**3P17/4P17 COMPUTABILITY 2014-2015 Lecture Handout 15**

## Boolean formulas and the satisfiability problem (SAT)

A **Boolean value** is true or false, often represented with 1 and 0, respectively.

A **Boolean variable** $x$ is a variable which takes a Boolean value, i.e., $x = 0$ or $x = 1$.

The following are **Boolean operations**: AND, OR, NOT, written as $\wedge$, $\vee$ and $\neg$, respectively: $x_1 \wedge x_2$, $x_1 \vee x_2$, $\neg x$. The latter is also written $\overline{x}$.

A **literal** is a Boolean variable or the negation of a Boolean variable, e.g., $x$, $\overline{x}$.

$$\begin{array}{lll} 0 \wedge 0 = 0 & 0 \vee 0 = 0 & \overline{0} = 1 \\ 0 \wedge 1 = 0 & 0 \vee 1 = 1 & \overline{1} = 0 \\ 1 \wedge 0 = 0 & 1 \vee 0 = 1 & \\ 1 \wedge 1 = 1 & 1 \vee 1 = 1 & \end{array}$$

A **Boolean formula** is an expression involving Boolean variables and the Boolean operations, e.g.,

$$\phi = x_1 \vee \overline{x_2} \vee (x_3 \wedge \overline{x_1}).$$

A Boolean formula is **satisfiable** if there is an assignment of 0s and 1s to the variables which make it evaluate to 1.

The **satisfiability problem** is to test whether or not a formula is satisfiable. More precisely, it is to decide the following language:

$$\text{SAT} = \{\langle\phi\rangle : \phi \text{ is a satisfiable Boolean formula}\}.$$

A **clause** is a group of literals connected with $\vee$s, e.g., $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$.

A Boolean formula is in **conjunctive normal form** and is called a **cnf-formula** if it is made up of clauses connected by $\wedge$s, e.g.,

$$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4) \wedge (x_3 \vee \overline{x_6} \vee x_7).$$

A special case is **3cnf Boolean formulas**, in which all clauses contain exactly three literals, e.g.,

$$(\overline{x_2} \vee \overline{x_3} \vee x_9) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_5 \vee \overline{x_6} \vee x_9) \wedge (x_3 \vee x_6 \vee \overline{x_8}).$$

A special case of the satisfiablility problem is the **3SAT problem**, which is deciding the following language:

$$\text{3SAT} = \{\langle\phi\rangle : \phi \text{ is a satisfiable 3cnf formula}\}.$$

## Polynomial time reducibility

A language $A$ is **polynomial time mapping reducible to B**, written $A \leq_p B$ if there is a mapping reduction $A \leq_m B$ and the mapping function $f$ is computable in polynomial time.

i.e., it's just a mapping reduction computable in polynomial time.

**Theorem 1.** *If $A \leq_p B$ and $B \leq_p C$ then $A \leq_p C$.*

*Proof.* Left as an exercise. □

**Theorem 2.** *If $A \leq_p B$ and $B \in \boldsymbol{P}$ then $A \in \boldsymbol{P}$.*

*Proof.* Suppose $A \leq_p B$ and $B \in \mathbf{P}$. Let $M_B$ decide $B$ in polynomial time and let $f$ be the polynomial time mapping reduction. Then $M_A$ is a TM that decides $A$ in polynomial time:
$\underline{M_A}$
On input $\sigma$,

(1) Compute $f(\sigma)$.

(2) Simulate $M_B$ on $f(\sigma)$. Accept if it accepts, reject if it rejects.

If $\sigma \in A$ then $f(\sigma) \in B$ because $f$ is a mapping reduction from $A$ to $B$. Then $M_B$ will accept $f(\sigma)$ so $M_A$ will accept $\sigma$. If $\sigma \notin A$ then $f(\sigma) \notin B$ (again because $f$ is a mapping reduction. Then $M_B$ will reject $f(\sigma)$ so $M_A$ will reject $\sigma$. Hence, $M_A$ decides $A$.

Furthermore, it decides it in polynomial time since each step is executed once and each step takes polynomial time.

Hence, $A \in \mathbf{P}$. □

**Theorem 3.** *3SAT $\leq_p$ CLIQUE.*

*Proof.* We define a reduction $f$ that maps the string $\langle\phi\rangle$, where $\phi$ is a Boolean formula in 3cnf form, to a string $\langle G, k\rangle$.

The nodes of $G$ are organised into $k$ groups, one group for each clause of $\phi$.

Each group contains three nodes, with each node $u$ labelled by a literal $\ell(u)$ of the clause its group is associated with - one literal to each node. Thus, for example, if there is a clause $(x_1 \vee \overline{x_2} \vee x_3)$, then for the three nodes $u, v, w$ associated with this clause, we have $\ell(u) = x_1, \ell(v) = \overline{x_2}, \ell(w) = x_3$.

There is an edge $(u, v)$ in $G$ except in the case that $\ell(u) = \overline{\ell(v)}$, or $u$ and $v$ are in the same group.

$f$ is polynomial time computable since it only involves generating a graph $G$ polynomial in the size of $\phi$. If $\phi$ has $k$ clauses, then $G$ has $3k$ vertices and at most $(3k)^2$ edges. As we add edges only between non-complementary literals/vertices in different groups, the process of constructing $G$ requires at most $(3k)^2$ steps.

Now suppose $\langle\phi\rangle \in$ 3SAT. Then there is a satisfying assignment. In that assignment, at least one literal is true in every clause. Select a vertex with true literal in each clause/group (pick arbitrarily if there is more than one). These $k$ vertices must be connected together since there is an edge between a pair except when they are in the same group or are negations of each other, which can't be the case if they are both true. Thus, we have a clique and $\langle G, k\rangle \in$ CLIQUE.

Now suppose $\langle G, k\rangle \in$ CLIQUE. Choose a $k$-clique in $G$ and set all literals to true. We can do this because if an edge connects a pair of vertices, their literals must either be identical or for different variables, meaning there is no logical contradiction. Also, since there are no edges between vertices in the same group, each vertex in the clique belongs to a different group. Hence, each clause is true and so $\langle\phi\rangle \in$ 3SAT.

For garbage inputs we can map $f$ to some error string as usual. Thus, we have demonstrated $\sigma \in$ 3SAT if and only if $f(\sigma) \in$ CLIQUE, where $f$ is polynomial time computable, meaning $f$ is a polynomial time mapping reduction.

□

## NP-completeness

A language $B$ is **NP-complete** if it satisfies the following two conditions:

1. $B$ is in **NP**.
2. Every $A$ in **NP** is polynomial time reducible to $B$.

**NP**-complete problems are, in a sense, the hardest problems in the class **NP**; if you can solve any one of them in polynomial time, you can solve every problem in **NP** in polynomial time.

**Theorem 4.** *If $B$ is* **NP***-complete and $B \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.*

*Proof.* Follows immediately from the definitions. □

**Theorem 5.** *If $B$ is* **NP***-complete and $B \leq_p C$ for $C \in \mathbf{NP}$, then $C$ is* **NP***-complete.*

*Proof.* Since $C \in \mathbf{NP}$ by assumption, it only remains to show the second condition: that any language $A$ in **NP** is polynomial time reducible to $C$.

Since $B$ is **NP**-complete, any $A$ in **NP** would be polynomial time reducible to it, i.e., $A \leq_p B$. But we have $B \leq_p C$, so by Theorem 1, $A \leq_p C$. □

**Theorem 6** (Cook's Theorem)**.** *SAT is* **NP***-complete.*

The proof is quite long, and is not covered in this course. It is not obvious, but a corollary to Cook's Theorem is the following:

**Corollary 7.** *3SAT is* **NP***-complete.*

**Theorem 8.** *CLIQUE is* **NP***-complete.*

*Proof.* CLIQUE is in **NP** (Handout 14). Now Combine Theorem 5, Theorem 3 and Corollary 7. □