TU Dresden, Fakultät Informatik
Daniel Borchmann

Summer Term 2016

Advanced Topics in Complexity Theory

Exercise 4: Revision on Boolean Circuits

2016-04-26

Circuits provide an important alternative model for computation, a model that may seem to be much closer to the actual way we perform computation on modern computers.

Definition 4.1 A *Boolean circuit* C is a finite, acyclic, directed graph. We shall call the vertices of C *gates* and the edges of C *wires*. Gates with no predecessors are called are called *input gates*. In a Boolean circuit every gate that is not an input gate is one of the following:

- an AND gate, with two input wires and one output wire,
- an OR gate, with two input wires and one output wire,
- a NOT gate, with one input wire and one output wire.

Every Boolean circuit has a specifically designated gate called the *output gate*. The size of a Boolean circuit is its number of its gates. ♢

Computation with Boolean circuits is done as expected: given values for the input gates all other gates compute their values from the corresponding logical connectives. The value of the output gate is called the *output* of the circuit. If C is a Boolean circuit with n input gates and $x \in \{ 0, 1 \}^n$, then the value of the output gate of C on input x is denoted $C(x)$.

A function $f \colon \{ 0, 1 \}^n \to \{ 0, 1 \}$ is called a *Boolean function* with n variables. It is not hard to see that all Boolean functions can be computed by a Boolean circuit.

Exercise 4.2 Show that every Boolean function with n variables can be computed with a circuit of size $\mathcal{O}(n2^n)$.

Solution Let $f \colon \{ 0, 1 \}^n \to \{ 0, 1 \}$. Let

$$D \coloneqq f^{-1}(1) = \{ (x_1, \ldots, x_n) \in \{ 0, 1 \}^n \mid f(x_1, \ldots, x_n) = 1 \}.$$

Then

$$f(x_1, \ldots, x_n) \equiv \bigvee_{i \in D} E_i^n(x_1, \ldots, x_n),$$

where $E_i^n(x_1, \ldots, x_n)$ checks whether the input equals i in binary, i.e.,

$$E_i^n(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } i = [x_n \ldots x_1]_2 \\ 0 & \text{otherwise.} \end{cases}$$

Each $E_i^n(x_1, \ldots, x_n)$ can be implemented with a circuit with size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$. Since D may have size up to 2^n, the big disjunction can be implemented with a circuit of size $\mathcal{O}(n2^n)$. □

To decide languages using circuits it is not enough to provide only *one* circuit. The problem here is that each circuit has a fixed number of inputs, and thus can only decide a finite number of words. To be able to decide infinite languages one has to use a different circuit for each input length.

Definition 4.3 A *circuit family* is a sequence $(C_n \mid n \in \mathbb{N})$ of circuits where each circuit C_n has exactly n input gates. A circuit family $(C_n \mid n \in \mathbb{N})$ is *f-size bounded* for some function $f\colon \mathbb{N} \to \mathbb{N}$ if for each $n \in \mathbb{N}$ the size of C_n is at most $f(n)$.

Let $L \subseteq \{\,0,1\,\}^*$. A circuit family *decides* L if for each $x \in \{\,0,1\,\}^*$ we have

$$x \in L \iff C_{|x|}(x) = 1.$$

We say that L has *circuit complexity* $t(n)$ if L can be decided by a $t(n)$-size bounded circuit family. ♢

It can be shown that circuits can efficiently simulate Turing machine computations.

Theorem 4.4 Let $L \in \mathsf{DTime}(t(n))$. Then L has circuit complexity $\mathcal{O}(t(n)^2)$ and these circuits can be computed in polynomial time in the length of the input.

The main idea of the proof is quite easy: using a suitable Turing machine M to decide L, one represents the tableau of the computation of M on inputs of length n using a circuit. Since the tableau has quadratic size in the runtime of M, the size of the resulting circuit will also be quadratic.

On the other hand, the technical details of the proof are intriguing. See Theorem 9.30 of Sipser for more.

One can use this theorem to show that the following problem is NP-complete:

$$\mathsf{CircuitSAT} \coloneqq \{\,\langle C\rangle \mid C \text{ a satisfiable Boolean circuit}\,\}.$$

Here, a circuit C is called *satisfiable* if there exists some input x of the correct length such that $C(x) = 1$.

Exercise 4.5 Show that CircuitSAT is NP-complete.

Using the NP-completeness of CircuitSAT we can provide an alternative proof that SAT is NP-complete.

Exercise 4.6 Show that CircuitSAT can be reduced to SAT in polynomial time.

One can define an analog to P that uses polynomial-sized circuits instead of polynomial time Turing machines.

Definition 4.7 The class P/poly is the class of all languages with polynomial circuit complexity. ♢

However, the class P/poly behaves quite differently from P, or other complexity classes, as it contains undecidable languages. The reason for this is that the circuit family deciding a language in P/poly may be *non-uniform*, i.e., there is no Turing machine that computes the circuits in this family.

Exercise 4.8 Show that every language $L \subseteq \{ 1^n \mid n \in \mathbb{N} \}$ is contained in P/poly. Conclude that P/poly contains undecidable languages.

Solution We define $C_n(x_1, \ldots, x_n) \coloneqq x_1 \wedge \neg x_1$ if $1^n \notin L$, and otherwise $C_n(x_1, \ldots, x_n) = x_1 \wedge \cdots \wedge x_n$. Those circuits are of polynomial size and decide L.

There are uncountably many sublanguages of $\{ 1^n \mid n \in \mathbb{N} \}$ and thus most of them must be undecidable, but all of them are contained in P/poly. □

There are also decidable languages in P/poly that are not solvable in P.

Exercise 4.9 (Optional) Find a decidable language in P/poly that is not contained in P.

Hint:
Take a language over $\{ 0, 1 \}$ that is $\mathsf{2ExpTime}$-hard and consider its unary encoding.

Solution Let $L \subseteq \{ 0, 1 \}^*$ that is hard for $\mathsf{2ExpTime}$. Those languages exist. Consider the language L' defined as the unary encoding of L, i.e.,

$$L' \coloneqq \{ 1^n \mid n \in L \}.$$

Then $L' \in \mathsf{P}/\mathsf{poly}$. We claim that $L' \notin \mathsf{P}$.

Suppose by contradiction that $L' \in \mathsf{P}$. Then $L \in \mathsf{ExpTime}$: to decide whether a word $w \in \{ 0, 1 \}^*$ is in L we first encode w in unary and check whether this encoding is in L'. This yields an $\mathsf{ExpTime}$-procedure showing that $L \in \mathsf{ExpTime}$. Since L is $\mathsf{2ExpTime}$-hard we conclude that $\mathsf{2ExpTime} \subseteq \mathsf{ExpTime}$, contradicting the Time Hierarchy Theorem. □

Finally, there is some neat connection to randomized computation.

Exercise 4.10 Show $\mathsf{BPP} \subseteq \mathsf{P}/\mathsf{poly}$.