DECIDABILITY AND UNDECIDABILITY

Decidable problems from language theory

For simple machine models, such as finite automata or pushdown automata, many decision problems are solvable. In the case of deterministic finite automata, problems like equivalence can be solved even in polynomial time. Also there are efficient parsing algorithms for context-free grammars.

If necessary we may briefly review some material on regular and context-free languages from chapters 1 and 2 in the textbook. Recall in particular the following important characterization:

Regular languages = languages denoted by regular expressions
= languages accepted by DFAs (deterministic finite automata)
= languages accepted by NFAs (nondeterministic finite automata).

The class of regular languages is strictly contained in the deterministic context-free languages (DCFL) which in turn are strictly contained in the (general) context-free languages. The class DCFL consists of languages recognized by deterministic pushdown automata.

We recall the following basic notions. A decision problem is a restricted type of an algorithmic problem where for each input there are only two possible outputs.

- A *decision problem* is a function that associates with each input instance of the problem a truth value *true* or *false*.
- A *decision algorithm* is an algorithm that computes the correct truth value for each input instance of a decision problem. The algorithm has to terminate on all inputs!
- A decision problem is *decidable* if there exists a decision algorithm for it. Otherwise it is *undecidable*.

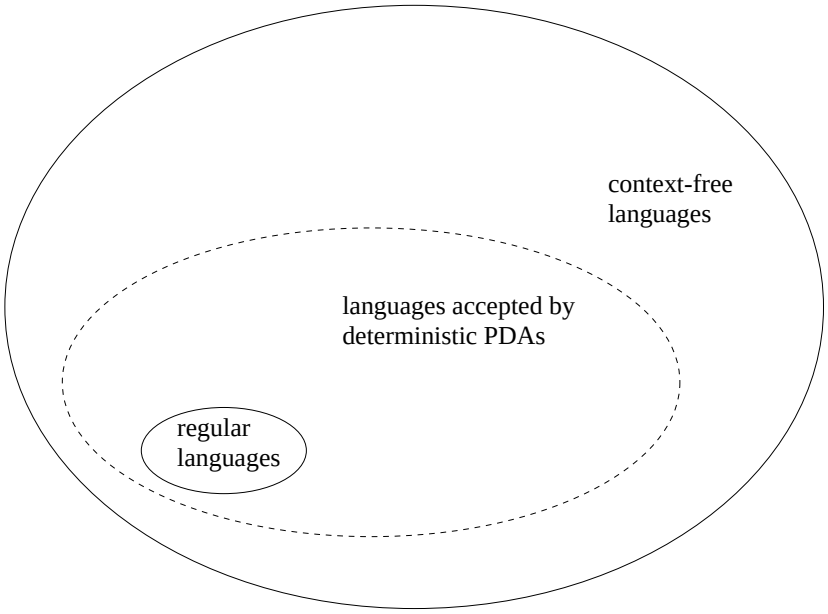

Figure 1: Regular, context-free and deterministic context-free languages

To show that a decision problem is decidable it is sufficient to give an algorithm for it. On the other hand, how could we possibly establish (= prove) that some decision problem is undecidable? This is one of the questions we will address in the course.

Decidability properties of regular languages

Important decision problems for finite automata include the following:

1. DFA membership

   INSTANCE: A DFA $M = (Q, \Sigma, \delta, q_0, F)$ and a string $w \in \Sigma^*$

   QUESTION: Is $w \in L(M)$?

   Proposition. DFA membership is decidable.

   Proof. To be explained in class: the algorithm simulates the given DFA on the given input.

2. DFA emptiness.

INSTANCE: A DFA $M = (Q, \Sigma, \delta, q_0, F)$

QUESTION: Is $L(M) = \emptyset$?

Theorem. DFA emptiness is decidable.

Proof. We note that $L(M) = \emptyset$ iff there is no path in the state diagram of M from q_0 to a final state. If $F = \emptyset$, then clearly $L(M) = \emptyset$. Otherwise, we use a graph reachability algorithm to enumerate all states that can be reached from q_0 and check whether this set contains some state of F. The algorithm terminates because the state diagram is finite.

(This result is explained in Ch. 4 of the textbook.)

3. DFA universality

INSTANCE: A DFA $M = (Q, \Sigma, \delta, q_0, F)$

QUESTION: Is $L(M) = \Sigma^*$?

Theorem. DFA universality is decidable.

Proof: in class

4. DFA containment

INSTANCE: Two DFAs $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M = (Q_2, \Sigma, \delta_2, q_2, F_2)$

QUESTION: Is $L(M_1) \subseteq L(M_2)$?

Theorem. DFA containment is decidable.

Proof hint: using closure properties of regular languages reduce the question to checking emptiness.

5. DFA equivalence

INSTANCE: Two DFAs $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M = (Q_2, \Sigma, \delta_2, q_2, F_2)$

QUESTION: Is $L(M_1) = L(M_2)$?

Theorem. DFA equivalence is decidable.

Proof hint: reduce the question to checking containment (see section 4.1).

Regular languages are useful for many practical applications due to the fact that "all natural" questions concerning regular languages are decidable.[1] The downside is that the family of regular languages is quite small.

As we will see, already for context-free languages some of the above questions are *undecidable* (universality, containment, equivalence).

For languages accepted by general Turing machines, as we will shortly find out, *all non-trivial questions are undecidable!*

Also the corresponding decision problems for context-free grammars are discussed in section 4.1.

Context-free membership A_{CFG}

INSTANCE: A CF grammar $G = (V, \Sigma, R, S)$ and a string $w \in \Sigma^*$.

QUESTION: Is $w \in L(G)$?

Formally, A_{CFG} is defined to be the language consisting of all encodings[2] of a pair consisting of a CFG and some string generated by the grammar:

$$A_{CFG} = \{< G, w > | \ G \text{ is a CFG and } w \in L(G)\}$$

Theorem. A_{CFG} is decidable.

[1] There are known exceptions, but these are somewhat "artificial" problems.

[2] Inputs to TMs must be encoded as strings. The notation for the encoding is explained at the end of chapter 3 in the textbook.

Note: In the context of computability theory, to show that A_{CFG} is decidable it is sufficient to use a simple brute-force parsing algorithm. Context-free grammars can be parsed efficiently and the best known parsing algorithms for general context-free grammars have time complexity (slightly less than) $O(n^3)$.

Similarly, the context-free emptiness problem is encoded as a language:

$$E_{CFG} = \{< G >| \ G \text{ is a CFG and } L(G) = \emptyset\}$$

Theorem. E_{CFG} is decidable.

Proof. In class.

Also, we can consider the equivalence problem for context-free languages. Formally this can be encoded as the language

$$EQ_{CFG} = \{< G, H >| \ G, H \text{ are CFGs and } L(G) = L(H)\}$$

Recall that we have an algorithm that decides equivalence of DFAs or NFAs. However, the same approach that was used to establish the decidability result for DFAs does not work if we try to show that EQ_{CFG} is decidable. Why not?

In fact, it turns out that EQ_{CFG} is not decidable, that is, the equivalence problem for context-free grammars is undecidable! We will come back to this later[3].

Undecidable problems

We will now discuss the notion of undecidability. This is section 4.2 in the textbook. First let us review some terminology.

[3]The above observations that our previous method cannot be used to establish decidability still in no way guarantee that the question is undecidable.

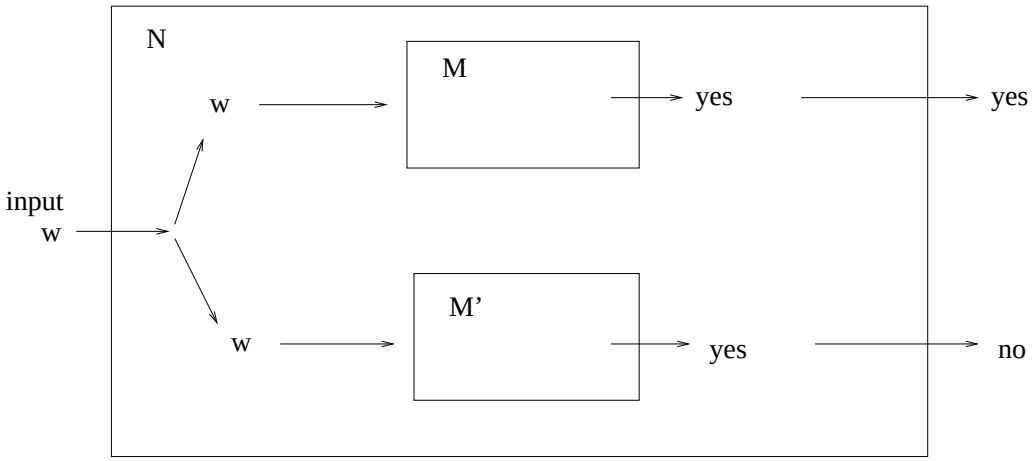

Figure 2: A decider for the language L.

- A language is *decidable* if some TM decides it (chapter 3). All computations of a decider TM must halt. Decidable languages are often called also *recursive languages.*

- A language is *Turing-recognizable* (or *recursively enumerable*) if it is recognized by a TM. That is, all words in the language are accepted by the TM. On words not belonging to the language, the computation of the TM either rejects or goes on forever.

Lemma. A language L is decidable if and only if $\bar{L}$ is decidable.
Proof: in class

Theorem. L is decidable if and only if both L and $\bar{L}$ are Turing-recognizable.

Proof. *(only if):* Follows from the previous lemma and the fact that every decidable language is Turing-recognizable.

(if): Let M be a TM recognizing L and M' a TM recognizing $\bar{L}$. We construct a decider N for the language L, see Figure 2.

The decider N can be implemented as a 2-tape TM that on its first tape simulates M and "in parallel" on the second tape simulates M'. If the simulation on tape one ac-

cepts, N accepts. If the simulation on tape two accepts, N rejects. One of the simulations necessarily halts in a finite number of steps. (Why?)

The standard example of an undecidable language is:

$$L_{TMaccept} = \{< M, w >| \ M \text{ is a TM and } M \text{ accepts } w\}$$

Theorem. $L_{TMaccept}$ is undecidable.

The proof (to be gone through in class) shows that, in fact, the more restricted language

$$L_{selfaccept} = \{< M, < M >>| \ M \text{ is a TM } \}$$

is undecidable. The crucial idea is diagonalization.

Universal Turing machines

General purpose computers operate as follows:

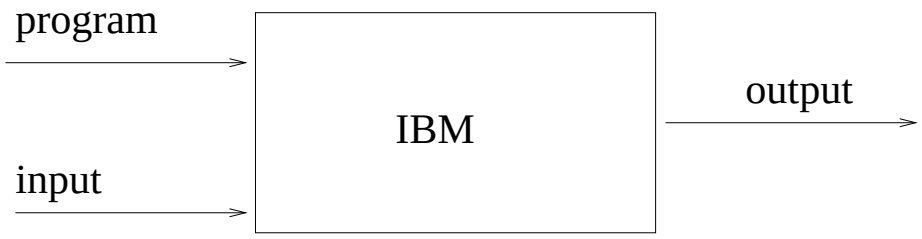

Figure 3: Programmable computer

Similarly, can view a universal Turing-machine to be "programmable":

$< M >$: encoding of TM M;

$< x >$: encoding of input x to M;

$< y >$: encoding of output produced by M.

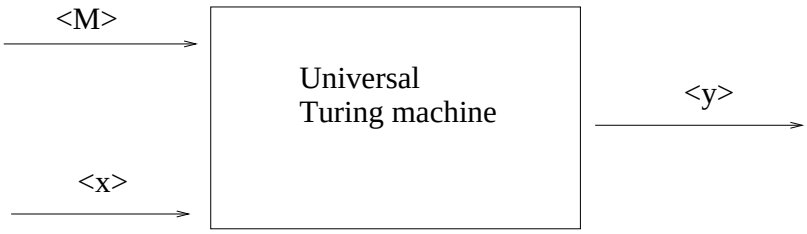

Figure 4: "Programmable" (universal) Turing-machine

The proof of the existence of universal TMs is *constructive.*

Outline of the proof:

- We use three tapes:

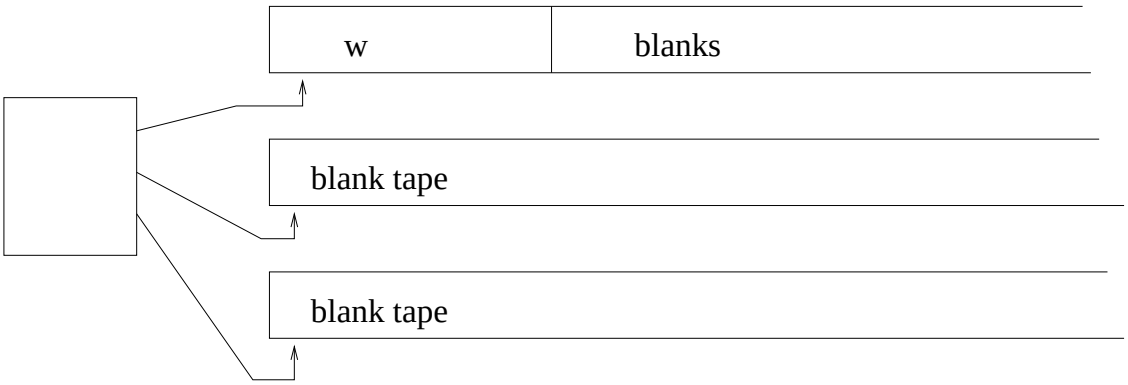

Figure 5: The configuration at the beginning, here $w =< M, x >$.

- Steps:

  1. Check the validity of the input (correct encoding of a TM M and an input x for M).
  2. Copy from the input tape the string x to the second tape.
  3. Write the start state of M onto third tape.
  4. Start the simulation, see Figure 6.

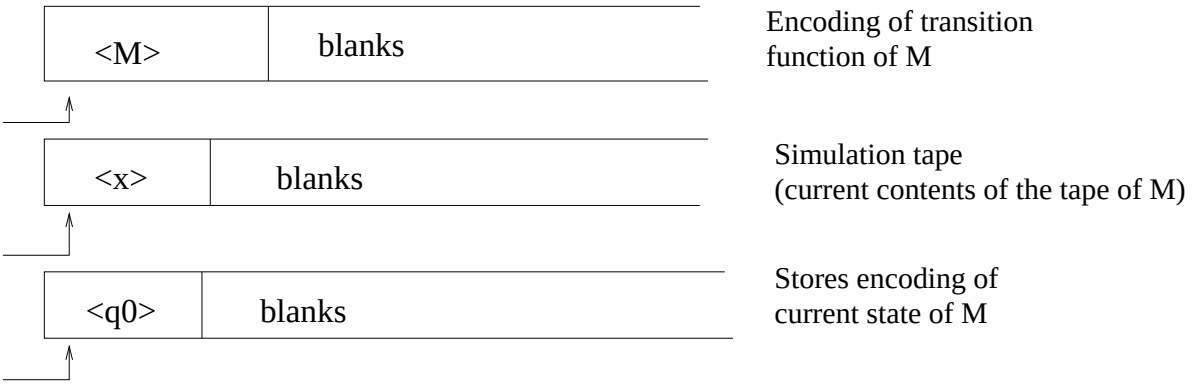

Figure 6: The contents of the tapes after steps 1., 2., 3. After this the universal machine is ready to begin the simulation of M.

Note:

- A universal TM has a fixed tape alphabet.
- Different TMs have different state sets and tape alphabets, and these may be arbitrarily large finite sets.

Consequently the TMs given as input for a universal TM must be encoded using a fixed alphabet. The encoding must include the state set, tape alphabet and transition function. This is illustrated in the below example.

Example.

The TM of Figure 7 could be first encoded as a string:

$[\delta(0, a) = (1, b, R);\ \delta(1, b) = (0, b, R);\ \delta(1, \sqcup) = (2, \sqcup, L);$ 1s; 2acc]

Above "1s" would denote that "1" is the start state and "2acc" denotes that "2" is the accept state.

The above string could then straightforwardly be encoded using a binary alphabet. In this way, any Turing machine can be encoded as a string over the binary alphabet.

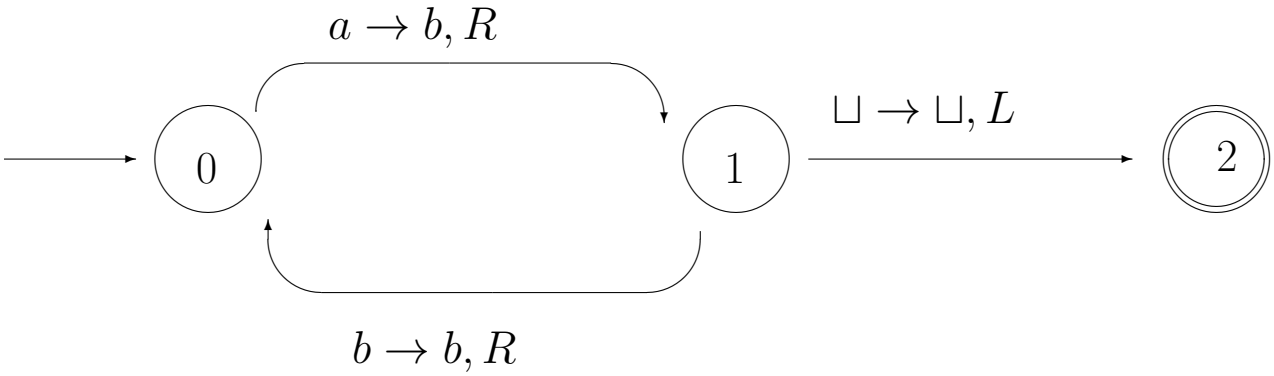

Figure 7: An example of a Turing machine.

- There exist fairly small universal TMs. For example, we can construct a universal TM that has 7 states and the tape alphabet has 4 symbols.
- There do not exist "universal DFAs", that is, DFAs that could simulate any other DFA. Why not?

The language

$$L_{TMaccept} = \{< M, w > |\ M \text{ is a TM and } M \text{ accepts } w\}$$

is recognized by a universal TM!

- This shows that there exist Turing-recognizable languages that are not decidable.
- On the other hand, the complement of $L_{TMaccept}$ is not Turing-recognizable. Why not?