# Problem Set 1

*Due*: February 17

**Reading:**

- Chapter 1. *What is a Proof?*,
- Chapter 2. *The Well Ordering Principle* through 2.3. *Factoring into Primes* (omit 2.4. *Well Ordered Sets*),
- Chapter 3. *Logical Formulas* through 3.3. *Equivalence and Validity*, and 3.5. *The SAT Problem* (optional: 3.4. *Algebra of Propositions*) in the course textbook.

These assigned readings do **not** include the Problem sections. (Many of the problems in the text will appear as class or homework problems.)

**Problem 1.**
Prove that $\log_4 6$ is irrational.

**Problem 2.**
Use the Well Ordering Principle to prove that

$$n \leq 3^{n/3} \tag{1}$$

for every nonnegative integer, $n$.

*Hint:* Verify (1) for $n \leq 4$ by explicit calculation.

**Problem 3.** **(a)** Verify by truth table that

$$(P \text{ IMPLIES } Q) \text{ OR } (Q \text{ IMPLIES } P)$$

is valid.

**(b)** Let $P$ and $Q$ be propositional formulas. Describe a single formula, $R$, using only AND's, OR's, NOT's, and copies of $P$ and $Q$, such that $R$ is valid iff $P$ and $Q$ are equivalent.

**(c)** A propositional formula is *satisfiable* iff there is an assignment of truth values to its variables—an *environment*—which makes it true. Explain why

$$P \text{ is valid} \quad \text{iff} \quad \text{NOT}(P) \text{ is } \textit{not} \text{ satisfiable.}$$

**(d)** A set of propositional formulas $P_1, \ldots, P_k$ is *consistent* iff there is an environment in which they are all true. Write a formula, $S$, so that the set $P_1, \ldots, P_k$ is *not* consistent iff $S$ is valid.

**Problem 4.**

There are adder circuits that are *much* faster, and only slightly larger, than the ripple-carry circuits of Problem 3.5 of the course text. They work by computing the values in later columns for both a carry of 0 and a carry of 1, *in parallel*. Then, when the carry from the earlier columns finally arrives, the pre-computed answer can be quickly selected. We'll illustrate this idea by working out the equations for an $(n + 1)$-bit parallel half-adder.

Parallel half-adders are built out of parallel *add1* modules. An $(n + 1)$-bit *add1* module takes as input the $(n + 1)$-bit binary representation, $a_n \ldots a_1 a_0$, of an integer, $s$, and produces as output the binary representation, $c\ p_n \ldots p_1 p_0$, of $s + 1$.

**(a)** A 1-bit *add1* module just has input $a_0$. Write propositional formulas for its outputs $c$ and $p_0$.

**(b)** Explain how to build an $(n + 1)$-bit parallel half-adder from an $(n + 1)$-bit *add1* module by writing a propositional formula for the half-adder output, $o_i$, using only the variables $a_i$, $p_i$, and $b$.

We can build a double-size *add1* module with $2(n + 1)$ inputs using two single-size *add1* modules with $n + 1$ inputs. Suppose the inputs of the double-size module are $a_{2n+1}, \ldots, a_1, a_0$ and the outputs are $c, p_{2n+1}, \ldots, p_1, p_0$. The setup is illustrated in Figure 1.

Namely, the first single size *add1* module handles the first $n + 1$ inputs. The inputs to this module are the low-order $n + 1$ input bits $a_n, \ldots, a_1, a_0$, and its outputs will serve as the first $n + 1$ outputs $p_n, \ldots, p_1, p_0$ of the double-size module. Let $c_{(1)}$ be the remaining carry output from this module.

The inputs to the second single-size module are the higher-order $n + 1$ input bits $a_{2n+1}, \ldots, a_{n+2}, a_{n+1}$. Call its first $n + 1$ outputs $r_n, \ldots, r_1, r_0$ and let $c_{(2)}$ be its carry.

**(c)** Write a formula for the carry, $c$, in terms of $c_{(1)}$ and $c_{(2)}$.

**(d)** Complete the specification of the double-size module by writing propositional formulas for the remaining outputs, $p_i$, for $n + 1 \le i \le 2n + 1$. The formula for $p_i$ should only involve the variables $a_i$, $r_{i-(n+1)}$, and $c_{(1)}$.

**(e)** Parallel half-adders are exponentially faster than ripple-carry half-adders. Confirm this by determining the largest number of propositional operations required to compute any one output bit of an $n$-bit add module. (You may assume $n$ is a power of 2.)

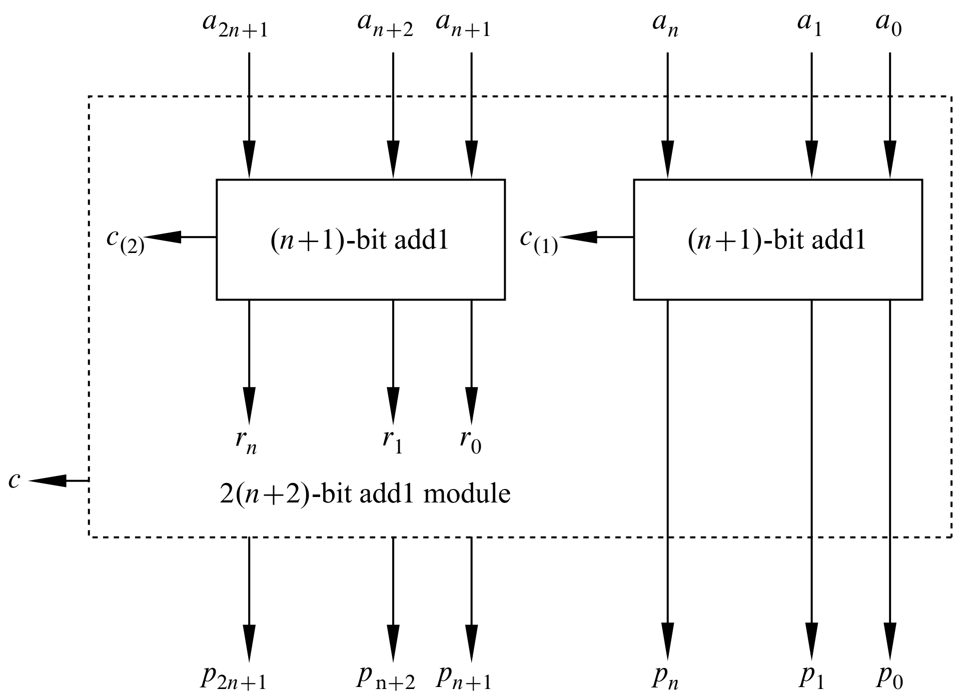


**Figure 1** Structure of a Double-size *add1* Module.

6.042J / 18.062J Mathematics for Computer Science
Spring 2015