From an informal or intuitive perspective what might we mean by computability? One natural interpretation is that something is computable if it can be calculated by a systematic procedure; we might think of this as a process that can be described by one person and carried out by another.

We will now assert that any such process can be written as a computer program. This is known as the Church-Turing thesis because they were the first to give formal mathematical specifications of what constituted an algorithm in form of the lambda calculus and Turing Machines, respectively. Note that the claim is a thesis; it is inherently not something provable, for the notion of a systematic process is unavoidably imprecise. We will return (later in the semester) to why this is a plausible assertion.

We remark that an instruction such as “guess the correct answer” does not seem to be systematic. An instruction such as “try all possible answers” is less clear cut: it depends on whether the possible answers are finite or infinite in number. For specificity, henceforth we will imagine working with a specific programming language, and using this as the definition of reasonable instructions (you can think of this as Java, C, or whatever is your preferred programming language). Actually, in practice we will describe algorithms in pseudo-code and English, but with enough detail that their programmability should be clear.

We define computability in terms of language recognition.

**Definition** $L \subseteq \Sigma^*$ is computable if there is a program $P$ with two possible outputs, “accept” and “reject” and on input $x \in \Sigma^*$, $P$ outputs “accept” if $x \in L$ and “reject” if $x \notin L$.

**Remark** Usually one wants to compute some function $f(x)$ of the input $x$. By allowing 2-input programs we can define computable functions $f$ as follows: $f$ is computable if there is a 2-input program $P$, such that on input $(x, y)$ $P$ outputs “accept” if $f(x) = y$ and “reject” if $f(x) \neq y$.

Often, we will want to treat the text of the program as a possible input to the program. To this end, we take the conventional approach that the program symbols are encoded in ASCII or more simply in binary. We do the same for the input alphabet. Of course, if we are not trying to treat a program as a possible input to another program, we can have distinct alphabets for programs and inputs.
As it turns out, not all languages are computable (indeed, we have already seen that the halting problem is not computable). First, however, we are going to see some computable languages.

The first languages we are going to look at are properties of automata. To this end, we need an agreed method for writing the description of an automata. This is similar to a standard input format for a graph, for example. To describe a DFA we specify it as a series of 4 sets: $\Sigma$, the input alphabet, $Q$, the set of states, $q_{start} \in Q$, the start state, $F \subseteq Q$, the set of accept states, and $\delta$, a description of the edges and their labels. If $\Sigma = \{a_1, a_2, \ldots, a_k\}$, $Q = \{q_1, \ldots, q_r\}$ we could write this as a string

\[
(\{a_1, a_2, \ldots, a_k\}, \{q_1, q_2, \ldots, q_r\}, q_{start}, \{q_{i1}, q_{i2}, \ldots, q_{ij}\},
\]

\[
q_{start}, \{(q_1, a_1) \rightarrow q_{j1}, (q_1, a_2) \rightarrow q_{j2}, \ldots, (q_1, a_k) \rightarrow q_{jk},
\]

\[
(q_2, a_2) \rightarrow q_{jk+1}, \ldots, (q_r, a_k) \rightarrow q_{jk+r}\},
\]

where $F = \{q_{i1}, q_{i2}, \ldots, q_{ij}\}$ and $\delta(q_i, a_j) = q_{jk(i-1)+l}$ for $1 \leq i \leq r, 1 \leq l \leq k$.

There is a further difficulty that we would like to use one alphabet to describe many different machines. One way of doing this is to write the states and the alphabet characters in binary. We then see that the above DFA description uses just 8 distinct characters: $\{, \}, (, ), 0, 1, \rightarrow, ,$. We could reduce this to two, say 0 and 1, by encoding each character using three 0s and 1s.

An input string is a descriptor of a DFA only if it has the above format, and in addition:

(i) $a_i \neq a_j$ for $i \neq j$, i.e. the relevant binary strings are not equal.

(ii) $q_i \neq q_j$ for $i \neq j$.

(iii) $q_{start} = q_i$ for some $i, 1 \leq i \leq r$.

(iv) For each $h, 1 \leq h \leq j$, $q_{lh} = q_i$, for some $l, 1 \leq l \leq r$.

(v) $q_{start} = q_i$ for some $i, 1 \leq i \leq r$.

(vi) In each expression $(q_i, a_l) \rightarrow q_{j(i-1)k+l} q_{j(i-1)k+l} = q_h$ for some $h, 1 \leq h \leq r$.

As this is rather tedious, henceforth we leave the description of correct encodings to the reader (to imagine). We will simply use angle brackets to indicate a suitable encoding in a convenient alphabet (binary say). So $< M >$ denotes an encoding of machine $M$, $< w >$ an encoding of string $w$ (note that $w$ is over some alphabet $\Sigma$, while $< w >$ may be in binary). We also use $< M, w >$ to indicate the encoding of the pair $M$ and $w$.

We are now ready to look at the decidability of some languages.

**Example 1** $A_{DFA} = \{< M, w > | M$ is a DFA and $M$ accepts input $w\}$.

**Claim** $A_{DFA}$ is decidable.
Proof To prove the claim we simply need to give an algorithm to determine whether $M$ accepts its input. Such an algorithm is easily seen at a high level: First check the input is legitimate and if not reject. By legitimate we mean that $<M,w>$ is the encoding of a DFA, followed by the encoding of a string $w$ over the input alphabet for $M$. If the input is legitimate continue by simulating $M$ input $w$, by reading its input character by character and keeping track of the state reached by $M$. When the input is all read check whether an accepting state has been reached and output accordingly (accept if it has, reject otherwise).

The details are a bit more painstaking: given the current state and the character read, look up the state reached by scanning the edge descriptions (the triples $(p,a)\rightarrow q$). In yet more detail, we store the current state in a variable, the input $w$ in a linked list, and the DFA in adjacency list format.

The details of how to implement the algorithm should be clear at this point. As they are not illuminating we are not going to spell them out further. Indeed, henceforth, a description at the level of detail of the first paragraph will be considered sufficient. □

Note. Henceforth we will take it as given that there is an initial step in our algorithms to check that the inputs are legitimate.

Example 2 $A_{NFA} = \{<M,w> | M$ is a description of an NFA and $M$ accepts $w\}$.

Claim $A_{NFA}$ is decidable.

Proof The decidability algorithm simulates $M$ on input $w$ by keeping track of all states reachable on reading the first $i$ characters of the input, for $i = 0, 1, 2, \cdots$ in turn. $M$ accepts $w$ exactly if an accept state is reachable on reading all of $w$, and our program outputs “accept” if $B$ accepts $w$ and outputs “reject” otherwise. □

Example 3 $A_{RFX} = \{<r,w> | r$ is a regular expression that generates $w\}$.

Claim $A_{RFX}$ is decidable.

Proof The algorithm converts $r$ to an NFA $M_r$ using the procedure we have seen in class (some while back). It then forms the encoding $<M_r,w>$ and inputs this to the program $P_{NFA}$ deciding $A_{NFA}$. Its answer (accept or reject) is the same as the one given by $P_{NFA}$ on input $<M_r,w>$.

This is correct for $M_r$ accepts $w$ if and only if $r$ accepts $w$. □

This procedure is taking advantage of an already constructed program and using it as a subroutine. This is a powerful tool which we are going to be using repeatedly.

Example 4 $E_{DFA} = \{<M> | M$ is a DFA and $L(M) = \phi\}$.

Note that $L(M) = \phi$ exactly if no accepting state of $M$ is reachable from its start state. It is easy to give a graph search algorithm to test this.

Claim $E_{DFA}$ is decidable.

Proof The algorithm to test this property determines the collection of states reachable from the start state. If this collection includes an accept state then the algorithm outputs “accept” and otherwise it outputs “reject”. □
Example 5 $EQ_{DFA} = \{ < M_A, M_B > | M_A \text{ and } M_B \text{ are DFAs and } L(M_A) = L(M_B) \}$.

Claim $EQ_{DFA}$ is decidable.

Proof let $A = L(M_A)$ and $B = L(M_B)$. We begin by observing that there is a DFA $\tilde{M}_{AB}$ such that $L(\tilde{M}_{AB}) = \phi$ exactly if $A = B$. For let $C = (A \cap \overline{B}) \cup (\overline{A} \cap B)$. Clearly, if $A = B$ $C = \phi$. While if $C = \phi$, $A \cap \overline{B} = \phi$, so $A \subseteq \overline{B} = B$, and similarly $\overline{A} \cap B$ so $B \subseteq A$; together, these imply $A = B$.

But given DFAs $M_A$ and $M_B$, we can construct DFAs $\overline{M}_A$ and $\overline{M}_B$ to accept $\overline{A}$ and $\overline{B}$ respectively. Then using $M_A$ and $\overline{M}_B$ we can construct DFA $M_{A\overline{B}}$ to accept $A \cap \overline{B}$ and $M_{\overline{A}B}$ to accept $\overline{A} \cap B$. Given $M_{A\overline{B}}, M_{\overline{A}B}$ we can construction $\tilde{M}_{AB}$ to accept $(A \cap \overline{B}) \cup (\overline{A} \cap B)$.

So the algorithm constructs $\tilde{M}_{AB}$ and forms the encoding $< \tilde{M}_{AB} >$. This encoding is then fed to the procedure $P_{EDFA}$ from the preceding example. The output of $P_{EDFA}$ on input $< \tilde{M}_{AB} >$ is the output of the present procedure.

This is correct for $P_{EDFA}$ outputs “accept” exactly if $L(\tilde{M}_{AB}) = \phi$ which is the case exactly if $A = B$. □

We have now given two examples of a use of a subroutine in a very particular form. More specifically, program $P$ has used program $Q$ as a subroutine and then used the answer of $Q$ to compute its own output. In these two cases, the calculation of its output has been the simplest possible: the output of $Q$ has become the output of $P$.

We call this form of algorithm design a reduction. If we have a solution to a problem $A$ (e.g. $E_{DFA}$) using a program (algorithm) $Q$, and we give a program to solve problem $B$ (e.g. $EQ_{DFA}$) using $Q$ as a subroutine then we say we have reduced problem $B$ to problem $A$. What this means is that given we know how to solve problem $A$ we have now also demonstrated how to solve problem $B$. 