FURTHER UNDECIDABLE PROBLEMS; REDUCTIONS

This material is covered in Chapter 5 of the textbook.

- The undecidability of the language A_{TM} means that there is no algorithm that decides, for an arbitrary given Turing machine M and input string w, whether or not M accepts w.

- Naturally in some individual cases it will be possible to predict whether a particular Turing machine accepts a given input. The undecidability of the general problem means just that there is no algorithm that gives the correct answer for all inputs.

- The undecidability of A_{TM} was established using *diagonalization.* Now that we have one language that is *known to be undecidable* it is possible to establish the undecidability of many other languages (problems) using a technique called *reduction.*

Informal definition of reductions

Problem A reduces to problem B, if an algorithm for solving B (a TM to decide B) can be used to construct an algorithm to solve A (a TM to decide A).
(Reducibility will be defined formally later.)

The above means that if A reduces to B and B is decidable, then also A is decidable.

On the other hand, if A is known to be undecidable and A reduces to B then also B is undecidable.

This observation is the basis for establishing the undecidability of new problems relying on the fact that we already know that some other problem is undecidable.

What is usually called the TM halting problem can be encoded as the following language:

$$HALT_{TM} = \{< M, w > | \ M \text{ is a TM and } M \text{ halts on input } w\}$$

The language $HALT_{TM}$ is somehow related to the language A_{TM} (the acceptance problem for TMs), however, the precise definitions differ. Due to the similarity of the problems, it is straightforward to reduce A_{TM} to $HALT_{TM}$, which is what we do in the following proof.

Theorem. $HALT_{TM}$ is undecidable.

Proof. For the sake of contradiction we assume that a TM N decides $HALT_{TM}$. We construct a TM P to decide A_{TM} as follows:

Operation of P:
On an input $< M, w >$ where M is a TM and w an input string for M we do the following:

1. Run N on input $< M, w >$
2. If N rejects, reject /* the computation of M on w is infinite */
3. If N accepts, simulate M on w until it halts
   /* now we can be sure that M halts */
4. If M accepts, return accept;
   if M rejects, return reject.

Assuming that N correctly decides $HALT_{TM}$, the above algorithm decides A_{TM}. The latter is impossible since we know that A_{TM} is undecidable.

We conclude that N cannot exist and $HALT_{TM}$ is undecidable. □

The above proof followed the typical schema used in reducibility proofs for undecidability:

1. We reduced A_{TM} to $HALT_{TM}$ (that is, used the hypothetical decider N to construct a decider for A_{TM}).

2. Since A_{TM} was known to be undecidable we could conclude that also $HALT_{TM}$ is undecidable.

Note that the argument does not in any way assume the existence of the decider N. In fact, as a consequence we conclude that N cannot exist!

Using reductions from A_{TM} it is shown in the textbook section 5.1 that for languages recognized by Turing machines also the following questions are undecidable:

- emptiness,
- equivalence,
- regularity.

Furthermore, also the following questions are undecidable for Turing machines:

- context-freeness,
- finiteness,
- etc. etc.

By generalizing the above proof techniques it is possible to prove a very strong undecidability result that is called *Rice's theorem.* Below we formulate this result and provide some discussion on it. (The statement of Rice's theorem is given as problem 5.28 and the proof can be found in the solutions section.)

A property P of a Turing machine M is called a semantic property if it depends only on the language recognized by M and not on the syntactic structure of M.

Examples of semantic properties include:

- M accepts the empty string;

- M accepts some string;
- M accepts infinitely many strings;
- the language recognized by M is regular.

Thus if $L(M_1) = L(M_2)$, then the TMs M_1 and M_2 have exactly the same semantic properties.

More formally, a property P can be expressed as a language consisting of exactly the encodings $< M >$ where M has the property P.

A semantic property P is said to be non-trivial if there exists a Turing machine M_1 such that $< M_1 > \in P$ and a Turing machine M_2 such that $< M_2 > \notin P$.

- What would be an example of a trivial semantic property?
- What is an example of a property of Turing machines that is not semantic?

Now Rice's theorem can be formulated as follows:

Theorem (Rice). All non-trivial semantic properties of Turing machines are undecidable.

Rice's theorem means that for an arbitrary given Turing machine M we cannot decide any properties concerning $L(M)$ except properties that are true for exactly all or exactly none of the languages recognized by Turing machines.

Linear bounded automata

Definition (see section 5.1.) A *linear bounded automaton* (LBA) is a Turing machine where the tape head is restricted to the part of the tape that initially contains the input.

In other words, an LBA cannot "expand workspace" to the part of the tape that initially consists of blank symbols. When an LBA reaches the end of the input (the first blank) it can only stay in place or make a left move.

Within the tape squares that contained the input, an LBA can move freely in both directions and rewrite the symbols occurring there.

The LBAs can recognize all context-free languages, and the usual examples of non-context-free languages. It is, in fact, not easy to find examples of languages that cannot be recognized by LBAs. Using diagonalization we can show that such languages do exist.

Chomsky Hierarchy

- Type 3 languages = regular languages
- Type 2 languages = context-free languages
- Type 1 languages = languages recognized by nondeterministic LBAs
- Type 0 languages = Turing-recognizable languages

- All inclusions in the Chomsky hierarchy are strict, that is, the family of Type $i+1$ languages is strictly included in the family of Type i languages, $i = 0, 1, 2$.
- It is an open question whether the deterministic and nondeterministic LBAs recognize the same family of languages.

The LBAs are an example of *resource bounded* Turing machines. In the second half of the course we will see that comparing the power of determinism and nondeterminism with the presence of resource bounds is usually very difficult.

Recall that (unrestricted) deterministic Turing machines can simulate nondeterministic Turing machines, and this result was not too difficult to obtain.

As before we can consider the membership and emptiness questions for LBAs. If not separately mentioned, here a linear bounded automaton means a deterministic LBA.

$$A_{LBA} = \{< M, w >| \ M \text{ is an LBA that accepts string } w\}$$

$$E_{LBA} = \{< M >| \ M \text{ is an LBA and } L(M) = \emptyset\}$$

Theorem.

1. A_{LBA} is decidable.

2. E_{LBA} is undecidable.

Post Correspondence Problem

The Post Correspondence Problem (named after Emil Post) is a particularly simple undecidable problem that is useful for establishing the undecidability of, for instance, many properties concerning context-free grammars.

Let Σ be an alphabet containing at least two symbols. An instance of the Post Correspondence Problem (PCB) is a sequence of pairs of strings

$$(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$$

where $u_i, v_i \in \Sigma^*$, $i = 1, \ldots, k$.

A solution of the above instance is a sequence $i_1, i_2, \ldots, i_r \in \{1, \ldots, k\}$ such that

$$u_{i_1} u_{i_2} \cdots u_{i_r} = v_{i_1} v_{i_2} \cdots v_{i_r}$$

Note: Elements of $\{1, \ldots, k\}$ may repeat in the sequence $i_1, i_2, \ldots, i_r$.

Example. Consider the following PCP instance:

$$(a^2, a^2b), (b^2, ba), (ab^2, b)$$

Does this instance have a solution?

We define the language consisting of PCP instances that have a solution:

$$L_{PCP} = \{< P >| \ P \text{ is a PCP instance that has a solution } \}$$

In spite of the apparent simplicity of the definition of PCPs we have the following.

Theorem L_{PCP} is undecidable.

We will omit the proof of this result; it is presented in detail in section 5.2 of the text if you are interested. The proof consists of a reduction of A_{TM} to L_{PCP}.

Using a reduction from PCP it is possible to prove, for instance, that many questions concerning context-free languages are unsolvable. All of the following are undecidable:

- $ISE_{CFG} = \{< G_1, G_2 >| \ G_1, G_2 \text{ are CFGs and } L(G_1) \cap L(G_2) = \emptyset\}$
  (intersection emptiness of CFLs)
- $ALL_{CFG} = \{< G >| \ G \text{ is a CFG with terminal alphabet } \Sigma \text{ and } L(G) = \Sigma^*\}$
  (CFL universality)
- $REGULAR_{CFG} = \{< G >| \ G \text{ is a CFG and } L(G) \text{ is regular } \}$
  (CFL regularity)
- $EQ_{CFG} = \{< G_1, G_2 >| \ G_1, G_2 \text{ are CFGs and } L(G_1) = L(G_2)\}$
  (CFL equivalence)
- $INC_{CFG} = \{< G_1, G_2 >| \ G_1, G_2 \text{ are CFGs and } L(G_1) \subseteq L(G_2)\}$
  (CFL inclusion)

As an example we prove the undecidability of "intersection emptiness" ISE_{CFG}. Recall that it is decidable whether or not the language generated by a given context-free grammar is empty.

Theorem. ISE_{CFG} is undecidable.

Proof. We use a reduction from PCP. Assuming that we would have an algorithm to decide whether or not the intersection of two given CFLs is empty, we show that we could decide also PCP.

Choose $\Sigma = \{a, b\}$ and let I_{PCP} be an arbitrary PCP instance over Σ:

$$(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)$$

We define the following languages over the alphabet $\Sigma \cup \{c\} = \{a, b, c\}$:

- $L_{mi} = \{w_1 c w_2 c w_2^R c w_1^R \mid w_1, w_2 \in \{a, b\}^*\}$
- $L(\alpha) = \{ba^{i_k} ba^{i_{k-1}} \cdots ba^{i_1} c \alpha_{i_1} \alpha_{i_2} \cdots \alpha_{i_k} \mid k \geq 1, 1 \leq i_j \leq n, j = 1, \ldots, k\}$
- $L(\beta) = \{ba^{i_k} ba^{i_{k-1}} \cdots ba^{i_1} c \beta_{i_1} \beta_{i_2} \cdots \beta_{i_k} \mid k \geq 1, 1 \leq i_j \leq n, j = 1, \ldots, k\}$
- $L_0 = L(\alpha) c L(\beta)^R$

It is not difficult to verify that L_{mi}, $L(\alpha)$, $L(\beta)$, L_0 are all context-free. (Will be done in class.)

We observe that

$$L_0 \cap L_{mi} \neq \emptyset \text{ iff } I_{PCP} \text{ has a solution.}$$

The languages L_0 and L_{mi} are context-free. Hence if we could decide the emptiness of the intersection of CFLs, we could also decide whether or not an arbitrary PCP instance has a solution. □

The proof of undecidability of ISE_{CFG} uses a reduction from PCP. A proof using a reduction directly from some property of Turing machines would be quite difficult!

Finally, a "rule of thumb" concerning decidability questions for the main computation models:

- "All" questions are decidable for regular languages – the known counter-examples are somewhat "artificial" problems.
- All questions are undecidable for languages recognized by general Turing machines (Rice's theorem). The only exceptions are the trivial questions that have only one possible answer for all inputs. (Note that here we are referring to properties of languages recognized by TM's since syntactic properties of TM's may be decidable.)
- For context-free grammars (pushdown automata) there are both decidable and undecidable questions. The majority of questions for LBAs which are equivalent to type 1 grammars in the Chomsky hierarchy (the so called "context-sensitive" grammars) are undecidable, but some problems, like the LBA membership problem, are decidable.