CoCoNuT - Complexity - Lecture 7

N.P. Smart

Dept of Computer Science
University of Bristol,
Merchant Venturers Building

May 5, 2015

Outline

Circuits

Turing machines are in one sense like a modern computer, but in another sense they are not.

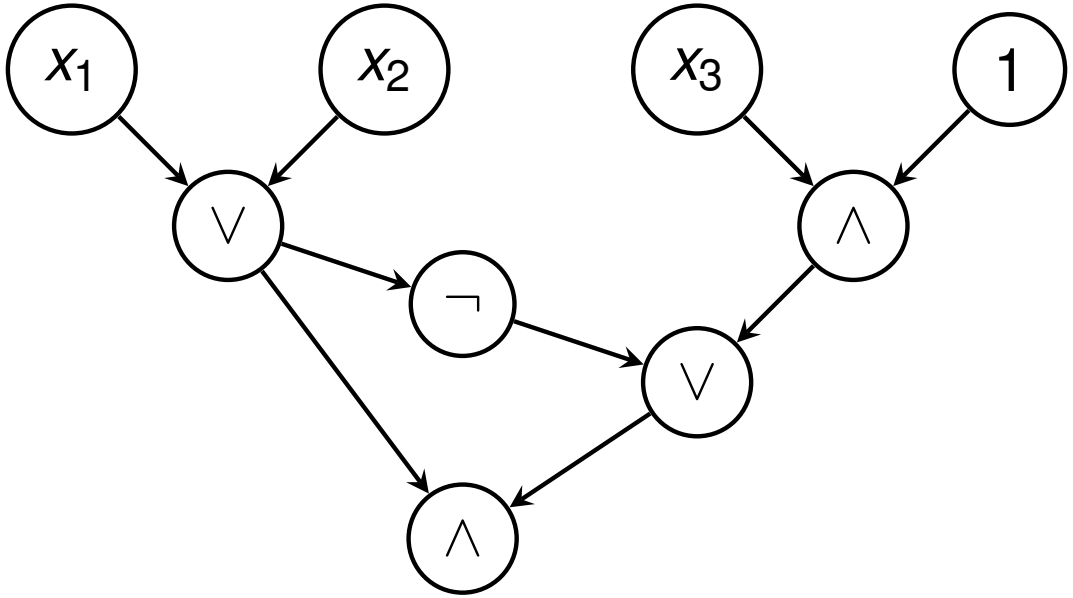
Computers are built out of circuits.

For us a circuit C will be

- Have n inputs.
- A single output (only doing decision problems in this course).
- Directed acylic graph.
- One node with out-degree zero (the output node).
- Source nodes (in degree zero) labelled with variables $x_1, \ldots, x_n$ or the constants $0, 1$.
- Non source nodes called "gates".
- Gates labelled $\vee$, $\wedge$ have in-degree two.
- Gates labelled $\neg$ have in-degree one.

The size of C, $|C|$, is the number of nodes in C.

Circuits



A circuit C computing a boolean function of 3 variables.

One can verify that in fact $C(x) = (x_1 \vee x_2) \wedge x_3$.

Circuits

A circuit C computes a boolean function

$$C : \{0,1\}^n \longrightarrow \{0,1\}.$$

Every boolean formulae can be written as a boolean circuit.

But: The size of the circuit could be of size at least $2^n/(2 \cdot n)$.

We want to restrict the type of circuits we will utilize.

Circuits

A circuit C computes a boolean function

$$C : \{0,1\}^n \longrightarrow \{0,1\}.$$

Every boolean formulae can be written as a boolean circuit.

But: The size of the circuit could be of size at least $2^n/(2 \cdot n)$.

We want to restrict the type of circuits we will utilize.

Poly Size Circuits

Definition: $T(n)$-size Circuits

Let $T :\rightarrow$ be a function. A $T(n)$-size circuit family $\mathcal{C}$ is a sequence $C_0, C_1, \ldots$ of boolean circuits, where C_n has n inputs and one output, and $|C_n| \leq T(n)$ for all n.

Let $\mathcal{L} \subseteq \{0,1\}^*$ be a language.

We say that a circuit family $\mathcal{C}$ decides $\mathcal{L}$ if, for every n and every $x \in \{0,1\}^n$, $x \in \mathcal{L} \Leftrightarrow C_n(x) = 1$.

Poly Sized Circuits

Definition: SIZE($T(n)$)

Let $T :\rightarrow$ be a function and $\mathcal{L}$ be a language. We say that $\mathcal{L} \in \text{SIZE}(T(n))$ if there exists a $T(n)$-size circuit family $\mathcal{C}$ such that $\mathcal{C}$ decides $\mathcal{L}$.

Note, for each n there is a different circuit

- Unlike the case of Turing machines where same PPT works for each input size.

Definition: The class P/poly

$$\text{P/poly} = \bigcup_{c>0} \text{SIZE}(n^c).$$

$\mathsf{P} \subseteq \mathsf{P}/\mathsf{poly}$

Let $\mathcal{L}$ be a language in P

The proof of $\mathsf{P} \subseteq \mathsf{P}/\mathsf{poly}$ is like the proof of Cook-Levin.

Instead of outputing a boolean formulae we output a boolean circuit.

By the locality of TM's we can make the circuit poly-sized.

Hence $\mathsf{P} \subseteq \mathsf{P}/\mathsf{poly}$

In fact P/poly is much bigger than P...

$\mathrm{UHALT} \in \mathrm{P/poly}$

Amazingly P/poly contains languages which a normal TM cannot even decide....

Define

$$\mathrm{UHALT} = \{1^n : \text{Binary expansion of } n \text{ encodes a TM } M \text{ and input } x \text{ s.t. } M \text{ halts on input } x\}.$$

The circuit family which decides UHALT is...

- The circuit of size n which outputs one if $1^n \in \mathrm{UHALT}$, i.e. the one which performs AND on all n input bits.
- The circuit of size n which outputs zero if $1^n \notin \mathrm{UHALT}$, i.e. the one which outputs zero.

Note: We show there is a circuit to decide UHALT without constructing the circuit!

- This is nuts!

L-uniform circuits

To avoid such nuts results we define

Definition: L-uniform

A circuit family $\mathcal{C} = C_0, C_1, \ldots$ is said to be L-uniform if there is a logarithmic-space Turing machine which, on input 1^n, outputs a description of the circuit C_n.

i.e. There exists an implicitly logspace-computable function which maps 1^n to C_n.

This type of circuit turns out to be precisely equivalent to P.

Another definition P/poly

We can also think of P/poly as the languages which are

- Recognized by a TM.
- Where the TM is given a poly-bounded "advice" function.

The advice function can depend on the length of the input

P-Completness

Definition: P-Complete

We say that a language $\mathcal{A}$ is P-complete if $\mathcal{A} \in \mathsf{P}$, and for all $\mathcal{B} \in \mathsf{P}$, $\mathcal{B} \leq_{\mathsf{L}} \mathcal{A}$.

Where $\mathcal{B} \leq_{\mathsf{L}} \mathcal{A}$ means a log-space reduction.

P-complete problems are the "hardest" problems in P

Indeed, if $\mathcal{L}$ is P-complete and $\mathcal{L} \in \mathsf{L}$, then $\mathsf{P} = \mathsf{L}$.

P-Completeness

Definition: CIRCUIT VALUE

The input is a pair (C, x), where C is the description of an n-input circuit and $x \in \{0,1\}^n$. The problem is to decide whether $C(x) = 1$.

CIRCUIT VALUE is P-complete.

CIRCUIT SAT is NP-complete.

Other Relations

Whilst not practically useful P/poly is useful theoretically:

- If NP $\subseteq$ P/poly then AM $=$ MA.
- If PSPACE $\subseteq$ P/poly then PSPACE $-$ MA.
- If EXP $\subseteq$ P/poly then EXP $=$ MA.

P/poly useful in crypto to define adversaries which do a lot of precomputation for a specific key size

- i.e. determining the advice function.

Getting Smaller: NC

Upto now, except when we looked at log-space computations, we wondered what happens if we give more resources to a TM.

We now look at restrictions in resources

A practically important one in the circuit complexity model is that of low depth circuits.

Definitions: NC^d

For every d, a language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class NC^d if $\mathcal{L}$ can be decided by an L-uniform family of circuits $\{C_n\}$ such that each C_n has size poly(n) and depth $O(\log^d n)$.

We write

$$\mathrm{NC} = \bigcup \mathrm{NC}^d.$$

Getting Smaller: AC

Definitions: AC^d

For every d, a language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class AC^d if $\mathcal{L}$ can be decided by an L-uniform family of circuits, with each OR and AND gate allowed unbounded fan-in, $\{C_n\}$ such that each C_n has size poly(n) and depth $O(\log^d n)$.

We write

$$AC = \bigcup_{d \geq 0} AC^d.$$

We have

$$NC^d \subseteq AC^d \subseteq NC^{d+1}.$$

NC and AC

Why study these?

By showing something does not lie in NC^d for some d we provide a lower bound.

Note, complexity is usually about upper bounds.

But really we want to know what the fastest way of doing something is, not the slowest!

Problem is not many lower bounds are known.

Course Summary

So what have we learned:

1. Some things are easy: P.
2. Most interesting problems have solutions which can be checked quickly: NP.
3. Some problems are NP-complete.
4. Randomness seems to give us more power: BPP, RP, ZPP
5. $\mathsf{P} \subseteq \mathsf{BPP}$, but it is conjectured they are equal.
6. Interaction gives us more power: $\mathsf{IP} = \mathsf{PSPACE}$.
7. We can learn the truth of something through interaction, but not why: ZK.
8. Circuits, or computing with advice, can solve apparently undecidable problems.
9. Circuit complexity can provide us with lower bounds.

But above all, we have learned that computer science theory has profound things to say about all of human knowledge.