INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"POR MI PATRIA Y POR MI BIEN"

Tesis:

"Programa simulador de una máquina de Turing no determinista"

Para obtener el grado de:

Maestro en Ciencias en Ciencias de la Computación

Presenta:

I.S.C Lorena Alcudia Chagala

Director de Tesis:

Dr. Rodolfo A. Pazos Rangel

Co-Director de Tesis:

Dr. José A. Martínez Flores

Ciudad Madero, Tamaulipas **Noviembre, 2012**

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

SECRETARÍA DE EDUCACIÓN PÚBLICA

Cd. Madero, Tamps; a 22 de Noviembre de 2012.

OFICIO No.: U5.249/12
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN
DE TESIS

C. ING. LORENA ALCUDIA CHAGALA
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"PROGRAMA SIMULADOR DE UNA MÁQUINA DE TURING NO DETERMINISTA"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"Por mi patria y por mi bien"

M. P. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

S.E.P.
DIVISION DE ESTUDIOS
DE POSGRADO E
INVESTIGACION
I T C M

c.c.p.- Archivo
Minuta
MYCHC 'MLCO ' jar

Ave. 1°. de Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, C.P. 89440 Cd. Madero, Tam.
Tels. (833) 3 57 48 20, Fax: (833) 357 48 20, Ext. 1002, email: itcm@itcm.edu.mx
www.itcm.edu.mx

DECLARACIÓN DE ORIGINALIDAD

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones.

En caso de infracción de los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de la infracción y revelo de esta a mi director y codirector de tesis, así como al Instituto Tecnológico de Ciudad Madero y sus autoridades.

Ciudad Madero, Tamaulipas, noviembre 2012

I.S.C. Lorena Alcudia Chagala

DEDICATORIAS

A Dios por guiar mi camino cada día y por tomarme de la mano en cada paso de mi vida.

A mis padres Griselda Lagos Palaceto, Jacinto Chagala Cuevas, e Isabel Chagala Lagos por su gran apoyo y comprensión.

A mi amado hijo Ángel Jacinto Baxin Alcudia por llegar en el momento preciso a mi vida y ser un aliciente y motivación para cada uno de los proyectos tanto profesionales como de vida.

A mi adorado esposo Ángel Manuel Baxin Lucho por su amor y apoyo incondicional.

AGRADECIMIENTOS

Especial agradecimiento a mi director de tesis de maestría, Dr. Rodolfo A. Pazos Rangel, por el tiempo dedicado en cada una de sus asesorías para lograr la culminación de este proyecto. De igual forma hago extensivo este agradecimiento a mi codirector de tesis, Dr. José A. Martínez Flores, así como a los miembros de mi comité tutorial por su retroalimentación y su tiempo en cada una de las presentaciones de avance de tesis, Dra. Laura Cruz Reyes, M.C. José Apolinar Ramírez Saldivar.

También quiero agradecer a la Dra. Claudia G. Gómez Santillán, así como a los miembros del programa de posgrado de la Maestría en Ciencias en Ciencias de la Computación: catedráticos y personal administrativo, por el apoyo y la orientación brindada.

Finalmente agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT), a la Dirección General de Educación Superior Tecnológica (DGEST), y sobre todo al Instituto Tecnológico de Ciudad Madero (ITCM), por las facilidades y el apoyo otorgado para la realización y culminación de este proyecto de tesis.

RESUMEN

Actualmente podemos resolver una gran cantidad de problemas mediante el uso de la programación; sin embargo, ¿cómo podemos saber cuál es la mejor manera de resolverlos?, y más aún ¿cómo saber si nuestra solución es buena? Para dar respuesta a estas interrogantes, recurrimos al análisis de la complejidad y nos adentramos a la teoría de la complejidad computacional, definida como la rama de la teoría de la computación que estudia (de manera teórica) los recursos requeridos en el proceso de cómputo de un algoritmo para resolver un problema. Tales recursos son el tiempo (número y tipo de pasos de ejecución de un algoritmo) y el espacio (cantidad de memoria utilizada para la solución de un problema).

En este trabajo se plantea la problemática en la resolución de problemas con máquinas de Turing, ya que en la literatura se explican problemas bastante rudimentarios, con lo cual imaginarse problemas complejos con máquinas de Turing, resulta ser muy complicado.

Para el entendimiento de problemas complejos se desarrolló una herramienta de simulación de máquinas de Turing, esta herramienta permite resolver problemas NP-completos. El programa de simulación se presenta en dos versiones, en la versión 1 las reglas de transición de la máquina de Turing de un problema en cuestión se introducen por archivos de texto y en la versión 2 las reglas de transición son introducidas mediante un editor de autómatas.

Para probar el funcionamiento de la herramienta de simulación se experimentó con el problema del circuito Hamiltoniano y el problema de Partición. En base a la experimentación realizada con los problemas de decisión anteriores se obtuvo una herramienta capáz de resolver problemas complejos, permitiendo la introducción de reglas de transición de manera modular, y facilitando al usuario mediante la introducción de reglas abreviadas la generación autómatica en un 80% de las reglas de transición pertenecientes a un problema NP-completo.

SUMMARY

Nowadays we can resolve large quantity of problems using programation; however ¿how can we know what th best answer? To answer these questions, delved to the computational complexity analysis and we move to the theory of computation that studies (in theory) the resources in the computing process of an algorithm to resolve a problema. Such resources are time (number and Type of execution steps of an algorithm) and space (amount of memory used for the solution problem).

This paper presents the problem in solving problems with Turing machines, as explained in the literature fairly rudimentary problems, thereby imagine complex problems with Turing machines, turns out to be very complicated.

For the understanding of complex problems we developed a simulation tool Turing machines, this tool can solve NP-complete problems. The simulation program comes in two versions, version 1 transition rules of the Turing machine problem in question are introduced by text files and version 2 transition rules are introduced by an automata editor.

To test the performance of the simulation tool are experienced with the Hamiltonian circuit problem and the problem of Partition. Based on experiments conducted with the above decision problems was obtained a tool capable of solving complex problems, allowing the introduction of transition rules in a modular way, and providing the user by introducing of rules abbreviated the automatic generation of 80% of transition rules belonging to an NP-complete problem.

CONTENIDO

CONTENIDO DE FIGURAS

CONTENIDO DE TABLAS

CAPÍTULO 1
Introducción

Este capítulo contiene la motivación para el desarrollo de la tesis, su justificación, la descripción del problema, los antecedentes, así como el objetivo, los alcances y limitaciones.

1.1 Motivación

Entender el modelo teórico denominado máquina de Turing (MT) parece ser muy sencillo, esto ocurre porque ya se han desarrollado herramientas de simulación de máquinas de Turing tanto deterministas como no deterministas. Sin embargo, estas herramientas permiten una ejemplificación rudimentaria, lo cual permite mostrar de manera didáctica ejemplos sencillos, con lo cual resulta difícil imaginar problemas reales, que son extremadamente más grandes, como se ve en el Anexo A.

Este proyecto se desarrolló en el área de Complejidad Computacional del Instituto Tecnológico de Ciudad Madero, y su principal interés está centrado en la implementación de una herramienta que simule una máquina de Turing no determinista. Esta máquina es capaz de resolver problemas NP-completos, permitiendo la generación de la configuración de la máquina y proporcionando como resultado una respuesta SI o NO a cualquier problema de decisión de tamaño limitado.

1.2 Justificación

Mediante el modelo teórico de máquina de Turing y el análisis de complejidad de algoritmos, fue posible la categorización de problemas computacionales. Tales problemas se denominaron P y NP, y cuyas soluciones se basan en el determinismo y no determinismo de las máquinas de Turing que se aplican para resolverlos.

Resulta complicado imaginar el funcionamiento de una máquina de Turing no determinista (MTND) que resuelva un problema NP-completo; por lo tanto, fue necesario elaborar un programa simulador de una máquina de Turing no determinista.

Existen herramientas que muestran el funcionamiento de máquinas de Turing. Sin embargo, el propósito principal de estas herramientas es mostrar de manera didáctica el modelado de autómatas. He aquí la importancia de haber implementado un programa simulador de una máquina de Turing no determinista (MTND), el cual permite a catedráticos y estudiantes entender

mejor su funcionamiento. Esta herramienta de simulación resuelve problemas NP-completos (en donde resolver un problema NP-completo significa contestar sí o no), que es prácticamente imposible resolver con las herramientas de simulación del estado del arte.

1.3 Definición del problema

El desarrollo de este proyecto está constituido en dos etapas. La etapa A que hace referencia a la versión 1 del programa simulador. Es decir se desarrolló un programa simulador no gráfico, el cual se alimenta de archivos de texto. Y la etapa B, perteneciente a una versión 2 del programa simulador, consistió en la implementación de una interfaz gráfica de usuario.

A su vez la etapa A está constituida por 2 fases: la fase 1 y la fase 2. La fase 1 está constituida por dos subfases: la subfase 1A y la subfase 1B. La subfase 1A consistió en la implementación del simulador de la MTND y la subfase 1B se refirió a la implementación de un traductor de autómatas.

1.4 Antecedentes

La tesis de maestría desarrollada por Ong de la Cruz, es el antecedente de gran relevancia para el desarrollo de este proyecto de tesis. Uno de sus principales objetivos, fue revisar transformaciones polinomiales entre pares de problemas NP-completos. Es decir, se revisaron de manera teórica problemas NP-completos con máquinas de Turing no deterministas, y por lo tanto se planteó la necesidad de ver el funcionamiento en la práctica de problemas NP-completos con MTND, mediante un programa de simulación. El objetivo general de la tesis de Ong se centró en determinar si existe alguna anomalía dentro de varias transformaciones polinomiales, enfocándose en aquellas transformaciones que se encuentran en la rama de transformaciones que van desde el problema de Partición al problema SAT.

1.5 Objetivo

Simular una máquina de Turing no determinista.

Objetivo particular:

Implementar una herramienta que simule una máquina de Turing no determinista.
Nota: el programa debe estar basado en la máquina de Turing propuesta por Garey and Johnson [Garey & Johnson, 1979].

1.6 Alcance y limitaciones

La versión 1 del programa simulador de MTND:

- Consta de una sola cinta y está basado en el modelo propuesto por Garey and Johnson [Garey & Johnson, 1979].
- Consta de un módulo de adivinación y un módulo de verificación. Se permite la introducción de reglas de transición en ambos módulos.
- Las reglas de transición del programa (autómata de estados finito) para el problema NP-completo son introducidas mediante archivos de texto.

La versión 2 del programa simulador de MTND:

- Consta de una sola cinta y está basado en el modelo propuesto por Garey and Johnson [Garey & Johnson, 1979].
- Consta de un módulo de adivinación y un módulo de verificación. Sin embargo, en la interfaz gráfica sólo se permite la introducción de las reglas de transición del módulo de verificación.
- Las reglas de transición abreviadas del programa para el problema NP-completo son introducidas por un editor de autómatas.

En las versiones 1 y 2 del programa simulador de una MTND se permite la introducción de programas para diferentes problemas NP-completos; sin embargo, el simulador fue probado con el problema del Circuito Hamiltoniano y el problema de Partición.

CAPÍTULO 2
Marco Teórico

En este capítulo se establece un marco teórico fundamental, en el cual se proporcionan las definiciones más importantes para la comprensión del presente proyecto de tesis. Para una explicación más profunda acerca de las definiciones aquí presentadas, se recomienda consultar las fuentes bibliográficas referidas en el texto.

2.1 Problema de decisión

Un problema de decisión (en un sistema formal) es aquél que sólo tiene dos posibles respuestas: sí o no, uno o cero. A un ejemplar *p* de un problema de decisión Π se le denomina ejemplar-sí si la respuesta del problema *p* es sí, y se le denomina ejemplar-no si su respuesta es no. En términos más formales podemos definir un problema de decisión $\Pi(D,Y)$ como una pareja de un conjunto de ejemplares D y un conjunto de ejemplares-sí Y donde $Y \subseteq D$ [Garey & Johnson, 1979].

2.2 Clasificación de problemas

P: es el conjunto de todos los problemas de decisión que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial [Garey & Johnson, 1979].

NP: es el conjunto de todos los problemas de decisión reconocibles por una máquina de Turing no determinista en tiempo polinomial [Garey & Johnson, 1979].

NP-completo: es el conjunto de todos los problemas de decisión de la clase NP que no se encuentran en P y que actualmente son considerados intratables [Garey & Johnson, 1979].

2.3 Máquina de Turing determinista y la clase P

Una máquina de Turing determinista consta de los siguientes elementos [Garey & Johnson, 1979]:

a) Una cinta infinita bidireccional en la cual estarán escritos los símbolos de un ejemplar codificado de un problema de decisión.

b) Una cabeza de lectura-escritura que pueda leer los símbolos de la cinta infinita para poder pasarlos al control de estados finitos, que escriba los símbolos especificados por el control de estados finitos y que se mueva hacia la izquierda o a la derecha de la cinta.

c) Un control de estados finitos implementado como un autómata finito determinista que procese los símbolos leídos en la cinta.

La figura 2.1 muestra un diagrama con los elementos que constituyen una MTD y sus interrelaciones.

Un programa para una MTD (máquina de Turing determinista) cuenta con la siguiente información:

1. Un conjunto finito Σ de símbolos de entrada y un conjunto $\Gamma = \Sigma \cup \{b\}$, donde b denota un símbolo blanco.

2. Un conjunto finito Q de estados, que incluye un estado inicial q_0 y tres estados de paro q_Y, q_N, y q_I (que corresponden a los estados donde la MTD se detiene cuando la cadena de entrada codifica un ejemplar-sí, un ejemplar-no o un ejemplar inválido, respectivamente). Nota: conviene aclarar que en esta definición se ha incluido un nuevo estado de paro (q_I) con el fin de distinguir los casos en que el ejemplar a procesar sea inválido; es decir, que no cumpla los requisitos del problema en cuestión, ya sea en lo que se refiere a la definición del problema o en su codificación para ser introducido a la máquina.

3. Una función de transición δ: $(Q - [q_Y, q_N, q_I]) \times \Gamma \rightarrow Q \times \Gamma \times [-1, +1]$ **[ ]**.

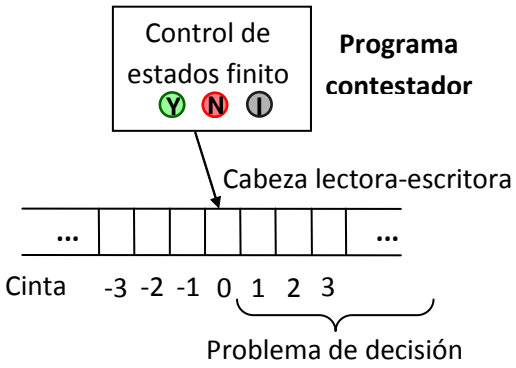


Figura 2.1.- Representación esquemática de una MTD. (Adaptado de [Garey & Johnson, 1979]).

2.4 Máquina de Turing no determinista y la clase NP

El modelo de una máquina de Turing no determinista (MTND) es muy semejante al de una MTD, excepto por la incorporación de un módulo de "adivinación" que tiene su propia cabeza de escritura [Garey & Johnson, 1979].

La MTND cuenta con los siguientes elementos:

a) Una cinta infinita bidireccional, igual a la de la MTD.

b) Una cabeza escritora, que escriba los símbolos provenientes del módulo de "adivinación" sobre la cinta bidireccional.

c) Un modulo de adivinación, el cual es implementado como un autómata finito no determinista. Este módulo genera aleatoriamente una solución candidata para el problema de decisión. También controla la cabeza escritora que graba en la cinta los símbolos de la solución candidata.

d) Una cabeza de lectura-escritura, igual a la de la MTD.

e) Un control de estados finitos que verifica si una solución candidata es en efecto una solución para el problema de decisión.

La figura 2.2 muestra un diagrama con los elementos que constituyen una MTND y sus interrelaciones.

Entonces, de manera informal, la clase NP está conformada por todos los problemas de decisión que pueden ser resueltos por una MTND de tiempo polinomial. Definiendo formalmente la clase NP tenemos:

NP = {*L*: existe una MTND de tiempo polinomial para un *programa M tal* que $L = L_M$}.

Un problema de decisión Π se dice que pertenece a la clase NP bajo un esquema de codificación *e* si el lenguaje $L[\Pi, e] \in$ NP; es decir, si existe una MTND de tiempo polinomial que resuelva Π bajo el esquema de codificación *e*.

Otra forma de ver a la clase NP es como el conjunto de lenguajes que pueden ser verificados en tiempo polinomial.

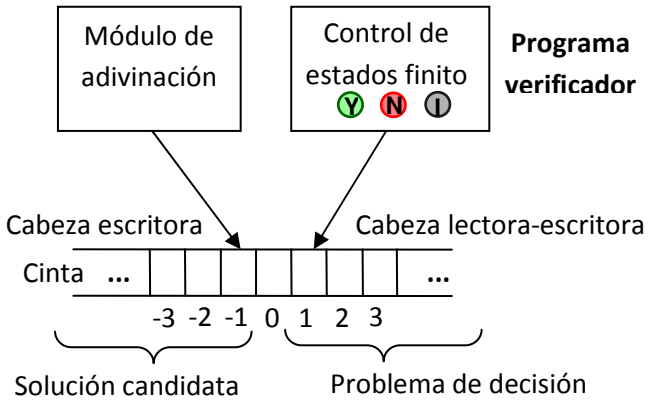


Figura 2.2.- Representación esquemática de una MTND. (Adaptado de [Garey & Johnson, 1979]).

2.5 Relación entre P y NP

Ya que cualquier problema Π que pueda ser resuelto en tiempo polinomial por una MTD también puede ser resuelto en tiempo polinomial por una MTND, entonces P $\subseteq$ NP. *"La comprobación de lo anterior puede verse cuando usamos un algoritmo determinista en la etapa de verificación para un algoritmo no determinista"*. Si existe un problema $\Pi \in$ P y una MTD *A* de tiempo polinomial para Π, entonces se puede obtener una MTND de tiempo polinomial para Π al aplicar *A* en la etapa de verificación. En resumen, se tiene que los problemas de la clase P son un subconjunto de la clase NP (figura 2.3) [Garey & Johnson, 1979].

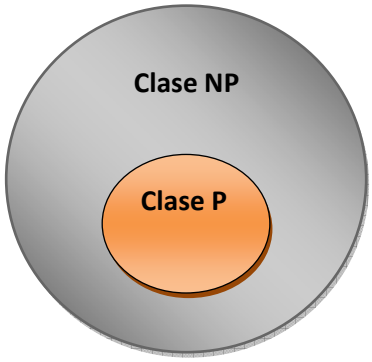


Figura 2.3.- Representación gráfica de la relación entre la clase P y la clase NP (Adaptado de [Garey & Johnson, 1979]).

2.6 Teoría de la completitud-NP

En 1971, Stephen Cook estableció las bases de lo que hoy se conoce como la teoría de la completitud-NP al suponer que el problema de Satisfactibilidad (SAT) es intratable y que por el momento no hay una solución determinista en tiempo polinomial para este problema [S.A. Cook, 1971].

La teoría de la completitud-NP provee varias técnicas para demostrar que un problema dado *es tan difícil* de resolver como una larga lista de otros problemas conocidos. Un problema se definirá como una pregunta general a ser contestada, el cual generalmente posee varios parámetros, o variables libres cuyos valores aún no están especificados. Un problema está descrito por:

1. una descripción general de todos los parámetros y
2. una declaración de qué propiedades son necesarias satisfacer.

Un ejemplar de un problema es una asignación de valores a todos los parámetros de un problema.

La base fundamental de la teoría de la completitud-NP es el problema de Satisfactibilidad, dado que es la base para la comprobación de que un problema dado es NP-completo. Esto se realiza al transformar un problema dado al problema SAT. De esta forma, dado un problema Π, si $\Pi \propto$ SAT entonces $\Pi \in$ NP-completo (donde la notación $\Pi_1 \propto \Pi_2$ significa transformación polinomial de Π_1 a Π_2).

De manera formal, un lenguaje L pertenece a la clase NP-completa si $L \in$ NP y, para todos los otros lenguajes $L' \in$ NP, $L' \propto L$.

2.7 Demostración de la completitud-NP

Para probar que un problema Π_1 pertenece al conjunto NP-completo se siguen los siguientes pasos:

1. Demostrar que $\Pi_1 \in$ NP.
2. Demostrar que $\Pi_1 \in$ NP-completo.

Para demostrar que $\Pi_1 \in$ NP, debe existir una MTND de tiempo polinomial que resuelva Π_1.

Para demostrar que $\Pi_1 \in$ NP-completo, primero es necesario encontrar un problema Π_2, tal que $\Pi_2 \in$ NP-completo. Después se formula una función de transformación de $\Pi_1 \propto \Pi_2$. Por último demostrar que la función de reducción $\Pi_1 \propto \Pi_2$ es polinomial [Garey & Johnson, 1979].

La figura 2.4 muestra cómo se ha demostrado la completitud NP de varios problemas a partir del problema de Satisfactibilidad. De estos problemas conviene destacar los problemas Partición (Partition) y Circuito Hamiltoniano (HC), ya que estos problemas se van a usar para las pruebas del simulador de la MTND.

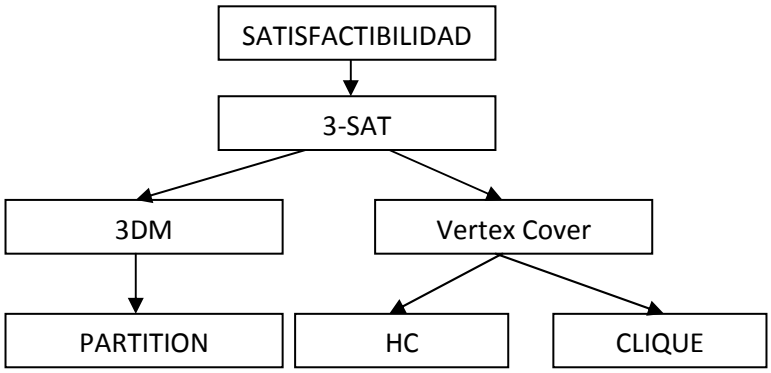


Figura 2.4.- Diagrama de secuencia de transformaciones para comprobación de completitud-NP (Adaptado de [Garey & Johnson, 1979]).

2.8 Simulación y emulación

Es la técnica de representación del mundo real mediante un programa de computadora. Una simulación debe imitar los procesos internos y no sólo los resultados de lo que se está simulando [www.thefreedictionary].

El término “emulación” proviene del verbo “emular”, el cual significa imitar o reproducir. Por lo tanto la emulación computacional es cuando un sistema imita o reproduce otro sistema [www.techterms.com].

De acuerdo con estas definiciones el software desarrollado es un simulador, y es como se le considera en los artículos consulados en el estado del arte. Sin embargo, si se considera que una MTND es una computadora, entonces la versión sin interfaz gráfica del software podría considerarse también como un emulador.

CAPÍTULO 3
Estado del Arte

En este capítulo se describen algunas de las herramientas que simulan máquinas de Turing no deterministas, así mismo se presentan las conclusiones que muestran las limitaciones del alcance de uso en su ambiente de simulación.

3.1 Simuladores de máquinas de Turing

Actualmente existen simuladores de máquinas de Turing no deterministas (MTND), cuya finalidad didáctica consiste en el aprendizaje de máquinas de Turing, lenguajes formales y autómatas. A continuación se describen algunos simuladores relacionados con este proyecto de tesis.

3.1.1 Uber Turing Machine (UTM)

UTM es un simulador de máquinas de Turing que permite programar máquinas de Turing mediante la creación de algoritmos, los cuales pueden ser almacenados para volver a ser utilizados. Cuando se habla de algoritmo se hace referencia a un conjunto de reglas de transición almacenadas y editadas en forma de tabla, en donde las filas de la tabla corresponden a los estados utilizados y las columnas, los símbolos involucrados en el procesamiento del problema escrito en la cinta. Los conjuntos de estados y símbolos del alfabeto pueden ser modificados, es decir, se permite la añadidura de nuevos estados y símbolos. Cabe mencionar que al añadir una fila en medio de la tabla causará la renumeración de las instrucciones existentes para conservar la lógica del algoritmo. El formato de las reglas de transición es el siguiente: <nuevo símbolo><movimiento><estado nuevo>. Para una comprensión más detallada, la figura 3.1 muestra un ejemplo de máquina de Turing con la herramienta UTM. Por ejemplo, la regla "_ R 1" significa que la cabeza debe escribir el símbolo "_" en la cinta, después debe moverse a la derecha, y la máquina deberá cambiar al estado 2.

	_	1	*	#
1	_R 1	1N 2		
2	*L 3	1R 2		
3	_R 4	1L 3		
4		_R 5	_R 13	
5		1R 5	*R 6	
6	*L 6	#R 6	*R 7	#L 6
7		1R 7	*L 11	1R 8
8	1L 9	1R 8	*R 8	#R 8
9		1L 9	*L 10	
10		1R 7		#L 10
11		#L 11	*L 12	
12	_R 4	1L 12		
13			_L 0	_R 13

Figura 3.1.- Ejemplo de un algoritmo de multiplicación con máquina de Turing.

Los símbolos aceptados por el simulador se muestran en la tabla 3.1.

Tabla 3.1.-Símbolos aceptados por UTM.

Carácter	Código decimal	Código hexadecimal
!	33	21h
#	35	23h
$	36	24h
%	37	25h
*	42	2Ah
+	43	2Bh
0—9	48-57	30h-39h
=	61	3Dh
?	63	3Fh
@	64	40h
A—Z	65-90	41h-5Ah
^	94	5Eh
_	95	5Fh
a—z	97-122	61h-7Ah

3.1.2 JFLAP

JFLAP tiene como finalidad el apoyo didáctico en lenguajes formales, teoría de autómatas y máquinas de Turing. Permite la construcción de máquinas de Turing de una cinta y múltiples cintas. Para poder crear una máquina de Turing, se deberá proceder primeramente a la creación del autómata, en donde por cada transición se leerá el valor actual de la cinta, el valor que se deberá escribir en la cinta y el movimiento que se deberá realizar (izquierda, derecha, no moverse) [Ryan, Finley, H. Susa].

En términos formales JFLAP define a una máquina de Turing *M* como una séptupla $M = (Q, \sum, \Gamma, \delta, q_s, \square, F)$ donde:

Q es el conjunto de estados internos $\{q_i \mid i$ es un entero no negativo$\}$.

∑ es el alfabeto de entrada.

Γ es el conjunto finito de símbolos en la cinta.

δ es la función de transición.

□ es el símbolo blanco.

q_s es miembro Q (es el estado inicial).

F es un subconjunto de Q (es el conjunto de estados finales).

3.1.3 JFAST

JFAST es una herramienta simple y muy fácil de utilizar para crear, editar y simular autómatas finitos y máquinas de estado. Provee un método de interacción simple e intuitivo con numerosos tipos de máquinas de estados finitos, incluyendo autómatas deterministas y no deterministas y máquinas de Turing. Es una herramienta didáctica, usada de manera muy sencilla. Es decir, para la creación de pequeños ejemplos de máquinas de Turing, sólo basta con elegir dentro de su ambiente gráfico un rectángulo para representar a los estados y un lazo que representa las conexiones entre los estados. Al colocar el lazo de conexión aparece una ventana en la cual se edita el símbolo de entrada, el movimiento y el símbolo de salida [T.M. White, 2004].

3.1.4 Automaton Simulator (AutoSim)

AutoSim permite diseñar y simular una variedad de máquinas teóricas, incluyendo autómatas finitos deterministas, autómatas finitos no deterministas y máquinas de Turing. Es una herramienta muy sencilla, la cual tiene como objetivo didáctico el aprendizaje de autómatas y máquinas de Turing. Las reglas de transición son introducidas en el ambiente de simulación, mediante un autómata. Este autómata es creado a partir de figuras como la elipse, la cual representa a los estados y un lazo, el cual representa la conexión entre estados. Al seleccionar un estado se presentan las opciones de “estado inicial”, “estado final”, y “eliminar”. Al seleccionar el lazo se presentan las opciones de “símbolo de entrada”, “símbolo de salida”, “movimiento”, y “eliminar” [C. Burch, 2001].

3.1.5 Visual Automata Simulator (VAS)

Fue diseñado para ser usado de forma fácil. Cuenta con características que permiten conocer el funcionamiento de autómatas de estados finitos (autómatas deterministas y no deterministas) y máquinas de Turing [Bovet 2005].

Cuando se crean operaciones de máquinas de Turing con VAS se pueden utilizar comandos de una lista de patrones, los cuales se muestran en la tabla 3.2

Tabla 3.2.-Comandos aceptados por VAS.

Descripción	comando
Mover a la izquierda	L
Mover a la izquierda hasta	L=<S>
Mover a la izquierda hasta que no	L!<S>
Mover a la derecha	R
Mover a la derecha hasta	R=<S>
Llamar a otra máquina	C<MACHINE>
Operación aceptada	Y
Operación rechazada	N

3.1.6 Conclusiones

Las herramientas de simulación del estado del arte presentan diversas limitaciones para el usuario, dentro de las cuales se encuentran las siguientes:

- Permiten el procesamiento de problemas muy sencillos.
- No permiten la introducción de reglas de transición abreviadas; por lo tanto, no se generan reglas de transición de manera automática.
- Los autómatas no se pueden definir de manera modular; lo cual dificulta definir autómatas muy grandes.

A semejanza de lo que ocurre con programas codificados en lenguajes de alto nivel, un programa (autómata de estados finito) muy grande para una MTND es indispensable organizarlo de manera modular. El simulador de MTND de este proyecto de tesis permite la introducción del autómata para algún problema NP-completo de manera modular.

Además, es importante destacar que, a diferencia de las herramientas del estado del arte descritas anteriormente, la herramienta de simulación producto de esta tesis permite la generación autómatica de reglas de transición para cada uno de los módulos en donde se requiere mover la cabeza de lectura-escritura a una posición de la cinta. Esta característica permite a los usuarios ahorrarse hasta 50% de la escritura de las reglas de transición de un autómata.

Para propósitos didácticos, esta herramienta permite un mejor entendimiento en el tema de complejidad computacional.

CAPÍTULO 4
Análisis y solución conceptual del problema

Antes de empezar a describir una máquina de Turing no determinista, es conveniente describir primero con cierto detalle una máquina de Turing determinista.

Una máquina de Turing determinista es un modelo teórico que contiene: una cinta infinita, una cabeza lectora-escritora (la cual se mueve a través de la cinta), y un mecanismo de control de estados finitos, el cual constituye el programa contestador (ver figura 2.1). A continuación se describe un ejemplo de máquina de Turing determinista (MTD).

Para entender el ejemplo de MTD, es importante conocer el concepto de autómata finito determinista (AFD). El AFD es un sistema determinista; es decir, para cada estado en que se encuentre el autómata, y con algún símbolo del alfabeto leído, existe a lo más una transición posible desde ese estado y con ese símbolo.

Ejemplo de MTD:

Considérese el problema de decisión de la suma de dos números binarios de dos dígitos, en el cual se busca dar respuesta a la pregunta siguiente: ¿la suma de dos números binarios de dos dígitos es menor o igual a un determinado número binario de tres dígitos?

Consideremos el siguiente ejemplar: ¿10 + 11 <= 110? En este caso se codifica el ejemplar en la cinta de la siguiente manera: 10,11,110b.

Cinta:

	b	*b*	*b*	1	0	,	1	1	,	1	1	0	*b*	
…	-2	-1	0	1	2	3	4	5	6	7	8	9	10	

Figura 4.1.- Representación codificada del ejemplar en la cinta.

En la figura 4.1 se puede observar la representación codificada del ejemplar en la cinta, donde las celdas desde la cero hacia la izquierda contienen símbolos blancos, a partir de la celda uno en adelante contiene los símbolos del ejemplar a evaluar, y por último, la celda final contiene un símbolo blanco que tiene la función de delimitador del ejemplar.

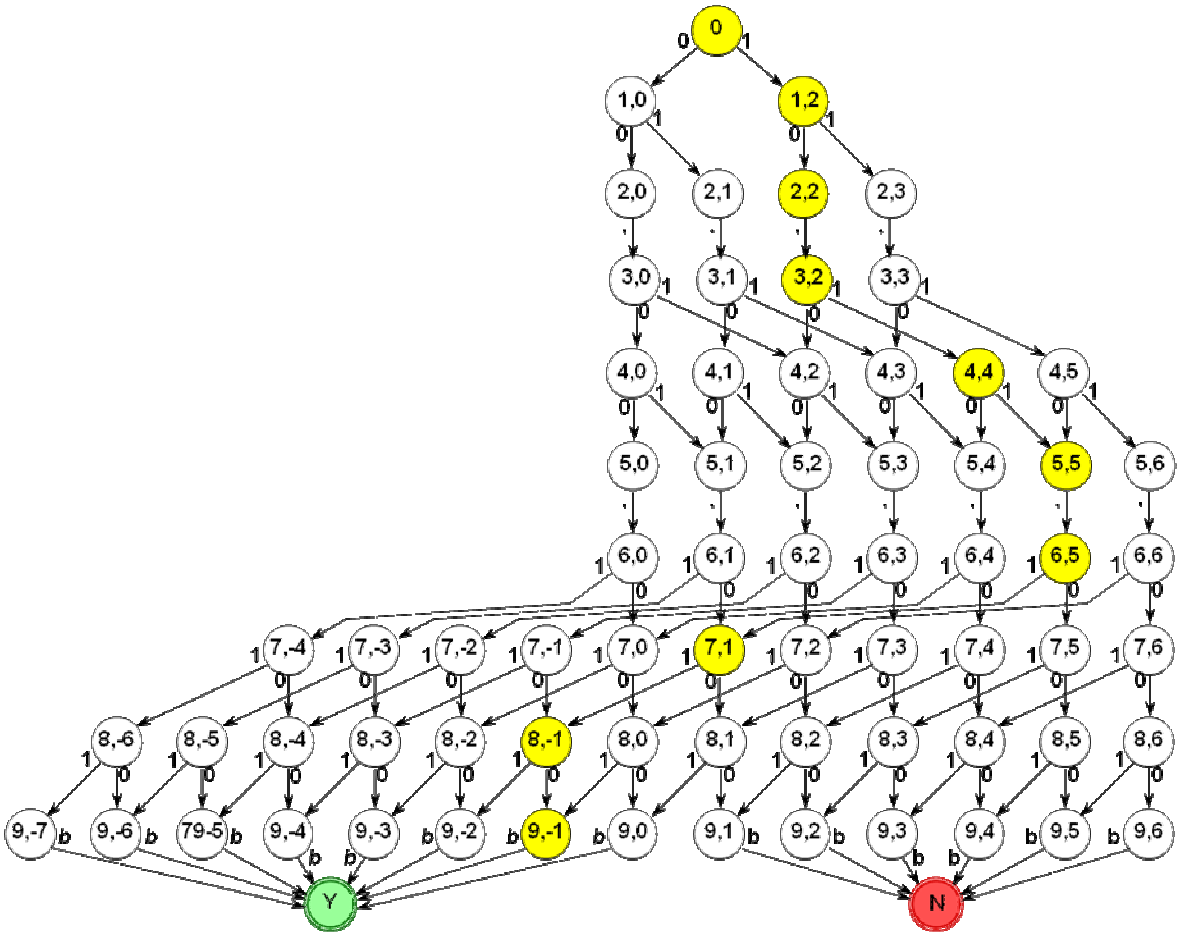


Figura 4.2.- Autómata finito determinista de la máquina de Turing.

El AFD de la figura 4.2 está diseñado de tal manera que la etiqueta de cada estado lleve el cálculo del valor numérico de los símbolos que se hayan leído hasta el momento de llegar al estado; específicamente, mediante una etiqueta del estado que indica el paso en el que se encuentra el programa y el valor acumulado de los dígitos binarios leídos hasta dicho paso. Nota: para cada estado, la etiqueta indica el subíndice de la letra "q" que usualmente se usa para representar los estados; por ejemplo, la etiqueta 4,3 indica el estado $q_{4,3}$, lo cual significa que el AFD se encuentra en el paso 4 y que el valor acumulado de los dígitos leídos hasta dicho paso es 3.

Para determinar si el tercer número es mayor o igual a la suma de los dos primeros, se realiza una resta, de tal manera que si al restarle a la suma de los dos primeros números el valor del tercer número nos da un número negativo o cero, entonces sí se cumple la condición y se considera como un ejemplar-sí. En caso contrario, no se cumple la condición y es un ejemplar-no.

Para los dos primeros dígitos, las etiquetas de los estados indican la suma de los valores de los dígitos binarios. Por ejemplo, si el primer número fuera "11" el AFD llegaría primero por al estado $q_{1,2}$, dado que el valor del primer "1" en binario es dos, y después llegaría al estado $q_{2,3}$, dado que

el valor binario del segundo “1” es uno, el cual se suma al dos que se llevaba acumulado en el estado anterior ($q_{1,2}$).

Para realizar la resta el proceso es similar al de la suma, solamente que en lugar de sumar el valor binario del dígito leído, se resta. Por ejemplo, si al final de la suma de los dos números binarios el AFD se encontrara en el estado $q_{5,4}$ y el primer dígito del tercer número binario fuera “1”, entonces después de leer este dígito el AFD llegaría al estado $q_{6,0}$, debido a que el símbolo “1” representa el valor de cuatro en el tercer número binario debido a que se encuentra en la posición más significativa; y por lo tanto, se le resta al cuatro que se llevaba acumulado en el estado anterior.

Al final, cuando la MTD llega un estado en el paso siete cuya etiqueta indique un valor negativo o nulo, entonces la MTD termina en el estado de aceptación q_Y, de lo contrario termina en el estado de rechazo q_N.

Cabe destacar que, si la MTD recibe un símbolo no esperado para el estado en el que se encuentra, entonces termina en un estado inválido. En el AFD se suprimieron las transiciones que llevan al estado inválido q_I debido a que ocuparían mucho espacio y harían que el diagrama fuera muy complejo. Por ejemplo, si el AFD se encontrara en el estado $q_{1,2}$ y el siguiente símbolo leído fuera una “,”, esto ocasionaría que la función de transición llevara al estado q_I, un estado inválido, puesto que se esperaría leer enseguida un símbolo “1” o “0”.

Para este ejemplo podemos definir nuestros conjuntos base:

Conjunto de símbolos del alfabeto: $\sum$ = {0, 1, “,”}.

Conjunto de símbolos de entrada: Γ = {0, 1, “,”, b}.

Conjunto de estados: Q = {q_0, $q_{0,1}$, ..., q_Y, q_N, q_I}.

Función de transición: δ= {$q \in Q, s \in \Gamma$}.

La tabla de transición de estados correspondiente a δ quedaría como se muestra en la tabla 4.1.

Tabla 4.1.-Tabla de transición de estados.

q	0	1	","	b
q_0	$(q_{1,0}, 0, +1)$	$(q_{1,2}, 1, +1)$	q_I	q_I
$q_{1,0}$	$(q_{2,0}, 0, +1)$	$(q_{2,1}, 1, +1)$	q_I	q_I
$q_{1,2}$	$(q_{2,2}, 0, +1)$	$(q_{2,3}, 1, +1)$	q_I	q_I

$q_{2,0}$	q_I	q_I	$(q_{2,0}, ",", +1)$	q_I
$q_{2,1}$	q_I	q_I	$(q_{2,1}, ",", +1)_I$	q_I
$q_{2,2}$	q_I	q_I	$(q_{2,2}, ",", +1)$	q_I
$q_{2,3}$	q_I	q_I	$(q_{2,3}, ",", +1)$	q_I
$q_{3,0}$	$(q_{4,0}, 0, +1)$	$(q_{4,2}, 1, +1)$	q_I	q_I
$q_{3,1}$	$(q_{4,1}, 0, +1)$	$(q_{4,3}, 1, +1)$	q_I	q_I
$q_{3,2}$	$(q_{4,2}, 0, +1)$	$(q_{4,4}, 1, +1)$	q_I	q_I
$q_{3,3}$	$(q_{4,3}, 0, +1)$	$(q_{4,5}, 1, +1)$	q_I	q_I
$q_{4,0}$	$(q_{5,0}, 0, +1)$	$(q_{5,1}, 1, +1)$	q_I	q_I
$q_{4,1}$	$(q_{5,1}, 0, +1)$	$(q_{5,2}, 1, +1)$	q_I	q_I
$q_{4,2}$	$(q_{5,2}, 0, +1)$	$(q_{5,3}, 1, +1)$	q_I	q_I
$q_{4,3}$	$(q_{5,3}, 0, +1)$	$(q_{5,4}, 1, +1)$	q_I	q_I
$q_{4,4}$	$(q_{5,4}, 0, +1)$	$(q_{5,5}, 1, +1)$	q_I	q_I
$q_{4,5}$	$(q_{5,5}, 0, +1)$	$(q_{5,6}, 1, +1)$	q_I	q_I
$q_{5,0}$	q_I	q_I	$(q_{5,0}, ",", +1)$	q_I
$q_{5,1}$	q_I	q_I	$(q_{5,1}, ",", +1)$	q_I
$q_{5,2}$	q_I	q_I	$(q_{5,2}, ",", +1)$	q_I
$q_{5,3}$	q_I	q_I	$(q_{5,3}, ",", +1)$	q_I
$q_{5,4}$	q_I	q_I	$(q_{5,4}, ",", +1)$	q_I
$q_{5,5}$	q_I	q_I	$(q_{5,5}, ",", +1)$	q_I
$q_{5,6}$	q_I	q_I	$(q_{5,6}, ",", +1)$	q_I
$q_{6,0}$	$(q_{7,0}, 0, +1)$	$(q_{7,-4}, 1, +1)$	q_I	q_I

$q_{6,1}$	$(q_{7,1}, 0, +1)$	$(q_{7,-3}, 1, +1)$	q_I	q_I
$q_{6,2}$	$(q_{7,2}, 0, +1)$	$(q_{7,-2}, 1, +1)$	q_I	q_I
$q_{6,3}$	$(q_{7,3}, 0, +1)$	$(q_{7,-1}, 1, +1)$	q_I	q_I
$q_{6,4}$	$(q_{7,4}, 0, +1)$	$(q_{7,0}, 1, +1)$	q_I	q_I
$q_{6,5}$	$(q_{7,5}, 0, +1)$	$(q_{7,1}, 1, +1)$	q_I	q_I
$q_{6,6}$	$(q_{7,6}, 0, +1)$	$(q_{7,2}, 1, +1)$	q_I	q_I
$q_{7,-4}$	$(q_{8,-4}, 0, +1)$	$(q_{8,-6}, 1, +1)$	q_I	q_I
$q_{7,-3}$	$(q_{8,-3}, 0, +1)$	$(q_{8,-5}, 1, +1)$	q_I	q_I
$q_{7,-2}$	$(q_{8,-2}, 0, +1)$	$(q_{8,-4}, 1, +1)$	q_I	q_I
$q_{7,-1}$	$(q_{8,-1}, 0, +1)$	$(q_{8,-3}, 1, +1)$	q_I	q_I
$q_{7,0}$	$(q_{8,0}, 0, +1)$	$(q_{8,-2}, 1, +1)$	q_I	q_I
$q_{7,1}$	$(q_{8,1}, 0, +1)$	$(q_{8,-1}, 1, +1)$	q_I	q_I
$q_{7,2}$	$(q_{8,2}, 0, +1)$	$(q_{8,0}, 1, +1)$	q_I	q_I
$q_{7,3}$	$(q_{8,3}, 0, +1)$	$(q_{8,1}, 1, +1)$	q_I	q_I
$q_{7,4}$	$(q_{8,4}, 0, +1)$	$(q_{8,2}, 1, +1)$	q_I	q_I
$q_{7,5}$	$(q_{8,5}, 0, +1)$	$(q_{8,3}, 1, +1)$	q_I	q_I
$q_{7,6}$	$(q_{8,6}, 0, +1)$	$(q_{8,4}, 1, +1)$	q_I	q_I
$q_{8,-6}$	$(q_{9,-6}, 0, +1)$	$(q_{9,-7}, 1, +1)$	q_I	q_I
$q_{8,-5}$	$(q_{9,-5}, 0, +1)$	$(q_{9,-6}, 1, +1)$	q_I	q_I
$q_{8,-4}$	$(q_{9,-4}, 0, +1)$	$(q_{9,-5}, 1, +1)$	q_I	q_I
$q_{8,-3}$	$(q_{9,-3}, 0, +1)$	$(q_{9,-4}, 1, +1)$	q_I	q_I
$q_{8,-2}$	$(q_{9,-2}, 0, +1)$	$(q_{9,-3}, 1, +1)$	q_I	q_I

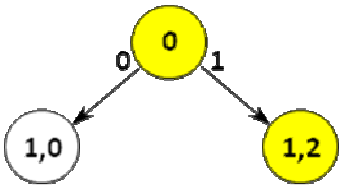
$q_{8,-1}$	$(q_{9,-1}, 0, +1)$	$(q_{9,-2}, 1, +1)$	q_I	q_I
$q_{8,0}$	$(q_{9,0}, 0, +1)$	$(q_{9,-1}, 1, +1)$	q_I	q_I
$q_{8,1}$	$(q_{9,1}, 0, +1)$	$(q_{9,0}, 1, +1)$	q_I	q_I
$q_{8,2}$	$(q_{9,2}, 0, +1)$	$(q_{9,1}, 1, +1)$	q_I	q_I
$q_{8,3}$	$(q_{9,3}, 0, +1)$	$(q_{9,2}, 1, +1)$	q_I	q_I
$q_{8,4}$	$(q_{9,4}, 0, +1)$	$(q_{9,3}, 1, +1)$	q_I	q_I
$q_{8,5}$	$(q_{9,5}, 0, +1)$	$(q_{9,4}, 1, +1)$	q_I	q_I
$q_{8,6}$	$(q_{9,6}, 0, +1)$	$(q_{9,5}, 1, +1)$	q_I	q_I
$q_{9,-7}$	q_I	q_I	q_I	q_Y
$q_{9,-6}$	q_I	q_I	q_I	q_Y
$q_{9,-5}$	q_I	q_I	q_I	q_Y
$q_{9,-4}$	q_I	q_I	q_I	q_Y
$q_{9,-3}$	q_I	q_I	q_I	q_Y
$q_{9,-2}$	q_I	q_I	q_I	q_Y
$q_{9,-1}$	q_I	q_I	q_I	q_Y
$q_{9,0}$	q_I	q_I	q_I	q_Y
$q_{9,1}$	q_I	q_I	q_I	q_N
$q_{9,2}$	q_I	q_I	q_I	q_N
$q_{9,3}$	q_I	q_I	q_I	q_N
$q_{9,4}$	q_I	q_I	q_I	q_N
$q_{9,5}$	q_I	q_I	q_I	q_N
$q_{9,6}$	q_I	q_I	q_I	q_N

Cada transición está formada por una triada de estado, símbolo y movimiento. El estado indica el estado al que se debe mover la MTD, el símbolo es el que la cabeza deberá escribir en la celda actual y el movimiento se representa con +1 si la cabeza se debe mover a la derecha, -1 a la izquierda y 0 si no se debe mover.

Ahora revisaremos cada uno de los pasos que sigue la MTD para determinar si el ejemplar propuesto al inicio (10,11,110b) es un ejemplar-sí o es un ejemplar-no.

Al inicio del proceso de la MTD, la cabeza se posiciona en la celda uno, para iniciar el proceso de transición de estados.

La cabeza lee el símbolo correspondiente a la celda uno, después transfiere el símbolo leído al CEF para determinar la transición correspondiente. Como el primer símbolo leído es "1" y el CEF se encuentra en el estado inicial q_0, entonces el CEF busca la transición para el símbolo "1" y el estado inicial q_0.

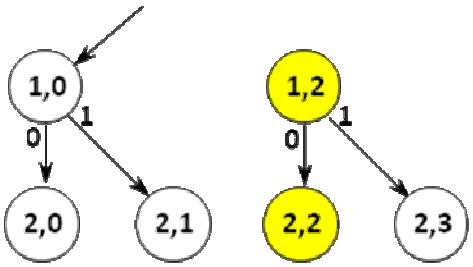


q	0	1	","	b
q_0	$(q_{1,0}, 0, +1)$	$(q_{1,2}, 1, +1)$	q_I	q_I

Figura 4.3.- Función de transición del primer símbolo.

La función de transición para el símbolo "1" indica una transición al estado $q_{1,2}$ (figura 4.3), y manda una orden a la cabeza para que escriba el símbolo uno y que se mueva una celda a la derecha (indicado por el +1 de la regla de transición). En la representación del estado, el primer número del subíndice indica el paso en el que se encuentra la MTD y el segundo número del subíndice indica el valor acumulado de la suma binaria. En este paso, como el primer símbolo es "1", entonces su valor binario es dos por encontrarse en la posición más significativa.

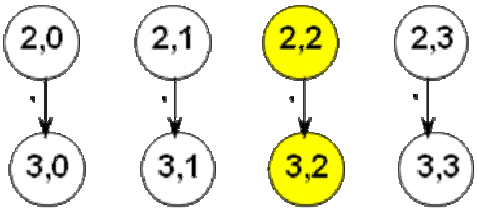
Después de cambiar de estado, la MTD lee el siguiente símbolo, el cual es "0". La función de transición para este estado lleva al estado $q_{2,2}$ (figura 4.4), el cual indica que la MTD se encuentra en el paso 2 con una suma de dos, debido a que el símbolo leído "0" en la posición menos significativa tiene el valor cero, el cual se agrega a la suma que llevábamos, que era de dos. Enseguida la cabeza realiza un movimiento de una celda a la derecha.



$q_{1,2}$	$(q_{2,2}, 0, +1)$	$(q_{2,3}, 1, +1)$	q_I	q_I

Figura 4.4.- Función de transición para el segundo símbolo.

En la tercera celda de la cinta, la MTD lee el separador "," que separa los números binarios. La función de transición para este símbolo se muestra en la figura 4.5.



$q_{2,2}$	ql	ql	$(q_{3,2},",",+1)$	ql

Figura 4.5.- Función de transición para el símbolo ",".

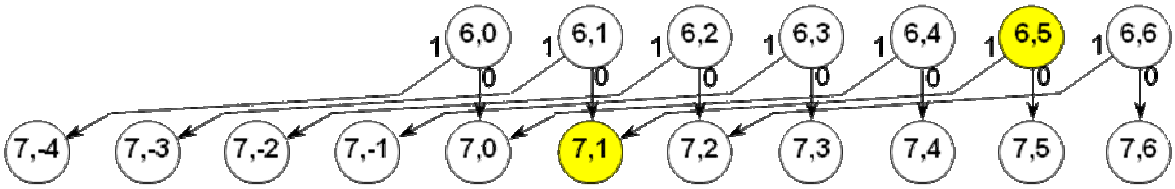
Ahora el CEF pasa al estado $q_{3,2}$, que indica que se encuentra en un estado de espera para leer el siguiente número, y por lo tanto, la suma se mantiene igual.

La MTD realiza la misma operación para los siguientes dos símbolos "1" y "1" del segundo número, por lo que la MTD transita a los estados $q_{4,4}$ y $q_{5,5}$. Como en el segundo número, el primer símbolo leído es "1" y se encuentra en la posición más significativa, se suma dos al acumulado de la suma, dando un valor de cuatro. Y en el último símbolo, en cambio, por encontrarse en la posición menos significativa, su valor es de uno y se suma a los cuatro acumulados, dando como resultado de la suma de los dos números binarios el valor de cinco.

Después de que la MTD realiza la suma, la cabeza se mueve a la siguiente celda que contiene el símbolo de separación ",", el cual lee. Enseguida la MTD pasa al siguiente estado $q_{5',5}$, para después mandar la orden a la cabeza de moverse una celda a la derecha.

En la siguiente parte del programa, se trata de determinar si la suma obtenida de los dos números binarios es menor o igual al tercer número binario, y para esto la MTD realiza una resta entre la suma y el número binario para determinar si el ejemplar en cuestión es un ejemplar-sí o ejemplar-no.

Para el ejemplar en cuestión, después de procesar la celda del último separador (“,”), la cabeza se mueve a la siguiente celda que contiene el símbolo “1” en la posición más significativa del número binario.



$q_{6,5}$	$(q_{7,5}, 0, +1)$	$(q_{7,1}, 1, +1)$	q_I	q_I

Figura 4.6.- Función de transición que resta cuatro a la suma de cinco.

La función de transición lleva al estado $q_{7,1}$ (figura 4.6), el cual indica que el programa está en el paso 6 con un valor acumulado de uno debido a que se le resta cuatro a la suma de los dos números binarios. Luego el CEF ordena a la cabeza que escriba el símbolo “1” en la celda actual y después se mueva una celda hacia la derecha.

El proceso se repite para los pasos 7 y 8 en donde, debido a la función de transición, la MTD pasa por los estados $q_{7,-1}$ y $q_{8,-1}$, después de lo cual el valor acumulado es –1. Este valor significa que la suma de los dos primeros números es menor que el tercer número, por lo cual el ejemplar en cuestión es un ejemplar-sí, y la MTD termina en el estado q_Y.

Habiendo adquirido con este ejemplo una comprensión más amplia de lo que es una máquina de Turing determinista (MTD), se procede con la explicación de una máquina de Turing no determinista (MTND). Para tal efecto, se explican en la siguiente subsección los conceptos de autómata finito no determinista, así como módulo de adivinación y módulo de verificación, los cuales conforman la MTND.

4.1 Máquina de Turing no determinista (MTND)

Una máquina de Turing no determinista es un modelo teórico que contiene: una cinta infinita, una cabeza lectora-escritora (la cual se mueve a través de la cinta), una cabeza escritora que escriba los símbolos provenientes del módulo de adivinación, un módulo de adivinación y un mecanismo de control de estados finitos. La MTND a diferencia de la MTD contiene un módulo de adivinación y un módulo de verificación.

Es importante tener claro el concepto de autómata finito no determinista (AFND). Un AFND es un autómata finito, en el que para un estado en el que se encuentre el autómata y un símbolo leído, existe una o más transiciones, desde ese estado y con ese símbolo.

Con el fin de comprender cómo debe funcionar un simulador de una MTND, es necesario conocer a detalle un ejemplo de un AFND para el módulo adivinación de la MTND y un ejemplo de un autómata finito determinista (AFD) para el módulo de verificación del la MTND. Para tal efecto, una buena parte de esta subsección se dedica a describir un ejemplo de AFND y de AFD para el problema del Circuito Hamiltoniano.

4.1.1 Módulo de adivinación

Como ya se mencionó, la máquina de Turing no determinista está compuesta por un módulo de adivinación y un módulo de verificación.

El módulo de adivinación es implementado por un AFND y genera aleatoriamente una solución candidata del problema de decisión. Este módulo cuenta con una cabeza escritora, la cual no puede leer. En tales circunstancias, para que este módulo pueda realizar transiciones, se recurre al artificio de usar un símbolo de entrada ε, el cual permite a cualquier autómata realizar una transición espontánea sin leer ningún símbolo de la cinta.

La cabeza escritora se encarga de grabar en el lado izquierdo de la cinta, los símbolos de la solución candidata (s_1, s_2, s_3,...) tal como se ilustra en la figura 4.7. Nota: en esta figura el símbolo "|" que se encuentra en la celda 0 de la cinta se usa como elemento de referencia para facilitar el desplazamiento de la cabeza a cualquier posición en la cinta.

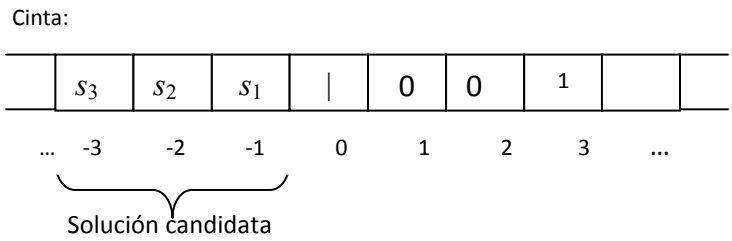


Figura 4.7.- Solución candidata generada por el módulo de adivinación.

4.1.2 Módulo de verificación

El módulo de verificación, a semejanza de una MTD, se implementa mediante un AFD. Este módulo se encarga de verificar si la solución candidata, grabada en la parte izquierda de la cinta, es realmente una solución del problema codificado, grabado en la parte derecha de la cinta (ver figura 4.8). Nota: en esta figura el símbolo "|", que se encuentra en la celda 0 de la cinta, se usa como elemento de referencia para ubicar la celda 0 y facilitar el desplazamiento de la cabeza a cualquier posición en la cinta.

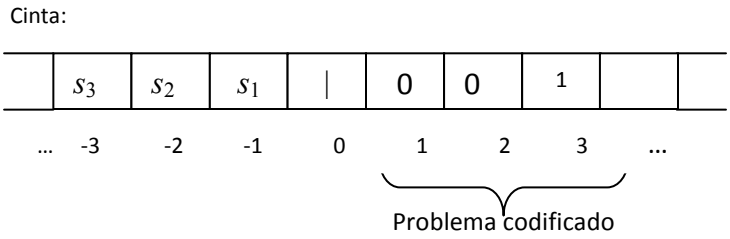


Figura 4.8.- Contenido de la cinta que debe procesar el módulo de verificación.

4.2 Diseño del autómata del problema del Circuito Hamiltoniano

Uno de los problemas que se abordaron para diseñar y simular una MTND en este proyecto, es el problema del Circuito Hamiltoniano, el cual se describe a continuación. (Nota: el diseño se realizó en base a la definición de MTND dada en [Garey & Johnson, 1979])

Un circuito hamiltoniano es un circuito que pasa por todos los vértices de un grafo una sola vez, con excepción del vértice inicial, en el cual comienza y termina el circuito.

Ejemplar: un grafo $G = (V, E)$.

Pregunta: ¿G contiene un circuito hamiltoniano, es decir, una secuencia ordenada $<v_1, v_2, \ldots, v_n>$ de los vértices de G donde $n = |V|$ y de modo que: $\{v_n, v_1\} \in E$ y $\{v_i, v_{i+1}\} \in E \; \forall i, 1 \leq i < n$?

El objetivo de este ejemplo es diseñar una MTND para determinar si hay un circuito hamiltoniano en el grafo de la figura 4.9.

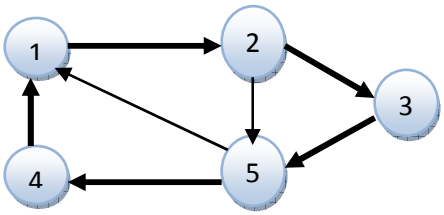


Figura 4.9.- Grafo del problema a resolver por una MTND.

Para resolver este problema con una MTND, es necesario codificar primero el problema, y para tal efecto se usa la matriz de adyacencia del grafo, la cual se muestra en la figura 4.10.

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	0	0
v_2	0	0	1	0	1
v_3	0	0	0	0	1
v_4	1	0	0	0	0
v_5	1	0	0	1	0

Figura 4.10.- Matriz de adyacencia del problema del grafo de la figura 4.9.

Tomando como base la matriz de adyacencia, el problema se puede codificar en la cinta de la manera que se muestra en la Figura 4.11.

b	0	1	0	0	0	*b*	0	0	1	0	1	*b*	0	0	0	0	1	*b*	1	0	0	0	0	*b*	1	0	0	1	0
1	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	29	30

Figura 4.11.- Problema codificado en la cinta.

4.2.1 Módulo de adivinación

El módulo de adivinación selecciona aleatoriamente del conjunto {1, 2,…, 5} una secuencia de cinco dígitos correspondientes a los vértices del grafo. Esta secuencia constituirá la solución candidata del problema. El módulo emplea un autómata finito no determinista, el cual se ilustra en la figura 4.12.

El AFND emplea la notación $s_1 \rightarrow s_2$, m para representar las acciones de lectura de un símbolo s_1, escritura de un símbolo s_2 y movimiento de la cabeza.

1.- s_1 representa el símbolo que lee el módulo de adivinanza en la celda actual. Para la operación de lectura se usa el símbolo $s_1=\varepsilon$ para indicar que la cabeza no efectúa lectura alguna; es decir, efectúa una transición espontáneamente.

2.- s_2 representa el símbolo que la cabeza escribirá en la celda actual. Para la operación de escritura se usa el símbolo $s_2=\varnothing$ para indicar que la cabeza no efectúa escritura alguna.

3.- *m* representa el movimiento de la cabeza. Si *m* = +1, se mueve una celda a la derecha, si *m* = –1, se mueve una celda a la izquierda, si *m* = 0 entonces no realiza movimiento, es decir, se mantiene en la misma celda.

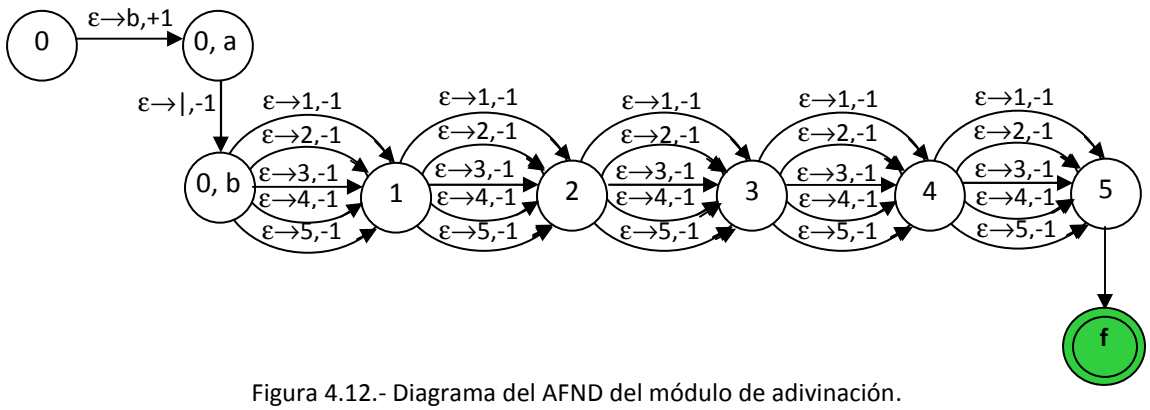


Figura 4.12.- Diagrama del AFND del módulo de adivinación.

La cabeza lectora del módulo de adivinación se ubica inicialmente en la posición cero de la cinta y escribe el símbolo "|", el cual se utiliza como elemento de referencia para facilitar el proceso del programa verificador.

En la figura 4.12 se observa cómo la transición del estado $q_{0,a}$ al estado $q_{0,b}$ ordena a la cabeza no leer nada, escribir el símbolo "|" y moverse una celda a la izquierda. Esta transición ocurre cuando la MTND se encuentra en la celda cero de la cinta.

b	*b*	*b*	#	4	5	3	2	1	|	*b*	0	1	0	0	0	*b*	0	0	1	0	1	.	.	.	1	0	0	1	0
				-5					0					5					10										30

Figura 4.13.- Contenido de la cinta al finalizar el módulo de adivinación.

Al ubicarse en la posición –1 de la cinta, la cabeza escribe aleatoriamente un símbolo que corresponde a un vértice del grafo. Esto está representado en el AFND en la transición del estado $q_{0,b}$ al estado q_1 mediante cinco posibles transiciones, de las cuales en ninguna ordena a la cabeza leer el símbolo en la celda –1, y solamente escribe. Cada una de las posibles transiciones escribe un símbolo (que puede ser igual o diferente de los escritos anteriormente) que represente un vértice del grafo, dando así la cualidad de aleatoriedad al AFND. Por último, se le ordena a la cabeza moverse una celda a la izquierda. Esto se repite durante la transición del estado q_1 al q_2, de éste al q_3, y así sucesivamente hasta el estado q_5, con lo cual se escriben cinco símbolos (que representan cinco vértices) en las celdas que van desde la –1 a la –5. La ultima transición del estado q_5 al estado q_f indica la finalización del proceso de adivinación de una solución candidata, y ocasiona la escritura del símbolo "#" en la celda –6 para delimitar la solución y facilitar el proceso de verificación.

4.2.2 El módulo de verificación

El módulo de verificación debe realizar las siguientes comprobaciones:

1. Comprobar que el ejemplar del problema es válido.
2. Comprobar que la secuencia de vértices generados no tenga elementos repetidos.
3. Comprobar que para cada par de vértices consecutivos (v_i, v_{i+1}) de la secuencia se cumple que (v_i, v_{i+1}) sea una arista de G y que (v_1, v_5) sea también una arista de G.

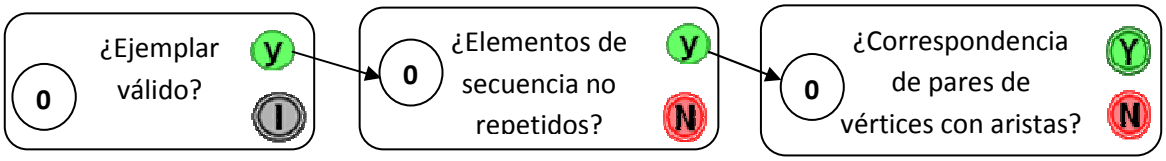


Figura 4.14.- Diagrama de submódulos de las tareas del módulo de verificación.

4.2.2.1 ¿Ejemplar válido?

El siguiente pseudocódigo tiene como objetivo validar el ejemplar del problema.

Definir i = cinta(−7)

1. Hacer i = 1.
2. Si la fila i válida, pasar al paso 3, de lo contrario parar en el estado q_l.
3. Hacer $i = i + 1$.
4. Si $i \leq 5$ ir al paso 2; si i = 6 ir al estado “y”.

Para este submódulo es necesario definir la variable i que tendrá la función de contador, el cual aumentará cada vez que se revise la validez de una fila de la matriz de adyacencia. El valor de i será almacenado en la celda −7 de la cinta, lo cual se muestra al inicio del pseudocódigo.

Validación de ejemplar: Paso 1

Ya que durante el proceso de revisión de las filas de la matriz de adyacencia será necesario incrementar la variable i, se creó un submódulo que posiciona la cabeza de la MTND en la celda −7 para poder actualizarla. La figura 4.15 muestra el submódulo antes mencionado.

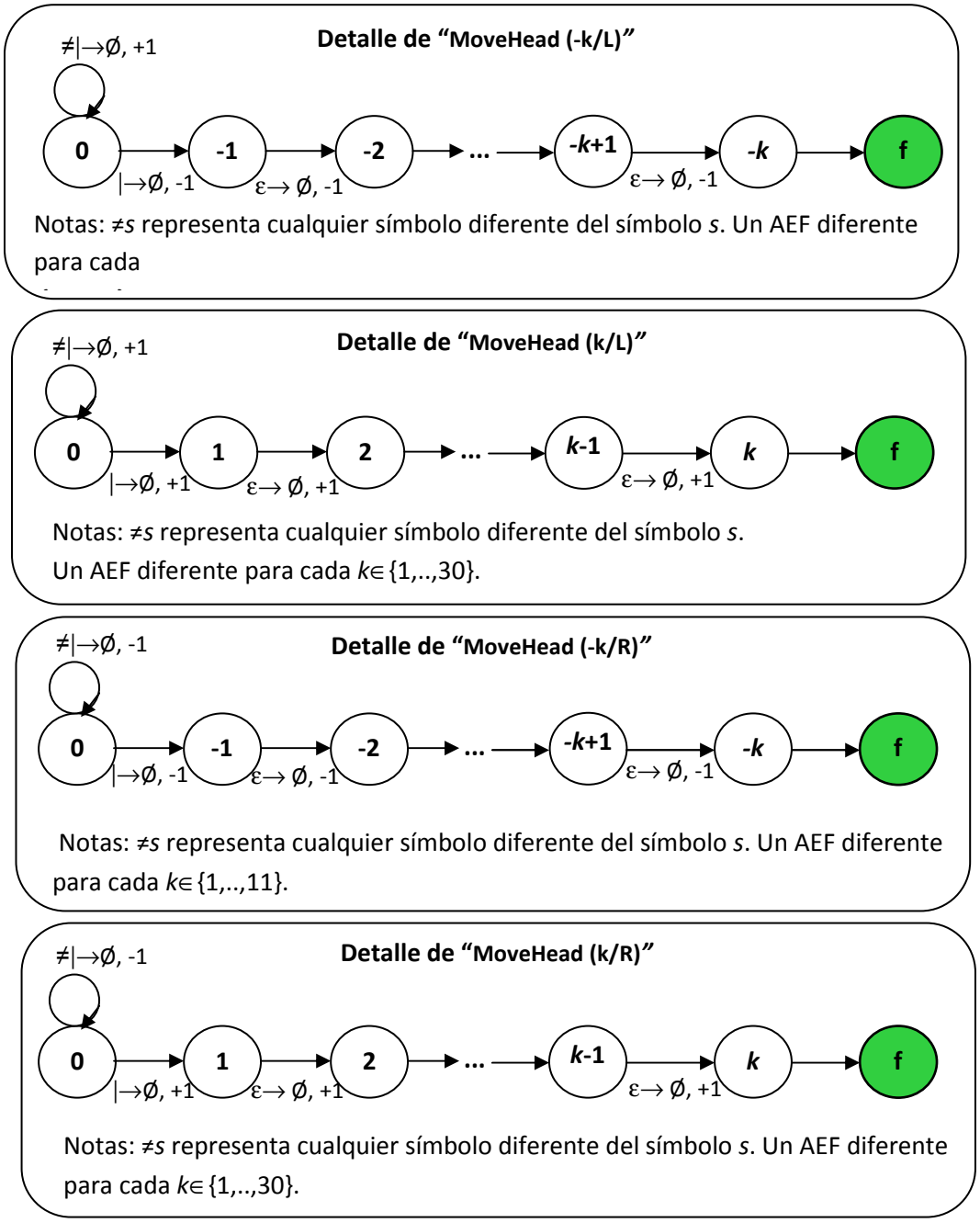


Figura 4.15.- Diagramas de los submódulos que posicionan la cabeza en cualquier celda de la cinta.

En la figura 4.15 se describen cuatro submódulos que se usan frecuentemente para posicionar la cabeza en cualquier celda de la cinta. El primer submódulo supone que la cabeza se encuentra en una celda menor a cero, por lo que primero se mueve la cabeza hacia la derecha mientras no se encuentre el símbolo "|" que está ubicado en la celda cero, y después de llegar a ésta, se mueve la cabeza hacia la izquierda (sin leer ni escribir símbolos) hasta que llega a la celda deseada, que en nuestro caso sería la celda $-k=-7$. El segundo submódulo realiza la misma función, sólo que una

vez encontrado el símbolo "|", se mueve la cabeza hacia la derecha para encontrar una celda específica *k* positiva. El tercer submódulo corresponde al submódulo que posiciona la cabeza en una celda negativa partiendo de una celda positiva por lo que, a diferencia de los dos submódulos anteriores, se mueve la cabeza hacia la izquierda para encontrar la celda cero con el símbolo "|", y de ésta se mueve la cabeza hasta la celda negativa $-k$. El cuarto submódulo posiciona la cabeza en una celda positiva partiendo de una celda positiva, por lo que a diferencia de la primera, la cabeza se moverá hacia la izquierda para encontrar la celda cero con el símbolo "|", y después hacia la derecha para encontrar la celda *k* positiva.

Con ayuda de los submódulos de posicionamiento de la figura 4.15 y en referencia al paso 1 del pseudocódigo de ejemplar válido, la cabeza lectora se posiciona en la celda -7, y en esa celda correspondiente a la variable *i*, se escribe un 1, quedando la cabeza sin hacer algún otro movimiento (ver figura 4.16). (Nota: los estados con etiqueta AN indican que el proceso continúa en el paso N; por ejemplo, para el caso de la Figura 4.16 el estado con etiqueta A2 indica que el proceso continúa en el paso 2.)

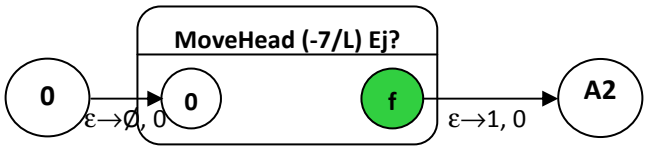


Figura 4.16.- Submódulo del paso 1: Hacer *i* = 1.

Validación de ejemplar: Paso 2

Primeramente, la cabeza se mueve hacia la celda -7 para leer el valor de la variable *i*. Después, la cabeza se mueve al inicio de la fila *i* (de la matriz de adyacencia) para empezar a validar la fila.

La figura 4.17 muestra el proceso que se realiza para posicionar la cabeza en el inicio de la fila de la matriz de adyacencia a validar. Primero se ubica en la celda -7 para leer el valor de *i*. Después la cabeza se ubica en el inicio de la fila *i*, de tal manera que si *i*=1 la cabeza se ubica en la posición 1, si *i*=2 se ubica en la posición 7, y así sucesivamente como se muestra en la figura. Enseguida se ejecuta el submódulo de verificación de fila, en el cual se determina si la fila es válida o no. Si la fila es inválida, llega a un estado q_l y termina el programa, y en caso de que la fila sea válida, posiciona la cabeza de nueva cuenta en la celda -7 para actualizar el valor de *i* y pasar a la siguiente fila.

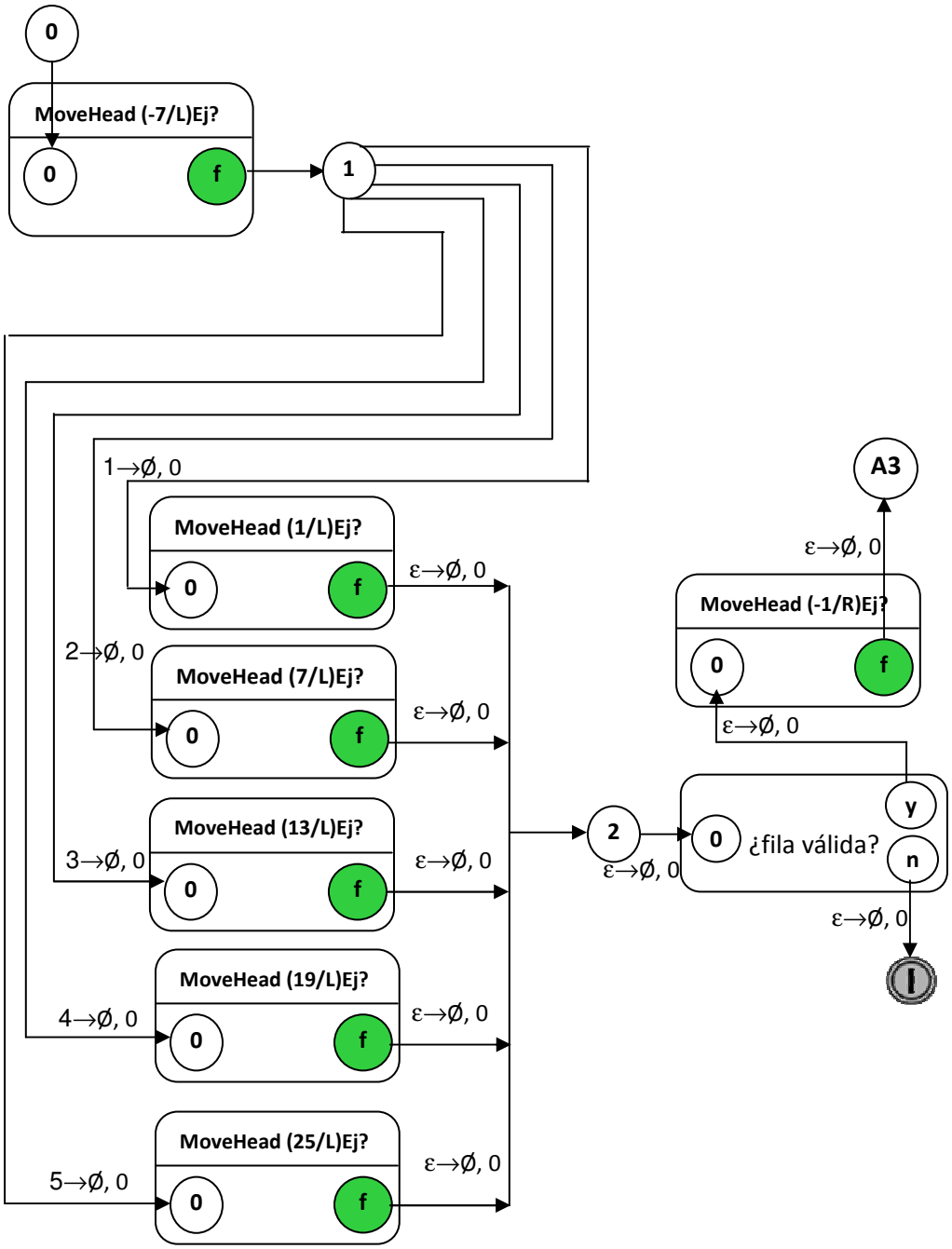


Figura 4.17.- Submódulo del paso 2: Si la fila *i* válida, pasar al paso 3, de lo contrario parar en el estado q_l.

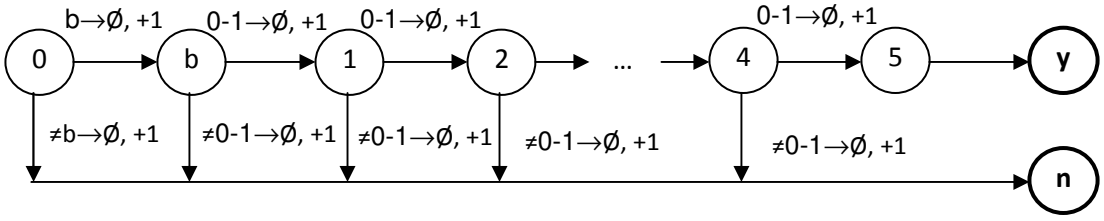


Figura 4.18.- Submódulo de verificación de fila.

En la figura 4.18 se muestra el submódulo de verificación de la fila *i*, en donde si el símbolo leído en la primera celda de la fila es diferente de "*b*" entonces la fila es inválida y el submódulo pasa al estado de negación q_n; en caso contrario el proceso de revisión del resto de las celdas continúa, y la cabeza se mueve a la siguiente celda. Si en las siguientes cinco celdas se lee un carácter diferente a "0" o "1", el submódulo pasa al estado de negación q_n y más adelante el programa termina en el estado q_l, en caso contrario el submódulo continúa y pasa al estado de aceptación local q_y. Nota: s_1- s_2 denota cualquier símbolo comprendido en el rango de s_1 a s_2; así que, $\neq s_1$- s_2 denota cualquier símbolo fuera del rango de s_1 a s_2.

Validación de ejemplar: Paso 3

Para aumentar el valor de *i* primero se posiciona la cabeza en la celda –7, y se efectúa una de cinco posibles transiciones para cambiar el valor de la variable *i*. La figura 4.19 muestra el submódulo de incremento de la variable *i*.

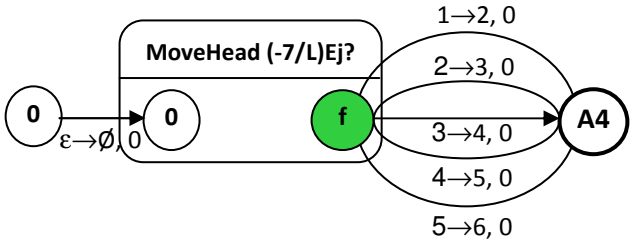


Figura 4.19.- Submódulo del paso 3: Hacer *i* = *i* + 1.

En la figura 4.19, la función de transición primero lee el valor de la celda –7, después escribe en la misma celda el valor leído más 1. Por ejemplo, si lee un "1", escribe un "2".

Validación de ejemplar: Paso 4

En el paso 4 primero es necesario mover la cabeza a la celda –7, si $i \leq 5$ el proceso continúa en el paso 2, de lo contrario el submódulo termina en el estado de aceptación local q_f y más adelante pasa al estado "y" (figura 4.20).

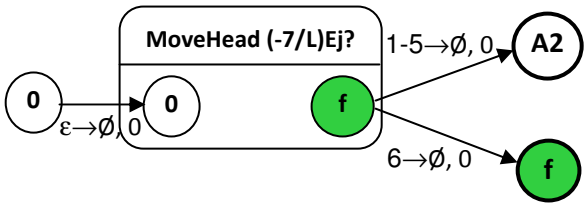


Figura 4.20.- Submódulo del paso 4: Si $i \leq 5$ ir al paso 2; si $i = 6$ ir al estado "y".

4.2.2.2 ¿Elementos de secuencia no repetidos?

El siguiente es el pseudocódigo del submódulo de elementos no repetidos.

Definir variables i = cinta(–7), j = cinta(–8), v_i = cinta(–9), v_j = cinta(–10), oc = cinta(–11). (Nota: oc representa el número de ocurrencias de un vértice en la secuencia aleatoria de vértices.)

1. Hacer $i = 1$.
2. Hacer $oc = 0$.
3. Hacer $j = 1$.
4. Copiar valor de cinta(–i) a v_i.
5. Copiar valor de cinta(–j) a v_j.
6. Si $v_i = v_j$, hacer $oc = oc + 1$ y si oc adquiere el valor 2, parar en el estado "N".
7. Hacer $j = j + 1$.
8. Si $j \leq 5$ ir al paso 5; si $j = 6$ hacer $i = i + 1$.
9. Si $i \leq 5$ ir al paso 2; si $i = 6$ ir al estado "y".

Elementos no repetidos: Paso 1

En la figura 4.21 podemos observar que se inicializa la variable i (en la celda –7) con el valor de 1. (Nota: los estados con etiqueta BN indican que el proceso continúa en el paso N; por ejemplo, para el caso de la Figura 4.21 el estado con etiqueta B2 indica que el proceso continúa en el paso 2.)

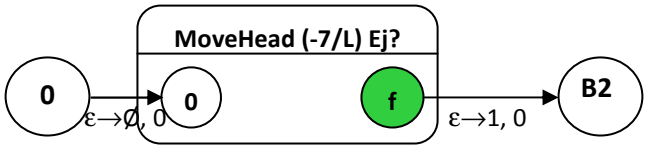


Figura 4.21.- Submódulo del paso 1: Hacer $i = 1$.

Elementos no repetidos: Paso 2

En la figura 4.22 podemos observar que se inicializa la variable *oc* (en la celda –11) con el valor de 0.

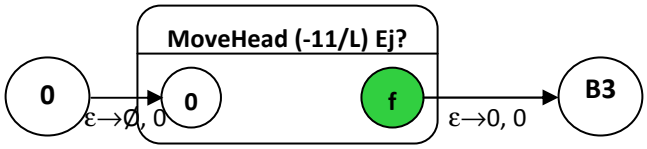


Figura 4.22.- Submódulo del paso 2: Hacer *oc* = 0.

Elementos no repetidos: Paso 3

En la figura 4.23 podemos observar que se inicializa la variable *j* (en la celda –8) con el valor de 1.

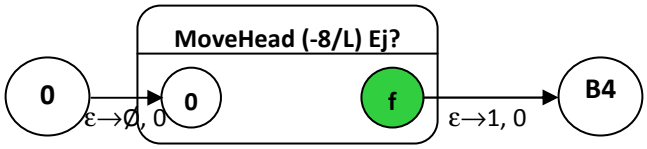


Figura 4.23.- Submódulo del paso 3: Hacer *j* = 1.

Elementos no repetidos: Pasos 4 y 5

Después de inicializar las variables, se copia el valor almacenado en las celdas –*i* y –*j* a las variables v_i, v_j que corresponden a las celdas –9 y –10 de la cinta. La figura 4.24 muestra el proceso del paso 4, el cual se explica a continuación:

- la cabeza primero se ubica en la celda de la variable *i* (celda –7),
- enseguida la cabeza lee el valor de *i*,
- después la cabeza se mueve a la celda –*i* (en donde *i* representa el valor leído en la celda –7),
- enseguida la cabeza lee el valor almacenado en la celda –*i*, y
- finalmente, copia este valor en la celda correspondiente a la variable v_i (celda –9).

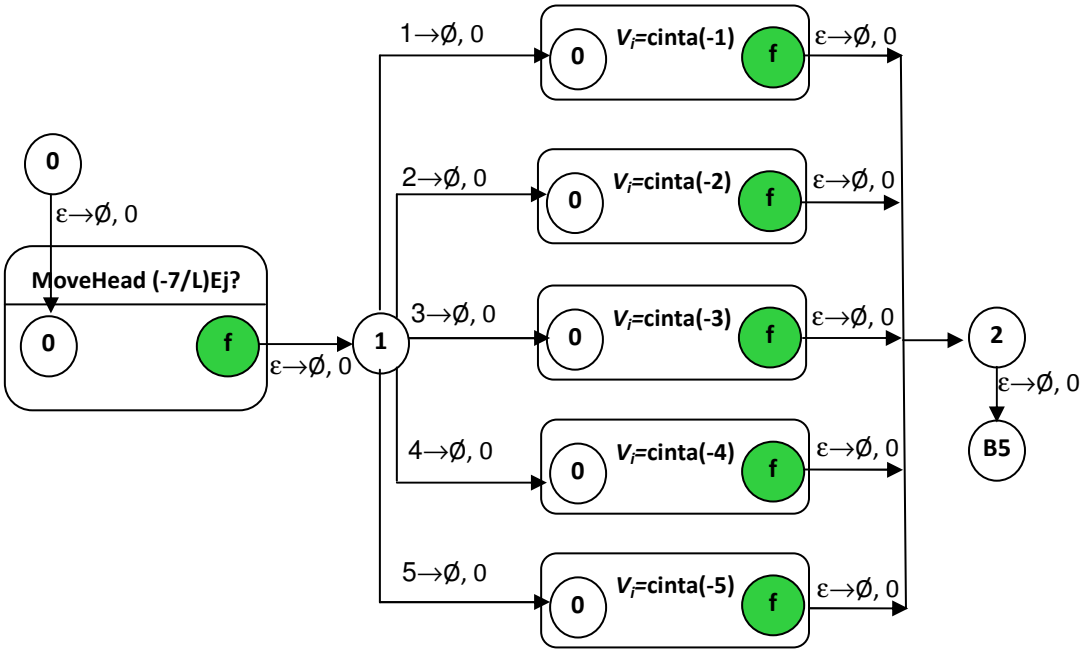


Figura 4.24.- Submódulo del paso 4: Copiar valor de cinta(–*i*) a v_i.

El submódulo del paso 5 realiza un proceso semejante al del paso 4 y está descrito en la figura 4.25.

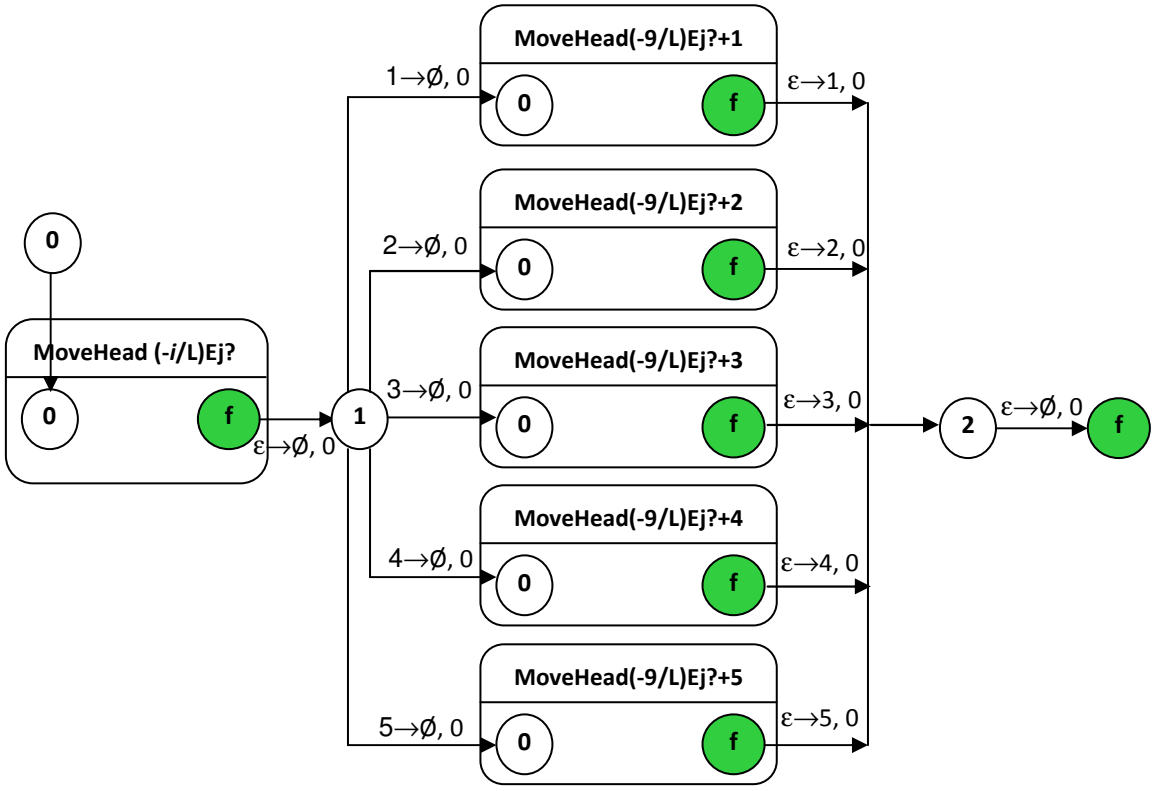


Figura 4.25.- Submódulo del paso 5: Copiar valor de cinta(–*j*) a v_j.

Elementos no repetidos: Paso 6

El paso 6 está descrito en la figura 4.26, el cual involucra varios submódulos. Primero hay un submódulo que tiene como función la de comprobar si $v_i = v_j$ y tiene dos estados de finalización, un estado sí (y) y un estado no (n). Si termina en el estado sí, aumenta en uno el valor de la variable *oc* y si este valor es igual a uno, continúa el proceso hacia el paso 7, pero si es igual a dos, para en un estado de negación "q_N". Si v_i y v_j son diferentes, pasa directamente al paso 7.

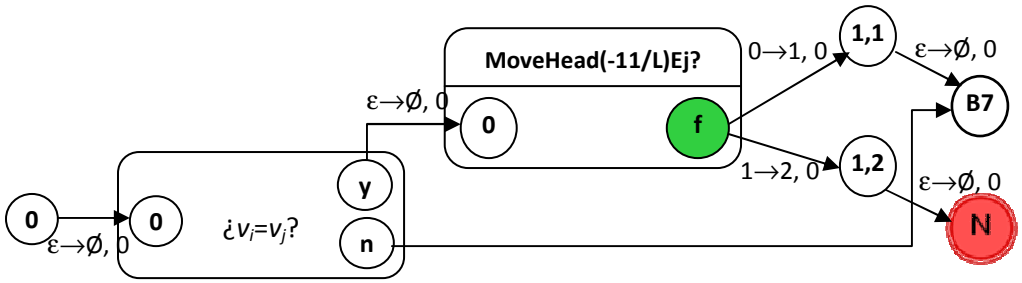


Figura 4.26.- Submódulo del paso 6: Si $v_i = v_j$, hacer *oc* = *oc* + 1 y si *oc* adquiere el valor 2, parar en el estado "N".

La figura 4.27 describe el proceso para comprobar si $v_i = v_j$. Para tal efecto, primero se lee el valor de la variable v_i en la celda –9 para luego mover la cabeza hacia la celda –10 en donde se lee el valor de la variable v_j para verificar si es igual al valor de la variable v_i. De ser iguales el submódulo llega al estado "q_y" que indica que sí son iguales, de lo contrario llega al estado "q_n".

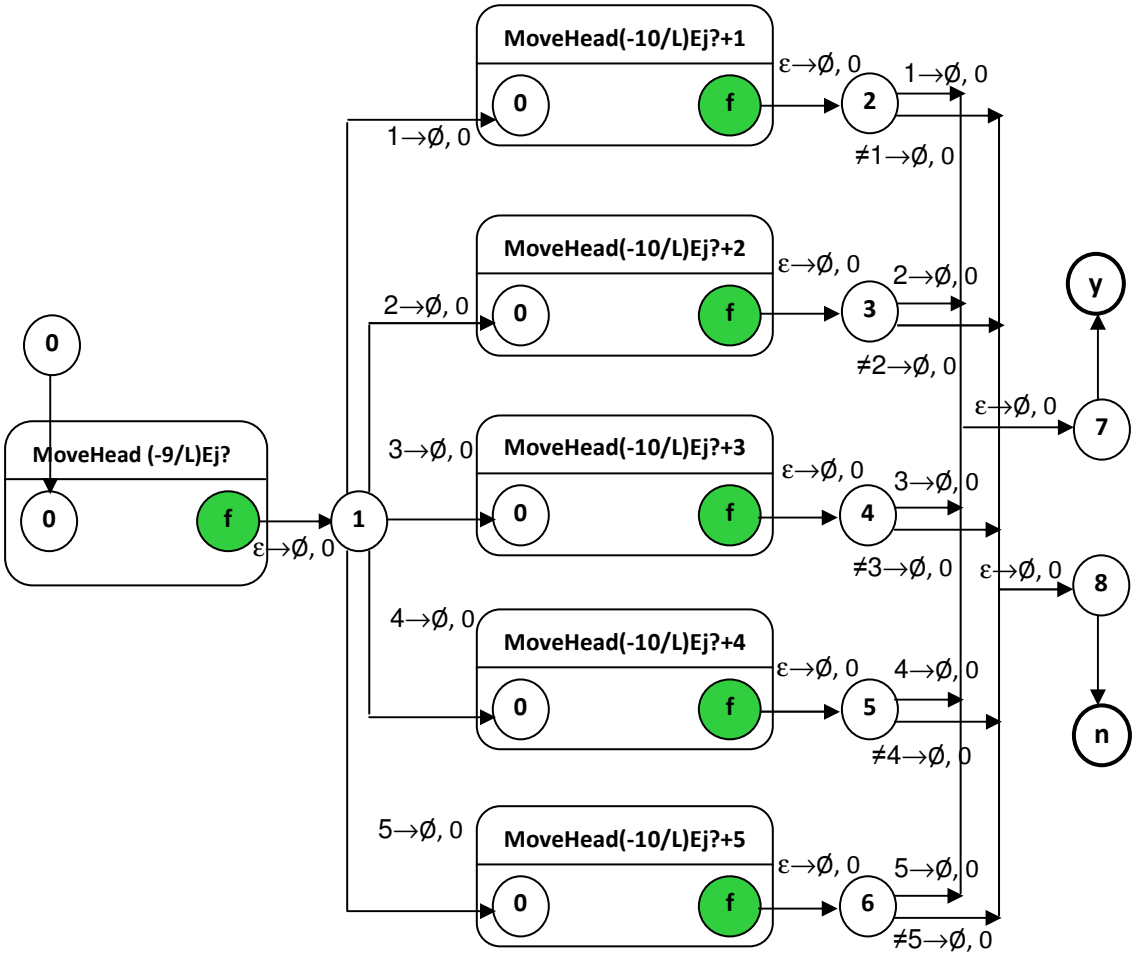


Figura 4.27.- Submódulo ¿$v_i = v_j$?

Elementos no repetidos: Paso 7

Para realizar el paso 7, se ubica la cabeza en la celda de la variable *j* (–8), y dependiendo del valor leído, se escribe el valor leído más 1 (*j* + 1). La figura 4.28 muestra este proceso.

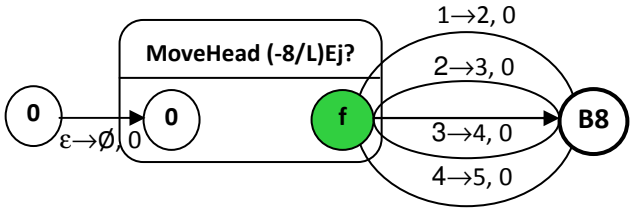


Figura 4.28.- Submódulo del paso 7: Hacer *j* = *j* + 1.

Elementos no repetidos: Paso 8

Para el paso 8 es necesario verificar si $j \leq 5$ y para esto se desarrolló un submódulo, en el cual primero se ubica la cabeza en la celda correspondiente a la variable j (–8), y después la cabeza lee el valor almacenado. Si el valor almacenado está en el rango 1 a 5, regresa al paso 5, de lo contrario, la cabeza se mueve hacia la celda de la variable i (–7) y aumenta su valor a $i + 1$. La figura 4.29 muestra este proceso.

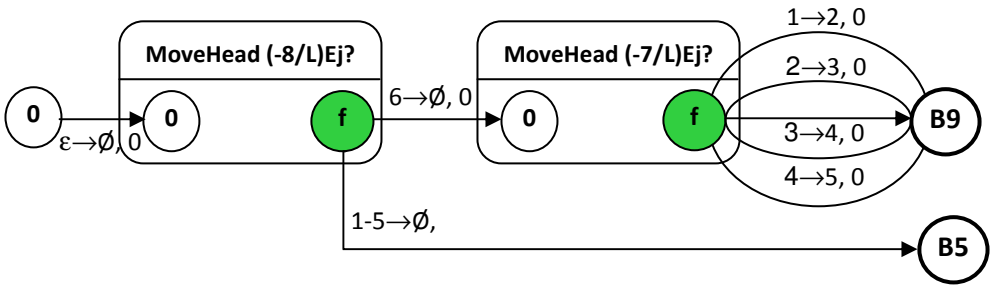


Figura 4.29.- Submódulo del paso 8: Si $j \leq 5$ ir al paso 5; si $j = 6$ hacer $i = i + 1$.

Elementos no repetidos: Paso 9

En el paso 9 se verifica si $i \leq 5$, y para esto se desarrolló un submódulo, en el cual primero se ubica la cabeza en la celda de la variable i (–7), y si el valor leído esta dentro del rango 1 a 5, regresa al paso 2, y en caso contrario, el submódulo termina en el estado de aceptación "q_f", lo cual significa que no hay vértices repetidos en la secuencia aleatoria.

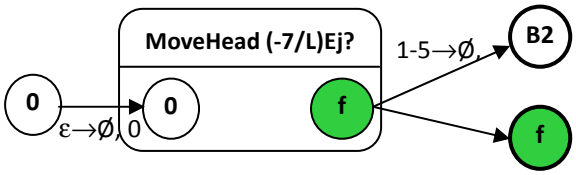


Figura 4.30.- Submódulo del paso 9: Si $i \leq 5$ ir al paso 2; si $i = 6$ ir al estado "y".

4.2.2.3 ¿Correspondencia entre pares de vértices con aristas?

Pseudocódigo del submódulo correspondencia de pares de vértices con aristas.

Definir i = cinta(–8), $j = i + 1$, $fila_1 = 1$, $fila_2 = 7$, $fila_3 = 13$, $fila_4 = 19$, $fila_5 = 25$.

1. Copiar valor de cinta(–1) a cinta(–6).
2. Hacer cinta(–7)=#. (Nota: este paso no es indispensable).
3. Hacer i=1.
4. Copiar valor de cinta(–j) a cinta($fila_1$), cinta($fila_2$), cinta($fila_3$), cinta($fila_4$), cinta($fila_5$).

5. Ir a posición $fila_i$ e ir cinta($fila_i$) posiciones a la derecha; si el contenido es 1 ir al paso 6 si no parar en el estado "N".
6. Hacer i=i+1.
7. Si i≤5 ir al paso 4; si i=6 parar en el estado "Y".

¿El par de vértices es arista?: Paso 1

En el paso 1 se copia el primer vértice de la secuencia aleatoria de vértices (es decir, el valor de la celda –1) al final de la secuencia (es decir, a la celda –6), y para esto se mueve la cabeza a la celda –1, para leer el valor y grabarlo en la celda –6). Este paso está descrito en la figura 4.31. (Nota: los estados con etiqueta CN indican que el proceso continúa en el paso N; por ejemplo, para el caso de la Figura 4.31 el estado con etiqueta C2 indica que el proceso continúa en el paso 2.)

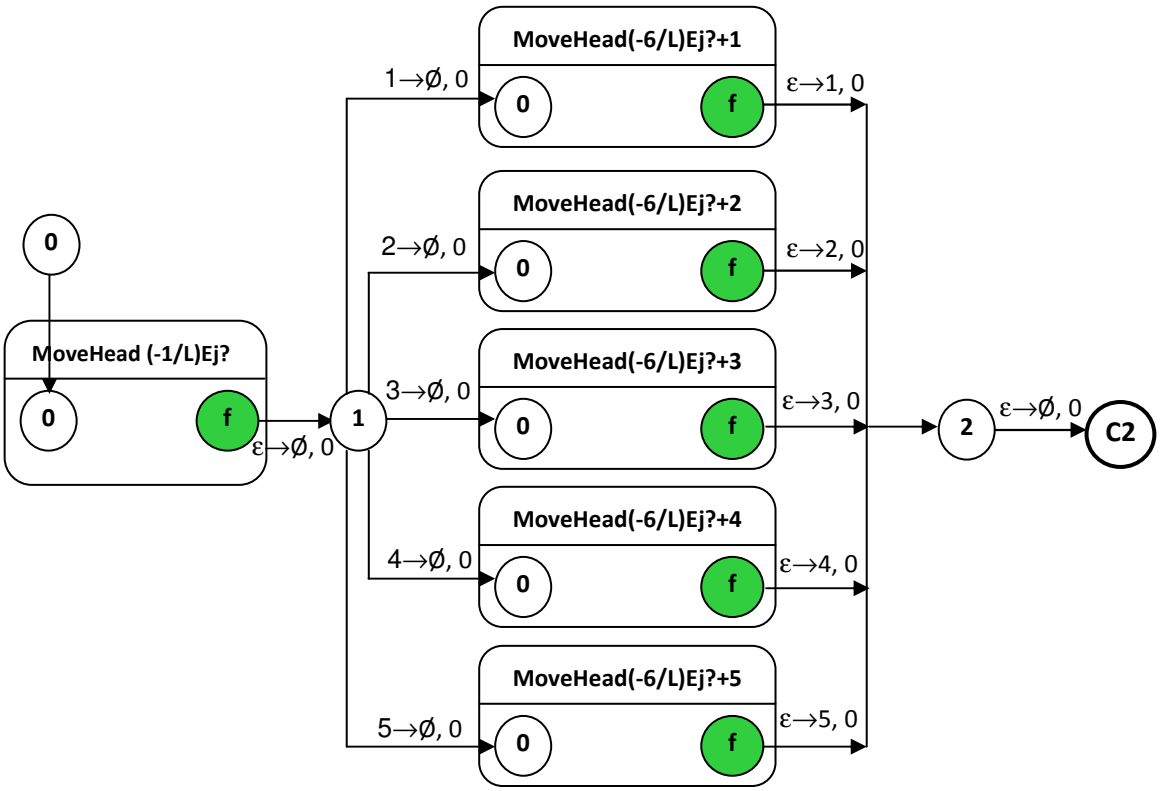


Figura 4.31.- Submódulo del paso 1: Copiar valor de cinta(–1) a cinta(–6).

¿El par de vértices es arista?: Paso 2

En el paso 2 se graba el símbolo "#" en la celda –7. La figura 4.32 muestra este proceso.

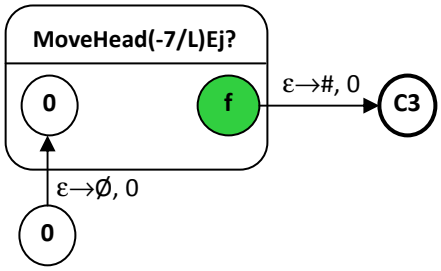


Figura 4.32.- Submódulo del paso 2: Hacer cinta(–7)=#.

¿El par de vértices es arista?: Paso 3

En el paso 3 se inicializa la variable i al escribir el símbolo "1" en la celda –8. Este paso está descrito en la figura 4.33.

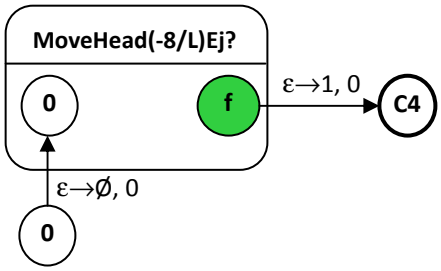


Figura 4.33.- Submódulo del paso 3: Hacer *i*=1.

¿El par de vértices es arista?: Paso 4

En este paso se realiza el copiado del valor de la celda *–j* a las celdas ($fila_1$), ($fila_2$), ($fila_3$), ($fila_4$) y ($fila_5$). Para tal efecto, primero se mueve la cabeza a la posición de la variable *j* (–8) y enseguida se lee el valor almacenado en esta posición, después se mueve la cabeza a la celda *–j* para leer su valor y copiarlo en las celdas (1), (7), (13), (19) y (26) correspondientes al valor de las variables $fila_i$. La figura 4.34 muestra este proceso.

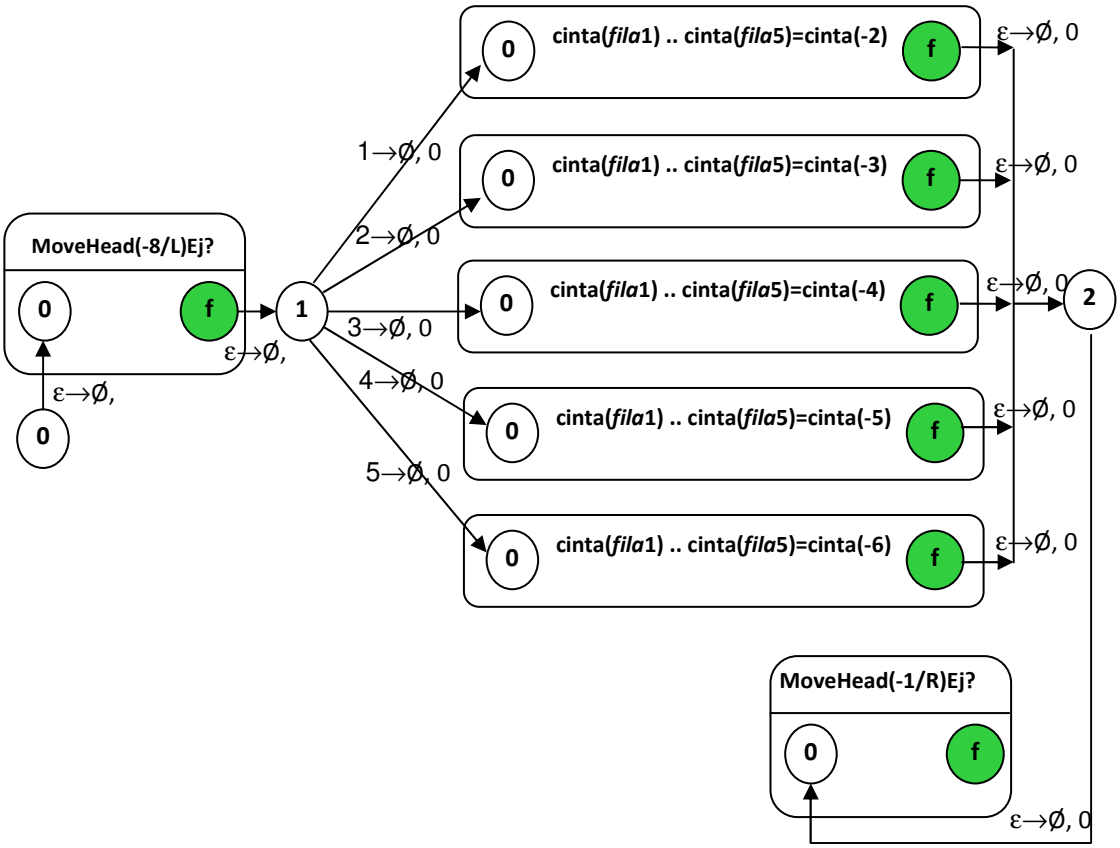


Figura 4.34.- Submódulo del paso 4: Copiar valor de cinta(–*j*) a cinta(*fila*$_1$), cinta(*fila*$_2$), cinta(*fila*$_3$), cinta(*fila*$_4$), cinta(*fila*$_5$).

En la figura 4.35, los submódulos "cinta(*fila*1) .. cinta(*fila*5)=cinta(-*j*)" tienen la función de moverse hacia la celda –*j* para leer su símbolo y copiarlo en las celdas (1), (7), (13), (19) y (26). La figura 4.35 muestra la descripción de este submódulo.

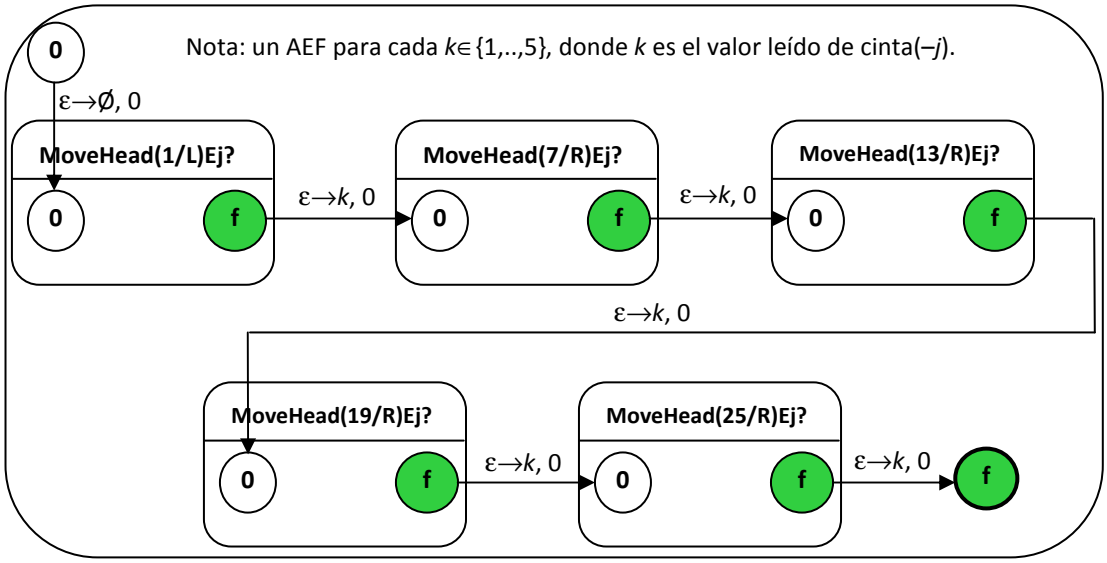


Figura 4.35.- Submódulo "cinta(*fila*1) .. cinta(*fila*5)=cinta(-*j*)".

¿El par de vértices es arista?: Paso 5

El proceso del paso 5, cuyo diagrama se muestra en la figura 4.36, incluye otros tres submódulos, los cuales están descritos en las figuras 4.37, 4.38 y 4.39). El proceso del paso 5 se describe a continuación:

- primero la cabeza se posiciona en la celda de la variable *i* (celda –8), enseguida la cabeza lee el valor de *i*, y después la cabeza se mueve a la celda *–i* (en donde *i* representa el valor leído en la celda –8) tal como se ilustra en la figura 4.37 (nota: es conveniente mencionar que en estas celdas (–1, –2, ..., –5) se encuentran almacenados los vértices v_1, v_2, ..., v_5 de la solución candidata);
- a continuación la cabeza lee el valor de la celda *–i* (cuyo valor es igual al *i*-ésimo vértice v_i de la solución candidata), y después la cabeza se mueve a la posición 1 (si v_i=1), a la posición 7 (si v_i=2), a 13 (si v_i=3), a 19 (si v_i=4), o a 25 (si v_i=5) tal como se muestra en la figura 4. 37; es decir, se mueve al inicio de la fila v_i de la matriz de adyacencia del grafo;
- después la cabeza lee el valor almacenado en la celda adecuada (1, 7, 13, 19 o 25) tal como se ilustra en la figura 4.38, el cual fue grabado en paso 4 y cuyo valor es igual al (*i*+1)-ésimo vértice v_{i+1} de la solución candidata, y a continuación la cabeza se mueve a la derecha un número de posiciones igual a v_{i+1} tal como se ve en las figuras 4.38 y 4.39 (nota: es importante mencionar que la cabeza ahora se encuentra en una celda que corresponde a la fila v_i y la columna v_{i+1} de la matriz de incidencia del grafo);
- finalmente, la cabeza lee el símbolo de la celda donde se encuentra (figura 4.39), y si el símbolo leído es 1, esto significa que (v_i, v_{i+1}) realmente es una arista del grafo, y el submódulo pasa a un estado q_y y continúa después hacia el paso 6 (figura 4.36); en caso de que el símbolo leído sea 0, entonces (v_i, v_{i+1}) no es una arista del grafo, y el submódulo pasa a al estado de paro q_N y termina el programa con el resultado de que la solución candidata no es una solución del problema.

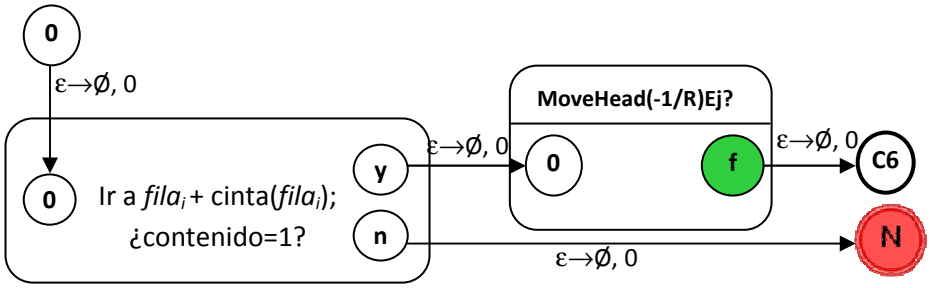


Figura 4.36.- Submódulo del paso 5: Ir a posición $fila_i$ e ir cinta($fila_i$) posiciones a la derecha; si el contenido es 1 ir al paso 6 si no parar en el estado "N".

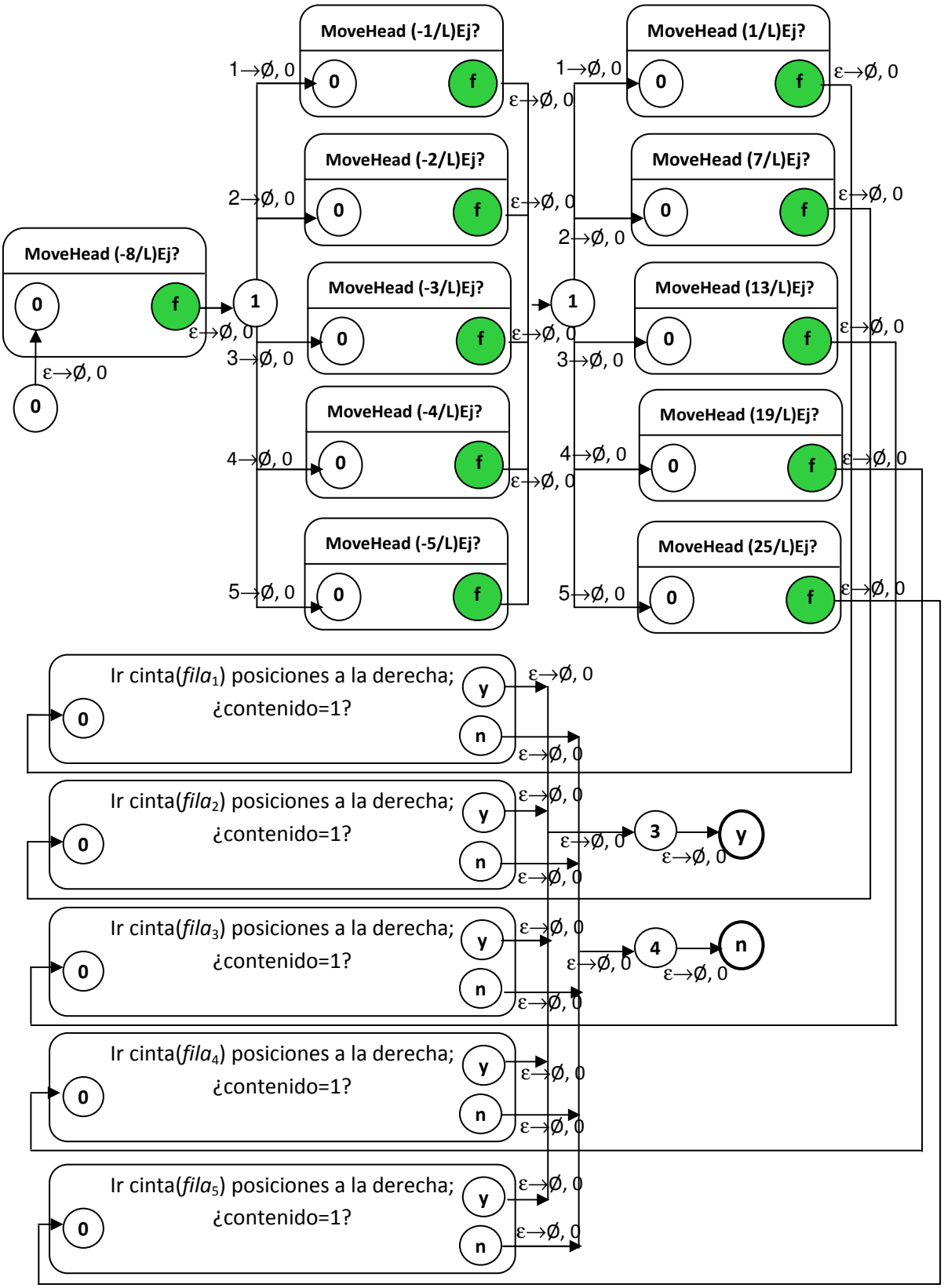


Figura 4.37.- Submódulo "Ir a $fila_i$ + cinta($fila_i$)".

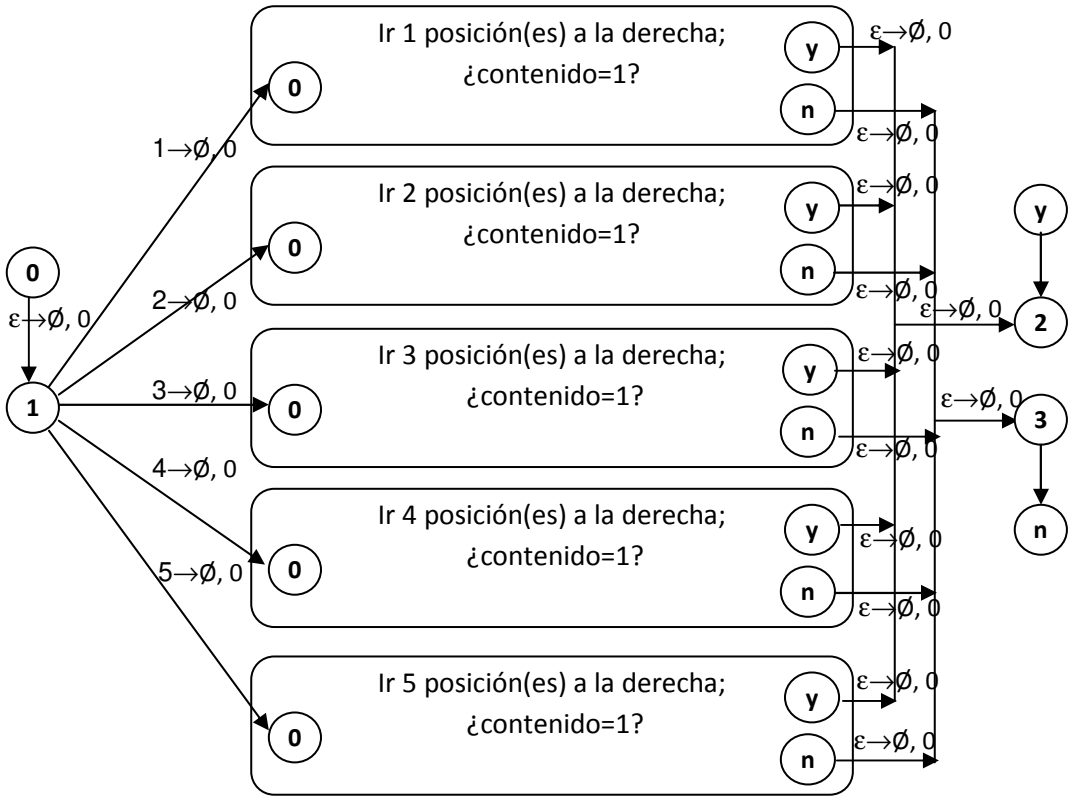


Figura 4.38.- Submódulo "Ir a cinta (*fila*$_i$) posiciones a la derecha".

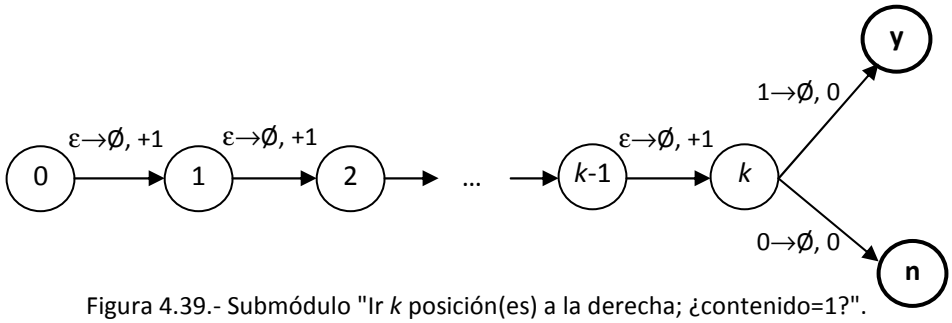


Figura 4.39.- Submódulo "Ir *k* posición(es) a la derecha; ¿contenido=1?".

¿El par de vértices es arista?: Paso 6

El paso 6 se encarga de incrementar la variable *i* una unidad. La figura 4.40 muestra el submódulo de este paso.

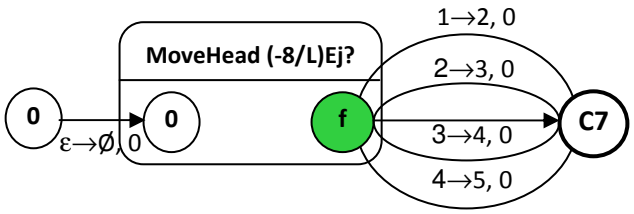


Figura 4.40.- Submódulo del paso 6: Hacer *i*=*i*+1.

¿El par de vértices es arista?: Paso 7

El séptimo y último paso verifica si la variable *i* no es mayor a cinco. Si $i \leq 5$, regresa al paso 4, de lo contrario, pasa al estado "q_f" (figura 4.41) y después al estado de aceptación "q_Y" y termina el proceso de la MTND, lo cual significa que la solución candidata sí es una solución del problema y éste es un ejemplar-sí.

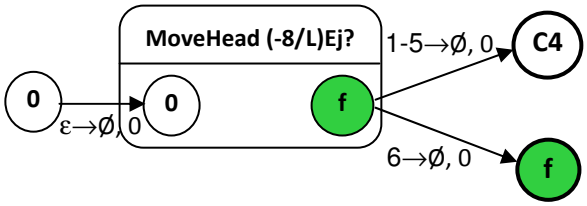


Figura 4.41.- Submódulo del paso 7: Si *i*≤5 ir al paso 4; si *i*=6 parar en el estado "Y".

4.3 Diagrama de flujo de datos del traductor de autómatas

Como ya se ha mencionado, el desarrollo de este proyecto involucra dos versiones. La versión 1 es un programa simulador, cuyas reglas de transición son introducidas mediante archivos. Para la implementación de la versión 2, se hizo uso de la versión 1 y se desarrolló un ambiente gráfico para el simulador.

El proceso de desarrollo estuvo divido en 2 fases: la fase 1 y la fase 2. La fase 1 (versión 1) a su vez estuvo constituida por la subfase 1A y la subfase 1B. La subfase 1A consistió en la implementación de un simulador de MTND, y la subfase 1B consistió en la implementación de un traductor de autómatas. La fase 2 consistió en la implementación de un ambiente gráfico para el simulador (versión 2).

Para entender el funcionamiento del traductor de autómatas, a continuación se describe brevemente su funcionamiento mediante un diagrama de flujo de datos (ver figura 4.42).

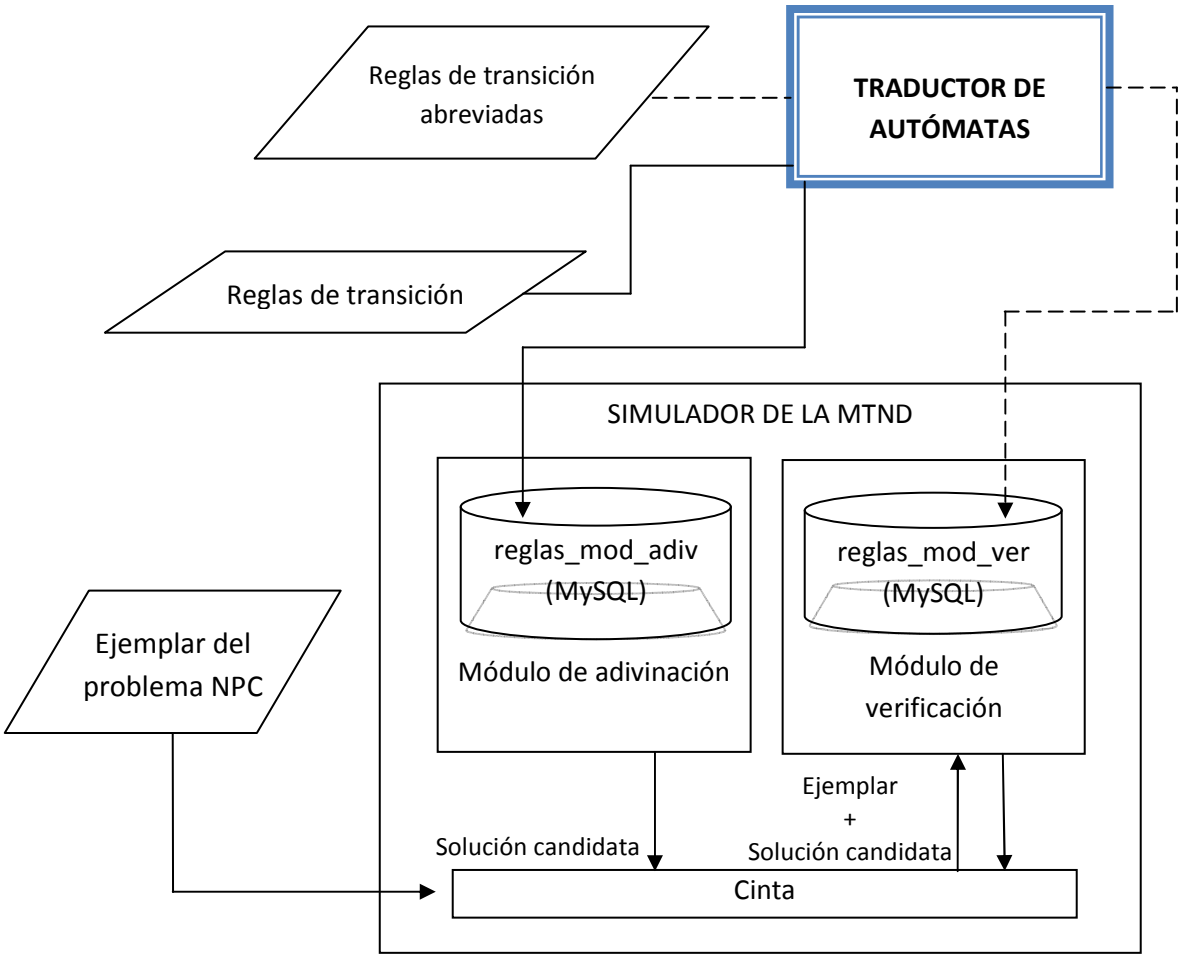


Figura 4.42.- Diagrama de flujo de datos del traductor de autómatas.

Las reglas de transición abreviadas, constituyen la descripción del autómata de estados finitos de un problema NPC. Estas reglas son traducidas a reglas completas por el traductor de autómatas. El propósito del traductor de autómatas es ahorrar trabajo a los usuarios, ya que permite escribir reglas abreviadas, las cuales después de ser procesadas por el traductor de autómatas, se pueden convertir en reglas más complejas o incluso cada regla abreviada se puede convertir en varias reglas. A propósito, este traductor de autómatas se encarga de generar automáticamente las reglas para los módulos "MoveHead (-k/L)", "MoveHead (k/L)", "MoveHead (-k/R)" y "MoveHead (k/R)" (ver figura 4.15).

Para el módulo de verificación, el traductor de autómatas almacena las reglas de transición completas en una tabla llamada "reglas_mod_ver" de la base de datos "transiciones" correspondiente al módulo de verificación. El módulo de verificación se alimenta de las reglas completas y del ejemplar del problema NP-completo y determina si la solución candidata es realmente o no una solución del ejemplar en cuestión.

Para el módulo de adivinación, las reglas de transición son almacenadas por el traductor de autómatas en la tabla “reglas_mod_adiv” de la base de datos “transiciones”.

Después de que el traductor ha procesado las reglas de transición, se puede ejecutar el simulador de la MTND.

La figura 4.43 muestra un conjunto de reglas abreviadas introducidas por el usuario para el submódulo A3 (figura 4.19). En la figura 4.43 los estados se denotan por medio de etiquetas simples y etiquetas compuestas. Una etiqueta simple es una cadena *c* de símbolos con excepción del punto “.“ y el símbolo blanco “b“. Una etiqueta compuesta es una cadena $c_1.c_2$, donde c_1 y c_2 son dos cadenas (con excepción de “.“ y “b“) separadas por un punto; y además c_1 representa el nombre de un submódulo. Las etiquetas simples se usan para denotar un estado de un submódulo dentro de la definición de reglas de éste; por ejemplo, la etiqueta simple 0 en la figura 4.43 representa el estado 0 del submódulo A3. Las etiquetas compuestas se emplean para denotar un estado de un submódulo ajeno; por ejemplo, la etiqueta compuesta MoveHead(-7/L)Ej?.f representa el estado f del submódulo MoveHead(-7/L)Ej?.

Para la definición de reglas, los símbolos ε y ∅ se denotan por las cadenas Epsilon y NoEscribir respectivamente. Es conveniente aclarar que estas cadenas se usan exclusivamente en la definición de reglas, y que no son símbolos de entrada (que se pueden leer de la cinta) ni símbolos de salida (que se pueden escribir en la cinta).

ESTADO_ACTUAL	SIMBOLO_ENTRADA	ESTADO_NUEVO	SIMBOLO_SALIDA	MOVIMIENTO
0	Epsilon	MoveHead(-7/L)Ej?.0	NoEscribir	0
MoveHead(-7/L)Ej?.f	1	a4	2	0
MoveHead(-7/L)Ej?.f	2	a4	3	0
MoveHead(-7/L)Ej?.f	3	a4	4	0
MoveHead(-7/L)Ej?.f	4	a4	5	0
MoveHead(-7/L)Ej?.f	5	a4	6	0

Figura 4.43.- Diagrama de flujo de datos del traductor de autómatas.

La figura 4.44 muestra las reglas completas correspondientes a las reglas abreviadas de la figura 4.43 después de que fueron procesadas por el traductor de autómatas.

ESTADO_ACTUAL	SIMBOLO_ENTRADA	ESTADO_NUEVO	SIMBOLO_SALIDA	MOVIMIENTO
A3.0	Epsilon	MoveHead(-7/L)001.0	NoEscribir	0
MoveHead(-7/L)Ej001.f	1	A3.a4	2	0
MoveHead(-7/L)Ej001.f	2	A3.a4	3	0
MoveHead(-7/L) Ej001.f	3	A3.a4	4	0
MoveHead(-7/L) Ej001.f	4	A3.a4	5	0
MoveHead(-7/L) Ej001.f	5	A3.a4	6	0
MoveHead(-7/L) Ej001.0	<>|	MoveHead(-7/L) Ej001.0	NoEscribir	+1
MoveHead(-7/L) Ej001.0	|	MoveHead(-7/L) Ej001.-1	NoEscribir	-1
MoveHead(-7/L) Ej001.-1	Epsilon	MoveHead(-7/L) Ej001.-2	NoEscribir	-1
MoveHead(-7/L) Ej001.-2	Epsilon	MoveHead(-7/L) Ej001.-3	NoEscribir	-1
MoveHead(-7/L) Ej001.-3	Epsilon	MoveHead(-7/L) Ej001.-4	NoEscribir	-1
MoveHead(-7/L) Ej001.-4	Epsilon	MoveHead(-7/L) Ej001.-5	NoEscribir	-1
MoveHead(-7/L) Ej001.-5	Epsilon	MoveHead(-7/L) Ej001.-6	NoEscribir	-1

MoveHead(-7/L) Ej001.-6	Epsilon	MoveHead(-7/L) Ej001.-7	NoEscribir	-1
MoveHead(-7/L) Ej001.-7	Epsilon	MoveHead(-7/L) Ej001.f	NoEscribir	0

Figura 4.44.- Diagrama de flujo de datos del traductor de autómatas.

Si en SIMBOLO_ENTRADA de la figura 4.43 el usuario necesitara los símbolos 1, 2, 3 o 4 en alguna de las seis reglas, sólo bastaría con colocar 1-4 de forma abreviada y automáticamente el traductor generaría la secuencia 1,2,3,4. Esto se puede observar en la figura 4.45. En este punto es importante aclarar que, con el fin de reducir el número de reglas de los autómatas, el simulador de la MTND fue dotado con la capacidad de interpretar reglas que como símbolo de entrada tengan una lista de símbolos $s_1,s_2,...,s_n$ separados por comas; en tal caso el autómata debe hacer una transición cuando el símbolo leído de la cinta sea s_1, o s_2, o ..., o s_n.

Regla abreviada				
ESTADO_ACTUAL	SIMBOLO_ENTRADA	SIMBOLO_ENTRADA	SIMBOLO_SALIDA	MOVIMIENTO
MoveHead(-7/L)Ej?.f	1-4	a4	2	0
Regla completa				
ESTADO_ACTUAL	SIMBOLO_ENTRADA	SIMBOLO_ENTRADA	SIMBOLO_SALIDA	MOVIMIENTO
MoveHead(-7/L)Ej001.f	1,2,3,4	A3.a4	2	0

Figura 4.45.- Diagrama de flujo de datos del traductor de autómatas.

El símbolo ≠, el cual se ha utilizado con el significado *cualquier símbolo diferente de* (figura 4.15), se denota por la cadena <> en las reglas abreviadas. En este caso, también con el fin de ahorrar reglas, el simulador de la MTND fue dotado de la capacidad de interpretar las reglas donde aparezca la cadena <>*s* como símbolo de entrada, en cuyo caso el autómata debe hacer una transición cuando el símbolo leído de la cinta sea diferente de *s*. Finalmente, es conveniente mencionar que es posible escribir una regla abreviada cuyo símbolo de entrada sea la cadena <>s_1-s_n, lo cual da como resultado una regla completa con la siguiente cadena como símbolo de entrada: <>$s_1,s_2,...,s_n$, la cual puede ser interpretada directamente por el simulador.

4.4 Diagrama de Flujo de Datos (DFD) del programa simulador

Para entender el funcionamiento del simulador de la MTND, a continuación se describe brevemente su funcionamiento mediante un diagrama de flujo de datos (ver figura 4.46).

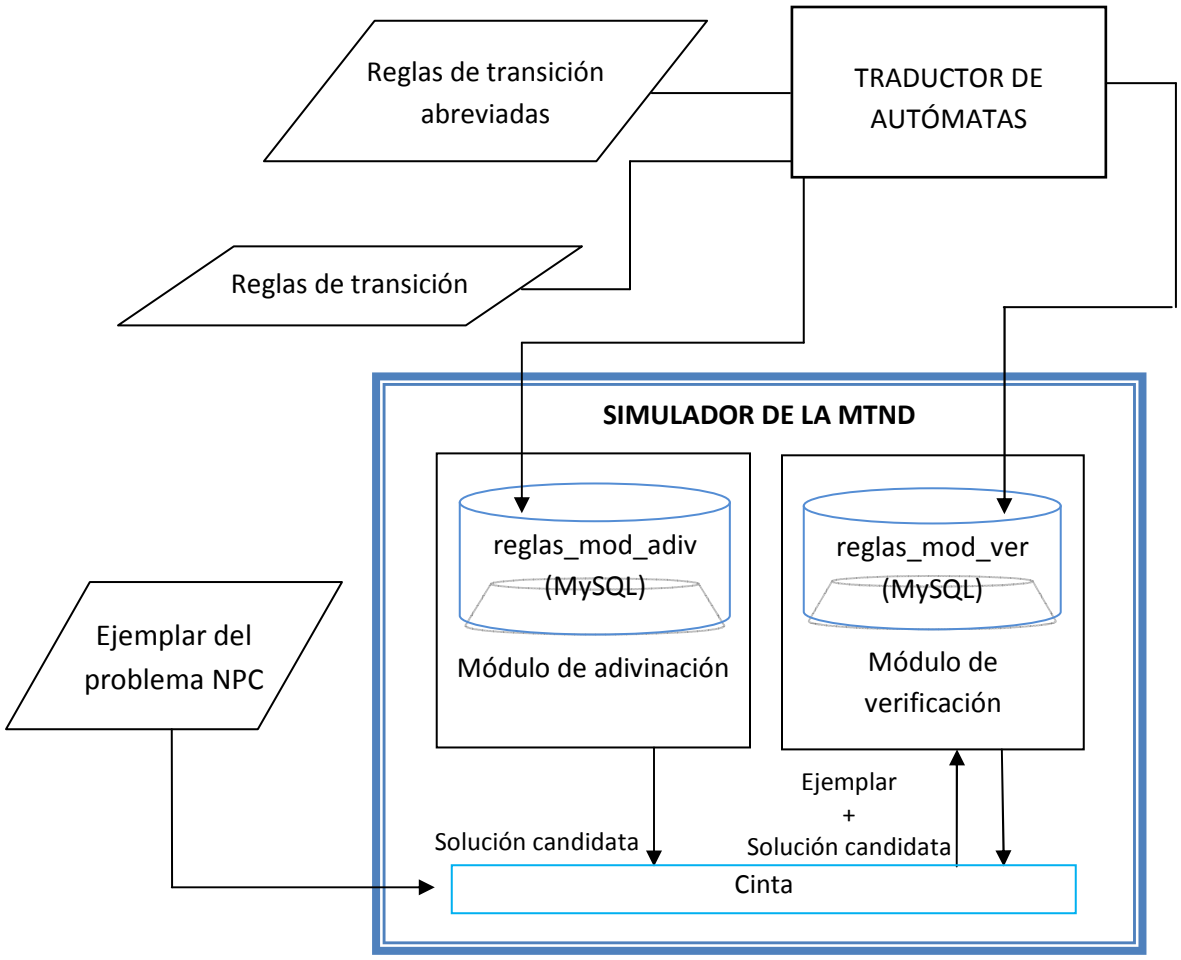


Figura 4.46.- Diagrama de flujo de datos del programa simulador.

Como se puede observar en la figura 4.46 el simulador de la MTND está formado por un módulo de adivinación y un módulo de verificación, los cuales se comunican a través de la cinta. La cinta almacena en la parte izquierda la solución candidata y en la parte derecha el ejemplar del problema NPC.

Para una mayor comprensión del módulo de adivinación y de verificación, se describe el funcionamiento de cada uno de ellos, mediante pseudocódigo.

Antes, es conveniente mencionar que la cinta de la MTND se representa internamente mediante un arreglo de caracteres, de tal manera que cada símbolo de la cinta se puede almacenar en una posición del arreglo. Desafortunadamente, es imposible implementar una cinta de longitud infinita mediante un arreglo; por lo tanto, es necesario informar al simulador la dimensión del arreglo que se necesita mediante un archivo de configuración. Para tal efecto, este archivo debe contener dos números enteros max_{izq} y max_{der}, escogidos de tal manera que durante el procesamiento de cualquier ejemplar de un problema dado, la cabeza nunca deba moverse a la izquierda más allá de la posición $-max_{izq}$ de la cinta ni deba moverse a la derecha más allá de la posición max_{der}.

El manejo de las posiciones negativas de la cinta ocasiona problemas para su manipulación mediante un arreglo. Este problema, se puede resolver fácilmente utilizando dos variables: *posición_virtual*, la cual representa una posición en la cinta (que puede ser positiva, cero o negativa), y *posición_real*, la cual representa una posición en el arreglo (que sólo puede ser positiva) que corresponde a la posición de cinta indicada por *posición_virtual*. Los valores de las dos variables están relacionados por la siguiente expresión: *posición_real* = *posición_virtual* + max_{izq}. Por ejemplo, de esta manera la posición $-max_{izq}$ de la cinta será representada en el arreglo por la posición 0, y la posición 0 de la cinta será representada en el arreglo por la posición max_{izq}.

Pseudocódigo del módulo de adivinación:

```
Definir continuar, posición_virtual, estado_actual, num_reglas, clave_error, número_regla, cinta(), mov
// continuar: variable booleana para controlar un ciclo Mientras
// posición_virtual: entero (positivo, cero o negativo) que indica una posición virtual en la cinta
// estado_actual: estado actual del autómata
// num_reglas: entero que indica el número de reglas que satisfacen cierta condición
// clave_error: indica si se encontraron reglas que satisfacen cierta condición
// número_regla: entero que indica la regla elegida aleatoriamente (ver tabla 5.1)
// cinta(): arreglo de caracteres que representa a la cinta del autómata
// mov: entero (+1, 0 o –1) que indica el movimiento de la cabeza indicado por una regla

Hacer continuar = verdadero.
Hacer posición_virtual = –1. //El proceso inicia con la cabeza en la posición –1 de la cinta
Hacer estado_actual = "0".

Mientras continuar = verdadero.

    Extraer num_reglas de la tabla "reglas_mod_adiv"
    tal que el estado actual de la regla sea estado_actual y
    el símbolo de entrada de la regla sea ε, es decir, transición espontánea.
    Si clave_error = 0. //Se encontraron una o más reglas con el símbolo ε
        Insertar números enteros consecutivos entre 1 y num_reglas
        en la columna "numero_regla" para cada regla de la tabla "reglas_mod_adiv"
        donde el estado actual de la regla sea estado_actual.
        Generar un número entero aleatorio número_regla en el intervalo de 1 a num_reglas.
        Extraer regla de transición de la tabla "reglas_mod_adiv"
        tal que el estado actual de la regla sea estado_actual y
        y el número de regla sea número_regla.
        Hacer cinta(posición_virtual) = símbolo de salida indicado por la regla.
        Hacer estado_actual = estado nuevo de la regla.
        Hacer posición_virtual = posición_virtual + mov.
    Fin si.
    Si clave_error ≠ 0. //No se encontraron reglas
       Terminar mientras. //Se detectó un error en el procesamiento
    Fin si.
    Imprimir configuración de la máquina.
    Si estado_actual = "f". //Se encontró el estado de paro f
       Terminar mientras.
    Fin si.

Fin mientras.
```

Pseudocódigo del módulo de verificación:

Definir *continuar*, *posición_virtual*, *estado_actual*, *clave_error*, *cinta*(), $max_{visitas}$, *mov*, max_{izq}, max_{der}, *símbolo_entrada*.
// *continuar*: variable booleana para controlar un ciclo **Mientras**
// *posición_virtual*: entero (positivo, cero o negativo) que indica una posición virtual en la cinta
// *estado_actual*: estado actual del autómata
// *clave_error*: indica si se encontraron reglas que satisfacen cierta condición
// *cinta*(): arreglo de caracteres que representa a la cinta del autómata
// $num\_visitas_k$: entero que indica el No. de visitas a un estado *k* almacenado en la BD
// $max_{visitas}$: entero que indica el No. máximo de visitas posibles a un estado
// *mov:* entero (+1, 0 o –1) que indica el movimiento de la cabeza indicado por una regla
// max_{izq}: entero positivo que indica la posición límite de la cabeza del lado izquierdo de la cinta
// max_{der}: entero positivo que indica la posición límite de la cabeza del lado derecho de la cinta
// *símbolo_entrada*: símbolo leído de una posición del arreglo *cinta*

Hacer *continuar* = verdadero.
Hacer *posición_virtual* = 1. //El proceso inicia con la cabeza en la posición 1 de la cinta
Hacer *estado_actual* = "0".

Mientras *continuar* = verdadero.

Extraer regla de transición de la tabla "reglas_mod_ver"
tal que el estado actual de la regla sea *estado_actual* y
el símbolo de entrada de la regla sea ε, es decir, transición espontánea.
Si *clave_error* = 0. //Se encontró regla para símbolo ε.
Hacer *cinta*(*posición_virtual*) = símbolo de salida indicado por la regla.
Hacer *estado_actual* = estado nuevo indicado por la regla.
Hacer $num\_visitas_{estado\_actual} = num\_visitas_{estado\_actual} + 1$.
Si $num\_visitas_{estado\_actual} > max_{visitas}$.
Terminar mientras. //Se detectó ciclado del autómata
Fin si.
Hacer *posición_virtual* = *posición_virtual* + *mov*.
Si *posición_virtual* $< -max_{izq}$ **o** *posición_virtual* $> max_{der}$.
Terminar mientras. //Se detectó intento de salir de la cinta
Fin si.
Fin si.
Si *clave_error* = 1. //No se encontró regla para símbolo ε.
Hacer *símbolo_entrada* = *cinta*(*posición_virtual*). //Leer de cinta
Extraer regla de transición de la tabla "reglas_mod_ver"
tal que el estado actual de la regla sea *estado_actual* y
el símbolo de entrada de la regla sea *símbolo_entrada*.
Si *clave_error* = 0. //Se encontró regla para *símbolo_entrada*
Hacer *cinta*(*posición_virtual*) = símbolo de salida indicado por la regla.
Hacer *estado_actual* = estado nuevo indicado por la regla.
Hacer $num\_visitas_{estado\_actual} = num\_visitas_{estado\_actual} + 1$.
Si $num\_visitas_{estado\_actual} > max_{visitas}$.
Terminar **mientras**. //Se detectó ciclado del autómata
Fin si.
Hacer *posición_virtual* = *posición_virtual* + *mov*.
Si *posición_virtual* $< -max_{izq}$ **o** *posición_virtual* $> max_{der}$.
Terminar mientras. //Se detectó intento de salir de la cinta
Fin si.

Fin si.
Fin si.
Si *clave_error* ≠ 0. //No se encontró regla para *símbolo_entrada*
Terminar **mientras**. //Se detectó un error en el procesamiento
Fin si.
Si *estado_actual* = "Y".
Terminar mientras. //El ejemplar es un ejemplar-sí
Fin si.
Si *estado_actual* = "N".
Terminar mientras. //El ejemplar es un ejemplar-no
Fin si.
Si *estado_actual* = "I".
Terminar mientras. //El ejemplar es inválido
Fin si.
Imprimir configuración de la máquina.

Fin mientras.

CAPÍTULO 5
Desarrollo y descripción de la herramienta

En este capítulo se hace una descripción de la herramienta, misma que hace referencia a la descripción de la base de datos empleada, a los elementos de la interfaz, así como el ambiente de de simulación.

5.1 Descripción de la base de datos

La base de datos empleada por la herramienta se implementó en MySQL. Se utilizó la biblioteca mysql-connector-java-5.1.18-bin.jar para poder acceder y operar en la base de datos desde un programa en Java. A continuación se presenta una breve descripción de cada una de las tablas de la base de datos "transiciones" utilizadas por la herramienta.

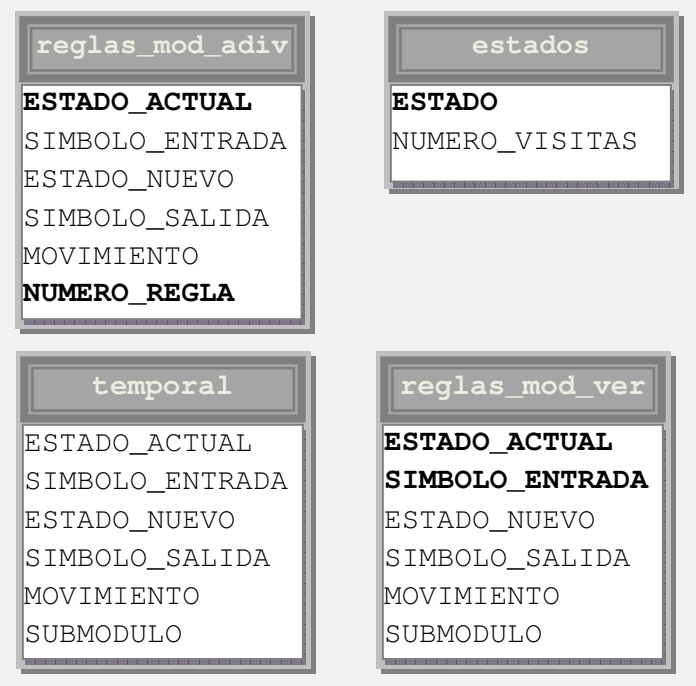


Figura 5.1.- Tablas de la base de datos transiciones.

Es importante mencionar que aunque la base de datos transiciones cuenta con las tablas reglas_mod_adiv, estados, temporal y reglas_mod_ver, para la versión 1 del programa simulador se hace uso de todas las tablas; sin embargo, para la versión 2, correspondiente al ambiente gráfico sólo se utiliza la tabla “temporal” y la tabla “reglas_mod_ver”.

Las tablas 5.1, 5.2, 5.3 y 5.4 muestran la estructura de las tablas de la base de datos. Para cada tabla se describen sus columnas, y para cada una de éstas se muestran su nombre, tipo de dato, longitud y propiedades.

Tabla 5.1.- Tabla reglas_mod_adiv.

Nombre de la columna	Tipo de dato	Longitud	Propiedades
ESTADO_ACTUAL	Texto	100	Índice
SIMBOLO_ENTRADA	Texto	100	
ESTADO_NUEVO	Texto	100	
SIMBOLO_SALIDA	Texto	100	

MOVIMIENTO	Texto	100	
NUMERO_REGLA	Numérico	100	Índice

La tabla reglas_mod_adiv (tabla 5.1) almacena las reglas de transición correspondientes al módulo de adivinación. ESTADO_ACTUAL almacena el estado actual del autómata, SIMBOLO_ENTRADA almacena el símbolo ε (sólo almacena el símbolo épsilon porque el módulo de adivinación no puede leer), ESTADO_NUEVO almacena el estado nuevo del autómata, SIMBOLO_SALIDA almacena el símbolo que se desea escribir en la cinta, y NUMERO_REGLA almacena un número diferente para cada una de las reglas que tienen el mismo estado en la columna ESTADO_ACTUAL, dicho número sirve para el proceso de selección aleatoria de una de estas reglas.

Tabla 5.2.- Tabla estados.

Nombre de la columna	Tipo de dato	Longitud	Propiedades
ESTADO	Texto	100	Llave primaria
NUMERO_VISITAS	Numérico	100	

La tabla estados (tabla 5.2) almacena el nombre de los estados y el número de veces que ha sido visitado cada estado. ESTADO almacena cada uno de los estados del autómata y NUMERO_VISITAS almacena el número de veces que ha sido visitado cada estado del autómata.

Tabla 5.3.- Tabla temporal.

Nombre de la columna	Tipo de dato	Longitud	Propiedades
ESTADO_ACTUAL	Texto	100	
SIMBOLO_ENTRADA	Texto	20	
ESTADO_NUEVO	Texto	100	
SIMBOLO_SALIDA	Texto	20	

MOVIMIENTO	Texto	20	
SUBMODULO	Numérico	20	

La tabla temporal (tabla 5.3) almacena las reglas abreviadas del módulo de verificación. ESTADO_ACTUAL almacena el estado actual del autómata, SIMBOLO_ENTRADA almacena el símbolo que se podría leer de la cinta, ESTADO_NUEVO almacena el estado nuevo del autómata, SIMBOLO_SALIDA almacena el símbolo que se desea escribir en la cinta, MOVIMIENTO almacena el movimiento (+1, 0 o –1) que se desea que haga la cabeza, y SUBMODULO almacena el nombre del submódulo del conjunto de reglas de transición del problema en cuestión.

Tabla 5.4.- Tabla reglas_mod_ver.

Nombre de la columna	Tipo de dato	Longitud	Propiedades
ESTADO_ACTUAL	Texto	100	Llave primaria
SIMBOLO_ENTRADA	Texto	20	Llave primaria
ESTADO_NUEVO	Texto	100	Índice
SIMBOLO_SALIDA	Texto	20	
MOVIMIENTO	Texto	20	
SUBMODULO	Texto	20	

La tabla reglas_mod_ver (tabla 5.4) almacena las reglas de transición completas del módulo de verificación. En las columnas de esta tabla se almacena información semejante a la de las columnas de la tabla 5.3. La diferencia consiste en que en la tabla temporal no se define llave primaria, mientras que en la tabla reglas_mod_ver la llave primaria está constituida por las columnas ESTADO_ACTUAL y SIMBOLO_ENTRADA.

5.2 Elementos de la interfaz

El proceso de resolver un problema de decisión mediante una máquina de Turing no determinista involucra los cuatro elementos fundamentales de la interfaz:

- Configuración de la base de datos.
- Editor para la captura de reglas.
- Traductor de autómatas.
- Simulador de la MTND.

Mediante la figura 5.2 se muestra el diagrama de flujo de la interfaz.

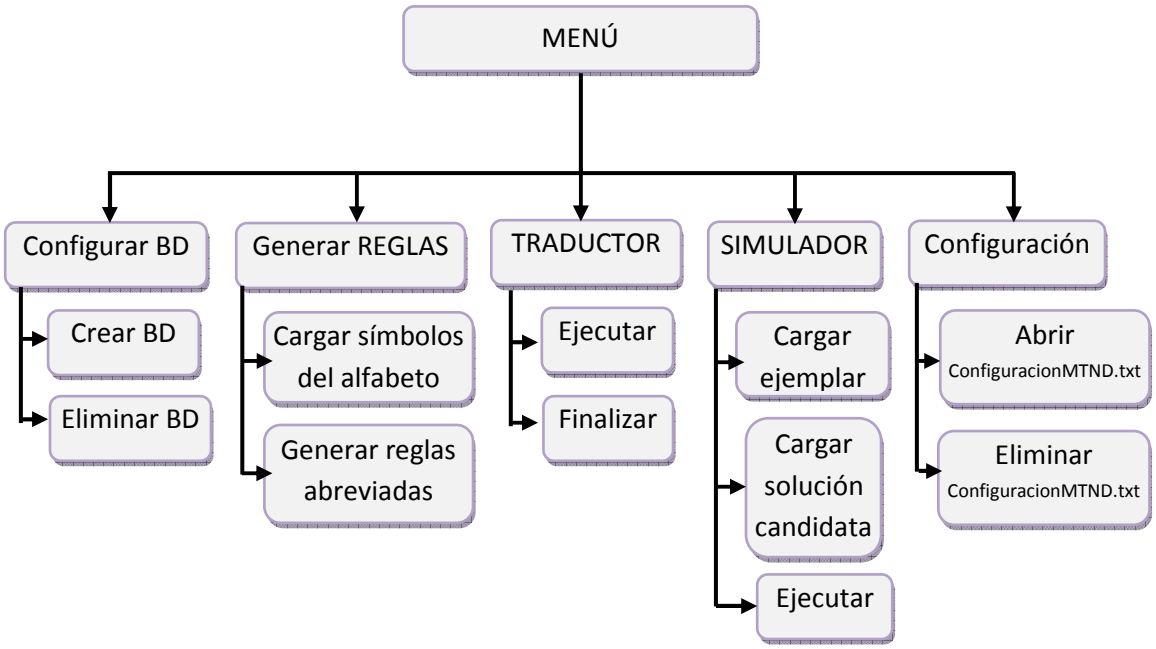


Figura 5.2.- Diagrama de flujo de la interfaz gráfica del simulador de MTND.

5.2.1 Configuración de la base de datos

Cuando el usuario utiliza por primera vez la herramienta, no conoce el estatus de la base de datos, si existe o no para proceder con el uso del simulador. La configuración de la base de datos se encuentra dentro de un menú llamado “Configurar BD”. A continuación se explica a detalle mediante la figura 5.3.

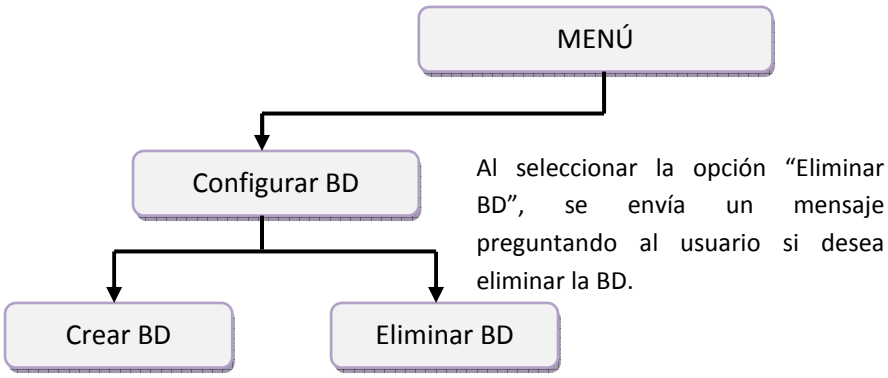


Figura 5.3.- Diagrama de flujo del menú Configurar BD.

5.2.2 Editor para la captura de reglas

Cuando ya se ha configurado la base de datos, se pueden generar las reglas abreviadas del problema de decisión, mediante un editor de autómatas. Sin embargo, es importante mencionar que primero se deberán cargar los símbolos del alfabeto, pero en caso de que los símbolos no se hayan cargado en el sistema, no se podrán generar las reglas abreviadas a través del editor. A continuación se explica a detalle mediante la figura 5.4.

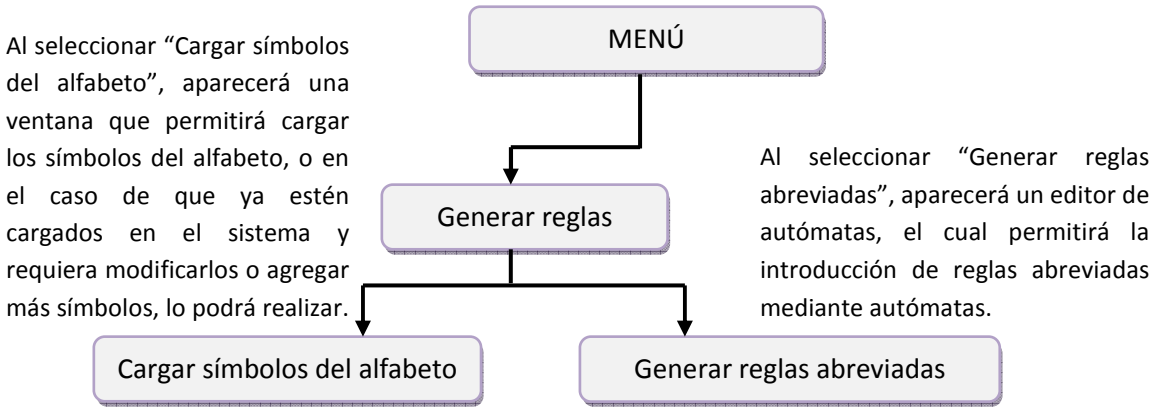


Figura 5.4.- Diagrama de flujo del menú Generar reglas.

Como ya se mencionó en la figura anterior, al seleccionar la opción “Generar reglas abreviadas”, aparece un editor de autómatas. A continuación se presenta el diagrama de flujo de la interfaz del editor, mediante la figura 5.5.

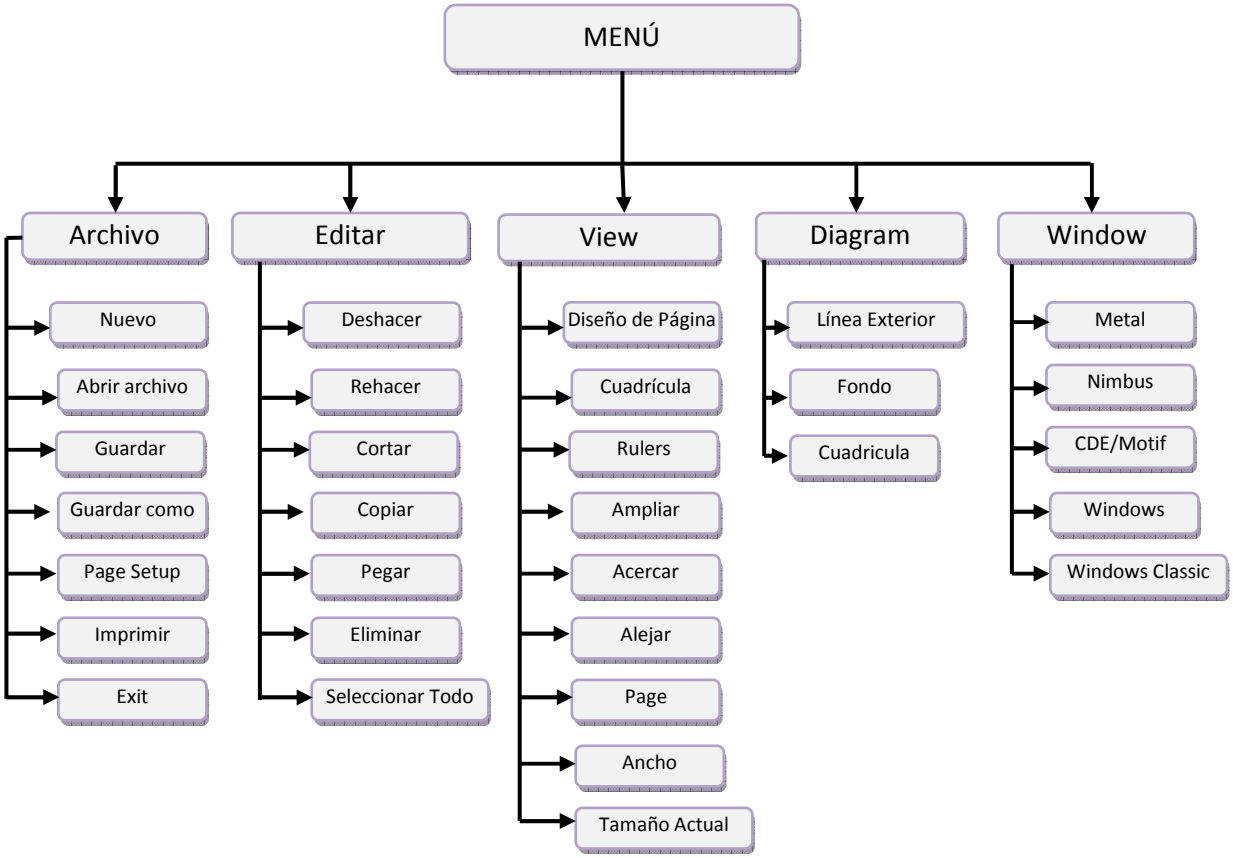


Figura 5.5.- Diagrama de flujo de la interfaz del editor de autómatas.

En el diagrama de flujo de la figura 5.5 al seleccionar la opción Archivo, se podrá generar un nuevo autómata (con la opción Nuevo), abrir un archivo ya sea en formato png, mxe o vdx (con la opción Abrir archivo), guardar los cambios de la modificación de un archivo ya existente (con la opción Guardar), también se podrá guardar un archivo recién creado (con la opción Guardar como). Al presionar “Page Setup” se podrá elegir el tamaño de la página sobre la cual se desea trabajar, así como la orientación de la misma; si se desea imprimir el archivo se debe seleccionar “Imprimir”, y si desea salir del editor, se debe presionar “Exit”.

La opción Editar hace referencia a los elementos del archivo que se está editando. Mediante dicho menú, si se colocó una figura correspondiente a un autómata, se podrá deshacer esa figura (Deshacer) o se puede rehacerla nuevamente (Rehacer), también es posible copiar (Copiar) figuras, eliminarlas (Eliminar) y seleccionarlas (Seleccionar todo).

Con la opción View se puede cambiar el diseño de la página (Diseño de Página), agregar o quitar la cuadrícula de la página (Cuadrícula); también si se desea mantener ocultas o visibles las reglas, se debe seleccionar “Rulers”. Al seleccionar “Ampliar” la interfaz despliega la página con un cierto porcentaje de aumento. Las opciones “Acercar” y “Alejar”, permiten acercar y alejar la página respectivamente. “Page” permite regresar a la página a su forma actual y posición actual. Al seleccionar “Ancho” se colocará a la página en una posición a lo ancho. Finalmente, la opción “Tamaño actual” regresa a la página a su forma actual.

En el menú de la opción Diagram, al seleccionar “Línea Exterior” se podrá quitar o visualizar la ventana en miniatura lo que permite visualizar a menor escala el archivo sobre el cual se está trabajando. Si se desea un ambiente más colorido, se puede seleccionar “Fondo” y a continuación se puede cambiar tanto el color de fondo de la página, como el color de fondo del editor. Con la opción “Cuadrícula” se puede cambiar el diseño de la cuadrícula.

La ventana del editor puede adoptar cinco estilos diferentes. Para adoptar un estilo, sólo es necesario seleccionar el estilo deseado (Metal, Nimbus, CDE/Motif, Windows, Windows Classic).

5.2.3 Traductor de autómatas

Después de generar los autómatas de cada uno de los submódulos del problema, se debe presionar “Ejecutar”, lo cual permitirá cargar las reglas de transición abreviadas a la tabla temporal de la base de datos. Cuando ya se hayan cargado todos los autómatas, se debe seleccionar “Finalizar” para que el traductor valide las reglas de transición abreviadas y genere automáticamente las reglas de transición completas.

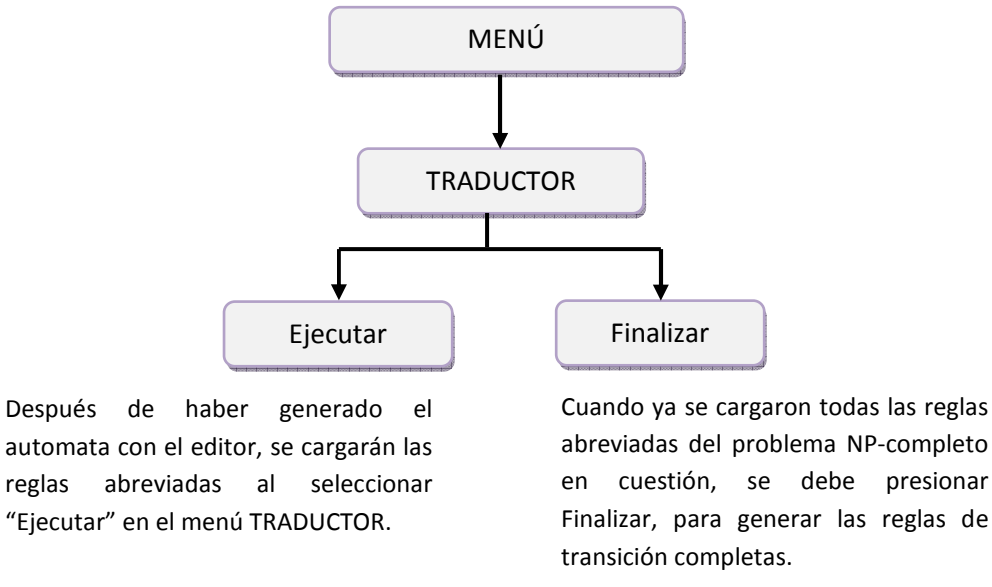


Figura 5.6.- Diagrama de flujo del menú TRADUCTOR.

5.2.4 Simulador de la MTN

Una vez que en la base de datos del MTND se encuentran las reglas de transición completas del problema de decisión, ya están listas para ser procesadas por el simulador de la MTND. Antes de ejecutar el simulador, se deberá cargar el ejemplar del problema codificado, para ello se debe seleccionar "Cargar ejemplar" (ver figura 5.7). Posteriormente se deberá cargar la solución candidata del problema de decisión. Cuando el ejemplar y la solución candidata ya fueron cargados, es momento de seleccionar "Ejecutar" para ejecutar el simulador de la MTND. Cuando el simulador termina su ejecución, éste genera un archivo de salida llamado "ConfiguracionMTND.txt", el cual contiene la configuración de la MTND del problema de decisión en cuestión. Para la manipulación de ese archivo, se puede hacer uso del menú "Configuración" (ver figura 5.8). (Nota: el archivo de configuración de la MTND muestra para cada paso de ejecución de la MTND la siguiente información: el contenido de las celdas de la cinta, la posición de la cabeza, el estado en que se encuentra el autómata de la MTND, el número de veces que ha sido visitado el estado, así como el número del paso de ejecución de la MTND).

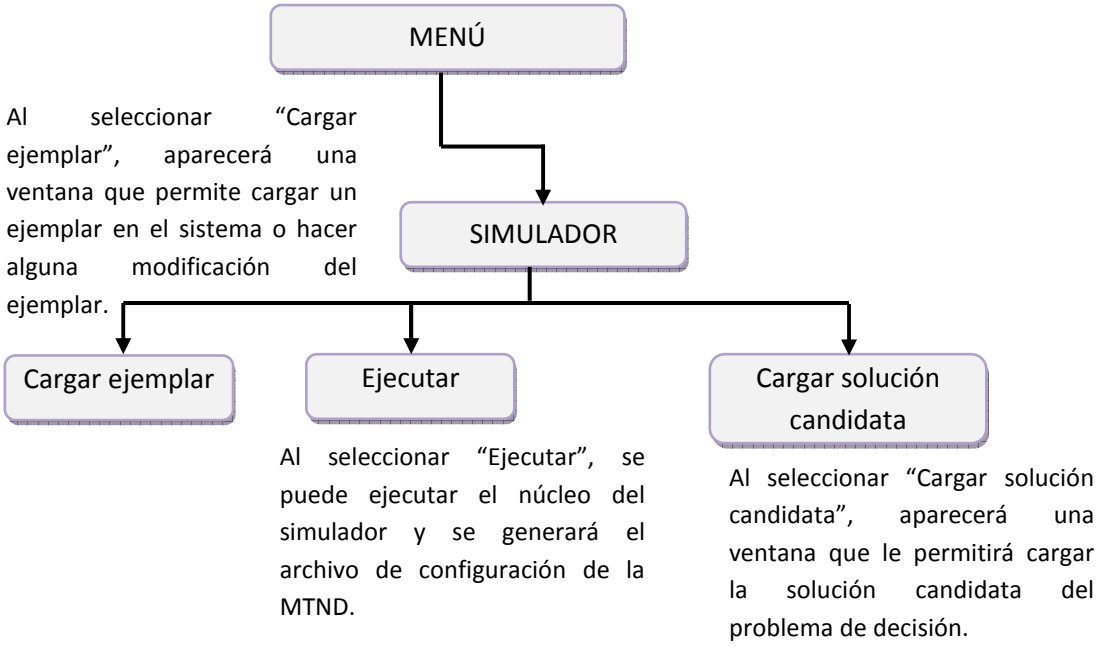


Figura 5.7.- Diagrama de flujo del menú simulador.

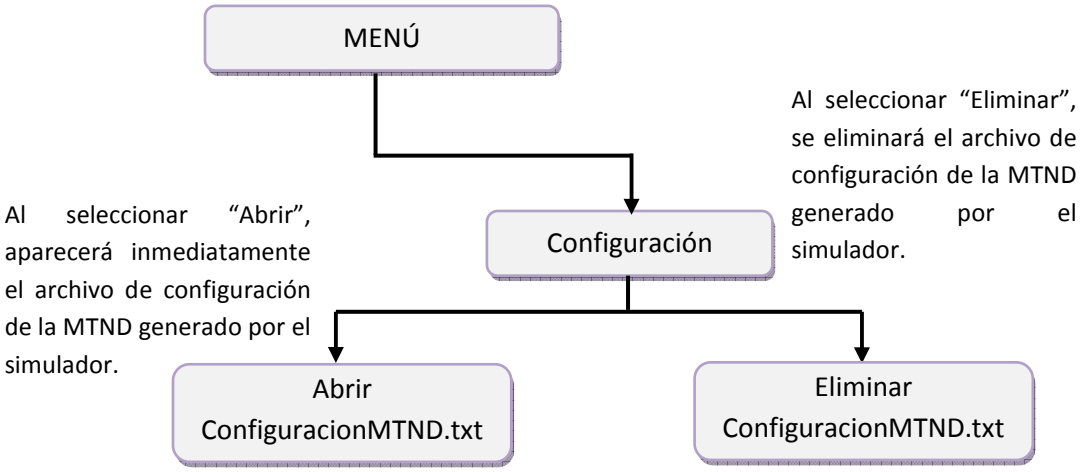


Figura 5.8.- Diagrama de flujo del menú Configuración.

5.3 Arquitectura de la interfaz del ambiente de simulación de MTND

El ambiente de simulación está constituido por una interfaz gráfica que consta de una barra de menús, tal como se presenta en la figura 5.9.

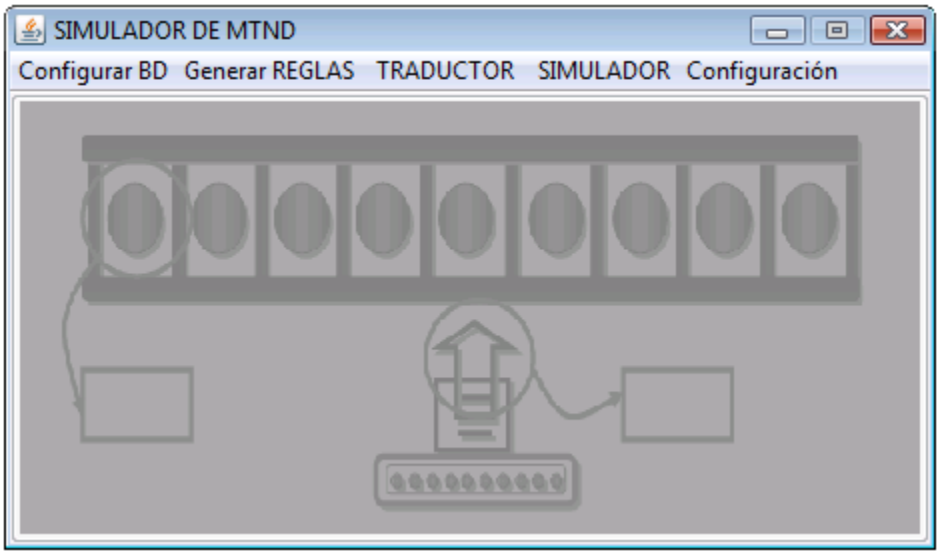


Figura 5.9.- Interfaz gráfica del simulador de la MTND.

La opción “Configurar BD” despliega como elemento de menú las opciones “Crear BD” y “Eliminar BD” (ver figura 5.10).

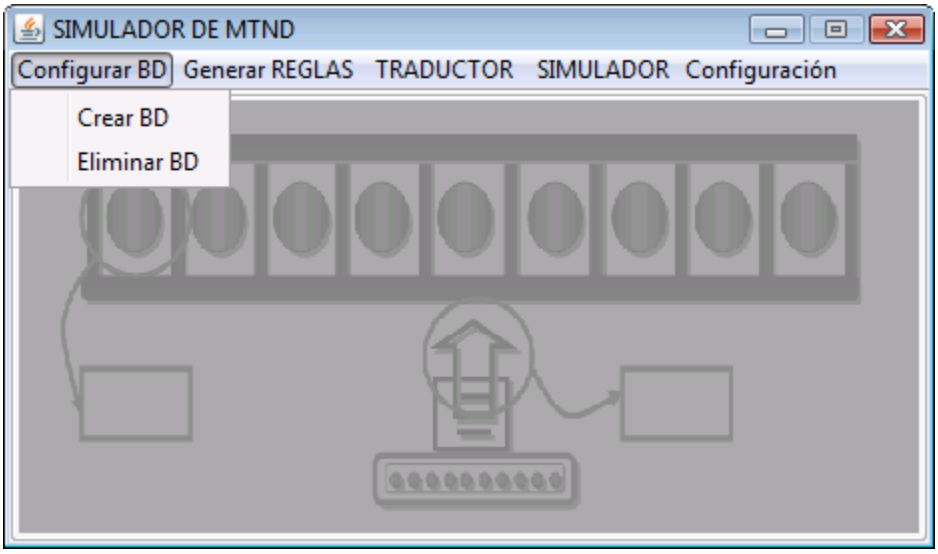


Figura 5.10.- Representación gráfica del menú Configurar BD.

La opción “Generar REGLAS” despliega como elemento de menú las opciones “Cargar símbolos del alfabeto” y “Generar reglas abreviadas”. Al seleccionar “Cargar símbolos del alfabeto”, aparece una ventana que permite cargar los símbolos del alfabeto mediante una ventana, la cual se muestra en la figura 5.11; y al seleccionar “Generar reglas abreviadas”, aparece el editor de autómatas (figura 5.12).

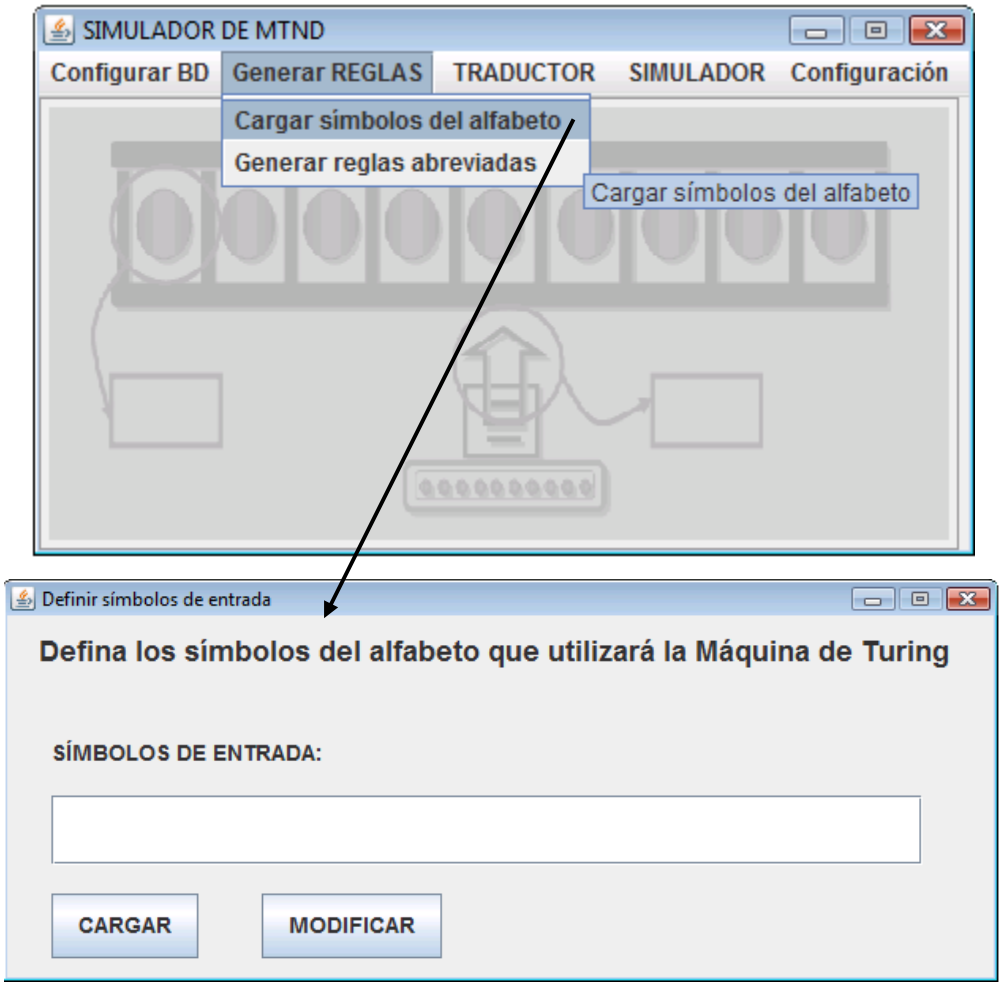


Figura 5.11.- Representación gráfica del menú Generar REGLAS.

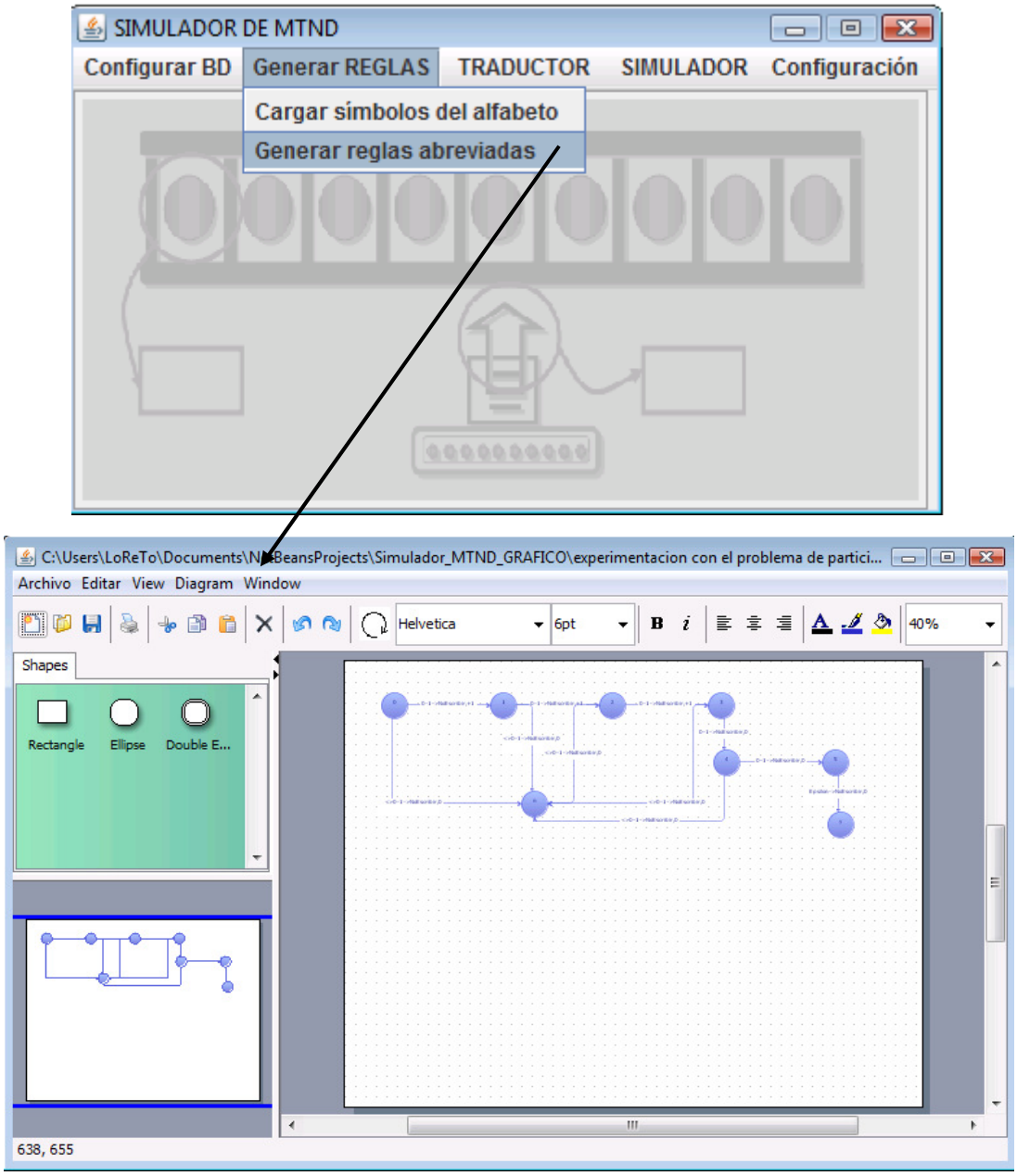


Figura 5.12.- Representación gráfica del editor de autómatas.

La opción “TRADUCTOR” despliega como elemento de menú las opciones “Ejecutar” y “Finalizar” (figura 5.13).

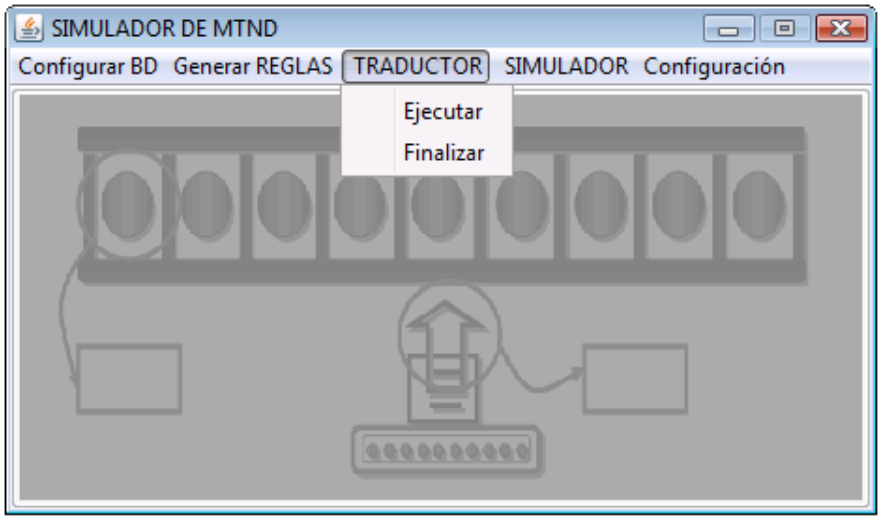


Figura 5.13.- Representación gráfica del menú TRADUCTOR.

La opción “SIMULADOR” despliega como elemento de menú las opciones “Cargar ejemplar”, “cargar solución candidata” y “Ejecutar”. Al seleccionar “Cargar ejemplar”, aparece una ventana, que permitirá cargar el ejemplar del problema NP-completo (figura 5.14).

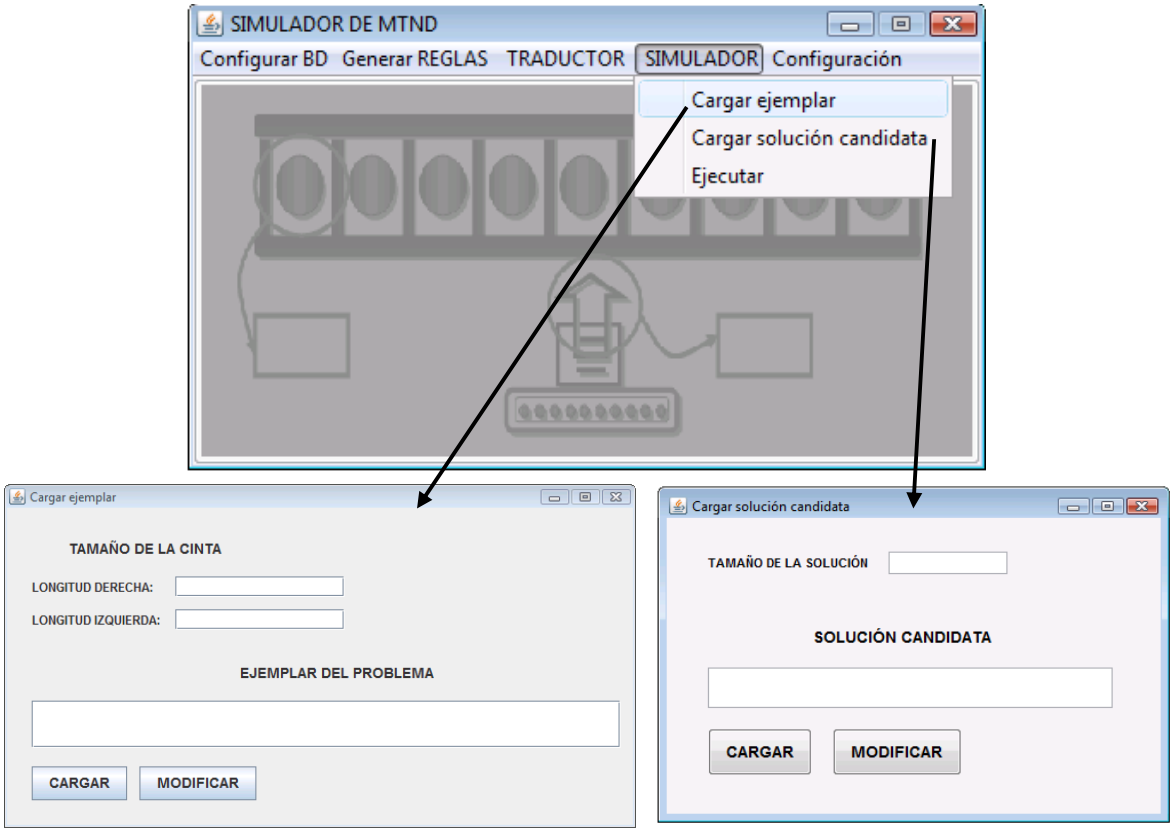


Figura 5.14.- Representación gráfica del menú SIMULADOR.

Al seleccionar “Cargar solución candidata”, aparece una ventana que permite cargar la solución candidata del problema de decisión (véase figura 5.14).

La opción “Configuración” despliega como elemento de menú las opciones “Abrir ConfiguracionMTND.txt” y “Eliminar ConfiguracionMTND.txt”.

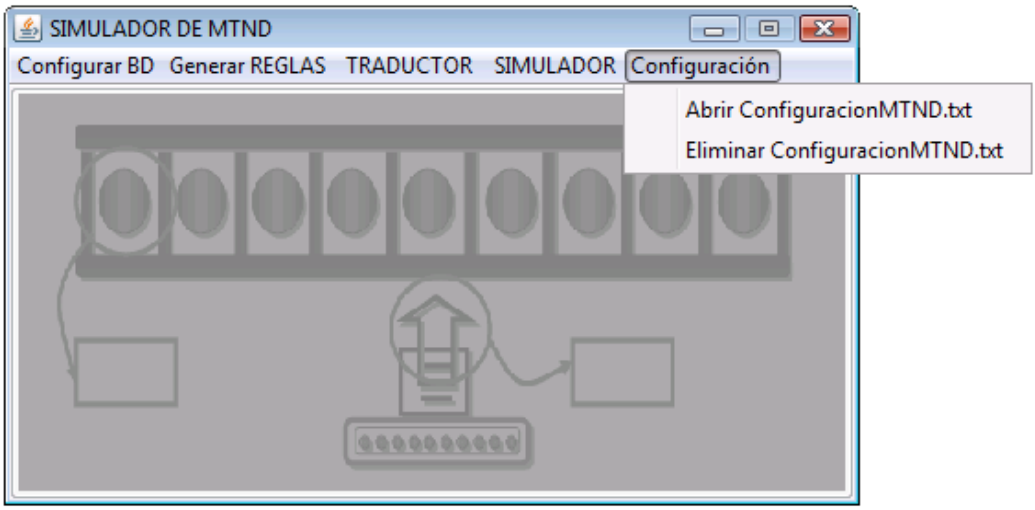


Figura 5.15.- Representación gráfica del menú Configuración.

5.4 Clases que constituyen el simulador de MTND

Las clases que constituyen el simulador de MTND, se representan mediante la figura 5.16.

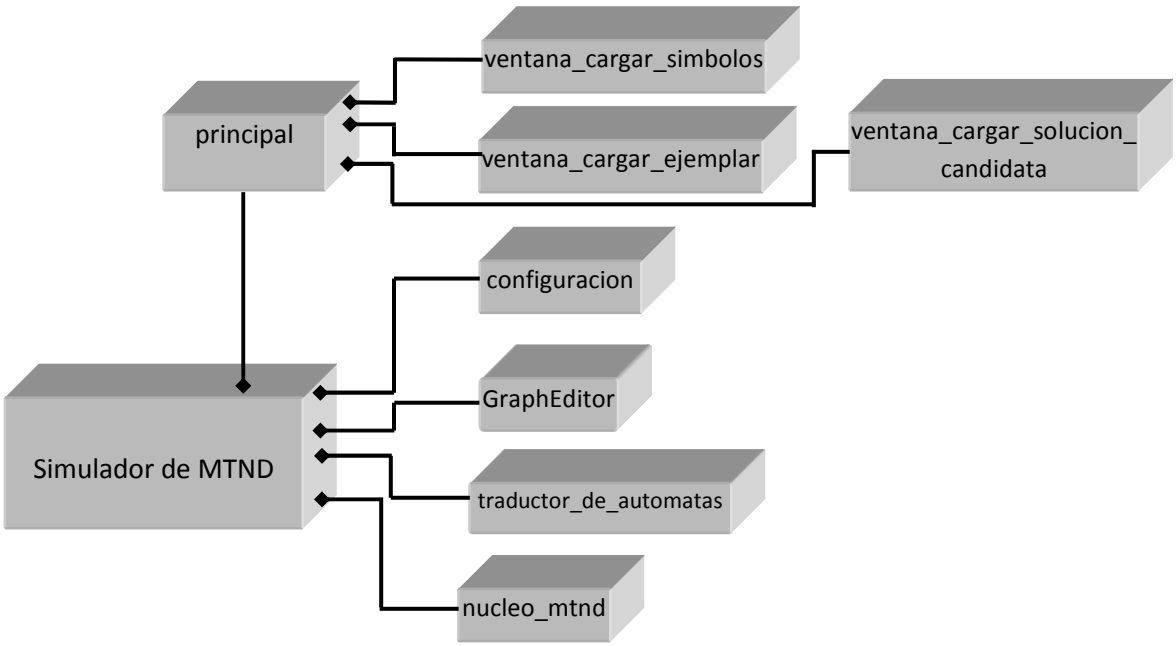


Figura 5.16.- Clases que constituyen parte del simulador de MTND.

Clase “principal”

Esta clase muestra en pantalla la ventana principal del sistema de simulación, la cual contiene una barra de menú. En dicha barra se presentan las opciones con las que cuenta la herramienta.

Clase “ventana_cargar_simbolos”

Esta clase presenta en pantalla la ventana que muestra los campos de edición, los cuales permiten al usuario de la herramienta capturar los símbolos del alfabeto.

Clase “ventana_cargar_ejemplar”

Esta clase presenta en pantalla la ventana que muestra los campos de edición, los cuales permiten al usuario de la herramienta, cargar el archivo que contiene el ejemplar del problema de decisión.

Clase “ventana_cargar_solucion_candidata”

Esta clase presenta en pantalla la ventana que muestra los campos de edición, los cuales permiten al usuario de la herramienta, cargar el archivo que contiene la solución candidata del problema de decisión.

Clase “GraphEditor”

Esta clase es una extensión de la biblioteca JGraph, la cual permite presentar en pantalla un editor para la captura de autómatas. La clase “GraphEditor” hace uso de otras clases, lo cual se explica a detalle en un informe técnico generado durante el desarrollo de este proyecto de tesis.

Clase “configuracion”

Esta clase permite la configuración de la base de datos (BD). Es decir, verifica la existencia de la BD. Si la BD existe envía un mensaje avisando al usuario que la BD ya existe y ofrece la opción de eliminarla; en caso contrario se envía un mensaje al usuario avisando de la no existencia de la BD y se proporciona la opción de crearla.

Clase “traductor_de_automatas”

Esta clase permite leer los símbolos de entrada, validar símbolos, validar estados, insertar datos en las tablas de la base de datos “transiciones”, traducir estados, traducir símbolos y generar reglas de transición completas para los submódulos “MoveHead (-k/L)”, “MoveHead (k/L)”, “MoveHead (-k/R)” y “MoveHead (k/R)”, en base a las posiciones indicadas por el usuario en las reglas de transición abreviadas.

Clase “nucleo_mtnd”

Esta clase permite leer la longitud de la cinta, este dato es proporcionado por el usuario al cargar el ejemplar. Así mismo, permite crear la cinta de la MTND, leer el ejemplar del problema, y presenta una ventana para elegir el archivo que se creó automáticamente al cargar los símbolos de entrada (simbolos_entrada.txt). Dentro de las funciones principales de esta clase se encuentran: leer de cinta, buscar regla con épsilon, extraer regla de transición, escribir en cinta, actualizar posición, actualizar estado e imprimir la configuración de la máquina. Esta clase permite ejecutar tanto el módulo de adivinación, como el módulo de verificación.

CAPÍTULO 6
Pruebas

6.1 Prueba de la MTND con el problema del circuito Hamiltoniano

Se realizaron pruebas con la versión 1 y con la versión 2 del programa simulador. Para ello se utilizaron ejemplares de dos problemas NP-completos, uno de los cuáles es el problema del Circuito Hamiltoniano. El diseño de la MTND de este problema ya fue descrito a detalle en la sección 4.2.

Con la versión 1 del programa simulador, se generaron 97 archivos de texto con reglas de transición abreviadas, correspondientes a los submódulos A, B y C de este problema.

Con la versión 2, mediante un editor de autómatas, se generaron 97 archivos .png correspondientes a los autómatas de los submódulos A, B, y C de este problema.

Con la solución candidata: 4, 3, 5, 2, 1 y el ejemplar b01000b00101b00001b10000b10010 se obtuvo un ejemplar-no (se obtuvo en 5,391 pasos de la MTND).

Con el ejemplar b01000b00101b00001910000b10010 se obtuvo un ejemplar inválido (se obtuvo en 228 pasos de la MTND).

Finalmente con la solución candidata: 1, 2, 3, 5, 4 y el ejemplar b01000b00101b00001b10000b10010 se obtuvo un ejemplar-sí (se obtuvo en 6,658 pasos de la MTND).

Las pruebas realizadas con el problema del Circuito Hamiltoniano resultaron exitosas con 803 reglas abreviadas y un total de 3,834 de reglas de transición generadas en un tiempo de 15

minutos en el proceso de traducción y un tiempo de 3 minutos en la ejecución de la simulación de la MTND. Esto se puede ver a detalle en el informe técnico adjunto a este documento de tesis.

6.2 Prueba de la MTND con el problema de Partición

El otro problema NP-completo, con el cual se realizaron pruebas, es el problema de Partición. El diseño de la MTND de este problema se encuentra descrito en el Anexo A de este documento de tesis.

Con la versión 1 del programa simulador, se generaron 460 archivos de texto con reglas de transición abreviadas, correspondientes a los submódulos A, B y C de este problema.

Con la versión 2, mediante un editor de autómatas, se generaron 460 archivos .png correspondientes a los autómatas de los submódulos A, B y C de este problema.

Con la solución candidata: 1, 1, 0 y el ejemplar 0011b001190010b0101 se obtuvo un ejemplar inválido (se obtuvo en 2,259 pasos de la MTND).

Por último, con la solución candidata: 1, 1, 0 y el ejemplar 0011b0011b0010b0101 se obtuvo un ejemplar-sí (se obtuvo en 23,870 pasos de la MTND).

Las pruebas realizadas con el problema de Partición resultaron exitosas con 6,280 reglas abreviadas y un total de 62,284 de reglas de transición generadas en un tiempo de 180 minutos en el proceso de traducción y un tiempo de 30 minutos en la ejecución de la simulación de la MTND. Esto se puede ver a detalle en el informe técnico adjunto a este documento de tesis.

CAPÍTULO 7
Conclusiones

La herramienta de simulación de este proyecto de tesis, a diferencia de las herramientas del estado del arte, permite resolver problemas NP-completos mediante la introducción de un programa de máquina de Turing no determinista (MTND), ya que aquéllas sólo permiten resolver problemas pequeños. Este programa es introducido de forma modular a través del autómata para el problema de decisión que se desea resolver.

Una de las grandes ventajas de este simulador es que permite definir reglas abreviadas, las cuales, mediante el traductor, pueden ser expandidas a muchas reglas. Por ejemplo, como los submódulos MoveHead que permiten mover la cabeza a una posición determinada (figura 4.15) se

utilizan con mucha frecuencia, el traductor puede generar automáticamente las reglas para dichos submódulos cuando se encuentra una regla en donde el estado tiene alguna de las siguientes formas: MoveHead (*k*/L)Ej?.*s* o MoveHead (*k*/R)Ej?.*s*, en donde *k* representa un número entero positivo o negativo, y *s* representa un "0" o una "f". Este mecanismo permite ahorrar la escritura de más del 50% de las reglas que se necesitan para el procesamiento de un problema de decisión.

El simulador también permite interpretar reglas en donde el símbolo de entrada se puede escribir de alguna de las siguientes maneras:

- $\neq s$, para indicar un símbolo cualquiera diferente de s,
- $s_i,s_j,...,s_m$, para indicar un símbolo cualquiera del conjunto $\{s_i, s_j, ..., s_m\}$,
- $\neq s_i,s_j,...,s_m$, para indicar un símbolo cualquiera que no pertenezca al conjunto $\{s_i, s_j, ..., s_m\}$,
- s_i-s_m, que es equivalente a la siguiente lista $s_i,s_j,...,s_m$, donde $s_i, s_j, ..., s_m$ son símbolos comprendidos en el rango de s_i a s_m, y

$\neq s_i$-s_m, que es equivalente a la siguiente lista $\neq s_i,s_j,...,s_m$, donde $s_i, s_j, ..., s_m$ son símbolos comprendidos en el rango de s_i a s_m

Tomando como base las reglas de transición abreviadas introducidas por el usuario, esta herramienta es capaz de generar reglas de transición de manera automática. Es decir, si para el problema del Circuito Hamiltoniano se introdujeron 803 reglas abreviadas, el programa simulador generó 3,834 reglas de transición, y para el problema de Partición de 6,280 reglas abreviadas se generaron automáticamente 62,284 reglas de transición.

Otra de las grandes ventajas de este simulador es que tiene dos mecanismos de detección de dos tipos de errores: cuando la cabeza intenta moverse más a la izquierda de la posición $-max_{izq}$ (definida por el usuario como la posición límite del lado izquierdo de la cinta), o intenta moverse más a la derecha de la posición max_{der}, (definida como la posición límite del lado derecho), o cuando el número de veces que un estado es visitado excede un número definido por el usuario. A semejanza de cualquier programa codificado en un lenguaje de alto nivel, donde suele ocurrir que el programa intente acceder a una posición de memoria fuera de los límites previstos (por ejemplo de un arreglo) o el programa se cicle; también estos problemas pueden ocurrir en un autómata. Los dos mecanismos de detección de errores del simulador facilitan la detección y corrección de errores, los cuales serían extremadamente difíciles de corregir sin estos mecanismos.

En el área de complejidad computacional, esta herramienta permitirá a catedráticos y estudiantes conocer de manera práctica la solución de problemas NP-completos utilizando MTNDs. Es importante mencionar que los libros de texto presentan ejemplos extremadamente pequeños que no permiten a los estudiantes comprender a cabalidad el funcionamiento de una MTND. Por ejemplo, en el libro [Garey & Johnson, 1979] se presenta un ejemplo muy pequeño para una máquina de Turing determinista, pero ninguno para una MTND.

El estudio de las herramientas de simulación existentes (presentadas en el Capítulo 3) revela que ninguna de tales herramientas tiene la capacidad para resolver problemas tan grandes como los dos resueltos (Circuito Hamiltoniano y Partición) por el simulador desarrollado en este proyecto de tesis.

ANEXO A
Diseño del autómata del problema de Partición

El problema de Partición es el segundo problema utilizado para probar el simulador de una MTND. El autómata para el problema de Partición está diseñado para un simulador que esté basado en la definición de MTND propuesta por [Garey & Johnson, 1979]. Este problema de decisión se define de la siguiente manera:

Considere un conjunto finito A de elementos y un tamaño $s(a) \in Z^+$ para cada elemento $a \in A$ (donde Z^+ denota el conjunto de enteros positivos), entonces el problema de Partición consiste en determinar si existe un subconjunto $A' \subset A$ tal que $\Sigma_{a \in A'} s(a) = \Sigma_{a \in A - A'} s(a)$.

El problema de Partición formulado como un problema de decisión se puede expresar de la siguiente manera:

Ejemplar: un conjunto A de elementos.

Pregunta: ¿existe un subconjunto S y un subconjunto T de un conjunto A tal que S sea igual a T, y $A = S \cup T$?

Además, este problema se puede formular matemáticamente de la siguiente forma:

$$¿S = T?$$

donde

$$S = \sum_{i=1}^{n} s(a_i) x_i$$

$$T = \sum_{i=1}^{n} s(a_i)(1 - x_i)$$

$x_i = 0$ o 1.

Para ilustrar la aplicación de la formulación anterior, considérese un ejemplar-sí p de Partición definido como sigue: un conjunto de elementos $A = \{a_1, a_2, a_3\}$ con sus respectivos tamaños $s(a_1)=3$, $s(a_2)=2$, $s(a_3)=5$. En este ejemplo, la formulación para el ejemplar p es la siguiente:

$$S \stackrel{?}{=} T$$

donde

$$S = 3x_1 + 2x_2 + 5x_3$$
$$T = 3(1–x_1) + 2(1–x_2) + 5(1–x_3)$$
$$x_i = 0 \text{ o } 1.$$

En vista de lo anterior, el objetivo de este ejemplo de prueba es diseñar una MTND para el ejemplo de la formulación anterior; es decir, determinar si existe un subconjunto *S* y un subconjunto *T* del conjunto *A* = {3, 2, 5} tal que *S* sea igual a *T*, y *A* = *S* ∪ *T*.

Notas: Este ejemplo está diseñado utilizando números en el sistema binario de cuatro dígitos; por lo tanto, el conjunto *A* del ejemplo se debe expresar de la siguiente manera: *A* = {0011, 0010, 0101}. El diseño del módulo de verificación de la MTND para el problema de Partición será descrito a nivel de pseudocódigo.

Tomando como base el conjunto *A*, el problema puede codificarse en la cinta como se muestra en la figura A.1.

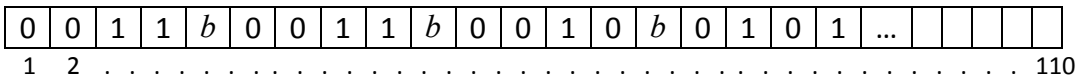


Figura A.1.- Problema codificado en la cinta.

Como se puede observar en la figura A.1 los primeros 4 bits representan el valor de 3 en binario, los cuales corresponden al número de elementos del conjunto *A*, los tres valores siguientes después de cada "*b*" corresponden a cada uno de los tamaños $s(a_i)$ de los elementos del conjunto *A*.

A.1 El módulo de adivinación

El módulo de adivinación selecciona aleatoriamente del conjunto {0, 1}. La cantidad de 0's y 1's elegidos del conjunto dependerá del número de elementos que contenga el conjunto *A*. Esta secuencia constituirá la solución candidata del problema. El módulo emplea un autómata finito no determinista el cual se ilustra en la figura A.2.

En el autómata finito no determinista (AFND) se emplea la notación $s_1 \rightarrow s_2, m$ para representar las acciones de lectura, escritura y movimiento.

1.- s_1 representa el símbolo que lee el módulo de adivinación en la celda actual. Cuando s_1=ε , significa que la cabeza no efectúa lectura alguna; es decir, efectúa una transición espontáneamente.

2.- s_2 representa el símbolo que la cabeza escribirá en la celda actual. El símbolo ∅ indica que la cabeza no efectúa escritura alguna.

3.- *m* representa el movimiento de la cabeza. Si *m* = +1, se mueve una celda a la derecha, si *m* = −1, se mueve una celda a la izquierda, si *m* = 0 entonces no realiza movimiento; es decir, se mantiene en la misma celda.

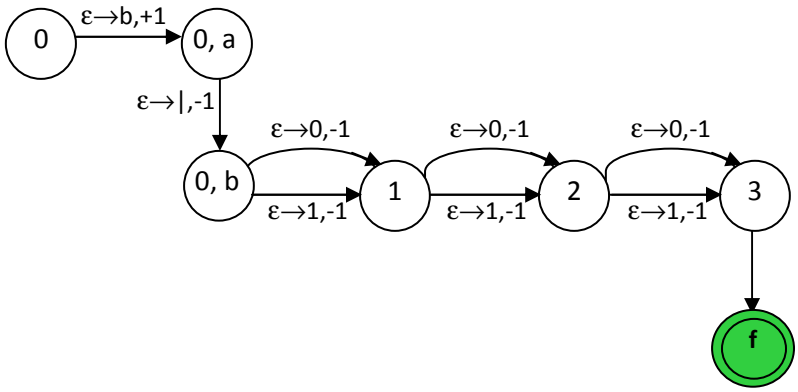


Figura A.2.- Diagrama del AFND del módulo de adivinación.

La cabeza lectora del módulo de adivinación se ubica en la posición cero de la cinta y escribe el símbolo "|" utilizado como delimitador para facilitar el cálculo del programa verificador.

En la figura A.2 se observa cómo la transición del estado $q_{0,a}$ al estado $q_{0,b}$ ocasiona que la cabeza espontáneamente (sin leer de la cinta) escriba el símbolo "|" y se mueva una celda a la izquierda. Esta transición ocurre cuando la MTND se encuentra en la celda cero de la cinta. Las siguientes tres transiciones del AFND sirven para elegir aleatoriamente un valor del conjunto {0, 1} e imprimirlo en la cinta. En este punto conviene mencionar que los valores generados por el AFND son los valores de las variables x_1, x_2 y x_3 de la solución candidata, lo cual significa que la solución candidata que se muestra en la figure A.3 está dada por la siguiente asignación de valores: x_1,.=1, x_2,.=1, x_3,.=0.

0	1	1	|	0	0	1	1	*b*	0	0	1	1	*b*	0	0	1	0	*b*	0	1	0	1	…		
-3				0					5					10					15							75

Figura A.3.- Contenido de la cinta al finalizar el módulo de adivinanza.

A.2 El módulo de verificación

El módulo de verificación debe realizar las siguientes comprobaciones:

1. Comprobar que el ejemplar es válido.
2. Comprobar que la suma de los tamaños en binario no rebase el número más grande que se representa con 4 bits, decir 15.
3. Comprobar que *S* sea igual a *T*.

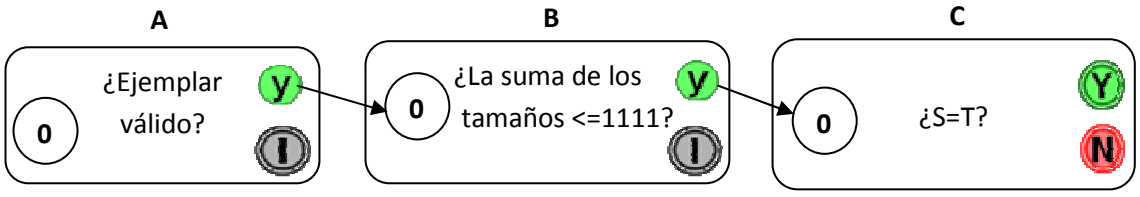


Figura A.4.- Diagrama de submódulos de las tareas del módulo de verificación.

A.2.1 ¿Ejemplar válido?

El siguiente pseudocódigo tiene como objetivo validar el ejemplar del problema:

Definir i = cinta(112-115), n = cinta(1-4), s_1 = cinta(5-9), s_2 = cinta(10-14), ..., s_{15} = cinta(75-79), N = cinta (82-85), donde s_i representa el tamaño del elemento a_i.

1. Hacer i=0001.
2. Hacer $N = n + 1$.
3. Si n=0 ir al estado “I”, en caso contrario ir al paso 4.
4. Si n válida ir al paso 5, en caso contrario ir al estado "I".
5. Si tamaño s_i válido ir al paso 6, en caso contrario ir al estado "I".
6. Hacer $i = i + 1$.
7. Si i=N ir al estado "y", en caso contrario ir al paso 5.

Validación del ejemplar: Paso 1

Como podemos observar en el pseudocódigo, se requerirán movimientos hacia las posiciones de cada una de las variables que se han definido. Para poder llevar a cabo esos movimientos haremos uso de los submódulos genéricos (descritos en la sección 4.2.2.1) que posicionan a la cabeza en alguna celda de la cinta.

En relación al paso 1 y haciendo uso de los submódulos de posicionamiento, la cabeza se mueve a la posición 115 de la cinta y se escribe el valor 0001 en la variable i (de la posición 112 a la posición 115).

Validación del ejemplar: Paso 2

Este paso tiene por objeto guardar en la variable N un valor igual a n+1. Para llevar a cabo este paso la cabeza se posiciona en 1, ya que es a partir de la posición 1 donde se encuentran los bits de la variable n. Posteriormente se leen los dígitos almacenados en las celdas de la 1 a la 4 mediante el uso de un submódulo denominado decodificador binario. Este submódulo determina el valor V de un número binario codificado en cuatro bits, donde V es igual a $8b_1 + 4b_2 + 2b_3 + b_4$, siendo b_1, b_2, b_3 y b_4 los valores del 1ro, 2do, 3er y 4to bits del número en cuestión. Una vez determinado el valor del número binario que se encuentra en la variable n, se mueve la cabeza a la posición 82 y se escribe a partir de esta posición hasta la posición 85 (donde se encuentran los bits

de la variable *N*) un número binario igual al valor de *n* más uno. Por ejemplo, si el valor de *n* fuera 0001 escribiría 0010, si el valor de *n* fuera 0010 escribiría 0011, y así sucesivamente.

Validación del ejemplar: Paso 3

El paso 3 tiene como fin verificar que el ejemplar a resolver tenga al menos un elemento; es decir, $n \neq 0$. Para tal efecto, la cabeza se debe mover a la posición 1 y leer todos los símbolos a partir de esta posición hasta la 4, que es donde se encuentran los bits de *n*; en caso de que todos los bits leídos sean 0, el autómata pasa al estado "I", y en caso contrario continúa en el paso 4.

Validación del ejemplar: Paso 4

Este paso tiene por objeto verificar que el valor de *n* sea válido, lo cual se comprueba revisando que todos los símbolos almacenados en las celdas de la 1 a la 4 sean dígitos binarios (0 o 1). Para comprobar si el valor de *n* es válido, la cabeza lecto-escritora se deberá ir a la posición 1, y a partir de esta celda hasta la celda 4 se verificará que los símbolos almacenados sean 0s o 1s. Si *n* es válida se continúa con el paso 5, en caso contrario se debe pasar a un estado inválido "I".

Validación del ejemplar: Paso 5

El paso 5 se encarga de verificar que el valor de la variable s_i sea válido para cada $i = 1, 2, ..., n$. Para determinar la posición de la variable s_i en la cinta, primero se necesita determinar al valor de la variable *i*. Para tal efecto, es necesario mover la cabeza a la posición 112 y leer los dígitos almacenados en las celdas de la 112 a la 115 mediante el uso de un decodificador binario. Una vez determinado el valor de *i*, la cabeza se mueve a la posición 5 si $i=1$, a la posición 10 si $i=2$, y así sucesivamente hasta la posición 75 si $i=15$. Finalmente se verifica que el valor de s_i sea válido, y para tal efecto se debe comprobar que en las celdas correspondientes a s_i la primera celda contenga un símbolo blanco "b" y las 4 celdas siguientes contengan dígitos binarios (0 o 1); en caso de que no se cumpla alguna de las condiciones anteriores se deberá ir a un estado inválido.

Validación del ejemplar: Paso 6

En este paso se incrementa en 1 el valor de la variable *i*. El proceso de este paso es similar al del paso 2, con la diferencia de que en lugar de la variable *N* se debe considerar *i* (la cual se encuentra almacenada en las posiciones de la 112 a la 115), y en lugar de la variable *n* se debe considerar *i*.

Validación del ejemplar: Paso 7

Para determinar si $i=N$ este submódulo debe comparar bit por bit los valores de las variables *i* y *N*. Para tal efecto primeramente la cabeza se debe posicionar en 112, que es donde se encuentra el primer bit de *i*, enseguida debe leer el valor almacenado (0 o 1); después la cabeza se debe posicionar en 82, que es donde se encuentra el primer bit de *N*, y enseguida debe leer el valor almacenado (0 o 1); si los dos valores leídos son diferentes, entonces $i \neq N$ y se deberá regresar al

paso 5; en caso contrario se procede a comparar el segundo bit de *i* (posición 113) contra el segundo bit de *N* (posición 83), y así sucesivamente con los demás bits. Si todos los bits de *i* resultan iguales a todos los bits de *N*, entonces *i*=*N* y se pasa a un estado "y", en caso contrario se deberá regresar al paso 5.

A.2.2 ¿Suma de los tamaños ≤ 1111?

El siguiente pseudocódigo tiene como objetivo verificar que la suma de los tamaños sea ≤ 1111:

Definir *n* = cinta(1-4), *i* = cinta(112-115), s_1 = cinta(6-9), s_2 = cinta(11-14), ..., s_{15} = cinta(76-79), *sumando*1 = cinta(87-90), *sumando*2 = cinta(92-95), *suma* = cinta(97-100), donde s_i representa el tamaño del elemento a_i.

1. Si *n*=0001 ir al estado "y", en caso contrario ir al paso 2.
2. Hacer *i*=0010.
3. Hacer *sumando*1 = s_1.
4. Hacer *sumando*2 = s_2.
5. Hacer *suma* = *sumando*1 + *sumando*2, y si *suma* > 1111 ir al estado "l".
6. Si *i*=*n* ir al estado "y", en caso contrario ir al paso 7.
7. Hacer *i* = *i* + 1.
8. Hacer *sumando*1 = *suma*.
9. Hacer *sumando*2 = s_i.
10. Hacer *suma* = *sumando*1 + *sumando*2, y si suma > 1111 ir al estado "l", en caso contrario ir al paso 6.

Suma ≤ 1111: Paso 1

Para determinar si *n* es igual a 0001, la cabeza de lectura-escritura debe posicionarse en 1 para leer los bits almacenados de la celda 1 a la 4. Si los bits leídos son 0, 0, 0 y 1, se pasa a un estado "y" (ya que *n*=1 implica que suma≤1111); en caso contrario se continúa con el paso 2.

Suma ≤1111: Paso 2

La cabeza se posiciona en 112 y a partir de la posición 112 hasta la posición 115 se escribe el valor 0010.

Suma ≤ 1111: Paso 3

Este paso tiene como fin copiar el valor de s_1 a *sumando*1. Para tal efecto, la cabeza se debe posicionar en 6 para leer los dígitos que se encuentran en las posiciones de la 6 a la 9, ya que la variable s_1 se encuentra en dichas posiciones. Posteriormente mediante un decodificador binario (descrito en el paso 2 de la subsección A.2.1) se determina el valor del número almacenado en las posiciones de la 6 a la 9. Finalmente, la cabeza se debe mover a la posición 87 para escribir en las posiciones de la 87 a la 90 (correspondientes a la variable *sumando*1) un número binario cuyo valor sea igual al leído previamente.

Suma ≤ 1111: Paso 4

Este paso se encarga de copiar el valor de s_2 a *sumando*2. El proceso de este paso es similar al del anterior, con la diferencia de que en lugar de la variable s_1 se debe considerar s_2 (la cual se encuentra almacenada en las posiciones de la 11 a la 14), y en lugar de la variable *sumando*1 se debe considerar *sumando*2 (almacenada en las celdas de la 92 a la 95).

Suma ≤ 1111: Paso 5

El propósito de este paso es sumar *sumando*1 y *sumando*2 y almacenar el resultado en la variable *suma*. En este paso se hace uso de un submódulo denominado sumador binario, el cual funciona de manera similar al autómata de la MTD descrito al principio del Capítulo 4. Dicho sumador primero mueve la cabeza a la celda 87, enseguida lee los bits que se encuentran en las posiciones de la 87 a la 90 (correspondientes a la variable *sumando*1) y determina un valor σ igual a $8b_{s1,1} + 4b_{s1,2} + 2b_{s1,3} + b_{s1,4}$, donde $b_{s1,1}$, $b_{s1,2}$, $b_{s1,3}$ y $b_{s1,4}$ son los valores del 1ro, 2do, 3er y 4to bits de *sumando*1. Después el sumador mueve la cabeza a la celda 92, enseguida lee los bits que se encuentran en las posiciones de la 92 a la 95 (correspondientes a la variable *sumando*2) y determina un valor σ igual a $(8b_{s1,1} + 4b_{s1,2} + 2b_{s1,3} + b_{s1,4}) + (8b_{s2,1} + 4b_{s2,2} + 2b_{s2,3} + b_{s2,4})$, donde $b_{s2,1}$, $b_{s2,2}$, $b_{s2,3}$ y $b_{s2,4}$ son los valores del 1ro, 2do, 3er y 4to bits de *sumando*2. Una vez determinado el valor σ, la cabeza se mueve a la celda 97 para escribir en las posiciones de la 97 a la 100 (correspondientes a la variable *suma*) un número binario cuyo valor sea igual al de σ. Cada vez que el sumador calcula una suma mayor que 1111 pasa a un estado "n" y más adelante termina en un estado "I", en caso contrario continúa al siguiente paso.

Suma ≤ 1111: Paso 6

Para determinar si $i=n$ este submódulo debe comparar bit por bit los valores de las variables *i* y *n*. Para tal efecto primeramente la cabeza se debe posicionar en 112, que es donde se encuentra el primer bit de *i*, enseguida debe leer el valor almacenado (0 o 1); después la cabeza se debe posicionar en 1, que es donde se encuentra el primer bit de *n*, y enseguida debe leer el valor almacenado (0 o 1); si los dos valores leídos son diferentes, entonces $i \neq n$ y se deberá ir al paso 7; en caso contrario se procede a comparar el segundo bit de *i* (posición 115) contra el segundo bit de *n* (posición 2), y así sucesivamente con los demás bits. Si todos los bits de *i* resultan iguales a todos los bits de *n*, entonces $i=n$ y se pasa a un estado "y", en caso contrario se deberá continuar al paso 7.

Suma ≤ 1111: Paso 7

En este paso se incrementa en 1 el valor de la variable *i*. El proceso de este paso es similar al del paso 2 descrito en la subsección A.2.1, con la diferencia de que en lugar de la variable *N* se debe considerar *i* (la cual se encuentra almacenada en las posiciones de la 112 a la 115), y en lugar de la variable *n* se debe considerar *i*.

Suma ≤ 1111: Paso 8

El propósito de este paso es copiar el valor de *suma* a *sumando*1. El proceso de este paso es semejante al del paso 3, con la diferencia de que en lugar de la variable s_1 se debe considerar la variable *suma* (la cual se encuentra almacenada en las posiciones de la 97 a la 100).

Suma ≤ 1111: Paso 9

El propósito de este paso es copiar el valor de s_i a *sumando*2. Para determinar la posición de la variable s_i en la cinta, primero se necesita determinar al valor de la variable *i*. Para tal efecto, es necesario mover la cabeza a la posición 112 y leer los dígitos almacenados en las celdas de la 112 a la 115 mediante el uso de un decodificador binario (descrito en el paso 2 de la subsección A.2.1). Una vez determinado el valor de *i*, la cabeza se mueve a la posición 6 si *i*=1, a la posición 11 si *i*=2, y así sucesivamente hasta la posición 76 si *i*=15. Después la cabeza debe leer los bits que se encuentran en las posiciones de la 5*i*+1 a la 5*i*+4, ya que la variable s_i se encuentra en dichas posiciones. Posteriormente mediante un decodificador binario se determina el valor del número almacenado en las posiciones de la 5*i*+1 a la 5*i*+4. Finalmente, la cabeza se debe mover a la posición 92 para escribir en las posiciones de la 92 a la 95 (correspondientes a la variable *sumando*2) un número binario cuyo valor sea igual al leído previamente.

Suma ≤ 1111: Paso 10

El propósito de este paso es sumar *sumando*1 y *sumando*2 y almacenar el resultado en la variable *suma*. El proceso de este paso es semejante al del paso 5, con la diferencia de que cada vez que el sumador calcula una suma mayor que 1111 pasa a un estado "n" y más adelante termina en un estado "I", en caso contrario regresa al paso 6.

A.2.3 ¿S = T?

El siguiente pseudocódigo tiene como objetivo verificar que S sea igual a T:

Definir *n* = cinta(1-4), *i* = cinta(112-115), s_1 = cinta(6-9), s_2 = cinta(11-14), ..., s_{15} = cinta(76-79), *sumando*1 = cinta(87-90), *sumando*2 = cinta(92-95), *suma* = cinta(97-100), x_1 = cinta(–1), x_2 = cinta(–2), ..., x_{15} = cinta(–15), *suma_S* = cinta(102-105), suma_T = cinta(107-110).

1. Si *n*=0001 ir al estado "N", en caso contrario ir al paso 2.
2. Hacer *i* = 0010.
3. Si x_1 = 1 ir al paso 3.1, en caso contrario ir al paso 3.2.

3.1. Hacer *sumando*1 = s_1, ir al paso 4.

3.2. Hacer *sumando*1 = 0, ir al paso 4.

4. Si x_2 = 1 ir al paso 4.1, en caso contrario ir al paso 4.2.

4.1. Hacer *sumando*2 = s_2, ir al paso 5.

4.2. Hacer *sumando*2 = 0, ir al paso 5.

5. Hacer *suma* = *sumando*1 + *sumando*2.
6. Si *i* = *n* ir al paso 12, en caso contrario ir al paso 7.

7. Hacer $i = i + 1$.
8. Hacer *sumando*1 = *suma*.
9. Si $x_i = 1$ ir al paso 9.1, en caso contrario ir al paso 9.2.
9.1. Hacer *sumando*2 = s_i, ir al paso 10.
9.2. Hacer *sumando*2 = 0, ir al paso 10.
10. Hacer *suma* = *sumando*1 + *sumando*2.
11. Hacer *suma_S* = *suma*, ir al paso 6.
12. Hacer *i* = 0010.
13. Si $x_1 = 0$ ir al paso 13.1, en caso contrario ir al paso 13.2.
13.1. Hacer *sumando*1 = s_1, ir al paso 14.
13.2. Hacer *sumando*1 = 0, ir al paso 14.
14. Si $x_2 = 0$ ir al paso 14.1, en caso contrario ir al paso 14.2.
14.1. Hacer *sumando*2 = s_2, ir al paso 15.
14.2. Hacer *sumando*2 = 0, ir al paso 15.
15. Hacer *suma* = *sumando*1 + *sumando*2.
16. Si $i = n$ ir al paso 22, en caso contrario ir al paso 17.
17. Hacer $i = i + 1$.
18. Hacer *sumando*1 = *suma*.
19. Si $x_i = 0$ ir al paso 19.1, en caso contrario ir al paso 19.2.
19.1. Hacer *sumando*2 = s_i, ir al paso 20.
19.2. Hacer *sumando*2 = 0, ir al paso 20.
20. Hacer *suma* = *sumando*1 + *sumando*2.
21. Hacer *suma_T* = *suma*, ir al paso 16.
22. Si *suma_S* = *suma_T* ir al estado "Y", en caso contrario ir al estado "N".

***S* = *T*: Paso 1**

Para determinar si *n* es igual a 0001, la cabeza de lectura-escritura debe posicionarse en 1 para leer los bits almacenados de la celda 1 a la 4. Si los bits leídos son 0, 0 , 0 y 1, se pasa a un estado "N" (ya que $n=1$ implica que $S \neq T$); en caso contrario se continúa con el paso 2.

***S* = *T*: Paso 2**

La cabeza se posiciona en 112 y a partir de la posición 112 hasta la posición 115 se escribe el valor 0010.

***S* = *T*: Paso 3**

Para determinar si x_1 es igual a 1 la cabeza se debe posicionar en –1, ya que en la posición –1 de la cinta se encuentra la variable x_1 de la solución candidata. Si x_1 es igual a 1 ir al paso 3.1 o si $x_1=0$ ir al paso 3.2.

***S* = *T*: Paso 3.1**

Este paso tiene como fin copiar el valor de s_1 a *sumando*1. Para tal efecto, la cabeza se debe posicionar en 6 para leer los dígitos que se encuentran en las posiciones de la 6 a la 9, ya que la variable s_1 se encuentra en dichas posiciones. Posteriormente mediante un decodificador binario

(descrito en el paso 2 de la subsección A.2.1) se determina el valor del número almacenado en las posiciones de la 6 a la 9. Finalmente, la cabeza se debe mover a la posición 87 para escribir en las posiciones de la 87 a la 90 (correspondientes a la variable *sumando*1) un número binario cuyo valor sea igual al leído previamente.

S = *T*: Paso 3.2

Primero la cabeza se posiciona en 87, ya que la variable *sumando*1 se encuentra de la posición 87 a la posición 90 de la cinta, y enseguida la cabeza escribe un símbolo 0 en cada una de dichas posiciones.

***S* = *T*: Paso 4**

Para determinar si x_2 es igual a 1 la cabeza se debe posicionar en –2, ya que en la posición –2 de la cinta se encuentra la variable x_2 de la solución candidata. Si x_2 es igual a 1 ir al paso 4.1 o si x_2=0 ir al paso 4.2.

***S* = *T*: Paso 4.1**

Este paso tiene como fin copiar el valor de s_2 a *sumando*2. El proceso de este paso es semejante al del paso 3.1, con la diferencia de que en lugar de la variable s_1 se debe considerar la variable s_2 (la cual se encuentra almacenada en las posiciones de la 11 a la 14), y en lugar de la variable *sumando*1 se debe considerar *sumano*2 (la cual se encuentra almacenada en las posiciones de la 92 a la 95).

***S* = *T*: Paso 4.2**

El propósito de este paso es escribir el valor 0 en la variable *sumando*2. Para tal efecto, primero la cabeza se posiciona en 92, ya que la variable *sumando*2 se encuentra de la posición 92 a la posición 95 de la cinta, y enseguida la cabeza escribe un símbolo 0 en cada una de dichas posiciones.

***S* = *T*: Paso 5**

El propósito de este paso es sumar *sumando*1 y *sumando*2 y almacenar el resultado en la variable *suma*. En este paso se hace uso de un submódulo denominado sumador binario, el cual funciona de manera similar al autómata de la MTD descrito al principio del Capítulo 4. Dicho sumador primero mueve la cabeza a la celda 87, enseguida lee los bits que se encuentran en las posiciones de la 87 a la 90 (correspondientes a la variable *sumando*1) y determina un valor σ igual a $8b_{s1,1} + 4b_{s1,2} + 2b_{s1,3}+ b_{s1,4}$, donde $b_{s1,1}$, $b_{s1,2}$, $b_{s1,3}$ y $b_{s1,4}$ son los valores del 1ro, 2do, 3er y 4to bits de *sumando*1. Después el sumador mueve la cabeza a la celda 92, enseguida lee los bits que se encuentran en las posiciones de la 92 a la 95 (correspondientes a la variable *sumando*2) y determina un valor σ igual a $(8b_{s1,1} + 4b_{s1,2} + 2b_{s1,3}+ b_{s1,4}) + (8b_{s2,1} + 4b_{s2,2} + 2b_{s2,3}+ b_{s2,4})$, donde $b_{s2,1}$, $b_{s2,2}$, $b_{s2,3}$ y $b_{s2,4}$ son los valores del 1ro, 2do, 3er y 4to bits de *sumando*2. Una vez determinado el valor σ, la cabeza se mueve a la celda 97 para escribir en las posiciones de la 97 a la 100 (correspondientes a la variable *suma*) un número binario cuyo valor sea igual al de σ.

***S* = *T*: Paso 6**

Este paso se encarga de determinar si $i=n$, y si esta condición se cumple se pasa al paso 12, en caso contrario continúa al paso 7. Para determinar si $i=n$, este submódulo debe comparar bit por bit los valores de las variables *i* y *n*. Para tal efecto primeramente la cabeza se debe posicionar en 112, que es donde se encuentra el primer bit de *i*, enseguida debe leer el valor almacenado (0 o 1); después la cabeza se debe posicionar en 1, que es donde se encuentra el primer bit de *n*, y enseguida debe leer el valor almacenado (0 o 1); si los dos valores leídos son diferentes, entonces $i\neq n$ y se deberá ir al paso 7; en caso contrario se procede a comparar el segundo bit de *i* (posición 113) contra el segundo bit de *n* (posición 2), y así sucesivamente con los demás bits. Si todos los bits de *i* resultan iguales a todos los bits de *n*, entonces $i=n$.

***S* = *T*: Paso 7**

En este paso se incrementa en 1 el valor de la variable *i*. El proceso de este paso es similar al del paso 2 descrito en la subsección A.2.1, con la diferencia de que en lugar de la variable *N* se debe considerar *i* (la cual se encuentra almacenada en las posiciones de la 112 a la 115), y en lugar de la variable *n* se debe considerar *i*.

***S* = *T*: Paso 8**

El propósito de este paso es copiar el valor de *suma* a *sumando*1. El proceso de este paso es semejante al del paso 3.1, con la diferencia de que en lugar de la variable s_1 se debe considerar la variable *suma* (la cual se encuentra almacenada en las posiciones de la 97 a la 100).

***S* = *T*: Paso 9**

Este paso tiene como fin determinar si $x_i = 1$ para decidir si agregar o no la variable s_i a la suma *S*. Para determinar la posición de la variable x_i en la cinta, primero se necesita determinar al valor de la variable *i*. Para tal efecto, es necesario mover la cabeza a la posición 112 y leer los dígitos almacenados en las celdas de la 112 a la 115 mediante el uso de un decodificador binario (descrito en el paso 2 de la subsección A.2.1). Una vez determinado el valor de *i*, la cabeza se mueve a la posición –1 si $i=1$, a la posición –2 si $i=2$, y así sucesivamente hasta la posición –15 si $i=15$. Después la cabeza debe leer el bit que se encuentra en la posición $-i$, ya que la variable x_i se encuentra en dicha posición. Finalmente, si el bit leído en dicha posición es 1 se continúa en el paso 9.1, y en caso contrario se continúa en el paso 9.2.

***S* = *T*: Paso 9.1**

El propósito de este paso es copiar el valor de s_i a *sumando*2. Para determinar la posición de la variable s_i en la cinta, primero se necesita determinar al valor de la variable *i*. Para tal efecto, es necesario mover la cabeza a la posición 112 y leer los dígitos almacenados en las celdas de la 112 a la 115 mediante el uso de un decodificador binario (descrito en el paso 2 de la subsección A.2.1). Una vez determinado el valor de *i*, la cabeza se mueve a la posición 6 si $i=1$, a la posición 11 si $i=2$, y así sucesivamente hasta la posición 76 si $i=15$. Después la cabeza debe leer los bits que se encuentran en las posiciones de la $5i+1$ a la $5i+4$, ya que la variable s_i se encuentra en dichas posiciones. Posteriormente mediante un decodificador binario se determina el valor del número almacenado en las posiciones de la $5i+1$ a la $5i+4$. Finalmente, la cabeza se debe mover a la

posición 92 para escribir en las posiciones de la 92 a la 95 (correspondientes a la variable *sumando*2) un número binario cuyo valor sea igual al leído previamente.

***S* = *T*: Paso 9.2**

El propósito de este paso es escribir el valor 0 en la variable *sumando*2. El proceso de este paso es igual que el del paso 4.2.

***S* = *T*: Paso 10**

Este paso tiene como fin sumar *sumando*1 y *sumando*2 y almacenar el resultado en la variable *suma*. El proceso de este paso es igual que el del paso 5.

***S* = *T*: Paso 11**

Este paso se encarga de copiar el valor de la variable *suma* a la variable *suma_S*. El proceso de este paso es parecido al del paso 3.1, con la diferencia de que en lugar de la variable s_1 se considera la variable *suma* (la cual se encuentra almacenada en las posiciones de la 97 a la 100) y en lugar de la variable *sumando*1 se considera la variable *suma_S* (la cual se encuentra almacenada en las posiciones de la 102 a la 105).

***S* = *T*: Paso 12**

La cabeza se posiciona en 112 y a partir de la posición 112 hasta la posición 115 se escribe el valor 0010.

***S* = *T*: Paso 13**

Para determinar si x_1 es igual a 0 la cabeza se debe posicionar en –1, ya que en la posición –1 de la cinta se encuentra la variable x_1 de la solución candidata. Si x_1 es igual a 0 ir al paso 13.1 o si x_1=1 ir al paso 13.2.

***S* = *T*: Paso 13.1**

Este paso tiene como fin copiar el valor de s_1 a *sumando*1. El proceso de este paso es igual que el del paso 3.1.

***S* = *T*: Paso 13.2**

Primero la cabeza se posiciona en 87, ya que la variable *sumando*1 se encuentra de la posición 87 a la posición 90 de la cinta, y enseguida la cabeza escribe un símbolo 0 en cada una de dichas posiciones.

***S* = *T*: Paso 14**

Para determinar si x_2 es igual a 0 la cabeza se debe posicionar en –2, ya que en la posición –2 de la cinta se encuentra la variable x_2 de la solución candidata. Si x_2 es igual a 0 ir al paso 14.1 o si x_2=1 ir al paso 14.2.

***S* = *T*: Paso 14.1**

Este paso tiene como fin copiar el valor de s_2 a *sumando*2. El proceso de este paso es igual que el del paso 4.1.

***S* = *T*: Paso 14.2**

El propósito de este paso es escribir el valor 0 en la variable *sumando*2. El proceso de este paso es igual que el del paso 4.2.

***S* = *T*: Paso 15**

El propósito de este paso es sumar *sumando*1 y *sumando*2 y almacenar el resultado en la variable *suma*. El proceso de este paso es igual que el del paso 5.

***S* = *T*: Paso 16**

Este paso se encarga de determinar si $i=n$, y si esta condición se cumple se pasa al paso 22, en caso contrario continúa al paso 17. Para determinar si $i=n$, este submódulo debe realizar un proceso igual que el del paso 6.

***S* = *T*: Paso 17**

En este paso se incrementa en 1 el valor de la variable *i*. El proceso de este paso es similar al del paso 2 descrito en la subsección A.2.1, con la diferencia de que en lugar de la variable *N* se debe considerar *i* (la cual se encuentra almacenada en las posiciones de la 112 a la 115), y en lugar de la variable *n* se debe considerar *i*.

***S* = *T*: Paso 18**

El propósito de este paso es copiar el valor de *suma* a *sumando*1. El proceso de este paso es igual que el del paso 8.

***S* = *T*: Paso 19**

Este paso tiene como fin determinar si $x_i = 0$ para decidir si agregar o no la variable s_i a la suma *T*. Para determinar la posición de la variable x_i en la cinta, primero se necesita determinar al valor de la variable *i*. Para tal efecto, es necesario mover la cabeza a la posición 112 y leer los dígitos almacenados en las celdas de la 112 a la 115 mediante el uso de un decodificador binario (descrito en el paso 2 de la subsección A.2.1). Una vez determinado el valor de *i*, la cabeza se mueve a la posición –1 si $i=1$, a la posición –2 si $i=2$, y así sucesivamente hasta la posición –15 si $i=15$. Después la cabeza debe leer el bit que se encuentra en la posición $-i$, ya que la variable x_i se encuentra en dicha posición. Finalmente, si el bit leído en dicha posición es 0 se continúa en el paso 19.1, y en caso contrario se continúa en el paso 19.2.

***S* = *T*: Paso 19.1**

El propósito de este paso es copiar el valor de s_i a *sumando*2. Proceso de este paso es igual que el del paso 9.1.

***S* = *T*: Paso 19.2**

El propósito de este paso es escribir el valor 0 en la variable *sumando*2. El proceso de este paso es igual que el del paso 4.2.

***S* = *T*: Paso 20**

Este paso tiene como fin sumar *sumando*1 y *sumando*2 y almacenar el resultado en la variable *suma*. El proceso de este paso es igual que el del paso 5.

***S* = *T*: Paso 21**

Este paso se encarga de copiar el valor de la variable *suma* a la variable *suma_T*. El proceso de este paso es parecido al del paso 3.1, con la diferencia de que en lugar de la variable s_1 se considera la variable *suma* (la cual se encuentra almacenada en las posiciones de la 97 a la 100) y en lugar de la variable *sumando*1 se considera la variable *suma_T* (la cual se encuentra almacenada en las posiciones de la 107 a la 110).

***S* = *T*: Paso 22**

Este paso se encarga de determinar si *suma_S*=*suma_T*, y si esta condición se cumple el proceso termina en el estado "Y", en caso contrario el proceso termina en el estado "N". Para determinar si *suma_S*=*suma_T*, este submódulo debe realizar un proceso similar al del paso 6, con la diferencia de que en lugar de la variable *i* se debe considerar la variable *suma_S* (la cual se encuentra almacenada en las posiciones de la 102 a la 105) y en lugar de la variable *n* se debe considerar la variable *suma_T* (la cual se encuentra almacenada en las posiciones de la 107 a la 110).

Bibliografía

[Garey & Johnson, 1979] M.R. Garey, D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman and Company, New York, 1979.

[S.A. Cook, 1971] S.A. Cook, "*The complexity of theorem-proving procedures*", In STOC '71: Proceedings 3rd Annual ACM symposium on theory of computing, Association for Computing Machinery, pp. 151-158. New York 1971 doi: doi.acm.org/10.1145/800157.805047.

[Uber Turing Machine] Simulateur de machine de Turing pour des étudiants, des universitaires et des fervents de programmation, Uber Turing Machine. [on-line]. Disponible vía http://mscerts.programming4.us/fr/ 820048.aspx. (2010).

[Ryan, Finley, H. Susa] Cavalcante Ryan, Finley Thomas, Rodger H. Susan. A Visual and Interactive Automata Theory Course with JFLAP 4.0. Duke University; Durham, NC (2004).

[T.M. White, 2004] T.M. White, jFAST. [On-line]. Disponible vía http://www46.homepage.villanova.edu/timothy.m.whit/. (2004).

[C. Burch, 2001] C. Burch, Automaton Simulator. [On-line]. Disponible en internet vía http://ozark.hendrix.edu/~burch/proj/autosim/. (2001).

[Bovet, 2005] Bovet, Visual Automata Simulator. [On-line]. Disponible en internet vía http://www.cs.usfca.edu/~jbovet/vas.html. (2005).

[www.techterms.com] http://www.techterms.com/definition/emulation.

[www.thefreedictionary] http://www.thefreedictionary.com/computer+simulation.