05.12.2023


# PROYECTO FINAL

## Autómatas I

## Universidad Autónoma de Aguascalientes

**Integrantes:**

- César Eduardo Elias del Hoyo
- José Luis Sandoval Pérez
- Diego Emanuel Saucedo Ortega
- Carlos Daniel Torres Macías

**Maestro:** Israel de la Parra González

Ingeniería en Computación Inteligente

5to semestre

# ESPECIFICACIONES DEL PROYECTO

Este Proyecto consiste en la creación de cartas de amor aleatoria manejadas a partir de una selección aleatoria de elementos que permiten conformarla. Nuestro objetivo fue crear un pequeño juego sencillo que permitirá al usuario crear una cadena de símbolos donde cada uno de los símbolos que contenga irá generando un camino o un patrón a seguir que permitirá conformar la propia carta de acuerdo a lo que buscas en ella. Además, este también tendrá un aceptador de cartas que permita verificar si tu carta previamente creada es una carta parcialmente alegra o no.

El objetivo de este proyecto es aplicar los conocimientos adquiridos a lo largo del semestre en un programa con funcionalidad única. Para nosotros, aplicar autómatas en un programa que pueda generar una carta de amor fue una decisión e idea única que permite representar de manera carismática y además practica el trabajo presentado. Este siendo un precedente de como podemos manipular el lenguaje con una sería de reglas sencillas donde se determina el fin de una carta, de acuerdo con la entrada del usuario.

Su funcionamiento consta de un pequeño ejercicio donde se le darán instrucciones al usuario sobre una cadena que el usuario pueda insertar. Esta cadena contendrá símbolos representativos para la carta. Su principal función es que usted como pueda crear una carta de amor, tristeza, decepción o incluso añoranza para esa persona con quien tienes dicho sentimiento.

Así es como, nosotros hemos colocado n cantidad de letras distintas que representan el resultado final de acuerdo con las respuestas dadas por la cadena validada. De esta manera, cada usuario recibirá una carta única que, sin duda, podrá dedicarle el usuario a su respectiva pareja, amante, novio, novia, amigo o familiar, según sea el caso de la carta.

# PLAN DE TRABAJO

Para la realización de este proyecto, nos tomamos el tiempo de dividir el trabajo de manera equitativa para que todos aportemos de nuestra parte y así complementar el trabajo con ideas y creatividad de todos.

**CRONOGRAMA**

| **Integrante** | **Responsabilidades** | **Fechas trabajadas** | **Cumplimiento** |
|---|---|---|---|
| Todo el grupo | Platica del proyecto | Del 25 de septiembre hasta 20 de noviembre | Se ha estado planteando el trabajo a realizar, dando ideas y comenzando a surgir alguna posibilidad con la cual realizar el trabajo. De momento, únicamente se hizo una lluvia de ideas para a lo largo del semestre |
| | Boceto y creación de los autómatas (borrador) | 27 de noviembre 28 de noviembre | Se realizó correctamente el modelo y boceto de los autómatas, definiendo los aspectos del proyecto y repartiendo roles para el trabajo |
| César Eduardo Elias del Hoyo | Modelo Lógico | 29 de noviembre 30 de noviembre 1 de noviembre 4 de diciembre | Creación del modelo matemático acerca de los autómatas establecidos. Modelo lógico presentado |
| | Reporte | 2 de diciembre 3 de diciembre 4 de diciembre 5 de diciembre | Implementación del trabajo realizado a lo largo del proyecto en el pdf de entrega |
| José Luis Sandoval Pérez | Líder del equipo | 27 de noviembre | Repartición de los roles |
| | Modelo Lógico | 30 de noviembre 4 de diciembre | Aporte en la creación y desarrollo del modelo lógico |

| | | | |
|---|---|---|---|
| | JFLAP | 2 de diciembre<br>3 de diciembre<br>4 de diciembre<br>5 de diciembre | Creación de la comprobación, función y uso de los aceptadores en JFLAP |
| Diego Emmanuel Saucedo Ortega | Modelo lógico | 4 de diciembre<br>5 de diciembre | Aporte en la implementación del modelo matemático para la creación de su modelo gráfico |
| | Programa | 1 de diciembre<br>2 de diciembre<br>3 de diciembre<br>4 de diciembre | Implementación del proyecto a lenguaje c++ aplicando los aceptadores de forma que cumpla con el trabajo. Programador principal. |
| Carlos Daniel Torres Macías | Programa | 30 de noviembre<br>1 de diciembre<br>2 de diciembre<br>4 de diciembre | Creación de los algoritmos y funciones enfocados a la implementación de los autómatas en el lenguaje c++. Creación de estructuras del modelo lógico dentro del programa |
| | Reporte | 4 de diciembre<br>5 de diciembre<br>6 de diciembre | Aportación en la implementación del código dentro del reporte pdf. Edición y estilo del mismo reporte. |

Para nuestro cronograma, se decidió anotar únicamente la aportación *principal* de cada uno de los integrantes del equipo, siendo así una repartición justa y concreta para cada integrante. Sin embargo, todos los integrantes del equipo aportaron en todas las áreas descritas, aunque siempre más enfocados en el área designada, por lo que se decidió no ser reportado en el cronograma. El trabajo en equipo fue muy equitativo y todos aportaron correctamente en el trabajo en conjunto e individual del equipo.

*NOTA: para cada uno de los días asignados, se estima un trabajo neto de 2-3 horas por día trabajo. En casos pudo ser mayor a eso

# MODELOS TÉCNICOS DE SOLUCIÓN

Cómo sabemos, este proyecto consta de la aplicación de al menos dos autómatas correspondientes a las unidades I y III. Estos pueden ser aplicados para un mismo ejercicio o en distintos. Nuestro caso, consta de unir ambos autómatas en un único programa que permita que, con un funcionamiento contrario, ambos se complementen para poder desarrollar el fin del programa objetivo.

## 1. Autómata Aceptador Finito No Determinista (NFA)

Para nuestro primer autómata, hemos tomado un NFA de la primera unidad para representar el modelo principal en la creación de la cadena que dará representación a la estructura general de la carta. En este tendremos el autómata descrito por la siguiente quíntupla:

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b, e, f, t, r\}, \delta, q_0, \{q_7\})$$

Donde:

| | |
|---|---|
| $\delta(q_0, a) = q_1$ | $\delta(q_3, b) = q_7$ |
| $\delta(q_1, f) = q_2$ | $\delta(q_4, e) = q_3$ |
| $\delta(q_1, e) = q_3$ | $\delta(q_4, t) = q_4$ |
| $\delta(q_1, r) = q_5$ | $\delta(q_4, b) = q_7$ |
| $\delta(q_2, f) = q_2$ | $\delta(q_5, r) = q_5$ |
| $\delta(q_2, t) = q_6$ | $\delta(q_5, f) = q_5$ |
| $\delta(q_2, r) = q_6$ | $\delta(q_5, b) = q_7$ |
| $\delta(q_2, b) = q_7$ | $\delta(q_6, t) = q_6$ |
| $\delta(q_3, e) = q_3$ | $\delta(q_6, r) = q_6$ |
| $\delta(q_3, t) = q_4$ | $\delta(q_6, b) = q_7$ |

En esta quíntupla podemos observar que nuestro NFA consta de los siguiente:

- Tenemos 8 estados representados.
- Nuestro alfabeto de entrada consta de las letras {a, b, e, f, t, r}, únicas aceptadas por el NFA.
- Tenemos 20 transiciones que permiten al NFA avanzar durante la lectura de la cadena. Cada una de ellas con su respectivo estado actual, símbolo de lectura y estado siguiente.
- Nuestro autómata consta de un estado inicial q0 y un estado final q7.
- El estado q7 no tiene transiciones ya que, si el NFA llega a dicho estado, al ser estado final, se acepta la cadena (en caso de ya haber leído todos los símbolos).

De esta manera, nuestro NFA permite definir con el usuario el estilo de carta que desea obtener, dependiendo de la cadena que este usuario represente. Observamos que tenemos un primer único camino dado por el símbolo “a”, este representa un saludo al inicio de la

carta. Por ello, toda cadena aceptada deberá contener un saluda al inicio, de lo contrario, la cadena será invalidada.

Tenemos que también toda cadena aceptada deberá finalizar con una letra b para ser aceptada, pues b la consideramos como la despedida de la carta que siempre debe ir incluida.

El contenido de la cadena (entre la "a" inicial y la "b" final, existen 3 caminos posibles a seguir, el primero se refiere a una carta de felicidad/añoranza, donde le expresas a esa persona lo alegre que te sientes, a lo mejor de su amistad o de lo que está sucediendo en tu vida (eso se definirá en la programación), pues para que esto se dé el autómata deberá incluir por lo menos una "f", después tendrá dos opciones, mantener la carta totalmente feliz continuando con las f´s, pero si se coloca una "t" después de la f, esta carta será catalogada como añoranza, dándose a entender que también estas triste porque extrañas a esa persona. Como se pretende que genere cartas para situaciones generales, no indicará un contexto preciso del sentimiento, pero sí brindará la información que se quiere transmitir al usuario final. Por ello, para este caso la cadena aceptará un lenguaje dado por $\{aff^nub : n < 3, \mathrm{m} < 3\}$ siendo n el número total de párrafos que incluya tu carta feliz, y m el número total de párrafos para la parte triste. Si tu colocas una única f, la carta contendrá un único párrafo, siendo una carta breve y sencilla. De lo contrario, tener 3 "*f*" será una carta más elaborada y bien estructurada, así mismo al colocar mayor cantidad de "*t*".

Para el segundo caso, la cadena aceptará un lenguaje dado por: $L = \{aewb : w = \{e,t\}*, w < 4\}$. Esto significa que la carta, deberá indicar una e al inicio, ya que dicha carta va a enfocada al enojo, cuando alguien siente que esa persona hizo algo que no debía y quieres mencionárselo de manera escrita. Sin embargo, contiene w, lo cual puede ser una combinación (no mayor a 3) de "e" y "*t*", ya que aquí se pueden combinar dos caminos. De solo tener "e" contenida, será una carta de colera, como ya explicamos anteriormente, pero si incluye alguna "*t*", está refiriéndose a tristeza, el enfoque del sentimiento que intentará transmitir la carta es de decepción hacía el destinatario.

Por último, el camino que tenemos en la parte inferior hace referencia a una cadena con el lenguaje aceptado: $\mathrm{L} = \{arwb : w = \{r,f\}*\}$ donde r refiere a "romántico". Esta cadena aceptará toda cadena que empiece por un párrafo romántico y pueda incluir también algún camino alegre. Siempre hay que ser claros con las intenciones que el remitente quiere transmitir, por eso pedimos primero una "r" antes de poder ingresar w, para que el sentimiento de la carta y las intenciones del remitente estén bien definidas, pues de lo contrario, una carta que se pretendía ser romantica puede no ser clara para quien la recibe.

De esta manera, podemos concluir que el lenguaje general del autómata es el siguiente:

$$\boldsymbol{L = \{aff^nub,\ aevb,\ arwb : n \geq 0, u = \{t,r\}*, u \geq |0|,\ v = \{e,t\}*,\ v \geq |0|,\ w = \{r,f\}*,\ w \geq |0|\}}$$

En nuestro caso, se aplicó este lenguaje que simula correctamente el uso de las transiciones de acuerdo al modelo matemático planteado, pues este nos servirá correctamente para la creación y validación de una carta con un contenido w.

Sin embargo, en la implementación del código, por cuestiones de una impresión correcta de la carta, se tomó la decisión de reducir a cierto números de w, siendo así una cardinalidad límitada. De esta manera, tenemos que nuestro lenguaje limitaría su impresión dentro del código a únicamente una cardinalidad de $|5|$ la cual se explicará más adelante en el código

## 2. Autómata Aceptador No-Determinista de Pila (NPDA)

Para nuestro segundo autómata aceptador, hemos seleccionado un NPDA (por las siglas en inglés, Non-Deterministic Pushdown Automata) que nos permitirá realizar una comprobación para leer una cadena de una carta que ya se conoce en el cuál la aceptará si esta es una carta con indicios de felicidad mayoritarios. Esta se representa de la siguiente manera:

Nuestro NPDA está definido por la siguiente séptupla:

$$M = (\{ q_0, q_1, q_2, q_3, q_4\}, \{\mathrm{a, b, e, f, t, r}\}, \{\mathrm{Z},\ 1,\ 2\}, \delta, q_0, Z, \{q_4\})$$

Donde:

| | |
|---|---|
| $\delta(q_0, a, Z) = \{ (q_1, Z) \}$ | $\delta(q_2, r, 1) = \{ (q_3, 11) \}$ |
| $\delta(q_1, e, Z) = \{ (q_2, 2Z) \}$ | $\delta(q_2, \lambda, Z) = \{ (q_3, Z) \}$ |
| $\delta(q_1, t, Z) = \{ (q_2, 2Z) \}$ | $\delta(q_3, e, 1) = \{ (q_2, \lambda) \}$ |
| $\delta(q_1, f, Z) = \{ (q_3, 1Z) \}$ | $\delta(q_3, t, 1) = \{ (q_2, \lambda) \}$ |
| $\delta(q_1, r, Z) = \{ (q_3, 1Z) \}$ | $\delta(q_3, e, Z) = \{ (q_2, 2Z) \}$ |
| $\delta(q_2, e, 2) = \{ (q_2, 22) \}$ | $\delta(q_3, t, Z) = \{ (q_2, 2Z) \}$ |
| $\delta(q_2, t, 2) = \{ (q_2, 22) \}$ | $\delta(q_3, f, 2) = \{ (q_3, \lambda) \}$ |
| $\delta(q_2, e, 1) = \{ (q_2, \lambda) \}$ | $\delta(q_3, r, 2) = \{ (q_3, \lambda) \}$ |
| $\delta(q_2, t, 1) = \{ (q_2, \lambda) \}$ | $\delta(q_3, f, 1) = \{ (q_3, 11) \}$ |
| $\delta(q_2, e, Z) = \{ (q_2, 2Z) \}$ | $\delta(q_3, r, 1) = \{ (q_3, 11) \}$ |
| $\delta(q_2, t, Z) = \{ (q_2, 2Z) \}$ | $\delta(q_3, f, Z) = \{ (q_3, 1Z) \}$ |
| $\delta(q_2, f, 2) = \{ (q_3, \lambda) \}$ | $\delta(q_3, r, Z) = \{ (q_3, 1Z) \}$ |
| $\delta(q_2, r, 2) = \{ (q_3, \lambda) \}$ | $\delta(q_3, b, Z) = \{ (q_4, \lambda) \}$ |
| $\delta(q_2, f, 1) = \{ (q_3, 11) \}$ | $\delta(q_3, b, 1) = \{ (q_4, \lambda) \}$ |

Para esta séptupla podemos observar las siguientes características:

- Tenemos 4 estados representados
- Nuestro alfabeto de entrada consta de las letras {a, b, e, f, t, r}, únicas aceptadas por el NPDA

- Tenemos 28 transiciones que permiten al NPDA avanzar durante la lectura de la cadena. Cada una de ellas con su respectivo estado actual, símbolo de lectura, símbolo de lectura de la PILA, estado siguiente y símbolo(s) a colocar en la PILA
- Nuestro autómata consta de un estado inicial q0 y un estado final q4
- Q4 no tiene transiciones ya que, si el NPDA llega a dicho estado, al ser estado final, se acepta la cadena (en caso de ya haber leído todos los símbolos de la cadena)

Dicho de esta manera, nuestro autómata tendrá la función de leer una cadena que represente una carta. Como sabemos, las cartas para el autómata anterior (NFA) solo podían tener 1 o 2 sentimientos relacionados entre sí para que tuviera coherencia la carta

Sin embargo, el trabajo de esta nueva máquina permite leer cualquier cadena de una letra con los 4 sentimientos encontrados. Puede ser que esa persona haya escrito una carta en donde está enojado al inicio, luego triste, termine sacando lo feliz dentro de todo y finalice de manera romántica. Eso dependerá del usuario, pues al ser una carta hecha por otra persona, él puede hacer lo que quiera con ella. Por ello, no importará que la carta contenga cualquier sentimiento. Sin embargo, esta deberá incluir su correspondiente saludo inicial y despedida final, sin estos dos argumentos, una carta no se consideraría de tal manera.

Ahora, nuestra máquina aceptadora NPDA es nombrada (por nosotros) **Máquina Aceptadora de cartas AMOROSAS o ALEGRES.** ¿Esto qué quiere decir? La máquina solo acepta todas las cartas que contengan un cierto nivel de felicidad/romanticismo.

Para ello empleamos la siguiente estructura:

- Cada que la carta contenga una parte de la estructura feliz (f) o romántica (r) y la pila no contenga ningún número (es decir, la pila tenga una Z, donde se considera neutro) se la agregará a la pila un, manteniendo la Z al fondo de la pila
- Cada que la carta tenga una parte de la estructura feliz (f) o romántica (r) y la pila contenga un número 1 (es decir, que de momento la carta contenga mayor felicidad o romanticismo) se mantendrá dicho 1 de la pila, añadiendo otro 1 a la pila
- Cada que la carta tenga una parte de la estructura feliz (f) o romántica (r) y la pila contenga un número 2 (es decir, que de momento la carta contenga mayor enojo o tristeza) se quitará dicho 2 de la pila, sin añadir nada a esta (lambda)
- Cada que la carta contenga una parte de la estructura enojada (e) o triste (t) y la pila no contenga ningún número (es decir, la pila tenga una Z, donde se considera neutro) se la agregará a la pila un 2, manteniendo la Z al fondo de la pila
- Cada que la carta tenga una parte de la estructura enojada (e) o triste (t) y la pila contenga un número 2 (es decir, que de momento la carta contenga mayor enojo o tristeza) se mantendrá dicho 2 de la pila, añadiendo otro 2 a la pila
- Cada que la carta contenga una parte de la estructura enojada (e) o triste (t) y la pila contenga un número 1 (es decir, que de momento la carta contenga mayor felicidad o romanticismo) se quitará dicho 1 de la pila, sin añadir nada a esta (lambda)

Estas condiciones permitirán indicarle a la pila que tan alegre es el estado de la carta actual. Para comprobar esto hicimos lo siguiente:

- Como ya mencionamos en las condiciones anteriores, al tener una f o r, sumamos 1 o restamos un 2, y caso contrario cuando tenemos una e o t, sumamos 2 o restamos un 1, según sea el caso
- Ahora, esto permitirá decirle a la máquina si el estado en el que se encuentra es mayormente alegre (incluído al menos un 1 en la pila), menormente alegre (incluído al menos un 2 en la pila) o neutro (Z en la pila)
- De esta manera, cuando leamos nuestra despedida de la carta (b), este podrá determinar si se acepta o no, sabiendo que la transición a q4 solo es aceptada si se tiene una Z o un 1 en la pila, sabiendo así que nuestra carta es neutra o mayormanete alegre
- De lo contrario, no importa si la cadena finaliza con una b siguiendo la estructura, si la pila contiene al menos un 2, está no será aceptada ya que la carta escrita no contiene un mensaje mayormente positivo incluído

De esta manera, poemos concluir mencionando que para aceptar nuestra cadena (carta) esta deberá contener la estructura general de las cartas: saludo, contenido y despedida (awb) y además, el contenido deberá ser mayormente alegre, esto es, contener una mayor cantidad de f´s y r´s en ella

De esta manera, nosotros podemos respresentar dicha explicación en un lenguaje que es aceptado por la máquina. El lenguaje aceptado por nuestro NPDA esta definido de la siguiente manera:

$$\mathbf{L} = \left\{ \boldsymbol{awb} : \boldsymbol{w} = \{\boldsymbol{e}, \boldsymbol{f}, \boldsymbol{t}, \boldsymbol{r}\}*, \ \left(\boldsymbol{n_f}(\boldsymbol{w}) + \boldsymbol{n_r}(\boldsymbol{w})\right) \geq \left(\boldsymbol{n_e}(\boldsymbol{w}) + \ \boldsymbol{n_t}(\boldsymbol{w})\right) \right\}$$

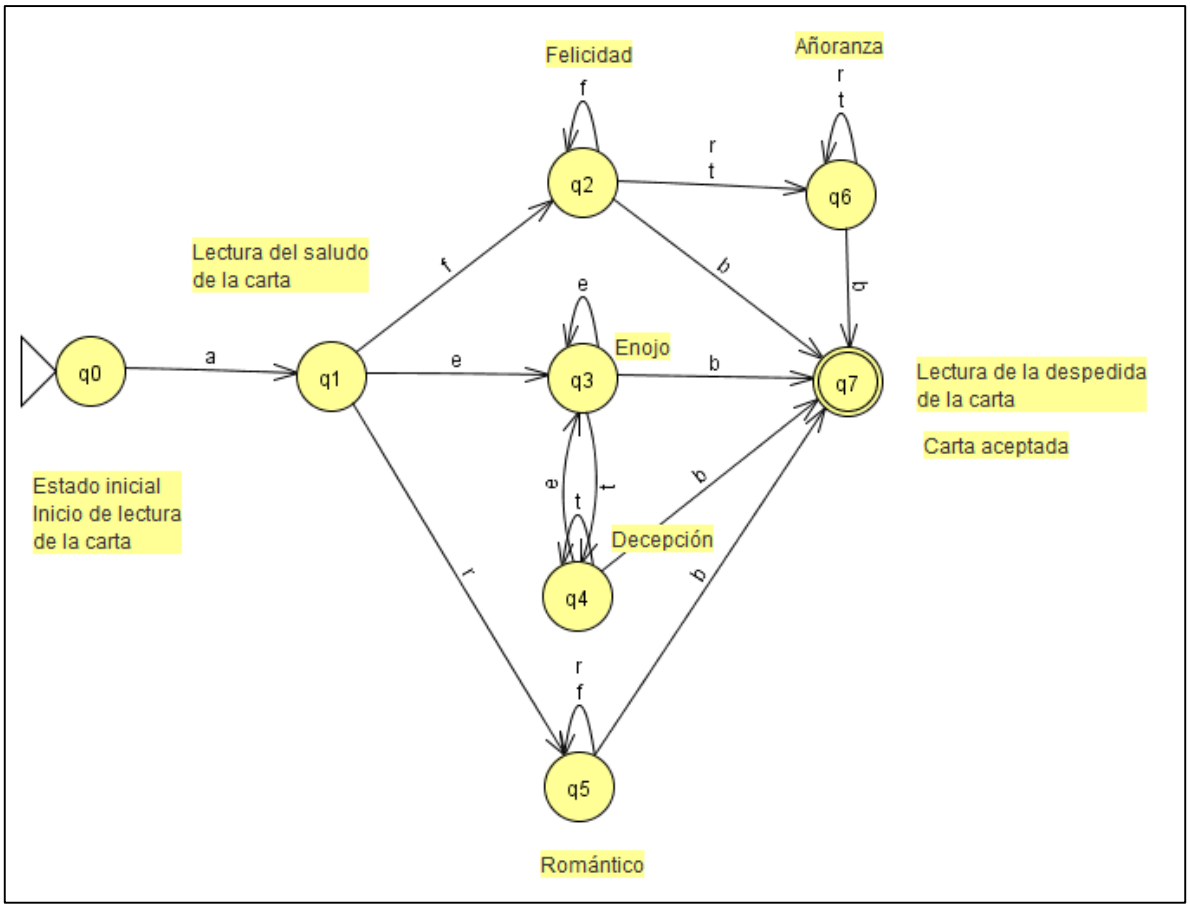
# COMPROBACIÓN EN JFLAP

Tenemos que, para los modelos técnicos previamente planteados (en nuestro caso se realizó únicamente un modelo matemático), existe un modelo gráfico que puede visualizarse de mejor manera la representación de nuestras máquinas de estados o autómatas definidos.

Para ello, se realizó la elaboración de dicha máquina y una comprobación con algunos ejemplos relacionados a la temática planteada (esto sin dar los resultados al problema, simplemente mostrar las cadenas aceptadas y cuales no)

## 1. Autómata Aceptador Finito No Determinista (NFA)

Para este primer autómata tenemos la representación del NFA en JFLAP de la siguiente manera:




*** *El diagrama JFLAP estará incluido como archivo NFA.jff en la entrega del proyecto*

Podemos observar que, para nuestro modelo, se le asignó un pequeño texto para cada estado con su respectiva representación y significado:

- Q0 es el estado inicial, donde comenzará a leer la cadena

- Q1 es el estado donde ya se leyó un saludo (a)
- Q2 será el primer camino donde tomará la carta un entorno de felicidad
- Q3 será el camino donde la carta tomará el tema de enojo
- Q4 convertirá la carta en un ánimo de decepción al combinar la tristeza y el enojo
- Q5 será el camino romántico
- Q6 convertirá la carta en un ánimo de añoranza al combinar la felicidad y tristeza
- Q7 será el estado final, donde aceptará la cadena proveniente de un estado después de leer una "b" final de tu cadena

En este caso, realizamos la comprobación de algunas cadenas que puedan confirmar nuestro modelo planteado. Para ello, realizamos la comprobación a:

- **Cadenas no aceptadas porque no cumplen con el alfabeto aceptado por el NPDA**

| Input | Result |
|---|---|
| ghlmn | Reject |
| hogar | Reject |
| jkjljl | Reject |
| 010120 | Reject |
| abcd | Reject |

Es lógico entender que, si dentro del autómata no se cumple el alfabeto de entrada establecido, nunca aceptará ninguna cadena de ellas

- **Cadenas no aceptadas que no cumplen la estructura awb**

| Input | Result |
|---|---|
| ffttb | Reject |
| aeete | Reject |
| rfrrf | Reject |
| aafftb | Reject |
| aetetbb | Reject |

Para este caso observamos que, si no cumple la estructura planteada, no funcionará para ningún caso. Ya sea que no contenga saludo, no contenga despedida, no contenga ninguna o tenga más de un saludo/despedida. Ninguna será aceptada ya que la estructura es clara: un saludo, contenido w (especificado en el lenguaje) y una despedida. Ni más ni menos

- **Cadenas no aceptadas que no cumplen las condiciones que conforman los tipos de cartas**

| Input | Result |
|---|---|
| affeeb | Reject |
| aerftb | Reject |
| aftftb | Reject |
| atrfeb | Reject |
| areb | Reject |

Las condiciones para generar las cadenas están definidas en el lenguaje donde nos dice que cada tipo de carta debe seguir el patrón de letras para cumplirlo. No podemos mezclar sentimientos porque no tendríamos una carta que ofrecer. Observamos que el primer caso mezcla felicidad y enojo, para el segundo mezcla las 4 emociones, en el tercero, a pesar de tener las dos emociones correctas para generar una carta, no cumple la estructura de tener primero la felicidad y después la tristeza; para el cuarto mezcla igualmente las 4 emociones y en el último caso mezcla dos emociones que no son válidas. Así se verá reflejado siempre que no se siga un patrón correcto

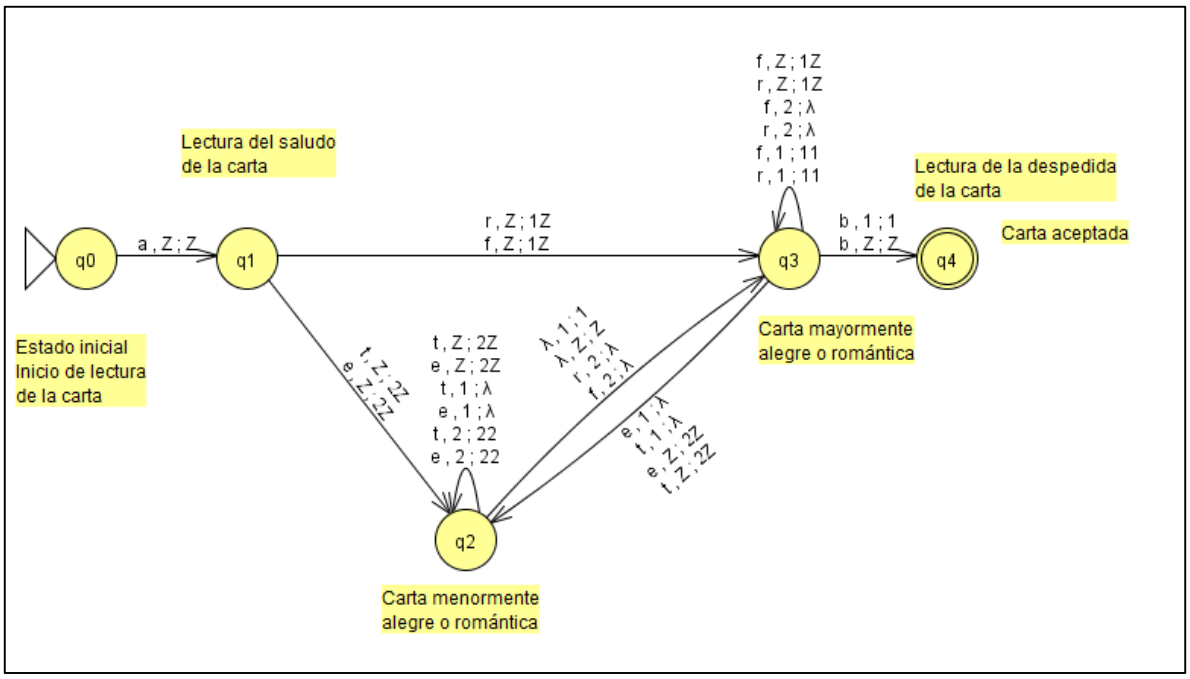
- **Cadenas aceptadas que SI cumplen con la estructura, el alfabeto y las condiciones dadas para la generación de cada tipo de carta**

| Input | Result |
|---|---|
| affffb | Accept |
| aftttb | Accept |
| aeeeb | Accept |
| aetetb | Accept |
| arfrrb | Accept |

En este último caso, se realizó la comprobación de cada uno de los 5 tipos de cartas aceptados, siendo así la cadena que debería aceptar nuestro NFA planteado. Cumpliendo nuestra estructura awb, las condiciones en w para obtener un enfoque de la carta, y seguir siempre el alfabeto aceptado, no habrá problema el autómata en arrojarte una aceptación de la carta

## 2. Autómata Aceptador No-Determinista de Pila (NPDA)

Para este segundo autómata planteado, tenemos el siguiente esquema realizado en JFLAP:




**** El diagrama JFLAP estará incluido como archivo NPDA.jff en la entrega del proyecto*

Podemos observar que, para nuestro modelo, se le asignó un pequeño texto para cada estado con su respectiva representación y significado:

- Q0 es el estado inicial, donde comenzará a leer la cadena
- Q1 es el estado donde ya se leyó un saludo (a)
- Q2 será el estado que se mantendrá siempre y cuando la pila contenga al menos un 2 (esto es, cuando sea mayormente negativa) o este neutro (Z en la pila)
- Q3 será el estado que se mantendrá siempre y cuando la pila contenga al menos un 1 (esto es, cuando sea mayormente positiva) o este neutro (Z en la pila)7
- Q4 será el estado final, donde aceptará la cadena si está tiene una Z o 1 en la pila

En este caso, realizamos la comprobación de algunas cadenas que puedan confirmar nuestro modelo planteado. Para ello, realizamos la comprobación a:

- **Cadenas no aceptadas porque no cumplen con el alfabeto aceptado por el NPDA**

| Input | Result |
|---|---|
| alksdlksb | Reject |
| casa | Reject |
| love | Reject |
| a01012b | Reject |
| alb | Reject |

Para este caso, como es evidente, no aceptará las cadenas que contengan símbolos que no se encuentren dentro de alfabeto planteado para nuestro autómata

- **Cadenas no aceptadas porque no cumplen la estructura awb**

| Input | Result |
|---|---|
| fretefb | Reject |
| aferfrf | Reject |
| fefrefe | Reject |
| aafb | Reject |
| affffbb | Reject |

Para este caso observamos que, si no cumple la estructura planteada, no funcionará para ningún caso. Ya sea que no contenga saludo, no contenga despedida, no contenga ninguna o tenga más de un saludo/despedida. Ninguna será aceptada ya que la estructura es clara: un saludo, contenido y una despedida.

- **Cadenas no aceptadas porque no cumplen la condición neutro o mayormente positiva** $\left(n_f(w) + n_r(w)\right) \geq \left(n_e(w) + n_t(w)\right)$

| Input | Result |
|---|---|
| aeeteb | Reject |
| afeetb | Reject |
| affttb | Reject |
| areetb | Reject |
| arfettb | Reject |

Este es el caso más importante, pues rechaza las cadenas que no cumplan la condición

Como observamos, todos y cada uno de estos ejemplos mostrados, cumplen con la estructura awb, todos cumplen con el alfabeto establecido, pero en los 5 casos se tiene un mayor número de e´s y t´s que de f´s y r´s. Por ello, a pesar de que toda la estructura es correcta, la cadena es rechazada ya que es mayormente negativa, triste o enojada la carta, por lo que nuestra máquina no lo aceptará

- **Cadenas aceptadas porque SI cumple la condición neutro o mayormente positiva** $\left(n_f(w) + n_r(w)\right) \geq \left(n_e(w) + n_t(w)\right)$ **, SI cumple la estructura y el alfabeto establecido**

| Input | Result |
|---|---|
| affffb | Accept |
| arfrfrb | Accept |
| arrrtb | Accept |
| afferb | Accept |
| afrteb | Accept |

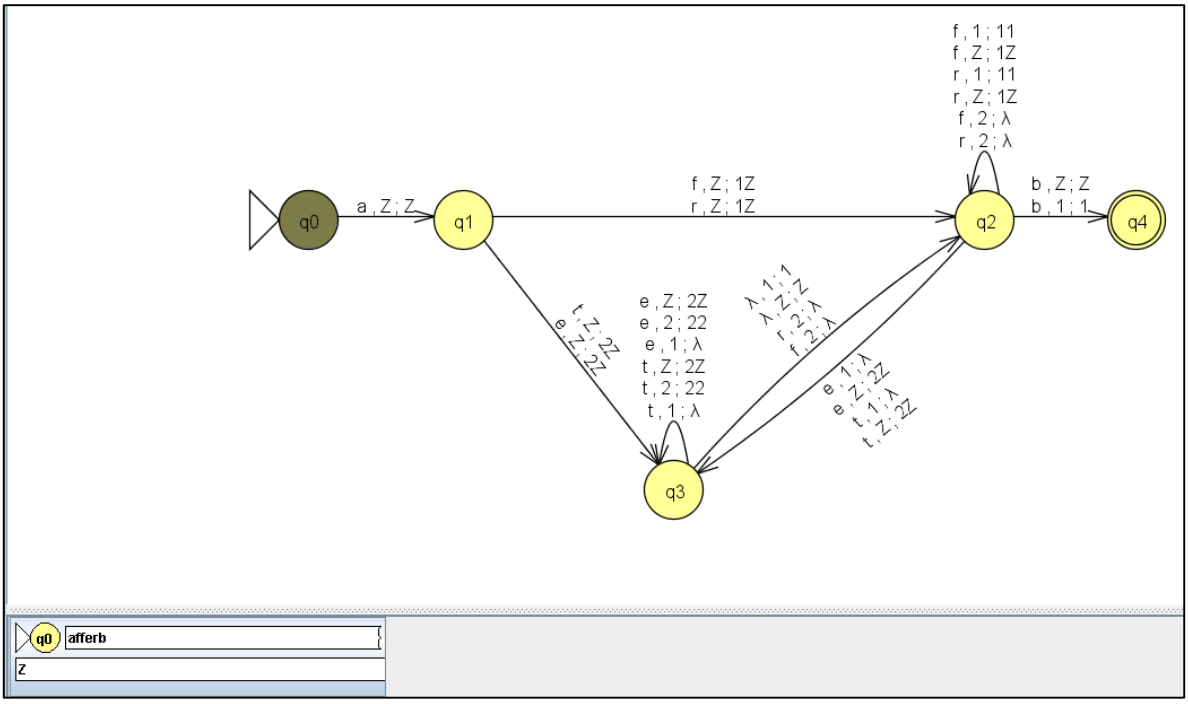
Para este último caso, y en el que es válida la cadena, consta de tener siempre bien planteada la estructura y el alfabeto asignado. Sabiendo que todo es correcto, comprobamos a analizar correctamente el ejercicio, incluso lo probamos paso a paso con el 4to ejemplo:

**Paso a paso para aceptar una cadena en el NPDA:**

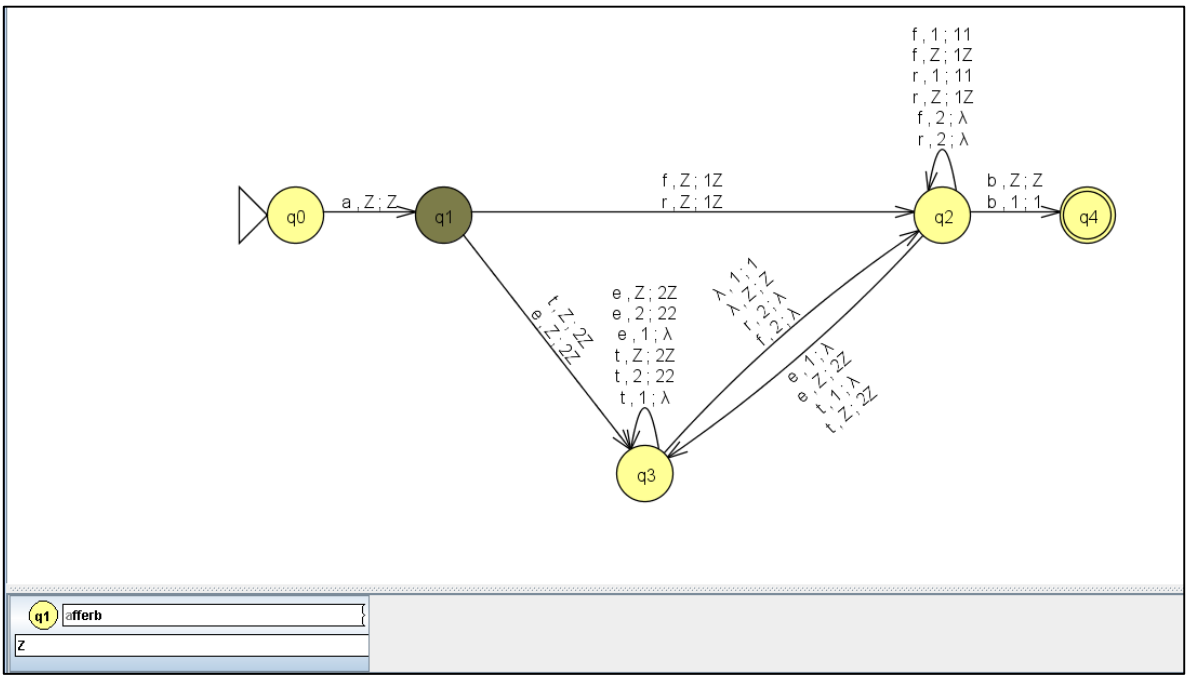
Decidimos realizar una comprobación en JFLAP para nuestro segundo autómata, de manera que se visualice como funciona dicho NPDA. Únicamente lo realizamos con esta segunda máquina ya que será más complicado de visualizar el procedimiento realizado por la mayor cantidad de transiciones a diferencia del NFA (creemos que no es necesario hacer una explicación paso a paso ya que es más evidente su funcionamiento) y por ello explicaremos aquí un ejemplo paso a paso del NPDA

Nuestra cadena a leer será "afferb". La cuál, viendo a simple vista consta de ffr, 3 símbolos que dan a la cadena un valor positivo y una única e que resta 1 símbolo negativo. A simple vista podemos constar qué si se trata de una carta mayormente positiva, pero vamos a comprobarlo en JFLAP:

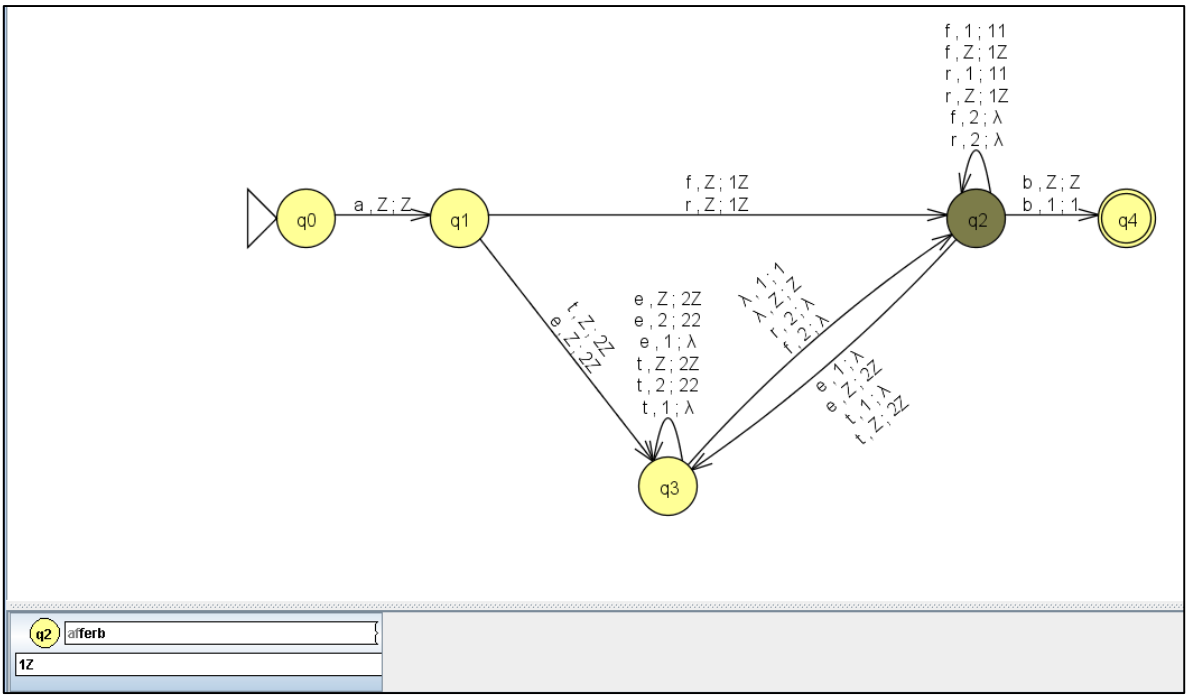
- Tenemos un estado inicial q0 donde comenzará a leer la cadena por primera vez. La pila está inicializada por Z



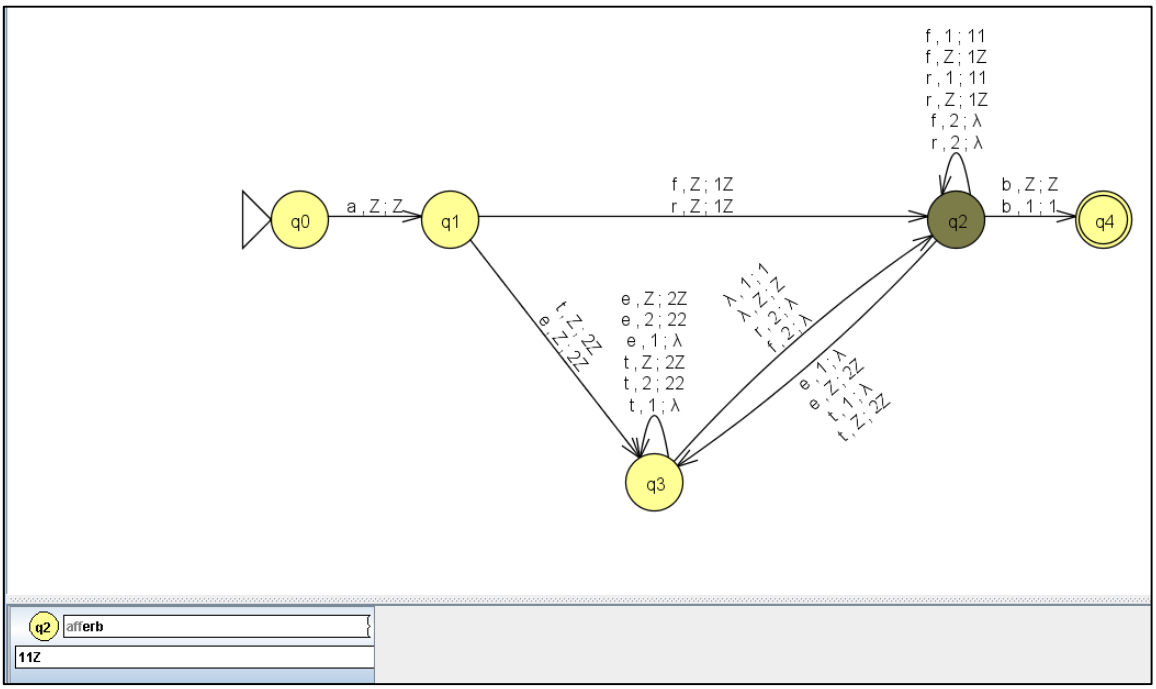
- Primer paso consta de leer el saludo de la carta, la cual lee y conserva la Z de la pila



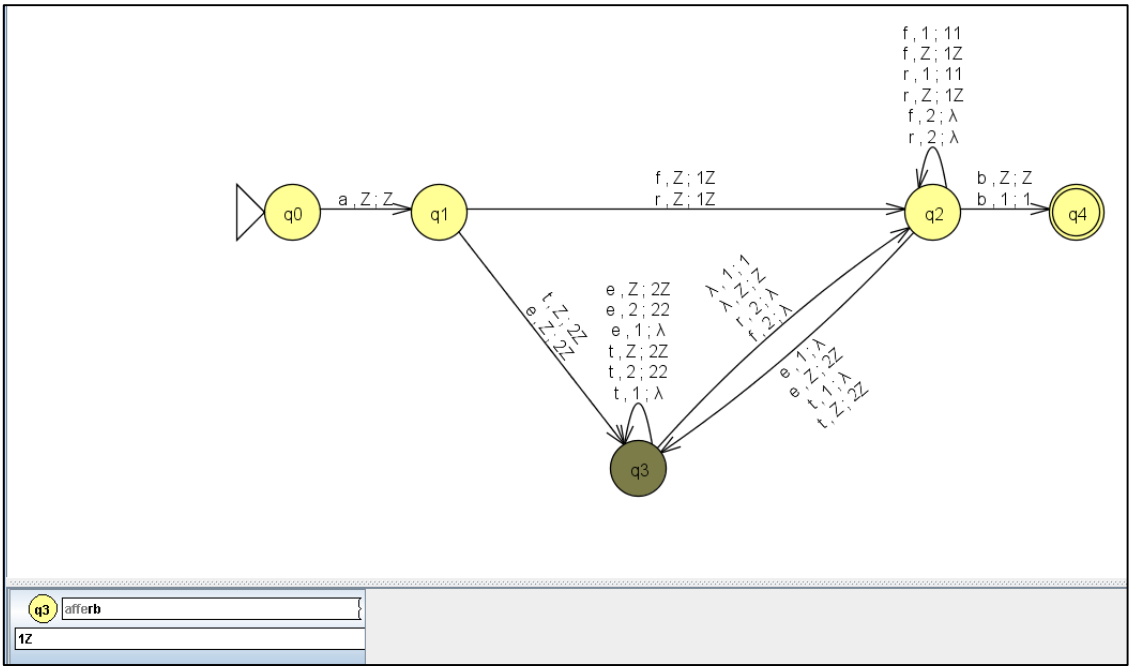

- Para este caso, tenemos como primer parte de la carta una f, quiere decir que la carta comienza con un tono de felicidad, dándole así una transición a q2 y añadiendo el primer 1 de la PILA



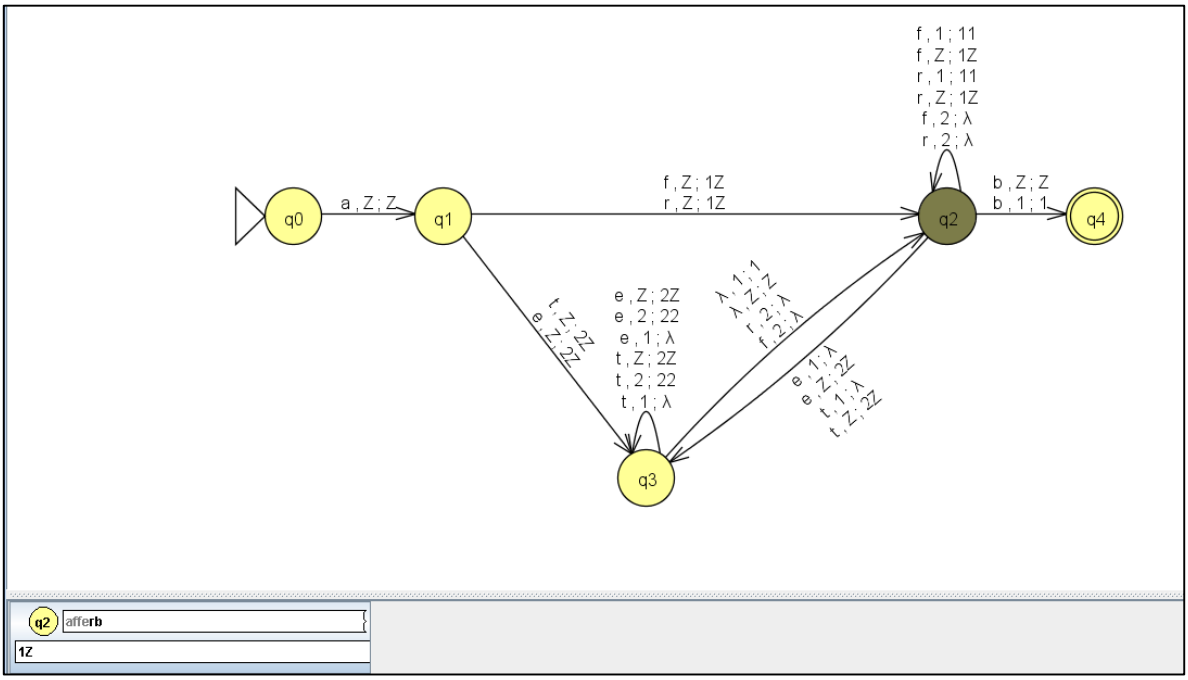
- Después tenemos una segunda f que seguirá sumando en el contador la cantidad de párrafos positivos de la carta, para este caso agregará un 1 más a la pila y se mantendrá en el mismo estado q2



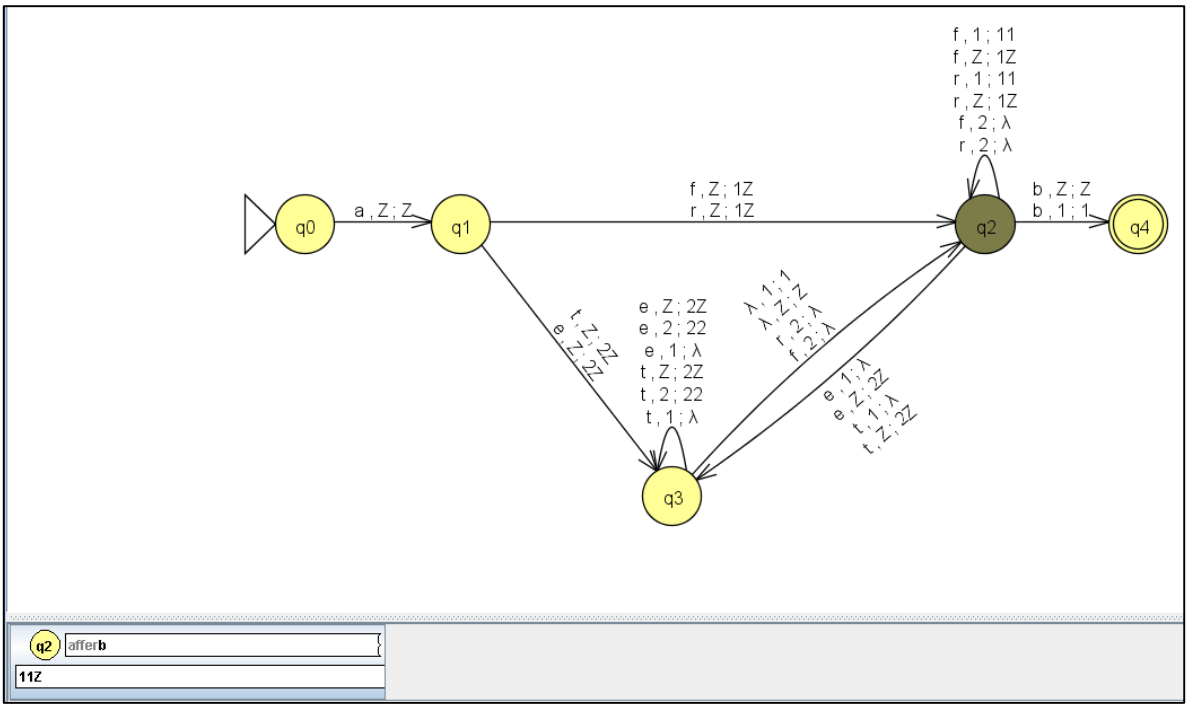

- Para el siguiente paso, el NPDA leerá una letra "e" que representa una parte negativa. Por lo cual, al tener 1´s en la pila, este sacará uno de ellos, realizará la transición a q3 y se mantendrá la pila un número 1 menos de los que había anteriormente al estar regresando lambda a la pila



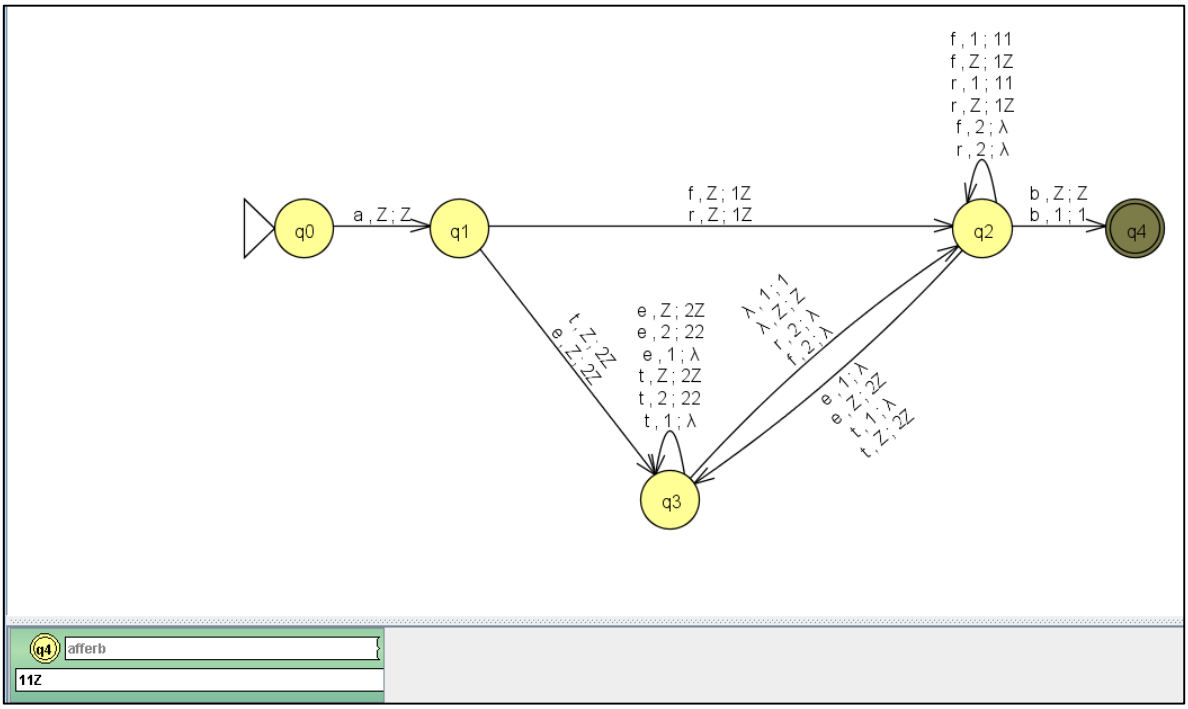
- Después de ello, el autómata leerá una r, pero como sabemos, q3 no tiene transiciones con r ni f. En este caso (y para todos en lo que se encuentre en esta posición) el autómata leerá una transición lambda que únicamente se realizará teniendo un 1 en la pila, esto con el fin de regresar al estado q2 y ahí procederá a seguir leyendo los símbolos positivos




- Estando en q2, ahora si procede a leer "r" y agregará un 1 a la pila



- El contenido de nuestra carta ya fue leído, solo que la despedida “b”, la cual se leerá correctamente ya que tenemos un 1 al tope de la pila. De esta manera, la cadena será aceptada automáticamente ya que llego al estado final (q4) y terminó de leer la cadena




Este sería un pequeño ejemplo paso a paso de como realiza el trabajo nuestra máquina aceptadora de pila NPDA. Este es uno de los muchos casos que acepta, siempre y cuando cumpla con nuestra estructura awb y las reglas para que la cadena termine siendo mayormente positiva

# CÓDIGO EN C++

Para la implementación en código de nuestro programa, utilizamos el lenguaje de programación C++ ya que, aunque para la parte visual no es la mejor opción, decidimos elegirlo ya que al ser un lenguaje de programación de *bajo nivel,* nos permitió enfocarnos más en el procedimiento y uso de los autómatas y aceptadores, ya que se puede tener un control y nivel de diseño que en lenguajes como Python ya es muy automatizado y no permite expresar de manera tan descriptiva el funcionamiento de los autómatas. En lugar de realizar un trabajo con mejor diseño, pero sin aplicar el uso de los conocimientos de la materia. Además, al tomar la decisión de realizar cartas de amor, no era tan necesario un lenguaje complejo, ya que lo realmente importante de la carta es el contenido, es decir, el texto, para esto es necesario únicamente la consola, para poder imprimir los resultados dados en cada ejecución. Para nuestro caso, trabajamos en Visual Studio Code, Dev-C++ y CodeBlocks, esto dependiendo del integrante del equipo que haya trabajado en el programa, ya que cada uno emplea la aplicación que más se acomoda a su gusto, pero asegurándonos, que el programa ejecute correctamente en las versiones estándar de C++.

Para nuestra programación, realizamos un pequeño menú con las instrucciones del juego, que permiten realizar 3 opciones

- Validar la estructura de una carta (NFA y NPDA)
```
bool verificarCarta(string cadena){
```
- Generar una carta (NFA)
```
void generarCarta(int tipo){
```
- Obtener el estado de ánimo de una carta (NPDA)
```
void validarAnimo(string cadena){
```

Como observamos en las imágenes, para cada ejercicio se tiene su propia función que permitirá evaluar el o los autómatas requeridos para dicho ejercicio

Nuestra intención fue crear 3 funcionalidades para obtener mejores resultados de nuestras máquinas aceptadoras previamente explicadas. Para ello, realizamos ejercicios en los que se les pueda aplicar un uso adecuado a cada máquina, siendo así un ejercicio en el que aplique los dos autómatas (verificar la estructura de una carta) y un ejercicio para comprobar a su vez su funcionamiento individual (generar una carta para el NFA, validar el estado de ánimo para el NPDA). Sin duda, validar la estructura de una carta ya era más que suficiente para comprobar el funcionamiento de nuestros dos autómatas, sin embargo, nos pareció mucha mejor idea comprobar a su vez de manera individual el proceso de cada una, además que se le dan usos distintos a la misma máquina. Generar una carta era la idea principal del proyecto, por ello también la generación de los ejercicios individuales.

Vamos a explicar cómo se realizó dentro del lenguaje cada una de las declaraciones de los autómatas, funciones establecidas, variables y como se aplica cada autómata en los ejercicios marcados.

## DECLARACIÓN

Primeramente, se requiere realizar la declaración de nuestros autómatas, sus estados correspondientes, transiciones y elementos. Aquí tenemos la captura de nuestras declaraciones

```
 9  struct Nodo{
10      int num;
11      vector<int>transicion;
12  };
13
14  Nodo ndfa[N_ESTADOS_NDFA];
15  Nodo ndpda[N_ESTADOS_NDPDA];
```

```
97  void construirNDPDA(Nodo ndpda[]){
98      //q0
99      ndpda[0].num = 0;
100     ndpda[0].transicion.PB(1);
101
102     //q1
103     ndpda[1].num = 1;
104     ndpda[1].transicion.PB(2);
105     ndpda[1].transicion.PB(3);
106
107     //q2
108     ndpda[2].num = 2;
109     ndpda[2].transicion.PB(2);
110     ndpda[2].transicion.PB(3);
111     ndpda[2].transicion.PB(4);
112
113     //q3
114     ndpda[3].num = 3;
115     ndpda[3].transicion.PB(2);
116     ndpda[3].transicion.PB(3);
117 }
```

```
119 void construirNDFA(Nodo ndfa[]){
120     //q0
121     ndfa[0].num = 0;
122     ndfa[0].transicion.PB(1);
123     //q1
124     ndfa[1].num = 1;
125     ndfa[1].transicion.PB(2);
126     ndfa[1].transicion.PB(3);
127     ndfa[1].transicion.PB(5);
128     //q2
129     ndfa[2].num = 2;
130     ndfa[2].transicion.PB(2);
131     ndfa[2].transicion.PB(6);
132     ndfa[2].transicion.PB(7);
133     //q3
134     ndfa[3].num = 3;
135     ndfa[3].transicion.PB(3);
136     ndfa[3].transicion.PB(4);
137     ndfa[3].transicion.PB(7);
138     //q4
139     ndfa[4].num = 4;
140     ndfa[4].transicion.PB(3);
141     ndfa[4].transicion.PB(4);
142     ndfa[4].transicion.PB(7);
143     //q5
144     ndfa[5].num = 5;
145     ndfa[5].transicion.PB(5);
146     ndfa[5].transicion.PB(7);
147     //q6
148     ndfa[6].num = 6;
149     ndfa[6].transicion.PB(6);
150     ndfa[6].transicion.PB(7);
151 }
```

Podemos observar en estas imágenes la declaración para el uso de nuestros autómatas. Primeramente, tenemos la declaración de una estructura Nodo que nos permitirá crear nuestros dos autómatas (declarados posteriormente). Dentro de nuestro struct, tenemos un variable int que representa le número del estado y un vector que permite saber a qué estado realizará la transición. De esta manera, podemos generar Sigma dentro de la función correspondiente a cada autómata

Si observamos en las siguientes dos imágenes tenemos dos funciones nombradas *construirNDPDA* y *construirNDFA,* estas hacen referencia a sus respectivos aceptadores. Cada función incluye una declaración para cada estado, asignándole el número y las transiciones.

Como ejemplo observamos el estado q1 del NPDA el cual nos dice lo siguiente:

```
//q1
    ndpda[1].num = 1;
    ndpda[1].transicion.PB(2);
    ndpda[1].transicion.PB(3);
```

Observamos que se declara el estado num=1, refiriéndose a q1, y tenemos sus transiciones. En este caso tiene dos transiciones, una a q2 y otra a q3

Para nuestra declaración, aún no se le asignan los símbolos que conforman cada transición, simplemente se le asigna el estado siguiente de la transición realizada

Esto únicamente sirve como declaración. El alfabeto, los símbolos leídos para cada transición y el cambio de estado se realizó dentro de las funciones de cada ejercicio asignado

## FUNCIONES PARA REALIZAR CADA EJERCICIO

- **Validar la estructura de una carta y generar dicha cadena (NFA y NPDA)**

```
if(seleccion==1){
            string cadena;
            cout<<"\n\tIngrese cadena de la estructura de la carta: ";
            cin>>cadena;

            bool aceptada = verificarCarta(cadena);

            if(aceptada==true){
                generarEjemplo(cadena);
                validarAnimo(cadena);
            }//si se acepta

          }//validar una cadena
```

Para nuestro programa, realizamos un menú con 3 opciones que constan de los diferentes usos de los autómatas, donde la primera opción será la descrita en este documento, ya que generaliza el trabajo de las otras dos opciones, por lo que no será necesario repetir la explicación con las demás instrucciones. Por ello, la selección 1 “Validar una cadena” permite implementar un NFA primeramente para validar que esta sea correcta de acuerdo a la estructura y emplea un NPDA que permitirá obtener un estado de ánimo de la carta una vez validado

```
153 bool verificarCarta(string cadena){
154     int nodo_actual = 0;
155     string tipo_carta = "Carta rechazada";
156     for(auto x:cadena){
157         //q0
158         bool hay_transicion = false;
159         if(x=='a' && nodo_actual==0){
160             nodo_actual = ndfa[nodo_actual].transicion[0];
161             hay_transicion = true;
162         }//fin transicion q0,a,q1
163 
164         //q1
165         if(x=='f' && nodo_actual==1){
166             nodo_actual = ndfa[nodo_actual].transicion[0];
167             hay_transicion = true;
168         }//fin transicion q1,f,q2
169         if(x=='e' && nodo_actual==1){
170             nodo_actual = ndfa[nodo_actual].transicion[1];
171             hay_transicion = true;
172             tipo_carta = "enojo";
173         }//fin transicion q1,e,q3
174         if((x=='r' || x=='f') && nodo_actual==1){
175             nodo_actual = ndfa[nodo_actual].transicion[2];
176             hay_transicion = true;
177         }//fin transicion q1,r,q5 - q1,f,q5
```

Tenemos primeramente el inicio del NFA el cual nos servirá para poder darle la estructura a la cadena ingresada. Como sabemos, la estructura de toda carta deberá ser awb, por lo cual se inicializará el autómata leyendo el símbolo “a” de manera obligatorio. Esto lo observamos en el apartado de //q0. Ahí tenemos nuestra condición inicial. Para q1, podemos empezar a leer cualquier variable, como sabemos, deberá elegir un camino para comenzar a crear la estructura válida de la cadena. Puede iniciar con la letra correspondiente para cada estado de ánimo

A todo esto, las transiciones las realizamos mediante condicionales if que nos permiten tomar el camino a seguir según sea el valor a leer de la cadena. Cada uno de los nodos, contendrá x cantidad de condicionales if, dependiendo x sea la cantidad de transiciones para dicho nodo

Para el resto de la función tenemos el mismo caso que q1, donde se tienen los condicionales if para las transiciones asignadas por cada estado en el que se encuentra. Dentro de los condicionales siempre tendremos dos instrucciones (en ocasiones 3 instrucciones):

```
nodo_actual = ndfa[nodo_actual].transicion[1];
hay_transicion = true;
tipo_carta = "enojo";
```

- nodo_actual nos permite guardar en el struct el nodo (estado) en el que se encuentra la lectura de la cadena
- hay_transición = true permite validar que se realizó una transición al estado en el que se encuentra
- tipo_carte permite delimitar el tipo de carta cuando ya se tiene el símbolo final de w en la cadena de lectura

Estas instrucciones se reflejan para los 6 estados centrales que permiten obtener la estructura de w y, por ende, obtener un estado de ánimo de la cadena.

Al finalizar de comprobar w, esta deberá leer el símbolo "b" de la despedida para llegar al estado q7 (estado final)

```
if(nodo_actual==7 || nodo_actual==3){
        cout<<"\n\tLa estructura de la carta fue aceptada.";
        cout<<"\n\tEl tipo de carta es: "<<tipo_carta<<endl;
        return true;
    }//if termina en estado final [q3,q7]
```

Esta función booleana tiene como función final comprobar que la cadena finalice de leerse estando en q7 y esta será aceptada por el NFA

```
if(aceptada==true){
                generarEjemplo(cadena);
                validarAnimo(cadena);
            }//si se acepta
```

Regresando al main, una vez aceptada la cadena, se entra en un condicional que nos manda a ocuparla nuestro segundo autómata contenido en la función validaranimo()

Esta función tiene como principal objetivo implementar el NPDA que permite comprobar que tantos párrafos de la cadena se tienen de manera positiva (felicidad y románticos) o negativos (enojo y tristeza). Para ello desarrollamos las siguientes instrucciones:

```
stack<char>pila;
    pila.push('z');
```

Para este caso, haremos el uso por primera vez de la pila, la cual será inicializada por z para comenzar nuestro recorrido de lectura del NPDA

```
 if(i==cadena.size()-1){ transicion_lambda = true; }
```

```
        bool hay_transicion = false;
        char x = cadena[i];
        //cout<<estado_actual<<" "<<x<<" "<<pila.top()<<endl;
        //q0
        if(x=='a' && pila.top()=='z' && estado_actual==0){
            estado_actual = ndpda[estado_actual].transicion[0];
            pila.pop();
            pila.push('z');
            hay_transicion = true;
        }//q0,a,z,z,q1
```

Esta máquina realizará la misma comprobación que el NFA, ya que debemos asegurarnos que siempre se cumpla la estructura awb. Por lo tanto, iniciaremos en un estado q0 donde leerá una "a" para comenzar con la estructura de la cadena.

Para este caso, tenemos instrucciones designadas para cada condicional planteada dentro de cada estado, como observamos en la imagen anterior.

Primeramente, tenemos el estado actual, instrucción que permite saber dónde se encuentra actualmente la lectura de la cadena (es decir, el estado o nodo donde nos situamos previos a la lectura de un siguiente símbolo)

Después tenemos pila.pop(), como sabemos, las pilas funcionan mediante la lectura y sacado del símbolo en lo más alto de esta misma, el cual se sacará con la función pop()

Pila.push() permite añadir a la pila el valor contenido dentro del paréntesis. Incluso habrá casos donde se introducirá más de un símbolo, esto se da en los estados q2 y q3

```
pila.pop();
pila.push('z');
pila.push('2');
```

Al final, tenemos la validación que comprueba una siguiente transición a realizar.

```
Esto ocurre para los 3 estados contenidos que conforman w dentro del NPDA. Una vez se llega a q4, la cadena se aceptará y se valiodará. Para la validación tenemos lo siguiente:
if(estado_actual == 4 && !pila.empty()){


         if(pila.top()=='1'){ cout<<"\n\tLa carta tiene un animo mayormente positivo.\n"; }
        else{ cout<<"\n\tLa carta tiene un animo neutro.\n"; }

        //vaciar pila
        while(!pila.empty()){ pila.pop(); }
        return;
    }//fin if es aceptada
```

```
    //vaciar pila
    while(!pila.empty()){ pila.pop(); }
    cout<<"\n\tLa carta tiene un animo mayormente negativo.\n"; return;
```

Como mencionamos anteriormente, una vez llegado a q4, este tendrá dos caminos que tomar:

- Si se realizó la transición a q4 con una Z en la pila, el estado de ánimo de la carta validad será neutro, pues una Z indica que no tenemos un estado de ánimo favorable o mayor a los demás
- Si se realizó la transición a q4 con un 1 en la pila, el estado de ánimo de la carta valida será posivo (mayormente positivo) pues incluye una cantidad mayor de estados positivos que negativos
- Sin embargo, si no se llega a un estado q4 o se tiene un 2 en la pila al finalizar de leer la cadena, esta no será aceptada y arrojará el mensaje de "animo mayormente negativo"

Como ya explicamos anteriormente, el evento 1 para validar la cadena permite emplear nuestros dos autómatas aceptadores establecidos. Para hacer uso de cada uno por separado tenemos:

- Validar el estado de ánimo (únicamente lo valida mediante el NPDA)
- Generar una carta (valida mediante la estructura del NFA e imprime una carta)

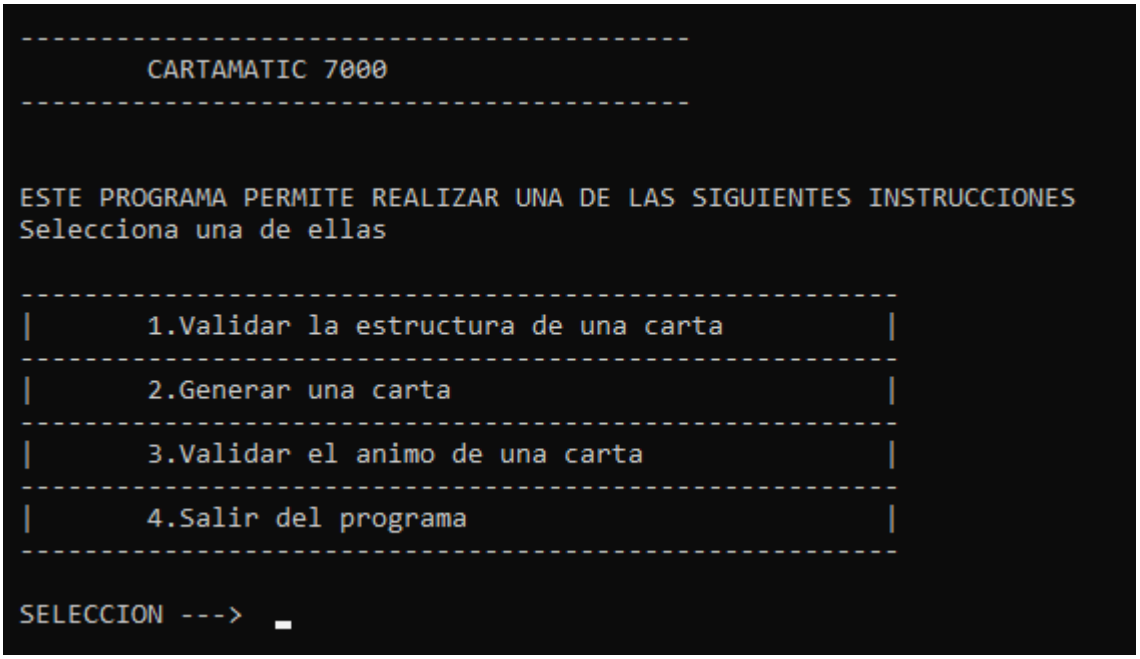
Este es el funcionamiento general del código a grandes rasgos y definiendo correctamente nuestros aceptadores empleados en el programa

**** El programa en c++ estará incluido como archivo Proyecto_Cartas_ELIAS_SANDOVAL_SAUCEDO_TORRES.cpp en la entrega del proyecto*

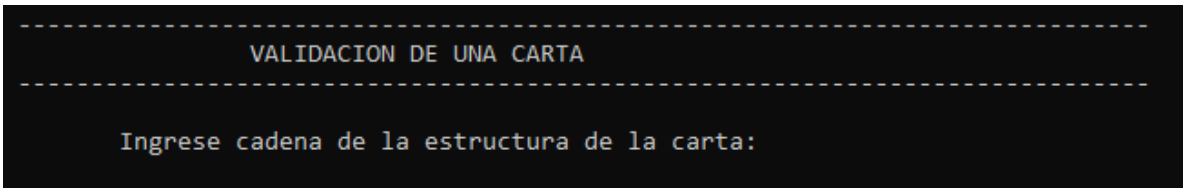
## IMPRESIÓN DE LOS RESULTADOS

Una vez explicado el procedimiento de nuestro programa, pasemos a visualizar los resultados del programa:

- Primeramente, tenemos un vistazo de la pantalla inicial con el menú de opciones




- Seleccionando cada una de las opciones podemos entrar en la resolución de cada opción. Primeramente, visualizaremos "Validar la estructura de una carta"




El programa te pide que ingreses una cadena de acuerdo a la estructura de la carta Sabemos que la estructura es awb, siendo w un conjunto de caracteres conformado por las emociones representadas en la carta

```
--------------------------------------------------------------------------------
                VALIDACION DE UNA CARTA
--------------------------------------------------------------------------------

       Ingrese cadena de la estructura de la carta:
       afftrb


       Validacion de la carta
       ----------------------------------------------
          La estructura de la carta fue aceptada.
          El tipo de carta es: anoranza
       ----------------------------------------------



       Validacion del estado de animo de la carta
       ----------------------------------------------
          La carta tiene un animo mayormente positivo.
       ----------------------------------------------
```

Observamos que al ingresar una cadena que sea aceptada, la validación de la carta lo confirmará dando el tipo de carta que es. Posteriormente, se aplica el NPDA para obtener el estado de ánimo

```
      Ejemplo de carta generada


      Hola, ¿Como vas?

      Eres una persona muy divertida y original, siempre tienes algo ingenioso que decir. Me encanta pasar tiempo contigo y compartir nuestras bromas.
      Me encanta tu sonrisa y tu forma de ser. Eres una persona muy especial para mi y te quiero mucho.
      Yo te juro que no me imaginaba como se sentia ni como seria un verano sin ti.
      Me gusta ver como el sol se refleja en tus ojos y como tu sonrisa ilumina tu rostro. Me siento feliz de tenerte a mi lado.

      Que tengas un buen dia, ciao.


Presione una tecla para continuar . . .
```

Por último, muestra un ejemplo de una carta que está basada en nuestra carta ingresada

```
        ------------PROYECTO REALIZADO POR------------

        CESAR EDUARDO ELIAS DEL HOYO
        JOSE LUIS SANDOVAL PEREZ
        DIEGO EMANUEL SAUCEDO ORTEGA
        CARLOS DANIEL TORRES MACIAS

--------------------------------
Process exited after 329.9 seconds with return value 0
Presione una tecla para continuar . . .
```

Para finalizar, si se selecciona la opción 4, el programa finaliza y nos muestra los nombres de los integrantes que conforman este equipo de trabajo

# CONCLUSIONES INDIVIDUALES

## César Eduardo Elias del Hoyo

Concluir con este trabajo me hizo reflexionar la cantidad de proyectos y aplicaciones que puede tener una máquina aceptadora o autómata de estados. Sin duda, para nosotros fue un tanto complicado tomar la decisión correcta de cómo llevar a cabo el enfoque del proyecto; nos surgían ideas mejores, cambiamos la problemática cada 20 minutos, buscábamos algo diferente o mejor, sabiendo que existen un mundo de aplicaciones.

Para mí, crear este proyecto fue un tanto divertido porque la temática de las cartas fue de nuestras primeras ideas que ya teníamos contemplada para implementarlo y resultaba divertido, además, la creación de dos autómatas que encajarán para la resolución y obtención de una carta, fue sin duda una experiencia satisfactoria. Poder crear un modelo lógico explicado, con fundamentos y una aplicación real en un programa fue una tarea que nos costó adaptarnos, pero que sin duda el resultado quedó de la mejor manera. Me llevo mucha experiencia tanto en el tema de las máquinas aceptadoras como en su implementación en la vida real

## José Luis Sandoval Pérez

Desde que me revisé el plan de estudios de la carrera realmente esta materia y su nombre tenían algo que me llamaba, este semestre descubrí que era, de que se trataba. La implementación de un proyecto de este tipo y de todo lo aprendí en clase realmente me fascinó, la idea de implementar un generador de cartas a través de autómatas, para mí un romántico empedernido me cayó como anillo al dedo. El principal reto fue la manera en cómo funcionaría un generador de cartas, ¿qué estados habría? ¿Qué representaría cada estado? ¿Cómo carajos escribiríamos los textos de la carta?

Ya al comenzar a hacer la investigación con todo el equipo, al generar los primeros bocetos pudimos ir resolviendo todos los pequeños problemas que se nos presentaron.

El desarrollo del proyecto y la complejidad de este me permitió descubrir nuevas formas de construir lenguajes y autómatas. Realmente descubrí el poder que tienen los autómatas

## Diego Emanuel Saucedo Ortega

Personalmente, siempre me ha atraído la idea de analizar el lenguaje y su interpretación, a medida que avanzaba en el curso, más claro era lo que a mí me gustaría hacer como proyecto.

Analizar cartas es complejo, debo decir que la idea por más que quería que fuera sencilla no podía hacerlo. Primero, ¿Cómo diferenciaba entre los sentimientos de una carta?, ¿Mediante que iba a distinguirlo? Eventualmente, logre ver una forma, pero había que

construir un aceptador que pudiera comprender la estructura de las cartas a como las veía yo.

Yo acredito mucho el poder construir un “aceptador de cartas” a practicar el diseño de aceptadores, porque me permitió encontrar nuevas formas de construir lenguaje.

Dado que lo que quería hacer podría ser “ambiguo”, para el final del curso me era más sencillo limitar la función de los aceptadores solo a lo que era útil en el proyecto.

## Carlos Daniel Torres Macías

La idea de crear un generador de texto a partir de sentimientos en un principio nos pareció una muy buena idea, pero al momento de querer implementarla en el proyecto, nos dimos cuenta de lo complicado que puede ser el lenguaje.

Tratamos de simplificar el programa con sentimientos menos ambiguos y mas concisos, como lo fueron, el enojo, la tristeza, la felicidad, etc. y que estos mismos sentimientos nos llevaran a crear sentimientos más complejos, como la nostalgia y la decepción, esto para un simulador de cartas, donde cada párrafo está ligado para un sentimiento en especifica no está mal, pero si deseamos implementar algo así con sentimientos más profundos, no creo que podría ser posible sólo con autómatas. Debido a que, al tratar de generar emociones aún más complejas, los párrafos tendrían que abarcar más de una emoción, por lo que no tendríamos que trabajar sobre párrafos, sino palabra por palabra, discerniendo que significa cada una en la suma total de la estructura.

Esto último me parece imposible con un autómata, como ejemplo nuestro evaluador, indica de que sentimiento es cada carta porque tiene evaluado de que tipo de entrada es cada párrafo, pero al momento de evaluar palabra por palabra debería tener en cuenta cosas mínimas como puntuación y nexos, que el orden y presencia de estos modifican enteramente el significado de la oración.

Por conclusión, este proyecto es nuestro Magnum Opus de los aprendido en este curso, donde además de usar autómatas que vimos en ejercicios ahora pudimos darles un sentido práctico, que incluso, en mi caso personal, me ha dado intriga por investigar más sobre el uso de lenguaje y cómo podemos relacionar este con estructuras lógico-matemáticas como lo son los autómatas.

# COEVALUACIÓN Y AUTOEVALUACIÓN

**César Eduardo Elias del Hoyo**

| Nombre | Calificación | Justificación |
|---|---|---|
| César Eduardo Elias del Hoyo | 10 | En mi caso, realicé mayor parte del modelo lógico y estructura de la creación de las máquinas (autómatas) a implementar, así como el modelo matemático y también la creación de las explicaciones en el reporte de entrega. Aporte mínimo en el programa, pero siempre corrigiendo errores en el modelo |
| José Luis Sandoval Pérez | 10 | Fue el generador de ideas en el inicio del proyecto, líder que definió los parámetros y asignaciones, realizó la comprobación y creación de los modelos lógicos, así como su comprobación en JFLAP. Apoyo en el modelo matemático a su vez |
| Diego Emanuel Saucedo Ortega | 10 | Principal aporte en la creación de las ideas. Su idea fue la principal tomada para el proyecto. Encargado de la creación del programa en c++. Implementó nuestros modelos de manera que pudiera validar una carta y generar otra. Permitió un manejo adecuado de nuestra estructura dentro del programa |
| Carlos Daniel Torres Macías | 10 | Fue clave en la generación de ideas para la creación y unión de los autómatas. Aportó en el desarrollo del modelo lógico y principalmente generó las funciones para creación de los ejercicios vistos en el programa, generando |

**José Luis Sandoval Pérez**

| Nombre | Calificación | Justificación |
|---|---|---|
| César Eduardo Elias del Hoyo | **10** | El principal participante en el desarrollo de la idea de los autómatas, su ayuda fue muy necesaria para los detalles finales. Elaboró el modelo lógico de una manera perfecta. |
| José Luis Sandoval Pérez | **10** | Fui el organizador del equipo, propuse la mayoría de las ideas para el desarrollo del proyecto. Desarrolle la idea final, ayude en la construcción del modelo lógico y perfeccionar código. |
| Diego Emanuel Saucedo Ortega | **10** | Desarrollo el boceto de los autómatas a implementar, el programa fue implementado por el en su mayoría. Su ayuda fue la más primordial. |
| Carlos Daniel Torres Macías | **10** | Ayudo a realizar los bocetos de los autómatas junto con Diego, dándole un toque único y de la manera más optimizada posible. No aporto demasiado en el modelo lógico y del programa. |

**Diego Emanuel Saucedo Ortega**

| Nombre | Calificación | Justificación |
|---|---|---|
| César Eduardo Elias del Hoyo | 10 | Muy participativo en el desarrollo de la idea de los autómatas y de su elaboración, comprobación y correcciones necesarias. Elaboración del modelo lógico del proyecto. |
| José Luis Sandoval Pérez | 10 | Principal organizador del equipo, ayudo a discernir ideas para la elección del proyecto. Ayudo a la conceptualización de los autómatas, construcción del modelo lógico y correcciones en el programa. |
| Diego Emanuel Saucedo Ortega | 9 | Participe en la elección de la idea del autómata, construcción del bosquejo de los autómatas y elaboración del programa. Pude ser más organizado con las ideas propuestas y así reducir el tiempo de elección de los autómatas. |
| Carlos Daniel Torres Macías | 9 | Participó en la elección y organización de ideas para los autómatas, realizo propuestas de autómatas y ayudo a la elaboración del bosquejo. Por razones extraescolares, tuvo menor participación en la elaboración del modelo lógico y del programa. |

**Carlos Daniel Torres Macías**

| Nombre | Calificación | Justificación |
|---|---|---|
| César Eduardo Elias del Hoyo | 10 | Fue el encargado principal del desarrollo del modelo lógico del proyecto, literalmente, de cimentar la idea para poder trasladar a un proyecto real. |
| José Luis Sandoval Pérez | 10 | Principal líder del equipo, puso orden a las ideas que teníamos para poder estructurarlas en el modelo lógico e implementarlas, comprobando en JFLAP todos los posibles autómatas que nos llevaron a los que finalmente usamos |
| Diego Emanuel Saucedo Ortega | 10 | Fue quien dio la idea que finalmente terminamos usando en el proyecto, construcción del bosquejo de los autómatas y elaboración del programa. Programador principal, al momento de desarrollar el archivo final en C++ |
| Carlos Daniel Torres Macías | 9 | Participé en la concepción del proyecto al aportar ideas para poder darle forma a los autómatas en la parte del modelado lógico, mientras que para el desarrollo del programa mi aporte fue en el desarrollo de funciones, aunque no tan participativamente como me hubiera gustado. |