Relatório Final projeto - Linguagens formais e autômatos

Gabriel de Oliveira Almeida Gustavo Lopes Santana João Victor Menezes

1 Introdução

Na disciplina de linguagens formais e autômatos foi proposto elaborar uma ferramenta junto a esse respectivo relatório, que diz a respeito de visualização e validação de autômatos de linguagens formais representadas por autômatos, gramáticas ou expressões regulares. A ferramenta será desenvolvida em um site Web, na qual será escrita em *javascript*, junto a API *JsPlumb Toolkit*.

2 Expressão regular

Uma expressão regular é uma das possíveis formas de se definir uma linguagem regular, a mesma é definida como uma cadeia de caracteres que possuem operadores e constantes, o recurso do *javascript* utilizado para a verificação da expressão regular foi a função **RegEx**('expressao'), no qual o algoritmo será explicitado abaixo.

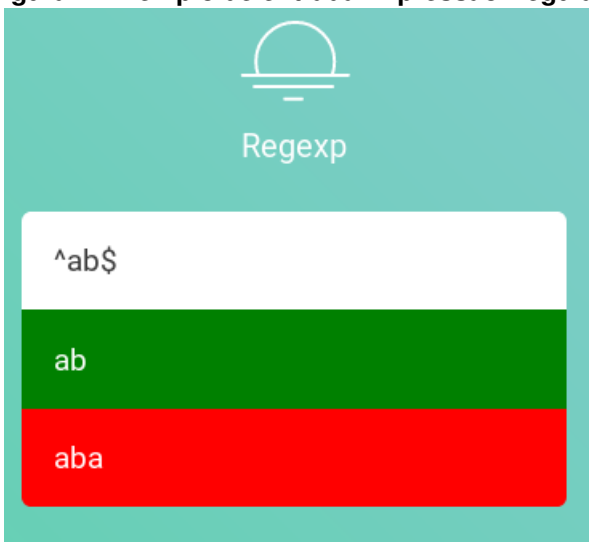
```
function myFunction(){
    var x = document.getElementById("expressao");
    var y = document.getElementById("entrada");
    var regExp = new RegExp(x.value);
    var entradas = y.value;
    if (regExp.test(entradas)){
        var x = "Valida";
    }
    else var x = "Invalida";
   alert("Entrada "+x);
}
```

Logo, no site podemos analisar que em uma expressão ^ ab$, e a partir da entrada, obtemos verde para entrada correta, caso contrario, cor vermelha, como demonstrada na Figura (1).

3 Autômato

No projeto separamos os arquivos .js dos autômatos em AFD e AFND onde estão os métodos para cada tipo de autômato e existe dois arquivos que faz a comunicação entre os autômatos e a biblioteca utilizada para a representação gráfica deles que é o delegateAFD e delegateAFND. Alguns métodos

Figura 1. Exemplo de entrada Expressão Regular



não se difere entre os dois autômatos, nesse momento será explicado os algoritmos que são iguais para ambos.

```
this.transitions = {};
this.startState = useDefaults ? 'start' : null;
this.acceptStates = useDefaults ? ['accept'] : [];
```

No trecho de código acima mostra as variáveis que serão utilizadas.

- Transitions é um objeto que será guardado o estado referente, o caractere da transição e o estado que está conectado.
- startState é uma variável que guarda o estado inicial.
- acceptStates é um vetor que guarda os estados finais do autômato

Para autômatos determinísticos:

Função que adicionar a transição de estados. Recebe o estado inicial, o caractere e o estado final da transição.

```
AFD.prototype.addTransition = function(stateA, character, stateB) {
  if (!this.transitions[stateA]) {this.transitions[stateA] = {};}
  this.transitions[stateA][character] = stateB;
  return this;
};
```

Funções para remoção de um estado ou uma transição. Se for por estado terá que remover todas as transições quem vem pra ele ou que saiam dele.

```
AFD.prototype.removeTransitions = function(state) {
  delete this.transitions[state];
  var self = this;
  $.each(self.transitions, function(stateA, sTrans) {
    $.each(sTrans, function(char, stateB) {
      if (stateB === state) {self.removeTransition(stateA, char);}
    });
  });
  return this;
};

AFD.prototype.removeTransition = function(stateA, character) {
  if (this.transitions[stateA]) {delete this.transitions[stateA][character];}
  return this;
};
```

```
this.processor = {
  input: null,  //string
  inputLength: 0, //tamanho da string
  state: null, //estado do automato em que encontra-se
  inputIndex: 0, //em que posição da string está sendo analisada
  status: null, //Active(estado não final) ou Accept (estado final)
};
```

Função accepts recebe uma variável string input, antes de começar o teste é feito o armazenamento das informações na função stepInit, Processor é um objeto onde tem as seguintes variáveis.

Na função Step analisa o estado em que se encontra e vê se tem uma transição para o caracter em que está se houver o State será atualizado para o próximo estado e se for o último caractere será retornado o status Accept, agora se não houver transição para aquele caractere será rejeitado, também tem o caso de ter a transição para o estado e esse estado não é Accept(Final) e será rejeitado.

```
AFD.prototype.accepts = function(input) {
  var _status = this.stepInit(input);
  while (_status === 'Active') {_status = this.step();}
  return _status === 'Accept';
};

AFD.prototype.stepInit = function(input) {
  this.processor.input = input;
  this.processor.inputLength = this.processor.input.length;
  this.processor.inputIndex = 0;
  this.processor.state = this.startState;
  this.processor.status = (this.processor.inputLength === 0 && this.acceptStates.indexOf(this.processor.state) >= 0) ? 'Accept' : 'Active';
  return this.processor.status;
};
AFD.prototype.step = function() {
  if ((this.processor.state = this.transition(this.processor.state, this.processor.input.substr(this.processor.inputIndex++, 1))) === null) {this.processor.status = 'Reject';}
  if (this.processor.inputIndex === this.processor.inputLength) {this.processor.status = (this.acceptStates.indexOf(this.processor.state) >= 0 ? 'Accept' : 'Reject');}
  return this.processor.status;
};

});
```

Então será analisado toda a string enquanto não acabar a string será analisada e se no final dela parar em um estado com o status Accept a string será aceita, senão será recusada

Para Autômatos não determinísticos

Função que adicionar a transição de estados. Recebe o estado inicial, o caractere e o estado final da transição.

```
AFND.prototype.addTransition = function(stateA, character, stateB) {
  if (!this.transitions[stateA]) {this.transitions[stateA] = {};}
  if (!this.transitions[stateA][character]) {this.transitions[stateA][character] = [];}
  this.transitions[stateA][character].push(stateB);
  return this;
};
```

Funções para remoção de um estado ou uma transição. Se for por estado terá que remover todas as transições quem vem pra ele ou que saiam dele.

```
AFND.prototype.removeTransitions = function(state) {
  delete this.transitions[state];
  var self = this;
  $.each(self.transitions, function(stateA, sTrans) {
    $.each(sTrans, function(char, states) {
      if (states.indexOf(state) >= 0) {
        self.removeTransition(stateA, char, state);
      }
    });
  });
  return this;
};

AFND.prototype.removeTransition = function(stateA, character, stateB) {
  if (this.hasTransition(stateA, character, stateB)) {
    this.transitions[stateA][character].splice(this.transitions[stateA][character].indexOf(stateB), 1);
  }
  return this;
};
```

Verificação se uma cadeia de caracteres é válida no autômato descrito.

Função accepts recebe uma variável string input, antes de começar o teste é feito o armazenamento das informações na função stepInit, Processor é um objeto onde tem as seguintes variáveis.

```
this.processor = {
  input: null, //string
  inputIndex: 0, //em que posição da string está sendo analisada
  inputLength: 0, //tamanho da string
  states: [], //estados que precisam ser checados
  status: null,  //Active(estado não final) ou Accept (estado final)
  nextStep: null //epsilons ou input
};
```

A função step verifica se há primeiro transições em vazio e chama a função followEpsilonTransitions basicamente ela verifica todas as transições daquele estado procurando transições vazias se houver colocará no vetor States onde ficam os estados que precisam ser verificados. Feito isso o próximo passo é verificar se o caractere analisado existe transição para ele, seguindo o vetor States do começo para o fim. Por fim a função updateStatus irá verificar se o último caracter parou em um Accept (estado final) e será aceito ou em um Active que será rejeitado.

```
AFND.prototype.accepts = function(input) {
  var _status = this.stepInit(input);
  while (_status === 'Active') {_status = this.step();}
  return _status === 'Accept';
};

AFND.prototype.stepInit = function(input) {
  this.processor.input = input;
  this.processor.inputLength = this.processor.input.length;
  this.processor.inputIndex = 0;
  this.processor.states = [this.startState];
  this.processor.status = 'Active';
  this.processor.nextStep = 'epsilons';
  return this.updateStatus();
};
AFND.prototype.step = function() {
  switch (this.processor.nextStep) {
    case 'epsilons':
      this.followEpsilonTransitions();
      this.processor.nextStep = 'input';
      break;
    case 'input':
      var newStates = [];
      var char = this.processor.input.substr(this.processor.inputIndex, 1);
      var state = null;
      while (state = this.processor.states.shift()) {
        var tranStates = this.transition(state, char);
        if (tranStates) {$.each(tranStates, function(index, tranState) {
            if (newStates.indexOf(tranState) === -1) {newStates.push(tranState);}
        });}
      };
      ++this.processor.inputIndex;
      this.processor.states = newStates;
      this.processor.nextStep = 'epsilons';
      break;
  }
  return this.updateStatus();
};
```

4 Gramática

Uma gramática define a estrutura geral de formação de uma sentença válida para uma linguagem, em que uma linguagem formal pode ser representada por um conjunto de regras que especifica a formação de cadeias, podendo ser caracterizadas por

$$G = (V, T, P, S)$$

onde,

V: representa as variáveis (Não terminal).

T: representa os terminais - simbolos que formarão as cadeias da linguagem.

P: representa as regras de produção, para definir a linguagem.

S: representa o simbolo inicial.

Assim, foi desenvolvido uma ferramenta capaz de a partir da definição de G verificar a entradas de cadeias inserida pelo o usuário com o intuito de verificar se pertence ou não a linguagem, para isso,

```
AFND.prototype.followEpsilonTransitions = function() {
  var self = this;
  var changed = true;
  while (changed) {
    changed = false;
    $.each(self.processor.states, function(index, state) {
      var newStates = self.transition(state, '');
      if (newStates) {$.each(newStates, function(sIndex, newState) {
          var match = false;
          $.each(self.processor.states, function(oIndex, checkState) {
            if (checkState === newState) {
              match = true;
              return false;
            }
          });
          if (!match) {
            changed = true;
            self.processor.states.push(newState);
          }
      });}
    });
  }
};
AFND.prototype.updateStatus = function() {
  var self = this;
  if (self.processor.states.length === 0) {
    self.processor.status = 'Reject';
  }
  if (self.processor.inputIndex === self.processor.inputLength) {
   $.each(self.processor.states, function(index, state) {
      if (self.acceptStates.indexOf(state) >= 0) {
        self.processor.status = 'Accept';
        return false;
      }
    });
  }
  return self.processor.status;
};
```

após a entrada do usuário, um vetor contendo todas as derivações de um respectivo não terminal, na qual também pertence a um campo do vetor, foi preenchido, desse modo, o algoritmo, a partir do simbolo inicial, realiza todas as verificações recursivamente, percorrendo o vetor formado, substituindo os não terminais por suas derivações até o momento em que a cadeia dada pelo o usuário seja idêntica a cadeia formada pela a recursão. As derivações podem possuir ou não possuir não-terminais como sufixo ou prefixo.

Podemos ver pelo o algoritmo abaixo, que ao verificar a derivação, encontramos o não-terminal e desse modo, sabemos qual derivação ocorrerá na gramatica, se a esquerda ou direita, a recursão é interrompida quando a partir das substituições encontra a cadeia de caracteres dada pelo o usuário, ou quando muitas das possibilidades são testadas, uma vez que, possui função para otimizar a busca da entrada, tal como a função *verificarRecursao()*, em que a partir da formação da palavra pela gramatica verifica se o prefixo e sufixo estão iguais a palavra de entrada, caso seja diferente, testará outra derivação, e outro controle é dado a partir do tamanho das palavras, aceitando a gramatica formar palavras com tamanho menor ou igual a entrada, desse modo, podemos simplificar demasiadamente a busca pela solução.

Vale ressaltar que, para evitarmos gramaticas que possui ciclos, tal como: $A \longrightarrow S$ e $S \longrightarrow A$, utilizamos a variável *expAnt*, que armazenará a expressão anterior, sendo assim, não deixará que a gramatica forme continuamente palavras repetidas.

```
        function resolver(exp, naoTerminal, entrada, expAnt){

            var tam = tabelaGramatica.length;
            if(exp == entrada && naoTerminal == "-0") return true;

            if(exp.length > entrada.length+1 )  return false;
            for (var i = 0; i < tam; i++) {

                if(tabelaGramatica[i].naoTerminal == naoTerminal){
                    for(var j = 0; j<tabelaGramatica[i].expressao.length; j
                       ++){
                        var der = tabelaGramatica[i].expressao[j];
                        var naoTerminalDer = verificarDerivacao(der);
                        var novoexp = exp.replace(naoTerminal, der);
                        console.log(novoexp + " | " + expAnt);
                        if(!verificarRecursao(novoexp, entrada)) continue;
                        if(expAnt == novoexp) continue;
                        if(resolver(novoexp, naoTerminalDer[0], entrada,
                           exp)){
                            return true;
                        }

                    }
                }
            }
            return false;
        }
```

Para a formação da gramatica, serão aceitas como não-terminais todas as letras maiúsculas e símbolos terminais são todos os números e letras minúsculas, e para o teste da gramatica, serão aceitas múltiplas entradas ou apenas uma, na qual todas devem possuir letras minusculas. Analisando o site vemos que a partir da entrada, obtemos verde para entrada correta, caso contrario, cor vermelha, como demonstrado na Figura (2).

5 Conversões

5.1 Gramática regular para autômato finito

A conversão só é possível para gramaticas regular linear a direita (GRUD), transformando-o em autômato finito não determinístico (AFND) apenas, antes de tudo temos que entregar a pagina web do autômato sua estrutura dado por:

```
{
        "type": "AFND",
        "afnd":{
        "transitions": transitions,
```

Figura 2. Entrada e teste gramática


```
            "startState": startState,
                "acceptStates": acceptStates
        },
        "states":states,
        "transitions":transitionslabel
}
```

Assim, através de uma expressão regular das derivações da tabela é verificado todas as entradas ao decorrer da conversão com intuito de verificar se realmente é um GRUD, caso contrário, será interrompida imediatamente, outra interrupção é dado por Não-Terminais presentes nas derivações mas não possuem derivações, tal como:

$$A \rightarrow aB$$

Vale ressaltar que todos os Não-terminais representados por letras maiúsculas são convertidos em $q\mathbb{N}$, e para manter a consistência entre a gramatica e o autômato devemos seguir as seguintes regras aplicadas pelo o algoritmo:

- Se, Não-terminal *A* deriva em carácter a.Não-Terminal *B*, então a transição de um Estado *q0* para *q1* com valor a é criado.
- Se, Não-Terminal *A* deriva em Não-Terminal *B*, então a transição de um Estado *q0* para *q1* com valor vazio é criado.
- Se, Não-Terminal *A* deriva em a, então a transição de um Estado *q0* para um estado final *q1* com valor a é criado.
- Se, Não-Terminal *A* deriva em vazio, então a transição de um Estado *q0* para um estado final com valor vazio é criado.

Após o preenchimento da estrutura, é transformada em string e adicionada a URL e a aplicação do autômato através da função *loadSerializedFSM(estrutura em string)* lerá cada campo da estrutura onde

contém o afnd no qual possui as transições (transitions), estados finais (acceptStates), estado inicial (startState), e também todos os estados (states) e transições (transitions) - vetor que possui todas as coordenadas da tela onde será plotada cada estado. Assim, carregará o automato e exibirá automaticamente ao usuário.

5.2 Autômato finito para gramática regular

A gramática regular aceita para essa conversão é dada apenas pelas gramáticas regular unitária a direita (GRUD), sabendo disso, dado um autômato finito, sendo que a partir da estrutura definida para o autômato, podemos percorrer todas as transições para que possamos preencher uma matriz $N \times 2$ (*Não-Terminais* $\times$ *derivações*), que corresponderá a gramática.

Inicialmente, a estrutura é convertida em 'string', para que possamos enviar através da URL, uma vez que as aplicações estão em páginas web diferentes, e assim teremos acesso ao autômato finito. Ao abrir a aplicação, modificações na URL são feitas afim de facilitar o reconhecimento do automato finito, utilizando o regex e a função match que consiste em recuperar as correspondência ao testar a string, tudo com a finalidade de encontrar padrões.

Os autômatos não determinístico apresenta um vetor de estados para cada transição, enquanto para transição do determinístico é utilizado apenas um estado, a partir disso, o algoritmo para criação da tabela é chamado de acordo com qual autômato é recebido.

Haverá a leitura de toda a 'string' correspondente ao autômato e também a substituição por vazios, indicando que os estados substituídos foram reconhecidos e adicionados a tabela, como podemos observar no código abaixo para a manipulação de uma AFD.

```
function getDerivacaoAFD(der, terminal){
    while(der.charAt(0) != "}"){
        var regra = "([a-zA-Z]|[0-9])\\:(q\\d\\d*)";
        var reg = new RegExp(regra);

        var derivacao = der.match(reg);
        der = der.replace(reg, "");

        //ajudar a terminal o laço de repeticao
        if(der.charAt(0) == ',' || der.charAt(0) == '{')
                der = der.substring(1 , der.length);

        addTabela(numParaLetra(terminal), (derivacao[1] + numParaLetra(
           derivacao[2])));
    }
    return der;
```

Os estados presentes no autômato são reconhecidos por qK, sendo $K \in \mathbb{N}$, e são convertidos pela função *numParaLetra(não-terminal)* nas letras presentes no alfabeto a partir da tabela ASCII.

As derivações são formadas pela a seguinte regras presente no algoritmo:

- Se, estado *q0* possui transição para o estado *q1* com valor a, então é criado a derivação de A $\rightarrow$ aB

- Se, estado *q0* possui transição para o estado *q1* com valor vazio, então é criado a derivação de A → B.
- Se, estado *q0* é estado final, então é criado a derivação de A → vazio.

Assim, podemos garantir representação idêntica do autômato finito, e por fim podemos preencher a tabela presente a interface de usuário e a conversão é concluída.

5.3 AFND para AFD

No projeto foi utilizado o algoritmo passado em sala de aula que usa a tabela de transições com os estados formados. Utilizamos uma estrutura para representar os novos estados criados onde cada estado contem quais estados anteriores formaram ele. E uma estrutura de novas transições já convertidos para AFD

```
var c = 0; //index carcteres
var arrNew = null;
var q;
var estadosTransitivos = Object.keys(transitions);
for(var i = 0; i< Object.keys(this.newStates).length; i++){ //estados
   criados

    while(c<caracteres.length){ //caracteres
    for (var e = 0; e  <  this.newStates['q'+i].length; e++) { //estados q
       preciso checar
      if(estadosTransitivos.includes(this.newStates['q'+i][e])){ //
         verifica se existe transição daquele estado q precisa checar
       temp = transitions[ this.newStates['q'+i][e] ][ caracteres[c] ] ;
       if( temp != undefined){
        (arrNew === null) ? arrNew = temp : arrNew = _.union(arrNew,temp);
      }
    }
  }
  if(arrNew != null){
    arrNew = arrNew.sort();
    q = checaNovoEstado(arrNew); //retorna a posicao do novo estado criado,
        caso nao exista inda
    if(q){
      if(!newTransitions['q'+i]) newTransitions['q'+i] =  {};
        newTransitions['q'+i][caracteres[c]]= [];
        newTransitions['q'+i][caracteres[c]].push(q);


      if(verFinais(arrNew,estadosfinais)) this.newAcceptStates.push(q);
    };
  };
  c++;
```

```
  arrNew = null;
};
```

5.4 Autômato finito para XML

Foi utilizado a estrutura XML que é o objeto em Javascript.
Essa primeira parte é o cabeçalho do XML que o JFlap usa.

```
        //criando o obj XML
        var parser = new DOMParser()
        var xml  = parser.parseFromString('<?xml version="1.0" encoding="
           utf-8" standalone="no"?><structure></structure>', "application/
           xml");
        //-----
        var newElement

        newElement = xml.createElement("type"); //cria um novo node
        xml.getElementsByTagName("structure")[0].appendChild(newElement);//
           aplica o novo node criado em um outro
        xml.getElementsByTagName("type")[0].appendChild(xml.createTextNode(
           'fa')); //atributo em um node

        newElement = xml.createElement("automaton");
        xml.getElementsByTagName("structure")[0].appendChild(newElement);
```

Depois vem a parte da tag ¡states¿

```
        $.each(model.states,function(state){

                if(state === 'q0'){
                        model.states[state].top = 55;
                        model.states[state].left = 55;
                        model.states[state].startState = true;
                        console.log(model.states[state]);
                };
                console.log(i);
                newElement = xml.createElement("state");
                newElement.setAttribute("id",state.slice(1));
                newElement.setAttribute("name",state);
                xml.getElementsByTagName("automaton")[0].appendChild(
                   newElement);

                newElement = xml.createElement("x");
                newElement.appendChild(xml.createTextNode(model.states[
                   state].top + i * 51));
                xml.getElementsByTagName("state")[i].appendChild(newElement
                   );
```

```
                newElement = xml.createElement("y");
                newElement.appendChild(xml.createTextNode(model.states[
                    state].left + i * 71));
                xml.getElementsByTagName("state")[i].appendChild(newElement
                    );

                if(model.states[state].isAccept){
                        newElement = xml.createElement("final"); //  final
                        xml.getElementsByTagName("state")[i].appendChild(
                            newElement);
                }else if(model.states[state].startState){
                        newElement = xml.createElement("initial"); //
                            initial
                        xml.getElementsByTagName("state")[i].appendChild(
                            newElement);

                }
                i++;
        });
```

E por fim as transições.

```
        var str  = new XMLSerializer().serializeToString(xml);
        console.log(str);

        var blob = new Blob([str],{type: "application/xml"});
        var url = URL.createObjectURL(blob);
        console.log(url);
        decisao = confirm("Deseja fazer o download do arquivo?");
        var a = document.createElement('a');
        if(decisao){
                a.href = url;
                a.download = name;
                a.click();
        }
};
```

5.5 XML para autômato finito

Utilizamos de novo a estrutura XML, onde uma variável recebe esse objeto e percorremos esse objeto para criar o autômato.

```
function convertJSON(xml){
        transitions = {};
        acceptStates = [];

```

```
        var node = xml.getElementsByTagName("type")[0];
        node.childNodes[0].nodeValue;

        for(var i = 0; i< xml.getElementsByTagName("state").length; i++){

                node = xml.getElementsByTagName("state")[i];

                if(node.getElementsByTagName("initial")[0]){
                        startState = ('q'+node.getAttribute("id"));
                };
                if(node.getElementsByTagName("final")[0]){
                        acceptStates.push('q'+node.getAttribute("id"));
                };
        };

        for(var i = 0; i< xml.getElementsByTagName("transition").length; i
           ++){

                node = xml.getElementsByTagName("transition")[i];

                var stateA = 'q'+node.getElementsByTagName("from")[0].
                   childNodes[0].nodeValue;
                var stateB = 'q'+node.getElementsByTagName("to")[0].
                   childNodes[0].nodeValue;
                var character = node.getElementsByTagName("read")[0].
                   childNodes[0].nodeValue;

                if (!transitions[stateA]) {transitions[stateA] = {}};
                if (!transitions[stateA][character]) {transitions[stateA][
                   character] = []} ;
                transitions[stateA][character].push(stateB);
        };

        return (serializeJSON());
}
```

5.6 Autômato finito para Expressão Regular

Para fazer a conversão foi utilizado o algoritmo de Transitive Closure Metod que utiliza recursão para ir montando a expressão a partir do autômato dado.

```
for(var i=0;i<n;i++){
    for(var j=0;j<n;j++){
      if(i!=j)
        if(pegarTransi(65+i,65+j)!="")
          R[i][j][0] = pegarTransi(65+i,65+j);
        else R[i][j][0] = 'v';
```

```
      else if(i==j){
        if(pegarTransi(65+i,65+i)!="")
          R[i][i][0]= String.fromCharCode(1013) + "|" + pegarTransi(65+i
             ,65+i);
        else R[i][i][0] = 'v';
      }
    }
  }
  for(var k=1;k<n+1; k++){
    for(var i=0; i<n; i++){
      for(var j=0; j<n; j++){
        temp = R[i][j][k-1] + "|" + R[i][k-1][k-1]+"("+ R[k-1][k-1][k-1]+")
           *"+R[k-1][j][k-1];
        R[i][j][k] = simplificarEx(temp);
      }
    }
  }
```

5.7 Expressão Regular para Autômato finito

Para essa conversão foi necessário fazer o reconhecimento de parênteses, para gerar uma recursão a partir dele e reconhecer cada parte da expressão separadamente para facilitar na montagem do autômato ao final do algoritmo.

```
var estadoatu = iniestado;
        var automato = [];
        automato.push(iniestado);
        while(posleitura < entrada.length){
                if(entrada[posleitura]=='('){
                        var nova = leparenteses(entrada, posleitura+1); //
                           elimina parenteses
                        var autrec = [];
                        autrec = converter2automato(nova); //cria um
                           automato para a expressao entre parentes
                        posleitura += nova.length + 2;
                        var terminal = recuperarfinal(autrec); //estado
                           final
                        if(entrada[posleitura] == '*'){
                                terminal.push({trans:'',dest:estadoatu[0].
                                   pos,final:terminal[0].final,pos:terminal
                                   [0].pos});
                                estadoatu.push({trans:'',dest:terminal[0].
                                   pos-1,final:estadoatu[0].final,pos:
                                   estadoatu[0].pos});
                                posleitura++;
                        }
                        estadoatu[0].final = false;
```

```
17                          estadoatu  = terminal;
18                          automato = mergestados(automato,autrec); //junta os
                                dois automatos
19
20                          continue;
21                  }
22                  if(entrada[posleitura] == '|'){ //atualiza os estados para
                        um "ou"
23                          iniestado = estadoatu = pipe(iniestado,automato);
24                          automato = temp2;
25                  }
26                  else if (entrada[posleitura+1] == '*') { //cria uma
                        transição para o propio estado atual
27                          estadoatu.push({trans:entrada[posleitura],dest:
                                estadoatu[0].pos,final:estadoatu[0].final,pos:
                                estadoatu[0].pos})
28                          posleitura++;
29                  }
30                  else {
31                          var novoe = [];
32                          novoe.push({trans:'',dest:-1,final:true,pos:inc});
33                          inc++;
34                          estadoatu[0].final = false;
35                          estadoatu.push({trans:entrada[posleitura],dest:
                                novoe[0].pos,final:estadoatu[0].final,pos:
                                estadoatu[0].pos});
36                          estadoatu=novoe;
37                          automato.push(novoe);
38                  }
39                  posleitura++;
40          }
41          return automato;
```