Regular Expressions (Motivation)

Regular expressions are everywhere. They are the go-to tool for anyone with an obligation to search or parse text. However, as one of my advisors points out, there are some problems for which regular expressions are not a good fit. Or, in his words, *usually solving a problem with regular expressions means now you have two problems.* In particular, we are now equipped to understand a key limitation of regular expressions: they may only describe regular languages! They are an *equivalent form of computation* to DFAs and NFAs, and thus have limitations that we will discuss next week when we discuss *irregularity.*

In gist, regular expressions...

1. offer a remarkably succinct way to express regular languages;
2. should help tie together the connection between languages and computing;
3. are an *inductive* definition.

As regular expression are equivalent to DFAs/NFAs, that means we may prove statements over *regular computation* inductively. This is a very powerful technique that many computer scientists do not choose to wield.

Examples

Regular expressions, in a way, are a syntactic representation of the semantic model of regular operators and their closures. In other words, a regular expression is *backed up* by the proofs that the operators used in the regular expression (namely union ($\cup$), concatenation ($\circ$), and star (*)) are regular. Let $\Sigma = \{0, 1\}$ and let's think on some examples.

1. $(1 \cup 0)$. Matches the language $\{0, 1\}$.
2. $(1 \cup \epsilon)$. Matches the language $\{1, \epsilon\}$.
3. $(01)^*$ Matches the language $\{\epsilon, 01, 0101, 010101, ...\}$.
4. $(0 \cup 1)^*$. Matches all strings.
5. $0\Sigma^*1$ Matches the language $\{0x1 \mid x \in \Sigma^*\}$.
6. $\emptyset$ is the *empty language*, which contains not even the empty string.
7. $\emptyset^*$ is the star-closure of the empty language, which is necessarily just $\{\epsilon\}$.

Formal Definition

This will not be the first *inductive definition* you've seen but it is likely the first one anyone has every *told you* is inductive.

Definition 1 (Regular Expressions). *Let Σ be some alphabet. We say that R is a* regular expression *if R is one of the cases below:*

- *a for some $a \in \Sigma$;*
- *ϵ, the empty string;*
- *$\emptyset$, the empty language;*
- *$R_1 \cup R_2$ for regular expressions R_1 and R_2;*
- *$R_1 R_2$ for regular expressions R_1 and R_2;*
- *R_1^* where R_1 is a regular expression.*

Another way of writing this definition is in Backus–Naur Form, or BNF, which looks like this

$$\begin{array}{lll} \text{symbols} & a \in \Sigma & \\ \text{Expressions } R & & ::= a \mid \epsilon \mid \emptyset \mid R \cup R \mid RR \mid R^* \end{array}$$

Note that in both cases, the definition is *recursive*. We are in the middle of defining regular expressions when we presume the existence of *smaller* subexpressions. This is the *inductive* idea that helps us be so succinct. I urge you to compare both definitions and see how the latter is shorthand for the first. Also, refer back to the examples from before: how does each match these definitions?

Another Inductive Definition

In your first lab, you were asked to consider the star-closure over alphabets, or Σ^*.

Recall the definition from lab 1A below.

$$\Sigma^* = \{ x_1 x_2 \cdots x_k \mid k \geq 0, x_i \in \Sigma \ \ \forall i \in [1, k] \}.$$

Let's consider an alternative *inductive* definition.

Definition 2 (Strings (inductive)). *Let Σ be an alphabet. Then Σ^* is the least set such that*

1. *$\epsilon \in \Sigma$, and*
2. *if $xs \in \Sigma^*$ and $x \in \Sigma$ then $x \circ xs \in \Sigma^*$.*

Or, in BNF form:

$$\begin{array}{lll} \text{symbols } a \in \Sigma & & \\ \text{Strings } \ \Sigma^* & & ::= \epsilon \mid a \circ \Sigma^* \end{array}$$

I believe the first lab asked you if ϵ is in Σ^* when Σ is empty. The answer is: yes, because $\epsilon \in \Sigma^*$ no matter the Σ. This much is clearer with an inductive definition. Further, proofs over strings can now be done by case analysis: given string w, it has either the form ϵ or the form $x \circ xs$ for some symbol x and string xs. In the latter case, you are given an inductive hypothesis: whatever you are trying to prove, you may assume that it is already true for the smaller string xs.

I will force more of this nonsense on you later on, if I can. For now I just want you to have seen it.

Equivalence with Finite Automata—building NFAs from regular expressions

To prove that regular expressions are another equivalent form of regular computation, we are obliged to (i) show how to build DFAs/NFAs from expressions and (ii) vice versa. I will give in this class only (i), which is easier. The direction (ii) is more involved and requires the development of an intermediate form of computation—GNFAs, or *generalized nondeterministic finite automaton*—which I would prefer to skip. See the rest of Sipser §1.3.

We will show direction (i) by considering each form a regular expression can take. In other words, we are going to split on the cases $a \mid \epsilon \mid \emptyset \mid R \cup R \mid RR \mid R^*$, where we are given inductive hypotheses for each of the recursive cases. For example, we may presume in the case that the regular expression has the form $R_1 \cup R_2$ that both R_1 and R_2 have equivalent NFAs. (I stress this again: this is what I mean by *inductive definitions have inductive proofs*.)

Claim 1. *If a language is desribed by a regular expression, then it is regular.*

Proof. Let A be a language described by the regular expression R. Proceed by case analysis on R.

Case ($R = a$). Suppose $R = a$ for some letter $a \in \Sigma$. Then $L(R) = \{a\}$ and the following NFA recognizes L(R). *(Too lazy for Tikz; see pg. 67 for sipser for drawings.)*

Case ($R = \epsilon$). Suppose $R = \epsilon$. Then $L(R) = \epsilon$, which is matched by a trivial NFA in which $Q = \{q\}$ and q is both the start and accept state. Imagine I put in a drawing below.

Case ($R = \emptyset$). Suppose $R = \emptyset$. Then $L(R) = \emptyset$ and any NFA in which $F = \emptyset$ will technically recognize R. The simplest one would be a start node with no out-arrows.

Case ($R = R_1 \cup R_2$). Invoke the IH on R_1 and R_2 such that N_1 and N_2 recognize the languages of R_1 and R_2, respectively. Then we can prefix each of these NFAs with a single start state q in which q has ϵ-arrows leading to the start of both N_1 and N_2. (Recall Sipser §1.2 and lecture notes 2B.)

Case ($R = R_1 \circ R_2$). I will give a proof of this in lecture 3B.

Case ($R = R_1^*$). I will give a proof of this in lecture 3B. □