The third exam is regularly scheduled as a final exam. As with the first two exams, it is open-textbook/open-notes. You will have access to the course website through a kiosk account.

This exam will be cumulative, but it will heavily emphasize material covered since the second exam. Thus while any material that's been covered in class or in the assigned reading might be included, questions on earlier topics are more likely to involve fundamental concepts like the differences between regular and context-free languages or the models of computation we've seen than they are to require you to carry out cumbersome constructions such as conversion from an NFA-ε to a DFA.

The new topics for this exam include:

1 Turing machines
   • Formal (tuple) specification and transition (state) diagrams
   • Using sequences of configurations (a.k.a., instantaneous descriptions) to show the moves of the TM
   • Relations between variants of Turing machines, e.g., with a tape infinite in one direction or two, single-tape or multi-tape, deterministic vs nondeterministic
   † Linear-bounded automata – a significantly restricted variation of Turing machines
   • Given a Turing machine's state diagram or specification, be able to describe the machine's behavior and language.
   • Design a Turing machine that recognizes a specific language and/or behaves in a specific way. Describe it using a state diagram – for very simple problems only! – or pseudo-code.
   • Definition of Turing-recognizable (RE) and Turing-decidable (R), and especially how they differ.
   † Enumerators, and their relation to Turing-recognizability
   • The universal Turing machine $U$.

† indicates an advanced topic that is less likely to appear on the exam.

2 Decidable and undecidable problems
   • Know the definitions of standard decision problems involving Turing machines, e.g. $A_{TM}$, $E_{TM}$.
   † Know the definitions of standard decision problems involving regular languages and context-free languages, e.g., $E_{DFA}$.
   • Be able to classify each of these standard problems as decidable (R), recognizable (RE), co-recognizable (co-RE), or none of these.
   • For the decidable or recognizable problems, sketch an algorithm that decides or recognizes it.
3 Proving languages undecidable (somewhat difficult parts)

- Know that a language is decidable if and only if the language and its complement are both recognizable.
- Know that decidable languages are closed under complement, but recognizable languages aren’t.
- Prove that a language is undecidable or not recognizable, using a simple proof by contradiction that relies on closure properties or similar basic facts.
- Be able to distinguish a trivial from a nontrivial property and know how Rice’s theorem applies to these properties.

4 Proving languages undecidable (more difficult parts)

- Write a simple diagonalization proof, e.g., show that the reals are uncountable, show that there are more languages than Turing machines\(^4\).
- Understand the diagonalization proof that \(A_{\text{TM}}\) is undecidable (even if you aren’t sure you fully believe in it!).
- Know how to prove that a language is undecidable using a reduction from another language already known to be undecidable.

4 See Theorem 4.18

Guide to Decidability

Determining the difference between a decidable and an undecidable problem

There’s no magic formula to determine if a problem is decidable or undecidable, but there are a few general guidelines you can follow:

1 A problem is **decidable** if, to test it, there are a finite number of solutions we have to try. If there is some finite set of conditions to check, and succeeding with one of these guarantees a “yes”, while none of them succeeding guarantees a “no”, then the problem is decidable. Note that this set of conditions could be very large, as long as it’s finite.

2 A problem is **undecidable** if the solution space is countably infinite and we can generate (enumerate) each possible solution in sequence and test it, yielding a “yes” or “no” answer for that string. In this case, the problem is recognizable but not decidable. For example, a problem asking “is there a natural number such that…” is recognizable because we can start at 0 and keep trying every number in sequence. If there is a solution, we are guaranteed to find it, but there isn’t necessarily a bound on the time it will take to find it. This procedure would never halt if no such integer existed, so we won’t know, in that case, if we have exhausted the search or not.
Here is a rule of thumb concerning decidability questions for the main computation models:

1. All questions are decidable for regular languages.

2. For context-free grammars and pushdown automata, there are both decidable and undecidable questions.

3. The majority of questions for context-sensitive languages are undecidable, but some problems, like the linear-bounded automata membership problem, are decidable.\(^5\)

4. All questions are undecidable for languages recognized by general Turing machines (Rice’s Theorem; see below for more). The only exceptions are the trivial questions that have only one possible answer for all inputs.\(^6\) To determine if a property is trivial, one strategy is to check if there is a TM that satisfies it as well as a TM that does not satisfy it. If both kinds exist, then the property is non-trivial. Otherwise it is trivial. For example, the property \(\{\langle M \rangle \mid L(M) = \emptyset\}\) is a non-trivial property because there exist both TMs with the empty language and TMs with a non-empty language.

Many decidable problems concern the structure of a TM. For example, the property stating that a TM has exactly seventeen states is syntactic because we can simply look at the machine and determine if it is true or not.

On the other hand, properties concerning the language of a TM are typically undecidable. E.g., the class of languages that are accepted by some Turing machine with exactly 17 states is undecidable by Rice’s theorem (see below).

Be careful, though – there are also syntactic properties that are undecidable, for example, the set of TMs that have a state that is never reached in any run.

**When to use diagonalization**

Diagonalization is most useful to show that a given set is uncountably infinite (i.e., non-Turing-recognizable or non-RE). The general proof technique is as follows:

1. Assume that you have a list of items \(S\), each member corresponding to an integer; i.e., there is a mapping \(\mathbb{N} \rightarrow S\).

2. Construct a grid with the rows containing the elements of your correspondence.

3. Construct a new item of your structure as an element of \(S\) by complementing the values on the major diagonal of the grid.

4. This is a contradiction since you have constructed an element in \(S\) not listed in the correspondence, therefore, the set \(S\) is not countable.

\(^5\) We’re mostly skipping context-sensitive languages and LBAs in this course, but I’m including this for reference.

\(^6\) Note that here we are referring to so-called semantic properties of the languages recognized by TMs, as opposed to syntactic properties of the TMs themselves, which are often decidable.
Undecidability and reductions

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 in class and in the book. 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 the technique called reduction.

Informally, 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 $A$).

The above means that if $A$ reduces to $B$ and $B$ is decidable, then $A$ is also decidable. On the other hand, if $A$ is known to be undecidable and $A$ reduces to $B$ then $B$ is also 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.

Reductions are presented in more detail in the previous handout, and in Sipser section 5.1, reductions from $A_{TM}$ are used to show that the following questions are undecidable:

- emptiness (i.e., is the language of a TM the empty language $\emptyset$)
- equivalence (i.e., are the languages of two TMs the same)
- regularity (i.e., is the language of a TM a regular language)

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

- context-freeness (i.e., is the language of a TM a context-free language)
- finiteness (i.e., does the TM accept only a finite number of strings)
- and infinitely many more!

By generalizing the above proof techniques, it’s possible to prove a very strong undecidability result called Rice’s theorem. Below we formulate this result and provide some discussion on it.\(^7\)

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:

\(^7\) In addition to the class slides, the statement of Rice’s theorem is given as problem 5.28 in the textbook, and the proof can be found in the solutions section.
• $M$ 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 $\langle M_i \rangle$ 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 $\langle M_1 \rangle \in P$ and a Turing machine $M_2$ such that $\langle M_2 \rangle \not\in P$.

Now Rice’s theorem can be formulated as follows:

**Rice’s Theorem.** 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 are an example of resource-bounded Turing machines.

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 – but using diagonalization we can show that such languages do exist.

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.

\[
A_{\text{LBA}} = \{ \langle M, w \rangle \mid M \text{ is an LBA that accepts string } w \} \\
E_{\text{LBA}} = \{ \langle M \rangle \mid M \text{ is an LBA and } L(M) = \emptyset \}
\]
Theorem.

1. $A_{LBA}$ is decidable.

2. $E_{LBA}$ is undecidable.

Proofs of both theorems are given on pages 222–3 of the textbook.

Acknowledgments

This handout is based on material created by Nancy Ide.