Topics in Logic and Complexity
Handout 1

Anuj Dawar

MPhil Advanced Computer Science, Lent 2010

What is This Course About?

Complexity Theory is the study of what makes some algorithmic problems inherently difficult to solve.

> Difficult in the sense that there is no *efficient* algorithm.

Mathematical Logic is the study of formal mathematical reasoning.

> It gives a *mathematical* account of meta-mathematical notions such as *structure*, *language* and *proof*.

In this course we will see how logic can be used to study complexity theory. In particular, we will look at how complexity relates to *definability*.

Computational Complexity

Complexity is usually defined in terms of *running time* or *space* asymptotically required by an algorithm. *E.g.*

- Merge Sort runs in time $O(n \log n)$.
- Any sorting algorithm that can sort an arbitrary list of n numbers requires time $\Omega(n \log n)$.

Complexity theory is concerned with the hardness of *problems* rather than specific algorithms.

> We will mostly be concerned with *broad* classification of complexity: *logarithmic* vs. *polynomial* vs. *exponential*.

Graph Properties

For simplicity, we often focus on *decision problems*.

As an example, consider the following three decision problems on *graphs*.

1. Given a graph $G = (V, E)$ does it contain a *triangle*?
2. Given a directed graph $G = (V, E)$ and two of its vertices $s, t \in V$, does G contain a *path* from s to t?
3. Given a graph $G = (V, E)$ is it *3-colourable*? That is,

   is there a function $\chi : V \to \{1, 2, 3\}$ so that whenever $(u, v) \in E$, $\chi(u) \neq \chi(v)$.

Graph Properties

1. Checking if G contains a triangle can be solved in *polynomial time* and *logarithmic space*.

2. Checking if G contains a path from s to t can be done in *polynomial time*.

Can it be done in *logarithmic space*?

Unlikely. It is NL-complete.

3. Checking if G is 3-colourable can be done in in *exponential time* and *polynomial space*.

Can it be done in *polynomial time*?

Unlikely. It is NP-complete.

Logical Definability

In what kind of formal language can these decision problems be *specified* or *defined*?

The graph $G = (V, E)$ contains a triangle.

$$\exists x \in V\, \exists y \in V\, \exists z \in V (x \neq y \wedge y \neq z \wedge x \neq z \wedge E(x,y) \wedge E(x,z) \wedge E(y,z))$$

The other two properties are *provably* not definable with only first-order quantification over vertices.

Second-Order Quantifiers

3-Colourability and *s-t-path* can be defined with quantification over *sets of vertices*.

$$\begin{aligned}
&\exists R \subseteq V\, \exists B \subseteq V\, \exists G \subseteq V \\
&\qquad \forall x (Rx \vee Bx \vee Gx) \wedge \\
&\qquad \forall x (\neg(Rx \wedge Bx) \wedge \neg(Bx \wedge Gx) \wedge \neg(Rx \wedge Gx)) \wedge \\
&\qquad \forall x \forall y (Exy \rightarrow (\neg(Rx \wedge Ry) \wedge \\
&\qquad\qquad \neg(Bx \wedge By) \wedge \\
&\qquad\qquad \neg(Gx \wedge Gy)))
\end{aligned}$$

$$\forall S \subseteq V (s \in S \wedge \forall x \forall y ((x \in S \wedge E(x,y)) \rightarrow y \in S) \rightarrow t \in S)$$

Course Outline

This course is concerned with the questions of (1) how definability relates to computational complexity and (2) how to analyse definability.

1. Complexity Theory—a review of the major complexity classes and their interrelationships (3L).
2. First-order and second-order logic—their expressive power and computational complexity (3L).
3. Lower bounds on expressive power—the use of games and locality (3L).
4. Fixed-point logics and descriptive complexity (3L)
5. Finite-variable logics; Random structures; (4L)

Useful Information

Some useful books:

- C.H. Papadimitriou. Computational Complexity. Addison-Wesley. 1994.
- H.-D. Ebbinghaus and J. Flum. Finite Model Theory (2nd ed.) 1999.
- N. Immerman. Descriptive Complexity. Springer. 1999.
- L. Libkin. Elements of Finite Model Theory. Springer. 2004.
- E. Grädel et al. Finite Model Theory and its Applications. Springer. 2007.

Course website: `http://www.cl.cam.ac.uk/teaching/0910/L15/`

Decision Problems and Algorithms

Formally, a *decision problem* is a set of strings $L \subseteq \Sigma^*$ over a finite alphabet Σ.

The problem is *decidable* if there is an *algorithm* which given any input $x \in \Sigma^*$ will determine whether $x \in L$ or not.

The notion of an *algorithm* is formally defined by a *machine model*: A *Turing Machine*; *Random Access Machine* or even a *Java program*.

The choice of machine model doesn't affect what is or is not decidable.

Similarly, we say a function $f : \Sigma^* \to \Delta^*$ is *computable* if there is an algorithm which takes input $x \in \Sigma^*$ and gives output $f(x)$.

Turing Machines

For our purposes, a Turing Machine consists of:

- K — a finite set of states;
- Σ — a finite set of symbols, including $\sqcup$.
- $s \in K$ — an initial state;
- $\delta : (K \times \Sigma) \to (K \cup \{a, r\}) \times \Sigma \times \{L, R, S\}$
  A transition function that specifies, for each state and symbol a next state (or accept acc or reject rej), a symbol to overwrite the current symbol, and a direction for the tape head to move (L – left, R – right, or S - stationary)

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Definition
A *configuration* is a triple (q, w, u), where $q \in K$ and $w, u \in \Sigma^\star$

The intuition is that (q, w, u) represents a machine in state q with the string wu on its tape, and the head pointing at the last symbol in w.

The configuration of a machine completely determines the future behaviour of the machine.

Computations

Given a machine $M = (K, \Sigma, s, \delta)$ we say that a configuration (q, w, u) *yields in one step* (q', w', u'), written

$$(q, w, u) \rightarrow_M (q', w', u')$$

if

- $w = va$;
- $\delta(q, a) = (q', b, D)$; and
- either $D = L$ and $w' = v$ $u' = bu$
  or $D = S$ and $w' = vb$ and $u' = u$
  or $D = R$ and $w' = vbc$ and $u' = x$, where $u = cx$. If u is empty, then $w' = vb\sqcup$ and u' is empty.

Computations

The relation $\rightarrow_M^\star$ is the reflexive and transitive closure of $\rightarrow_M$.

A sequence of configurations $c_1, \ldots, c_n$, where for each i, $c_i \rightarrow_M c_{i+1}$, is called a *computation* of M.

The language $L(M) \subseteq \Sigma^\star$ *accepted* by the machine M is the set of strings

$$\{x \mid (s, \triangleright, x) \rightarrow_M^\star (\text{acc}, w, u) \text{ for some } w \text{ and } u\}$$

A machine M is said to *halt on input* x if for some w and u, either $(s, \triangleright, x) \rightarrow_M^\star (\text{acc}, w, u)$ or $(s, \triangleright, x) \rightarrow_M^\star (\text{rej}, w, u)$

Complexity

For any function $f : \mathbb{N} \rightarrow \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f(n))$ if there is a machine $M = (K, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f(n))$.

Similarly, we define $\mathsf{SPACE}(f(n))$ to be the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

In defining space complexity, we assume a machine M, which has a read-only input tape, and a separate work tape. We only count cells on the work tape towards the complexity.

Nondeterminism

If, in the definition of a Turing machine, we relax the condition on δ being a function and instead allow an arbitrary relation, we obtain a *nondeterministic Turing machine*.

$$\delta \subseteq (K \times \Sigma) \times (K \cup \{a, r\} \times \Sigma \times \{R, L, S\}).$$

The yields relation $\rightarrow_M$ is also no longer functional.

We still define the language accepted by M by:

$$L(M) = \{x \mid (s, \triangleright, x) \rightarrow_M^\star (\text{acc}, w, u) \text{ for some } w \text{ and } u\}$$

though, for some x, there may be computations leading to accepting as well as rejecting states.

Nondeterministic Complexity

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{NTIME}(f(n))$ if there is a *nondeterministic* machine $M = (K, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f(n))$.

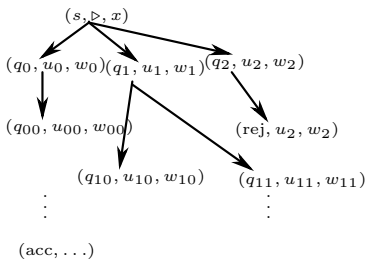
The last statement means that for each $x \in L(M)$, there is a computation of M that accepts x and whose length is bounded by $O(f(|x|))$.

Similarly, we define $\mathsf{NSPACE}(f(n))$ to be the languages accepted by a *nondeterminstic* machine which uses $O(f(n))$ tape cells on inputs of length n.

As before, in reckoning space complexity, we only count work space.

Computation Trees

With a nondeterministic machine, each configuration gives rise to a tree of successive configurations.



Complexity Classes

A complexity class is a collection of languages determined by three things:

- A model of computation (such as a deterministic Turing machine, or a nondeterministic TM, or a parallel Random Access Machine).
- A resource (such as time, space or number of processors).
- A set of bounds. This is a set of functions that are used to bound the amount of resource we can use.

Polynomial Bounds

By making the bounds broad enough, we can make our definitions fairly independent of the model of computation.

> The collection of languages recognised in *polynomial time* is the same whether we consider Turing machines, register machines, or any other deterministic model of computation.
>
> The collection of languages recognised in *linear time*, on the other hand, is different on a one-tape and a two-tape Turing machine.

We can say that being recognisable in polynomial time is a property of the language, while being recognisable in linear time is sensitive to the model of computation.

Reading List for this Handout

1. Papadimitriou. Chapters 1 and 2.
2. Grädel et al. Chapter 1 (Weinstein).