Caveat lector: This is the first edition of this lecture note. Please send bug reports and suggestions to jeffe@illinois.edu.

> *But the Lord came down to see the city and the tower the people were building. The Lord said, "If as one people speaking the same language they have begun to do this, then nothing they plan to do will be impossible for them. Come, let us go down and confuse their language so they will not understand each other."*
>
> — Genesis 11:6–7 (New International Version)

> *Imagine a piano keyboard, eh, 88 keys, only 88 and yet, and yet, hundreds of new melodies, new tunes, new harmonies are being composed upon hundreds of different keyboards every day in Dorset alone. Our language, tiger, our language: hundreds of thousands of available words, frillions of legitimate new ideas, so that I can say the following sentence and be utterly sure that nobody has ever said it before in the history of human communication: "**Hold the newsreader's nose squarely, waiter, or friendly milk will countermand my trousers.**" Perfectly ordinary words, but never before put in that precise order. A unique child delivered of a unique mother.*
>
> — Stephen Fry, *A Bit of Fry and Laurie*, Series 1, Episode 3 (1989)

2 Regular Languages

2.1 Definitions

A ***formal language*** (or just a ***language***) is a set of strings over some finite alphabet Σ, or equivalently, an arbitrary subset of Σ^*. For example, each of the following sets is a language:

- The empty set $\emptyset$.[1]
- The set $\{\varepsilon\}$.
- The set $\{0,1\}^*$.
- The set $\{\texttt{THE}, \texttt{OXFORD}, \texttt{ENGLISH}, \texttt{DICTIONARY}\}$.
- The set of all subsequences of $\texttt{THE}\diamond\texttt{OXFORD}\diamond\texttt{ENGLISH}\diamond\texttt{DICTIONARY}$.
- The set of all words in *The Oxford English Dictionary*.
- The set of all strings in $\{0,1\}^*$ with an odd number of 1s.
- The set of all strings in $\{0,1\}^*$ that represent a prime number in base 13.
- The set of all sequences of turns that solve the Rubik's cube (starting in some fixed configuration)
- The set of all python programs that print "Hello World!"

As a notational convention, I will always use italic upper-case letters (usually L, but also A, B, C, and so on) to represent languages.

Formal languages are not "languages" in the same sense that English, Klingon, and Python are "languages". Strings in a formal language do not necessarily carry any "meaning", nor are they necessarily assembled into larger units ("sentences" or "paragraphs" or "packages") according to some "grammar".

It is ***very*** important to distinguish between three "empty" objects. Many beginning students have trouble keeping these straight.

[1] The empty set symbol $\emptyset$ derives from the Norwegian letter Ø, pronounced like a sound of disgust or a German ö, and *not* from the Greek letter ϕ. Calling the empty set "fie" or "fee" makes the baby Jesus cry.

- $\emptyset$ is the empty *language*, which is a set containing zero strings. $\emptyset$ is not a string.
- $\{\varepsilon\}$ is a language containing exactly one string, which has length zero. $\{\varepsilon\}$ is not empty, and it is not a string.
- ε is the empty *string*, which is a sequence of length zero. ε is not a language.

2.2 Building Languages

Languages can be combined and manipulated just like any other sets. Thus, if A and B are languages over Σ, then their union $A \cup B$, intersection $A \cap B$, difference $A \setminus B$, and symmetric difference $A \oplus B$ are also languages over Σ, as is the complement $\overline{A} := \Sigma^* \setminus A$. However, there are two more useful operators that are specific to sets of *strings*.

The ***concatenation*** of two languages A and B, again denoted $A \bullet B$ or just AB, is the set of all strings obtained by concatenating an arbitrary string in A with an arbitrary string in B:

$$A \bullet B := \{xy \mid x \in A \text{ and } y \in B\}.$$

For example, if $A = \{\texttt{HOCUS}, \texttt{ABRACA}\}$ and $B = \{\texttt{POCUS}, \texttt{DABRA}\}$, then $A \bullet B = \{\texttt{HOCUSPOCUS}, \texttt{ABRACAPOCUS}, \texttt{HOCUSDABRA}, \texttt{ABRACADABRA}\}$. In particular, for every language A, we have

$$\emptyset \bullet A = A \bullet \emptyset = \emptyset \qquad \text{and} \qquad \{\varepsilon\} \bullet A = A \bullet \{\varepsilon\} = A.$$

The ***Kleene closure*** or ***Kleene star***[2] of a language L, denoted L^*, is the set of all strings obtained by concatenating a sequence of zero or more strings from L. For example, $\{\texttt{0}, \texttt{11}\}^* = \{\varepsilon, \texttt{0}, \texttt{00}, \texttt{11}, \texttt{000}, \texttt{011}, \texttt{110}, \texttt{0000}, \texttt{0011}, \texttt{0110}, \texttt{1100}, \texttt{1111}, \texttt{00000}, \texttt{00011}, \texttt{00110}, \ldots, \texttt{000111100110111111}, \ldots\}$. More formally, L^* is defined recursively as the set of all strings w such that either

- $w = \varepsilon$, or
- $w = xy$, for some strings $x \in L$ and $y \in L^*$.

This definition immediately implies that

$$\emptyset^* = \{\varepsilon\}^* = \{\varepsilon\}.$$

For any other language L, the Kleene closure L^* is infinite and contains arbitrarily long (but *finite!*) strings. Equivalently, L^* can also be defined as the smallest superset of L that contains the empty string ε and is closed under concatenation (hence "closure"). The set of all strings Σ^* is, just as the notation suggests, the Kleene closure of the alphabet Σ (where each symbol is viewed as a string of length 1).

A useful variant of the Kleene closure operator is the ***Kleene plus***, defined as $L^+ := L \bullet L^*$. Thus, L^+ is the set of all strings obtained by concatenating a sequence of ***one*** or more strings from L.

2.3 Regular Languages and Regular Expressions

A language L is ***regular*** if and only if it satisfies one of the following (recursive) conditions:

- L is empty;
- L contains a single string (which could be the empty string ε);
- L is the union of two regular languages;

[2] after Stephen Kleene, who pronounced his last name "***clay***-knee", not "clean" or "cleanie" or "claynuh" or "dimaggio".

- L is the concatenation of two regular languages; or
- L is the Kleene closure of a regular language.

Regular languages are normally described using a slightly more compact representation called ***regular expressions***, which omit braces around one-string sets, use $+$ to represent union instead of $\cup$, and juxtapose subexpressions to represent concatenation instead of using an explicit operator $\bullet$. By convention, in the absence of parentheses, the * operator has highest precedence, followed by the (implicit) concatenation operator, followed by $+$. Thus, for example, the regular expression $\mathtt{10}^*$ is shorthand for $\{\mathtt{1}\} \bullet \{\mathtt{0}\}^*$. As a larger example, the regular expression

$$\mathtt{0} + \mathtt{0}^*\mathtt{1}(\mathtt{10}^*\mathtt{1} + \mathtt{01}^*\mathtt{0})^*\mathtt{10}^*$$

represents the language

$$\{\mathtt{0}\} \cup \big(\{\mathtt{0}\}^* \bullet \{\mathtt{1}\} \bullet \big((\{\mathtt{1}\} \bullet \{\mathtt{0}\}^* \bullet \{\mathtt{1}\}) \cup (\{\mathtt{0}\} \bullet \{\mathtt{1}\}^* \bullet \{\mathtt{0}\})\big)^* \bullet \{\mathtt{1}\} \bullet \{\mathtt{0}\}^*\big).$$

Here are a few more examples of regular expressions and the languages they represent.

- $\mathtt{0}^*$ — the set of all strings of $\mathtt{0}$s, including the empty string.
- $\mathtt{00000}^*$ — the set of all strings consisting of at least four $\mathtt{0}$s.
- $(\mathtt{00000})^*$ — the set of all strings of $\mathtt{0}$s whose length is a multiple of 5.
- $(\varepsilon + \mathtt{1})(\mathtt{01})^*(\varepsilon + \mathtt{0})$ — the set of all strings of alternating $\mathtt{0}$s and $\mathtt{1}$s, or equivalently, the set of all binary strings that do not contain the substrings $\mathtt{00}$ or $\mathtt{11}$.
- $((\varepsilon + \mathtt{0} + \mathtt{00} + \mathtt{000})\mathtt{1})^*(\varepsilon + \mathtt{0} + \mathtt{00} + \mathtt{000})$ — the set of all binary strings that do not contain the substring $\mathtt{0000}$.
- $((\mathtt{0} + \mathtt{1})(\mathtt{0} + \mathtt{1}))^*$ — the set of all binary strings whose length is even.
- $\mathtt{1}^*(\mathtt{01}^*\mathtt{01}^*)^*$ — the set of all binary strings with an even number of $\mathtt{0}$s.
- $\mathtt{0} + \mathtt{1}(\mathtt{0} + \mathtt{1})^*\mathtt{00}$ — the set of all non-negative binary numerals divisible by 4 and with no redundant leading $\mathtt{0}$s.
- $\mathtt{0} + \mathtt{0}^*\mathtt{1}(\mathtt{10}^*\mathtt{1} + \mathtt{01}^*\mathtt{0})^*\mathtt{10}^*$ — the set of all non-negative binary numerals divisible by 3, possibly with redundant leading $\mathtt{0}$s.

The last example should ***not*** be obvious. It is straightforward, but rather tedious, to prove by induction that every string in $\mathtt{0} + \mathtt{0}^*\mathtt{1}(\mathtt{10}^*\mathtt{1} + \mathtt{01}^*\mathtt{0})^*\mathtt{10}^*$ is the binary representation of a non-negative multiple of 3. It is similarly straightforward, and similarly tedious, to prove that the binary representation of *every* non-negative multiple of 3 matches this regular expression. In a later note, we will see a systematic method for deriving regular expressions for some languages that avoids (or more accurately, automates) this tedium.

Most of the time we do not distinguish between regular expressions and the languages they represent, for the same reason that we do not normally distinguish between the arithmetic expression "2+2" and the integer 4, or the symbol π and the area of the unit circle. However, we sometimes need to refer to regular expressions themselves *as strings*. In those circumstances, we write $\boldsymbol{L(R)}$ to denote the language represented by the regular expression R. String w ***matches*** regular expression R if and only if $w \in L(R)$.

Two regular expressions R and R' are ***equivalent*** if they describe the same language; for example, the regular expressions $(0+1)^*$ and $(1+0)^*$ are equivalent, because the union operator is commutative.

Almost every regular language can be represented by infinitely many distinct but equivalent regular expressions, even if we ignore ultimately trivial equivalences like $L = (L\emptyset)^*L\varepsilon + \emptyset$. The following identities, which we state here without (easy) proofs, are useful for designing, simplifying, or understanding regular expressions.

Lemma 2.1. *The following identities hold for all languages A, B, and C:*
(a) $\emptyset A = A\emptyset = \emptyset$.
(b) $\varepsilon A = A\varepsilon = A$.
(c) $A + B = B + A$.
(d) $(A + B) + C = A + (B + C)$.
(e) $(AB)C = A(BC)$.
(f) $A(B + C) = AB + AC$.

Lemma 2.2. *The following identities hold for every language L:*
(a) $L^* = \varepsilon + L^+ = L^*L^* = (L + \varepsilon)^* = (L \setminus \varepsilon)^* = \varepsilon + L + L^+L^+$.
(b) $L^+ = L^* \setminus \varepsilon = LL^* = L^*L = L^+L^* = L^*L^+ = L + L^+L^+$.
(c) $L^+ = L^*$ *if and only if* $\varepsilon \in L$.

Lemma 2.3 (Arden's Rule). *For any languages A, B, and L such that $L = AL + B$, we have $A^*B \subseteq L$. Moreover, if A does not contain the empty string, then $L = AL + B$ if and only if $L = A^*B$.*

2.4 Things What Ain't Regular Expressions

Many computing environments and programming languages support patterns called ***regexen*** (singular ***regex***, pluralized like *ox*) that are considerably more general and powerful than regular expressions. Regexen include special symbols representing negation, character classes (for example, upper-case letters, or digits), contiguous ranges of characters, line and word boundaries, limited repetition (as opposed to the unlimited repetition allowed by *), back-references to earlier subexpressions, and even local variables. Despite its obvious etymology, a regex is ***not*** necessarily a regular expression, and it does ***not*** necessarily describe a regular language![3]

Another type of pattern that is often confused with regular expression are ***globs***, which are patterns used in most Unix shells and some scripting languages to represent sets file names. Globs include symbols for arbitrary single characters (`?`), single characters from a specified range (`[a-z]`), arbitrary substrings (`*`), and substrings from a specified finite set (`{foo,ba{r,z}}`). Globs are significantly *less* powerful than regular expressions.

2.5 Not Every Language is Regular

You may be tempted to conjecture that *all* languages are regular, but in fact, the following cardinality argument *almost all* languages are *not* regular. To make the argument concrete, let's consider languages over the single-symbol alphabet $\{\diamond\}$.

- Every regular expression over the one-symbol alphabet $\{\diamond\}$ is itself a string over the 7-symbol alphabet $\{\diamond, +, (,), *, \varepsilon, \emptyset\}$. By interpreting these symbols as the digits 1 through 7, we can interpret any string over this larger alphabet as the base-8 representation of some unique integer. Thus, the set of all regular expressions over $\{\diamond\}$ is *at most* as large as the set of integers, and is therefore countably infinite. It follows that the set of all regular *languages* over $\{\diamond\}$ is also countably infinite.

[3]However, regexen are not all-powerful, either; see http://stackoverflow.com/a/1732454/775369.

- On the other hand, for any real number $0 \le \alpha < 1$, we can define a corresponding language

$$L_\alpha = \left\{ \diamond^n \;\middle|\; \alpha 2^n \bmod 1 \ge 1/2 \right\}.$$

  In other words, L_α contains the string $\diamond^n$ if and only if the $(n+1)$th bit in the binary representation of α is equal to 1. For any distinct real numbers $\alpha \neq \beta$, the binary representations of α and β must differ in some bit, so $L_\alpha \neq L_\beta$. We conclude that the set of ***all*** languages over $\{\diamond\}$ is *at least* as large as the set of real numbers between 0 and 1, and is therefore uncountably infinite.

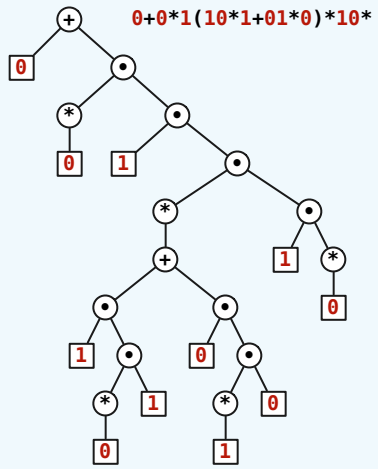
We will see several explicit examples of non-regular languages in future lectures. For example, the set of all regular expressions over $\{0, 1\}$ is not itself a regular language!

2.6 Parsing Regular Expressions

Most algorithms for regular expressions require them in the form of *regular expression trees*, rather than as raw strings. A regular expression tree is one of the following:

- A leaf node labeled $\emptyset$.
- A leaf node labeled with a string in Σ^*.
- A node labeled $+$ with two children, each the root of an expression tree.
- A node labeled $*$ with one child, which is the root of an expression tree.
- A node labeled $\bullet$ with two children, each the root of an expression tree.

In other words, a regular expression tree directly encodes a sequence of alternation, concatenation and Kleene closure operations that defines a regular language. Similarly, when we want to prove things about regular expressions or regular languages, it is more natural to think of subexpressions *as subtrees* rather than *as substrings*.



★★★

Given any regular expression of length n, we can ***parse*** it into an equivalent regular expression tree in $O(n)$ time. Thus, when we see an algorithmic problem that starts "Given a regular expression. . . ", we can assume without loss of generality that we're actually given a regular expression *tree*.

We'll see more on this topic later.

Exercises

1. (a) Prove that $\{\varepsilon\} \bullet L = L \bullet \{\varepsilon\} = L$, for any language L.
   (b) Prove that $\emptyset \bullet L = L \bullet \emptyset = \emptyset$, for any language L.
   (c) Prove that $(A \bullet B) \bullet C = A \bullet (B \bullet C)$, for all languages A, B, and C.

(d) Prove that $|A \bullet B| = |A| \cdot |B|$, for all languages A and B. (The second $\cdot$ is multiplication!)

(e) Prove that L^* is finite if and only if $L = \emptyset$ or $L = \{\varepsilon\}$.

(f) Prove that $AB = BC$ implies $A^*B = BC^* = A^*BC^*$, for all languages A, B, and C.

2. Recall that the reversal w^R of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

The reversal L^R of any language L is the set of reversals of all strings in L:

$$L^R := \{w^R \mid w \in L\}.$$

(a) Prove that $(AB)^R = B^R A^R$ for all languages A and B.

(b) Prove that $(L^R)^R = L$ for every language L.

(c) Prove that $(L^*)^R = (L^R)^*$ for every language L.

3. Prove that each of the following regular expressions is equivalent to $(\mathtt{0} + \mathtt{1})^*$.

(a) $\varepsilon + \mathtt{0}(\mathtt{0} + \mathtt{1})^* + \mathtt{1}(\mathtt{1} + \mathtt{0})^*$

(b) $\mathtt{0}^* + \mathtt{0}^*\mathtt{1}(\mathtt{0} + \mathtt{1})^*$

(c) $((\varepsilon + \mathtt{0})(\varepsilon + \mathtt{1}))^*$

(d) $\mathtt{0}^*(\mathtt{10}^*)^*$

(e) $(\mathtt{1}^*\mathtt{0})^*(\mathtt{0}^*\mathtt{1})^*$

4. For each of the following languages in $\{\mathtt{0}, \mathtt{1}\}^*$, describe an equivalent regular expression. There are infinitely many correct answers for each language. (This problem will become significantly simpler after we've seen finite-state machines, in the next lecture note.)

(a) Strings that end with the suffix $\mathtt{0}^9 = \mathtt{000000000}$.

(b) All strings except $\mathtt{010}$.

(c) Strings that contain the substring $\mathtt{010}$.

(d) Strings that contain the subsequence $\mathtt{010}$.

(e) Strings that do not contain the substring $\mathtt{010}$.

(f) Strings that do not contain the subsequence $\mathtt{010}$.

(g) Strings that contain an even number of occurrences of the substring $\mathtt{010}$.

*(h) Strings that contain an even number of occurrences of the substring $\mathtt{000}$.

(i) Strings in which every occurrence of the substring $\mathtt{00}$ appears before every occurrence of the substring $\mathtt{11}$.

(j) Strings w such that *in every prefix of* w, the number of $\mathtt{0}$s and the number of $\mathtt{1}$s differ by at most 1.

*(k) Strings w such that *in every prefix of w*, the number of `0`s and the number of `1`s differ by at most 2.

*(l) Strings in which the number of `0`s and the number of `1`s differ by a multiple of 3.

*(m) Strings that contain an even number of `1`s and an odd number of `0`s.

★(n) Strings that represent a number divisible by 5 in binary.

5. Prove that for any regular expression R such that $L(R)$ is nonempty, there is a regular expression equivalent to R that does not use the empty-set symbol $\emptyset$.

6. Prove that if L is a regular language, then L^R is also a regular language. *[Hint: How do you reverse a regular expression?]*

7. (a) Describe and analyze an efficient algorithm to determine, given a regular expression R, whether $L(R)$ is empty.

   (b) Describe and analyze an efficient algorithm to determine, given a regular expression R, whether $L(R)$ is infinite.

   In each problem, assume you are given R as a regular expression tree, not just a raw string.