Lecture 7

September 9, 2005

1 Complexity Theory

1.1 Resource Consumption

Complexity theory, or more precisely, Computational Complexity theory, deals with the resources required during some computation to solve a given problem.

The process of computing involves the consumption of different resources like time taken to perform the computation, amount of memory used, power consumed by the system performing the computation, etc. A theory of resource consumption looks at how much resources are needed to solve any particular computational problem. In order to consider such a theory, we first have to decide what the resources are.

In this class the resources that we consider will be time and space (in terms of memory). What does it mean to measure 'time' and 'space'? One possibility is that we run algorithm A on machine I and measure the time t in seconds that the program executes for and space k in KB as the amount of memory used. However, this method has the drawback of being machine-dependent. Thus, the same algorithm run on two different machines may give us different results for time and space consumption.

To overcome this limitation, we should use an abstract model of a computer. The model we use is the *Turing machine*. Time is measured in terms of the number of the steps taken by the Turing machine, and space is measured in terms of the number of cells used. In reality, it is tedious to construct the equivalent Turing machine for a given algorithm. Instead, we perform a rough analysis for the algorithm itself, under the assumption that calculations involving fixed input sizes take constant time and space.

The performance of an algorithm needs to be measured over all inputs. We club all inputs of a given sized together and analyze performance for all inputs of a given size. For an input of size n, let t_n be the time consumed and s_n be the space consumed. Typically, we want to find functions t and s, such that $t(n) = t_n$ and $s(n) = s_n$.

One might wonder why don't simply use benchmarks. The two major problems with using benchmarking as the basis of our performance analysis are:

- Benchmarks do not convey any information about the scalability of the problem.

- Algorithms may be optimized specifically for the set of benchmarks. We will not know whether our performance analysis holds outside that set. This is called *design for benchmarking*.

1.2 Scalability and Complexity classes

Scalability deals with the following issue: How much more resources (space and time) do I need when my problem size increases?

So the scalability of the problem is the issue of the growth rate of the time and space requirements with increase in the input size. In this class we are not interested in a very sophisticated performance analysis. For any problem, we will be interested in knowing whether it falls into one of three broad categories: slow, moderate and fast growth, which we define as follows:

- Slow — Logarithmic
- Moderate — Polynomial
- Fast — Exponential

Thus, we can define categories of problems according to their scalability. This is summarized in table 1.

Table 1: **Complexity Classes and Scalability**

	Slow	Moderate	Fast
Time	LOGTIME	PTIME	EXPTIME
Space	LOGSPACE	PSPACE	EXPSPACE

Informally, the classes can be explained as follows:

- PTIME — Problems that can be solved in polynomial amount of time.
- EXPTIME — Problems that can be solved in exponential amount of time.
- LOGSPACE — Problems that can be solved in logarithmic amount of space.
- PSPACE — Problems that can be solved in polynomial amount of space.
- EXPSPACE — Problems that can be solved in exponential amount of space.

Defining LOGTIME is problematic since reading an input of length n takes time n. Thus this class is machine dependent. In this course we will ignore this class and focus on the other classes when we talk about complexity. The categories above are called *complexity classes*.

1.3 Relationships between the classes

By definition,

$$\text{PTIME} \subseteq \text{EXPTIME}$$

$$\text{LOGSPACE} \subseteq \text{PSPACE} \subseteq \text{EXPSPACE}$$

It is easy to see that

$$\text{PTIME} \subseteq \text{PSPACE}$$

and

$$\text{EXPTIME} \subseteq \text{EXPSPACE}$$

This is because consuming space takes time, so any computation that takes polynomial space must take at least polynomial time. Therefore any computation that takes at most polynomial time cannot take more than polynomial space.

Next we consider LOGSPACE. If a problem is in LOGSPACE, it means that in solving an instance of the problem at most $c\log n$ cells will be used, where c is a positive constant and n is the size of input. Let the alphabet of the machine have size d. (Real-world computers have an alphabet of size 2 consisting of the binary digits 0 and 1.) Then the number of possible configurations of the memory is $d^{c\log n} = n^{c\log d}$, which is polynomial in n. So we can run through all possible configurations of the memory in polynomial time. Since a solution to the problem must be one of those possible configurations, we can compute it in polynomial time. Thus we have

$$\text{LOGSPACE} \subseteq \text{PTIME}$$

and similarly,

$$\text{PSPACE} \subseteq \text{EXPTIME}$$

Putting together all the relationships between the classes, we get,

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$$

An immediate question that arises is whether the containments above are strict. It was discovered in the 1960's that an exponential gap matters. So,

$$\text{LOGSPACE} \subsetneq \text{PSPACE} \subsetneq \text{EXPSPACE}$$

$$\text{PTIME} \subsetneq \text{EXPTIME}$$

So we know that the long chain of containments has gaps. The question is where exactly the gaps lie. This has been the central question of complexity theory for the last 40 years and it is still unresolved.

2 Truth Evaluation

Truth evaluation is concerned with the following question:
Given $\phi \in Form$ and $\tau \in 2^{AP(\phi)}$, does $\tau \models \phi$?

To answer the question above, we can equivalently evaluate $\phi(\tau)$. We can write a simple recursive algorithm to evaluate $\phi(\tau)$.

$$
\begin{array}{l}
\phi(\tau) = \\
\qquad \text{case} \\
\qquad\qquad \begin{array}{lcl}
\phi \text{ is } p \in Prop & : & \phi(\tau) = \tau(p) \\
\phi \text{ is } (\neg\theta) & : & \phi(\tau) = \neg(\theta(\tau)) \\
\phi \text{ is } (\theta \circ \psi) & : & \phi(\tau) = \circ(\theta(\tau), \psi(\tau))
\end{array} \\
\qquad \text{esac}
\end{array}
$$

Theorem: Truth evaluation is in PTIME and LOGSPACE.