CS 70 Fall 2000 — Discrete Mathematics for CS — Papadimitriou/Russell/Sinclair — MT1 Sol

Solutions to Midterm 1

1. (16 pts.) Theorems and proofs

(a) (4 pts) Prove that if a and b are rational, then ab is rational.

Since a and b are rational they can be written as the ratio of integers a_1,a_2,b_1,b_2 as follows: $a = \frac{a_1}{a_2}$ and $b = \frac{b_1}{b_2}$. We then substitute these equalities in to ab to get $ab = \frac{a_1 b_1}{a_2 b_2}$. Because the integers are closed under multiplication the numerator and denominator of the right hand side are integers, and therefore ab is rational.

(b) (6 pts) Prove that if x is irrational, $x^{\frac{1}{3}}$ is irrational.

The contrapositive, " If $x^{\frac{1}{3}}$ is rational then x is rational." is logically equivalent to the statement in the problem and straightforward to prove. If $x^{\frac{1}{3}}$ is rational then by the first part of this problem, $x^{\frac{1}{3}}x^{\frac{1}{3}}$ is rational. Then since both $x^{\frac{1}{3}}$ and $x^{\frac{1}{3}}x^{\frac{1}{3}}$ are rational, again by the first part of the problem $x^{\frac{1}{3}}x^{\frac{1}{3}}x^{\frac{1}{3}} = x$ is rational.

(c) (6 pts) Write down the *converse* of the implication in part (b). State whether the converse is true or false, and prove your assertion.

The converse is: if $x^{\frac{1}{3}}$ is irrational, then x is irrational. A counter example is $x = 2$. It is important to note that this requires knowing that $2^{\frac{1}{3}}$ is in fact irrational, which was not proven in class, but can be proven in a similar fashion. This was not required for full credit on the problem.

2. (25 pts.) Induction

(a) (10) Prove by induction that $5^n - 1$ is divisible by 4 for all $n \geq 1$.

Base Case: $P(1)$: $5^1 - 1 = 4$ which is clearly divisible by 4.

Inductive Step: $P(n) \implies P(n+1)$: By the inductive hypothesis $5^n - 1$ is divisible by 4 so $5^n - 1 = 4k$ for some integer k. Now $5^{n+1} - 1 = 5(5^n - 1) + 4 = 5(4k) + 4 = 4(5k+1)$ so 4 divides $5^{n+1} - 1$.

(b) (10) Let $S(n) = \sum_{i=1}^{n} i$ denote the sum of the first n natural numbers. What is wrong with the following "proof" that $S(n) = \frac{1}{2}\left(n + \frac{1}{2}\right)^2$ for all $n \geq 1$.

Base Case: The claim is obviously true for n=1.

Inductive Step:

$$\begin{aligned}
S(n+1) &= S(n) + (n+1) \\
&= \frac{1}{2}(n+\frac{1}{2})^2 + n + 1 \qquad \text{by inductive hypothesis} \\
&= \frac{1}{2}(n^2 + n + \frac{1}{4}) + n + 1 \\
&= \frac{1}{2}(n^2 + 3n + \frac{9}{4}) \\
&= \frac{1}{2}((n+1)^2 + \frac{1}{2})
\end{aligned}$$

The claim that the base case is obviously true is false because $S(1) = \sum_{i=1}^{n} i = 1 \neq \frac{9}{8} = \frac{1}{2}(1+\frac{1}{2})^2$. The inductive step is valid.

3. (18 pts.) Satisfiability and All that

For each of the following Boolean expressions, decide if it is (i) valid; or (ii) satisfiable but not valid; or (iii) unsatisfiable. Justify all your answers.

(a) (6 pts) $(A \vee B) \wedge (B \vee C) \wedge (C \vee A)$; For $A = B = C = F$ the expression is false, and for $A = B = C = T$ the expression is true, so it is satisfiable but not valid.

(b) (6 pts) $(A \wedge \neg B) \vee (B \wedge \neg C) \vee (C \wedge \neg A)$; For $A = T$ and $B = F$ the expression is true and for $A = B = C = F$ The expression is false, so it is satisfiable but not valid.

(c) (6 pts) $A \wedge (\neg A \vee B) \wedge (\neg A \vee \neg B)$; For the expression to be true, A must be true, and therefore by examining the second clause B must be true. Finally this implies that the third clause is in fact false since A and B are both true. The conclusion is that the expression cannot be true, and is unsatisfiable. This can also be checked directly by a truth table.

4. (24 pts.) Recursion

The following algorithm, given as inputs a real number x and a natural number y, is supposed to output the value x^y.

```
algorithm exp($x, y$)
  if $y = 0$ then return(1)
  else
    $z :=$ exp($x, \lfloor \frac{y}{2} \rfloor$)
    if $y$ is even then return($z \times z$)
    else return($x \times z \times z$)
```

(a) (4 pts) Hand-turn the algorithm to compute exp(3, 7). Show your working.

The sequence of recursive calls made by the algorithm is as follows:

$$\exp(3,3); \quad \exp(3,1); \quad \exp(3,0).$$

The last of these is the base case, and is immediately evaluated to 1. Unwinding the recursion, the algorithm computes the following successive values of z:

$$3 * 1 * 1 = 3; \quad 3 * 3 * 3 = 27; \quad 3 * 27 * 27 = 2187.$$

This last number is indeed 3^7, as expected.

Almost everybody answered this correctly (modulo trivial arithmetic errors).

(b) (10 pts) Prove by induction on y that the algorithm is correct.

Let $P(y)$ be the proposition "Given any input of the form (x, y), where x is a real number, algorithm exp halts and returns the value x^y." Our goal is to prove $P(y)$ for all natural numbers y, by strong induction on y.

The base case, $y = 0$, is immediate: here the algorithm returns the value 1, which is correct since $x^0 = 1$ for all x.

Now consider some value $y \geq 1$. By the strong induction hypothesis, we may assume that a call of the form $\exp(x, y')$, for any $y' < y$, halts and returns the correct value $x^{y'}$. Now since $y \geq 1$, the algorithm will follow the first "else" clause and will evaluate the recursive call $z = \exp(x, \lfloor \frac{y}{2} \rfloor)$. But since $\lfloor \frac{y}{2} \rfloor$ is less than y, we may assume that this recursive call returns the correct value, i.e., that $z = x^{\lfloor \frac{y}{2} \rfloor}$. To finish off, we consider two cases (following the algorithm):

Case 1 y is even. In this case the algorithm returns

$$z * z = \left(x^{\lfloor \frac{y}{2} \rfloor}\right)^2 = \left(x^{\frac{y}{2}}\right)^2 = x^y.$$

Case 2 y is odd. In this case the algorithm returns

$$x * z * z = x\left(x^{\lfloor \frac{y}{2} \rfloor}\right)^2 = x\left(x^{\frac{y-1}{2}}\right)^2 = x x^{y-1} = x^y.$$

Thus in both cases the algorithm returns the correct value, x^y, which completes the proof of $P(y)$ for all y by induction.

This part was answered correctly, or almost correctly, by many people. The most common mistake was to try to prove $P(y)$ using $P(y-1)$, or equivalently to prove $P(y+1)$ using $P(y)$ (i.e., using simple induction rather than strong induction). Note that this is hopeless because the recursive call made by the algorithm involves the parameter $\lfloor \frac{y}{2} \rfloor$, not $y - 1$. An inductive proof of properties of a recursive procedure should always follow the recursive structure of the procedure itself.

(c) (10 pts) Prove, also by induction, that the number of multiplications performed by the algorithm is no more than $2b(y)$, where $b(y)$ is the number of bits in the binary representation of y.

Let $Q(y)$ be the proposition "Given any input of the form (x, y), where x is a real number, algorithm exp performs at most $2b(y)$ multiplications." We will prove $Q(y)$ by induction on y, following exactly the same pattern as we did in part (b).

The base case is again $y = 0$. Here the algorithm performs no multiplications, while $b(y) = 1$. So the base case certainly holds. Actually (as we'll see in a moment) it is convenient to define $b(y) = 0$ when $y = 0$. Note that the base case holds even with this more stringent definition of $b(0)$.

Now consider $y \geq 1$. By the strong induction hypothesis, we know that the recursive call $\exp(x, \lfloor \frac{y}{2} \rfloor)$ requires at most $2b(\lfloor \frac{y}{2} \rfloor)$ multiplications. Now it is convenient to observe that, for any natural number y, we have

$$b(\lfloor \tfrac{y}{2} \rfloor) = b(y) - 1. \qquad (*)$$

To see this, just note that the binary representation of $\lfloor\frac{y}{2}\rfloor$ is obtained by simply deleting the least significant bit of y. [Note: To make this correct when $y = 1$, we need to define $b(0) = 0$; this is why we did this above. We could avoid this little trick by handling the case $y = 1$ as an additional base case instead.]

Now we can consider the same two cases as we did in part (b).

Case 1 y is even. In this case the number of multiplications performed is at most $2b(\lfloor\frac{y}{2}\rfloor) + 1$, where the first term comes from the recursive call and the extra one is the multiplication to compute $z * z$. Using $(*)$, this evaluates to

$$2b(\lfloor\tfrac{y}{2}\rfloor) + 1 = 2(b(y) - 1) + 1 = 2b(y) - 1 < 2b(y).$$

Case 2 y is odd. In this case the number of multiplications is at most $2b(\lfloor\frac{y}{2}\rfloor) + 2$, where again the first term comes from the recursive call and the extra two from the computation $x * z * z$. Using $(*)$ again, this evaluates to

$$2b(\lfloor\tfrac{y}{2}\rfloor) + 2 = 2(b(y) - 1) + 2 = 2b(y).$$

Thus in both cases the number of multiplications is at most $2b(y)$. This completes the proof of $Q(y)$ for all y by induction. *People seemed to have a bit more trouble with this part than with part (b). Again, one of the common problems was trying to prove $Q(y)$ using $Q(y-1)$: as in part (b), this is doomed. Other people were confused about the relationship between the number of bits in the binary representations of y and $\lfloor\frac{y}{2}\rfloor$: see $(*)$ above.*

Some people tried to do an induction not on y itself, but on the number of bits in y. In other words, you are proving the proposition $R(n)$, which says "Given any input of the form (x, y), where x is a real number and y is a natural number whose binary representation has exactly n bits, algorithm exp performs at most $2n$ multiplications." This is fine — in fact it is arguably more elegant than the above proof. However, you still need to use $(*)$ to show that in the recursive call the number of bits decreases by one, so that the inductive hypothesis $R(n-1)$ can be applied. This is a simple induction, not a strong induction.

5. (20 pts.) Kowalski Normal Form

A Boolean expression is in *Kowalski Normal Form* (KNF) if it is a conjunction of implications, where the premise of each implication has a conjunction of variables and the conclusion of each implication is a disjunction of variables. (Note: *variables*, not *literals*.) For example, the expression $(B \wedge C) \implies (A \vee D)$ is in KNF.

(a) Prove that $(B \wedge C) \implies (A \vee D)$ is logically equivalent to $(A \vee \neg B \vee \neg C \vee D)$.

$$\begin{aligned}(B \wedge C) \implies (A \vee D) &\equiv \neg(B \wedge C) \vee (A \vee D) \quad \text{(defn of } \implies \text{)}\\ &\equiv (\neg B \vee \neg C) \vee (A \vee D) \quad \text{(de Morgan)}\\ &\equiv (\neg B \vee \neg C \vee A \vee D) \quad \text{(associativity)}\\ &\equiv (A \vee \neg B \vee \neg C \vee D) \quad \text{(commutativity)}\end{aligned}$$

This part was answered very well by almost everybody.

(b) Prove by induction that $\neg(X_1 \vee \cdots \vee X_n) \equiv (\neg X_1 \wedge \cdots \wedge \neg X_n)$.
The proof is by simple induction on n, the number of variables.

$P(n)$ is the assertion that $\neg(X_1 \vee \cdots \vee X_n) \equiv (\neg X_1 \wedge \cdots \wedge \neg X_n)$.
Base case: $n = 1$. The disjunction or conjunction of one expression is logically equivalent to that expression, hence $P(1)$ is the assertion that $\neg X_1 \equiv \neg X_1$, which is true.
Inductive step: prove $P(n) \implies P(n+1)$.
Inductive hypothesis: $\neg(X_1 \vee \cdots \vee X_n) \equiv (\neg X_1 \wedge \cdots \wedge \neg X_n)$.
To prove: $\neg(X_1 \vee \cdots \vee X_n \vee X_{n+1}) \equiv (\neg X_1 \wedge \cdots \wedge \neg X_n \wedge \neg X_{n+1})$.

$$\begin{aligned}
\neg(X_1 \vee \cdots \vee X_n \vee X_{n+1}) &\equiv \neg((X_1 \vee \cdots \vee X_n) \vee X_{n+1}) \quad \text{(associativity)}\\
&\equiv \neg(X_1 \vee \cdots \vee X_n) \wedge \neg X_{n+1} \quad \text{(simple de Morgan)}\\
&\equiv (\neg X_1 \wedge \cdots \wedge \neg X_n) \wedge \neg X_{n+1} \quad \text{(inductive hypothesis)}\\
&\equiv (\neg X_1 \wedge \cdots \wedge \neg X_n \wedge \neg X_{n+1}) \quad \text{(associativity)}
\end{aligned}$$

Almost everyone who did a simple induction on n got this right, although there were some rather Byzantine ways of dividing up n+1 literals into smaller pieces. Many people tried to do this as an induction over all Boolean expressions, which fails altogether because the assertion is just about a disjunction of variables, not all Boolean expressions. The failure was usually not noticed because the statement of $P(\cdot)$ was omitted*; as soon as one tries to state* $\forall b \in B\ P(b)$*, one realizes this doesn't make sense.*

(c) *Now prove that every Boolean expression is logically equivalent to an expression in KNF.*
The key step is to realize that KNF is a conjunction of implications like the one in part (a). Part (a) shows that the implication can be converted to a disjunction of literals, so we're looking at a conjunction of disjunctions of literals—i.e., CNF. The formal argument goes as follows:
Every Boolean expression b is equivalent to an expression in CNF, $CNF(b)$, which is a conjunction of clauses $C_1 \wedge \cdots \wedge C_m$. Each clause C_i contains positive and negative literals; without loss of generality, call these $X_1, \ldots, X_k$ and $\neg Y_1, \ldots, \neg Y_l$. By commutativity, C_i can be written as $(\neg Y_1 \vee \cdots \neg Y_l \vee X_1 \vee \cdots X_k)$. By associativity and the dual version of the result in (b), this is equivalent to $\neg(Y_1 \wedge \cdots \wedge Y_l) \vee (X_1 \vee \cdots X_k)$, which in turn is equivalent to $(Y_1 \wedge \cdots \wedge Y_l) \implies (X_1 \vee \cdots X_k)$, i.e., a KNF clause. Hence, every Boolean expression is equivalent to an expression in KNF.

Almost everyone who saw the connection to CNF got this right; the only problem was proving in the wrong direction*—i.e., proving that every KNF expression is equivalent to a CNF expression. Obviously, this could be true even if the set of KNF expressions is empty!*

Others made valiant attempts to prove the assertion by structural induction over Boolean expressions. This is a proper use of induction in this case, but no one managed to get it quite right.