CS 154 - Introduction to Automata and Complexity Theory

Spring Quarter, 2000

Handout 31 (5/17/00) – Supplementary Lecture Notes

1 Recursive versus recursively enumerable

First, let us clarify the precise relationship between the classes of languages *recursively enumerable* and *recursive*. The class *recursively enumerable* consists of all languages for which there exists a Turing machine (or a Procedure). The class *recursive* consists of all languages for which there exists a special type of Turing machine (a Halting Turing machine or an Algorithm) which can only reject by halting, unlike Turing machine or Procedures which are allowed to reject by not halting at all. Quite clearly, every *recursive* language is also *recursively enumerable*. Note that we sometimes also refer to a *recursive* language as being *decidable*, and to a non-*recursive* language as being *undecidable*.

Given the closure properties for these two classes of languages, we can infer that for any language L exactly one of the following three possibilities apply:

1. both L and $\overline{L}$ are *recursive*,
2. neither L nor $\overline{L}$ is *recursively enumerable*, or
3. one of L and $\overline{L}$ is *recursively enumerable* but not *recursive*, while the other is not *recursively enumerable*.

This may be represented as the following 3×3 table of possibilities.

	$\overline{L}$ *recursive*	$\overline{L}$ *recursively enumerable* (but not recursive)	$\overline{L}$ not *recursively enumerable*
L *recursive*	$\surd$	X	X
L *recursively enumerable* (but not *recursive*)	X	X	$\surd$
L not *recursively enumerable*	X	$\surd$	$\surd$

An important point is that both in the middle row and the middle column, we are talking about languages that are "recursively enumerable but not recursive." The notation $\surd$ indicates that the combination of properties for L and $\overline{L}$ is possible, while X denotes that the combination is impossible

2 Undecidability of Emptiness

Consider the language:

$$L_\emptyset = \{<M>|\ L(M) = \emptyset\}.$$

We would like to classify $L_\emptyset$ and its complement $\overline{L}_\emptyset$ with respect to the classes *recursive* and *recursively enumerable*. This is covered in greater detail in Example 8.2 on page 185 of the textbook.

Theorem 1. *$\overline{L}_\emptyset$ is recursively enumerable.*

Why is this? Note that a string w belongs to $\overline{L}_\emptyset$ if it is the case that $w =< M >$ for some Turing machine M such that $L(M)$ is non-empty. To establish Fact 1, we have to demonstrate the existence of a Turing machine P (a mere procedure) with the following property: when given $<M>$ as input, P will accept if $L(M)$ is non-empty and reject otherwise (possibly by not halting). That is, we have to find a Turing machine P for the language $\overline{L}_\emptyset$. This is easily achieved via non-determinism. The Turing machine P first guesses a string $x \in L(M)$ and then executes M on input x to verify that M accepts x. Clearly, if $L(M)$ is non-empty then there exists a valid choice of x that will cause P to accept $<M>$; otherwise, P will reject $<M>$.

Theorem 2. *$L_\emptyset$ is* **not** *recursively enumerable.*

For this, it suffices to show that $L_\emptyset$ is not *recursive*. This is because by Theorem 1 we know that $\overline{L}_\emptyset$ is *recursively enumerable*, which leaves open only two possibilities:

1. both $L_\emptyset$ and $\overline{L}_\emptyset$ are *recursive*, or
2. $\overline{L}_\emptyset$ is *recursively enumerable* but not *recursive*, while $L_\emptyset$ is not *recursively enumerable*.

If we are able to show that $L_\emptyset$ is not *recursive*, then we rule out the first possibility and are left with only the second one, thereby establishing Theorem 2.

We now turn to the task of showing that $L_\emptyset$ is not *recursive*. That is, we will prove the following lemma:

Lemma. *$L_\emptyset$ is non-recursive or undecidable.*

Proof: Our proof will involve establishing a reduction from $\overline{L}_U$ to $L_\emptyset$. Since $\overline{L}_U$, the complement of the universal language L_U, is already known to be non-recursive, the existence of such an reduction will immediately imply the lemma.

Consider the two languages involved in the reduction:

$$\overline{L}_U = \{<M, w>|\ w \notin L(M)\}$$

$$L_\emptyset = \{<M'>|\ L(M') = \emptyset\}$$

Our reduction f will take as input an instance of the first language, i.e., the encodings of a Turing machine M and its input w. It will then produce as output an instance of the second language, i.e., the encoding of another Turing machine M'. In other words,

$$f(<M, w>) =<M'> .$$

The reduction f will ensure that M' when executed on an input string w' will behave as follows:

1. First, M' ignores the input w' and instead merely simulates M on w.
2. If M accepts w, then M' will accept its own input w' without even looking at it.
3. If M rejects w, then M' will reject its own input w' without even looking at it. Note that M may reject w by not halting at all, which is fine since then M' will not halt and will thus reject w' as required.

To establish the correctness of the reduction, we have to demonstrate the following two properties.

a). First, we have to show that the function $f(<M, w>) =<M'>$ can be computed by a halting Turing machine. That is, there exists an *algorithm* which when started off with $<M, w>$ on its tape, will halt within a finite number of steps ensuring that its tape now contains $<M'>$. Observe that M' is really quite a simple modification of M — basically, M' first blanks out its tape (dispensing with its own input w'), then it writes w on the tape, and finally it starts behaving like M. Modifying M into M' is a trivial task for an algorithm.

b). We also need to show that $<M, w>$ belongs to $\overline{L}_U$ if and only if $<M'>$ belongs to $L_\emptyset$. This may be reasoned as follows.

Consider first the case where $<M, w>$ belongs to $\overline{L}_U$. In this case, $w \notin L(M)$ and so M' rejects any input w' written on its tape. That is, $L(M') = \emptyset$ or $<M'> \in L_\emptyset$.

Consider now the case where $<M, w>$ does not belong to $\overline{L}_U$. In this case, $w \in L(M)$ and so M' accepts any input w' written on its tape. That is, $L(M') = \Sigma^*$ or $<M'> \notin L_\emptyset$.

This completes the proof of the lemma and hence Theorem 2.