Compiler: Note #1

ant-hengxin

0130

Big Picture:

we are used to expressing lexical rules in regular expressions, but we are writing the lexical analyzer through a `DFA`. That means that there exists some way to convert from regular expressions to `DFA`.

$$\texttt{re} \to \texttt{DFA}$$

But this approach is overly complicated, and for simplicity's sake, let's do this thing in multiple steps:

$$\texttt{re} \xrightarrow[\text{Construction}]{\text{Thompson}} \texttt{NFA} \xrightarrow[\text{Construction}]{\text{Subset}} \texttt{DFA}$$

And we also want to know how to convert a `DFA` to a `re`.

Alphabet Σ: A set of finite symbols.

String `s` over alphabet Σ: A sequence of symbols from Σ. A special string ϵ is the empty string with the property $|\epsilon| = 0$.

String operation: concatenation: `x = dog, y = house` $\Rightarrow$ `xy = doghouse`.

Language L over alphabet Σ: A countable set of strings over Σ.

Language operation:

Suppose `L, M` are languages, we can use set operations to construct new languages:

1. `L` $\cup$ `M:= { s | s` $\in$ `L or s` $\in$ `M }`
2. `LM:= { s` $\in$ `L and s` $\in$ `M }`
3. $\texttt{L}^*\texttt{:=} \bigcup_{i=0}^{\infty} \texttt{L}^i$ `(Kleene closure)`
4. $\texttt{L}^+\texttt{:=} \bigcup_{i=1}^{\infty} \texttt{L}^i$ `(Positive closure)`

The following logic is, we firstly talk about the definition of regular expression, then we talk about the definition of non-deterministic finite automaton (NFA), and we define what's the meaning of a regular expression is equivalent to a NFA.

Regular expression:

A regular expression over alphabet Σ is defined as follows:

1. ϵ is a regular expression.
2. $\forall$ `a` $\in \Sigma$, `a` is a regular expression.
3. If `r` is a regular expression, then (`r`) is also a regular expression.
4. If `r, s` are regular expressions, then `r|s, rs, r*` are also regular expressions.

Regular expressions are prioritized as follows: `()` > `*` > `concat` > `|`.
Regular expressions define a language we call the regular language. The rules are as follows:

1. L(ϵ) = { ϵ }.

2. `L(a)` $= \{$ `a` $\}$, $\forall$ `a` $\in \Sigma$.
3. `L((r))` $=$ `L(r)`.
4. `L(r|s)` $=$ `L(r)` $\cup$ `L(s)`, `L(rs) = L(r)L(s)`, `L(r*) = (L(r))*`.

Note: If `r1, r2` is a regular expression, then:

1. `r1?` represents `0 or 1 r1`.
2. `r1/r2` represents Indicates that `r1` when followed by `r2`.

NFA: non-deterministc-finite automaton:

A NFA is a five-tuple represented as $\mathscr{A} = (\Sigma, \mathrm{S}, \mathrm{s}_0, \delta, \mathrm{F})$ with the following properties:

1. Alphabet $\Sigma(\epsilon \notin \Sigma)$.
2. S is a finite set of states.
3. $\mathrm{s}_0 \in \mathrm{S}$ is the only start state.
4. $\delta : \mathrm{S} \times (\Sigma \cup \{\epsilon\}) \to 2^{\mathrm{S}}$
5. F is a set of accept states.

Note that the requirement **only** in property 3 is not necessary, because if there exists more then one start state, we can define a new start state and the original start states is constructed as a ϵ-closure of the new start state.

And the definition of NFA actually define a language `L`($\mathscr{A}$), which is composed of all the strings that can be accepted by the NFA.

Two important questions of NFA:

1. Given a string `s`, is `s` in `L`($\mathscr{A}$)?
2. What is `L` ($\mathscr{A}$)?

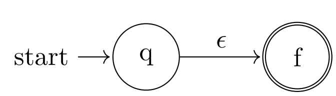
Equivalence of regular expression and NFA:

Now we can talk about what is the meaning of a regular expression is equivalent to a NFA. We call a regular expression `r` is equivalent to a NFA $\mathscr{A}$ if and only if `L(r)` is equivalent to `L` ($\mathscr{A}$) as two sets.

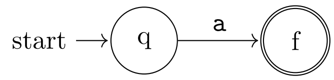
Let's start to talk about how to convert a `re` to a `NFA`, which let us introduce an algorithm called **Thompson's construction**.

So the idea of Thompson's construction is very simple, in order to know how to convert a regular expression to a NFA, what we only need to do is to know the conversion rules corresponding to the definition of regular expression.

1. if ϵ is a regular expression, then the corresponding NFA is:



2. if `a` is a regular expression, then the corresponding NFA is:



3. The NFA of `(s)` is the same as the NFA of `s`.

4. The NFA of `r|s` (Given `NFA(s)` and `NFA(r)`) is: