# **Thompsons Construction Documentation**

## By James Mullarkey

## Shunting Yard Algorithm

(Reference Wikipedia <https://en.wikipedia.org/wiki/Shunting-yard_algorithm>)

In [computer science](https://en.wikipedia.org/wiki/Computer_science), the **shunting-yard algorithm** is a method for parsing mathematical expressions specified in [infix notation](https://en.wikipedia.org/wiki/Infix_notation). It can produce either a postfix notation string, also known as [Reverse Polish notation](https://en.wikipedia.org/wiki/Reverse_Polish_notation) (RPN), or an [abstract syntax tree](https://en.wikipedia.org/wiki/Abstract_syntax_tree) (AST). The [algorithm](https://en.wikipedia.org/wiki/Algorithm) was invented by [Edsger Dijkstra](https://en.wikipedia.org/wiki/Edsger_Dijkstra) and named the "shunting yard" algorithm because its operation resembles that of a [railroad shunting yard](https://en.wikipedia.org/wiki/Classification_yard).

Converting an infix regular expression to postfix using the Shunting Yard Algorithm

1. Input: 3 + 4
2. Push 3 to the output queue (whenever a number is read it is pushed to the output)
3. Push + (or its ID) onto the operator stack
4. Push 4 to the output queue
5. After reading the expression, pop the operators off the stack and add them to the output.

In this case there is only one, "+".

1. Output: 3 4 +

This already shows a couple of rules:

* All numbers are pushed to the output when they are read.
* At the end of reading the expression, pop all operators off the stack and onto the output.

## Shunting Example

(Reference The Oxford Math Centre <http://www.oxfordmathcenter.com/drupal7/node/628>)

A \* B + C becomes A B \* C +

The order in which the operators appear is not reversed. When the '+' is read, it has lower precedence than the '\*', so the '\*' must be printed first.

We will show this in a table with three columns. The first will show the symbol currently being read. The second will show what is on the stack and the third will show the current contents of the postfix string. The stack will be written from left to right with the 'bottom' of the stack to the left.

A \* B + C --> A B \* C +

Current Symbol Operator Stack Postfix String

1 A A

2 \* \* A

3 B \* A B

4 + + A B \* (pop and print \* before pushing +)

5 C + A B \* C

6 A B \* C +

The rule used in lines 1, 3 and 5 is to print an operand when it is read. The rule for line 2 is to push an operator onto the stack if it is empty. The rule for line 4 is if the operator on the top of the stack has higher precedence than the one being read, pop and print the one on top and then push the new operator on. The rule for line 6 is that when the end of the expression has been reached, pop the operators on the stack one at a time and print them.

## Regular Expressions

(References Wikipedia <https://en.wikipedia.org/wiki/Regular_expression>)

A regular expression, regex or regexp (sometimes called a rational expression) is a sequence of characters that define a *search pattern*. Usually this pattern is used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique that developed in theoretical computer science and formal language theory.

A regular expression, often called a pattern, is an expression used to specify a set of strings required for a particular purpose. A simple way to specify a finite set of strings is to list its elements or members.

A quantifier after a token (such as a character) or group specifies how often that a preceding element can occur. The most common quantifiers are the [question mark](https://en.wikipedia.org/wiki/Question_mark) ?, the asterisk \* (derived from the Kleene star), and the plus sign + (Kleene plus).

|  |  |
| --- | --- |
| **?** | The question mark indicates *zero or one* occurrences of the preceding element. For example, colou?r matches both "color" and "colour". |
| **\*** | The asterisk indicates *zero or more* occurrences of the preceding element. For example, ab\*c matches "ac", "abc", "abbc", "abbbc", and so on. |
| **+** | The plus sign indicates *one or more* occurrences of the preceding element. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac". |
| **{n}** | The preceding item is matched exactly *n* times. |
| **{min,}** | The preceding item is matched *min* or more times. |
| **{min,max}** | The preceding item is matched at least *min* times, but not more than *max* times. |

|  |  |
| --- | --- |
| **Metacharacter** | **Description** |
| **?** | Matches the preceding element zero or one time. For example, ab?c matches only "ac" or "abc". |
| **+** | Matches the preceding element one or more times. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac". |
| **|** | The choice (also known as alternation or set union) operator matches either the expression before or the expression after the operator. For example, abc|defmatches "abc" or "def". |

**Examples:**

* [hc]?at matches "at", "hat", and "cat".
* [hc]\*at matches "at", "hat", "cat", "hhat", "chat", "hcat", "cchchat", and so on.
* [hc]+at matches "hat", "cat", "hhat", "chat", "hcat", "cchchat", and so on, but not "at".
* cat|dog matches "cat" or "dog".

## Nondeterministic Finite Automaton

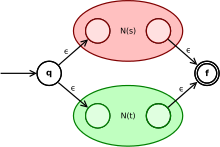
(References Wikipedia <https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton>)

In automata theory, a finite state machine is called a deterministic finite automaton (DFA), if

* each of its transitions is *uniquely* determined by its source state and input symbol, and
* reading an input symbol is required for each state transition.

A **nondeterministic finite automaton** (**NFA**), or nondeterministic finite state machine, does not need to obey these restrictions. In particular, every DFA is also an NFA. Sometimes the term **NFA** is used in a narrower sense, referring to a NFA that is *not* a DFA, but not in this article.

## **Closure properties**

[](https://en.wikipedia.org/wiki/File:Thompson-or.svg)

Composed NFA accepting the union of the languages of some given NFAs *N*(*s*) and *N*(*t*). For an input word *w* in the language union, the composed automaton follows an ε-transition from *q* to the start state (left colored circle) of an appropriate subautomaton — *N*(*s*) or *N*(*t*) — which, by following *w*, may reach an accepting state (right colored circle); from there, state *f* can be reached by another ε-transition. Due to the ε-transitions, the composed NFA is properly nondeterministic even if both *N*(*s*) and *N*(*t*) were DFAs; vice versa, constructing a DFA for the union language (even of two DFAs) is much more complicated.

NFAs are said to be closed under a (binary/unary) operator if NFAs recognize the languages that are obtained by applying the operation on the NFA recognizable languages. The NFAs are closed under the following operations.

* Union (cf. picture)
* Intersection
* Concatenation
* Negation
* Kleene closure

Since NFAs are equivalent to nondeterministic finite automaton with ε-moves (NFA-ε), the above closures are proved using closure properties of NFA-ε. The above closure properties imply that NFAs only recognize regular languages.

NFAs can be constructed from any regular expression using Thompson's construction algorithm.

## Kleene Star and Plus

(References Wikipedia <https://en.wikipedia.org/wiki/Kleene_star>)

In mathematical logic and computer science, the **Kleene star** (or **Kleene operator** or **Kleene closure**) is a unary operation, either on sets of strings or on sets of symbols or characters. In mathematics it is more commonly known as the free monoid construction. The application of the Kleene star to a set *V* is written as *V*\*. It is widely used for regular expressions, which is the context in which it was introduced by Stephen Kleene to characterize certain automata, where it means "zero or more".

1. If *V* is a set of strings, then *V*\* is defined as the smallest superset of *V* that contains the empty string ε and is closed under the string concatenation operation.
2. If *V* is a set of symbols or characters, then *V*\* is the set of all strings over symbols in *V*, including the empty string ε.

The set *V*\* can also be described as the set of finite-length strings that can be generated by concatenating arbitrary elements of *V*, allowing the use of the same element multiple times. If *V* is either the empty set ∅ or the singleton set {ε}, then *V*\* = {ε}; if *V* is any other finite set, then *V*\* is a countably infinite set.[[1]](https://en.wikipedia.org/wiki/Kleene_star" \l "cite_note-1)

## Kleene Plus

In some formal language studies, (e.g. AFL theory) a variation on the Kleene star operation called the *Kleene plus* is used. The Kleene plus omits the *V*0 term in the above union. In other words, the Kleene plus on *V* is:

V+ = U VI = V1 U V2 U V3 U….

For every set *L*, the Kleene plus of *L* (denoting *L*+) equals the concatenation of *L* with *L*\*; this holds because every element of *L*+ must either be composed from one element of *L* and finitely many non-empty terms in *L* or is just an element of *L* (where *L* itself is retrieved by taking *L* concatenated with ε). Conversely, *L*\* = {ε} ∪ *L*+.

(Reference <https://chortle.ccsu.edu/FiniteAutomata/Section07/sect07_16.html>)

|  |  |
| --- | --- |
| **Regular Expression** | **Matches** |
| **a\*** | ZERO or more 'a' |
| **ba\*** | b, ba, baa, baaa, baaaa, ... |
| **[ab]\*** | Ø, a, ab, aaa, ababb, bbb, ... zero or more characters, each character an 'a' or 'b' |
| **[^0-9]\*** | Ø, A, ABC, zw$nn, ... zero or more characters, no character a digit |
| **a\*b\*** | Ø, a, aaa, aaab, abbb, b, bbb, ... zero or more 'a', followed by zero or more 'b' |

## Operators ‘+’, ‘?’ and ‘|’

(References <http://web.mit.edu/gnu/doc/html/regex_3.html>)

### The Match-one-or-more Operator (+ or \+)

If the syntax bit RE\_LIMITED\_OPS is set, then Regex doesn't recognize this operator. Otherwise, if the syntax bit RE\_BK\_PLUS\_QM isn't set, then `+' represents this operator; if it is, then `\+' does.

This operator is similar to the match-zero-or-more operator except that it repeats the preceding regular expression at least once; see section The Match-zero-or-more Operator (\*), for what it operates on, how some syntax bits affect it, and how Regex backtracks to match it.

For example, supposing that `+' represents the match-one-or-more operator; then `ca+r' matches, e.g., `car' and `caaaar', but not `cr'.

### The Match-zero-or-one Operator (? or \?)

If the syntax bit RE\_LIMITED\_OPS is set, then Regex doesn't recognize this operator. Otherwise, if the syntax bit RE\_BK\_PLUS\_QM isn't set, then `?' represents this operator; if it is, then `\?' does.

This operator is similar to the match-zero-or-more operator except that it repeats the preceding regular expression once or not at all; see section The Match-zero-or-more Operator (\*), to see what it operates on, how some syntax bits affect it, and how Regex backtracks to match it.

For example, supposing that `?' represents the match-zero-or-one operator; then `ca?r' matches both `car' and `cr', but nothing else.

## The Alternation Operator (| or \|)

If the syntax bit RE\_LIMITED\_OPS is set, then Regex doesn't recognize this operator. Otherwise, if the syntax bit RE\_NO\_BK\_VBAR is set, then `|' represents this operator; otherwise, `\|' does.

Alternatives match one of a choice of regular expressions: if you put the character(s) representing the alternation operator between any two regular expressions a and b, the result matches the union of the strings that a and b match. For example, supposing that `|' is the alternation operator, then `foo|bar|quux' would match any of `foo', `bar' or `quux'.

The alternation operator operates on the largest possible surrounding regular expressions. (Put another way, it has the lowest precedence of any regular expression operator.) Thus, the only way you can delimit its arguments is to use grouping. For example, if `(' and `)' are the open and close-group operators, then `fo(o|b)ar' would match either `fooar' or `fobar'. (`foo|bar' would match `foo' or `bar'.)

The matcher usually tries all combinations of alternatives so as to match the longest possible string. For example, when matching `(fooq|foo)\*(qbarquux|bar)' against `fooqbarquux', it cannot take, say, the first ("depth-first") combination it could match, since then it would be content to match just `fooqbar'.

## Infix and Postfix Regular Expressions

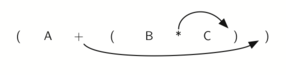
(References <http://interactivepython.org/runestone/static/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html>)

| **Infix Expression** |  | **Postfix Expression** |
| --- | --- | --- |
| A + B |  | A B + |
| A + B \* C |  | A B C \* + |
| A + B \* C + D |  | A B C \* + D + |
| (A + B) \* (C + D) |  | A B + C D + \* |
| A \* B + C \* D |  | A B \* C D \* + |
| A + B + C + D |  | A B + C + D + |

## Converting Infix expression to Postfix

The first technique that we will consider uses the notion of a fully parenthesized expression that was discussed earlier. Recall that A + B \* C can be written as (A + (B \* C)) to show explicitly that the multiplication has precedence over the addition. On closer observation, however, you can see that each parenthesis pair also denotes the beginning and the end of an operand pair with the corresponding operator in the middle.

Look at the right parenthesis in the subexpression (B \* C) above. If we were to move the multiplication symbol to that position and remove the matching left parenthesis, giving us B C \*, we would in effect have converted the subexpression to postfix notation. If the addition operator were also moved to its corresponding right parenthesis position and the matching left parenthesis were removed, the complete postfix expression would result



## Thompsons Construction

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

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.