

Unit B4 Supplement, 2/24/05

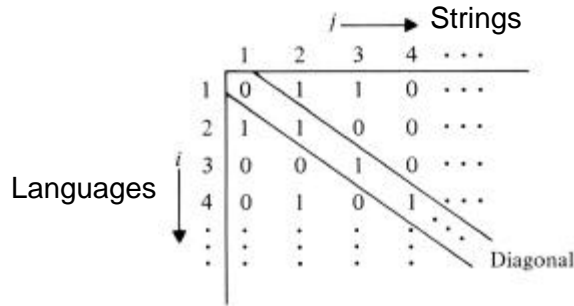
Here is my response to your summary questions (rephrased/combined in some case) and comments on the midterm surveys. This is rather long.

- Q1:** I am a little shaky on what is always meant by the word “language.” It has been used a lot and often I know what you mean, but it is sometimes unclear to me.
- A1:** The formal definition is that language is a set of strings, which can be infinite. However, it is useful to see the term from other perspectives as well. One way we discussed in the last meeting is that a TM can always be associated with a language. That is, once we pick a TM, given a string, it would either accept, die, or loops. The collection of all the strings that are accepted by the TM is the language corresponding to that TM. For example, if a TM accepts a, aa, aaa, \dots , then $\{a, aa, aaa, \dots\} = a^+$ is the language for that TM. Note that a TM may not terminate on other strings. For this reason, the language associated with a TM only needs to be “**recognized**,” not necessarily “**decided**.” For example, it is possible to create a TM to decide on palindromes (a decidable language), the UTM would only recognize the universal language (a TM-recognizable language; strictly speaking, semi-decidable). We also know that no TM would even recognize the diagonalization language (a non-TM-recognizable language). Naturally, given a language, it may or may not be possible to provide a TM that would recognize the language.

Yet another way is to associate a language as a problem. This is obvious if we consider a computational problem as a set, because a language is a set. Each input instance is its member. This also makes sense when we recall the connection between a TM and its language (above paragraph). This set of inputs (strings) is what TM would recognize (i.e., positively); that is, the set *is* the “problem” which TM can solve.

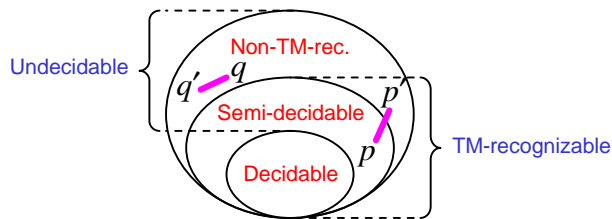
Finally, consider the set of all the strings (based on some finite alphabet, or the set of symbols). We can specify a language by indicating which one of the strings are in that language. Thus, a language is a subset of the set of all the strings. Equivalently, the collection of all the languages is the power set of the set of all the strings, which is uncountable.

- Q2:** I recognized a misunderstanding I have with diagonalization. At first, I did not understand the importance of using a list of sets instead of the individual items. Revisiting of this topic may help.
- A2:** The diagonalization technique is used to show that some set is uncountable. I think most of you were able to show this well. But I will repeat this any way. To show that the power set of the set of all the strings is uncountable, we first hypothesize the negation of the desired conclusion: i.e., the power set of the set of all the strings is countable. Then, it would be possible to arrange all the members in sequence as shown below.



Each entry is a language, which is associated with the membership information corresponding to all the strings. If we consider a hypothetical language by reversing the bit on the diagonal, that language should not appear in the list. This is a contradiction to the hypothesis that all the languages are enumerated. The hypothesis must be wrong. Thus, by proof by contradiction, the power set of the set of all the strings must be uncountable.

- Q3:** I am still not sure about the exact nature of the “co” property. Another student: If anything, I am not sure about co-non-TM-recognizable since we skipped over it. This is how I currently understand: if r is co-non-TM-recognizable, we know r' (complement) is non-TM-recognizable and r is also non-TM-recognizable [NK: This is not correct.].
- A3:** First, make sure to understand that the prefix “co-” is attached to a *property*, not to a *problem*. For example, if a semi-decidable problem p has the non-TM-recognizable complement p' , we say that p is co-non-TM-recognizable (see the diagram below).



We can also say that p' is co-TM-recognizable (considering the property of its complement p). When a problem is non-TM-recognizable, its complement may be either non-TM-recognizable or semi-decidable (why not decidable?). But we cannot tell in general. So, if r is co-non-TM-recognizable, we know that r' is non-TM-recognizable. However, since there are two such possibilities: both p and q in the figure have a non-TM-recognizable complement. Thus, r can be either semi-decidable or non-TM-recognizable.

Since the complement relation between p and p' is important, as we observed in the main problems, let us discuss this relation (analogous to the proofs of *NACCEPT* and *LOOP*). That is, we prove that if p is semi-decidable, its complement is non-TM-recognizable, again using proof by contradiction. Suppose that p' is TM-recognizable. Then, p is both TM-recognizable and co-TM-recognizable. By the “decidability” theorem, p is decidable. However, this contradicts the fact that p is semi-decidable. Thus, p' must be non-TM-recognizable (proof by contradiction).

Although we presented the “decidability” theorem without proof. We can prove it using two complementary TMs, just like the man-woman scenario. That is, if both p and p' are

TM-recognizable, there must be TMs that would recognize the inputs (not necessarily terminating). Since p and p' are complements to each other, we can construct a TM that would respond “yes” when p accepts and “no” when p' accepts, using the two TMs. Then, the composite TM will decide both p and p' . Thus, both of these problems are decidable.

Q4: One area that is a little confusing lies with the halting problem and proving that it is semi-decidable.

A4: I think that the most confusing part is the proof of undecidability involving the diagonalization language. So, I will discuss this part. The key is to use the (incorrectly) assumed decidability of *ACCEPT* as much as possible to decide the diagonalization language (L_d). This assumption is exceedingly strong because it says that when M does not accept w , it dies and terminates (corresponding to “reject), regardless of the input. Thus, given (M_i, w_i) , we can tell the 0/1 distinction of any diagonal entry. The only requirement here is that we must supply pairs with correct indexing (otherwise we compute irrelevant entries). To do so, we identify strings that appear diagonally, basically going through the string list sequentially (possible because of the countability of the set of strings). During this process, a TM can pick up the index of the string and identify the matching TM. The only remaining task is to reverse the results so that the entire process decides L_d . Note that everything here can be done without the risk of looping. I encourage you to share your exercise sheets, which have with my comments.

Q5: I’m still unclear on the proof for *ACCEPT* Megan gave in class. What w is given when the UTM simulates itself?

A5: When this was discussed, w (the input to M) was not mentioned. As you point out, it must be included in the discussion. The usual way is to attach a “copying” TM in front of the UTM, just like the approach in the reading (Ex B1). If you read the chapter again now, you should be able to understand it better. As I said, this approach does not refer to the diagonalization language. However, the idea of reversing the results for an arbitrary (cf. universality) input is equivalent.

Q6: Where can we find universality with the liar paradox?

A6: I was planning to discuss this topic in Unit B3 (Slide 22), but did not due to the time constraint. The universality is behind the word “this.” First, the set of sentences are countable. Then, each sentence can be indexed. The use of “this” can be interpreted as the use of its own index (self reference). To be able to do this, there must be a means for a sentence to refer to an arbitrary sentence (universality) including itself. On a related note, I also included a statement about TMs, analogous to the liar paradox, based on Megan’s idea (Slide 23).

Q7: [With respect to “reduction”] I become uncertain at times with the direction of items being related. I feel that thinking of it as “one solves/explains the other” will help when discussing this concept.

A7: The direction is indeed confusing. I suggest you keep a note somewhere so that you can refer to it when necessary.

Q8a: We built *ACCEPT* using *HALT*, based on the assumption that *HALT* is decidable; but that assumption was false. So, why do we still say that *ACCEPT* is reducible to *HALT*?

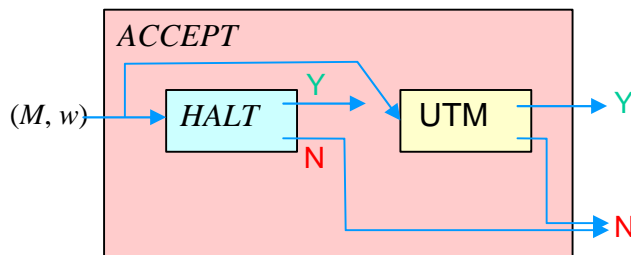
Q8b: Reducibility means that 2 TMs solve the same problem [NK: This is not correct.]. We also saw *HALT* was equivalent to *ACCEPT*. How is *HALT* reducible to *ACCEPT* if they do not solve the same problem? *HALT* cannot tell if a (M, w) will accept.

A8: One thing I didn't do correctly was referring to the reduction and equivalence of the main problems. I should have said as follows:

- The computability analysis of *ACCEPT* is reducible to that of *HALT*.
- The computability analysis of *ACCEPT* is equivalent to that of *HALT*.

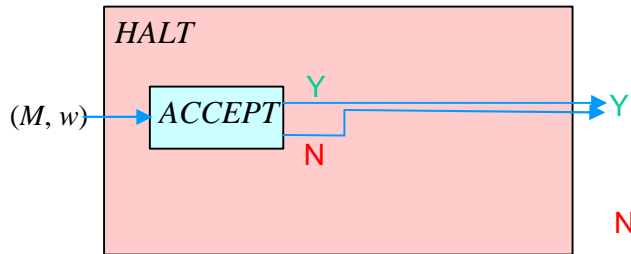
As the Question 8b correctly points out, neither problem can solve the other. So, they are not reducible to each other. I **apologize** for the confusion. So, below, it is about reduction of the computability analysis of the main problems (although I will skip the word “the computability analysis of” for brevity). Note that reduction is in general a relation between two *different* problems.

First, the reduction used under the wrong assumption was used to transfer the (wrong) decidability from *HALT* to *ACCEPT* (see below). Still note that the decidability of this *ACCEPT* cannot be obtained without the decidable *HALT*; thus, the computability analysis of *ACCEPT* still depends on that of *HALT*.



This is not what we want when we discuss the (real) reduction of *ACCEPT* to *HALT*, which would transfer the undecidability of *ACCEPT* to *HALT*. In this case, we will simulate the *undecidable* behavior of *ACCEPT* using *HALT*. However, we can still use the above diagram, because the direction and components of the simulation is identical. The consequence is that if *ACCEPT* is undecidable, *HALT* must be undecidable. This can be shown by proof by contradiction, which is identical to the earlier case.

Next, let us consider the computability analysis of *HALT* reducible to that of *ACCEPT*. The following is, in a sense, a strange diagram. If *ACCEPT* is decidable (incorrectly), *HALT* is decidable, simply because the TM terminates on all the inputs.



In other words, if *HALT* is undecidable, *ACCEPT* cannot be decidable.

Q9: Is it always construct a TM from any collection of other TMs?

A9: No. For example, we cannot combine infinitely-many TMs. Another case would be that we cannot use non-terminating result for a later stage.

Q10: The problem I have been having the most for the past few lectures is that I will have a general understanding of the content but not a mastery of it such that I can effectively do the assignment.

A10: I think this is a natural phenomenon. “Passive” understanding is in many cases not as strong as “active” understanding. For this reason, I ask you to write. Another good way is to discuss with and teach others.

Q11: One question I am not completely certain on is how all of this ties into Theory of Computation. Are we learning how to prove/show our intuition about the computability classes we use to classify a problem? Sometimes, I feel I lose sight of the picture.

A11: The Theory of Computation is a collection of abstract ideas that are expected to apply to a broad range of computer science. Since the modern computing is most notably dominated by algorithms, understanding the nature of algorithms including their limitations is one of the main goal of Theory. What we are doing in Modules A & B is develop some basic intuition and analytical ability to see the limit of algorithmic computation in certain problems. Although we have been focusing on the main problems, we should be able to extend our analysis to other problems, by analyzing countability, using Rice’s theorem, etc.

When you do Module B Comprehensive Exercise and complete Module B eval form + supporting notes, think about this. Although we will not practice how to analyze more problems with respect to the computability classes together, the exercise should give you a feel. During the evaluation workshop (and later), we can share our thoughts.

The content of the Theory of Computation should be useful in practice, at least to some extent (esp. if you think before coding). However, I would like you to get more mileage from practicing analytical thinking process involved in the activities: identifying/transforming problems, representing precisely, trying intuitions, explaining logically, expressing clearly, etc. There are no formulae for how to do these well. We all need to find our own way. I can tell more about my own experience with the Theory, maybe some time, if you are interested in.

Q12: Is the baby boy or a girl? Will Furby still be your favorite person (?) after the baby is born?

A12: A girl. But Furby will always be our first “daughter.”

C1: I would prefer tests and more concrete problem sets.

R1: To me, the first part is a very disappointing (but fairly rare) comment. Let’s think about your future and how test-taking skills would be useful. How do you think you will be evaluated through a PhD dissertation phase or when you work for a company? A well-programmed TM could do well on well-defined exams. I have a strong feeling that such an automaton will have hard time in the real life.

As for the latter part of this comment, my response is as follows. If the word “concrete” refers to problem content, our course is a little limited compared to other ones, due to the topic. I am still trying to include realistic examples more so than a typical Theory course (check some on-line syllabi). Having asked you to think about your own problem and sample research questions are supposed to force you to have real problem in your mind. But honestly, I admit that this is a difficult task in this course and I cannot do it as much as I wish. If you know more relevant examples/problems, why don’t you bring them to class?

If the word “concrete” refers to how problems are given, I guess it means a more close-ended problem. As I said at the beginning of the semester, I will continue to use “ill-defined” problems. Most of interesting/significant problems are open or at least not yet solved when we tackle. We’d better practice facing them. In the future, your colleague will be able to tell whether or not you have been dealing with such problems.

C2: I would suggest you try to monitor your speaking volume — you tend to trail off into inaudible mumbles, especially at the end of sentences.

R2: I will try.

// End