Notes on Formal Languages

Foundations of Computer Science

January 28, 2016

1 Set Theory Review and Notation

I expect all of this section is just a refresher.

A set is a collection of elements, which may be sets themselves.

The empty set is denoted $\varnothing$ (or sometimes $\{\}$).

The two core relations on sets are $a \in A$ (a is an element of set A) and $A \subseteq B$ (A is a subset of B, meaning that every element of A is an element of B).

Some properties of $\subseteq$:

> $\varnothing \subseteq A$ for every A
>
> $A \subseteq A$ for every A
>
> If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$
>
> $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$, that is, if A and B have exactly the same elements.

If P is a property, then $\{x \mid P(x)\}$ is the set of all elements satisfying the property.

Common operations on sets:

> $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
>
> $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
>
> $\overline{A} = \{x \in \mathcal{U} \mid x \notin A\}$ (where $\mathcal{U}$ is a universe of elements such that $A \subseteq \mathcal{U}$ — the definition of $\overline{\ }$ therefore depends on the universe under consideration)

A set A is *finite* if it has a finite number of elements, that is, if there is a natural number $n \in \mathbb{N}$ such that A has n elements. If no such n exists, then A is *infinite*.

A function $f : A \longrightarrow B$ associates (or maps) every element of A to an element of B. Set A is the domain of the function, and B is the codomain. The image of A under f is the subset of B defined by $\{b \in B \mid f(a) = b \text{ for some } a \in A\}$.

If $f : A \longrightarrow B$ and $g : B \longrightarrow C$, then the *composition* $g \circ f : A \longrightarrow C$ defined by $(g \circ f)(x) = g(f(x))$.

A function $f : A \longrightarrow B$ is *one-to-one* (or injective) if it maps distinct elements of A into distinct elements of B (that is, if $f(a) = f(b)$ impllies $a = b$).

A function $f : A \longrightarrow B$ is *onto* (or surjective) if every element of B is in the image of A under f (that is, if for every elemebt $b \in B$ there is an element $a \in A$ with $f(a) = b$).

A function $f : A \longrightarrow B$ is a *correspondance* (or bijective) if it is both one-to-one and onto.

2 Decision Problems

Intuitively, a computation is a way to "implement" a function $f : A \longrightarrow B$. A big part of the course will be figuring out exactly what that intuition means. But looking at arbitrary functions between arbitrary sets A and B is way too general. It doesn't really give us a place to start.

Historically, researchers have looked at two large classes of functions to study computation:

1. **Natural number functions** of the form $f : (\mathbb{N} \times \cdots \times \mathbb{N}) \longrightarrow (\mathbb{N} \times \cdots \times \mathbb{N})$
2. **Decision problems** of the form $f : A \longrightarrow \{Y, N\}$ (or, really, any two-element set).

We will study decision problems. The domain of decision problems is usually constrained a bit further to be sets of *strings*.

Let Σ be a non-empty finite set we will call the *alphabet*. A *string over* Σ is a finite sequence of elements of Σ, usually written $a_1 \cdots a_k$, where $a_i \in \Sigma$.

The empty string is written ϵ.

The set of all strings over Σ is denoted Σ^*. Note that this is an infinite set.

If $\sigma_1 = a_1 \ldots a_k$ and $\sigma_2 = b_1 \cdots b_m$ are strings over Σ, then the *concatenation* of $\sigma_1\sigma_2$ is the string $a_1 \cdots a_k b_1 \cdots b_m$. Note that $\epsilon\sigma = \sigma\epsilon = \sigma$ for every string σ.

We define $\sigma^0 = \epsilon$, $\sigma^1 = \sigma$, $\sigma^2 = \sigma\sigma$, $\sigma^3 = \sigma\sigma\sigma$, etc.

Why do we look at decision problems? A decision problem $f : \Sigma^* \longrightarrow \{Y, N\}$ can be thought of as the *characteristic function* of a set A_f defined by $A_f = \{\sigma \in \Sigma^* \mid f(\sigma) = Y\}$. Conversely, every set $A \subseteq \Sigma^*$ defines a decision problem $f_A : \Sigma^* \longrightarrow \{Y, N\}$ by taking $f_A(\sigma) = Y$ exactly when $\sigma \in A$. Thus, decision problems with domain Σ^* and subsets of Σ^* are essentially interchangeable. And studying sets is a lot easier than studying functions.

This means we are going to work with sets of strings.

3 Languages

Fix an alphabet Σ. A set A of strings over Σ is usually called a *language* over Σ.

Since languages are sets, we inherit the usual set operations $A \cup B$, $A \cap B$, $\overline{A}$ (where the universe of A is taken to be Σ^*).

Because a language A is a set of strings, we can also define more specific operations.

$A \cdot B = \{\sigma_A \sigma_B \mid \sigma_A \in A \text{ and } \sigma_B \in B\}$, that is, the set of all strings obtained by concatenating a string of A and a string of B.

$A^0 = \{\epsilon\}$

$A^1 = A$

$A^2 = A \cdot A$

$A^3 = A \cdot A \cdot A$, etc

$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \cdots = \bigcup_{k \geq 0} A^k$

The $\cdot^*$ operation is called the *Kleene star*.

Some properties that are easy to verify:

$\varnothing \cdot A = A \cdot \varnothing = \varnothing$

$\{\epsilon\} \cdot A = A \cdot \{\epsilon\} = A$

The operations $\cup$, $\cdot$, and $\cdot^*$ are called the *regular operations*.

4 Regular Expressions and Regular Languages

A regular expression is a convenient notation for a certain class of languages.

A regular expression (over alphabet Σ) is defined by the following syntax:

$$
\begin{array}{rll}
r ::= & \texttt{1} & \\
 & \texttt{0} & \\
 & \texttt{a} & \text{for every } a \in \Sigma \\
 & r_1\texttt{+}r_2 & \\
 & r_1 r_2 & \\
 & r_1^* & \\
 & \texttt{(}r_1\texttt{)} &
\end{array}
$$

Intuitively, concatenation $r_1 r_2$ binds tighter than r_1`+`r_2, and $\cdot^*$ binds tighter than concatenation. For example, `ab+ac` is a regular expression, as is `a(b+c)` and `a`*`(b+c)`*.

A regular expression R denotes a language $L(R)$ over Σ in the following way:

$$\begin{aligned} L(\texttt{1}) &= \{\epsilon\} \\ L(\texttt{0}) &= \varnothing \\ L(\texttt{a}) &= \{a\} \\ L(r_1\texttt{+}r_2) &= L(r_1) \cup L(r_2) \\ L(r_1 r_2) &= L(r_1) \cdot L(r_2) \\ L(r_1^*) &= L(r_1)^* \\ L(\texttt{(}r_1\texttt{)}) &= L(r_1) \end{aligned}$$

For example:

$$\begin{aligned} L(\texttt{ab+ac}) &= L(\texttt{ab}) \cup L(\texttt{ac}) \\ &= (L(\texttt{a}) \cdot L(\texttt{b})) \cup (L(\texttt{a}) \cdot L(\texttt{c})) \\ &= (\{a\} \cdot \{b\}) \cup (\{a\} \cdot \{c\}) \\ &= \{ab\} \cup \{ac\} \\ &= \{ab, ac\} \end{aligned}$$

$$\begin{aligned} L(\texttt{a(b+c)}) &= L(\texttt{a}) \cdot L(\texttt{b+c}) \\ &= L(\texttt{a}) \cdot (L(\texttt{b}) \cup L(\texttt{c})) \\ &= \{a\} \cdot (\{b\} \cup \{c\}) \\ &= \{a\} \cdot (\{b, c\}) \\ &= \{ab, ac\} \end{aligned}$$

$$\begin{aligned} L(\texttt{a}^*\texttt{(b+c)}^*) &= L(\texttt{a}^*) \cdot L(\texttt{(b+c)}^*) \\ &= L(\texttt{a})^* \cdot L(\texttt{b+c})^* \\ &= \{a\}^* \cdot (L(\texttt{b}) \cup L(\texttt{c}))^* \\ &= \{a\}^* \cdot (\{b\} \cup \{c\})^* \\ &= \{a\}^* \cdot \{b, c\}^* \end{aligned}$$

And thinking about this last set (which is difficult to write down), it is basically the set of all strings obtained by concatenating a sequence of as (including none) to a sequence of bs and cs (in any order, including none). So *aaaaaa* is in this set, as is *aaaab*, *aaaabbbb*, *aaaabbbcbcbc*, etc.

A language A is called *regular* if there is a regular expression r such that $L(r) = A$.