CS 6743 **Lecture 3**[1] **Fall 2007**

1 Complexity Classes

Unless explicitly stated, by a TM we will mean a *multi-tape* TM. When we talk about the space used by a multi-tape TM, we only count the number of cells on the worktape(s) that are touched by the TM during its computation. That is, we *do not* count the input tape or the output tape.

Below n will denote the input size.

For some function $f : \mathbb{N} \to \mathbb{N}$,

- $\mathsf{Time}(f(n)) = \{L \mid L$ is decided by some TM in at most $f(n)$ steps$\}$
- $\mathsf{Space}(f(n)) = \{L \mid L$ is decided by some TM that touches at most $f(n)$ cells of its worktapes$\}$

For example,

- $\mathsf{P} = \cup_{k\geq 1}\mathsf{Time}(n^k)$,
- $\mathsf{EXP} = \cup_{k\geq 1}\mathsf{Time}(2^{n^k})$,
- $\mathsf{L} = \mathsf{Space}(\log n)$,
- $\mathsf{PSPACE} = \cup_{k\geq 1}\mathsf{Space}(n^k)$.

In general, a complexity class is a collection of languages decidable within a given amount of computational resources.

The relationship between these classes is as follows:

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

We will show in the next lecture how to simulate space by time and vice versa. Note that although all these inclusions are not strict (e.g., it is not impossible that $P = L$ or that $P = PSPACE$, albeit unlikely), we can show that $\mathsf{L} \subsetneq \mathsf{PSPACE}$ and $\mathsf{P} \subsetneq \mathsf{EXP}$. The general form of a result like this is called the hierarchy theorems (for space and time, in this case).

2 Hierarchy Theorems

First we recall a basic result from Computability Theory that there are undecidable languages. This can be argued as follows. There is a 1-1 correspondence between algorithms (TMs) and natural numbers. There is a 1-1 correspondence between languages (over the binary alphabet) and real numbers (in the interval $[0, 1]$). Since the set of reals is bigger

[1] This lecture is a modification of notes by Valentine Kabanets

than the set of natural numbers (as argued by Cantor a couple of hundred years ago), we have the existence of an undecidable language.

The problem with the argument above is its non-constructiveness. We haven't exhibited a particular example of a language that is not decidable. Turing was the first to give such examples. The most famous is his Halting Problem.

Recall $Halt = \{(M, x) \mid \text{TM } M \text{ halts on } x\}$. (This is a version of the Halting Problem. A different definition considers $Halt = \{M \mid \text{TM } M \text{ halts on blank tape}\}$)

Theorem 1. *Halt is undecidable.*

We will prove the theorem in two stages. First, we define a certain language *Diag* and show that *Diag* is undecidable. Then we will show that *Diag* is reducible to *Halt*, and hence, *Halt* is also undecidable. (Note the use of a reduction to prove a lower bound on the complexity of *Halt*.)

To define *Diag*, we consider an enumeration $x_1, x_2, x_3, \dots$ of all binary strings. Let $M_1, M_2, M_3, \dots$ be an enumeration of all Turing machines whose descriptions are $x_1, x_2, x_3, \dots$. (Note some binary strings correspond to "broken" TMs. We will assume that a broken TM does not accept any string.) We define *Diag* as follows:

$$Diag = \{M \mid M \text{ does not accept input } M\}.$$

Lemma 2 (Turing)**.** *Diag is undecidable.*

Proof. Our proof is by contradiction. Suppose some TM D decides *Diag*. Then we have one of two cases:

1. D accepts D, OR
2. D does not accept D.

In the first case, we get $D \notin Diag$ (by definition of *Diag*). Since D decides *Diag* (by our assumption), D does not accept D. A contradiction.

In the second case, we get $D \in Diag$ (by definition of *Diag*). Since D decides *Diag* (by our assumption), D accepts D. Again a contradiction.

Thus, our assumption that *Diag* is decidable must be wrong. □

Proof of Theorem 1. Now we reduce *Diag* to *Halt*. If *Halt* were decidable by some TM H, then we could decide *Diag* as follows:

> "On input M, run the TM H on (M, M), and negate the output (i.e., Accept if H rejects, and Reject if H accepts)."

It is easy to see that the described algorithm decides *Diag*. But, by our Lemma 2, *Diag* is undecidable. Hence, so is *Halt*. □

The proof of Theorem 1 uses two very important techniques:

- simulation, and
- diagonalization.

The same ideas are used to prove *Hierarchy Theorems*: with more time (space), one can compute more languages.

In this course, we will always use *proper* complexity functions $f(n)$. A function $f(n)$ is called *proper* if there is a TM M that, on input 1^n, outputs exactly $f(n)$ symbols and runs in time $O(f(n)+n)$ and space $O(f(n))$. The usual functions like $\log n$, $\sqrt{n}$, n^2, 2^n, $n!$ are proper. Also, if f and g are proper, then so are $f+g$, fg, $f(g)$, f^g, 2^g.

Lemma 3. *Let $f(n)$ be any proper complexity function. The language*

$$Halt_f = \{(M, x) \mid \text{ TM } M \text{ accepts } x \text{ in at most } f(|x|) \text{ steps}\}$$

is decidable in time $g(n) = (f(n))^3$.

Proof. We can construct a universal TM H with 3 tapes that does the following. Given an input (M, x), our TM H converts a (possibly multi-tape) TM M to an equivalent one-tape TM M'. If M accepts x in at most $f(|x|)$ steps, then the new TM M' will accept x in at most $(f(|x|))^2$ steps.

Next, H will simulate M' on x for at most $(f(|x|))^2$ steps. Each step of M' can be simulated by H in at most $|M'|$ steps (i.e., the size of the description of the TM M', which is some constant dependent on M'); this constant is at most $f(|x|)$. Thus, our entire simulation will take at most $(f(|x|))^3$ steps. (Note that we needed $f(n)$ to be a proper function in order to be able to simulate M for at most $f(n)$ steps!) □

Consider the language

$$Diag_f = \{M \mid \text{ TM } M \text{ does not accept input } M \text{ in at most } f(|M|) \text{ steps}\}$$

By Lemma 3, the language $Diag_f$ is in $\mathsf{Time}(g(2n))$.

Theorem 4. *$Diag_f$ is not in $\mathsf{Time}(f(n))$.*

Proof. The proof is virtually the same as the one showing that the language *Diag* (defined earlier) is undecidable. The details are left as an exercise. □

Hence, we have

Theorem 5 (Time Hierarchy)**.** *For every proper complexity function $f(n) \geq n$,*

$$\mathsf{Time}(f(n)) \subsetneq \mathsf{Time}((f(2n))^3).$$

Note that the version in Sipser's book is a stronger result:

$$\mathsf{Time}(f(n)/\log n) \subsetneq \mathsf{Time}(f(n))$$

Similarly, we can prove

Theorem 6 (Space Hierarchy)**.** *For every proper complexity function $f(n) \geq \log n$,*

$$\mathsf{Space}(f(n)) \subsetneq \mathsf{Space}((f(n)) \log f(n)).$$

Again, the version in the book is stronger: it states that for any proper $f(n)$ there exists a language A decidable in space $O(f(n))$ but not in space $o(f(n))$.

As a simple application of these Hierarchy Theorems, we can prove that $\mathsf{P} \subsetneq \mathsf{EXP}$ and $\mathsf{L} \subsetneq \mathsf{PSPACE}$.

3 Reductions and completeness

Recall the definitions of reducibilities from the last lecture. In this lecture, we will give some examples of many-one reductions $\leq_m$.

In the previous section we talked about several different variants of the halting problem, in particular variant called A_{TM} in the book, $A_{TM} = \{M, x \mid \text{TM } M \text{ accepts input } x\}$ and a variant with halting on blank tape, $HaltB = \{M \mid$ TM M halts when started on blank tape. Now we will show that these problems are essentially equivalent by showing that they can be reduced to each other.

Lemma 7. *$HaltB \leq_m A_{TM}$.*

Proof. We need to present a computable function f which takes instances of $HaltB$ and maps them to instances of A_{TM}, preserving the membership in the respective languages. That is, we need $f(< M >) = (< M' >, x)$ such that $M \in HaltB$ iff $(< M' >, x) \in A_{TM}$. (recall that $< M >$ is a string encoding the description of a Turing machine; we need the descriptions since languages are sets of strings). Consider the following M':

```
M’: simulate M starting with blank tape
    if M halted, accept
```

Technically M' is the same as M, except the rejecting state of M is an accepting state of M'. Now, set $x = \lambda$ (empty string).

Suppose that $M \in HaltB$. Then M' accepts when starting on blank tape, because M halts when starting on blank and both halting states of M are accepting states of M'. Now suppose that $M \notin HaltB$. Then the simulation of M will run forever, so M' will never have a chance to halt and accept.

□

The reduction in other direction is a little bit more involved.

Lemma 8. *$A_{TM} \leq_m HaltB$.*

Proof. We construct a function f such that $f(< M >, x) =< M' >$. Note that in this case x becomes part of the description of M'. Define M' as follows:

```
M': write x on the tape
    simulate M on x
    if M rejects, go into infinite loop
```

The first step can be done by using $|x|$ new states, so x becomes encoded as a part of transition table. To go into infinite loop, two states should be enough: one moves left, another right, and they switch between each other independently of the tape contents. □

Note that the relation $\leq_m$ (as well as $\leq_p$ and most other) is transitive. That is, if $L_1 \leq L_2$ and $L_2 \leq L_3$, then $L_1 \leq L_3$. The usefulness of the following definition is based on this fact.

Definition 9. *For a complexity class C, we say that a language L is C-*complete *if*

1. *$L \in C$, and*
2. *every language in C is reducible to L.*

Interpretation A C-complete language L is a "hardest" language in C (everybody else in C is at most as hard as L.)

Usefulness A C-complete language *captures* the complexity of the entire class C. So, we can reason about C by thinking about a single concrete problem from C.

We say that a language L is *C-hard* if every language in C is reducible to L (i.e., L may or may not be in C).