**به نام خدا**

**گزارش کار پروژه درس نظریه زبان‌ها و ماشین‌ها**

**اعضای گروه: پریا خان‌جان (۴۰۱۱۷۷۳۳) و مهتا رنجبر دامغانی (۴۰۱۱۸۸۱۳)**

- ## پروژه ۳ : پذیرفته شدن یک رشته در یک NFA

پیاده‌سازی ما برای این پروژه به دو قسمت منطق کد و گرافیک تقسیم می‌شود که ابتدا منطق کد را توضیح می‌دهیم:

NFA یک اتوماتای متناهی است که به صورت یک ۵ تایی توصیف می‌شود و یک زبان را توصیف می‌کند: $(Q, \Sigma, \delta, q0, F)$

پس کلاس NFA را با فیلدهای زیر برای آن تعریف کردیم:

- Q مجموعه حالات اتوماتا است؛ در این پروژه ما ArrayList<String> states را به این مجموعه اختصاص دادیم. در این لیست، اسم حالات NFA گرفته شده از کاربر ذخیره می‌شود.
- Σ الفبای مختص به این اتوماتا است که ArrayList<Character> alphabet را به منظور ذخیره این حروف تعریف کرده ایم.
- δ تابع انتقال حالات موجود در Q به حالات دیگر توسّط حروف موجود در الفبای این زبان و یا رشته به طول صفر (λ یا ε) است: $\delta: Q \times (\Sigma \cup \{\}) \rightarrow P(Q)$
- در NFA برخلاف DFA ممکن است با یک حرف الفبا به چند حالت دیگر برویم یا ممکن است یک حالت کلاً انتقالی نداشته باشد. به این منظور transitions Map را تعریف کردیم که به عنوان key، sate مبدا را در آن ذخیره می‌کنیم و از آن‌جایی که یک استیت ممکن است با چند حرف الفبا به استیت‌های دیگر برود، value آن یک Map دیگر است. کلید این Map حرف الفبای این ترنزیشن است و چون همانطور که بالاتر توضیح دادیم، ممکن است یک استیت با یک ورودی یکسان به استیت‌های مختلف برود، value آن یک ArrayList است که حالات مقصد در آن ذخیره می‌شود.
- q0 حالت ابتدایی NFA است که به صورت String initialState آن را تعریف کردیم.
- F حالات نهایی اتوماتا است که از آن جایی که ممکن است چند حالت نهایی داشته باشیم، به صورت ArrayList<String> finalStates آن را ذخیره کرده‌ایم.

```
public class NFA extends JComponent {
    2 usages
    private final ArrayList<Character> alphabet;
    6 usages
    private final ArrayList<String> states;
    3 usages
    private final String initialState;
    3 usages
    private final ArrayList<String> finalStates;
    15 usages
    private final Map<String, Map<Character, ArrayList<String>>> transitions;
    6 usages
```

فیلدهای کلاس NFA

```
public NFA(ArrayList<Character> alphabet, ArrayList<String> states, String initialState,
           ArrayList<String> finalStates, Map<String, Map<Character, ArrayList<String>>> transitions) {
    this.alphabet = new ArrayList<>(alphabet);
    this.states = new ArrayList<>(states);
    this.initialState = initialState;
    this.finalStates = new ArrayList<>(finalStates);
    this.transitions = new HashMap<>(transitions);
    JFrame window = new JFrame();
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    window.setBounds( x: 30,  y: 30,  width: 450,  height: 450);
    window.getContentPane().add(this);
    window.setVisible(true);
}
```

کانستراکتور کلاس NFA

- **بررسی پذیرفته شدن یا پذیرفته نشدن یک رشته توسط NFA:**

برای این موضوع، دو متد acceptString و checkTransitions را تعریف کرده‌ایم:
متد acceptString رشته موردنظر را به عنوان ورودی دریافت می‌کند و ابتدا چک می‌کند که اگر طول رشته صفر بود (یعنی رشته λ یا ε است) به جای آن کاراکتر ε را در نظر می‌گیرد تا در صورت وجود انتقال با اپسیلون در استیت ابتدایی به مشکل برنخوریم. سپس تابع بازگشتی checkTransitions را صدا می‌زنیم. چون این اولّین فراخوانی این تابع است، استیت شروع را به عنوان استیت فعلی و اندیس صفرم را به عنوان اندیس فعلی رشته را به عنوان ورودی به آن می‌دهیم. ورودی‌های مورد نیاز دیگر، خود رشته ورودی و بولین مربوط به اتمام بررسی هستند که این بولین در ابتدا باید false باشد. در نهایت نتیجه نهایی را بر اساس خروجی تابع صدا زده شده چاپ می‌کنیم.

```
1 usage
void acceptString(String input) {
    if (input.equals("") || input.equals(" ")) {
        input = "ε";
    }
    if (checkTransitions(initialState,  charIndex: 0, input,  finish: false)) {
        System.out.println("This NFA accepts this string.\n" + input + " ∈ language");
    } else {
        System.out.println("This NFA doesn't accept this string.\n" + input + " ∉ language");
    }
}
```

متد acceptString

```
4 usages
boolean checkTransitions(String currState, int charIndex, String input, boolean finish) {
    if (charIndex == input.length()) {
        if (finalStates.contains(currState)) {
            finish = true;
        }
        return finish;
    }
    if (transitions.containsKey(currState)) {
        if (transitions.get(currState).containsKey('ε')) {
            for (int i = 0; i < transitions.get(currState).get('ε').size() && !finish; i++) {
                finish = checkTransitions(transitions.get(currState).get('ε').get(i), charIndex, input, finish: false);
            }
        } else if (transitions.get(currState).containsKey('λ')) {
            for (int i = 0; i < transitions.get(currState).get('λ').size() && !finish; i++) {
                finish = checkTransitions(transitions.get(currState).get('λ').get(i), charIndex, input, finish: false);
            }
        }
        if (transitions.get(currState).containsKey(input.charAt(charIndex))) {
            for (int i = 0; i < transitions.get(currState).get(input.charAt(charIndex)).size() && !finish; i++) {
                finish = checkTransitions(transitions.get(currState).get(input.charAt(charIndex)).get(i), charIndex: charIndex + 1, input, finish: false);
            }
        }
    }
    return finish;
}
```

## توضیح متد checkTransitions:

**این متد، متدی بازگشتی است.** می‌خواهیم بدانیم با هر حرف رشته ورودی به کدام استیت‌ها دسترسی داریم و از این استیت مبدا با این حرف به کدام استیت‌ها می‌توان رفت. همچنین چون ممکن است با استیت مقصد به نتیجه‌ای نرسیم و در این حالت به استیت‌های طی شده قبلی نیاز داریم، پیاده سازی این متد را به صورت بازگشتی انجام دادیم.

ورودی‌های این تابع به این صورت هستند: استیت فعلی (حالتی که در آن هستیم)، شماره اندیس حرفی از رشته ورودی که در حال بررسی آن هستیم، خود رشته ورودی و یک بولین که در ابتدا false است و هروقت در وضعیت accept قرار گرفتیم و پیمایش ما در NFA به اتمام رسید، true می‌شود.

base case این متد بازگشتی: هروقت به انتهای رشته ورودی رسیدیم، یعنی جست و جوی ما به اتمام رسیده است. اگر حالتی که در آن هستیم، از حالات نهایی ما باشد، رشته پذیرفته می‌شود و بولین finish، true می‌شود. در غیر این صورت باید حالات مختلف دیگر چک شوند.

recursive case این متد: در این حالت ابتدا چک می‌کنیم که برای استیت فعلی انتقالی وجود دارد یا خیر. سپس باید ببینیم در استیت فعلی انتقالی با اپسیلون وجود دارد یا خیر. اگر چنین انتقال‌هایی وجود داشتند، به تک تک استیت‌های مقصد می‌رویم تا تمام این مسیرها بررسی شوند. برای بررسی تمامی این مسیرها به استفاده از حلقه نیاز داریم. در این حلقه، این تابع را دوباره صدا می‌زنیم، با این تفاوت که یکی از استیت‌های مقصد این انتقال را به عنوان ورودی مربوط به استیت فعلی به تابع می‌دهیم و از آنجایی که ورودی ما اپسیلون بوده است، اندیس کاراکتر موردنظر تغییری نمی‌کند. از آن جایی که نتیجه این فراخوانی را در بررسی حالات دیگر هم نیاز داریم، اگر از دستور return استفاده کنیم به مشکل می‌خوریم چون این دستور باعث توقّف حلقه می‌شود و ممکن است همه حالات چک نشده باشند. پس نتیجه هر بررسی را در بولین finish ذخیره می‌کنیم. یکی از شروط توقف این حلقه، true شدن این بولین است چون در این صورت یکی از حالات پذیرفته شدن این رشته پیدا شده است و نیازی به ادامه بررسی نیست.

اگر کاربر به جای اپسیلون برای مشخّص کردن رشته به طول صفر از لاندا استفاده کرده باشد، همین روند را برای لاندا تکرار می‌کنیم.

پس از چک کردن انتقال‌های با رشته به طول صفر، باید ببینیم استیت فعلی با کارکتری از رشته ورودی که در حال بررسی آن هستیم به چه استیت‌های مقصدی می‌تواند انتقال بیابد. برای کار، مشابه روند بالا را تکرار می‌کنیم تا همه مسیرها بررسی شوند با این تفاوت که چون این کاراکتر خوانده شده است، در فراخوانی دوباره تابع، با کاراکتر بعدی در رشته استیت مقصد را چک می‌کنیم.

```java
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("**********WELCOME**********");
    System.out.println("Please enter the NFA alphabet(Σ):");
    String[] alphabetArray = scanner.nextLine().split( regex: " ");
    ArrayList<Character> alphabet = new ArrayList<>(alphabetArray.length);
    for (String s : alphabetArray) {
        alphabet.add(s.charAt(0));
    }
    System.out.println("Please enter the NFA states:");
    ArrayList<String> states = new ArrayList<>(Arrays.asList(scanner.nextLine().split( regex: " ")));
    System.out.println("Please enter the initial state:");
    String initialState = scanner.nextLine();
    while (!states.contains(initialState)) {
        System.out.println("This state doesn't exist! Please enter a proper initial state:");
        initialState = scanner.nextLine();
    }
    System.out.println("Please enter the final states:");
    ArrayList<String> finalStates = new ArrayList<>(Arrays.asList(scanner.nextLine().split( regex: " ")));
    while (!states.containsAll(finalStates) && !finalStates.contains("")) {
        System.out.println("Some states don't exist! Please enter proper final states:");
        finalStates = new ArrayList<>(Arrays.asList(scanner.nextLine().split( regex: " ")));
    }
```

**دریافت کردن الفبا، استیت‌ها، استیت‌های شروعی و پایانی از کاربر**

- **توضیحات کلاس Main:**

این کلاس برای اجرای برنامه و دریافت ورودی از کاربر می‌باشد.

در تابع main، ابتدا ۵ تایی مربوط به NFA مورد نظر کاربر را از او دریافت می‌کنیم.

- ابتدا حروف الفبای این زبان را از کاربر دریافت کرده و هر کاراکتر را در لیست alphabet ذخیره می‌کنیم.
- سپس نام استیت‌ها را از کاربر می‌گیریم و در یک لیست از جنس string به نام states ذخیره می‌کنیم.
- سپس استیت شروع را از کاربر دریافت می‌کنیم. اگر این استیت در لیست استیت‌ها وجود نداشت، تا زمانی که کاربر یک استیت معتبر را به عنوان استیت ابتدایی وارد کند از او می‌خواهیم این استیت را دوباره وارد کند.
- در مرحله بعد، استیت‌های پایانی را از کاربر دریافت می‌کنیم و در لیست finalStates ذخیره می‌کنیم. مشابه قسمت قبل، اگر کاربر استیتی که در لیست استیت‌ها وجود ندارد را وارد کند، تا زمانی که او استیت‌های معتبری را به عنوان ورودی بدهد از او می‌خواهیم دوباره این مرحله را تکرار کند.
- سپس از کاربر می‌خواهیم تعداد ترنزیشن‌های این اتوماتا را وارد کند.
- پس از آن، برای گرفتن این ترنزیشن‌ها از تابع scanTransitions کمک می‌گیریم و در نهایت آن‌ها را در یک map ذخیره می‌کنیم.
- حالا کانستراکتور کلاس NFA را صدا می‌زنیم و با این ۵تایی که به عنوان ورودی از کاربر دریافت کردیم، اتوماتای موردنظر را می‌سازیم.
- از کاربر می‌خواهیم رشته موردنظر خود را وارد کند و بررسی می‌کنیم که این رشته با الفبای مربوط به این زبان قابل ساختن باشد. اگر این شرط برقرار نبود تا زمانی که کاربر رشته‌ای معتبر وارد کند از او می‌خواهیم که دوباره آن را وارد کند.
- در نهایت رشته وارد شده را به متد acceptString کلاس NFA ساخته شده پاس می‌دهیم تا بررسی کند این رشته پذیرفته می‌شود یا خیر.

```
System.out.println("Please enter number of transitions:");
int transitionSize = scanner.nextInt();
Map<String, Map<Character, ArrayList<String>>> transitions = new HashMap<>(transitionSize);
scanner.nextLine();
if(transitionSize != 0) {
    System.out.println("Please enter the NFA transitions:\n*** For example:δ(q0, a) = q2 or f(q0, a) = q2");
}
scanTransitions(transitionSize, states, alphabet, scanner, transitions);
NFA nfa = new NFA(alphabet, states, initialState, finalStates, transitions);
System.out.println("Please enter the string:");
String input = scanner.nextLine();
for (int i = 0; i < input.length(); i++) {
    if (!alphabet.contains(input.charAt(i))) {
        System.out.println("This input doesn't exist in the alphabet! Please enter a proper input:");
        input = scanner.nextLine();
        i = 0;
    }
}
nfa.acceptString(input);
scanner.close();
```

دریافت ترنزیشن‌ها و رشته ورودی از کاربر

- **توضیحات متد scanTransitions:**

در این تابع، به تعداد ترنزیشن‌ها از کاربر ورودی دریافت می‌کنیم و طوری این ورودی را تقسیم بندی می‌کنیم که استیت مبدا, حرف ورودی و استیت مقصد در String[3] line ذخیره شوند. همچنین چک می‌کنیم که ترنزیشن‌های ورودی معتبر باشند؛ یعنی حرفی که در الفبا وجود ندارد (به غیر از لاندا و اپسیلون) و یا استیتی که در استیت‌ها موجود نیست به عنوان ورودی داده نشود. در غیر این صورت دوباره این ترنزیشن را از کاربر دریافت می‌کنیم. سپس بررسی می‌کنیم که اگر استیت مبدا قبلا در کلیدهای مپ وجود داشته است چک می‌کنیم که آیا این کاراکتر عضو value های آن هست یا نه. اگر این کاراکتر موجود بود، استیت مقصد را به لیست اضافه می‌کنیم. اگر در هر مرحله مپ یا لیست موردنظر ما موجود نبود، آن را با new کردن تعریف می‌کنیم. در نهایت کلید مپ transitions استیت مبدا و valueآن به صورت مپی باشد که کلید آن کاراکتر ورودی و value آن لیستی از استیت‌های مقصد باشد.

```
static void scanTransitions(int transitionSize, ArrayList<String> states, ArrayList<Character> alphabet,
                            Scanner scanner, Map<String, Map<Character, ArrayList<String>>> transitions) {
    for (int i = 0; i < transitionSize; i++) {
        String[] input = scanner.nextLine().split( regex: "[f\\(|δ\\(|=|\\)|,");
        String[] line = new String[3];
        int index = 0;
        for (int j = 0; j < input.length && index < 3; j++) {
            if (!input[j].equals("")) {
                line[index] = input[j];
                index++;
            }
        }
        if ((!alphabet.contains(line[1].charAt(0)) && !line[1].equals("λ") && !line[1].equals("ε"))
                || !states.contains(line[0]) || !states.contains(line[2])) {
            System.out.println("Invalid transition! Please re-enter this transition:");
            i--;
            continue;
        }
        ArrayList<String> newArray = new ArrayList<>();
        newArray.add(line[2]);
        if (transitions.containsKey(line[0])) {
            Map<Character, ArrayList<String>> curr = transitions.get(line[0]);
            if (curr.containsKey(line[1].charAt(0))) {
                curr.get(line[1].charAt(0)).add(line[2]);
            } else {
                curr.put(line[1].charAt(0), newArray);
            }
        } else {
            Map<Character, ArrayList<String>> newMap = new HashMap<>();
            newMap.put(line[1].charAt(0), newArray);
            transitions.put(line[0], newMap);
```

- **بخش گرافیکی کلاس NFA:**

```
public void paint(Graphics g) {
    Ellipse2D circle = new Ellipse2D.Double();
    Graphics2D g2 = (Graphics2D) g;
    g2.setColor(Color.black);
    for (int i = 0; i < states.size(); i++) {
        circle.setFrameFromCenter( centerX: 100 + i * 100,  centerY: 250,  cornerX: 125 + i * 100,  cornerY: 275);
        g2.draw(circle);
        g.drawString(states.get(i),  x: 95 + i * 100,  y: 250);
        statesPositions.put(states.get(i), (float) 100 + i * 100);
        if (finalStates.contains(states.get(i))) {
            circle.setFrameFromCenter( centerX: 100 + i * 100,  centerY: 250,  cornerX: 120 + i * 100,  cornerY: 270);
            g2.draw(circle);
        }
        if (states.get(i).equals(initialState)) {
            Line2D line = new Line2D.Float( x1: 50 + i * 100,  y1: 250,  x2: 75 + i * 100,  y2: 250);
            g2.draw(line);
            g2.drawString( str: "start",  x: 25 + i * 100,  y: 250);
            Polygon triangle = new Polygon(new int[]{65 + i * 100, 65 + i * 100, 75 + i * 100},
                    new int[]{240, 260, 250},  npoints: 3);
            g2.drawPolygon(triangle);
        }
    }
```

**ترسیم استیت‌ها**

- ابتدا به ازای هر استیت اتوماتا یک دایره رسم می‌کنیم و درون آن اسم استیت را نشان می‌دهیم. اگر استیت مورد نظر استیتی نهایی بود، دایره‌ای دیگر هم درون آن اضافه می‌کنیم تا پایانی بودن این استیت در شکل نشان داده شود. اگر استیت مورد نظر استیت ابتدایی بود، در کنار آن یک فلش (تشکیل شده از یک خط و یک مثلث) می‌گذاریم و عبارت start را نمایش می‌دهیم. از آن‌جایی که برای نشان دادن ترنزیشن‌ها به مکان هر استیت نیاز داریم، هر استیت و مکان x آن را در Map<String, Float> statesPositions ذخیره می‌کنیم.

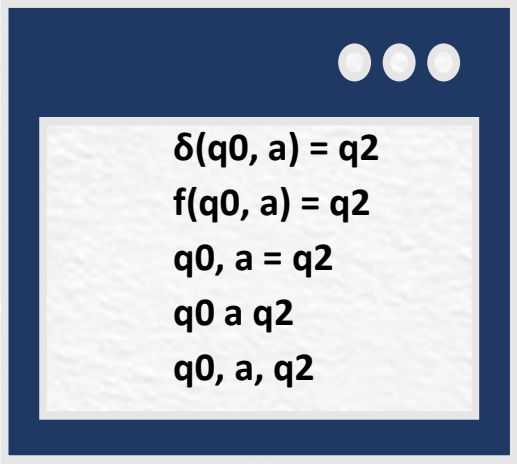
- به ازای هر ترنزیشن، یک خط شکسته بین استیت مبدا و مقصد و همچنین یک فلش(خط و مثلث) که به استیت مقصد اشاره می‌کند، ترسیم کردیم. و در نهایت کنار هر ترنزیشن حرف ورودی را هم نشان می‌دهیم.

```
int num = 0;
for (String s : transitions.keySet()) {
    for (Character c : transitions.get(s).keySet()) {
        for (int i = 0; i < transitions.get(s).get(c).size(); i++) {
            float endPosition = statesPositions.get(transitions.get(s).get(c).get(i));
            Line2D line = new Line2D.Float(statesPositions.get(s),  y1: 225,
                    x2: (statesPositions.get(s) + endPosition) / 2,  y2: 200 - 5 * num);
            g2.draw(line);
            Line2D line2 = new Line2D.Float( x1: (statesPositions.get(s) + endPosition) / 2,
                    y1: 200 - 5 * num, endPosition,  y2: 225);
            g2.draw(line2);
            Polygon triangle = new Polygon(new int[]{(int) endPosition + 10, (int) endPosition,
                    (int) endPosition - 10}, new int[]{215, 225, 215},  npoints: 3);
            g2.drawPolygon(triangle);
            g.drawString(String.valueOf(c),
                    x: (int) ((statesPositions.get(s) + endPosition) / 2) + 5,  y: 200 - 5 * num);
            num++;
        }
    }
```

**ترسیم ترنزیشن‌ها**

- **نحوه اجرا و ران گرفتن کدها:**

- کاربر باید ابتدا در یک خط حروف الفبای زبان مورد نظرش را با فاصله وارد کند؛ به طور مثال: a b c
- در خط بعد، کاربر باید اسم استیت‌های اتوماتای موردنظر را با فاصله وارد کند؛ به طور مثال: q0 q1 q2 q3
- در خط بعد کاربر باید یک استیت را به عنوان استیت ابتدایی وارد کند. این استیت باید حتماً در استیت‌های اتوماتا موجود باشد وگرنه باید کاربر دوباره آن را ورود کند.
- در خط بعد کاربر باید اسم استیت‌های فاینال را با فاصله وارد کند. مشابه قسمت قبل، اگر کاربر استیت‌هایی ناموجود را در این مرحله وارد کند دوباره از او ورودی گرفته می‌شود. به طور مثال: q2 q1
- سپس کاربر باید تعداد ترنزیشن‌های مدّنظرش را به صورت int وارد کند. فرض کنیم این مقدار n باشد.
- در n خط بعدی، کاربر باید ترنزیشن‌های مورد نظرش را به یکی ازفرم‌های رو به رو وارد کند:




- در خط آخر، کاربر رشته موردنظرش را وارد می‌کند. حتما باید این رشته از حروف الفبای دریافت شده تشکیل شده باشد وگرنه این ورودی دوباره از کاربر گرفته می‌شود.
- در نهایت در ترمینال به کاربر نتیجه بررسی پذیرفته شدن رشته در این NFA را به کاربر نشان می‌دهیم و همچنین NFA وارد شده را به صورت گرافیکی به کاربر نمایش می‌دهیم.

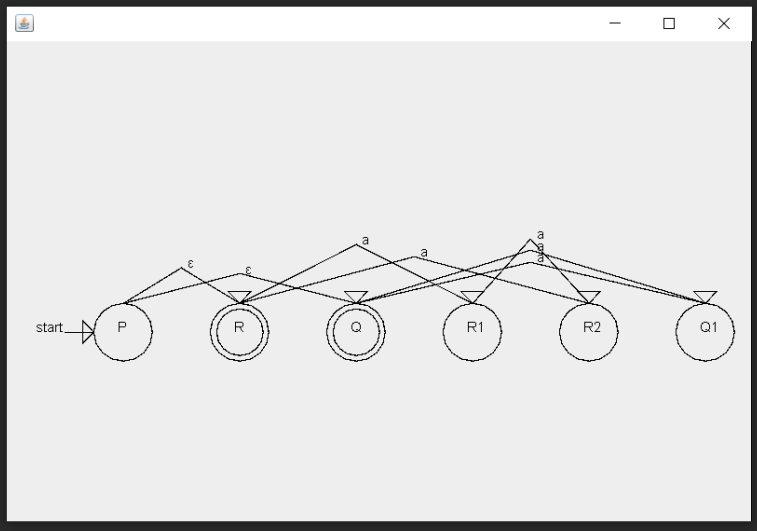
# نمونه‌های ورودی و خروجی:

## • NFA اوّل:

- Σ = { a }
- Q = { P, R, Q, R1, R2, Q1}
- q0 = P
- F = {R, Q}
- 7 transitions:
- δ(P, ε) = Q
- δ(P, ε) = R
- δ(Q, a) = Q1
- δ(Q1, a) = Q
- δ(R, a) = R1
- δ(R1, a) = R2
- δ(R2, a) = R

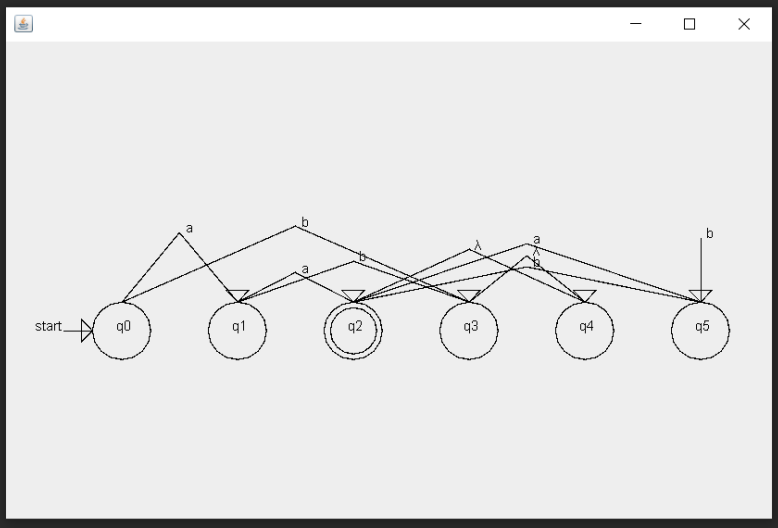
این اتوماتا رشته‌های عضو این زبان را $L = \{ a^n \mid n \geq 0 ; n\%3 = 0 \text{ or } n\%2 = 0\}$ می‌پذیرد.

```
**********WELCOME**********
Please enter the NFA alphabet(Σ):
a
Please enter the NFA states:
P R Q R1 R2 Q1
Please enter the initial state:
p
This state doesn't exist! Please enter a proper initial state:
P
Please enter the final states:
q r
Some states don't exist! Please enter proper final states:
Q R
Please enter number of transitions:
7
Please enter the NFA transitions:
*** For example:δ(q0, a) = q2 or f(q0, a) = q2
δ(P, ε) = Q
δ(P, ε) = R
δ(Q, a) = Q1
δ(Q1, a) = Q
δ(R, a) = R1
δ(R1, a) = R2
δ(R2, a) = R
Please enter the string:
aaaaa
This NFA doesn't accept this string.
aaaaa ∉ language
```



## • NFA دوم:

```
**********WELCOME**********
Please enter the NFA alphabet(Σ):
a b
Please enter the NFA states:
q0 q1 q2 q3 q4 q5
Please enter the initial state:
q0
Please enter the final states:
q2
Please enter number of transitions:
9
Please enter the NFA transitions:
*** For example:δ(q0, a) = q2 or f(q0, a) = q2
δ(q0, a) = q1
δ(q0, b) = q3
δ(q1, a) = q2
δ(q2, b) = q5
δ(q3, λ) = q4
δ(q3, b) = q1
δ(q4, λ) = q2
δ(q5, a) = q2
δ(q5, b) = q5
Please enter the string:
aabbbbbbbbbbb
This NFA doesn't accept this string.
aabbbbbbbbbbb ∉ language
```




Σ = { a , b }

Q = {q0, q1, q2, q3, q4, q5}

q0 = q0

F = {q2}

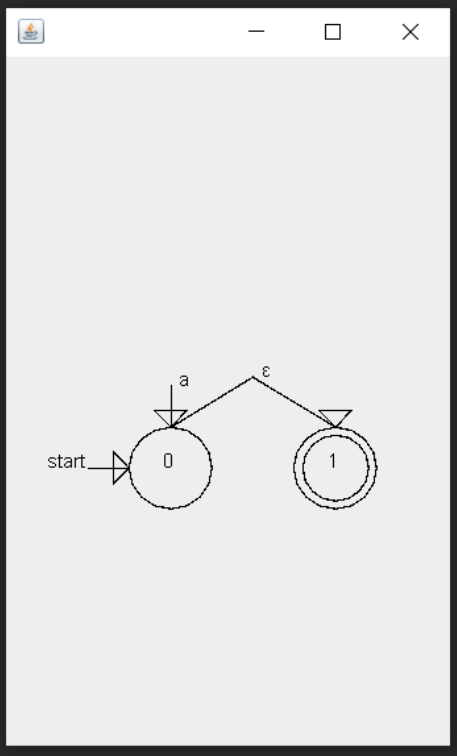
9 transitions:

- δ(q0, a) = q1
- δ(q0, b) = q3
- δ(q1, a) = q2
- δ(q2, b) = q5
- δ(q3, λ) = q4
- δ(q3, b) = q1
- δ(q4, λ) = q2
- δ(q5, a) = q2
- δ(q5, b) = q5

## • NFA سوم:

```
**********WELCOME**********
Please enter the NFA alphabet(Σ):
a
Please enter the NFA states:
0 1
Please enter the initial state:
q0
This state doesn't exist! Please enter a proper initial state:
0
Please enter the final states:
1
Please enter number of transitions:
2
Please enter the NFA transitions:
*** For example:δ(q0, a) = q2 or f(q0, a) = q2
0 a 0
0 ε 1
Please enter the string:

This NFA accepts this string.
ε ∈ language
```




Σ = { a}

Q = {0, 1}

q0 = 0

F = {1}

2 transitions:

- 0 a 0
- 0 ε 1

این NFA رشته‌های پذیرفته شده در زبان $L = \{a^n \mid n \geq 0\}$ را قبول می‌کند. پس رشته وارد شده (اپسیلون) را هم می‌پذیرد.

## • NFA چهارم

$\Sigma = \{ 0 , 1 \}$

$Q = \{q0, q1, q2\}$

$q0 = q0$

$F = \{q2\}$

4 transitions:

$\delta(q0, 0) = q0$

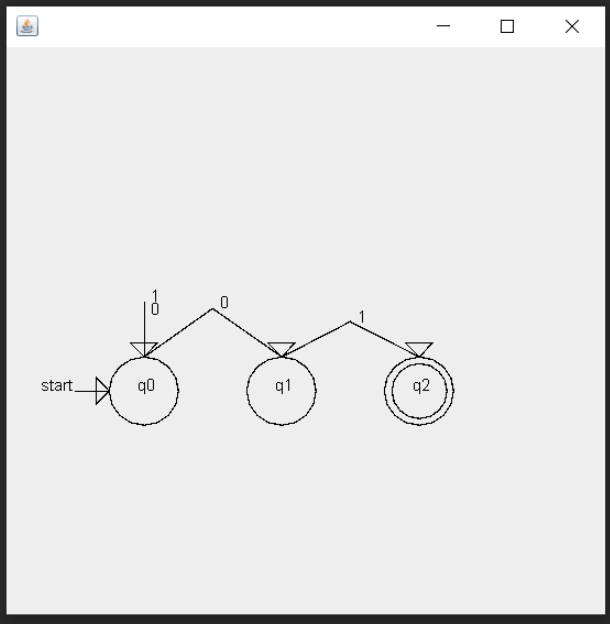
$\delta(q0, 0) = q1$

$\delta(q0, 1) = q0$

$\delta(q1, 1) = q2$

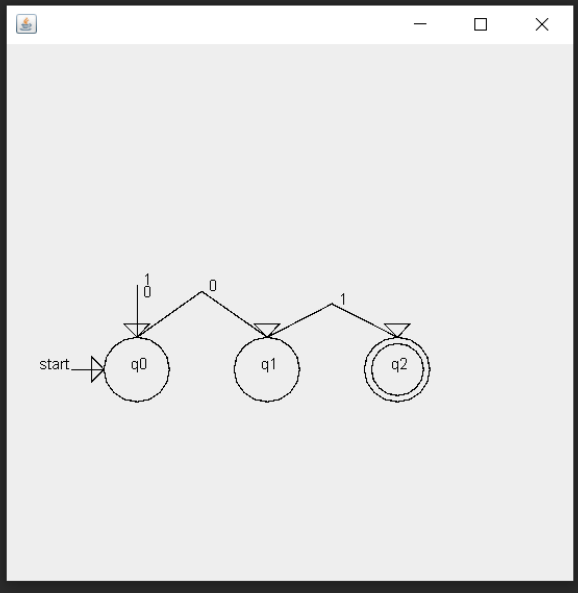
این NFA رشته‌هایی که با ۰۱ تمام می‌شوند را می پذیرد.

```
**********WELCOME**********
Please enter the NFA alphabet(Σ):
0 1
Please enter the NFA states:
q0 q1 q2
Please enter the initial state:
q0
Please enter the final states:
q2
Please enter number of transitions:
4
Please enter the NFA transitions:
*** For example:δ(q0, a) = q2 or f(q0, a) = q2
δ(q0, 0) = q0
δ(q0, 0) = q1
δ(q0, 1) = q0
δ(q1, 1) = q2
Please enter the string:
0101
This NFA accepts this string.
0101 ∈ language
```




**$\delta(q0, 0) = q0$ -> $\delta(q0, 1) = q0$ -> $\delta(q0, 0) = q1$ -> $\delta(q1, 1) = q2$**

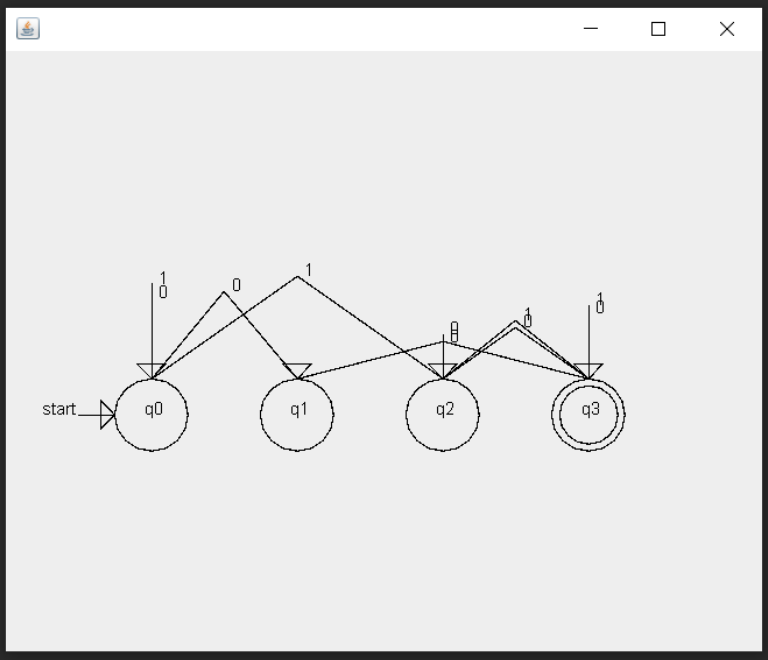
```
**********WELCOME**********
Please enter the NFA alphabet(Σ):
0 1
Please enter the NFA states:
q0 q1 q2
Please enter the initial state:
q0
Please enter the final states:
q2
Please enter number of transitions:
4
Please enter the NFA transitions:
*** For example:δ(q0, a) = q2 or f(q0, a) = q2
δ(q0, 0) = q0
δ(q0, 0) = q1
δ(q0, 1) = q0
δ(q1, 1) = q2
Please enter the string:
aaa
This input doesn't exist in the alphabet! Please enter a proper input:
000
This NFA doesn't accept this string.
000 ∉ language
```




**رشته 000 پذیرفته نمی‌شود.**

## • NFA پنجم:

```
**********WELCOME**********
Please enter the NFA alphabet(Σ):
0 1
Please enter the NFA states:
q0 q1 q2 q3
Please enter the initial state:
q0
Please enter the final states:
q3
Please enter number of transitions:
10
Please enter the NFA transitions:
*** For example:δ(q0, a) = q2 or f(q0, a) = q2
δ(q0, 0) = q0
δ(q0, 0) = q1
δ(q0, 1) = q0
δ(q0, 1) = q2
δ(q1, 0) = q3
δ(q2, 0) = q2
δ(q2, 0) = q3
δ(q2, 1) = q3
δ(q3, 0) = q3
δ(q3, 1) = q3
Please enter the string:
10
This NFA accepts this string.
10 ∈ language
```




Σ = { 0, 1}

Q = {q0, q1, q2, q3}

q0 = q0

F = {q2}

10 transitions:

- δ(q0, 0) = q0
- δ(q0, 0) = q1
- δ(q0, 1) = q0
- δ(q0, 1) = q2
- δ(q1, 0) = q3
- δ(q2, 0) = q2
- δ(q2, 0) = q3
- δ(q2, 1) = q3
- δ(q3, 0) = q3
- δ(q3, 1) = q3