Instituto Politécnico Nacional
Escuela Superior de Cómputo

Compiladores

Practica 1: Generador de Autómatas Finitos.

Hernández Martínez Carlos David.

davestringm@gmail.com.

Grupo: 3CV7.

November 2, 2020

Contents

1 Introducción

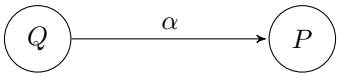
Un autómata finito es un modelo matemático de una máquina de estados que acepta cadenas de un lenguaje definido sobre un alfabeto. Consiste en un conjunto finito de estados y un conjunto de transiciones sobre esos estados que dependen de los símbolos de la cadena de entrada. El autómata finito acepta una cadena X si la secuencia de transiciones correspondientes a los símbolos de X conducen desde el estado inicial al estado final.

1.1 Autómata Finito Determinista

Un autómata finito determinista (AFD) se refiere al hecho de que para cada entrada solo existe uno y solo un estado al que el autómata puede hacer la transición a partir de su estado actual. La *función de transición* toma como argumentos un estado y un símbolo de entrada, como resultado devuelve un estado. La función de transición se designa habitualmente como σ. Entonces, si Q es un estado y α es un símbolo de entrada, entonces podríamos decir que la ecuación (1) es el estado P.

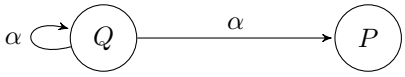
$$\sigma(Q, \alpha) \tag{1}$$

De tal forma que lo descrito anteriormente se puede describir gráficamente de la siguiente forma:



1.2 Autómata Finito No Determinista

Un autómata finito no determinista (AFND) tiene la capacidad de estar en varios estados a la vez, es decir, posee al menos un estado tal que para un símbolo existe mas de una transición (1) posible, entonces si Q es un estado y α es un símbolo de entrada, tenemos:



2 Desarrollo

La practica consta de desarrollar un programa que sea capaz de generar autómatas finitos no deterministas por medio de una quintupla definida como:

$$(Q, \Sigma, \alpha, q_0, F) \tag{2}$$

Donde:

- Q: Conjunto de estados finito y no vació.
- Σ: Alfabeto de entrada.
- α: Función de transición.
- q_0: Estado inicial.
- F: Conjunto de estados finales.

Estas quintuplas serán almacenadas en archivos TXT de tal forma que nuestro programa pueda leer el archivo y extraer los datos de el.

2.1 Lectura del archivo

El siguiente fragmento de código nos permite leer el archivo TXT, limpiar los datos en caso de que existan símbolos como saltos de linea o espacios en blanco y construir la quintupla que nos permitira generar el autómata finito que será capaz de aceptar cadenas pertenecientes a su alfabeto y decidir si tienen un estado de aceptación:

```
class FileHandler:
    """File Handler.

    This class handles everything that has to do with the file where the
    automaton quintuple is located; the purpose of this class it's to read the
    automaton quintuple file, clean it from any 'special character' stored in
    the file (like the backslash for example) and build the quintuple in a
    comprehensive tuple for the program '(states, alphabet, start_state,
    accept_states, transitions)'.

    Attributes
    ----------
    path: str
        Path where the quintuple file is located.

    """

    __slots__ = ("path",)

    def __init__(self, path: str):
        """Constructor."""
        self.path: str = path

    def read_file(self) -> Tuple:
        """Read file.

        Reads the file in the 'path' attribute.

        Returns
        -------
        Tuple
            An AFN quintuple.

        """
        with open(self.path, "r") as f:
            data: List = f.readlines()
        data = self.clean_data(data)
        return self.build_quintuple(data)

```

```
40      def build_quintuple(self, data: List) -> Tuple:
41          """Build Quintuple.
42
43          From the data collected from the file in the `path` attribute builds
44          the automata quintuple.
45
46          Parameters
47          ----------
48          data: List
49              Raw data readed for the file in the `path` attribute.
50
51          Returns
52          -------
53          Tuple
54              Automata quintuple.
55
56          """
57          states: List = data.pop(0).split(",")
58          alphabet: List = data.pop(0).split(",")
59          start_state: str = data.pop(0)
60          accept_states: List = data.pop(0).split(",")
61          transitions: List = [element.split(",") for element in data]
62          return (states, alphabet, start_state, accept_states, transitions)
63
64      def clean_data(self, data: List) -> List:
65          """Clean data.
66
67          Removes the backslash character for each element into a list.
68
69          Parameters
70          ----------
71          data: List
72              Raw data readed for the file in the `path` attribute.
73
74          Returns
75          -------
76          List
77
78          """
79          return [element.rstrip() for element in data]
```

2.2 Algoritmo

El siguiente algoritmo recursivo es el que permitirá generar el autómata finito y determinar si la cadena entrante pertenece o no al alfabeto. La clase AFN recibe como atributos la quintupla extraída del archivo TXT con el fragmento de código mostrado anteriormente, estos datos serán utilizados por la función **validate** que recibe como parámetros la cadena a validar, un apuntador a la posición actual de la cadena, el conjunto de caminos que se han generado y el estado actual en el que se encuentra el símbolo de la cadena que esta siendo evaluado.

```
1   class AFN:
2       """AFN class.
3
4       Object that contains the information of an AFN.
5
6       Attributes
7       ----------
8       states: List
9           Finite set of states.
10      alphabet: List
11          Finite set of symbols.
12      start_state: str
13          State before any input been processed.
14      accept_state: List
15          Set of states.
16      transitions: List
```

```
        Transition function

    """

    __slots__ = (
        "states",
        "alphabet",
        "start_state",
        "accept_states",
        "transitions",
        "_result",
        "_error",
    )

    def __init__(
        self,
        states: List,
        alphabet: List,
        start_state: str,
        accept_states: List,
        transitions: List,
    ):
        """Constructor."""
        self.states: List = states
        self.alphabet: List = alphabet
        self.start_state: str = start_state
        self.accept_states: List = accept_states
        self.transitions: List = transitions
        self._result: List = []
        self._error: str = "x"

    def _filter_transitions(
        self, data: str, position: int, current: str
    ) -> List:
        """Obtain Transition Set.

        Gets the possible next transitions that the automaton can follow for
        the given input.

        Parameters
        ----------
        data: str
            Input to validate.
        position: int
            Pointer to the current position of the input.
        current: str
            Current state of the automaton in which the input is positioned.

        Returns
        -------
        List
            Transitions that the automaton can follow for the input in the
            current position.

        """
        return list(
            filter(
                lambda x: x[0] == current and data[position] == x[1],
                self.transitions,
            )
        )

    def validate(
        self,
        data: str,
        position: int = 0,
```

```
83          path: List = [],
84          current: Optional[str] = None,
85      ) -> List:
86          """Validate.
87
88          Validates if a given input belongs to the automaton alphabet.
89
90          Parameters
91          ----------
92          data: str
93              Input to validate.
94          position: int
95              Pointer to the current position of the input.
96          path: List
97              Route that the automaton is generating for the input.
98          current: optional, str
99              Current state of the automaton in which the input is positioned.
100
101         Returns
102         -------
103         List
104             All posible routes that the automaton generate for the input.
105
106         """
107         current = self.start_state if current is None else current
108         if len(data) < position + 1:
109             path.append(current)
110             self._result.append(path)
111             return path
112         transition_set = self._filter_transitions(data, position, current)
113         if not transition_set:
114             path.append(current)
115             self.validate(data, position + 1, path.copy(), self._error)
116         path.append(current)
117         for t in transition_set:
118             self.validate(data, position + 1, path.copy(), t[2])
119         return self._result
```

3 Resultados

Todo el código mostrado anteriormente carece de significado su su operación no es mostrada. A continuación veremos 3 quintuplas en operación con nuestro algoritmo y dadas algunas cadenas de entrada, veremos el resultado que producen:

3.1 Autómata Finito 1

El siguiente autómata finito dado por la quintupla (2):

- Q: {0, 1, 2, 3}
- Σ: {a, b}
- q_0: 0
- F: {2}
- α: {(0,a,0), (0,b,0), (0,a,1), (1,b,2), (1,a,3), (2,a,3), (2,b,3), (3,a,3), (3,b,3)}

Produce la siguiente salida en nuestro programa con la cadena de entrada **aaab**:

Figure 1.0: Caminos que la cadena **aaab** puede seguir para llegar a su estado de aceptación.

Produce la siguiente salida en nuestro programa con la cadena de entrada **ababab**:

Figure 1.1: Caminos que la cadena **ababab** puede seguir para llegar a su estado de aceptación.

3.2 Autómata Finito 2

El siguiente autómata finito dado por la quintupla (2):

- Q: {1, 2, 3, 4, 5, 6}
- Σ: {a, b}
- q_0: 1
- F: {3, 4, 5}
- α: {(1,a,2), (1,a,5), (1,b,4), (2,a,2), (2,b,3), (3,a,6), (3,b,6), (4,a,6), (4,b,6), (5,a,6), (5,b,5), (6,a,6), (6,b,6)}

Produce la siguiente salida en nuestro programa con la cadena de entrada **aaab**:

Figure 2.0: Caminos que la cadena **aaab** puede seguir para llegar a su estado de aceptación.

Produce la siguiente salida en nuestro programa con la cadena de entrada **ab**:

Figure 2.1: Caminos que la cadena **ab** puede seguir para llegar a su estado de aceptación.

Produce la siguiente salida en nuestro programa con la cadena de entrada **abab**:

Figure 2.2: Caminos que la cadena **abab** puede seguir para llegar a su estado de aceptación.

3.3 Autómata Finito 3

El siguiente autómata finito dado por la quintupla (2):

- Q: {0, 1, 2, 3, 4}
- Σ: {a, +, -, .}
- q_0: 0
- F: {3}
- α: {(0,+,1), (0,-,1), (0,.,2), (0,a,4), (1,.,2), (1,a,1), (1,a,4), (2,a,3), (3,a,3), (4,.,3)}

Produce la siguiente salida en nuestro programa con la cadena de entrada **a.aa**:

Figure 3.0: Caminos que la cadena **a.aa** puede seguir para llegar a su estado de aceptación.

Produce la siguiente salida en nuestro programa con la cadena de entrada **+a.a**:

Figure 3.1: Caminos que la cadena **+a.a** puede seguir para llegar a su estado de aceptación.

Produce la siguiente salida en nuestro programa con la cadena de entrada **+aa.aa**:

Figure 3.2: Caminos que la cadena **+aa.aa** puede seguir para llegar a su estado de aceptación.

4 Conclusiones

Es muy interesante ver un programa que genere autómatas y valide cadenas dado una quintupla en acción. En esta ocasión haciendo uso de un algoritmo recursivo[1] se pudo cumplir con el objetivo sin tener que realizar un algoritmo que exceda el orden n-cuadrado.

5 Referencias Bibliográficas

[1] Baase and Van Gelder. "Computer Algorithms: Introduction to Design and Analysis". Addison-Wesley.

[2] Anderson James. "Automata theory with modern applications". Cambridge University Press.

[3] Linz Peter. "An Introduction to formal languages and automata". Jones and Bartlett Publishers.