Collaborators: None. Written Sources: Textbook

Problem 1

A *formula of set theory* is a predicate formula that only uses the predicate $x \in y$. The domain of discourse is the collection of sets, and $x \in y$ is interpreted to mean the set x is one of the elements in the set y.
For example, since x and y are the same set iff they have the same members, here's how we can express equality of x and y with a formula of set theory:

$$(x == y) ::= \forall z.(z \in x \text{ IFF } z \in y)$$

a. Explain how to write a formula Members(p, a, b) of set theory that means $p = \{a, b\}$.

$$\forall x.\, x \in p \text{ IFF } (x = a \text{ OR } x = b)$$

A *pair* (a, b) is simply a sequence of length two whose first item is a and whose second is b. Sequences are a basic mathematical data type we take for granted, but when we're trying to show how all of mathematics can be reduced to set theory, we need a way to represent the ordered pair (a, b) as a set. One way that will work is to represent (a, b) as

$$\text{pair}(a, b) ::= \{a, \{a, b\}\}$$

b. Explain how to write a formula Pair(p, a, b) of set theory that means $p = \text{pair}(a, b)$.

$$\text{Pair}(p, a, b) = \text{Members}(p, a, \{a, b\})$$

c. Explain how to write a formula Second(p, b) of set theory that means p is a pair whose second item is b.

$$\text{Second}(p, b) = \forall x, y.\, x \in p \text{ IFF } (x = y \text{ OR } x = \{y, b\})$$

Problem 2

Prove De Morgan's Law for set equality

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

by showing with a chain of IFF's that $x \in$ the left hand side IFF $x \in$ the right hand side. You may assume the proportional version of De Morgan's Law:

$$\text{NOT}(P \text{ AND } Q) = \overline{P} \text{ OR } \overline{Q}$$

$$\begin{aligned} x \in \overline{A \cap B} &\text{ IFF } x \in \overline{A \text{ AND } B} \\ &\text{ IFF } x \in \text{NOT}(A \text{ AND } B) \\ &\text{ IFF } x \in \overline{A} \text{ OR } x \in \overline{B} \\ &\text{ IFF } x \in \overline{A} \cup \overline{B} \end{aligned}$$

Problem 3

A *binary word* is a finite sequence of `0's` and `1's`. For example, `(1, 1, 0)` and (1) are words of length three and one, respectively. We usually omit the parentheses and commas in the descriptions of words, so the preceding binary words would just be written as `110` and `1`.
The basic operation of placing one word immediately after another is called *concatentation*. For example, the concatentation of `110` and `1` is `1101`, and the concatentation of `110` with itself is `110110`.

$$c = c_{(1)} \text{ AND } c_{(2)}$$

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 $c_{(1)}$.

Intuitively, if $c_{(1)}$ is zero, then p_{n+i} should be equal to the input a_{n+i}. Else, it should be equal to the current output r_{i-1}. Translating that into propositional logic, we have

$$\begin{aligned} P_{n+i} &= ((\text{NOT}(c_{(1)}) \text{ OR } r_{i-(n+1)}) \text{ AND } c_{(1)}) \text{ OR } ((c_{(1)} \text{ OR } a_{n+i}) \text{ AND } \text{NOT}(c_{(1)})) \\ &= (r_{i-(n+1)}) \text{ AND } c_{(1)}) \text{ OR } (a_{n+i} \text{ AND } \text{NOT}(c_{(1)})) \end{aligned}$$

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.

Since there are 4 operations to determine any P_{n+i} and there are $\log n$ levels, there are at most $4 \log n$ propositional operations which is significantly faster than conventional *add1* modules.