CSE 311: Foundations of Computing I

Section 1: Logic

0. Exclusive Or

For each of the following, decide whether inclusive-or or exclusive-or is intended:

(a) Experience with C or Java is required.

(b) Lunch includes soup or salad.

(c) Publish or perish

(d) To enter the country you need a passport or voter registration card.

1. Translations

For each of the following, define propositional variables and translate the sentences into logical notation.

(a) I will remember to send you the address only if you send me an e-mail message.

(b) If berries are ripe along the trail, hiking is safe if and only if grizzly bears have not been seen in the area.

(c) Unless I am trying to type something, my cat is either eating or sleeping.

2. Teatime

Consider the following sentence:

> If I am drinking tea then I am eating a cookie, or, if I am eating a cookie then I am drinking tea.

(a) Define propositional variables and translate the sentence into an expression in logical notation.

(b) Fill out a truth table for your expression.

(c) Based on your truth table, classify the original sentence as a contingency, tautology, or contradiction.

3. Truth Tables

Write a truth table for each of the following:

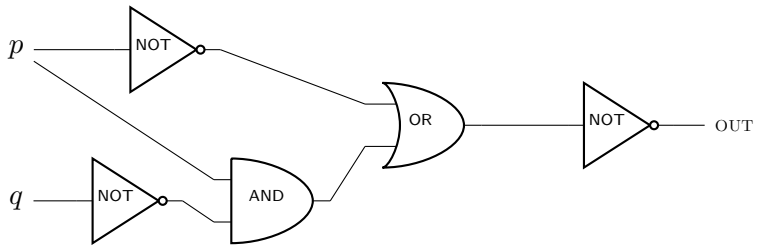
(a) $(p \oplus q) \vee (p \oplus \neg q)$

(b) $(p \vee q) \rightarrow (p \oplus q)$

(c) $p \leftrightarrow \neg p$

4. Circuitous

Translate the following circuit into a logical expression.



5. The Curious Case of The Lying TAs

A new UW student wandered around the Paul Allen Center on their first day at UW. They found (as many do) that there is a secret room in its basements. On the door of this secret room is a sign that says:

> `All ye who enter, beware! Every inhabitant of this room is either a TA who always lies or a student who always tells the truth!`

The UW student somehow magically divines that this sign is telling the truth and enters the room. Now, consider the following scenarios:

(a) After entering the room, two inhabitants suddenly walk up to the UW student. One of them one of them says: "*At least one of us is a TA*".

Our goal is to figure out whether each inhabitant is a TA or a student. However, we also don't want to sit down and try and reason out a solution; that's way too much work. Instead, let's write a program to find the answer for us! (After all, aren't we supposed to be computer scientists or something?)

Your job is to model this puzzle using Z3. We've given you some helper functions (documented on the next page) to help you do so. Once your model is complete, we'll run Z3's solver to find the solution(s) to this puzzle for us – you can assume you're given the code that does this.

Model this scenario in the following method. Hint: your method should consist of a series of calls to the `assume(...)` method.

```
public static void modelPartA(BoolExpr xIsTa, BoolExpr xIsStudent,
                              BoolExpr yIsTa, BoolExpr yIsStudent) {
    // Your code here
}
```

(b) Now, consider the same scenario as part (a), only this time three inhabitants walk up to the student. Model this new scenario:

```
public static void modelPartB(BoolExpr xIsTa, BoolExpr xIsStudent,
                              BoolExpr yIsTa, BoolExpr yIsStudent,
                              BoolExpr zIsTa, BoolExpr zIsStudent) {
    // Your code here
}
```

(c) What if n inhabitants walk up to the student? Model this situation.

```
// Preconditions: n == isTa.length and n == isStudent.length and n >= 2
public static void modelPartC(int n, BoolExpr[] isTa, BoolExpr[] isStudent) {
    // Your code here
}
```

(d) Let's consider a new scenario. Suppose three inhabitants walk up and surround the UW student. One of them says: "*Every TA in this circle has a TA to her immediate right*". Model this situation:

```
public static void modelPartD(BoolExpr xIsTa, BoolExpr xIsStudent,
                              BoolExpr yIsTa, BoolExpr yIsStudent,
                              BoolExpr zIsTa, BoolExpr zIsStudent) {
    // Your code here
}
```

You should use the following API to help you solve this problem:

Note: this API is the same one you are asked to use in one of the problems on homework 1. We've omitted a few irrelevant methods for brevity.

`public static void` **`assume`**`(BoolExpr expr)`

This method tells the solver to assume that the provided expression, `expr`, is true. The solver will *find* the solution, all you have to do is specify what a solution looks like.

`public static BoolExpr` **`and`**`(BoolExpr a, BoolExpr b)`

This method returns an *unevaluated* expression that represents the assertion $a \wedge b$.

`public static BoolExpr` **`or`**`(BoolExpr a, BoolExpr b)`

This method returns an *unevaluated* expression that represents the assertion $a \vee b$.

`public static BoolExpr` **`xor`**`(BoolExpr a, BoolExpr b)`

This method returns an *unevaluated* expression that represents the assertion $a \oplus b$.

`public static BoolExpr` **`iff`**`(BoolExpr a, BoolExpr b)`

This method returns an *unevaluated* expression that represents the assertion $a \leftrightarrow b$.

`public static BoolExpr` **`implies`**`(BoolExpr a, BoolExpr b)`

This method returns an *unevaluated* expression that represents the assertion $a \rightarrow b$.

Note: we've given you this method for this section problem, but you will need to implement yourself on your homework!

`public static BoolExpr` **`not`**`(BoolExpr a)`

This method returns an *unevaluated* expression that represents the assertion $\neg a$.