Finite Automata to Regular Expressions using the State Elimination method

Eduardo Martins, João Cepa, Luís Carvalho, Vitor Esteves

Compiladores
Departamento de Informática
Faculdade de Engenharia da Faculdade do Porto

Abstract. The framework shall receive the FA in dot format and DSL code with rules for selecting each state to eliminate (influencing the ordering). The project includes parsing the dot file and representing the FA as a graph (it is suggested to use an existent graph library), defining the DSL and parsing the DSL code and then use the rules specified in the DSL code in the context of the state elimination method. The output of the framework is a regular expression representing the input FA language and possibly a report of the steps performed during the conversion.

1 Introdução

2 Análise Lexical

An **ID** is one of the following:

- Any string of alphabetic ([a-zA-Z\200-\377]) **characters**, **underscores** ('_') or **digits** ([0-9]), not beginning with a digit;
- a **numeral** [-]?(.[0-9]+ | [0-9]+(.[0-9]*)?);
- any **double-quoted** string ("...") possibly containing escaped quotes ('")1; an **HTML string** (<...>).

3 Análise Sintática

The following is an abstract grammar defining the DOT language. Terminals are shown in bold font and nonterminals in italics. Literal characters are given in single quotes. Parentheses (and) indicate grouping when needed. Square brackets [and] enclose optional items. Vertical bars | separate alternatives.

```
graph : [ strict ] (graph | digraph) [ ID ] '{' stmt_list '}'
stmt_list : [ stmt [ ';' ] stmt_list ]
stmt : node_stmt
     | edge_stmt
     | attr_stmt
     | ID '=' ID
     | subgraph
attr_stmt : (graph | node | edge) attr_list
attr\_list : '[' [ a_list ] ']' [ attr_list ]
a_list : ID '=' ID [ (';' | ',') ] [ a_list ]
edge_stmt : (node_id | subgraph) edgeRHS [ attr_list ]
edgeRHS :       edgeop (node\_id | subgraph) [ edgeRHS ]
node_stmt : node_id [ attr_list ]
node_id : ID [ port ]
port : ':' ID [ ':' compass_pt ]
     | ':' compass_pt
subgraph : [ subgraph [ ID ] ] '{' stmt_list '}'
compass_pt : (n | ne | e | se | s | sw | w | nw | c | _)
```

The keywords node, edge, graph, digraph, subgraph, and strict are case-independent. Note also that the allowed compass point values are not keywords, so these strings can be used elsewhere as ordinary identifiers and, conversely, the parser will actually accept any identifier.

4 Abstract Syntax Tree (AST) generation

There are several methods. Here I will describe the one usually taught in school which is very visual. I believe it is the most used in practice. However, writing the algorithm is not such a good idea.

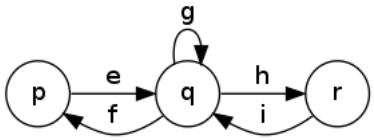
4.1 State removal method

This algorithm is about handling the graph of the automaton and is thus not very suitable for algorithms since it needs graph primitives such as ... state removal. I will describe it using higher-level primitives.

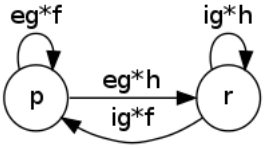
The key idea

The idea is to consider regular expressions on edges and then removing intermediate states while keeping the edges labels consistent.

The main pattern can be seen in the following to figures. The first has labels between **p**,**q**,**r** that are regular expressions **e**,**f**,**g**,**h**,**i** and we want to remove **q**.

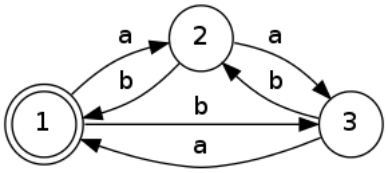


Once removed, we compose **e**,**f**,**g**,**h**,**i** together (while preserving the other edges between **p** and **r** but this is not displayed on this):

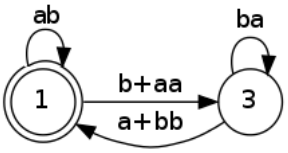


Example

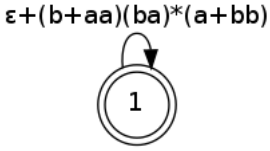
We start from:



we successively remove **q2**:



and then **q3**:



then we still have to apply a star on the expression from **q1** to **q1**. In this case, the final state is also initial so we really just need to add a star:

(æ+(b+aa)(ba)*(a+bb))*

Algorithm

L[i,j] is the regexp of the language from **qi** to **qj**. First, we remove all multi-edges:

```
for i = 1 to n:
  for j = 1 to n:
    if i == j then:
      L[i,j] := epsilon
    else:
      L[i,j] := empty
    for a in SUM:
      if trans(i, a, j):
        L[i,j] := L[i,j] + a
```

Now, the state removal. Suppose we want to remove the state **qk**:

```
remove(k):
  for i = 1 to n:
    for j = 1 to n:
      L[i,i] += L[i,k] . star(L[k,k]) . L[k,i]
      L[j,j] += L[j,k] . star(L[k,k]) . L[k,j]
      L[i,j] += L[i,k] . star(L[k,k]) . L[k,j]
      L[j,i] += L[j,k] . star(L[k,k]) . L[k,i]
```

Note that both with a pencil of paper and with an algorithm you should simplify expressions like **star(æ)=æ**, **e.æ=e**, **Ø+e=e**, **Ø.e=Ø** (By hand you just don't write the edge when it's not ØØ, or even ææfor a self-loop and you ignore when there is no transition between **qi** and **qk** or **qj** and **qk**)

Now, how to use **remove(k)**? You should not remove final or initial states lightly, otherwise you will miss parts of the language.

```
for i = 1 to n:
  if not(final(i)) and not(initial(i)):
    remove(i)
```

If you have only one final state qfqf and one initial state qsqs then the final expression is:

```
e := star(L[s,s]) . L[s,f] . star(L[f,s] . star(L[s,s]) . L[s,f] + L[
```

If you have several final states (or even initial states) then there is no simple way of merging these ones, other than applying the transitive closure method. Usually this is not a problem by hand but this is awkward when writing the algorithm. A much simpler workaround is to enumerate all pairs **(s,f)** and run the algorithm on the (already state-removed) graph to get all expressions **es,f** supposing ss is the only initial state and **f** is the only final state, then doing the union of all **es,f**.

This, and the fact that this is modifying languages more dynamically than the first method make it more error-prone when programming. I suggest using any other method.

Cons

There are a lot of cases in this algorithm, for example for choosing which node we should remove, the number of final states at the end, the fact that a final state can be initial, too etc.

Note that now that the algorithm is written, this is a lot like the transitive closure method. Only the context of the usage is different. I do not recommend implementing the algorithm, but using the method to do that by hand is a good idea.

References