Thompson's construction

In computer science, **Thompson's construction** algorithm, also called the McNaughton-Yamada-Thompson algorithm[1], is a method of transforming a regular expression into an equivalent nondeterministic finite automaton (NFA).[2] This NFA can be used to match strings against the regular expression.

Regular expressions and nondeterministic finite automata are two representations of formal languages. For instance, text processing utilities use regular expressions to describe advanced search patterns, but NFAs are better suited for execution on a computer. Hence, this algorithm is of practical interest, since it can compile regular expressions into NFAs. From a theoretical point of view, this algorithm is a part of the proof that they both accept exactly the same languages, that is, the regular languages.

An NFA can be made deterministic by the powerset construction and then be minimized to get an optimal automaton corresponding to the given regular expression. However, an NFA may also be interpreted directly.

To decide whether two given regular expressions describe the same language, each can be converted into an equivalent minimal deterministic finite automaton via Thompson's construction, powerset construction, and DFA minimization. If, and only if, the resulting automata agree up to renaming of states, the regular expressions' languages agreed.

Contents

The algorithm

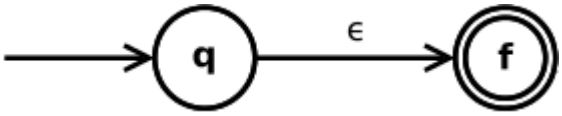
The algorithm works recursively by splitting an expression into its constituent subexpressions, from which the NFA will be constructed using a set of rules.[3] More precisely, from a regular expression E, the obtained automaton A with the transition function δ respects the following properties:

- A has exactly one initial state q_0, which is not accessible from any other state. That is, for any state q and any letter a, $\delta(q, a)$ does not contain q_0.
- A has exactly one final state q_f, which is not co-accessible from any other state. That is, for any letter a, $\delta(q_f, a) = \emptyset$.
- Let c be the number of concatenation of the regular expression E and let s be the number of symbols apart from parentheses — that is, $|$, *, a and ε. Then, the number of states of A is $2s - c$ (linear in the size of E).
- The number of transitions leaving any state is at most two.
- Since an NFA of m states and at most e transitions from each state can match a string of length n in time $O(emn)$, a Thompson NFA can do pattern matching in linear time, assuming a fixed-size alphabet.[4]

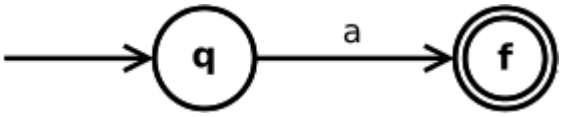
Rules

The following rules are depicted according to Aho et al. (2007),[1] p. 122. In what follows, $N(s)$ and $N(t)$ are the NFA of the subexpressions s and t, respectively.

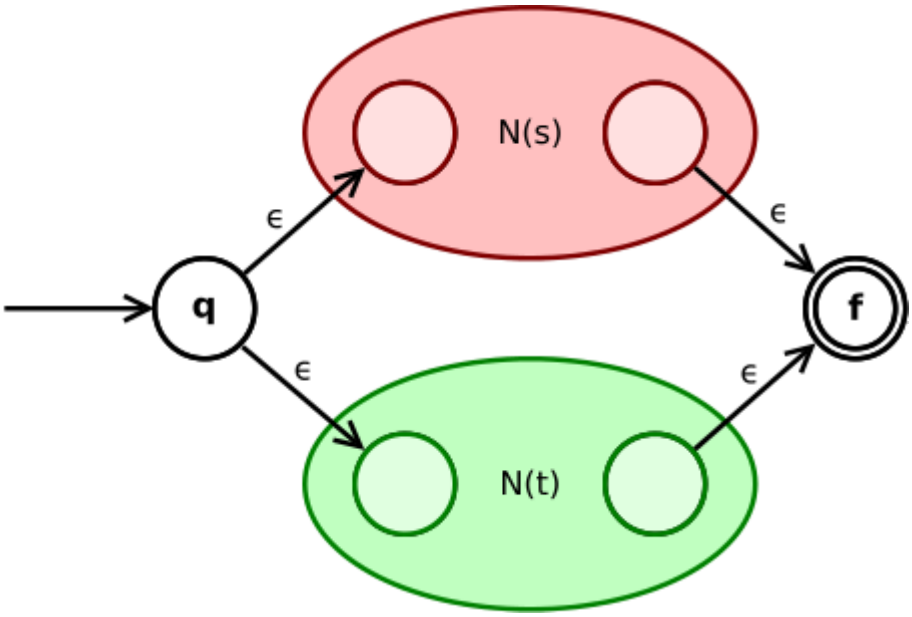
The **empty-expression** ε is converted to



A **symbol** *a* of the input alphabet is converted to

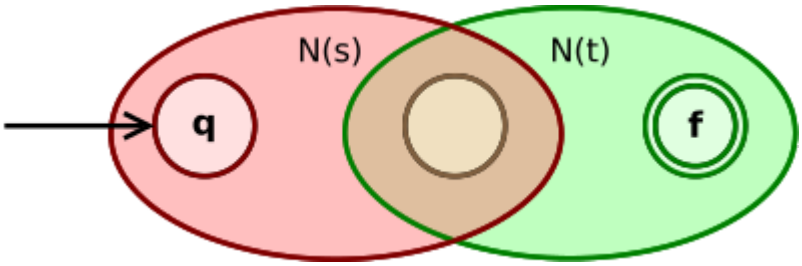


The **union expression** *s*|*t* is converted to



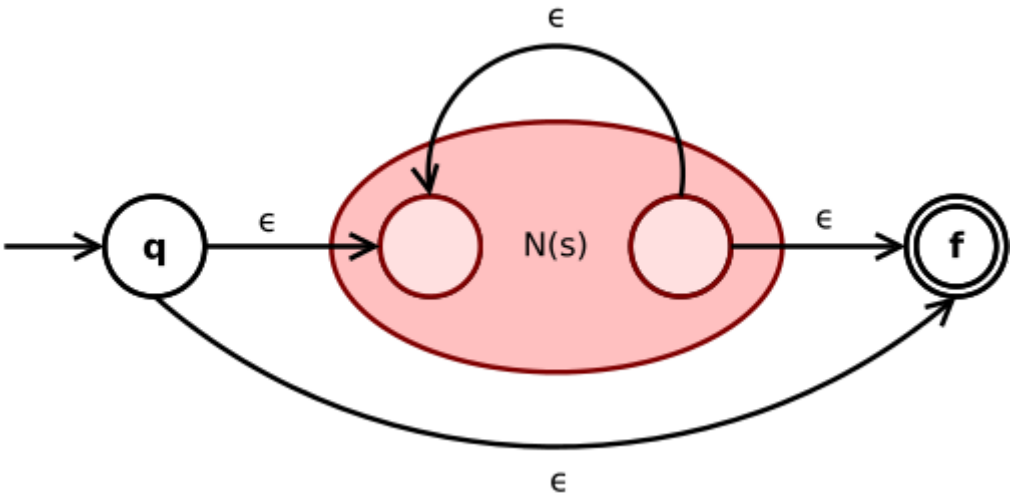
State *q* goes via ε either to the initial state of *N*(*s*) or *N*(*t*). Their final states become intermediate states of the whole NFA and merge via two ε-transitions into the final state of the NFA.

The **concatenation expression** *st* is converted to



The initial state of *N*(*s*) is the initial state of the whole NFA. The final state of *N*(*s*) becomes the initial state of *N*(*t*). The final state of *N*(*t*) is the final state of the whole NFA.

The **Kleene star expression** s^* is converted to



An ε-transition connects initial and final state of the NFA with the sub-NFA $N(s)$ in between. Another ε-transition from the inner final to the inner initial state of $N(s)$ allows for repetition of expressions according to the star operator

- The **parenthesized expression** (s) is converted to $N(s)$ itself.

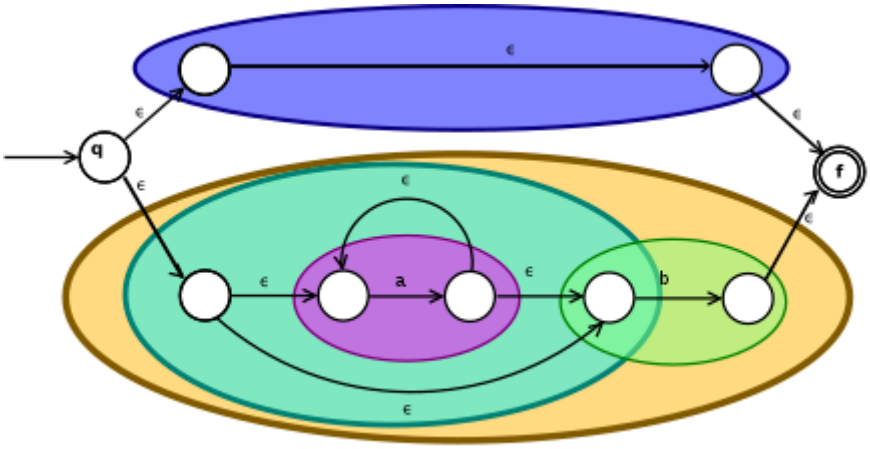
With these rules, using the **empty expression** and **symbol** rules as base cases, it is possible to prove with mathematical induction that any regular expression may be converted into an equivalent NFA.[1]

Example

Two examples are now given, a small informal one with the result, and a bigger with a step by step application of the algorithm.

Small Example

The picture below shows the result of Thompson's construction on `(ε|a*b)`. The pink oval corresponds to *a*, the teal oval corresponds to *a**, the green oval corresponds to *b*, the orange oval corresponds to *a*b*, and the blue oval corresponds to *ε*.



Example of `(ε|a*b)` using Thompson's construction, step by step

Application of the algorithm

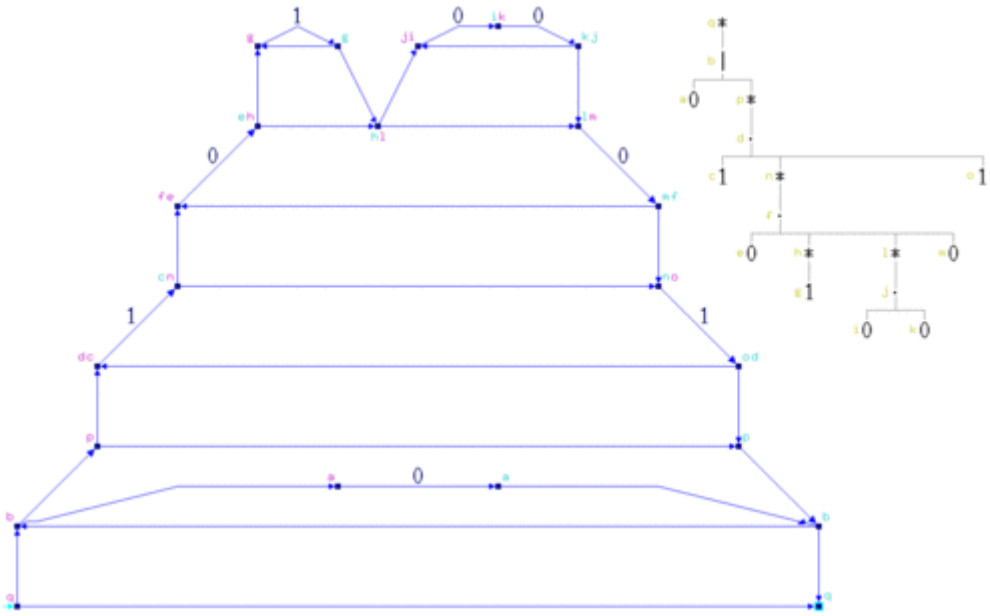
As an example, the picture shows the result of Thompson's construction algorithm on the regular expression `(0|(1(01*(00)*0)*1)*)*` that denotes the set of binary numbers that are multiples of 3:

{ ε, "0", "00", "11", "000", "011", "110", "0000", "0011", "0110", "1001", "1100", "1111", "00000", ... }.

The upper right part shows the logical structure (syntax tree) of the expression, with "." denoting concatenation (assumed to have variable arity); subexpressions are named a-q for reference purposes. The left part shows the nondeterministic finite automaton resulting from Thompson's algorithm, with the entry and exit state of each subexpression colored in magenta and cyan, respectively. An ε as transition label is omitted for clarity — unlabelled transitions are in fact ε transitions. The entry and exit state corresponding

to the root expression q is the start and accept state of the automaton, respectively

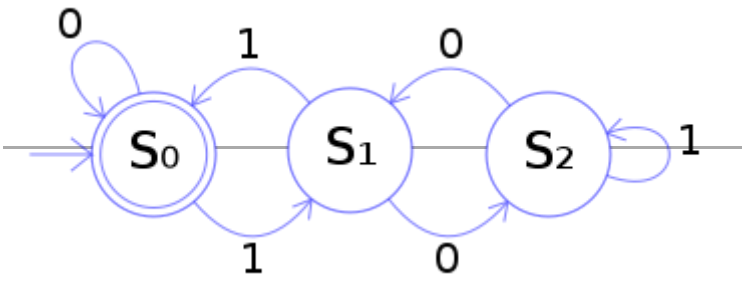
The algorithm's steps are as follows:



NFA obtained from regular expression(`0|(1(01*(00)*0)*1)*)*`

q: start converting Kleene star expression	`(0|(1(01*(00)*0)*1)*)*`
b: start converting union expression	`0|(1(01*(00)*0)*1)*`
a: convert symbol	`0`
p: start converting Kleene star expression	`(1(01*(00)*0)*1)*`
d: start converting concatenation expression	`1(01*(00)*0)*1`
c: convert symbol	`1`
n: start converting Kleene star expression	`(01*(00)*0)*`
f: start converting concatenation expression	`01*(00)*0`
e: convert symbol	`0`
h: start converting Kleene star expression	`1*`
g: convert symbol	`1`
h: finished converting Kleene star expression	`1*`
l: start converting Kleene star expression	`(00)*`
j: start converting concatenation expression	`00`
i: convert symbol	`0`
k: convert symbol	`0`
j: finished converting concatenation expression	`00`
l: finished converting Kleene star expression	`(00)*`
m: convert symbol	`0`
f: finished converting concatenation expression	`01*(00)*0`
n: finished converting Kleene star expression	`(01*(00)*0)*`
o: convert symbol	`1`
d: finished converting concatenation expression	`1(01*(00)*0)*1`
p: finished converting Kleene star expression	`(1(01*(00)*0)*1)*`
b: finished converting union expression	`0|(1(01*(00)*0)*1)*`
q: finished converting Kleene star expression	`(0|(1(01*(00)*0)*1)*)*`

An equivalent minimal deterministic automaton is shown below



Relation to other algorithms

Thompson's is one of several algorithms for constructing NFAs from regular expressions;[5] an earlier algorithm was given by McNaughton and Yamada.[6] Converse to Thompson's construction, Kleene's algorithm transforms a finite automaton into a regular expression.

Glushkov's construction algorithm is similar to Thompson's construction, once the ε-transitions are removed.

References

1. Alfred Vaino Aho; Monica S. Lam; Ravi Sethi; Jeffrey D. Ullman (2007). "3.7.4 Construction of an NFA from a Regular Expression". *Compilers : Principles, Techniques, & Tools* (https://www.pearson.com/us/higher-education/program/Aho-Compilers-Principles-Techniques-and-Tools-2nd-Edition/PGM167067.html) (print) (2nd ed.). Boston, MA, USA: Pearson Addison-Wesley. p. 159-163. ISBN 9780321486813.
2. Louden, Kenneth C. (1997). "2.4.1 From a Regular Expression to an NFA". *Compiler construction : Principles and Practice* (https://dl.acm.org/citation.cfm?id=523017) (print) (3rd ed.). 20 Park Plaza Boston, MA 02116-4324, US: PWS Publishing Company pp. 64–69. ISBN 0-534-93972-4.
3. Ken Thompson (Jun 1968). "Programming Techniques: Regular expression search algorithm" *Communications of the ACM*. **11** (6): 419–422. doi:10.1145/363347.363387 (https://doi.org/10.1145/363347.363387)
4. Xing, Guangming. "Minimized Thompson NFA" (http://people.wku.edu/guangming.xing/thompsonnfa.pdf) (PDF).
5. Watson, Bruce W. (1995). *A taxonomy of finite automata construction algorithms* (http://alexandria.tue.nl/extra1/wskrap/publichtml/9313452.pdf) (PDF) (Technical report). Eindhoven University of Technology. Computing Science Report 93/43.
6. R. McNaughton, H. Yamada (Mar 1960). "Regular Expressions and State Graphs for Automata" *IEEE Transactions on Electronic Computers* **9** (1): 39–47. doi:10.1109/TEC.1960.5221603 (https://doi.org/10.1109/TEC.1960.5221603)