Homework 2:
Boolean Logic and Related Topics

Homework due in class on the date indicated above. Late homeworks will be penalized by $5 \cdot 2^{-n-1} \%$, where $n$ is the number of days the assignment is late. Please indicate which problems, if any, took extra time.

1. Insurance policy design

You are presented with a set of requirements under which an insurance policy will be issued. The applicant must be:

1. A married female 25 years or over, or
2. A female under 25, or
3. A married male under 25 who has not been involved in a car accident, or
4. A married male who has been involved in a car accident, or
5. A married male 25 years or over who has not been involved in a car accident.

i. Define

- $w =$applicant has been involved in a car accident
- $x =$applicant is married
- $y =$applicant is male
- $z =$applicant is under 25

and write the expression in $w, x, y, z$ for the company rules on issuance.

ii. Simplify the expression and hence provide a simpler set of rules (in English).

2. De Morgan and the importance of proper notation

Augustus de Morgan was a mathematician born to a Scottish family in Madras (Chennai), India in 1806. His work involved expanding Boole’s early work on the laws of logic, which were really a rediscovery of much that was already known by the classical Greeks as well as medieval philosophers like William of Ockham. De Morgan is best known for expressing the rules for negating conjunctions and disjunctions:

\[
\neg(p \land q) = \neg p \lor \neg q
\]

\[
\neg(p \lor q) = \neg p \land \neg q
\]

De Morgan was handicapped by the fact that he did not have a symbol ($\neg$) for negation. He had to render the complement of a boolean literal $x$ by writing it as a capital $X$. While this may seem attractive and elegant because it succinctly expresses that negation is its own inverse, it has a severe drawback. What is the drawback?

3. Boolean algebras

We can summarize the properties of a boolean algebra as follows:

A boolean algebra $B$ is a set of elements $a, b, c, \ldots$, together with two binary operations $\land$ and $\lor$, which satisfy the idempotent, commutative, absorption, and associative laws, and are mutually distributive. $B$ contains two bounds, $\top$ and $\bot$, which are the greatest and least elements, respectively. $B$ has a unary operation of complementation, which assigns to every element in $B$ its complement.

Show that:
i. The complement $\neg a$ of any element $a \in B$ is unique.

ii. No boolean algebra of three elements $\{\bot, \top, \neg\}$ exists.

iii. The partial ordering of all integers that divide the number 30 is a boolean algebra of eight elements. Specify the results of the three operations $(\lor, \land, \neg)$ on their possible operands. What is the identification of the operations to operations on integers?

iv. Consider the set of integers that divide the integer $N$ as in the previous subpart. State a sufficient condition on $N$ for this set to form a (nontrivial) boolean algebra.

4. Simplify 10%

Use the rules from lecture and the class notes to simplify the following expressions. You may want to keep in mind the useful fact that $a \Rightarrow b \equiv \neg a \lor b$. (Note that here, as elsewhere, we follow Dijkstra in applying $\equiv$ only to boolean values and in taking $\equiv$ as having lowest binding power of all operators. Otherwise, $= \equiv$ mean the same thing applied to booleans.)

Example. $p \lor (\neg p \lor \neg q)$

Answer.

\[
p \lor (\neg p \lor \neg q) \equiv (p \lor \neg p) \lor \neg q \quad \text{(associativity)}
\]
\[
\equiv \text{true} \lor \neg q \quad \text{(excluded middle)}
\]
\[
\equiv \text{true} \quad \text{(\lor-simplification)}
\]

i. $\neg p \lor (p \lor q)$

ii. $p \Rightarrow p \lor q$

iii. $p \Rightarrow p \land q$

iv. $p \land (p \lor q)$

v. $(p \Rightarrow q) \land (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$

5. The Liar’s Paradox 10%

Epimenides was a Cretan mystic that lived in the sixth century B.C. Legend has it that Epimenides woke up in a bad mood one day—angry at his fellow Cretans, one must assume—and said,

Cretans always lie.

(The legend also has it that he expressed himself in a form of verse known as dactylic hexameter catalectic, but since all that is known about Epimenides is from second-hand sources, this is hard to verify.)

This statement has confused many people. The Apostle Paul writes of Epimenides, (King James Authorized Version, Titus 1:12–13),

One of themselves, even a prophet of their own, said, the Cretians are always liars, evil beasts, slow bellies. This witness is true. Therefore rebuke them sharply…

We can rewrite Epimenides’s statement in the form

\[
ES \overset{\text{def}}{=} \forall c : c \text{ is a Cretan : } (\forall t :: c \text{ is lying at time } t).
\]

i. For the statement $ES$ to be paradoxical, its truth would have to imply its falsehood ($ES \Rightarrow \neg ES$) and its falsehood would have to imply its truth ($\neg ES \Rightarrow ES$). Show that this need not be the case.
ii. Was Paul mistaken when he wrote, “This witness is true”?

iii. Are there conditions under which $ES$ would truly be paradoxical?

A true paradox would be a statement like

$$TP \overset{\text{def}}{=} \text{This sentence is false.}$$

$TP$ cannot be true, for then it would be false; and vice versa. The earliest recorded statements of this form are from the fourth century B.C. More recently (in 1930), Kurt Gödel showed that any sufficiently powerful logical system contains paradoxes of this kind; interestingly, not much power is required, as Gödel’s system included only statements involving the natural numbers with addition and multiplication and basic rules of logic. But in propositional calculus the paradoxes appear to be ruled out by the “law of the excluded middle” (*tertium non datur*), which holds that

$$p \lor \neg p \equiv \text{true.}$$

iv. Assuming that the basic propositions (literals) $p, q, \ldots$ contain no paradoxes (i.e., that the law of the excluded middle holds for them), is it possible to create a paradox using only the connectives $\land, \lor, \Rightarrow, \equiv$?

6. Quantifiers

Before we leave logic completely behind us, let us take a last look at quantifiers. Express the following statements using one or more of the quantifiers $\forall, \exists,$ and $N$. (Remember that $N$ is the “counting quantifier”.) $N$ is the set of natural numbers; $Z$ is the set of integers. Read through the examples carefully; note that most of them can be stated in several different ways.

*Example 1.* “For all $x$, $x + 1 > x$.”

*Answer.* $\forall x :: x + 1 > x$

*Example 2.* “There are five odd natural numbers less than 10.”

*Answer.* $(\exists n :: n \in N \land n < 10 : (\exists k :: k \in Z : n = 2k + 1)) = 5$

*Example 3.* “The largest element in the array $a$ is equal to $p$.”

*Answer.* $(\forall i :: a[i] \leq p) \land (\exists i :: a[i] = p)$

*Example 4.* “The index of the largest element in the array $a$ is equal to $j$.”

*Answer.* $\forall i :: a[i] \leq a[j]$

Now do the following.

i. Define a predicate $perm(a, b)$ that says that array $a$ and $b$ are permutations of each other.

ii. Define a predicate $sorted(a, b)$ that says that $a$ is a sorted permutation of $b$ (assume $a$ and $b$ are arrays of numbers). You can use the predicate $perm(a, b)$ in your definition.

7. The “predicate transformer” for weakest preconditions $wp$

The predicate transformer $wp$ was introduced by E. W. Dijkstra in 1976. The value of $wp(S, Q)$ for a predicate $Q$ is the weakest predicate $P$ such that the Hoare triple $\{P\}S\{Q\}$ holds under total correctness, i.e., $P$ is the weakest statement that can be made about the state of the system before activating statement $S$ that guarantees that $S$ will terminate in a state satisfying $Q$. (As in class, we omit explicit mention of the state $\Sigma$ in the predicates $P(\Sigma)$ and $Q(\Sigma)$ unless necessary.)
It should be obvious why \textit{wp} is called a “predicate transformer”: it takes the predicate that you want to be true after a statement has executed and turns it into the predicate that must have been true before the statement was activated. This is actually what makes it more useful than the Hoare triple; you can start with the postcondition of a large program and apply \textit{wp} repeatedly, moving backwards through the program text, until you get to the precondition. If the precondition matches the one given in the specification, or if it is implied by the specification’s precondition, then you have (1) written a program \textit{(backwards!)} that conforms to the given specification; and (2) simultaneously produced a proof of correctness for that program.

Please calculate the following; \* means multiplication, sqrt means square root, \(x\) and other variables are integers.

\begin{enumerate}
\item \(\text{wp}(x := 2 \times x, x > 10)\)
\item \(\text{wp}(x := y, x > 10)\)
\item \(\text{wp}(x := \text{sqrt}(y), x > 10)\)
\item \(\text{wp}(x := y \times y, x > 100)\)
\item \(\text{wp}(x := y \times z, \text{true})\)
\item \(\text{wp}(x := y / z, \text{true})\) (Careful!)
\item \(\text{wp}(x := y \times y, x = -1)\)
\end{enumerate}

8. Nondeterminism

Consider a program that when run twice, starting in the same state \(\Sigma_0\), may give different answers. It is a powerful feature of the \textit{wp} theory that it can handle such programs without any further embellishment. Remember from lecture that if we are interested in a postcondition \(X\), activating \(S\) in some state \(\Sigma_0\) will lead to precisely one of three possible results:
\begin{enumerate}
\item Termination in a state where \(X\) holds, “finally \(X\)”
\item Termination in a state where \(\neg X\) holds, “finally \(\neg X\)”
\item An infinite loop or some other form of non-termination (e.g., the execution of \texttt{abort} somewhere in the program), “eternal”
\end{enumerate}
(Each computation under the control of \(S\) has to fall into one of the three categories, but two separate activations of \(S\) do not necessarily have to fall into the same category.)

Now we can study more closely what \textit{wp} and \textit{wlp} mean by considering particular evaluations of the two predicate transformers, e.g.,

\(\textit{wp}(S, \text{true})\) holds precisely in those initial states for which each computation under \(S\) belongs either to the class “finally \(X\)” or the class “finally \(\neg X\)”

\(\textit{wlp}(S, \neg X)\) holds precisely in those initial states for which each computation under \(S\) belongs either to the class “eternal” or the class “finally \(\neg X\)”

\begin{enumerate}
\item We call a program \(S\) \textit{deterministic} if when it is activated in any particular state \(\Sigma\), its behavior is always the same, that is, the computation that unfolds always falls into the same of the three categories for any predicate \(X\) on the final state. Write an expression in \textit{wp}(\(S, \cdot\)) and \textit{wlp}(\(S, \cdot\)) that is true if and only if \(S\) is deterministic.
\item Recall the semantics of the statements introduced in lecture:
\texttt{havoc}. \(\textit{wlp}(\texttt{havoc}, X) \equiv \forall \Sigma :: X, \textit{wp}(\texttt{havoc}, \text{true}) \equiv \text{true}\)
\end{enumerate}
abort. \( \text{wp}(\text{abort}, X) \equiv \text{true} \), \( \text{wp}(\text{abort}, \text{true}) \equiv \text{false} \)

skip. \( \text{wp}(\text{skip}, X) \equiv X \), \( \text{wp}(\text{skip}, \text{true}) \equiv \text{true} \)

For each of havoc, abort, and skip, determine (using the formula you developed in the previous part) whether or not the statement is deterministic.

9. Continuation-passing style in Scheme

In lecture, we have discussed how functional programming languages such as pure LISP reduce computer programming to “known concepts” of mathematical functions. In this exercise we will explore to what extent this position is an exaggeration.

In Artificial Intelligence Laboratory Memo AIM-353 “LAMBDA: The Ultimate Imperative,” Steele and Sussman describe a programming technique known as Continuation-Passing Style, or CPS for short, with which one can mechanically convert an imperative program to a functional program. The report is available at (among other places, including the MIT Libraries):

http://repository.readscheme.org/ftp/papers/ai-lab-pubs/AIM-353.pdf

The basic idea of CPS is the following. Consider any imperative program. Introduce a label for each statement. Recode changes of control flow with a combination of IF and GO TO statements. Finally, convert the program to Scheme while minding the following two rules:

1. Pass the values of all variables of the program as parameters to lambda.

2. Explicitly transfer control to the next statement by calling the lambda expression corresponding to the label in the imperative program; pass the current value of all the program variables as parameters to the lambda.

A good example is on page 8 of AIM-353.

i. Consider the following procedure written in the Modula-3 programming language (no language-specific features are used):

PROCEDURE Example(a : INTEGER) =
    VAR
        q, r : INTEGER;
    BEGIN
        q := 0; r := a + 1;
        WHILE r >= 4 DO
            q := q + r DIV 4; r := r DIV 4 + r MOD 4
        END;
        Write(q, r-1)
    END Example;

Convert this code to Scheme in CPS style. The conversion should be very mechanical, as described above. Use modulo for MOD, div for DIV. Prepare your Scheme environment by typing (or putting in your file that you load) the following into the interpreter:

(require-modules "display")

Now you can print the result with the following code:

(dis q "," r dnl)

(dnl stands for “delimiter new line.”)

Finally, note that in modern Scheme, Steele and Sussman’s LABELS is now written letrec.
ii. What does the code compute?

iii. Is the code made clearer by the translation into functional form?

10. **Time management**

   For 2% assignment credit, keep track of how much time you spend on each problem and turn that in with your other answers. We need the feedback to improve the class.

11. **Extra Credit: Suggestions to improve homework assignments**

   You will receive an extra 2% assignment credit, up to a maximum of 10% total, for each suggestion you make for improving this assignment or HW1 that we accept.