Notes on Formal Languages

Foundations of Computer Science

January 26, 2017

Set Theory Review

I expect most of this section to be a refresher.

A set is a collection of elements. Those elements can be anything, including other sets.

Sets can be described by listing their elements, such as $\{a, b, c, \ldots\}$.

The empty set is denoted $\varnothing$, or $\{\,\}$.

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*.

The main relation on sets is *set membership*, written $a \in A$: a is an element of set A.

Two sets are equal if they have exactly the same elements.

Another relation on sets is *set inclusion*, written $A \subseteq B$: 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 $B \subseteq A$.

If P is a property, then $\{x \mid P(x)\}$ is the set of all elements satisfying the property. (Technically speaking, there are some restrictions on what makes up an acceptable property in this context — but they won't impact us.)

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 $^-$ therefore depends on the universe under consideration)

$A \times B = \{\langle x, y\rangle \mid x \in A, y \in B\}$, the set of all pairs of elements from A and B. This generalizes in the obvious way to products $A_1 \times A_2 \times \cdots \times A_k$.

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 $a \neq b$, then $f(a) \neq f(b)$).

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 element $b \in B$ there is an element $a \in A$ with $f(a) = b$).

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

Decision Problems

Intuitively, a computation is a way to "implement" a mathematical function $f : A \longrightarrow B$. A big part of the course will be understanding exactly what that intuition means.

Arbitrary functions between arbitrary sets A and B is too broad a class of functions to work with. Historically, researchers have looked at two 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

$$f : D \longrightarrow \{1, 0\}$$

(where 1 can be interpreted as true and 0 as false). Decisison problems represent predicates on the domain D. For example, determining if a graph is planar.

We will study decision problems. The domain D of decision problems is usually constrained 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 (possibly empty) finite sequence of elements of Σ, usually written $a_1 \cdots a_k$, where $a_i \in \Sigma$. For example, `aaccabc` is a string over alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. (It is also a string over alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.) We will use u, v, w to range over strings.

The length of $u = a_1 \ldots a_k$, written $|u|$, is k.

The empty string is written ϵ. It has length 0.

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

If $u = a_1 \ldots a_k$ and $v = b_1 \cdots b_m$ are strings over Σ, then the *concatenation* uv is the string $a_1 \cdots a_k b_1 \cdots b_m$. Note that $\epsilon u = u\epsilon = u$ for every string u.

We define $u^0 = \epsilon$, $u^1 = u$, $u^2 = uu$, $u^3 = uuu$, etc.

A decision problem

$$f : \Sigma^* \longrightarrow \{1, 0\}$$

can be thought of as the *characteristic function* of a set $S(f)$ defined by

$$S(f) = \{u \in \Sigma^* \mid f(u) = 1\}$$

Conversely, every set $A \subseteq \Sigma^*$ defines a decision problem $F(A) : \Sigma^* \longrightarrow \{1, 0\}$ by taking $F(A)(u) = 1$ exactly when $u \in A$. Note that $S(F(A)) = A$ and $F(S(f)) = f$.

This shows that decision problems with domain Σ^* and subsets of Σ^* are essentially interchangeable. And studying sets is a lot easier than studying functions, since sets can be manipulated with all the operations we saw in the last section.

We are going to study decision problems via sets of strings.

Languages

A *language* over alphabet Σ is a set of strings 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 specifically, we can also define more specific operations.

> $A \cdot B = \{uv \mid u \in A \text{ and } v \in B\}$, 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 * operation is called the *Kleene star*.

Some properties that are easy to verify:

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

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

Example: $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and $A = \{\mathtt{aa}, \mathtt{bb}\}$. Then $A^* = \{\epsilon, \mathtt{aaaa}, \mathtt{aabb}, \mathtt{bbaa}, \mathtt{bbbb}, \mathtt{aaaaaa}, \mathtt{aaaabb}, \mathtt{aabbaa}, \mathtt{aabbbb}, \mathtt{bbaaaa}, \mathtt{bbaabb}, \mathtt{bbbbaa}, \mathtt{bbbbbb}, \ldots\}$.

Also, we can now understand why we use the notation Σ^* for the set of all strings: if we consider Σ as a set of strings, each of length 1, then Σ^* according to the above definition indeed gives the set of all strings over alphabet Σ.

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

A language over alphabet $\Sigma = \{a_1, \ldots, a_k\}$ is *regular* if it can be obtained from the sets $\varnothing, \{\epsilon\}, \{a_1\}, \ldots, \{a_k\}$ and finitely many applications of the regular operations.

Example: consider the language of all even-length strings over alphabet $\{\mathtt{a}, \mathtt{b}\}$. It can be obtained as:

$$((\{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \{\mathtt{c}\}) \cdot (\{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \{\mathtt{c}\}))^*$$

Therefore, that language is regular.

Example: consider the language of all strings over $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ that start and end with an $\mathtt{a}$:

$$(\{\mathtt{a}\} \cdot (\{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \{\mathtt{c}\})^* \cdot \{\mathtt{a}\}) \cup \{\mathtt{a}\}$$

Therefore, that language is also regular.

Regular Expressions

Regular expressions are a convenient notation for regular languages.

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

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

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:

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

For example:

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

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

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

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 `a`s (including none) to a sequence of `b`s and `c`s (in any order, including none). So `aaaaaa` is in this set, as is `aaaab`, `aaaabbbb`, `aaaabbbcbcbc`, etc.

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

From this definition and from the definition of the language of a regular expression, it is easy to see that:

- Every finite language (i.e., language with a finite number of strings) is regular, including the empty language $\varnothing$. (Why?)
- if A and B are regular languages, then $A \cup B$ and $A \cdot B$ are regular languages. (For example, if A and B are regular, then there are regular expressions r_A and r_B such that $L(r_A) = A$ and $L(r_B) = B$, and it is immediate that $L(r_A + r_B) = A \cup B$, so $r_A + r_B$ is the regular expression showing that $A \cup B$ is regular.)
- If A is regular, then A^* is regular. (In particular, if we consider Σ as a language (one string per alphabet symbol) which is regular because it is finite, then Σ^*, the set of all strings over Σ, is regular.)

We can also use regular expressions to show that if A is regular, then $rev(A) = \{rev(u) \mid u \in A\}$, where $rev(u)$ is the reverse of string u, is regular: since A is regular, there is a regular expression r with $L(r) = A$. We take this regular expression and transform it into regular expression $\widehat{r}$ as follow:

$$\begin{aligned}
\widehat{\mathtt{1}} &= \mathtt{1} \\
\widehat{\mathtt{0}} &= \mathtt{0} \\
\widehat{a} &= a \\
\widehat{r_1 \mathtt{+} r_2} &= \widehat{r_2} \mathtt{+} \widehat{r_1} \\
\widehat{r_1 r_2} &= \widehat{r_2} \cdot \widehat{r_1} \\
\widehat{r_1^*} &= (\widehat{r_1})^* \\
\widehat{(r_1)} &= (\widehat{r_1})
\end{aligned}$$

Now, if $L(r) = A$, then $L(\widehat{r}) = rev(A)$, and therefore $rev(A)$ is regular.

It's also the case that when A is regular, then $\overline{A}$ is regular. We can't show that just yet.

Not every language is regular. The set of all *palindromes* over an alphabet Σ is not regular, for instance. Again, we can't show that just yet.