DISCLAIMER: These notes are not an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

- Algorithms and decidability.
- Quiz 9.
- Undecidable problems.

Last week we introduced Turing machines, which are finite automatas augmented with tapes, instead of stacks or read-only input. What’s particularly interesting about Turing machines is that they can simulate themselves: one can give a Turing machine \( M \) as well as its input \( w \) to another Turing machine (known as the universal Turing machine), which then simulates running \( M \) on \( w \). This says that in a sense, Turing machine are the most powerful computing objects that we know.

So the question becomes: what can’t Turing machines compute. For this we need the notion of decidability: a Turing machine may either accept, reject, or never terminate on some input. Formally, recall the distinction between a recognizer and a decider:

- A Turing machine recognizes a language \( L \) if it (only) accepts all strings in \( L \).
- A Turing machine decides \( L \) if it always terminates (outputs accept or reject), and recognizes \( L \).

Most of the tasks that we discussed with regular / context free languages are decidable. Some examples from the text book are:

1. \(< B, w >: B \) is a DFA that accepts \( w \).
2. \(< B, w >: B \) is a NFA that accepts \( w \).
3. \(< R, w >: R \) is a regular expression that generates \( w \).
4. \(< A >: A \) is a DFA and \( L(A) = \emptyset \).
5. \(< A, B >: A \) and \( B \) are DFAs and \( L(A) = L(B) \).
6. \(< G, w >: G \) is a CFG that generates \( w \).
7. \(< G >: G \) is a CFG that generates a non-empty language.
1 A Turing-Undecidable Language

The goal today is to give a language that is not decidable, and it has to do with universal Turing machines:

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} . \]

Recall from the definition of universal Turing machines that \( A_{TM} \) is Turing-recognizable. So this is also an example of a problem that is Turing-recognizable, but not Turing-decidable.

The proof is by contradiction: assume \( A_{TM} \) is decidable. Then let \( H \) be a decider, aka

\[ H \left( \langle M, w \rangle \right) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \smallskip \text{reject} & \text{if } M \text{ rejects } w, \text{ or does not halt on } w \end{cases} \]

Now consider a machine \( D \) that negates the output of running a Turing machine \( M \) on its own input:

\[ D (M) = \neg H \left( M, < M > \right) . \]

In other words,

\[ D \left( \langle M \rangle \right) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } < M > \\ \text{reject} & \text{if } M \text{ accepts } < M > \end{cases} \]

Now let’s loop this argument onto itself once again. What happens if we run \( D \) on itself? We would get

\[ D \left( \langle D \rangle \right) = \begin{cases} \text{accept} & \text{if } D \text{ does not accept } < D > \\ \text{reject} & \text{if } D \text{ accepts } < D > \end{cases} \]

which gives a contradiction in either case! So the only thing that this could contradict is the existence of such a machine \( D \) that can tell whether a computation terminates.

2 A Turing-Unrecognizable Language

We now take these ideas further to show that there are also languages that are not Turing-recognizable. For this, we use \( \overline{A_{TM}} \), the complement of the languages consists of Turing machines that accept input \( w \).

The proof is by contradiction. Suppose \( \overline{A_{TM}} \) is Turing-recognizable by some Turing machine \( M_2 \), Recall that \( A_{TM} \) is Turing recognizable by some Turing machine \( M_1 \). we show that we can decide on \( A_{TM} \) by running \( M_1 \) and \( M_2 \) in parallel.

Given some input \( w \), consider alternating running \( M_1 \) and \( M_2 \) on some input \( w \), one step each, and return the outcome when either one of them accepts. We claim in either case of \( w \in A_{TM} \) or \( w \notin A_{TM} \), this process will halt:
1. If \( w \) is in \( A_{TM} \), then \( M_1 \) accepts \( w \) after \( k \) steps for some value \( k \), and the overall process halts after \( 2k \) steps.

2. If \( w \) is in \( \overline{A_{TM}} \), then \( M_2 \) accepts \( w \) after \( k \) steps for some value \( k \), and the overall process halts after \( 2k \) steps as well.

Section 4.2 of the textbook extends this to a theorem giving that a language \( A \) is decidable if and only if it’s both Turing-recognizable and co-Turing-recognizable.