Formulas and Circuits

1 Formulas as circuits

So far, we have looked at logic from the mathematician's point of view. Now let's look at it from the electrical engineer's point of view.

- If for all truth assignments τ we have $\varphi(\tau) = 1$, then mathematicians say that φ is valid. EEs say the corresponding circuit is "stuck at one".
- If for all turth assignments τ we have $\varphi(\tau) = 0$, then mathematicians say that φ is unsatisfiable. EEs say the corresponding circuit is "stuck at zero".
- If for some τ_1, $\varphi(\tau_1) = 1$, and for some τ_2, $\varphi(\tau_2) = 0$, then mathematicians say φ is satisfiable but not valid. To an electrical engineer, such circuits are the only interesting ones.
- Equivalence of logical formulas: $\varphi \models\!\!=\!\!| \psi$ if and only if for all $\tau \in 2^{Prop}$ we have $\varphi(\tau) = \psi(\tau)$. In circuitry, the idea of equivalence is the basis for modularity.

2 Building formulas from functions

We would like to show that for every boolean function, we can create a formula and, hence, a circuit that corresponds to that formula.

Definition 1 *We say a formula φ MATCH a boolen function f, if for all $\tau \in 2^{Prop}$, we have $f(\tau) = 1$ iff $\tau \models \varphi$*

Theorem 1 *Let $Prop = \{p_1, p_2, \ldots, p_n\}$. For every function $f : 2^{Prop} \mapsto \{0, 1\}$, there is a macthing formula φ_f such that for all $\tau \in 2^{Prop}$, we have $f(\tau) = 1$ iff $\tau \models \varphi_f$*

First let's see an example of how we construct this formula φ_f.

p_1	p_2	$f(p_1, p_2)$
0	0	1
0	1	1
1	0	1
1	1	0

We recognize this immediately as the NAND function, so one equivalent formula would be $\neg(p_1 \wedge p_2)$. However, if we want to construct the formula methodically, we can create a conjunction of the literals in every line of the truth table where $f = 1$ and then create the disjunction of these conjunctions. For this function, we get $(\neg p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2)$.
Before proving the theorem,we give following definition:

Definition 2 *A literal is a proposition or the negation of a proposition. In the following proof, we will denote a positive literal p by p^1 and a negative literal $\neg p$ by p^0.*

Now let's prove theorem 1 by construction:
Proof: Let

$$\varphi_f = \bigvee_{f(\tau)=1} (\bigwedge_{j=1}^{n} p_j^{\tau(p_j)}),$$

where we consider all truth assignments $\tau : Prop \rightarrow 0, 1$. We need to show that this formula is equivalent to the function.

Consider the case where $f(\tau) = 1$. We need to show that $\varphi_f(\tau) = 1$, i.e., $\tau \models \varphi_f$. Since φ_f is a disjunction of conjunctions, $\varphi_f(\tau) = 1$ when τ satisfies one of the conjunctions, so let's show

$$\tau \models \bigwedge_{j=1}^{n} p_j^{\tau(p_j)}$$

We know, however, that $\tau(p_j) = 1 \Rightarrow \tau \models p_j^1$, and $\tau(p_j) = 0 \Rightarrow \tau \models p_j^0$, so for all j we have $\tau \models p_j^{\tau(p_j)}$, so τ models the conjunction, and $\tau \models \varphi_f$.

The other case to consider is where $f(\tau) = 0$. We leave this as an exercise for the reader.

Corollary 1 *Any boolean function can be expressed using only the symbols $\neg$, $\wedge$, and $\vee$.*

Proof: Immediate from the construction in the theorem.

Definition 3 *A formula is in disjunctive normal form (DNF) if it has the form $\bigvee \bigwedge literals$.*

Corollary 2 *Every formula φ has an equivalent formula φ' in DNF such that $\varphi \models\dashv \varphi'$.*

Proof: Take φ as a function from $2^{Prop} \rightarrow \{0, 1\}$. Using the construction in the theorem, we can build φ_φ in DNF such that for all $\tau \in Prop$, we have $\varphi(\tau) = \varphi_\varphi(\tau)$.

Example: Let's express the statement "if p_1 then p_2, else p_3". The truth table for this statement is as follows:

p_1	p_2	p_3	$f(p_1, p_2, p_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Applying the construction gives us $(\neg p_1 \wedge \neg p_2 \wedge p_3) \vee (\neg p_1 \wedge p_2 \wedge p_3) \vee (p_1 \wedge p_2 \wedge \neg p_3) \vee (p_1 \wedge p_2 \wedge p_3)$.

However, we can reduce this to $(\neg p_1 \wedge p_3) \vee (p_1 \wedge p_2)$. We can also bypass the construction entirely and simply from the statement construct the following formula: $(p_1 \rightarrow p_2) \wedge (\neg p_1 \rightarrow p_3)$, which is equivalent to $(\neg p_1 \vee p_2) \wedge (p_1 \vee p_3)$.

Thus, there are many different ways to express the same formula, and our construction does not necessarily give us the shortest. In fact, the worst case is valid formulas, which clearly result in exponential size.

3 Size of formulas

Up until now, we have used the length of a formula as the measure of its size, but this is not entirely accurate. Sure, we can represent most symbols, like (,), $\neg$, $\wedge$, and $\vee$ with just a few bits each, but what about a proposition, like p_5? In binary, we can represent this as #101. Thus, a proposition grows to size $\log n$, where n is the length of the formula, and the entire formula grows to $O(n \log n)$.

Now let's try to bound the size of formulas. Our usual notion of efficiency is polynomial time or space, so we would like to bound the size of a formula by a polynomial. We will see that we cannot do this, however.

Let $|Prop| = n$, and let $f : 2^{Prop} \mapsto \{0, 1\}$. We would like to find φ_f such that for all $\tau \in 2^{Prop}$, we have $\varphi_f(\tau) = f(\tau)$, and furthermore, we want $\|\varphi_f\| \leq p(n)$ for some polynomial $p(n)$.

Theorem 2 *For every polynomial p, there is a set $Prop$ such that $|Prop| = n$ and a function $f : 2^{Prop} \rightarrow \{0, 1\}$ such that for all φ, if φ implements f, then $\|\varphi\| > p(n)$.*

Proof: Consider all formulas φ such that $\|\varphi\| \leq p(n)$. Clearly there are at most $2^{p(n)}$ such formulas. On the other hand, there are 2^{2^n} Boolean functions from $2^{\{p_1, p_2, \ldots, p_n\}} \rightarrow \{0, 1\}$. Thus, there must be functions that cannot be represented by polynomial-size formulas.

4 Converting between DNF and CNF

A formula in DNF is of the form $\bigvee \bigwedge literals$ while a formula in CNF is of the form $\bigwedge \bigvee literals$. There is also a normal form called PNF (positive normal form), in which negations are applied only to atoms.

The negation of a formula in CNF is a formula in DNF because we can push the negation through the formula: $\neg \bigwedge \bigvee literals \models\!\!\!\!\dashv \bigvee \neg \bigvee literals \models\!\!\!\!\dashv \bigvee \bigwedge \neg literals$. Similarly, the negation of a formula in DNF is a formula in CNF.

Corollary 3 *Every formula φ has a formula φ'' in CNF such that $\varphi \models\!\!\!\!\dashv \varphi''$.*

Proof: Start with $\neg\varphi$ and using the earlier theorem, find φ' in DNF such that $\neg\varphi \models\!\!=\!\!| \varphi'$. Then negate φ' and push the negation through to get φ'' in CNF.

Fact: For formulas in CNF, SAT is NP-complete. In fact, even 3-SAT is NP-complete.

Fact: For formulas in DNF, SAT is polynomial time.

Proof: $\bigvee_i \bigwedge_j l_{ij}$ is satisfiable if and only if for some i we have that $\bigwedge_j l_{ij}$ is satisfiable. But a conjunction is satisfiable if and only if it does not contain contradictory literals, i.e. l_{ij} and $\neg l_{ij}$. We can certainly check this in polynomial time.

Fact: For formulas in DNF, VAL is co-NP-complete.

Proof: φ is valid if and only iff $\neg\varphi$ is unsatisfiable. But $\neg\varphi$ is in CNF, and SAT is NP-complete for formulas in CNF.

Fact: For formulas in CNF, VAL is polynomial time.

Proof: Similar to the previous one.

5 How many types of gates are enough?

We think of formulas as trees, but to an electrical engineer they are really graphs. For example, $(p \to p) \to (p \to p)$ could be represented as the following graph:

$\to$

()

$\to$

()

p

So we can represent formulas more compactly, but can we get by with fewer types of gates? We have already seen that $\{\neg, \wedge, \vee\}$ is sufficient, and DeMorgan's laws show us how to represent conjunction in terms of disjunction and negation and also disjunction in terms of conjunction and negation, so $\{\neg, \vee\}$ and $\{\neg, \wedge\}$ are also sufficient. We would like to find some minimal set of operators that allow us to express any formula.

Definition 4 *: A base K is a finite set of Boolean functions. K-logic is defined as follows:*

1. *Formulas: $Form^K$*
   - $Prop \subseteq Form^K$.
   - *For each m-ary function $\alpha \in K$ and K-formulas $\varphi_1, \varphi_2, \ldots, \varphi_m$, $\alpha(\varphi_1, \varphi_2, \ldots, \varphi_m) \in Form^K$.*
2. *Interpretation*
   - $\tau \models p_i$ *if* $\tau(p_i) = 1$.
   - $\tau \models \alpha(\varphi_1, \varphi_2, \ldots, \varphi_m)$ *if* $\alpha(\tau(\varphi_1), \tau(\varphi_2), \ldots, \tau(\varphi_m)) = 1$.

Definition 5 *A base K is* adequate *if every Boolean function can be implemented by a formula in K-logic.*

Proving that K is not adequate by showing that there exists some f such that $f \notin Form^k$, this can be done in two steps:

- Choose a property C such that every boolen function in $Form^k$ satisfies C.
- Show that there exists a boolean function f that f doesn't satisfies C(here "property C" is not well defined,will show it in example).

Example:
K={+}

- Every function f in $Form^+$ satisfies $f(\tau_0) = 0$,where $\tau_0(p_i) = 0$ for all p_i; Proof by induction on $Form^+$:
  - Base case:$\tau_0(p_i) = 0$
  - Induction step:
    Assume that $f_1, f_2 \in Form^+$ and $f = f_1 + f_2$. The induction hypothesis implies that $f_1(\tau_0) = 0$,$f_2(\tau_0) = 0$, the definition of "+" implies $f(\tau_0) = 0$.
- The function f_1 that assign 1 to every τ doesn't satisfy this property.
  Thus $f_1 \notin Form^+$,i.e. $Form^+$ is not adequate.

By this definition, then, $\{\neg, \wedge, \vee\}$ is adequate. $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are also adequate.