MTSolution
Una Herramienta Visual e Interactiva para Cursos Introductorios de Lenguajes Formales y Autómatas

Mariano Andrés Martínez

24 de agosto de 2007

Trabajo Final presentado como parte de los requisitos para la obtención del título de Ingeniero en Sistemas de la Facultad de Ciencias Exactas de la Universidad Nacional del Centro de la Provincia de Buenos Aires.

Agradecimientos

En principio deseo agradecerles a Liliana, Rosana y Virginia, por darme la oportunidad de realizar este trabajo y también por la confianza que depositaron en mi para desarrollar esta aplicación. Les quiero agradecer la ayuda que me brindaron, las correcciones y todos sus consejos.

A mis jefes y compañeros de trabajo por su consideración y por alentarme en el progreso y finalización de mi carrera.

A mis padres y hermanos por todo el apoyo y por estar siempre presentes a lo largo de toda la carrera.

A Nadia, por acompañarme todas estas tardes y trasnoches, y por cuidar a nuestra pequeña Agustina mientras escribo estas páginas.

"La mayoría de las ideas fundamentales de la ciencia
son esencialmente sencillas y, por regla general
pueden ser expresadas en un lenguaje comprensible para todos."
Albert Einstein.

Índice general

Capítulo 1

Introducción

El alcance de la teoría de la computación ha cambiado en los últimos años y está continuamente influenciado por el surgimiento de nuevas herramientas, técnicas y paradigmas. Por esta razón, es necesaria una reevaluación continua de los tópicos cubiertos por esta ciencia, así como también de los enfoques pedagógicos que se utilizan. En particular, sería ventajoso relacionar los aspectos teóricos con los prácticos durante toda la carrera de Ingeniería de Sistemas, comenzando con los primeros cursos. En nuestra Facultad, el plan de estudios de la carrera Ingeniería de Sistemas incluye la materia "Ciencias de la Computación I" como un curso introductorio a la teoría de lenguajes y autómatas. El principal propósito de este curso es presentar una introducción al estudio de los procesos computacionales y explorar su alcance en el contexto de una jerarquía de autómatas, con un enfoque adecuado para alumnos de primer año. La idea es motivar el estudio de la naturaleza y límites computacionales por medio de aplicaciones concretas a lo largo de toda la carrera. La metodología seguida para enseñar cada uno de los temas incluidos en los contenidos del curso, comienza introduciendo las ideas fundamentales informalmente por medio de muchos ejemplos, para motivar su uso con fines prácticos. Luego, se dan las definiciones formales correspondientes [16].
Tradicionalmente el dictado de cursos, como el de "Ciencias de la Computación I", se ha desarrollado sin el uso de computadoras. Los alumnos trabajan sobre ejercicios con lápiz y papel sin recibir *feedback* inmediato. Como resultado, para algunos alumnos el curso puede resultar menos incentivador que otros cursos donde interactúan directamente con la computadora. Además, de la misma manera en que cuando desarrollan un programa en algún lenguaje de programación, por lo general no es correcto la primera vez que se compila y ejecuta, la experiencia muestra también que los autómatas diseñados a mano pueden ser frecuentemente incorrectos. En base a las consideraciones previas, sería ventajoso integrar la práctica manual con *feedback* inmediato mediante el uso de herramientas interactivas y visuales. Si bien, herramientas existentes especializadas en estos temas, como JFLAP [17], DEM [11], Turing's World [5] entre otras proveen la posibilidad de una representación visual alternativa para crear

y simular autómatas, ninguna de ellas concordaba con el enfoque y la metodología dados en el curso "Ciencias de la Computación I" de nuestra carrera. Por esta razón, se desarrolló *Minerva* [13], con la que se experimentó en los años 2001 y 2002 y permitió probar que este tipo de herramientas ayudan al alumno en el proceso de aprendizaje. Además considerando que se trata de alumnos que recién se integran al ambiente universitario, se observó que la aplicación debería ser bastante intuitiva en su utilización. Esta experiencia mostró que la interfaz gráfica es un requerimiento importante a satisfacer, debiendo ser amigable, intuitiva, predecible y ágil. Esto motiva el desarrollo de una nueva herramienta, visual e interactiva que atraiga el interés de los alumnos y ofrezca *feedback* inmediato, lo que se traduce usualmente en una mejor comprensión y control del material en los cursos.
Se desarrolló entonces *MTSolution*, una aplicación cliente-servidor que resuelve la ejecución de los distintos autómatas reduciéndolos a una máquina de Turing multicinta, no determinística. Trabajando en un ambiente didáctico visual, más intuitivo y amigable, los alumnos podrán, por ejemplo, diseñar y ejecutar los diferentes autómatas. De esta manera, podrán experimentar con diversos formalismos como gramáticas, expresiones regulares, y autómatas detectando y corrigiendo errores. Además, se propone incluir funcionalidad no cubierta por Minerva para componer Máquinas de Turing. Esto permitirá introducir conceptos de modularización y reutilización desde etapas tempranas de la formación de los alumnos. Estos aportes son originales en el área ya que no existen herramientas que tengan el mismo enfoque que el dado en este trabajo. Por otro lado, la simplicidad en el diseño y *testing* de autómatas podría alentar a alumnos avanzados a adentrarse en el desarrollo de ejemplos más complejos, pudiendo así utilizarse la nueva herramienta en otras materias.

MTSolution soporta la creación de distintos tipos de autómatas (finitos, de pila y máquinas de Turing), gramáticas y la comprobación de resultados, en particular permite:

- Probar si una cadena pertenece al lenguaje reconocido por el autómata diseñado.
- Decidir si una cadena puede ser generada por una gramática.
- Generar cadenas que pertenecen al lenguaje representado por una gramática dada.
- Realizar comprobaciones de resultados al aplicar los siguientes algoritmos de conversión: autómata finito no determinístico con transiciones vacías ($AF_{ND} - \epsilon$) a autómata finito no determinístico (AF_{ND}), AF_{ND} a autómata finito determinístico (AF_D), AF_D a AF_D con cantidad mínima de estados, AF_D a expresión regular, expresión regular a AF_{ND}, AF_{ND} a gramática regular y gramática regular a AF_{ND}.
- Obtener el autómata finito equivalente a partir de una expresión regular.

MTSolution fue desarrollada en el lenguaje de programación orientado a objetos Microsoft Visual C++ .NET debido a la eficiencia que provee en entornos

visuales sobre sistemas operativos Windows. Se ha implementado sobre esta plataforma, sacrificando la portabilidad en beneficio de la facilidad de instalación, utilización y actualización, además de ser el sistema operativo que generalmente más conocen los alumnos al ingresar a la universidad.

1.1. Trabajos Relacionados

La construcción de esta herramienta educativa se basó, no sólo en analizar los contenidos del curso "Ciencias de la Computación I" y la metodología seguida para enseñar cada uno de los temas incluidos en los contenidos del curso, sino también en el estudio de herramientas similares como

- JFlap [17]
- Turing's World [5]
- Constructive Algorithm Visualization Environment (CAVE) [27]
- Deus Ex Machina [28]
- Minerva[12]

Estas herramientas existen actualmente y cubren algunos conceptos de la teoría de la computación. En las siguientes secciones se describen las ventajas de cada una de ellas.

1.1.1. JFlap

JFlap es una herramienta para diseñar y ejecutar versiones no determinísticas de autómatas finitos, de pila y máquinas de Turing. Una de sus principales ventajas es que el usuario puede visualizar el comportamiento del autómata durante el reconocimiento de una cadena de entrada, lo cual le permite verificar si el autómata se comporta como él desea. JFlap fue diseñada para ser utilizada en cursos más avanzados de Ciencias de la Computación, por lo cual el enfoque dado es más abarcativo que el que se pretende dar en el curso introductorio de nuestra Facultad. Por ejemplo, JFlap permite trabajar con algoritmos de conversión sobre gramáticas y expresiones regulares; la mayoría de los algoritmos sobre gramáticas no son estudiados durante el curso "Ciencias de la Computación I". Por otra parte, desde el punto de vista educativo, JFlap no le provee al usuario ayuda para la corrección de conceptos erróneos; por ejemplo, no hace diferencia entre autómatas determinísticos y no determinísticos, con lo cual, el usuario puede desconocer el concepto de determinismo y la herramienta no le provee ayuda alguna con respecto a este concepto. Además, JFlap no utiliza la jerarquía de Chomsky estudiada en el curso para definir el formato de las reglas de producción de las gramáticas, lo cual podría confundir al alumno tanto en la creación de los distintos tipos de gramáticas como en los temas que las involucran (conversión de gramáticas a expresiones regulares o a autómatas finitos).

1.1.2. Turing's World

Permite al usuario crear una máquina de Turing y seguir el comportamiento de la misma debido a que muestra una vista de las cintas de la máquina creada, en donde se pueden ver cuáles son los símbolos de cada una de la cintas, y cuáles son los movimientos que se van efectuando sobre cada una de ellas, además brinda la posibilidad de diseñar máquinas de Turing modularizadas, pero definiendo las máquinas en forma de grafos.

1.1.3. Constructive Algorithm Visualization Environment (CAVE)

Esta aplicación permite realizar operaciones sobre AF_D's, como ser:

- Uniones
- Intersecciones
- Clausura de Kleene
- Concatenaciones
- Complemento

Tiene muchas limitaciones. Entre ellas, sólo trabaja con autómatas finitos determinísticos y en una forma muy limitada. Las transiciones se definen sólo para un máximo de ocho estados. Sólo admite dos símbolos para el alfabeto. Finalmente, no posee interfaz gráfica, la información se ingresa y se muestra en modo de texto.

1.1.4. Deus Ex Machina

Al igual que JFLAP permite diseñar los autómatas en una forma interactiva. Es interesante que permita comentarios para los nodos y arcos. También brinda la posibilidad de ejecutar paso a paso. La gran ventaja sobre JFLAP es la sencillez en la instalación. JFLAP requiere un intérprete de JAVA y esto puede requerir un usuario con un poco de experiencia. Sin embargo la instalación de Deus Ex Machina, sólo requiere un doble *click*.

1.1.5. Minerva

Cabe destacar la herramienta Minerva[12], desarrollada en esta Facultad, la cual ha sido un referente en el diseño de *MTSolution*. Al estar programada en JAVA, es de gran portabilidad, es decir es soportada por cualquier sistema operativo compatible con este lenguaje. Incluye funciones para realizar prácticas sobre el Lema Pumping para lenguajes regulares, permitiendo al alumno interpretar el enunciado del lema. También permite comprobar que un lenguaje dado no es regular, lo cual ayuda a comprender la utilización del contrarrecíproco del lema.

1.1.6. Resultado del análisis de las herramientas

De las herramientas citadas previamente, a pesar de que muchas de ellas consideran parte de los temas del curso "Ciencias de la Computación I", ninguna a excepción de Minerva, se ajusta en forma total al enfoque con que se tratan los contenidos del curso. El análisis llevado a cabo sobre cada una de las herramientas citadas anteriormente fue de gran utilidad para la construcción de *MTSolution*, dado que se pudieron incorporar las ventajas que aportan cada una de ellas; y, por otra parte, se trató de cubrir aquellos aspectos que no son tenidos en cuenta por las distintas herramientas. A continuación un detalle por herramienta.

- JFLAP y Minerva: se ha intentado incorporar y mejorar la interfaz gráfica y las funcionalidades de ejecución paso a paso.
- Deus Ex Machina: La sencillez en la instalación y ejecución.
- Turing's World: La posibilidad de diseñar máquinas de Turing modularizadas pero en forma de tabla. También el seguimiento de cintas en la ejecución.
- Constructive algorithm Visualization Environment: No aportó nada que pudiera incluirse.

1.2. Organización del Informe

El informe se encuentra organizado en diferentes capítulos por medio de los cuales se describe el trabajo realizado durante el desarrollo de *MTSolution*.

- En el capítulo 2 se citan algunos conceptos teóricos incluidos en el curso "Ciencias de la Computación I" que fueron utilizados para desarrollar la aplicación.
- El capítulo 3 muestra el análisis y diseño de la aplicación. En el mismo se detallan los requerimientos, la metodología de desarrollo, y los distintos análisis y diseños de los programas que componen a MTSolution. Este capítulo se encuentra ampliado en los apéndices A("Algoritmos para obtener Máquinas de Turing"), B("Traducción de transiciones"), C("Esquema de los archivos de intercambio"), D("Framework Unidraw") y E("Adaptación de algoritmos a eventos").
- El capítulo 4 describe la interfaz a usuario de toda la aplicación y puede ser leído tal como un manual de usuario. Se explican las distintas funcionalidades y capacidades de esta herramienta.
- En el capítulo 5 se desarrollan diversos ejemplos del uso de la herramienta con el objetivo de mostrar algunas de las ventajas que MTSolution aporta al usuario.

- Finalmente, en el capítulo 6 se citan algunas conclusiones a las que se han podido arribar luego de desarrollar la herramienta MTSolution. Además, se mencionan posibles extensiones que se pueden llevar a cabo en la aplicación.

Capítulo 2

Conceptos Teóricos

En este capítulo se describirán los temas incluidos en el contenido del curso Ciencias de la Computación I, que serán tenidos en cuenta en capítulos posteriores durante el diseño de la herramienta *MTSolution*.

2.1. Autómatas

Un autómata es un modelo matemático de una máquina que acepta cadenas de un lenguaje definido sobre un alfabeto. Consiste en un conjunto finito de estados y un conjunto de transiciones entre los mismos que dependen de los símbolos de la cadena de entrada.

2.1.1. Autómatas Finitos

Un autómata finito (AF) permite reconocer un tipo de lenguajes en particular que se denominan lenguajes regulares.

Definición 1:

Un autómata finito formalmente se define como una 5-upla
$M =< E, A, \delta, e_0, F >$ donde:

- E : Conjunto finito de estados
- A : Alfabeto de entrada
- δ : Función de transición
- e_0 : Estado inicial; $e_0 \in E$
- F : Conjunto de estados finales; $F \subseteq E$

Dentro de esta clase de autómatas se diferencian dos tipos, los determinísticos (AF_D) y los no determinísticos (AF_{ND}). La diferencia entre ellos consiste en la forma en que definen su función de transición.

- Para un AF_D la función de transición se define como $\delta : E \times A \rightarrow E$; dado un estado del autómata y un símbolo del alfabeto, existe como máximo un estado que puede ser alcanzado.
- La definición formal de la función de transición para un AF_{ND} es $\delta : E \times A \rightarrow P(E)$; dado un estado del autómata y un símbolo del alfabeto, existe un conjunto de estados que pueden ser alcanzados ($P(E)$ es el conjunto potencia de E, es decir el conjunto de todos los subconjuntos de E).

A continuación se cita un ejemplo en donde se muestra la diferencia que existe entre AF_D y AF_{ND}.

Ejemplo 1: Supongamos el lenguaje L que se define de la siguiente forma: $L = \{\ x/x \in \{0,1\}^*$ y x contiene la subcadena 00 ó x contiene la subcadena $11\}$.

Autómata finito determinístico que reconoce cadenas del lenguaje L del ejemplo 1: $M_D =< \{e_0, e_1, e_2, e_3\}, \{0,1\}, \delta_D, e_0, \{e_3\} >$ donde δ_D está definida por el diagrama de transición de estados de la Figura 2.1.

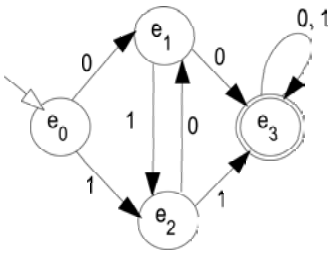


Figura 2.1: AF_D que reconoce cadenas del lenguaje L

Existe determinismo porque a partir de cada estado y con cada símbolo del alfabeto se puede alcanzar a lo sumo otro estado. Por ejemplo, desde el estado e_0 con el símbolo 0 se puede alcanzar el estado e_1 y con el símbolo 1 el estado e_2. Es decir,

- $\delta(e_0, 0) = e_1$
- $\delta(e_0, 1) = e_2$

Autómata finito no determinístico que reconoce cadenas del lenguaje L: $M_{ND} =< \{e_0, e_1, e_2, e_3, e_4\}, \{0,1\}, \delta_{ND}, e_0, \{e_2, e_4\} >$ donde δ_{ND} está definida por el diagrama de transición de estados de la Figura 2.2.

El no determinismo está dado porque desde el estado e_0 con el símbolo 0 se puede ir al estado e_0 ó e_3; lo mismo sucede con el símbolo 1, ya que se puede alcanzar el estado e_0 ó e_1. Es decir,

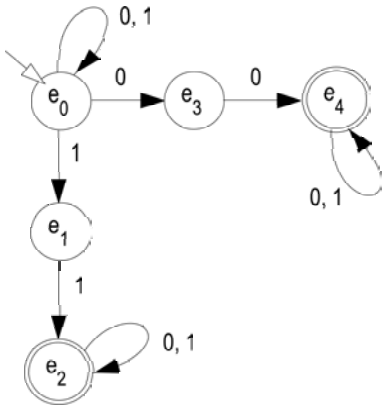


Figura 2.2: AF_{ND} que reconoce cadenas del lenguaje L

- $\delta(e_0, 0) = \{e_0, e_3\}$
- $\delta(e_0, 1) = \{e_0, e_1\}$

Expresiones Regulares

El tipo de lenguaje que es reconocido por los autómatas finitos puede ser descrito por las expresiones regulares. Las expresiones regulares se pueden construir a partir de las siguientes reglas:

- $\oslash$ es una expresión regular que describe el lenguaje vacío;
- ϵ es una expresión regular que describe el lenguaje $\{\epsilon\}$, esto es el lenguaje que contiene únicamente la cadena vacía;
- Para cada símbolo $a \in A$, donde A es un alfabeto, a es una expresión regular que describe el lenguaje $\{a\}$, esto es el lenguaje que contiene únicamente la cadena a;
- Si r y s son expresiones regulares que describen los lenguajes $L(r)$ y $L(s)$ respectivamente:
  - $r + s$ es una expresión regular que describe el lenguaje $L(r) \cup L(s)$
  - $r \cdot s$ es una expresión regular que describe el lenguaje $L(r) \cdot L(s)$
  - r^* es una expresión regular que describe el lenguaje $L(r)^*$.

El operador de clausura es el que tiene mayor precedencia, seguido por el operador de concatenación y por último el operador de unión.

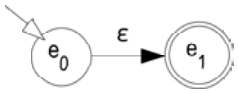
Ejemplo 2: Supongamos que tenemos el lenguaje L que se define de la misma forma que para el ejemplo de autómatas finitos $L = \{x / x\{0,1\}^*$ y x contiene la subcadena 00 ó x contiene la subcadena $11\}$. La expresión regular que describe al lenguaje L es la siguiente: ((1+01)(01)*(1+00)+00)(0+1)*

Algoritmo para Construir el AF a partir de la Expresión Regular
A partir de una expresión regular se puede obtener un autómata finito que reconoce cadenas del lenguaje descrito por la misma. El autómata generado es un autómata finito no determinístico con la particularidad de que puede poseer algunas transiciones vacías ($AF_{ND}-\epsilon$), es decir, que para pasar desde un estado a otro del autómata no se lee un símbolo en la cadena de entrada.

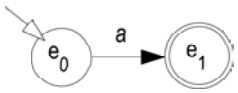
Algoritmo 1: Construcción del autómata finito con transiciones ϵ. Si r es una expresión regular con n operadores y sin variables como operandos atómicos, existe un $AF_{ND}-\epsilon$ M que acepta solamente aquellas cadenas que están en $L(r)$. M tiene un estado final, no entran arcos al estado inicial y no salen arcos del estado final. r puede ser una expresión sin operadores ($\oslash$, ϵ o un símbolo) o con operadores (+, ., *). Si r no tiene operadores, entonces:

- Para $r = \oslash$ el $AF_{ND} - \epsilon$ es

- Para $r = \epsilon$ el $AF_{ND} - \epsilon$ es



- Para $r = a, (a \in A)$ el $AF_{ND} - \epsilon$ es



Si r tiene operadores, se dan tres casos dependiendo de la forma de r:

1. $r = r_1 + r_2$
   Sean $M_1 =< E_1, A, \delta_1, e_{01}, \{e_{f1}\} >$ y $M_2 =< E_2, A, \delta_2, e_{02}, \{e_{f2}\} >$, los autómatas correspondientes a r_1 y r_2. Se construye un nuevo autómata M que une a estos dos autómatas M_1 y M_2 agregando un estado inicial e_0 y un estado final e_{f0}; $M =< E_1 \cup E_2 \cup \{e_0, e_{f0}\}, A, \delta, e_0, \{e_{f0}\} >$. El estado inicial de M tiene transiciones ϵ a los estados iniciales de M_1 y M_2; los estados finales de estos autómatas tienen transiciones ϵ al estado final del autómata M. (Figura 2.3)

2. $r = r_1.r_2$
   Sean $M_1 =< E_1, A, \delta_1, e_{01}, \{e_{f1}\} >$ y $M2 =< E_2, A, \delta_2, e_{02}, \{e_{f2}\} >$, los autómatas correspondientes a r_1 y r_2. Se construye un nuevo autómata M,

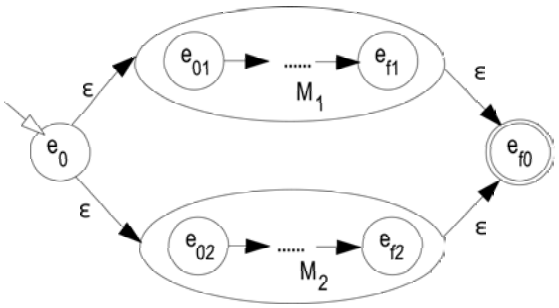


Figura 2.3: Esquema del autómata correspondiente a $r = r_1 + r_2$

$M =< E_1 \cup E_2, A, \delta, e_{01}, \{e_{f2}\} >$ que tiene como estado inicial al estado inicial de M_1 y como estado final al estado final de M_2; tiene además un arco rotulado ϵ desde el estado final de M_1 al estado inicial de M_2.(Figura 2.4)

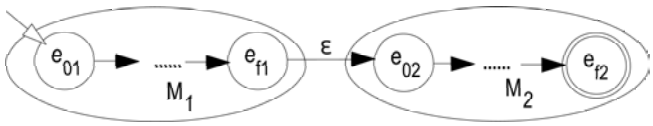


Figura 2.4: Esquema del autómata correspondiente a $r = r_1.r_2$

3. $r = r_1^*$
Sea $M_1 =< E_1, A, \delta_1, e_{01}, \{e_{f1}\} >$ el autómata correspondiente a r_1. Se construye un nuevo autómata M, $M =< E_1 \cup \{e_0, e_{f0}\}, A, \delta, e_0, \{e_{f0}\} >$; y se agregan arcos rotulados ϵ desde e_0 al estado inicial de M_1 y al estado final de M, y desde el estado final de M_1 al estado inicial de M_1 y a e_{f0}.(Figura 2.5)

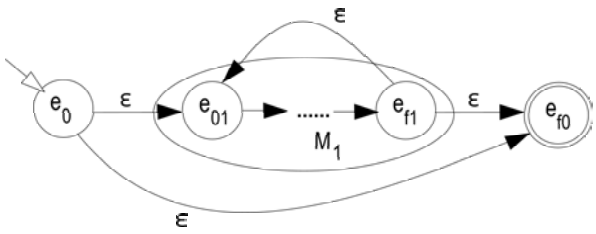


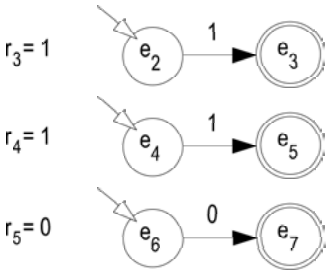
Figura 2.5: Esquema del autómata correspondiente a $r = r_1^*$

Ejemplo 3: Construcción del $AF_{ND} - \epsilon$ a partir de la siguiente expresión regular:
$r = (1 + 10)(01)^*$

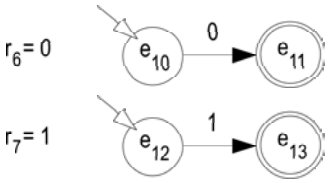
donde r se puede expresar de la siguiente forma:
$r = r_1.r_2$ donde

- $r_1 = (1 + 10)$
- $r_2 = (01)^*$

r_1 puede expresarse de la siguiente forma:
$r_1 = r_3 + r_4.r_5$ donde



El autómata de r_1 se puede observar en la Figura 2.6.
r2 se puede expresar de la siguiente forma:
$r_2 = (r_6.r_7)^*$ donde



El autómata correspondiente a r_2, por lo tanto, se puede observar en la Figura 2.7.

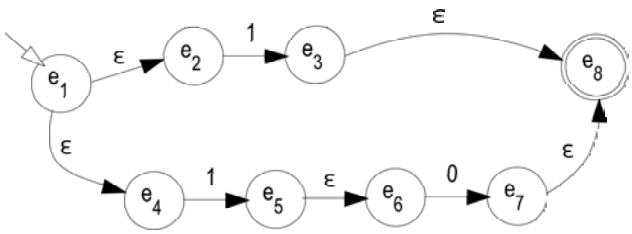


Figura 2.6: Autómata correspondiente a la expresión regular r_1

El autómata correspondiente a r es: $AF_{ND}-\epsilon = <\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}\}, \{0,1\}, \delta, e_1, \{e_{14}\} >$, con δ definida por el diagrama de transición de estados de la Figura 2.8.

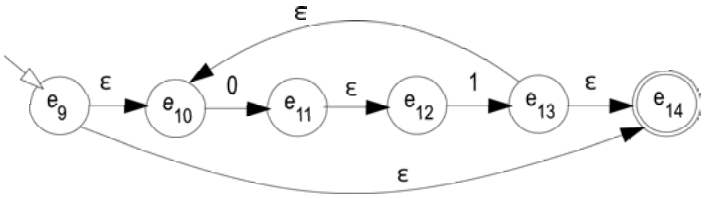


Figura 2.7: Autómata correspondiente a la expresión regular r_2

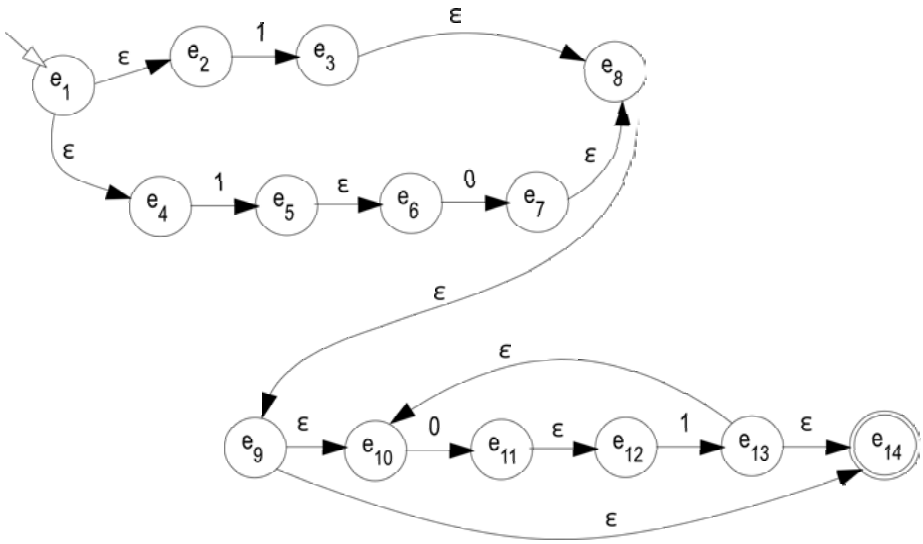


Figura 2.8: Autómata correspondiente a la expresión regular r

Algoritmo para Obtener la Expresión Regular a partir de un AF A partir de cualquier autómata finito es posible obtener una expresión regular que describe el lenguaje reconocido por el autómata. Esta conversión consiste en ir eliminando los estados del autómata uno por uno, reemplazando los rótulos sobre los arcos, que inicialmente son símbolos, por expresiones regulares más complejas. A continuación se describen cuáles son los pasos que se deben seguir durante la eliminación de un estado. Si se desea eliminar el estado u, se deben mantener los rótulos de las expresiones regulares sobre los arcos de modo tal que los rótulos de los caminos entre cualquier par de estados de los restantes no cambien. En la Figura 2.9 se pueden apreciar los estados predecesores y sucesores del estado al eliminar u.

Si no existe arco de u a u se puede agregar uno rotulado $\oslash$. Los nodos s_i, para $i = 1,2,...,n$, son nodos predecesores del nodo u, y los nodos t_j, para $j = 1,2,...,m$, son nodos sucesores del nodo u. Existe un arco de cada s_i a u, rotulado por una expresión regular S_i, y un arco de u a cada t_j rotulado por una expresión regular T_j. Si se elimina el nodo u, estos arcos y el arco rotulado

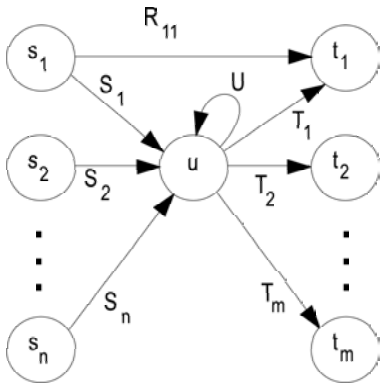


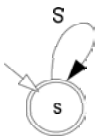
Figura 2.9: Esquema de autómata previo a la eliminación del estado u

U desaparecerán. Para preservar estas cadenas, se debe considerar cada par s_i y t_j y agregar al rótulo del arco de s_i a t_j, una expresión regular que represente lo que desaparece. En general se puede suponer que existe un arco rotulado R_{ij} de s_i a t_j para $i = 1, 2, ..., n$ y $j = 1, 2, ..., m$. Si el arco de s_i a t_j no está presente se puede agregar con rótulo $\oslash$. El conjunto de cadenas que rotulan los caminos de s_i a u, incluyendo el ciclo de u a u, y luego de u a t_j, se puede describir por la expresión regular $S_i U^* T_j$. Por lo tanto, después de eliminar u y todos los arcos que llegan y salen de u, se debe reemplazar el rótulo R_{ij} del arco de s_i a t_j por la expresión regular $R_{ij} + S_i U^* T_j$

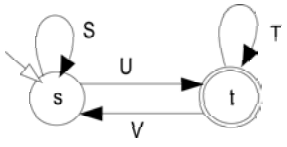
Algoritmo para construir la expresión regular a partir del autómata finito Los pasos a seguir son los siguientes:

1. Repetir para cada estado final:

   - Si el estado final es también inicial, eliminar todos los estados excepto el estado inicial. La expresión regular correspondiente es S^*.



   - Sino, eliminar los estados del autómata hasta que queden únicamente el estado inicial y el estado final en consideración.



La expresión regular que nos lleva del estado s al estado t es:

$$S^*U(T+VS^*U)^* \equiv S^*U(T^*(VS^*U)^*)^*$$

2. Realizar la unión de las expresiones regulares obtenidas para cada estado final del autómata

Algoritmos sobre Autómatas Finitos

A continuación se describen distintos algoritmos para aplicar sobre autómatas finitos.

Construcción del AF_{ND} a partir del $AF_{ND} - \epsilon$ Dado un $AF_{ND} - \epsilon$ es posible construir un AF_{ND} equivalente sin transiciones vacías que reconoce el mismo lenguaje. Para la obtención del AF_{ND} equivalente a partir del $AF_{ND} - \epsilon$ existe un algoritmo, el cual se cita a continuación.

Algoritmo para obtener el AF_{ND} a partir de un $AF_{ND} - \epsilon$ Los estados del nuevo autómata son los estados importantes del $AF_{ND} - \epsilon$ y el estado inicial del $AF_{ND} - \epsilon$. Se denominan estados importantes aquellos estados a los que llega un arco con un símbolo real como rótulo. El estado inicial es el estado inicial del $AF_{ND} - \epsilon$. La función de transición se define teniendo en cuenta que existe una transición del estado importante e_i al estado importante e_j con el símbolo x, si existe algún estado e_k tal que:

- se puede llegar desde el estado e_i al estado e_k con un camino de 0 ó más transiciones vacías (ϵ); se permite $e_i = e_k$;
- en el $AF_{ND} - \epsilon$ existe una transición del estado e_k al estado e_j rotulada con el símbolo x.

Los estados finales del nuevo autómata son los estados finales del $AF_{ND} - \epsilon$ y todos los estados e_i del $AF_{ND} - \epsilon$ para los cuales existe un camino con transiciones vacías, en el $AF_{ND} - \epsilon$, a algún estado final del $AF_{ND} - \epsilon$.

Ejemplo 4: Construcción del AF_{ND} a partir del $AF_{ND} - \epsilon$ obtenido en el ejemplo 3.

- El conjunto de estados está formado por e_1 (estado inicial) y por los estados e_3, e_5, e_7, e_{11} y e_{13} (estados importantes).
- Los estados finales son e_3, e_7 y e_{13} (existe un camino en el $AF_{ND} - \epsilon$ con transiciones ϵ a un estado final del $AF_{ND} - \epsilon$).
- El estado inicial es e_1.

- Transiciones para el estado e_1: desde el estado e1 se pueden alcanzar con transiciones ϵ los estados e_1, e_2 y e_4. En el $AF_{ND} - \epsilon$ existe una transición de e_2 a e_3 con 1 y una transición de e_4 a e_5 con 1. Entonces en el nuevo autómata hay una transición de e_1 a e_3 con 1 y una transición de e_1 a e_5 con 1.
- Transiciones para el estado e_3: desde el estado e_3 se pueden alcanzar con transiciones ϵ los estados e_3, e_8, e_9, e_{10} y e_{14}. En el $AF_{ND} - \epsilon$ existe una transición de e_{10} a e_{11} con 0. Entonces en el nuevo autómata hay una transición de e_3 a e_{11} con 0.
- Transiciones para el estado e_5: desde el estado e_5 se pueden alcanzar con transiciones ϵ los estados e_5 y e_6. En el $AF_{ND} - \epsilon$ existe una transición de e_6 a e_7 con 0. Entonces en el nuevo autómata hay una transición de e_5 a e_7 con 0.
- Transiciones para el estado e_7: desde el estado e_7 se pueden alcanzar con transiciones ϵ los estados e_7, e_8, e_9, e_{10} y e_{14}. En el $AF_{ND} - \epsilon$ existe una transición de e_{10} a e_{11} con 0. Entonces en el nuevo autómata hay una transición de e_7 a e_{11} con 0.
- Transiciones para el estado e_{11}: desde el estado e_{11} se pueden alcanzar con transiciones ϵ los estados e_{11} y e_{12}. En el $AF_{ND} - \epsilon$ existe una transición de e_{12} a e_{13} con 1. Entonces en el nuevo autómata hay una transición de e_{11} a e_{13} con 1.
- Transiciones para el estado e_{13}: desde el estado e_{13} se pueden alcanzar con transiciones ϵ los estados e_{10}, e_{13} y e_{14}. En el $AF_{ND} - \epsilon$ existe una transición de e_{10} a e_{11} con 0. Entonces en el nuevo autómata hay una transición de e_{13} a e_{11} con 0.
- Son estados finales e_3, e_7 y e_{13} porque desde esos estados en el $AF_{ND} - \epsilon$ se puede alcanzar con transiciones ϵ un estado final.
- El autómata correspondiente sin transiciones ϵ se define como $AF_{ND} = < \{e_1, e_3, e_5, e_7, e_{11}, e_{13}\}, \{0, 1\}, \delta, e_1, \{e_3, e_7, e_{13}\} >$, con δ definida en el diagrama de transición de estados de la Figura 2.10

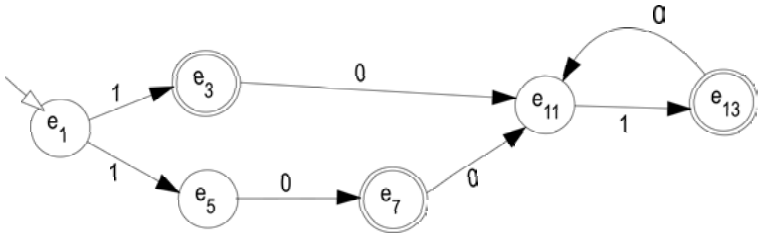


Figura 2.10: AF_{ND} correspondiente al $AF_{ND} - \epsilon$ del ejemplo 3

Construcción del AF_D a partir del AF_{ND} Dado el AF_{ND} $M_{ND} =< E_{ND}, A, \delta_{ND}, e_{0_{ND}}, F_{ND} >$, se define el autómata finito determinístico correspondiente $M_D =< E_D, A, \delta_D, e_{0_D}, F_D >$ aplicando el siguiente algoritmo.

Algoritmo para obtener el AF_D a partir de un AF_{ND}

- $E_D = P(E_{ND})$ (conjunto potencia de E_{ND}). Cada elemento de E_D se representa como $[e_1, e_2, ..., e_i]$ donde $e_1, e_2, ..., e_i \in E_{ND}$. Se debe notar que $[e_1, e_2, ..., e_i]$ es un único estado de M_D que corresponde a un conjunto de estados de M_{ND}.
- A: alfabeto
- δ_D: $E_D \times A \to E_D$, se define como
  - $\delta_D([e_1, e_2, ..., e_i], a) = [e_l, e_m, ..., e_k]$ sii
  - $\delta_{ND}(\{e_1, e_2, ..., e_i\}, a) = \delta^G(\{e_1, e_2, ..., e_i\}, a) = \{e_l, e_m, ..., e_k\}$, donde δ^G se define como $\delta^G(C, a) = \bigcup_{p \in C} \delta(p, a)$ (C: conjunto de estados) y $\delta^G(\oslash, a) = \oslash$

  Es decir que δ_D aplicada a un elemento $[e_1, e_2, ..., e_i]$ de E_D se calcula aplicando δ_{ND} a cada estado $e_1, e_2, ..., e_i$ de E_{ND}.
- $e_{0_D} = [e_{0_{ND}}]$
- F_D: conjunto de todos los estados de E_D que contienen al menos un estado final de M_{ND}.

Ejemplo 5: Construcción del AF_D correspondiente al AF_{ND} del ejemplo 1. El AF_D M_D correspondiente al AF_{ND} M_{ND} se define como:
$M_D =< E_D, \{0, 1\}, \delta_D, [e_0], F_D >$ donde la función de transición δ_D se encuentra definida en el cuadro 2.1.

δ_D	0	1
$[e_0]$	$[e_0, e_3]$	$[e_0, e_1]$
$[e_0, e_3]$	$[e_0, e_3, e_4]$	$[e_0, e_1]$
$[e_0, e_1]$	$[e_0, e_3]$	$[e_0, e_1, e_2]$
$[e_0, e_3, e_4]$	$[e_0, e_3, e_4]$	$[e_0, e_1, e_4]$
$[e_0, e_1, e_2]$	$[e_0, e_2, e_3]$	$[e_0, e_1, e_2]$
$[e_0, e_1, e_4]$	$[e_0, e_3, e_4]$	$[e_0, e_1, e_2, e_4]$
$[e_0, e_2, e_3]$	$[e_0, e_2, e_3, e_4]$	$[e_0, e_1, e_2]$
$[e_0, e_1, e_2, e_4]$	$[e_0, e_2, e_3, e_4]$	$[e_0, e_1, e_2, e_4]$
$[e_0, e_2, e_3, e_4]$	$[e_0, e_2, e_3, e_4]$	$[e_0, e_1, e_2, e_4]$

Cuadro 2.1: Definición de la función de transición para δ_D

Como $\delta_{ND}(e_0, 0) = \{e_0, e_3\}$, entonces $\delta_D([e_0], 0) = [e_0, e_3]$.
Como $\delta_{ND}(\{e_0, e_3\}, 0) = \delta^G(\{e_0, e_3\}, 0) = \delta_{ND}(e_0, 0) \cup \delta_{ND}(e_3, 0) = \{e_0, e_3\} \cup$

$\{e_4\} = \{e_0, e_3, e_4\}$ entonces $\delta_D([e_0, e_3], 0) = [e_0, e_3, e_4]$. De la misma forma se calcula δ_D para el resto de los estados. Se debe notar que se ha calculado δ_D para únicamente aquellos estados alcanzables desde el estado inicial y a partir de los cuales se puede alcanzar un estado final. Por lo tanto, el conjunto de estados E_D es $E_D = [e_0]$, $[e_0, e_3]$, $[e_0, e_1]$, $[e_0, e_3, e_4]$, $[e_0, e_1, e_2]$, $[e_0, e_1, e_4]$, $[e_0, e_2, e_3]$, $[e_0, e_1, e_2, e_4]$, $[e_0, e_2, e_3, e_4]$. El conjunto de estados finales F_D está formado por aquellos estados de E_D que contienen al menos un estado final de M_{ND}. Entonces $F_D = [e_0, e_3, e_4]$, $[e_0, e_1, e_2]$, $[e_0, e_1, e_4]$, $[e_0, e_2, e_3]$, $[e_0, e_1, e_2, e_4]$, $[e_0, e_2, e_3, e_4]$ Renombrando los estados correspondientes al AF_D se obtiene el cuadro 2.2

$[e_0]$	q_0
$[e_0, e_3]$	q_1
$[e_0, e_1]$	q_2
$[e_0, e_3, e_4]$	q_3
$[e_0, e_1, e_2]$	q_4
$[e_0, e_1, e_4]$	q_5
$[e_0, e_2, e_3]$	q_6
$[e_0, e_1, e_2, e_4]$	q_7
$[e_0, e_2, e_3, e_4]$	q_8

Cuadro 2.2: Renombrado de los estados para el AF_D

Función de transición δ_D se puede rescribir tal como se indica en el cuadro 2.3.

δ_D	0	1
q_0	q_1	q_2
q_1	q_3	q_2
q_2	q_1	q_4
q_3	q_3	q_5
q_4	q_6	q_4
q_5	q_3	q_7
q_6	q_8	q_4
q_7	q_8	q_7
q_8	q_8	q_7

Cuadro 2.3: Definición de la función de transición δ_D

Entonces $M_D = < \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}, \{0, 1\}, \delta_D, q_0, \{q_3, q_4, q_5, q_6, q_7, q_8\} >$ En la Figura 2.11 se encuentra el diagrama de transición de estados correspondiente a este autómata.

Minimización de un AF_D Para cada AF_D existe un AF_D con cantidad mínima de estados que acepta el mismo lenguaje. El algoritmo que permite obtener el AF_D con cantidad mínima de estados a partir de un AF_D se cita a continuación.

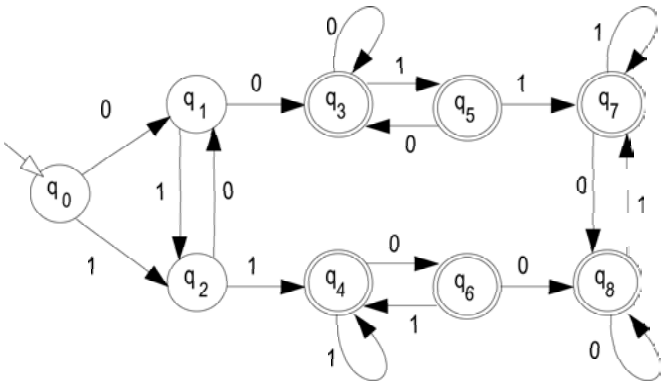


Figura 2.11: Diagrama de transición de estados para el AF_D resultante del ejemplo 5

Algoritmo para obtener un AF_D mínimo El algoritmo de minimización divide el conjunto de estados del AF_D en clases de equivalencia. Los pasos a seguir son los siguientes:

1. Eliminar los estados no alcanzables desde el estado inicial.
2. Eliminar los estados desde los cuales no es posible alcanzar un estado final.
3. Construir una partición $\prod_0$ del conjunto de estados, que consiste en dos grupos: estados finales y estados no finales.
4. Sea $K = 0$,
5. Definir $\prod_{K+1}$ de la siguiente manera: para cada grupo G de la partición $\prod_K$, dividir a G en subgrupos tales que dos estados s y t están en el mismo grupo sí y solo sí para todo símbolo a del alfabeto de entrada, los estados s y t van al mismo grupo de $\prod_K$.
6. $K = K + 1$
7. Si $\prod_K <> \prod_{K-1}$ volver al paso 5. En caso contrario, terminar.

Al finalizar el algoritmo se tiene que cada grupo de la partición $\prod_K$ representa un conjunto de estados equivalentes.

Ejemplo 6 : Minimización del AF_D resultante del ejemplo 5.

- Paso 1: no existen estados no alcanzables desde el estado inicial.
- Paso 2: no existen estados desde los cuales no es posible alcanzar un estado final.
- Paso 3: Como q_0, q_1, q_2 son estados no finales y $q_3, q_4, q_5, q_6, q_7, q_8$ son estados finales, tenemos $\prod_0 = \{\overline{q_0, q_1, q_2}, \overline{q_3, q_4, q_5, q_6, q_7, q_8}\}$

- Paso 4: Grupo $\{\overline{q_0, q_1, q_2}\}$. Como
  - desde q_0 con 0 se pasa al grupo $\overline{q_0, q_1, q_2}$
  - desde q_1 con 0 se pasa al grupo $\overline{q_3, q_4, q_5, q_6, q_7, q_8}$
  - desde q_2 con 0 se pasa al grupo $\overline{q_0, q_1, q_2}$

  se debe separar el estado q_1 de los estados q_0 y q_2.
  Para ver si q_0 y q_2 pueden quedar en el mismo grupo, se debe analizar qué ocurre con el símbolo 1:
  - desde q_0 con 1 se pasa al grupo $\overline{q_0, q_1, q_2}$
  - desde q_1 con 1 se pasa al grupo $\overline{q_3, q_4, q_5, q_6, q_7, q_8}$

  Entonces, como con 1 van a distintos grupos se deben separar también. Después de realizar un análisis similar para el grupo $\overline{q_3, q_4, q_5, q_6, q_7, q_8}$ la partición resultante es: $\prod_1 = \{\overline{q_0}, \overline{q_1}, \overline{q_2}, \overline{q_3, q_4, q_5, q_6, q_7, q_8}\}$
- Paso 5: Analizando $\prod_1$ se puede concluir que no hay manera de seguir particionando, ya que $\prod_2 = \prod_1$. Renombrando los estados según el cuadro 2.4 el AF_D mínimo se define como $MD_{min} = < \{p_0, p_1, p_2, p_3\}, \{0, 1\}, \delta, p_0, \{p_3\} >$, donde δ está definida por el diagrama de transición de estados de la Figura 2.12.

$\overline{q_0}$	p_0
$\overline{q_1}$	p_1
$\overline{q_2}$	p_2
$\overline{q_3, q_4, q_5, q_6, q_7, q_8}$	p_3

Cuadro 2.4: Estados renombrados para el autómata minimizado

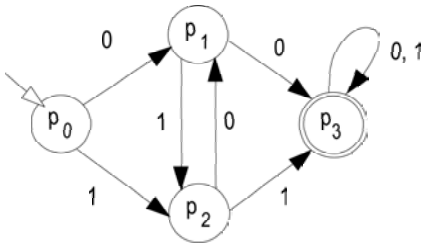


Figura 2.12: Diagrama de transición de estados para el autómata minimizado del ejemplo 6

2.1.2. Autómatas de Pila

A diferencia de los autómatas finitos, los autómatas de pila (AP) cuentan con una memoria auxiliar llamada *pila*, en la cual se pueden insertar o extraer

símbolos (denominados símbolos de pila), de acuerdo con el manejo *Last-In-First-Out (LIFO)*. Un AP permite reconocer un tipo de lenguajes en particular que se denominan lenguajes libres del contexto.

Definición 2:

Formalmente, un AP se define como una 7-upla
$M =< E, A, \delta, e_0, F, P, Z_0 >$ donde:

- E : Conjunto finito de estados
- A : Alfabeto de entrada
- δ: Función de transición
- e_0 : Estado inicial; $e_0 \in E$
- F : Conjunto de estados finales; $F \subseteq E$
- P : Alfabeto de la pila; $P \cap A = \oslash$
- Z_0 : Símbolo distinguido; $Z_0 \in P$

Como en los autómatas finitos, los de pila pueden ser clasificados en determinísticos (AP_D) y no determinísticos (AP_{ND}), diferenciándose en la forma en que se define su función de transición.

- Función de transición de un AP_D:

  $\delta : E \times (A \cup \{\epsilon\}) \times P \rightarrow E \times P^*$; dado un estado, un símbolo del alfabeto de entrada (o el símbolo ϵ) y un símbolo de la pila, se puede alcanzar un nuevo estado, modificando el estado de la pila.

  1. $\delta(e_i, a, X) = (e_j, \alpha)$
  2. $\delta(e_i, \epsilon, X) = (e_j, \alpha)$; sólo se garantiza que el autómata es determinístico si $\forall s \in A$, $\delta(e_i, s, X)$ no está definida, donde $a \in A$; $X \in P$; $\alpha \in P^*$; $e_i, e_j \in E$.

  Ejemplo 7: Supongamos el siguiente lenguaje L libre del contexto
  $L = \{\omega c \omega^R / \omega \in \{a,b\}^*\}$
  Un AP_D que reconoce cadenas del lenguaje L es el siguiente: $APD =< \{e_0, e_1, e_2\}, \{a, b, c\}, \{X, Y, Z_0\}, \delta, e_0, Z_0, \{e_2\} >$, en donde δ está definida por el diagrama de transición de estados de la Figura 2.13

- Función de transición para un AP_{ND} :
  $\delta : E \times (A \cup \{\epsilon\}) \times P \rightarrow P_f(E \times P^*)$; dado un estado, un símbolo del alfabeto de entrada (o el símbolo ϵ) y un símbolo de la pila, se puede alcanzar un conjunto de estados junto con la modificación del estado de la pila.

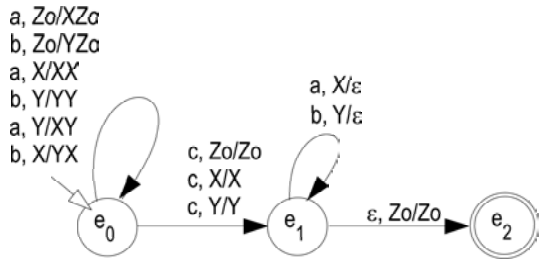


Figura 2.13: Diagrama de transición de estados para el AP del ejemplo 7

1. $\delta(e_i, a, X) = \{(e_j, \alpha_1), (e_k, \alpha_2), ...\}$
2. $\delta(e_i, \epsilon, X) = \{(e_j, \alpha_1), (e_k, \alpha_2), ...\}$

Donde $a \in A$; $X \in P$; $\alpha_1, \alpha_2 \in P^*$; $e_i, e_j, e_k \in E$.

En ambos casos, para determinismo y no determinismo, la transición $\delta(e_i, \epsilon, X)$, significa que se cambia de estado sin leer un nuevo símbolo de la cadena de entrada.

Ejemplo 8: Supongamos el siguiente lenguaje L libre del contexto

$$L = \{\omega\omega^R / \omega \in \{a,b\}^*\}$$

Un AP_{ND} que reconoce cadenas del lenguaje L es el siguiente: AP_{ND}=< $\{e_0, e_1, e_2\}$,$\{a,b,c\}$,$\{X,Y,Z_0\}$, δ, e_0, Z_0, $\{e_2\}$ >, en donde δ está definida por el diagrama de transición de estados de la Figura 2.14

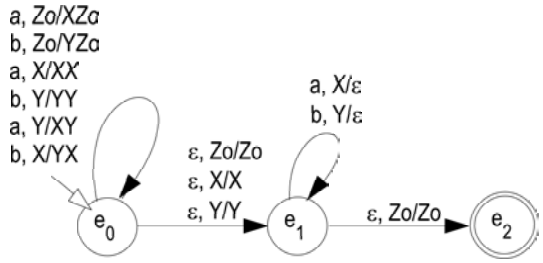


Figura 2.14: Diagrama de transición de estados para el AP_{ND} del ejemplo 8

El no determinismo de este autómata está dado por las transiciones del siguiente tipo:

- $\delta(e_0, a, X) = \{(e_0, XX)\}$
- $\delta(e_0, \epsilon, X) = \{(e_1, X)\}$

El no determinismo se produce debido a que en un mismo estado del autómata se pueden llevar a cabo dos acciones diferentes:

- Con la primera transición, consumiendo el símbolo a que se está leyendo en la cadena de entrada y teniendo el elemento X en el tope de la pila, se queda en el estado e_0.
- Con la segunda transición, sin avanzar en la cadena de entrada y teniendo el elemento X en el tope de la pila, se pasa desde el estado e_0 al estado e_1.

2.1.3. Máquinas de Turing

Las máquinas de Turing (MT) contienen una cinta de entrada en la que se encuentra la cadena a reconocer y una cabeza lectora que lee un solo símbolo de la cinta por vez. A diferencia de los autómatas finitos y los autómatas de pila sobre la cinta perteneciente a una MT se puede leer un símbolo y sobreescribirlo por otro, además, la cabeza lectora puede desplazarse a la izquierda, a la derecha o quedarse en el mismo lugar. Una MT permite reconocer un tipo de lenguaje en particular que se denominan lenguajes estructurados por frases.

Definición 3:

Las MT se definen formalmente como una 7-upla
$M =< E, A, \delta, e_0, F, C, B >$ donde:

- E : Conjunto finito de estados
- A : Alfabeto de entrada
- δ : Función de transición
- e_0 : Estado inicial; $e_0 \in E$
- F : Conjunto de estados finales; $F \subseteq E$
- C : Alfabeto de la cinta; $C = A \cup \{B\} \cup$ Símbolos_Auxiliares
  $A \cap$ Símbolos_Auxiliares $= \oslash$
- B : Símbolo blanco; $B \notin A$ y $B \in C$

La función de transición se define de acuerdo al tipo de MT. Para las MT determinísticas, la función de transición es $\delta : E \times C \rightarrow E \times C \times \{D, I, N\}$; donde $\{D, I, N\}$ son los posibles movimientos de la cabeza lectora (I significa que la cabeza lectora se mueve a la izquierda, D que se mueve a la derecha y N que no efectúa ningún movimiento).
Los tipos de transiciones de estados de una MT determinística se definen como

1. $\delta(e_i, x_k) = (e_j, Y, D)$
2. $\delta(e_i, x_k) = (e_j, Y, I)$
3. $\delta(e_i, x_k) = (e_j, Y, N)$

donde $x_k, Y \in C$; $e_i, e_j \in E$
La función de transición correspondiente a las MT no determinísticas se define como $\delta : E \times C \rightarrow P_f(E \times C \times \{D,I,N\})$; donde $\{D,I,N\}$ son los posibles movimientos de la cabeza lectora y P_f denota un subconjunto finito de $E \times C \times \{D,I,N\}$.
Las MT, además de poseer una cinta de entrada como se ha descrito anteriormente, pueden contener un conjunto de cintas auxiliares. Cada una de ellas posee una cabeza lectora independiente. Las MT que poseen más de un cinta se denominan MT multicinta y se diferencian de las de una cinta por la definición de su función de transición.
La función de transición de una MT multicinta determinística se define como $\delta : E \times C^k \rightarrow E \times (C \times \{D,I,N\})^k$; donde $\{D,I,N\}$ son los posibles movimientos de la cabeza lectora y k es la cantidad de cintas.

Ejemplo 9: Supongamos el siguiente lenguaje L sensible al contexto

$$L = \{\omega c \omega / \omega \in \{a,b\}^*\}$$

A continuación se muestra una MT determinística de dos cintas que reconoce cadenas del lenguaje L.
$MT_D =< \{e_0, e_1, e_2, e_3, e_4\}, \{a,b,c\}, \{a,b,c,X,B\}, \delta, B, \{e_4\} >$ donde la función de transición de estados es $\delta : E \times C_1 \times C_2 \rightarrow E \times (C_1 \times \{D,I,N\}) \times (C_2 \times \{D,I,N\})$ (cuadro 2.5)

δ	C_1	C_2	C_1		C_2		Nuevo estado
			N. Sím.	Mov.	N. Sím.	Mov.	
$e0$	a	B	a	N	X	D	e_1
	b	B	b	N	X	D	e_1
	c	B	c	D	B	N	e_3
e_1	a	B	a	D	a	D	e_1
	b	B	b	D	b	D	e_1
	c	B	c	N	B	I	e_2
e_2	c	a	c	N	a	I	e_2
	c	b	c	N	b	I	e_2
	c	X	c	D	X	D	e_3
e_3	b	b	b	D	b	D	e_3
	a	a	a	D	a	D	e_3
	B	B	B	N	B	N	e_4
e_4	-	-	-	-	-	-	-

Cuadro 2.5: Definición de la función de transición del ejemplo 9

La función de transición correspondiente a las MT multicinta no determinísticas se define como $\delta : E \times C^k \rightarrow P_f(E \times (C \times \{D,I,N\})^k)$; donde $\{D,I,N\}$ son los posibles movimientos de la cabeza lectora, k es la cantidad de cintas y P_f denota un subconjunto finito de $E \times (C \times \{D,I,N\})^k$.

Ejemplo 10: Supongamos el siguiente lenguaje L sensible al contexto $L = \{\omega\omega/\omega \in \{a,b\}^*\}$ A continuación se muestra una MT no determinística de 2-cintas que reconoce cadenas del lenguaje L.
$MT_{ND} =< \{e_0,e_1,e_2,e_3,e_4,e_5\},\{a,b\},\{a,b,X,B\},\delta,B,\{e_5\} >$, donde la función de transición de estados es $\delta : E \times C_1 \times C_2 \rightarrow E \times (C_1 \times \{D,I,N\}) \times (C_2 \times \{D,I,N\})$

δ	C_1	C_2	C_1		C_2		Nuevo
			N. Sím.	Mov.	N. Sím.	Mov.	estado
e_0	a	B	a	N	X	D	e_1
	b	B	b	N	X	D	e_1
e_1	a	B	a	D	a	D	e_1
	b	B	b	D	b	D	e_1
	a	B	a	N	B	I	e_2
	b	B	b	N	B	I	e_2
e_2	a	a	a	N	a	I	e_2
	a	b	a	N	b	I	e_2
	b	a	b	N	a	I	e_2
	b	b	b	N	b	I	e_2
	a	X	a	N	X	D	e_3
	b	X	b	N	X	D	e_3
e_3	a	a	a	D	a	D	e_3
	b	b	b	D	b	D	e_3
	B	B	B	N	B	N	e_4
e_4	-	-	-	-	-	-	

Cuadro 2.6: Definición de la función de transición del ejemplo 10

El no determinismo en la MT diseñada previamente está dado por las siguientes transiciones:

- $\delta(e_1,a,B) = \{(e_1,(a,D)(a,D));(e_2,(a,N)(B,I))\}$
- $\delta(e_1,b,B) = \{(e_1,(b,D)(b,D));(e_3,(b,N)(B,I))\}$

La primera transición significa que estando en el estado e_1 leyendo en la cinta C_1 el símbolo a y en la cinta C_2 el símbolo B, se pueden tomar dos acciones diferentes que dan origen al no determinismo. Una de estas acciones es pasar al estado e_1 escribiendo el símbolo a y moviéndose a la derecha, tanto en la cinta C_1 como en la C_2; mientras que la otra acción es pasar al estado e_2 escribiendo el símbolo a y no moviéndose en la cinta C_1 y, escribiendo el símbolo B y moviéndose a la izquierda en la cinta C_2. De forma análoga se puede interpretar el significado de la segunda transición.

Autómatas Linealmente Acotados

Un autómata linealmente acotado (ALA) es una máquina de Turing particular. La restricción sobre el modelo general de máquina de Turing es que las

transiciones se realizan sobre una cantidad de casilleros acotada. Es decir, en vez de tener una cinta de entrada infinita sobre la cual calcular, se restringe a una porción entre un casillero que contiene un símbolo de inicio "#", y un símbolo final "$", y se garantiza que las transiciones se van a realizar dentro de este conjunto de casilleros en la cinta de entrada. Para el caso del modelo multicinta se extiende para cada cinta la condición de acotar el espacio de trabajo. Un *ALA* permite reconocer un tipo de lenguajes en particular que se denominan lenguajes sensibles al contexto.

Definición 4:

$ALA =< E, A, C, \delta, e_0, B, F, \#, \$ >$ donde:

- E : Conjunto finito de estados
- A : Alfabeto de entrada
- e_0 : Estado inicial; $e_0 \in E$
- F : Conjunto de estados finales; $F \subseteq E$
- C : Alfabeto de la cinta; $C = A \cup \{B\}$ Símbolos_Auxiliares; $A \cap$ Símbolos_Auxiliares $= \oslash$
- B : Símbolo blanco; $B \notin A$ y $B \in C$
- $\#$: Símbolo que marca el casillero de inicio en la cinta
- \$: Símbolo que marca el casillero final en la cinta
- δ: $E \times C \rightarrow E \times C \times \{D, I, N\}$; donde $\{D, I, N\}$ son posibles movimientos de la cabeza lectora. No se permiten movimientos a la izquierda de "#" ni a la derecha de "$".

Los *ALA*, de la misma forma que las máquinas de Turing, también puede ser multicinta.

Definición 5:

$ALAM =< E, A, C, \delta, e_0, B, F, \#, \$ >$ donde:

- E : Conjunto finito de estados
- A : Alfabeto de entrada
- e_0 : Estado inicial; $e_0 \in E$
- F : Conjunto de estados finales; $F \subseteq E$
- C : Alfabeto de la cinta; $C = A \cup \{B\} \cup$ Símbolos_Auxiliares; $A \cap$ Símbolos_Auxiliares $= \oslash$

- B : Símbolo blanco; $B \notin A$ y $B \in C$
- $\#$: Símbolo que marca el casillero de inicio en la cinta
- \$: Símbolo que marca el casillero final en la cinta
- $\delta: E \times C^k \rightarrow E \times (C \times \{D, I, N\})^k$; donde D, I, N son posibles movimientos de la cabeza lectora. En ninguna de las cintas, se permiten movimientos a la izquierda de $\#$ ni a la derecha de \$.

Todo lo que se hace en multicinta puede realizarse con el modelo de una cinta. Es decir, que son modelos equivalentes ya que pueden reconocer los mismos lenguajes.

Máquinas de Turing compuestas

Gran parte del trabajo que se realiza durante el cómputo en una máquina de Turing consiste en tareas rutinarias y repetitivas, como mover la cabeza de la cinta de un lado de una cadena a otro, borrar una porción de la cinta o copiar el contenido de una cinta sobre otra auxiliar. Desde el punto de vista práctico sería muy complejo poder hacer un seguimiento de la máquina de Turing. La manera natural de construir una máquina de Turing compleja (o cualquier otro algoritmo o software complejos) es trabajar a partir de componentes más sencillos y reutilizables.
Esta situación es análoga a la programación. Los grandes sistemas se construyen combinando elementos reutilizables. De la misma forma se pueden construir máquinas de Turing a partir de otras. Y a su vez estos componentes pueden estar compuestos por otras máquinas de Turing. En el desarrollo de la Teoría de la Computación este concepto es necesario para construir demostraciones que muestran los límites de la computabilidad [8][25], aunque los autores no profundizan demasiado en algunos detalles sobre el diseño de tales máquinas. La forma que este tipo de autómatas se aborda en el presente trabajo, apunta a corresponderse con la invocación a procedimientos en un lenguaje estructurado.

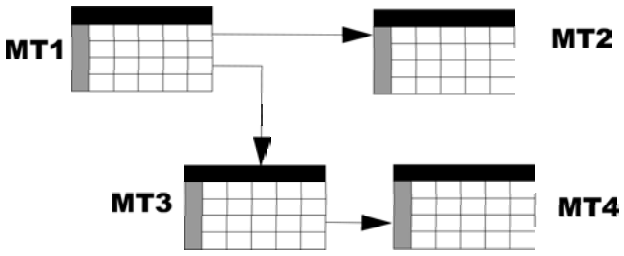


Figura 2.15: Composición de máquinas de Turing

Esquemas de composición En la bibliografía, en general se trata la composición de máquinas sólo para el caso en que la ejecución de una máquina sigue

a la ejecución de otra, y estas dos componen una nueva. Ampliando las posibilidades, se puede suponer que una máquina puede invocar la ejecución de otra y luego de la ejecución de ésta retomar el control, tal como sería una llamada a función. La Figura 2.15, muestra una máquina de Turing MT_1, que invoca la ejecución de otras dos máquinas: MT_2 y MT_3. A su vez, MT_3 invoca a la máquina MT_4. Al finalizar la ejecución de MT_2 la máquina podría continuar ejecutando normalmente o incluso invocar la ejecución de MT_3.

Mecanismo de ejecución Otro punto débil en las definiciones teóricas de máquinas compuestas es el mecanismo por el cual una máquina invoca a otra. En general el concepto de modularización existe a nivel teórico pero no se encuentra en la bibliografía existente cómo construir procedimientos basados en la composición de máquinas de Turing. En este trabajo, se considera, que una máquina puede invocar la ejecución, indicándolo en el nuevo estado de la definición de transición. Se debe hacer referencia a la máquina que se desea invocar, y a los estados en los que deberá continuar la ejecución en caso de terminar (la máquina invocada) en un estado de aceptación y de la misma forma para un estado de rechazo. Por lo tanto la definición para un nuevo estado podrá ser:

- Un estado de la misma máquina.
- Una llamada de invocación a otra máquina, especificando:
  - nombre de la máquina a invocar
  - estado de la máquina invocadora a donde debe continuar la ejecución en caso de aceptación de la cadena por parte de la máquina invocada.
  - estado de la máquina invocadora a donde debe continuar la ejecución en caso de rechazo de la cadena por parte de la máquina invocada.
  - pasaje de cintas como parámetros (se explica a continuación)

Pasaje de cintas como parámetros De alguna forma la máquina invocada debe recibir la cadena a evaluar. Esto se corresponde con el pasaje de parámetros en las invocaciones a métodos. Cada cinta tendrá un nombre. La máquina invocadora, deberá conocer las cintas que componen a la máquina invocada, de manera que en la invocación se pueda especificar, qué cintas de la máquina invocadora serán utilizados como parámetros de referencia por la máquina invocada.

En la Figura 2.16, se puede apreciar como la máquina MT_1 solicita la ejecución de la máquina MT_2. MT_1 posee una cinta denominada $Aux1$ que va a vincular con la cinta X de MT_2. Esto siginifica que el procesamiento que realice MT_2 quedará registrado en la cinta $Aux1$ de MT_1. En otras palabras, las modificaciones que realice MT_2 sobre su cinta X, ya sea de contenido como de posición de la cabeza lectora, estarán replicados en la cinta $Aux1$ de MT_1.
En el capítulo 4 se explica cómo se han implementado estos conceptos. En el capítulo 5, se desarrolla un ejemplo de máquina que involucra cuatro máquinas de Turing.

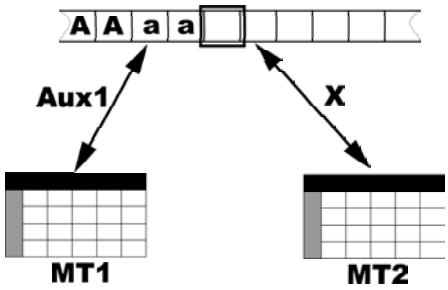


Figura 2.16: Compartiendo una cinta

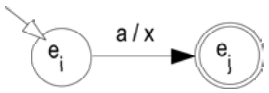
2.1.4. Autómatas Traductores

Este tipo de autómata, además de reconocer una cadena de entrada, permite generar una salida. En el presente trabajo se trataron los autómatas finitos y de pila traductores. La definición formal de cada uno de ellos se muestra a continuación.

Definición 6:

Definición formal de un autómata finito traductor
$MT =< E, A, \delta, e_0, F, S, \gamma >$ donde:

- E: Conjunto finito de estados,
- A: Alfabeto o conjunto finito de símbolos de entrada,
- δ: Es la función de transición de estados definida $\delta : E \times A \rightarrow E$
- e_0: Estado inicial $e_0 \in E$.
- F: Conjunto de estados finales o estados de aceptación. $F \subseteq E$.
- S: Alfabeto o conjunto finito de símbolos de salida
- γ Es la función de traducción definida $\gamma : E \times A \rightarrow S^*$ Ambas funciones $\delta : E \times A \rightarrow E$ y $\gamma : E \times A \rightarrow S^*$ están definidas sobre $E \times A$. Si existen $\delta(e_i, a) = e_j$ y $\gamma(e_i, a) = x$ donde $e_i, e_j \in E$; $a \in A$; $x \in S^*$ en el diagrama de transición de estados el valor de la traducción x se agrega sobre los arcos.

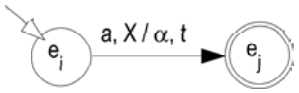


Definición 7:

Definición formal de un autómata de pila traductor AP_T. Un AP_T es simplemente un AP que se define como una 9-upla $AP_T =< E, A, P, \delta, e_0, Z_0, F, \gamma, S >$ donde:

- E : Conjunto finito de estados,
- A : Alfabeto o conjunto finito de símbolos de entrada,
- δ Es la función de transición de estados definida $\delta : E \times (A \cup \{\epsilon\}) \times P \to E \times P^*$
- e_0 : Estado inicial $e_0 \in E$.
- F : Conjunto de estados finales o estados de aceptación. $F \subseteq E$.
- S : Alfabeto o conjunto finito de símbolos de salida
- γ función de traducción definida como $\gamma : E \times (A \cup \{\epsilon\}) \times P \to S^*$

En el diagrama de transición de AP_T puede describirse como una extensión de la notación usada para AP. Si existe $\delta(e_i, a, X) = (e_j, \alpha)$ y además $\gamma(e_i, a, X) = t$ luego el arco queda rotulado de la siguiente manera:



donde $ei, ej \in E$; $a \in A$; $X \in P$; $\alpha \in P^*$; $t \in S^*$
Tanto para autómatas finitos traductores, como para autómatas de pila traductores, γ está definida siempre que δ esté definida, es decir que es generada una salida, para una cadena de entrada dada, si esta cadena de entrada ha sido reconocida por el autómata.

2.2. Gramáticas

Las gramáticas formales definen un lenguaje describiendo cómo se pueden generar las cadenas del mismo. Formalmente una gramática se define como una 4-upla

Definición 8: $G =< N, T, P, S >$

- N : Conjunto finito de símbolos no terminales
- T : Conjunto finito de símbolos terminales; $N \cap T = \oslash$
- P : Conjunto finito de reglas de producción; $\alpha \to \beta$, $\alpha = \varphi A \rho$ y $\beta = \varphi \omega \rho$ tal que $\varphi, \omega, \rho \in (N \cup T)^*$, $A \in (N \cup \{S\})$

- S : Símbolo distinguido; $S \notin (N \cup T)$

Existen cuatro tipos diferentes de gramáticas (de tipo 0, 1, 2 y 3) de acuerdo al tipo de lenguaje que describen, las cuales sólo difieren entre sí en el formato de las reglas de producción permitidas.

Definición 9: Derivaciones
Dada una gramática $G = (N, T, P, S)$, para definir una derivación, primero se deben definir dos relaciones $\overset{\Rightarrow}{G}$ y $\overset{*}{\underset{G}{\Rightarrow}}$ entre cadenas en $(N \cup T)^*$. Si $A \to \beta$ es una producción (o regla) de P y α y γ son cualquier cadena en $(N \cup T)^*$, entonces $\alpha A \gamma \overset{\Rightarrow}{G} \alpha\beta\gamma$. Se dice que la regla de producción $A \to \beta$ es aplicada a la cadena $\alpha A \gamma$ para obtener $\alpha\beta\gamma$. Dos cadenas están relacionadas por $\overset{\Rightarrow}{G}$ exactamente cuando la segunda es obtenida de la primera por una aplicación de alguna de las reglas de producción de P.
Supongamos que $\alpha_1, \alpha_2, ..., \alpha_m$ son cadenas en $(N \cup T)^*$, tal que $m \geq 1$, y

$$\alpha_1 \overset{\Rightarrow}{G} \alpha_2,\ \alpha_2 \overset{\Rightarrow}{G} \alpha_3,\ ...,\ \alpha_{m-1} \overset{\Rightarrow}{G} \alpha_m$$

Entonces se dice que $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$ o que α_1 deriva a α_m en la gramática G. Es decir que $\overset{*}{\underset{G}{\Rightarrow}}$ es la clausura reflexiva y transitiva de $\overset{\Rightarrow}{G}$. Dicho de otra forma, $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ si β se obtiene de α por la aplicación de cero o más reglas de producción de P.

2.2.1. Gramáticas Regulares (Tipo 3)

Describen los lenguajes regulares (aquellos reconocidos por un AF), y el formato de sus reglas de producción puede ser

- Lineal a derecha: $A \to aB$ ó $A \to a$
- Lineal a izquierda: $A \to Ba$ ó $A \to a$

donde $A \in (N \cup \{S\})$, B$\in N$ y $a \in T$. En ambos casos se puede incluir la regla de producción $S \to \epsilon$ si el lenguaje que se quiere generar contiene la cadena vacía.

Ejemplo 11: supongamos el siguiente lenguaje $L = \{\ x/x\{0,1\}^*$ y x contiene la subcadena 00 ó x contiene la subcadena $11\}$.
La gramática correspondiente a este lenguaje es $G = (\{A, B, C\}, \{0,1\}, P, S)$, donde P puede ser uno de los conjuntos del cuadro 2.2.1, dependiendo de si se consideran las reglas de producción lineales a derecha o a izquierda.

A continuación se muestra cómo se puede verificar si de la gramática descrita anteriormente se puede derivar una cadena del lenguaje L. Si la cadena a derivar es 10110, el árbol de derivación correspondiente es el de la Figura 2.17. En este árbol se puede ver en color más oscuro la elección de la regla que conduce a la cadena buscada, en cada punto. Por ejemplo, desde el símbolo distinguido, existen dos reglas aplicables, la regla 1 o la regla 2. En este punto, la elección de la regla 1 no conducirá a la cadena buscada, por lo que se abandona su derivación

	Reglas de Producción lineales a derecha	Reglas de Producción lineales a izquierda
1.	$S \to 0A$	$S \to A0$
2.	$S \to 1B$	$S \to B1$
3.	$A \to 0C$	$A \to C0$
4.	$A \to 0$	$A \to 0$
5.	$A \to 1B$	$A \to B1$
6.	$B \to 0A$	$B \to A0$
7.	$B \to 1C$	$B \to C1$
8.	$B \to 1$	$B \to 1$
9.	$C \to 0C$	$C \to C0$
10.	$C \to 0$	$C \to 0$
11.	$C \to 1C$	$C \to C1$
12.	$C \to 1$	$C \to 1$

y la rama correspondiente se puede ver en gris. Continuando por la otra rama, la que corresponde a la elección de la regla 2, se presentan tres alternativas que son, elegir la regla 6, 7, ó la regla 8. Sólo la elección de la regla 6 conduce a la cadena buscada, por lo cual, las demás ramas se colorean en gris. Por lo tanto siguiendo las ramas oscuras del arbol, desde la raíz (símbolo distinguido), se puede alcanzar la cadena en cuestión.

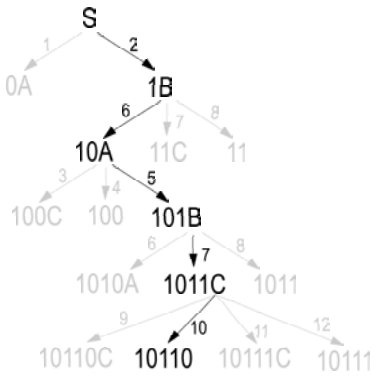


Figura 2.17: Árbol de derivación correspondiente a la cadena 10110

Algoritmo para Obtener la Gramática Regular desde el AF

Existe un algoritmo que permite obtener una gramática regular que genera un lenguaje regular dado a partir del AF que reconoce ese lenguaje. Este algoritmo es descrito a continuación.

Algoritmo 7: Obtención de la gramática regular a partir de un AF. Los pasos a seguir son los siguientes:

1. Asociar al estado inicial el símbolo distinguido S.
2. Asociar a cada estado del autómata (menos el estado inicial) un símbolo no terminal. Si al estado inicial llega algún arco asociar también un símbolo no terminal (además del símbolo distinguido).
3. Para cada transición definida $\delta(e_i, a) = e_j$, agregar al conjunto de reglas de producción, la regla $A \rightarrow aB$, siendo A y B los símbolos no terminales asociados a e_i y e_j respectivamente. Si e_j es un estado final, agregar también la regla $A \rightarrow a$. Si e_j es el estado inicial (tiene dos símbolos asociados, el distinguido y un no terminal), utilizar el símbolo no terminal (de esta manera se evita que el símbolo distinguido aparezca a la derecha de una regla de producción).
4. Si el estado inicial es también final agregar la regla de producción $S \rightarrow \epsilon$.

2.2.2. Gramáticas Libres del Contexto (Tipo 2)

Describen los lenguajes libres del contexto (aquellos reconocidos por un autómata de pila), y el formato de sus reglas de producción es el siguiente $A \rightarrow \omega$ donde $A \in (N \cup \{S\})$ y $\omega \in (N \cup T)^* - \{\epsilon\}$ Se puede incluir la regla de producción $S \rightarrow \epsilon$ si el lenguaje que se quiere generar contiene la cadena vacía.

Ejemplo 12: supongamos el siguiente lenguaje
$L = \{\omega c \omega^R / \omega \in \{a,b\}^*\}$
$G = (\{A\}, \{a,b,c\}, P, S)$
donde P contiene las siguientes reglas de producción:
$S \rightarrow A$
$A \rightarrow aAa$
$A \rightarrow bAb$
$A \rightarrow c$

2.2.3. Gramáticas Sensibles al Contexto (Tipo 1)

Los lenguajes sensibles al contexto (aquellos reconocidos por un autómata linealmente acotado) pueden generarse por gramáticas sensibles al contexto cuyo formato de reglas de producción es el siguiente $\gamma A \beta \rightarrow \gamma \omega \beta$ donde $A \in (N \cup \{S\})$, β, $\gamma \in (N \cup T)^*$ y $\omega \in (N \cup T)^* - \{\epsilon\}$. Se puede incluir la regla de producción $S \rightarrow \epsilon$ si el lenguaje que se quiere generar contiene la cadena vacía.

Ejemplo 13: supongamos el siguiente lenguaje L
$L = \{a^n b^n c^n / n > 0\}$
$G = < \{A,B,C\}, \{a,b,c\}, S, P >$
donde P contiene las siguientes reglas de producción:
$S \rightarrow A$
$A \rightarrow aABC$

$A \to abC$
$CB \to XB$
$XB \to XC$
$XC \to BC$
$bB \to bb$
$bC \to bc$
$cC \to cc$

Capítulo 3

Análisis y Diseño de la aplicación

Inicialmente se recabaron y analizaron los requerimientos funcionales y no funcionales que debía cubrir la aplicación. Posteriormente se analizaron interfaces de otros productos similares a fines de recabar las virtudes de cada uno y la posibilidad de incorporarlas en la aplicación. Para el desarrollo se eligió una metodología en espiral, que permitiera incrementar las funcionalidades esperadas y que pudiera aportar resultados sin estar finalizada la aplicación. En consideración de los requerimientos, se decidió utilizar una arquitectura cliente-servidor. Esto permitiría separar y potenciar, por un lado la funcionalidad de reconocimiento de cadenas y por otro la interfaz con el usuario.

3.1. Requerimientos

Los requerimientos para la aplicación, claramente se pueden clasificar en funcionales[1],y no funcionales[2].

3.1.1. Requerimientos funcionales

- Permitir editar distintos tipos de autómatas, como son el AF, AP, y MT.
- Decidir si una cadena pertenece o no a un lenguaje.
- Componer una MT a partir de otros autómatas.
- Guardar y recuperar los archivos que representan los autómatas y gramáticas.

[1] Aquellas tareas que se espera que pueda cumplir la aplicación
[2] La forma en que se espera que se puedan cumplir las tareas

- Editar gramáticas de tipo GR[3], GLC[4], GSC[5].
- Generar cadenas de distintos tipos de gramáticas
- Obtener la derivación de cadenas respecto de una gramática en particular
- Aplicación de distintos algoritmos sobre AF, como obtención de AF mínimo, conversión a determinístico, generación de expresión regular, generación de gramática equivalente y viceversa.

3.1.2. Requerimientos no funcionales

- Interfaz amigable: El usuario deberá poder ingresar los datos al sistema (definir autómatas y cargar cadenas) siguiendo su intuición y experiencia en programas similares o no tan similares. Las funcionalidades deben ser fáciles de acceder, y los resultados sencillos de interpretar. Se debe entender que los usuarios potenciales del sistema son alumnos en una etapa de ingreso a la universidad, por lo que su conocimiento sobre sistemas puede ser escaso o nulo, o peor aún viciado por sistemas de interfaces de aplicaciones de ofimática o diseño gráfico, que hacen importante hincapié en el diseño de la interfaz.
- Eficiencia de ejecución. La aplicación no puede demorar en arrojar resultados. Esto podría aburrir al usuario, de manera que prefiera verificar sobre papel sus ejercicios, abandonando el uso de la herramienta.
- Mantenibilidad. La utilidad de todo sistema depende de que tan bien han sido resueltos los requerimientos funcionales, y si estos requerimientos varían, debido a modificaciones del universo del discurso, el sistema debe tener la facilidad de readaptarse a estos cambios. Si los contenidos del curso "Ciencias de la Computación I" variaran en el tiempo, como es muy probable, el sistema deberá poder modificarse en forma sencilla para soportarlos.

3.2. Metodología de desarrollo

Como metodología de desarrollo se eligió una metodología en espiral (ver Figura 3.1), ya que esta metodología permite una evolución incremental de la aplicación. A medida que se va desarrollando se van obteniendo distintas funcionalidades y se pueden ir obteniendo y verificando resultados. Se eligió esta metodología debido a las características de la aplicación. Como la función principal de reconocimiento de cadenas es común a todos los autómatas[6], y la MT

[3] Gramáticas Regulares
[4] Gramáticas Libres del Contexto
[5] Gramáticas Sensibles al Contexto
[6] No así los requerimientos de aplicaciones de algoritmos

contiene la potencialidad de los otros autómatas[15], es natural pensar en diseñar la solución primero para ésta y luego para los demás autómatas. Por otro lado, se pensó en separar la edición de la interpretación, para permitir mejoras independientes tanto en el servidor como en el editor. Esta separación durante el desarrollo integral, permite un mejor *testing*, ya que se pueden aislar y analizar los errores, clasificándolos según si se tratan de la edición de los autómatas o de su ejecución en el servidor. Ambos desarrollos, tanto como la aplicación en su totalidad, seguirían metodologías en espiral.

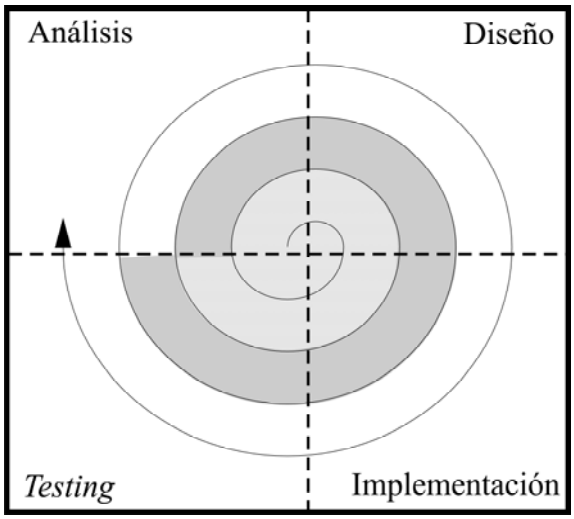


Figura 3.1: Esquema de desarrollo.En gris claro, el desarrollo de la aplicación intérprete, y en gris oscuro, el desarrollo del editor de máquinas.

3.3. Diseño de la aplicación

El diseño global de la aplicación sigue una arquitectura cliente-servidor, donde el servidor actúa como intérprete de las máquinas editadas por los clientes. Estos clientes a su vez interpretan los resultados del servidor. De esta manera los clientes podrán ser las máquinas de los alumnos, con menos recursos de procesamiento. El servidor procesará sobre una sola máquina de mayor potencia. Esto reporta diversas ventajas. En principio la separación de funciones, permitirá eventuales mejoras, del cliente o del servidor. Siempre que se conozca y respete la interfaz de comunicación, se podrán diseñar editores mejores, de otras plataformas, idiomas, e incluso servidores más eficientes. Por otro lado, las aplicaciones gráficas tienden a tener cierta inestabilidad, más aún en condiciones de desarrollo. Por este motivo, el desarrollo del cliente gráfico separado del resto de la aplicación permitirá un mejor y más rápido *testing* de la interfaz a usuario al descartar errores en el servidor. Para la descarga de la aplicación por Internet,

este fraccionamiento reviste una gran ventaja al momento de actualizar, ya que sólo bastará indicar qué parte ha sido modificada. En las secciones siguientes se desarrollan el diseño general de la aplicación, y los diseños particulares del editor y del servidor.

3.3.1. Diseño global

Tal como se indicó previamente, la aplicación consiste en un programa servidor y un programa cliente. Una configuración de ejecución implica un proceso servidor en comunicación con uno o más procesos cliente. Los clientes envían solicitudes de ejecución de un autómata con una cadena al servidor y éste responde con el resultado.
El proceso de reconocimiento de una cadena, independientemente del tipo de autómata, sería el siguiente:

1. El editor envía la definición de autómata, junto con la cadena a reconocer.
2. El servidor convierte ese autómata a la MT equivalente. Dependiendo de la interpretación, las alternativas son
   a) La cadena no es aceptada: el servidor envía un mensaje de "cadena no aceptada".
   b) El servidor entra en un ciclo de iteración muy largo y envía un mensaje "ejecución muy larga"
   c) La cadena es aceptada: el servidor envía un mensaje de "cadena aceptada", junto con los estados intermedios por los que el intérprete pasó hasta alcanzar un estado final.
3. El editor recibe una lista de tuplas[7], con lo cual puede simular sobre el autómata que se está editando, la ejecución. Existirá una tupla activa, que inicialmente será la inicial, y el usuario podrá navegar el resultado hacia adelante y hacia atrás, en forma secuencial o aleatoria cambiando la tupla activa, alterando la representación visual del autómata y el estado de reconocimiento de la cadena.

Por lo tanto la solución integra los siguientes componentes:

- MTServer: Es el servidor.
- MTEditor: El editor de autómatas. También permite la ejecución de algunos algoritmos.
- Gramática: El editor de gramáticas.
- AftTools: No es una aplicación, sino una librería de enlace dinámico, que contiene componentes comunes y funciones reutilizables para las aplicaciones.

[7]Estructuras que representan los estados intermedios por lo que atraviesa el autómata desde el estado inicial hasta un estado final.

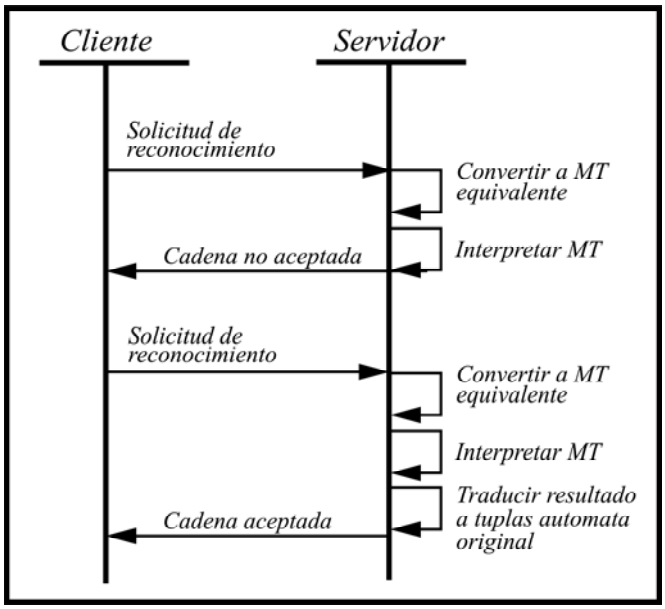


Figura 3.2: Ejemplo de reconocimiento para dos cadenas

3.3.2. Interfaz de comunicación

El servidor y el cliente se comunican a través de una interfaz de tipo *Socket asíncrono*, sobre un protocolo TCP/IP. Por medio de estos *sockets*, se puede abstraer la comunicación entre cliente y servidor, a un intercambio de información por medio de archivos. Todo el diseño se realiza considerando que cliente y servidor se comunican por medio de un archivo de texto, aunque en realidad se trata de un flujo de datos. El diagrama de eventos de la Figura 3.2, representa un ejemplo de comunicación entre el cliente y el servidor. Cuando el cliente desea enviar información al servidor, escribe sobre un archivo especial, el cual es enviado por el *socket*. El servidor recibe un evento indicando que se ha recibido un archivo e inicia un proceso de lectura. Cuando la interpretación de la máquina de Turing termina, el servidor envía un archivo al cliente, con el resultado de la ejecución. El editor, que no se bloquea esperando respuesta, reacciona a un evento de archivo recibido, y lee el archivo especial de comunicación y muestra el resultado. En el apéndice A se puede ver un detalle del esquema de los archivos de intercambio.

3.4. Diseño del servidor

El servidor tiene dos responsabilidades principales:

- Comunicación con los clientes: debe organizar la comunicación con los clientes, encolando los pedidos de interpretación, emitiendo los resultados,