CMPSCI 250: Introduction to Computation

Lecture #39: The Halting Problem and Unsolvability
David Mix Barrington
30 April 2012

The Halting Problem and Unsolvability

- Proving Something to be Impossible
- Representing TM's By Strings
- The Universal Turing Machine
- The Barber of Seville Language
- Undecidable and Non-Recognizable Languages
- Getting More Undecidable Languages
- Turing Complete Languages

Proving Something to Be Impossible

- When a problem can be solved with a particular set of resources, we can prove this to be the case by showing how to do it.
- But what about when a problem can't be solved with those resources? We can't just show algorithms that don't work, because these don't rule out the existence of other algorithms that do.
- We have one example in this course -- if a language cannot be decided by a DFA, the Myhill-Nerode Theorem can be used to prove it. This also shows that the language has no regular expression.
- Gödel proved in 1931 that there is a true statement of number theory that can't be proved (or a false statement that can be proved). The idea is that the statement can be interpreted as "I am not provable".

Proving Limits on Turing Machines

- By the Church-Turing Thesis, if we prove that no Turing machine can decide a particular language, that means that no algorithm can decide it.
- Deciding a language means solving a general class of problems, not just a single instance.
- The basic idea is called **diagonalization**, for reasons we won't be able to go into here. Like the Gödel argument, we get a contradiction out of applying a hypothetical Turing machine *to itself*. The assumption that our target problem is decidable leads to this contradiction, so it is false and the problem is not decidable.
- To formulate this argument, we will have to say more about Turing machines that take other Turing machines as input.

The Universal Turing Machine

- We could, with some effort, formalize a scheme for representing Turing machines by strings. We would need the string to code the number of states, the number of letters in the input alphabet and in the tape alphabet, the special states, and the transition function.

- It doesn't really matter how this information is stored, as long as it's possible for an algorithm (and therefore a Turing machine) to answer questions about the states and transition function.

- Once this is done, it is possible to build a **universal Turing machine**. This machine takes two inputs, a Turing machine M and a string w over M's input alphabet. It **simulates** the action of M on w, accepting or rejecting if and only if M would accept or reject w.

- Now we have a Turing machine that acts on Turing machines.

The Barber of Seville Language

- *The Barber of Seville shaves exactly those men of Seville who do not shave themselves.* Bertrand Russell proposed this statement as a logical paradox.
- If the barber is a man of Seville who does not shave himself, the rule obligates him to shave himself. And if he is a man of Seville who *does* shave himself, the rule forbids him to shave himself.
- The only solution is that he is not from Seville, or that she is not a man.
- Define the **Barber of Seville language** to be the set of TM's that do not accept themselve -- formally, L_{BS} is the set {M: M $\notin$ L(M)} or {M: (M, M) $\notin$ L(U)} where U is the universal TM.
- A **Barber of Seville Turing machine** would be a TM M_{BS} such that $L(M_{BS}) = L_{BS}$, a TM that accepts exactly those TM's that don't accept themselves.

Undecidable and Non-Recognizable Languages

- Just as the Barber can't be a man of Seville, the machine M_{BS} cannot exist. If it did, M_{BS} either would accept M_{BS} or it wouldn't. If it does, by definition it doesn't, and if it doesn't, by definition it does.

- This tells us that the language L_{BS} is *not Turing recognizable* because it is not the language of any Turing machine. Since all decidable languages are also recognizable, L_{BS} is not decidable either.

- But note that the language L_{BS}-bar *is* recognizable. It is the union of the set of strings that don't code Turing machines at all, and the set of TM's that *do* accept themselves. We can recognize the latter set by taking any machine M and feeding the pair (M, M) to the universal TM. The former set is decidable, assuming that we have defined our coding scheme unambiguously.

- So we have an example of a language that is recognizable but not decidable.

Getting More Undecidable Languages

- Of course it would be much more interesting to have an undecidable language that we actually might have wanted to decide.

- We can do this by the **method of reduction**. Given a language X, we prove that we could decide L_{BS} *if* we had a decider for X. Then since the decider for L_{BS} cannot exist, the decider for X cannot exist either.

- For example, let L_{halt} be the set of pairs (M, w) such that M is a TM that eventually halts on the input string w. Suppose I had a decider for L_{halt}. Given any Turing machine M, I can now decide whether M is in L_{BS} by forming the pair (M, M) and feeding it to the L_{halt} decider. If the decider says that M will not halt on M, then M is in L_{BS}. If it will halt, I can then run M on M and see whether it accepts, knowing that this computation will not run forever.

- As we build up a library of undecidable languages, we can use any of them in place of L_{BS} in this kind of argument.

Turing Complete Languages

- In CMPSCI 311 and 401, you will spend a lot of time with the concept of **complete** languages for a class.

- L_{halt} turns out to be **Turing complete**, or complete for the set of recognizable languages. We can take any recognizable language X, and any string w, and transform w into a string f(w) such that w is in X if and only if f(w) is in L_{halt}.

- This means that the language L_{halt} captures every possible Turing recognizable computation.

- The class **NP** or **nondeterministic polynomial time** is the set of languages that are recognized by nondeterministic Turing machines in polynomial time. If we prove a language to be **NP-complete** by showing that any NP language can be reduced to it, we are pretty sure that it is not actually decidable in polynomial time by a deterministic TM. This is because if it were the classes P and NP would be the same, and we are pretty sure that they are not.