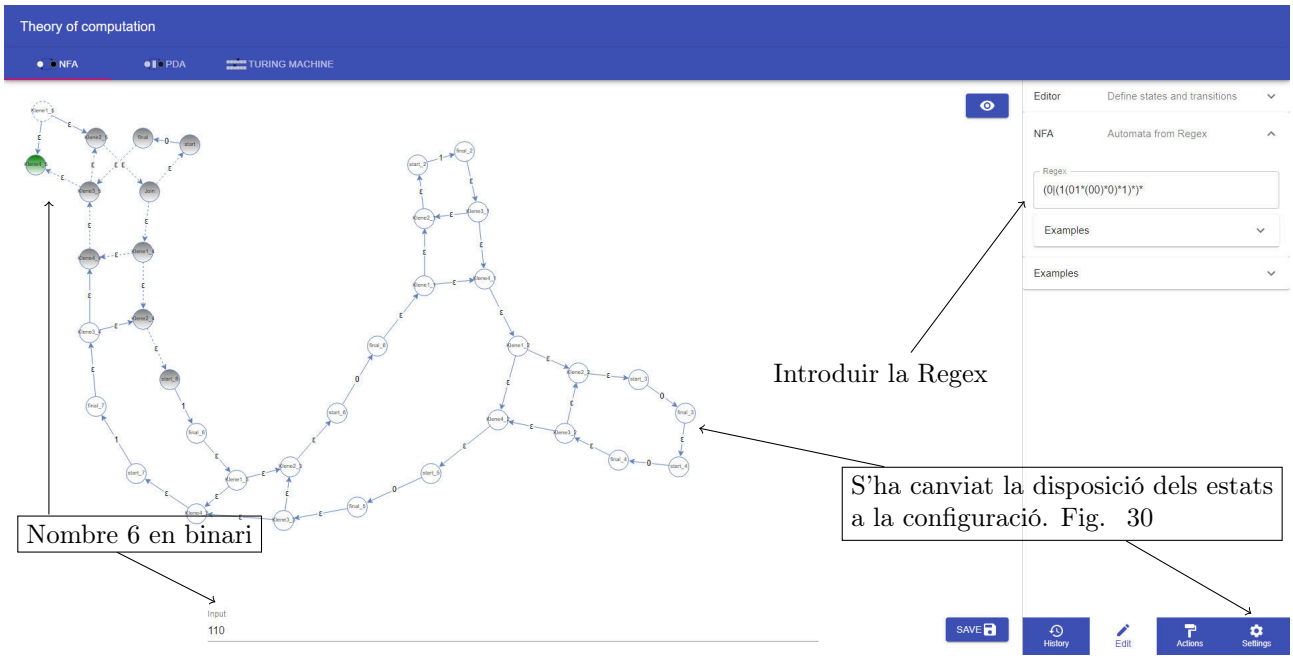


Figura 29: NFA generat a partir de la Regex $(0|(1(01^*(00)^*0)^*1)^*)^*$, Regex → NFA

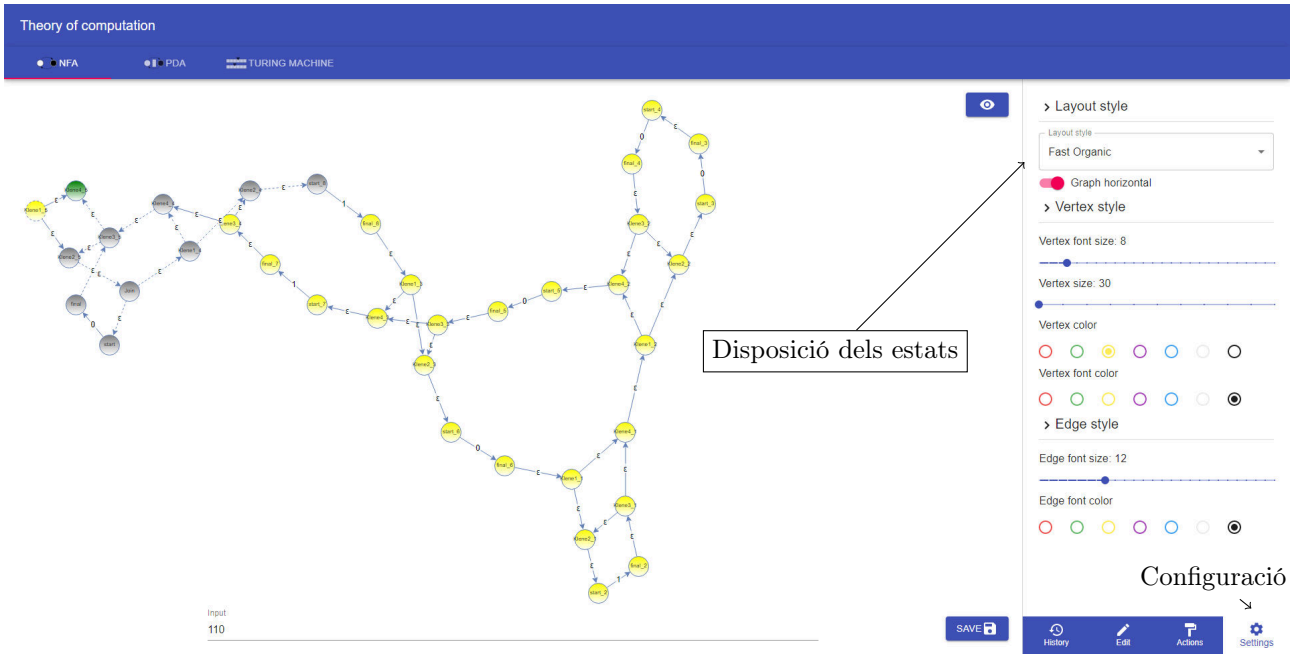


Figura 30: Configuració de representació del diagrama d'estats

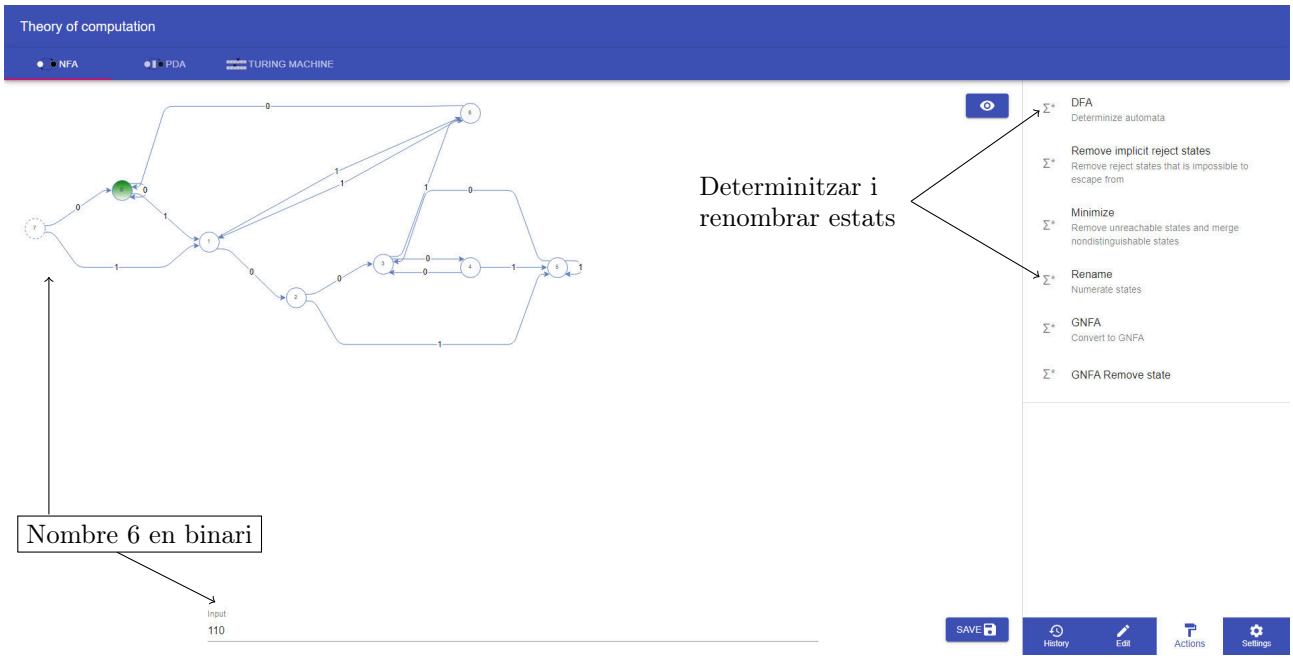


Figura 31: NFA → DFA

Figura 32: DFA → DFA_{min}

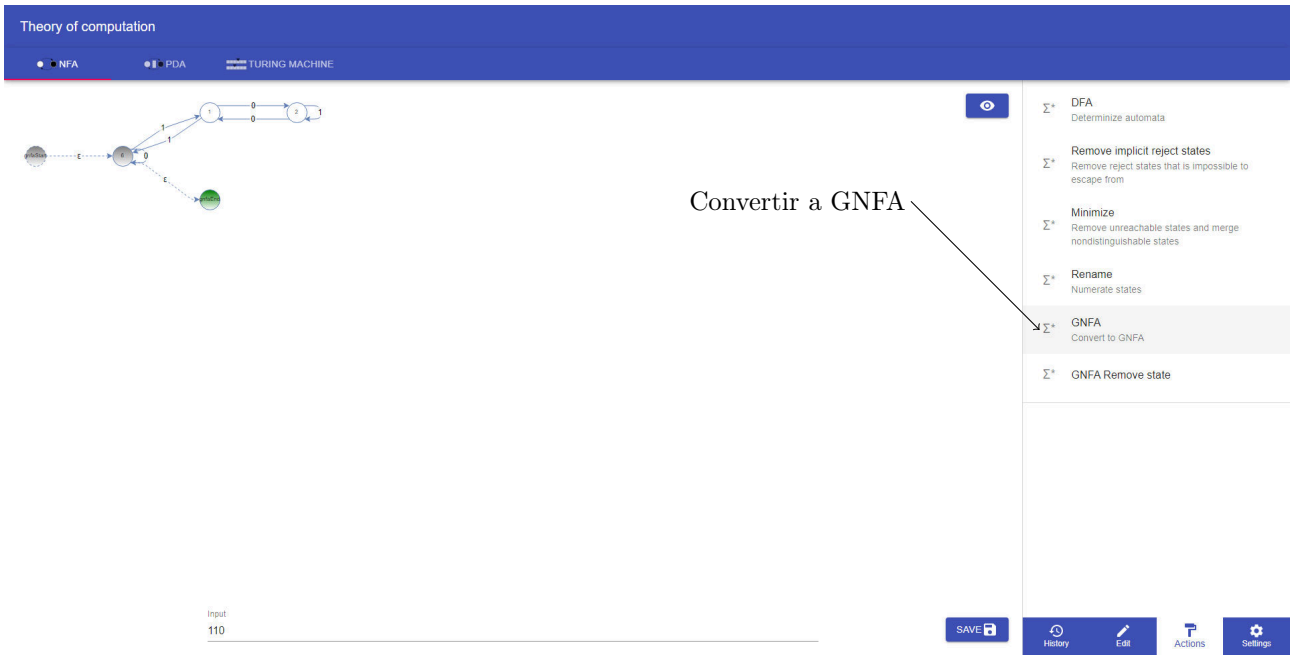


Figura 33: Comença el procés invers per obtenir una Regex. $\text{DFA}_{min} \rightarrow \text{GNFA}_{k=5}$

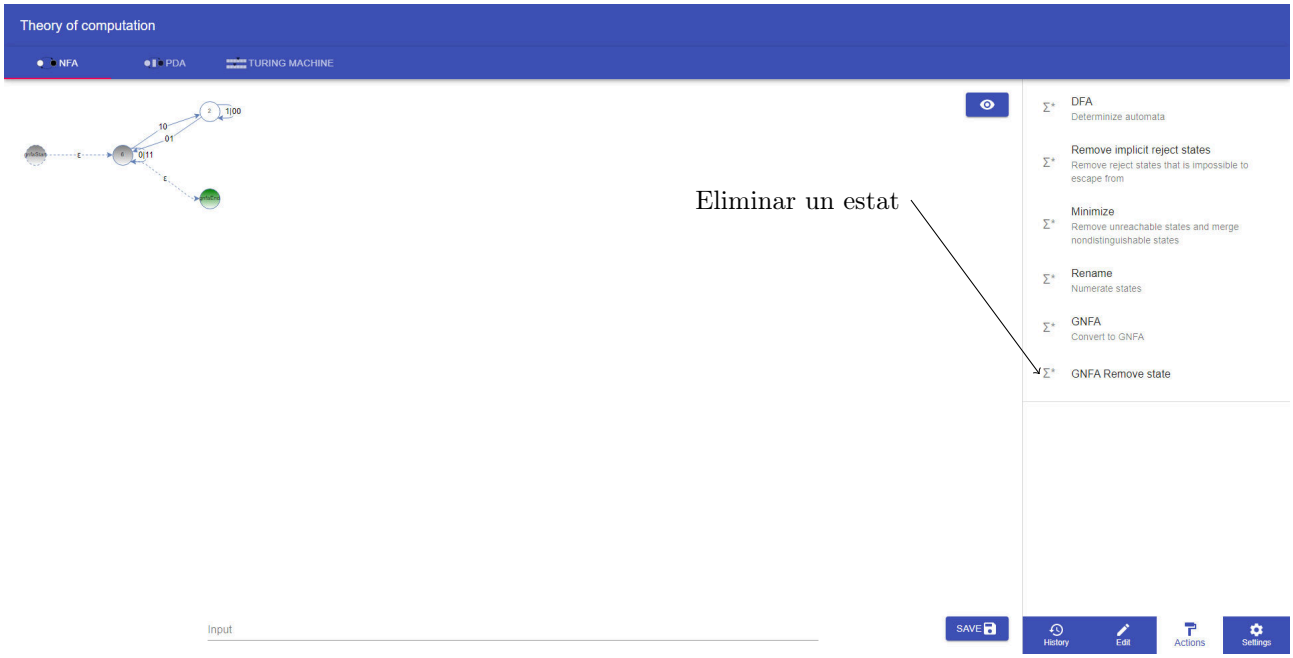


Figura 34: S'eliminen estats fins k=2. $\text{GNFA}_{k=5} \rightarrow \text{GNFA}_{k=4}$

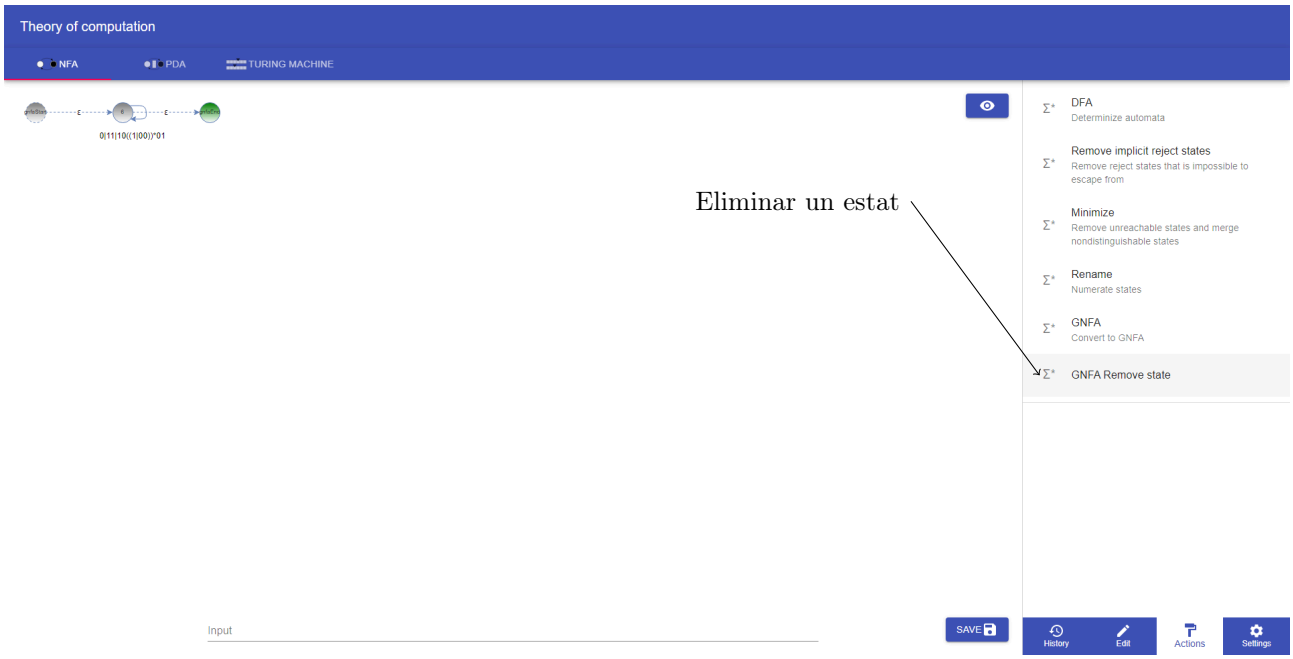


Figura 35: S'eliminen estats fins k=2. $\text{GNFA}_{k=4} \rightarrow \text{GNFA}_{k=3}$

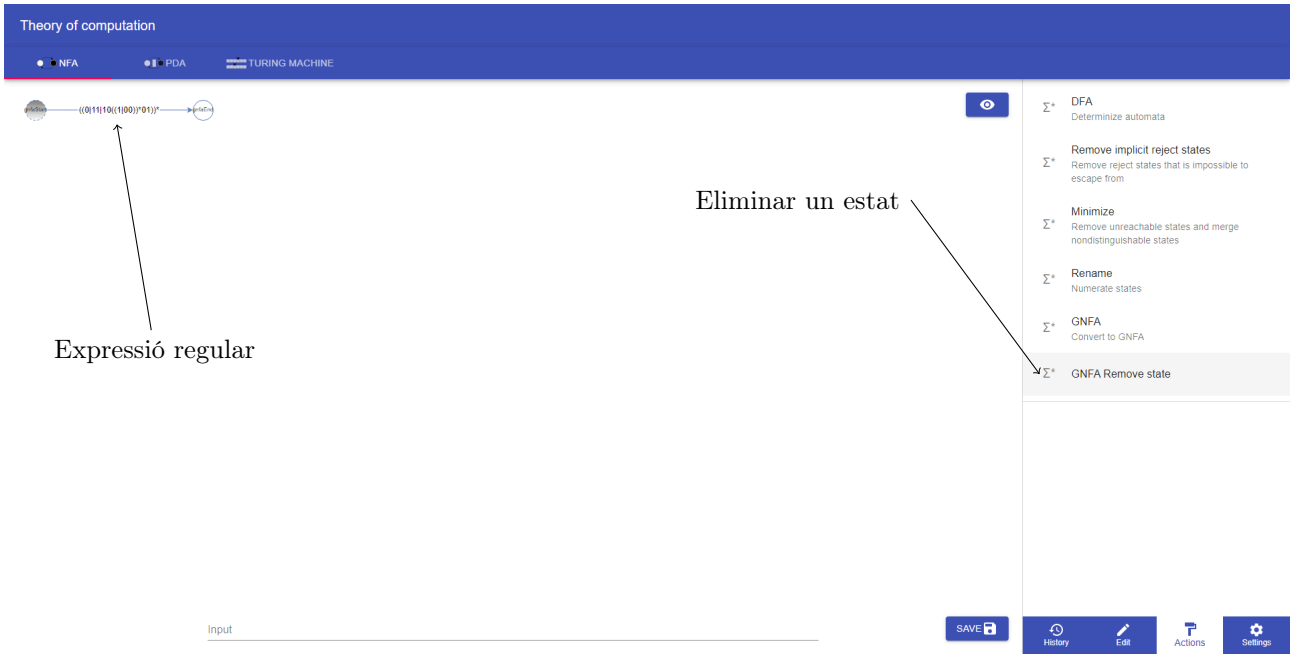


Figura 36: Obtenim l'expressió regular i procedim a comprovar que son equivalents a través del DFA_{min}. $\text{GNFA}_{k=3} \rightarrow \text{GNFA}_{k=2}$

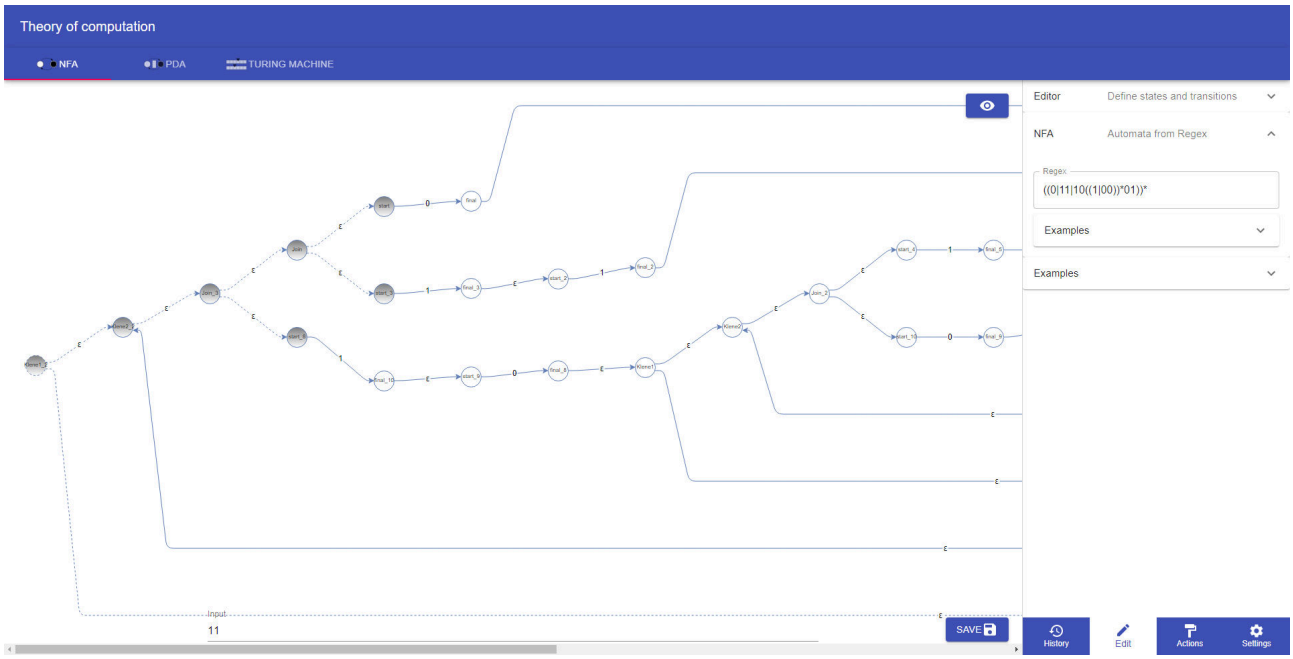


Figura 37: Es carrega l'expressió regular obtinguda. Regex → NFA

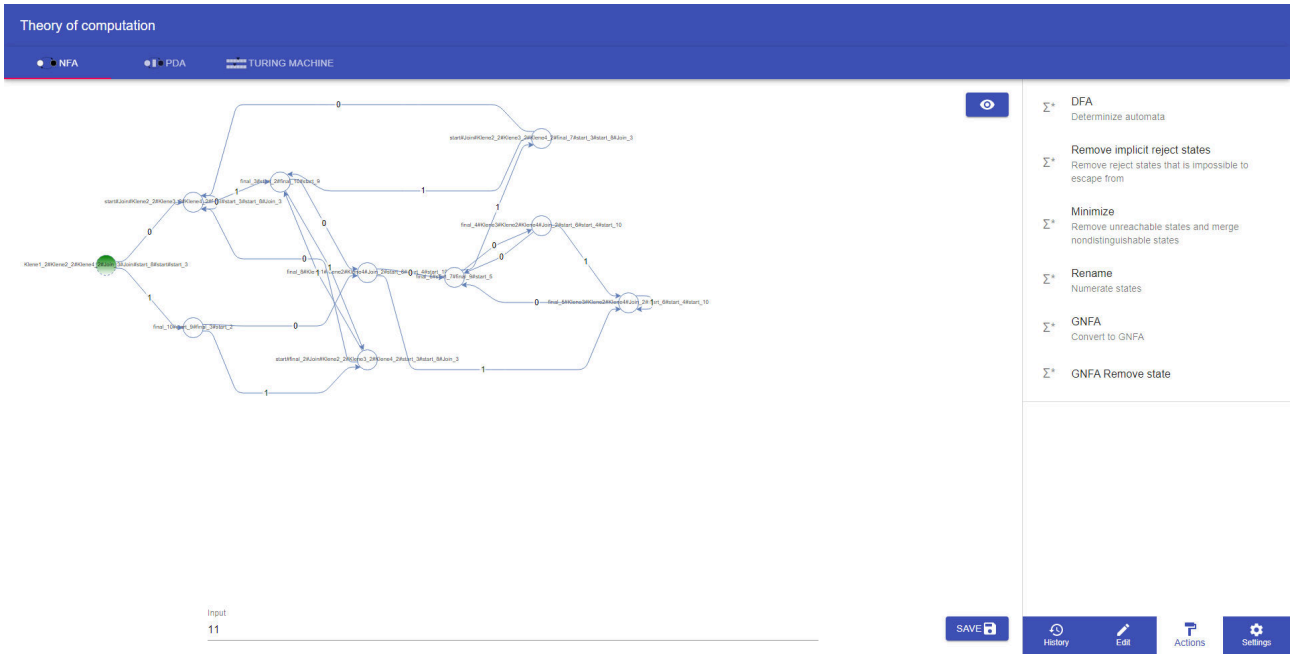


Figura 38: Es determinitza. NFA → DFA i al minimitzar el DFA s'obté el mateix autòmat

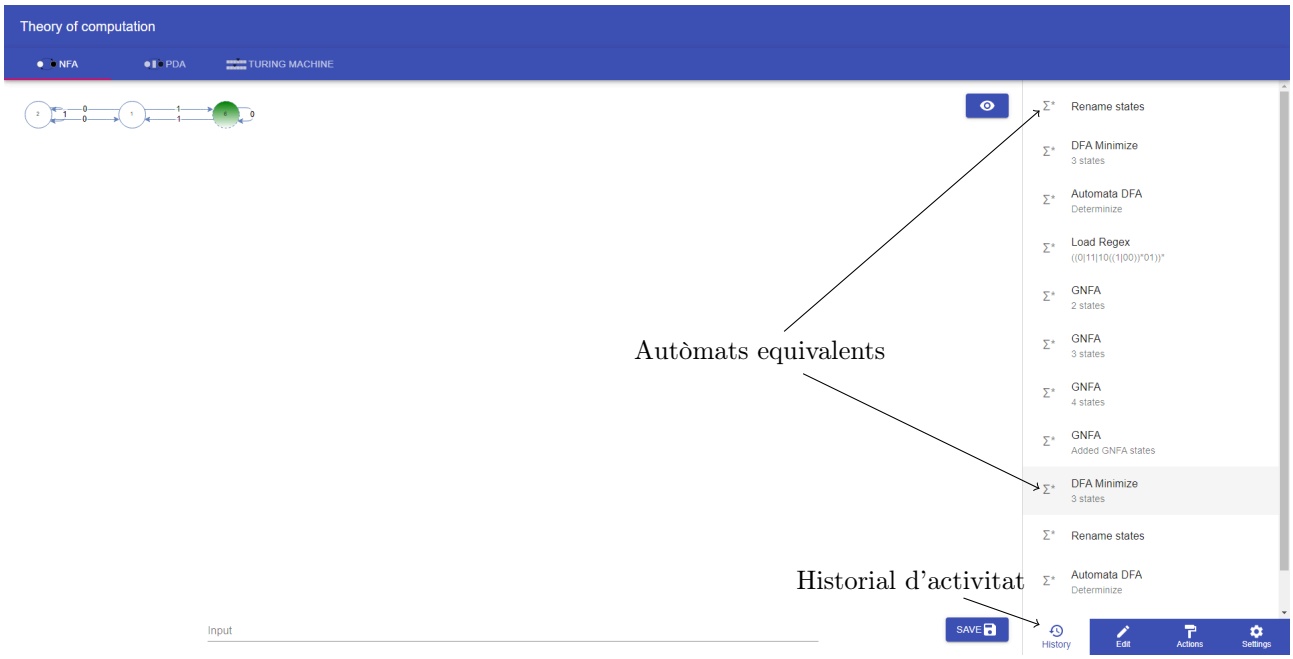


Figura 39: En el pas anterior ens pot ser d'ajuda la funció d'historial que permet recuperar l'autòmat minimitzat de la Fig. 32 per verificar que efectivament son iguals

Figura 40: Diagrama TCP. Visualització de la descripció dels estats mantenint el cursor a sobre durant un segon

El funcionament de la interfície dels autòmats de pila és similar al dels autòmats finits. Totes les opcions de representació del diagrama d'estats segueixen disponibles.

A continuació es mostrarà en detall els apartats i les possibilitats d'aquests. En la pantalla d'inici de la Figura 23 es selecciona el segon llenguatge d'exemple i es carrega automàticament el diagrama d'estats de l'autòmat.

Figura 41: PDA que reconeix el llenguatge 0^n1^n. Interfície similar a NFA

Com s'explica en la Secció 9, cada vegada que l'autòmat s'indetermina, es genera un nou autòmat per cada indeterminació. En la Fig. 41 s'indica com accedir a totes les indeterminacions i es mostra com es veu el llistat en la Fig. 49.

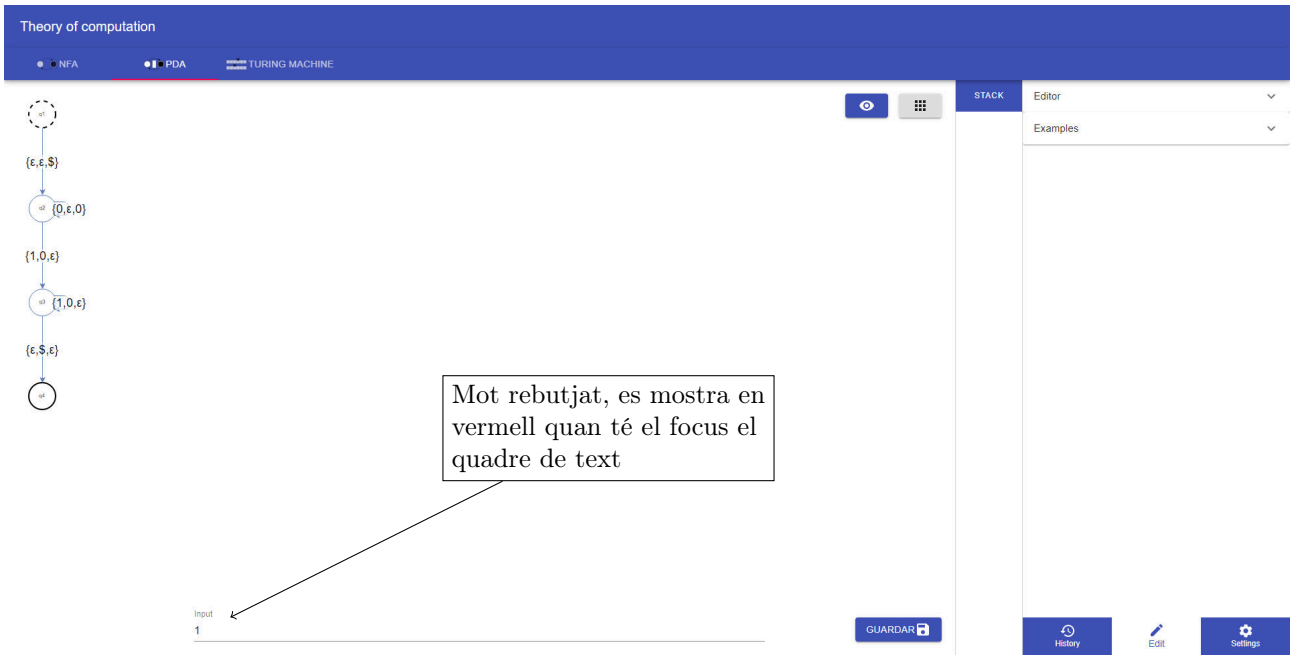


Figura 42: Introduir un mot que no pertany al llenguatge

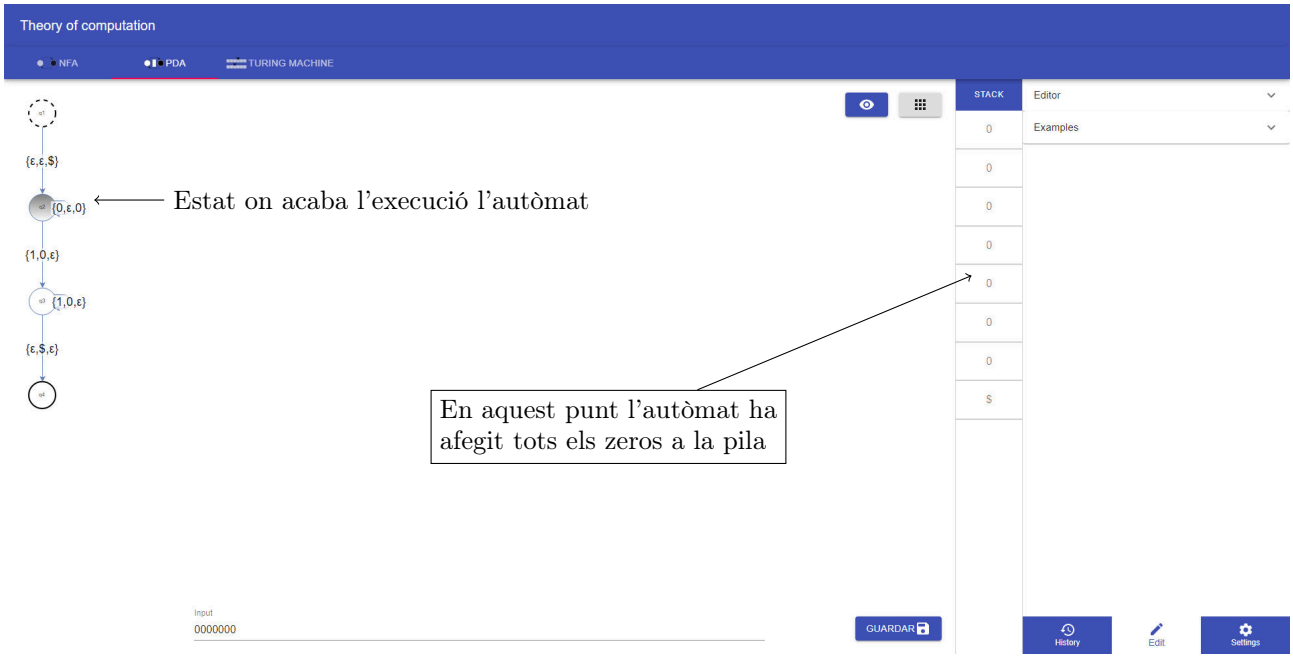


Figura 43: Exemple amb un mot rebutjat però que si s'afegissin els 1 necessaris al final seria acceptat

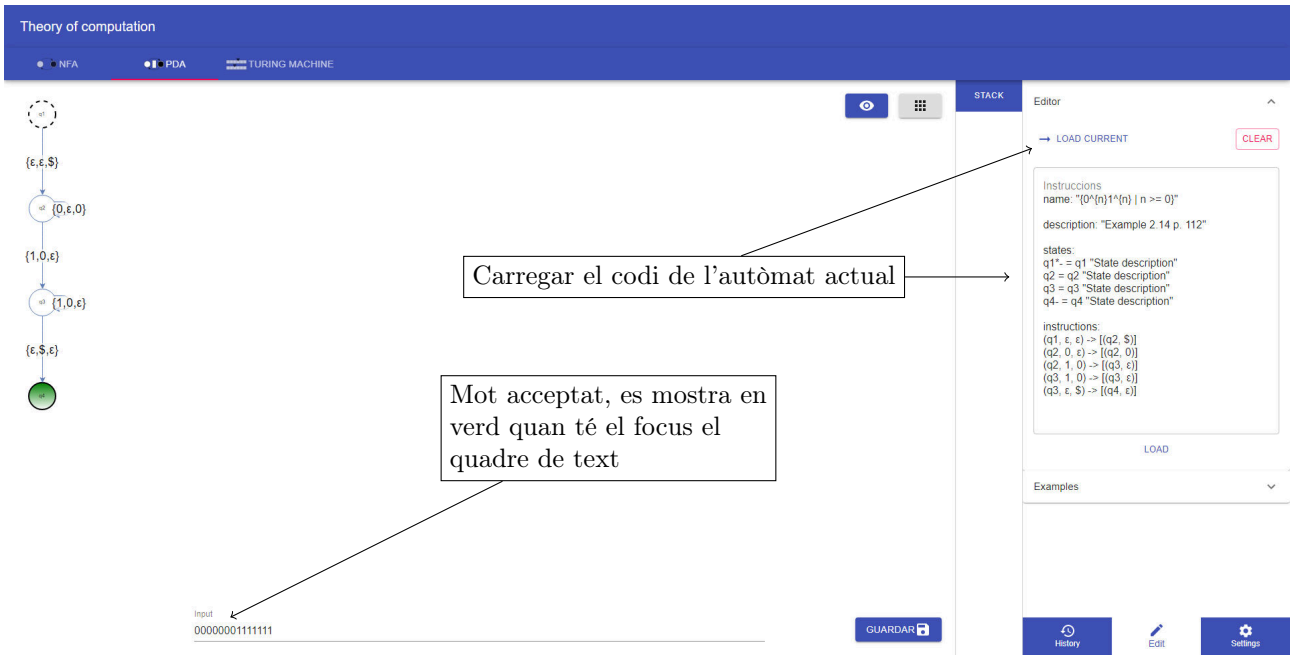


Figura 44: El mot de la figura anterior però afegint els 1's necessaris per que sigui acceptat

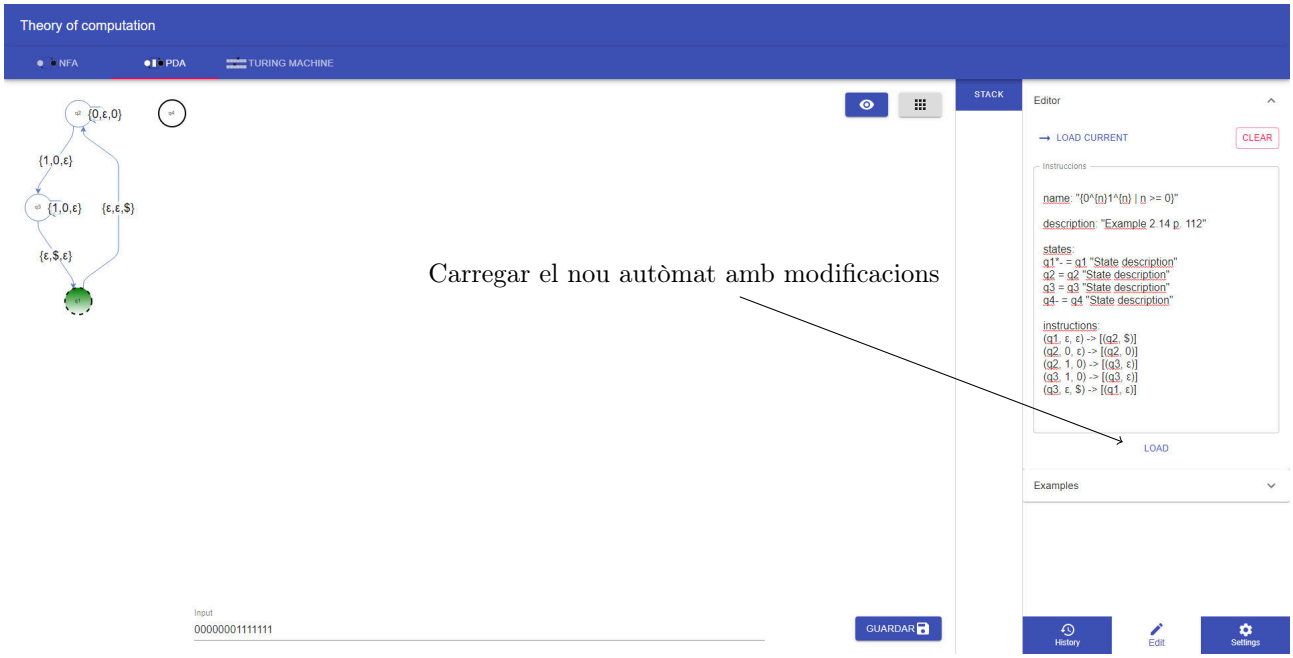


Figura 45: Exemple de modificació de l'autòmat

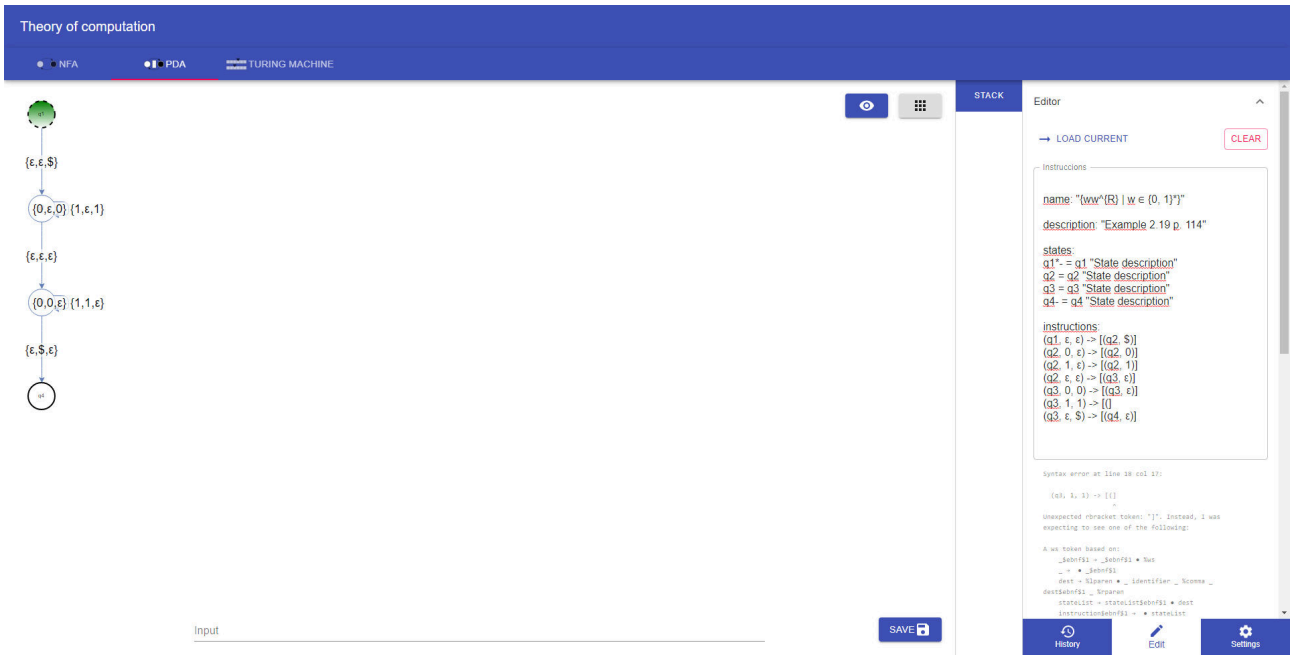


Figura 46: Exemple d'errors en la definició de l'autòmat, de manera similar a com es fa en la Fig. 26 es mostra la posició de l'error i el que s'esperava.

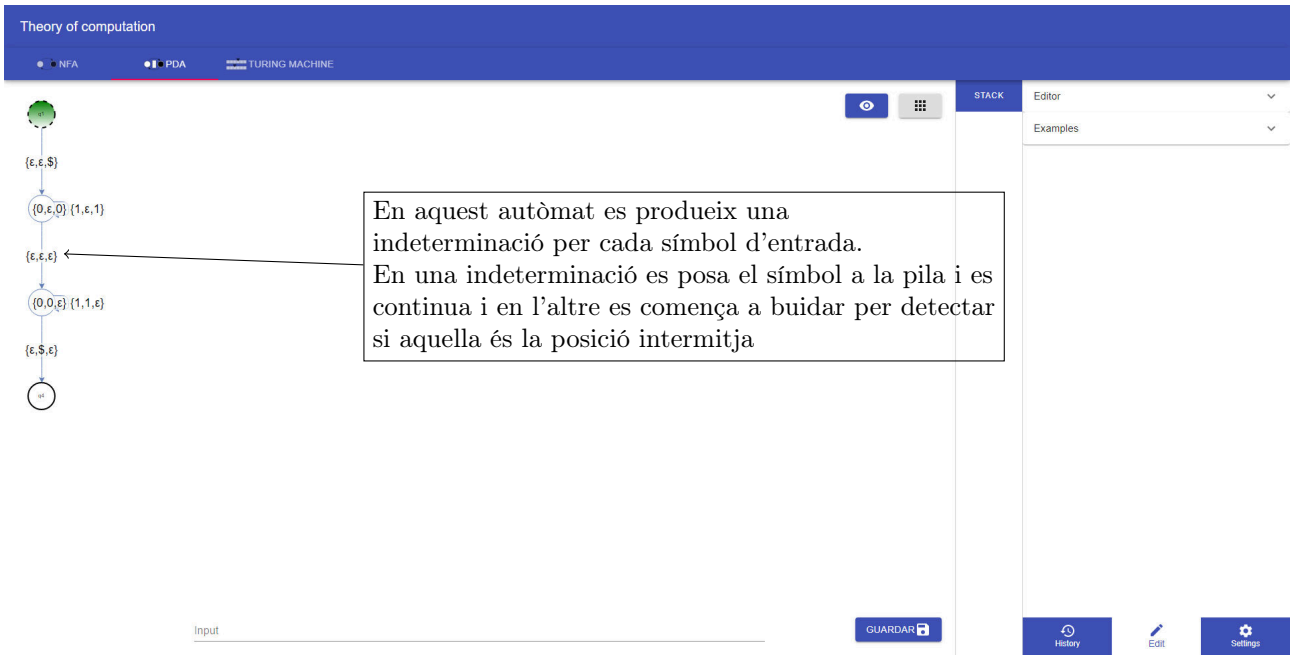


Figura 47: Autòmat d'exemple llenguatge $\{ww^R | w \in \{0,1\}^*\}$

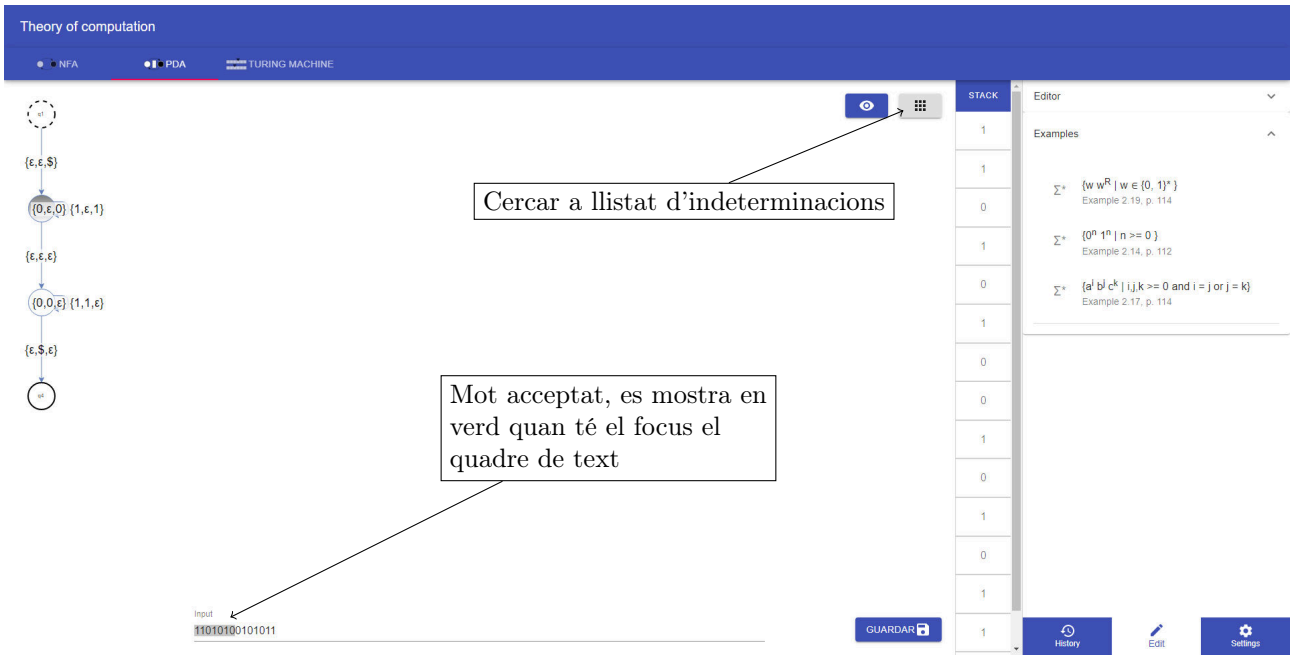


Figura 48: Mot acceptat però aquesta indeterminació el rebutja

Figura 49: Llista d'instàncies o indeterminacions de l'autòmat

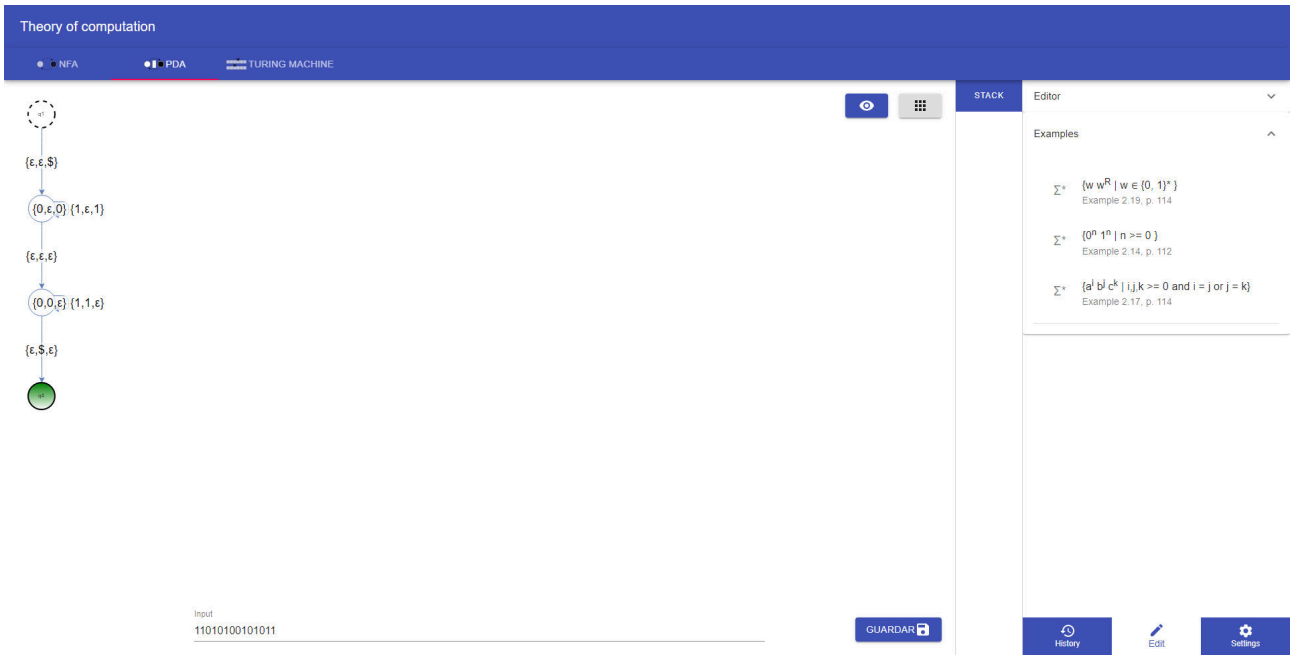


Figura 50: Mot acceptat. Indeterminació que l'accepta. S'observa que la pila és buida

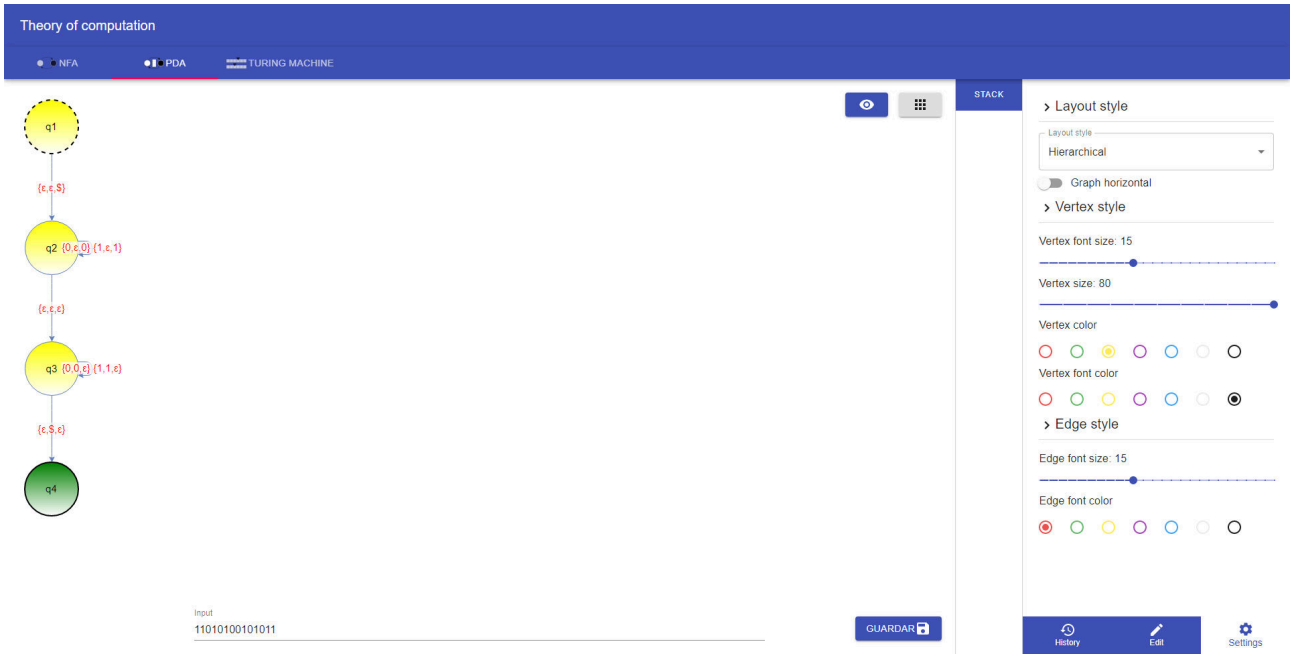


Figura 51: Exemple de canvis en la representació del diagrama d'estats

La interfície de les màquines de Turing és diferent a les anteriors, no es mostra el diagrama d'estats sinó el contingut de la cinta i l'estat en el que es troba. A continuació es detallarà el funcionament a través dels exemples proporcionats.

Tornant a la pàgina inicial, es selecciona el primer exemple, el llenguatge és $\{w\#w|w \in \{0,1\}^*\}$. Es mostra l'editor amb la definició del programa i es substitueix la màquina del simulador per la corresponent.

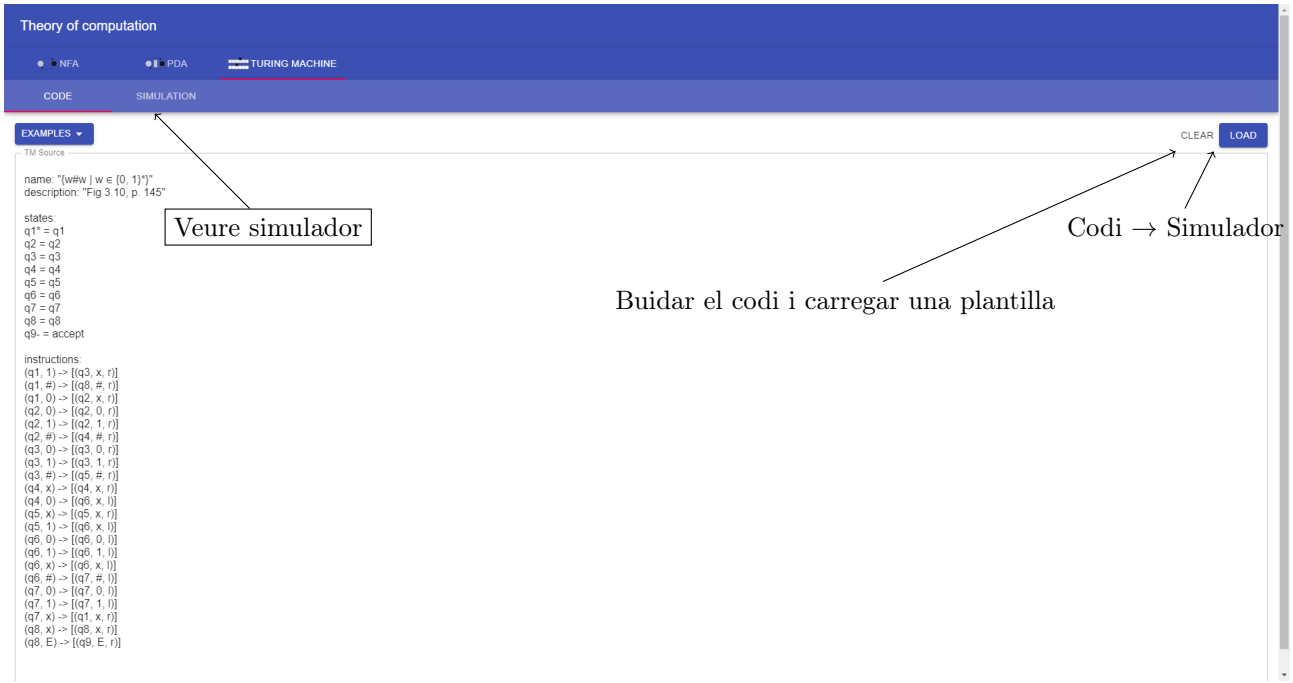


Figura 52: Visualització del codi de la màquina de Turing

En la Fig. 53 s'han afegit dos elements visuals dinàmics que fan la interfície més amigable. Un és el dibuix d'un tambor de la màquina *Bombe* dissenyada inicialment per Alan Turing que esta en moviment donant voltes. L'altre és que cada vegada que s'avança un pas es desplaça la cinta de manera horitzontal i el capçal de manera vertical.

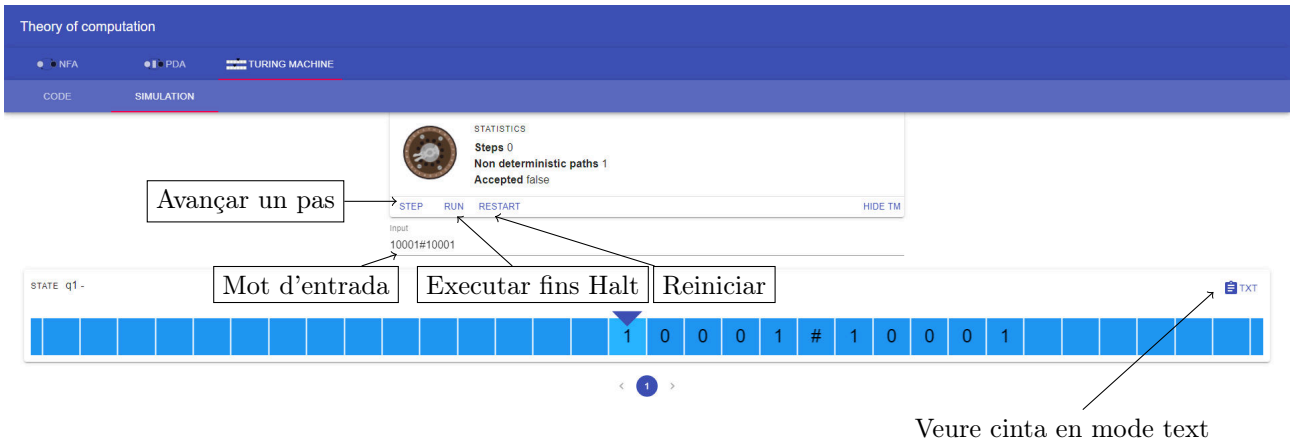


Figura 53: Simulador de la màquina de Turing

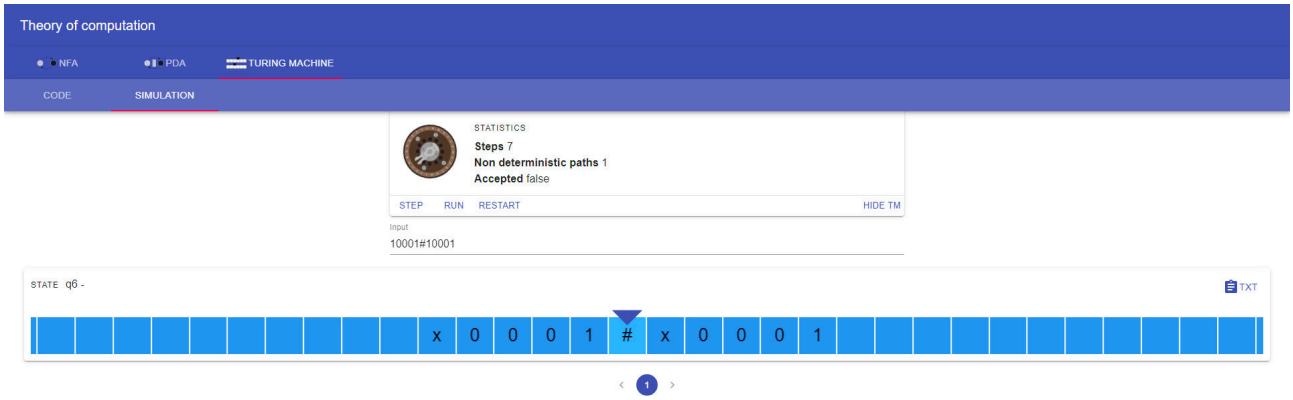


Figura 54: Execució pas per pas, en aquest punt la màquina ha verificat que els dos primers caràcters de cada mot son iguals i els ha substituït per una 'x'

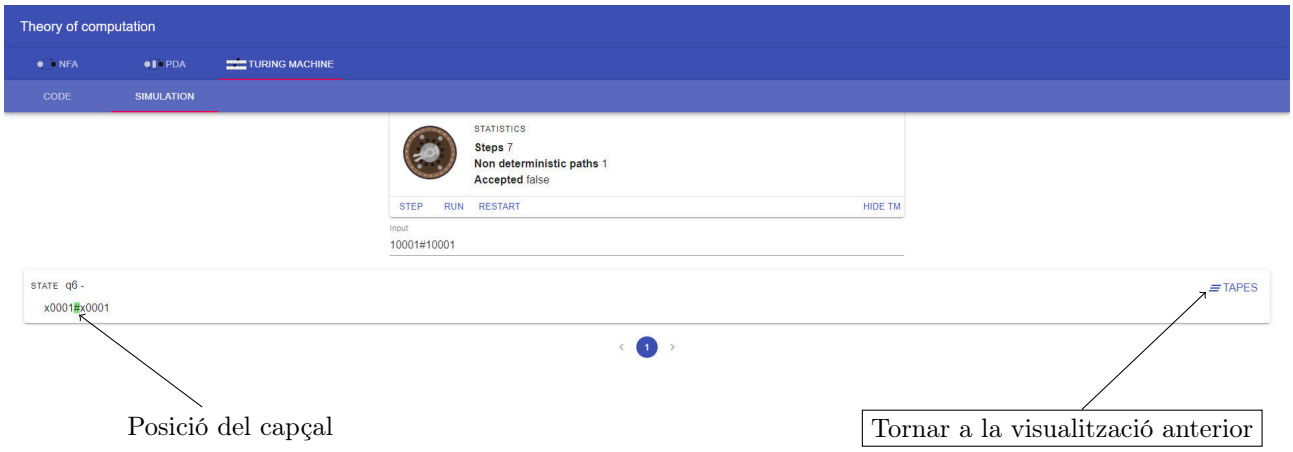


Figura 55: Visualització de la cinta en mode text, s'observa que la posició del capçal es distingeix en color verd

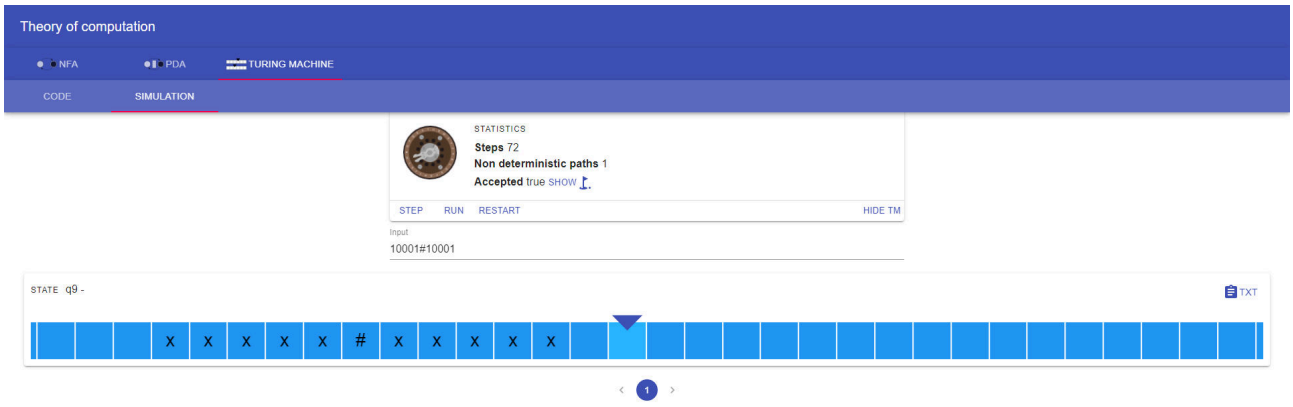


Figura 56: Acceptació del mot. S'observa que en total han estat necessaris 72 passos.

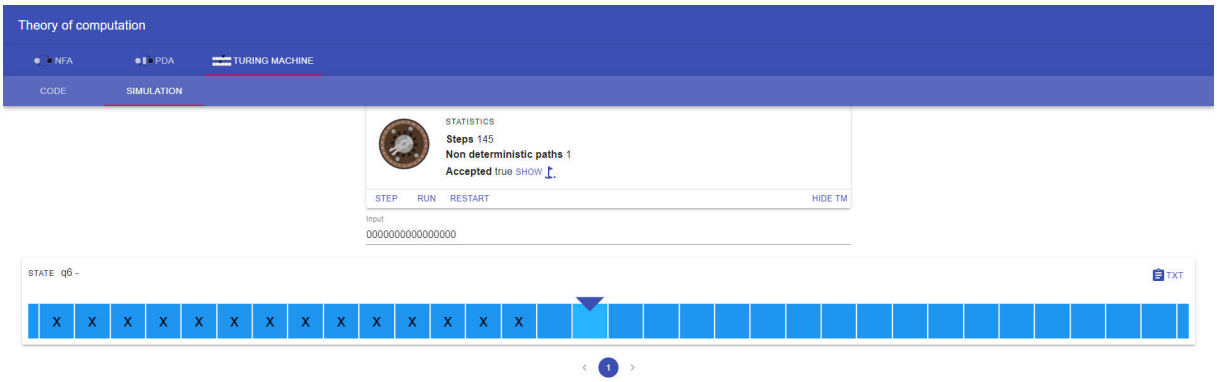


Figura 57: Exemple de TM que reconeix el llenguatge 0^{2^n} acceptant l'entrada 0^{2^4}

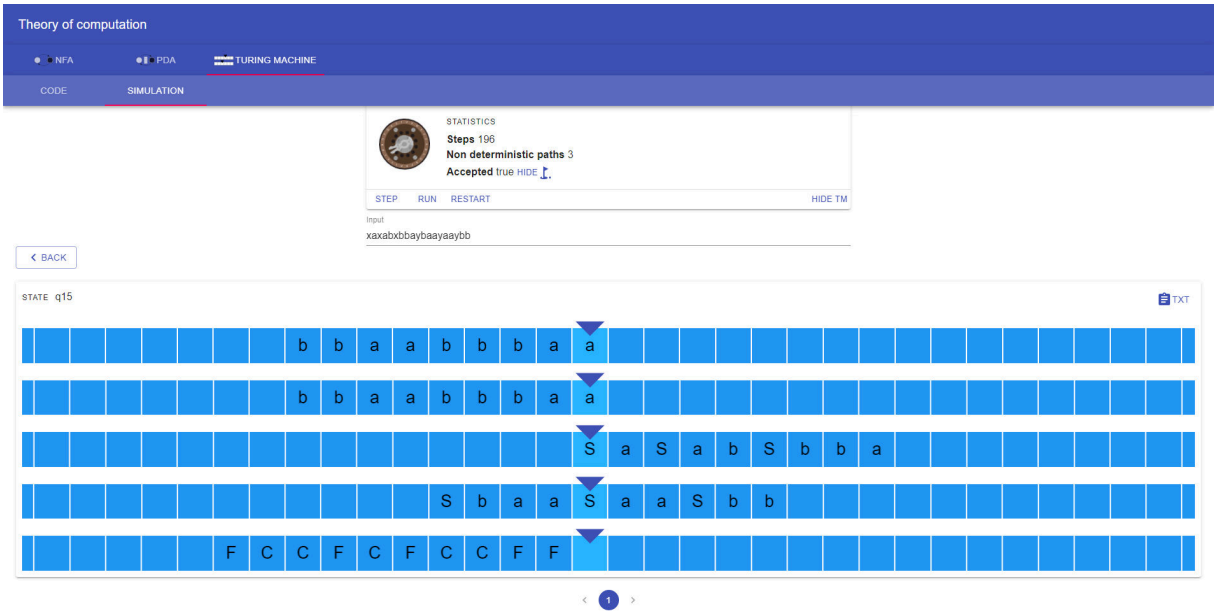


Figura 58: Exemple PCP, NDTM multicinta. Accepta els mots que tenen solució a la instància PCP que representen i es penja o rebutja les que no. S'observa com accepta el mot 'xaxabxbbaybaayaaybb' que representa la combinació {a, ab, bba} i {baa, aa, bb} i mostra la solució a l'última cinta en el nombre de C+1 que hi ha entre les F {3, 2, 3, 1}.

Tan en el cas dels autòmats finits com els de pila és possible arrastrar directament el fitxer amb el codi de l'autòmat a l'aplicació. Per accedir a la zona de càrrega d'autòmats és necessari que no n'hi hagi un de carregat, o si ja n'hi ha un, ocultar-lo fent servir la opció que s'indica en la següent figura.

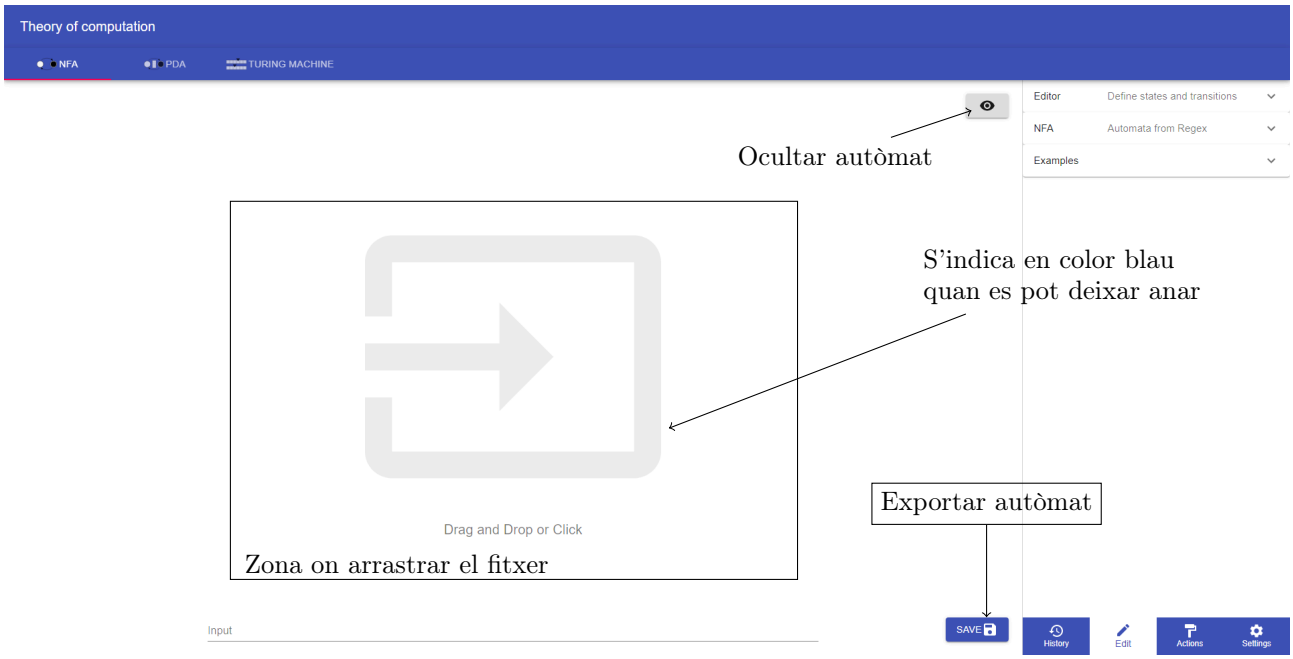


Figura 59: Importació i exportació d'autòmats finits o de pila

A continuació es mostra que l'aplicació suporta altres resolucions i s'adapta a les dimensions de la pantalla.

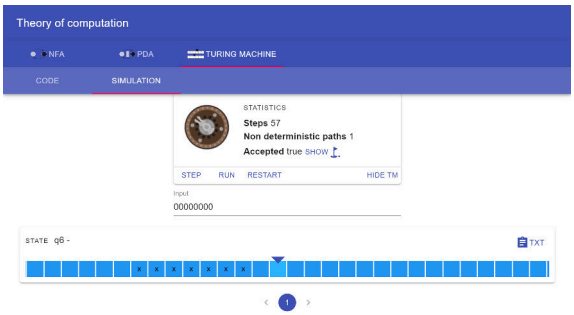


Figura 60: TM

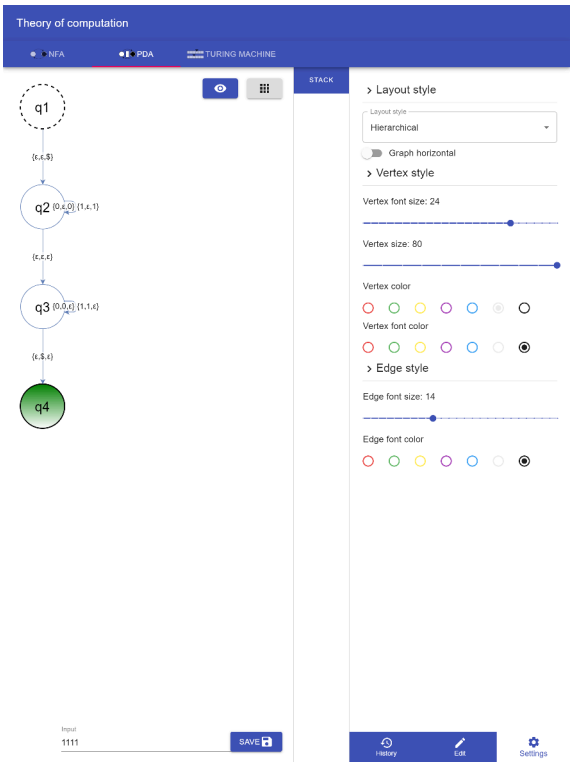


Figura 61: PDA

Figura 62: Resolució Ipad Pro 1732x2048px

11 Conclusions

L'aplicació que s'ha desenvolupat compleix l'objectiu inicial principal, mostra el funcionament i el potencial de cada un dels models de còmput proposats. S'ha fet èmfasi en la simplicitat d'ús afegint exemples de tots els models i totes les seves variants i un disseny de la interfície senzill i comú que no necessiti un aprenentatge extra.

La metodologia ha tingut un paper important a l'hora de desenvolupar l'aplicació. Separar el projecte en dues parts ha agilitzat el procés de programar cada una d'elles per separat evitant incompatibilitats o errors difícils de rastrejar.

Fent una comparativa amb altres aplicacions similars que s'utilitzen en la docència com el JFlap, s'observa que els continguts que comprèn l'aplicació desenvolupada en quant a contingut teòric és només una part del que permet JFlap. En JFlap s'inclouen continguts sobre gramàtiques, parsers, sistemes L, etc i aquesta aplicació es centra exclusivament en autòmats finits, autòmats de pila i màquines de Turing. JFlap és una eina local que s'executa en un entorn Java i l'aplicació desenvolu-

pada s'executa al navegador sense necessitat d'instal·lar Software. En JFlap es poden dissenyar de manera interactiva els autòmats afegint transicions i estats de manera visual, en el cas de l'aplicació desenvolupada s'han d'introduir a través de la definició dels estats i transicions en mode text.

Incloure en la mateixa aplicació els tres models d'autòmats i les accions que es poden aplicar sobre ells ofereix a l'alumne una visió cohesionada sobre el seu funcionament i ús.

12 Treball futur

En quant a rendiment caldria revisar el problema detallat en l'apartat anterior 9.1 que actualment força a representar l'autòmat sencer quan es produeix una modificació.

A nivell d'usabilitat es podria millorar la traçabilitat dels errors i dissenyar un sistema que en cas d'aturada imprevista del programa permetés una recuperació dels autòmats anteriors, una possible implementació d'això seria utilitzant el LocalStorage del navegador.

A nivell de desenvolupament seria molt interessant poder testejar la interfície gràfica de manera automàtica per detectar problemes d'implementació i regressions. Un dels projectes que es podria utilitzar es Enzyme [38], una eina que permet testejar els components de React de manera individual i utilitzar una DOM virtual que permet simular els clics i els events tal i com si es tractes d'un navegador habitual.

En quant a temari o conceptes un dels fronts més propers per continuar el treball són les gramàtiques lliures de context, de la mateixa manera que es pot definir un llenguatge regular amb expressions regulars, es poden definir els llenguatges lliures de context amb gramàtiques lliures de context, per tant permetent fer les conversions entre CFG $\rightarrow$ NPDA i NPDA $\rightarrow$ CFG.

Unes millores teòriques interessants podrien ser les conversions de màquines de Turing multicinta a màquines de Turing d'una sola cinta, o de la versió no determinista a la determinista.

Referències

[1] M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[2] K. Beck, *Test-driven Development: By Example*. Addison Wesley, 2000.

[3] M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing, 1997. Text original destinat a l'estudiant: You are about to embark on the study of a fascinating and important subject: The theory of computation. It comprises the fundamental mathematical properties of computer hardware, software, and certain applications thereof. In studying this subject we seek to determine what can and cannot be computed, how quickly, with how much memory, and on which type of computational model. The subject has obvious connections with engineering practice, and as in many sciences, it also has purely philosophical aspects.

[4] M. Sipser, *Introduction to the Theory of Computation*, p. 36. PWS Publishing, 1997.

[5] M. Sipser, *Introduction to the Theory of Computation*, p. 55. PWS Publishing, 1997.

[6] M. Sipser, *Introduction to the Theory of Computation*, p. 64. PWS Publishing, 1997.

[7] S. H. Rodger and T. W. Finley, *JFLAP: An Interactive Formal Languages and Automata Package*, pp. Preface xi–x. Jones & Bartlett Publishers, Sudbury, MA, 2006.

[8] "Nodejs license." `https://github.com/nodejs/node/blob/master/LICENSE`.

[9] "Npm license." `https://github.com/npm/cli/blob/latest/LICENSE`.

[10] "Babel license." `https://github.com/babel/babel/blob/master/LICENSE`.

[11] "Immutable license." `https://github.com/immutable-js/immutable-js/blob/master/LICENSE`.

[12] "Mocha license." `https://github.com/mochajs/mocha/blob/master/LICENSE`.

[13] "Nearley license." `https://github.com/kach/nearley/blob/master/LICENSE.txt`.

[14] "Moo license." `https://github.com/no-context/moo/blob/master/LICENSE`.

[15] "Istanbul license." `https://github.com/istanbuljs/nyc/blob/master/LICENSE.txt`.

[16] "Stack overflow developer survey 2019." `https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-web-frameworks`.

[17] "React license." `https://github.com/facebook/react/blob/master/LICENSE`.

[18] "React dropzone license." `https://github.com/react-dropzone/react-dropzone/blob/master/LICENSE`.

[19] "Redux license." `https://github.com/reduxjs/redux/blob/master/LICENSE.md`.

[20] "React redux license." `https://github.com/reduxjs/react-redux/blob/master/LICENSE.md`.

[21] "React router license." `https://github.com/ReactTraining/react-router/blob/master/LICENSE`.

[22] "Material ui license." https://github.com/mui-org/material-ui/blob/master/LICENSE.

[23] "Mxgraph license." https://github.com/jgraph/mxgraph/blob/master/LICENSE.

[24] "Webpack license." https://github.com/webpack/webpack/blob/master/LICENSE.

[25] "Webpack cli license." https://github.com/webpack/webpack-cli/blob/next/LICENSE.

[26] "Webpack dev server license." https://github.com/webpack/webpack-dev-server/blob/master/LICENSE.

[27] "Babel loader license." https://github.com/babel/babel-loader/blob/master/LICENSE.

[28] "Style loader license." https://github.com/webpack-contrib/style-loader/blob/master/LICENSE.

[29] "Css loader license." https://github.com/webpack-contrib/css-loader/blob/master/LICENSE.

[30] "Sass loader license." https://github.com/webpack-contrib/sass-loader/blob/master/LICENSE.

[31] "Script loader license." https://github.com/webpack-contrib/script-loader/blob/master/LICENSE.

[32] "React svg loader license." https://github.com/boopathi/react-svg-loader/blob/master/LICENSE.

[33] "Sass loader license." https://github.com/sass/node-sass/blob/master/LICENSE.

[34] "React hot loader license." https://github.com/gaearon/react-hot-loader/blob/master/LICENSE.

[35] "Cse 135: Introduction to theory of computation - optimal dfa." https://faculty.ucmerced.edu/sim3/teaching/spring15/lecture_note/lecture5-2.pdf.

[36] M. Sipser, *Introduction to the Theory of Computation*, pp. 71–72. PWS Publishing, 1997.

[37] M. Sipser, *Introduction to the Theory of Computation*, pp. 70–71. PWS Publishing, 1997.

[38] "Enzyme." https://enzymejs.github.io/enzyme/.