Lecture 18

Further discussion of computability

In this lecture we will discuss a few points relating to Turing machines and computability that were not covered in previous lectures. We will begin with some basic closure properties of decidable and semidecidable languages. We will then briefly discuss nondeterministic Turing machines, and relate them to the semidecidable and decidable languages. Finally we will prove an interesting characterization of semidecidability, which is that a nonempty language is semidecidable if and only if it is equal to the *range* of a computable function.

18.1 Closure properties of decidable and semidecidable languages

The decidable and semidecidable languages are closed under many (but not all) of the operations on languages that we have considered thus far in the course. Some examples are described in this section.

Closure properties of decidable languages

First let us observe that the decidable languages are closed under the regular operations. The proof is quite straightforward.

Proposition 18.1. *Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be decidable languages. The languages $A \cup B$, AB, and A^* are decidable.*

Proof. Because the languages A and B are decidable, there must exist a DTM M_A that decides A and a DTM M_B that decides B. The DTMs described in Figures 18.1, 18.2, and 18.3 decide the languages $A \cup B$, AB, and A^*, respectively. It follows that these languages are all decidable. □

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
  1. Run $M_A$ on input $w$.
  2. Run $M_B$ on input $w$.
  3. If either $M_A$ or $M_B$ has accepted, then accept, otherwise reject.
```

Figure 18.1: A DTM M that decides $A \cup B$, given DTMs M_A and M_B that decide A and B, respectively.

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
  1. For every choice of strings $u, v \in \Sigma^*$ satisfying $w = uv$:
       1.1 Run $M_A$ on input $u$.
       1.2 Run $M_B$ on input $v$.
       1.3 If both $M_A$ and $M_B$ have accepted, then accept.
  2. Reject.
```

Figure 18.2: A DTM M that decides AB, given DTMs M_A and M_B that decide A and B, respectively.

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
  1. If $w = \varepsilon$, then accept.
  2. For every way of writing $w = u_1 \cdots u_m$ for nonempty strings
     $u_1, \ldots, u_m$:
       2.1 Run $M_A$ on each of the strings $u_1, \ldots, u_m$.
       2.2 If $M_A$ has accepted on all of these runs, then accept.
  3. Reject.
```

Figure 18.3: A DTM M that decides A^*, given a DTM M_A that decides A.

The decidable languages are also closed under complementation, as the next proposition states. Perhaps this one is simple enough that we can safely skip the proof; it is clear that if a DTM M decides A, and we simply swap the accept and reject states of M, we obtain a DTM deciding $\overline{A}$.

Proposition 18.2. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a decidable language. The language $\overline{A}$ is decidable.*

There are a variety of other operations under which the decidable languages are closed. For example, because the decidable languages are closed under union and complementation, we immediately have that they are closed under intersection and symmetric difference. Another example is string reversal: if a language A is decidable, then A^{R} is also decidable, because a DTM can decide A^{R} simply by reversing its input string and then deciding whether or not that string is contained in A.

There are, however, some natural operations under which the decidable languages are not closed. The following example shows that this is the case for the prefix operation.

Example 18.3. The language $\mathrm{Prefix}(A)$ might not be decidable, even if A is decidable. To construct an example that illustrates that this is so, let us first take H to be a DTM satisfying $\mathrm{L}(H) = \mathrm{HALT}$. We may then define a language $A \subseteq \{0, 1, \#\}^*$ as follows:

$$A = \{w\#0^t \;:\; H \text{ accepts } w \text{ within } t \text{ steps}\}. \tag{18.1}$$

This is a decidable language, but $\mathrm{Prefix}(A)$ is not—for if $\mathrm{Prefix}(A)$ were decidable, then one could easily decide HALT by using the fact that a string $w \in \{0, 1\}^*$ is contained in HALT if and only if $w\# \in \mathrm{Prefix}(A)$; both inclusions are equivalent to the existence of a positive integer t such that $w\#0^t \in A$.

Closure properties of semidecidable languages

The semidecidable languages are also closed under a variety of operations, although not precisely the same operations under which the decidable languages are closed.

Let us begin with the regular operations, under which the semidecidable languages are indeed closed. In this case, one needs to be a bit more careful than in the proof of the analogous proposition for decidable languages; the DTMs semideciding the languages in question might run forever on inputs not in those languages. However, it is nothing that the method for searching infinite spaces from the previous lecture cannot handle.

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
  1. Set $t \leftarrow 1$.
  2. Run $M_A$ on input $w$ for $t$ steps.
  3. Run $M_B$ on input $w$ for $t$ steps.
  4. If either $M_A$ or $M_B$ has accepted, then accept.
  5. Set $t \leftarrow t + 1$ and goto step 2.
```

Figure 18.4: A DTM M that semidecides $A \cup B$, given DTMs M_A and M_B that semidecide A and B, respectively.

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
  1. Set $t \leftarrow 1$.
  2. For every choice of strings $u, v$ satisfying $w = uv$:
       1.1 Run $M_A$ on input $u$ for $t$ steps.
       1.2 Run $M_B$ on input $v$ for $t$ steps.
       1.3 If both $M_A$ and $M_B$ have accepted, then accept.
  3. Set $t \leftarrow t + 1$ and goto step 2.
```

Figure 18.5: A DTM M that semidecides AB, given DTMs M_A and M_B that semidecide A and B, respectively.

Proposition 18.4. *Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be semidecidable languages. The languages $A \cup B$, AB, and A^* are semidecidable.*

Proof. Because the languages A and B are semidecidable, there must exist DTMs M_A and M_B such that $\mathrm{L}(M_A) = A$ and $\mathrm{L}(M_B) = B$. The DTMs described in Figures 18.4, 18.5, and 18.6 semidecide the languages $A \cup B$, AB, and A^*, respectively. It follows that these languages are all semidecidable. □

The semidecidable languages are also closed under intersection. This can be proved through a similar method to closure under union, but in fact this is a situation in which we do not actually need to be as careful about running forever.

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
1. If $w = \varepsilon$, then accept.
2. Set $t \leftarrow 1$.
3. For every way of writing $w = u_1 \cdots u_m$ for nonempty strings
   $u_1, \ldots, u_m$:
   3.1 Run $M_A$ on each of the strings $u_1, \ldots, u_m$ for $t$ steps.
   3.2 If $M_A$ has accepted in all $m$ of these runs, then accept.
4. Set $t \leftarrow t + 1$ and goto step 3.
```

Figure 18.6: A DTM M that semidecides A^*, given a DTM M_A that semidecides A.

```
The DTM $M$ operates as follows on input $w \in \Sigma^*$:
1. Run $M_A$ on input $w$. If $M_A$ rejects $w$, then reject.
2. Run $M_B$ on input $w$. If $M_B$ rejects $w$, then reject.
3. Accept.
```

Figure 18.7: A DTM M that semidecides $A \cap B$, given DTMs M_A and M_B that semidecide A and B, respectively. Note that if either M_A or M_B runs forever on input w, then so does M, but this does not change the fact that M semidecides $A \cap B$.

Proposition 18.5. *Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be semidecidable languages. The language $A \cap B$ is semidecidable.*

Proof. Because the languages A and B are semidecidable, there must exist DTMs M_A and M_B such that $L(M_A) = A$ and $L(M_B) = B$. The DTM M described in Figure 18.7 semidecides $A \cap B$, which implies that $A \cap B$ is semidecidable. $\square$

The semidecidable languages are not closed under complementation. This follows from Theorem 17.1 from the previous lecture. In particular, if $\overline{A}$ were semidecidable for every semidecidable language A, then we would conclude from that theorem that every semidecidable language is decidable, which is not the case. For example, HALT is semidecidable but not decidable.

Finally, there are some operations under which the semidecidable languages are closed, but under which the decidable languages are not. For example, if A is semidecidable, then so are the languages $\text{Prefix}(A)$, $\text{Suffix}(A)$, and $\text{Substring}(A)$.

18.2 Nondeterministic Turing machines

We have focused on a deterministic variant of the Turing machine model, but it is possible to define a nondeterministic variant of it. This is done in a way that is completely analogous to the definition of nondeterministic finite automata, as compared with deterministic finite automata.

Definition 18.6. A *nondeterministic Turing machine* (or NTM, for short) is a 7-tuple

$$N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}), \tag{18.2}$$

where Q is a finite and nonempty set of states, Σ is an alphabet called the input alphabet, which may not include the blank symbol $\sqcup$, Γ is an alphabet called the tape alphabet, which must satisfy $\Sigma \cup \{\sqcup\} \subseteq \Gamma$, δ is a transition function having the form

$$\delta : Q\backslash\{q_{\text{acc}}, q_{\text{rej}}\} \times \Gamma \rightarrow \mathcal{P}\big(Q \times \Gamma \times \{\leftarrow, \rightarrow\}\big), \tag{18.3}$$

$q_0 \in Q$ is the initial state, and $q_{\text{acc}}, q_{\text{rej}} \in Q$ are the accept and reject states, which must satisfy $q_{\text{acc}} \neq q_{\text{rej}}$.

As one would naturally guess, nondeterministic Turing machines may have multiple choices for which actions can be performed on a given step. Specifically, for a given nonhalting state $q \in Q\backslash\{q_{\text{acc}}, q_{\text{rej}}\}$ and tape symbol a, the value $\delta(q, a)$ taken by the transition function is a subset of $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, rather than a single element of this set, indicating a set of choices for the machine's actions when it is in state q and reading the tape symbol a.

For example, if we have an NTM N whose transition function satisfies

$$\delta(p, a) = \{(q, b, \leftarrow), (r, c, \rightarrow)\}, \tag{18.4}$$

then when N is in state p and scanning a tape square containing the symbol a, then it may either change state to q, overwrite a with b on the tape, and move the tape head left; or it may change state to r, overwrite a with c, and move the tape head right.

Yields relation and acceptance for an NTM

The yields relation of an NTM is defined in a similar manner to DTMs, allowing for multiple choices of configurations that are reachable from a given one. Specifically, the definition is precisely the same as Definition 12.2, except that the conditions

$$\delta(p,a) = (q,b,\rightarrow) \quad \text{and} \quad \delta(p,a) = (q,b,\leftarrow) \tag{18.5}$$

are replaced by

$$(q,b,\rightarrow) \in \delta(p,a) \quad \text{and} \quad (q,b,\leftarrow) \in \delta(p,a), \tag{18.6}$$

respectively.

Acceptance for an NTM is defined in a similar way to NFAs (and PDAs as well). That is, if there *exists* a sequence of computation steps that reach an accepting configuration, then the NTM accepts, and otherwise it does not. Formally speaking, an NTM N accepts an input string w if there exist strings $u, v \in \Gamma^*$ and a symbol $a \in \Gamma$ such that

$$(q_0, \sqcup)w \vdash_N^* u(q_{\text{acc}}, a)v. \tag{18.7}$$

As usual, we write $\mathrm{L}(N)$ to denote the language of all strings w accepted by an NTM N, which we refer to as the language recognized by N.

Computation trees

When we think about the computation of an NTM N on an input string w, it is natural to envision a *computation tree*. This is a (possibly infinite) tree in which each node is labeled by a configuration: the root node of the tree is labeled by the initial configuration of N on input w, and in general each node in the tree has one child labeled by each configuration that can be reached in one step from the one labeling the original node. As a point of clarification, note that distinct nodes in the computation tree may be labeled by the same configuration. That is to say, there could be more than one *computational path* that leads to a given configuration. Finally, note that the leaves of the computation tree are the ones labeled by halting configurations.

It should perhaps be noted that the computation tree of a *deterministic* Turing machine on a given input can be defined in exactly the same way, but it just does not happen to be very interesting. It is a stick, finite or infinite, without branching.

In terms of the computation tree of N on input w, saying that N accepts w is equivalent to saying that *somewhere* in the computation tree there exists an accepting configuration. There might or might not exist rejecting configurations in the computation tree, and there might or might not exist infinitely long branches in the tree, but it does not matter: all that matters when it comes to defining acceptance is whether or not there exists an accepting configuration that is reachable from the initial configuration.

Semidecidability and decidability through NTMs

As the following theorem states, the class of languages recognized by NTMs is exactly the same as for DTMs.

Theorem 18.7. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language. The language A is semidecidable if and only if there exists an NTM N such that $\mathrm{L}(N) = A$.*

It is not difficult to prove this theorem, but these notes will only briefly discuss the main idea behind the proof.

It is clear that every semidecidable language is recognized by some NTM, as we may view a DTM as being an NTM just like we can view a DFA as being an NFA. The other implication required to prove the theorem is that every language recognized by an NTM is semidecidable. That is, one must prove that if $A = \mathrm{L}(N)$ for an NTM N, then A is semidecidable. The key idea through which this may be proved is to perform a *breadth-first search* of the computation tree of N on a given input, using a DTM. If there is an accepting configuration anywhere in the tree, a breadth-first search will find it. Of course, because computation trees may be infinite, the search might never terminate—this is unavoidable, but it is not an obstacle for proving the theorem.

It should perhaps be noted that an alternative approach of performing a *depth-first search* of the computation tree would not work: it might happen that such a search descends down an infinite path of the tree, potentially missing an accepting configuration elsewhere in the tree.

One may also characterize decidable languages through nondeterministic Turing machines, although the required conditions on nondeterministic computations for decidable languages is not quite as simple to state. The following theorem provides such a characterization.

Theorem 18.8. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language. The language A is decidable if and only if there exists an NTM N for which the following two conditions are met:*

1. $A = \mathrm{L}(N)$.
2. *For every input string $w \in \Sigma^*$, the computation tree of N on input w is finite.*

This theorem can be proved using exactly the same technique, using breadth-first search, as the previous theorem. In this case, the DTM performing the search

The DTM M operates as follows on input $w \in \Sigma^*$:

```
1. Set $x \leftarrow \varepsilon$.
2. Compute $y = f(x)$, and accept if $w = y$.
3. Increment $x$ with respect to the lexicographic ordering of $\Gamma^*$ and goto
   step 2.
```

Figure 18.8: A DTM M that semidecides $A = \text{range}(f)$ for a computable function $f : \Gamma^* \to \Sigma^*$.

rejects whenever the entire computation tree has been searched and an accepting configuration has not been found.

18.3 The range of a computable function

Finally, we will observe an interesting alternative characterization of semidecidable languages (excepting the empty language), which is that they are precisely the languages that are equal to the *range* of a computable function. Recall that the range of a function $f : \Gamma^* \to \Sigma^*$ is defined as follows:

$$\text{range}(f) = \{f(w) \,:\, w \in \Gamma^*\}. \tag{18.8}$$

Theorem 18.9. *Let Σ and Γ be alphabets and let $A \subseteq \Sigma^*$ be a nonempty language. The following two statements are equivalent:*

1. *A is semidecidable.*
2. *There exists a computable function $f : \Gamma^* \to \Sigma^*$ such that $A = \text{range}(f)$.*

Proof. Let us first prove that the second statement implies the first. That is, we will prove that if there exists a computable function $f : \Gamma^* \to \Sigma^*$ such that $A = \text{range}(f)$, then A is semidecidable. Consider the DTM M described in Figure 18.8. In essence, this DTM searches over Γ^* to find a string that f maps to a given input string w. If it is the case that $w \in \text{range}(f)$, then M will eventually find $x \in \Gamma^*$ such that $f(x) = w$ and accept, while M will run forever if $w \notin \text{range}(f)$. Thus, we have $L(M) = \text{range}(f) = A$, which implies that A is semidecidable.

For the other implication, which is slightly more difficult, suppose that A is semidecidable. Thus, there exists a DTM M such that $L(M) = A$. We will also make use of the assumption that A is nonempty—there exists at least one string in A, so we may take w_0 to be such a string. If you like, you may define w_0 more concretely as the first string in A with respect to the lexicographic ordering of Σ^*, but it is not important for the proof that we make this particular choice.

Now define a function $f : \Gamma^* \rightarrow \Sigma^*$ as follows:

$$f(x) = \begin{cases} w & \text{if } x = \langle\langle w\rangle, \langle t\rangle\rangle, \text{ for } w \in \Sigma^* \text{ and } t \in \mathbb{N}, \\ & \text{and } M \text{ accepts } w \text{ within } t \text{ steps} \\ w_0 & \text{otherwise.} \end{cases} \tag{18.9}$$

Here we assume that $\langle\langle w\rangle, \langle t\rangle\rangle$ refers to any encoding scheme through which the string $w \in \Sigma^*$ and the natural number $t \in \mathbb{N}$ are encoded into a single string over the alphabet Γ. As we discussed earlier in the course, this is possible for any alphabet Γ, even if it contains only a single symbol.

It is evident that the function f is computable: a DTM M_f can compute f by checking to see if the input has the form $\langle\langle w\rangle, \langle t\rangle\rangle$, simulating M for t steps on input w if so, and then outputting either w or w_0 depending on the outcome. If M accepts a particular string w, then it must be that $w = f(\langle\langle w\rangle, \langle t\rangle\rangle)$ for some sufficiently large natural number t, so $A \subseteq \text{range}(f)$. On the other hand, every output of f is either a string w accepted by M or the string w_0, and therefore $\text{range}(f) \subseteq A$. It is therefore the case that $A = \text{range}(f)$, which completes the proof. □

Remark 18.10. The assumption that A is nonempty is essential in the previous theorem because it cannot be that $\text{range}(f) = \varnothing$ for a computable function f. Indeed, it cannot be that $\text{range}(f) = \varnothing$ for any function whatsoever.

Theorem 18.9 provides a very useful characterization of semidecidable languages. For instance, one can use this theorem to come up with alternative proofs for all of the closure properties of the semidecidable languages stated in the first section of this lecture.

For example, suppose that $A, B \subseteq \Sigma^*$ are nonempty semidecidable languages. By Theorem 18.9, for whatever alphabet Γ we choose, there must exist computable functions $f : \Gamma^* \rightarrow \Sigma^*$ and $g : \Gamma^* \rightarrow \Sigma^*$ such that $\text{range}(f) = A$ and $\text{range}(g) = B$. Define a new function

$$h : (\Gamma \cup \{\#\})^* \rightarrow \Sigma^* \tag{18.10}$$

as follows:

$$g(x) = \begin{cases} f(y)g(z) & \text{if } x = y\#z \text{ for } y \in \Gamma^* \text{ and } z \in \Gamma^* \\ f(\varepsilon)g(\varepsilon) & \text{otherwise.} \end{cases} \tag{18.11}$$

Here, we are naturally assuming that $\# \notin \Gamma$. One sees that h is computable, and $\text{range}(h) = AB$, which implies that AB is semidecidable.

The DTM M operates as follows on input $w \in \Sigma^*$:

1. Set $x \leftarrow \varepsilon$.
2. Compute $y \leftarrow f(x)$.
3. If $y = w$ then *accept*.
4. If $y > w$ (with respect to the lexicographic ordering of Σ^*) then *reject*.
5. Increment x with respect to the lexicographic ordering of Σ^* and goto step 2.

Figure 18.9: A DTM M for Corollary 18.11.

Here is another example of an application of Theorem 18.9, establishing that every infinite semidecidable language must have an *infinite* decidable language as a subset.

Corollary 18.11. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be any infinite semidecidable language. There exists an infinite decidable language $B \subseteq A$.*

Proof. Because A is infinite (and therefore nonempty), there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that $A = \text{range}(f)$.

We will now define a language B by first defining a DTM M and then taking $B = \text{L}(M)$. In order for us to be sure that B satisfies the requirements of the corollary, it will need to be proved that M never runs forever (so that B is decidable), that M only accepts strings that are contained in A (so that $B \subseteq A$), and that M accepts infinitely many different strings (so that B is infinite). The DTM M is described in Figure 18.9.

The fact that M never runs forever follows from the assumption that A is infinite. That is, because A is infinite, the function f must output infinitely many different strings, so regardless of what input string w is input into M, the loop will eventually reach a string x so that $f(x) = w$ or $f(x) > w$, causing M to halt.

The fact that M only accepts strings in A follows from the fact that the condition for acceptance is that the input string w is equal to y, which is contained in $\text{range}(f) = A$.

Finally, let us observe that M accepts precisely the strings in this set:

$$\left\{ w \in \Sigma^* \;:\; \begin{array}{l} \text{there exists } x \in \Sigma^* \text{ such that } w = f(x) \\ \text{and } w > f(z) \text{ for all } z < x \end{array} \right\}. \tag{18.12}$$

The fact that this set is infinite follows from the assumption that $A = \text{range}(f)$ is infinite—for if the set were finite, there would necessarily be a maximal output of f with respect to the lexicographic ordering of Σ^*, contradicting the assumption that $\text{range}(f)$ is infinite.

The language $B = \text{L}(M)$ therefore satisfies the requirements of the corollary, which completes the proof. □