Trabajo Práctico 3
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 2do cuat. 2020

Fecha límite de entrega: 12 de Noviembre (devolución: 19/11)

1. Introducción

En este TP trabajaremos con *autómatas finitos*. Un autómata es una máquina de estados —en este caso, finitos— cuyo objetivo es reconocer palabras de un lenguaje. Así, un lenguaje se entiende como el conjunto de todas las palabras que reconoce el autómata. En adelante profundizaremos en la descripción de los autómatas en sí y el análisis de las palabras que reconocen.

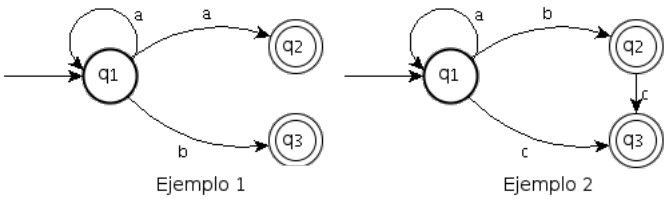

Figura 1: ejemplos de autómatas finitos

Un autómata consiste en un conjunto de estados, representados en la figura por círculos, y las transiciones entre ellos, representadas por flechas. Cada transición tiene una etiqueta correspondiente a un carácter del alfabeto que se utiliza para ese autómata. Para reconocer una palabra el autómata parte de su estado inicial, el que se indica con una flecha proveniente del exterior, y en cada transición acumula el carácter indicado por la etiqueta correspondiente. La palabra es reconocida si se logra acumular sucesivamente todos sus caracteres en el orden correcto, terminando en un estado final, señalado en la figura con un círculo doble. Puede haber varios estados finales, y se permite pasar por un estado final y seguir acumulando caracteres, pero el final de la palabra debe coincidir con la llegada a un estado final. Por ejemplo, el autómata del Ejemplo 1 de la Figura 1 reconoce cadenas como `"aaaaaa"`, `"aaab"` o `"b"` pero no `"aba"` o `"aaaac"`.

Los autómatas pueden ser o no ser determinísticos. Un autómata es determinístico si desde cada estado sale, a lo sumo, una transición con una misma etiqueta. En la Figura 1 se aprecia que el autómata 'Ejemplo 1' no es determinístico, ya que hay más de una transición para la etiqueta `a` desde `q1`, y el 'Ejemplo 2' sí lo es. Adicionalmente, ambos pueden reconocer una cantidad infinita de palabras, lo que es resultado de que los autómatas poseen ciclos.

Representación utilizada

Un estado de autómata se representará en JavaScript como un objeto con los atributos `esFinal` y `transiciones`, donde `esFinal` indica si se trata de un estado final, y `transiciones` es otro objeto en el cual los nombres de los atributos (todos de longitud 1) son las etiquetas, y sus valores son los estados a los que se llega mediante las respectivas transiciones. Si no hay transiciones que salgan de un estado, este tendrá como su atributo `transiciones` un objeto vacío.

Por simplicidad, no distinguiremos estados iniciales y en cambio trataremos a cada estado como si fuese el inicio de un autómata. Es decir, se puede empezar a reconocer una cadena desde cualquier estado. Esto nos permite a la vez analizar si un estado acepta (reconoce) o no una cadena, sin tener que incorporar el concepto de autómata como una entidad diferente.

2. Ejercicios

Ejercicio 1

1. Definir el objeto `qf`, que representa un estado final sin transiciones. Este objeto debe responder además el mensaje `acepta(s)` que, dada una cadena de caracteres `s`, devuelve `true` si `s` es vacía y `false` en caso contrario.

2. Definir los objetos `q1`, `q2` y `q3` que representen los estados del Ejemplo 2 de la Figura 1 con sus respectivas transiciones (por ahora no es necesario definir `acepta`).

Ejercicio 2

Extender el prototipo de `String` para que todos los Strings entiendan los mensajes `head()` y `tail()`, que devuelvan respectivamente el primer elemento y el resto de la cadena. Estos mensajes no deben modificar el objeto receptor, y deben devolver un resultado siempre que el receptor no sea vacío (no es necesario contemplar el caso de la cadena vacía).

Por ejemplo, `"unString".head()` debe devolver `"u"` y `"unString".tail()` debe devolver `"nString"`.

Ejercicio 3

1. Definir la función constructora `Estado(esFinal, transiciones)` que permita construir los objetos que representan los estados de un autómata determinístico. Además de los atributos `esFinal` y `transiciones`, cada objeto debe responder el mensaje `acepta(s)` que, dada una cadena de caracteres `s` indica si la cadena es aceptada por el autómata comenzando desde el estado actual, es decir, si se llega a un estado final luego de consumir toda la cadena de entrada.

   Por ejemplo, `new Estado(true, {x: q1}).acepta("xaaac")` debe devolver `true`.

2. Agregar a los estados creados en el ejercicio 1 la funcionalidad de saber responder el mensaje `acepta(s)`, sin modificar la solución ya implementada ni repetir código.

Ejercicio 4

Extender las definiciones anteriores de manera tal que los estados respondan al mensaje `nuevaTransicion(etiqueta, destino)`, de manera tal que si un estado recibe este mensaje, pase a tener una transición al estado `destino` mediante la etiqueta `etiqueta`.

Si el estado receptor ya tenía una transición definida para esta etiqueta, la nueva transición reemplaza a la anterior.

Es importante verificar que, si el estado anterior se creó a partir de otro estado, se modifique solamente el receptor y no su prototipo o el estado a partir del cual el receptor fue creado.

Ejercicio 5

Definir la función `algunoAcepta(s, qs)` que, dada una cadena de caracteres `s` y un estado o arreglo de estados `qs`, indica si alguno de ellos acepta la cadena de entrada.

Sugerencia: usar `Array.isArray(obj)` y `array.some(condición)`.

Ejercicio 6

Hasta ahora todos los autómatas definidos fueron determinísticos. En este ejercicio se deberá introducir indeterminismo con el mensaje `nuevaTransicionND(etiqueta, destino)`, que debe comportarse de la siguiente manera:

- Si el estado receptor no tenía una transición definida con la etiqueta `etiqueta`, se creará la nueva transición de la misma manera que en el ejercicio 4.
- Si el receptor tenía una transición con la etiqueta `etiqueta` hacia un estado diferente a `destino`, se reemplazará el valor de la transición correspondiente por un `Array` de estados que contenga tanto al destino original como al nuevo.
  En este caso el comportamiento de `acepta` debe cambiar para que el receptor pueda aceptar una cadena no vacía `s` si, o bien hay una transición mediante `s.head()` a un estado que acepta `s.tail()`, o hay una transición mediante `s.head()` a un `Array` de estados, alguno de los cuales acepta `s.tail()`. Las cadenas vacías siguen siendo aceptadas únicamente por estados finales.
- Si el receptor ya tenía una transición hacia `destino` mediante `etiqueta`, ya sea de manera determinística o no, entonces no debe realizarse modificación alguna.
- Si el receptor tenía una transición mediante `etiqueta` hacia `Array` de estados que no incluye a `destino`, se deberá agregar `destino` a este `Array`.

Al igual que en el ejercicio 4, si existen otros estados que comparten transiciones con el receptor, estos no deben modificarse.

Sugerencia: usar `array.includes(obj)`.

Ejercicio 7

Definir la función `esDeterminístico(q)` que, dado un estado `q` indique si todos los estados alcanzables desde `q` son determinísticos, es decir, si cada una de sus transiciones tiene un único estado destino posible.

Sugerencia: tener en cuenta que puede haber ciclos.

3. Pautas de Entrega

Se debe entregar el código impreso con la implementación de los ejercicios. Cada función o método asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en JavaScript a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-POO]` seguido inmediatamente del nombre del grupo.
- El código JavaScript debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `taller.js` (puede adjuntarse un **.zip** o un **.tar.gz**).

No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté **adecuadamente** comentado (son comentarios adecuados los que ayudan a entender lo que no es evidente o explican decisiones tomadas; no son adecuadas las traducciones al castellano del código). Los objetivos a evaluar son:

- Corrección.
- Declaratividad.
- Prolijidad: evitar repetir código innecesariamente y usar adecuadamente los metodos previamente definidos.

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias del lenguaje JavaScript

Como principales referencias del lenguaje de programación JavaScript, mencionaremos:

- **W3Schools JavaScript Reference:** disponible online en: https://www.w3schools.com/jsref/default.asp.
- **MDN Web Docs: Mozilla - Referencia de JavaScript**: disponible online en: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia.