CL exercise for
Tutorial 8

Introduction

Objectives

In this tutorial, you will:

- learn to apply the Tseytin transformation
- use the arrow rule to count satisfying valuations

Tasks

Exercises 1 and 2 are mandatory. Exercise 3 is optional.

Submit

a file called `cl-tutorial-8` with your answers (image or pdf).

Deadline

16:00 Tuesday 15 November

Reminder

Good Scholarly Practice

Please remember the good scholarly practice requirements of the University regarding work for credit.

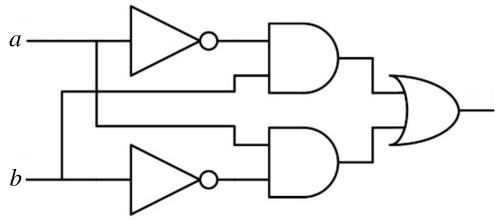
You can find guidance at the School page

https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

Exercise 1 –mandatory–-marked–

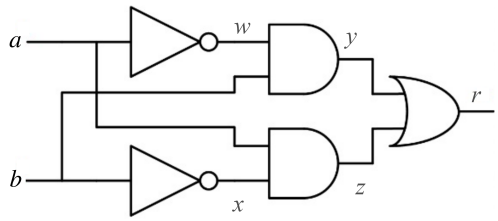
Consider the following circuit:



Give an equivalent logical expression.

Apply the Tseytin transformation to give an equisatisfiable CNF expression.

Solution to Exercise 1 We start by labelling the internal wires:



Writing down the list of equivalences and their corresponding CNF representations gives:

$$
\begin{array}{ll}
w \leftrightarrow \neg a: & (a \vee w) \wedge (\neg w \vee \neg a) \\
x \leftrightarrow \neg b: & (b \vee w) \wedge (\neg x \vee \neg b) \\
y \leftrightarrow w \wedge b: & (\neg y \vee w) \wedge (\neg y \vee b) \wedge (\neg w \vee \neg b \vee y) \\
z \leftrightarrow a \wedge x: & (\neg z \vee a) \wedge (\neg z \vee x) \wedge (\neg a \vee \neg x \vee z) \\
r \leftrightarrow y \vee z: & (\neg y \vee r) \wedge (\neg z \vee r) \wedge (\neg r \vee y \vee z)
\end{array}
$$

We then take the conjunction of these, set r to 1, and simplify, and we get:

$$
\begin{array}{l}
(a \vee w) \wedge (\neg w \vee \neg a) \\
\wedge\, (b \vee x) \wedge (\neg x \vee \neg b) \\
\wedge\, (\neg y \vee w) \wedge (\neg y \vee b) \wedge (\neg w \vee \neg b \vee y) \\
\wedge\, (\neg z \vee a) \wedge (\neg z \vee x) \wedge (\neg a \vee \neg x \vee z) \\
\wedge\, (y \vee z)
\end{array}
$$

Exercise 2 –mandatory–-marked–

Read Chapter 23 (*Counting Satisfying Valuations*) of the textbook.

Use the arrow rule to count the number of satisfying assignments for the CNF expression

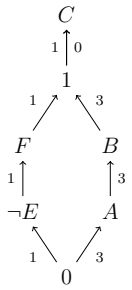
$$(E \vee F) \wedge (\neg A \vee B) \wedge C$$

Solution to Exercise 2

The CNF expression is equivalent to the following conjunction of implications:

$$(\neg E \to F) \wedge (A \to B) \wedge (1 \to C)$$

These give the following diagrams of upward-pointing implications:



So $0 + 3 + 3 + 3 = 9$ combinations of values.

Exercise 3 –optional—marked–

A *boolean algebra* is a set B containing elements **0** and **1**, together with operations $\wedge$, $\vee$ and $\neg$ that satisfy the boolean algebra axioms on slide 4 of the week 8 lectures. The set $\mathbb{B} = \{0, 1\}$ with the usual operators is the simplest (non-trivial) boolean algebra.

For example, $\mathbb{B} \times \mathbb{B}$ with $\mathbf{0} = (0, 0)$, $\mathbf{1} = (1, 1)$ and pointwise operators (that is, $(a, b) \wedge (c, d) = (a \wedge c, b \wedge c)$ etc.) is a boolean algebra.

On slide 2, we saw several (though not all) of the 16 possible binary boolean operators: in other words, functions $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$.

Show how to view the set of functions $\mathbb{F} = \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ as a boolean algebra: that is, identify the **0** and **1** elements of $\mathbb{F}$, and define the $\wedge$, $\vee$ and $\neg$ operations on elements of $\mathbb{F}$.

If you find this exercise rather abstract, here is a Haskell presentation of it. However, please submit written answers, not Haskell code.

Haskell prefers to use curried functions, so instead of $\mathbb{F} = \mathbb{B} \times \mathbb{B} \to \mathbb{B}$, we'll use $\mathbb{G} = \mathbb{B} \to \mathbb{B} \to \mathbb{B}$. While $\mathbb{F}$ and $\mathbb{G}$ are not the same, they are *isomorphic*.

Define the type

```
type TwoBool = Bool -> Bool -> Bool
```

Note that the standard Haskell boolean operators `&&` and `||` have type `TwoBool`.

Your task is to complete the following Haskell code so that the defined variables and functions 'do the right thing' as defined above:

```
type TwoBool = Bool -> Bool -> Bool

twoTrue :: TwoBool
twoTrue _ _ = undefined

twoFalse :: TwoBool
twoFalse _ _ = undefined

twoNot :: TwoBool -> TwoBool
twoNot x a b = undefined

(&&&) :: TwoBool -> TwoBool -> TwoBool
(x &&& y) a b = undefined

(|||) :: TwoBool -> TwoBool -> TwoBool
(x ||| y) a b = undefined
```

The definitions are *very* short, and you have done something similar in FP tutorial 6, and in CL tutorial 3 ex. 7.

Once you've produced code to your satisfaction, translate it back to mathematical statements about $\mathbb{F}$.

Solution to Exercise 3

The point of this exercise is to require a bit of abstract thinking, and maybe even reading outside.

The answer is, of course, pointwise lifting the operators:

0/1 are the constant 0/1 functions; $(f \wedge g)(x, y) = f(x, y) \wedge g(x, y)$ and so on. This guarantees that the axioms still hold.

They have already seen lifting, as we used it to define boolean operators on predicates, that is on unary functions $X \to \mathbb{B}$. So extending it to binary functions should be straightforward ... but I suspect the conceptual jump may be too high for some.

To get the mark, it suffices to give the operators. If any student gives the argument to show that what they've done actually is a boolean algebra, that's bonus bonus, and can cancel out slips in the mandatory part.