Adequate set of connectives, logic gates and circuits

Lila Kari

University of Waterloo
We have mentioned so far one unary and four binary connectives. There are many more unary and binary connectives, and also $n$-ary connectives, for $n > 2$.

- How many unary connectives?
- How many binary connectives?
## Binary connectives

Adequate set of connectives, logic gates and circuits

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>$g_3$</th>
<th>$g_4$</th>
<th>$g_5$</th>
<th>$g_6$</th>
<th>$g_7$</th>
<th>$g_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$g_9$</th>
<th>$g_{10}$</th>
<th>$g_{11}$</th>
<th>$g_{12}$</th>
<th>$g_{13}$</th>
<th>$g_{14}$</th>
<th>$g_{15}$</th>
<th>$g_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Which binary connectives do you recognize?

**Question:** How many $n$-ary connectives are there?

For an $n$-ary connective (that involves $n$ variables), the truth table has $2^n$ rows, and the number of possible columns equals the number of possible binary numbers of length (height) $2^n$.

**Answer:** $2^{(2^n)}$
Adequate set of connectives

- **Definition**: Any set of connectives with the capability to express all truth tables is said to be **adequate**.
- Post observed that the set of **standard connectives**, \{\neg, \land, \lor, \to, \leftrightarrow\}, is adequate.

Emil Post, 1897-1954
Proving adequacy of a connective set

- We can show that a new set $S$ of connectives is adequate if we can express all standard connectives in terms of $S$.
- This is achieved by “translating” all the standard connectives in terms of the new connectives in $S$, by using tautological equivalences.
- **Example:** $A \rightarrow B$ and $\neg A \vee B$ are tautologically equivalent.
- This means that $\rightarrow$ is definable in terms of (“is reducible to”; “can be expressed in terms of”) $\neg$ and $\vee$.
- Similarly, $\vee$ is definable in terms of $\neg$ and $\rightarrow$ because $A \vee B$ is tautologically equivalent to $\neg A \rightarrow B$. 
Theorem. \( \{\neg, \land, \lor\} \) is an adequate set of connectives.

Proof. We can use the theorem that states that any formula is equivalent to a formula in DNF. Alternatively, observe that, for any formulas \( A, B \),

\[
\neg A \models \neg A
\]

\[
A \land B \models A \land B
\]

\[
A \lor B \models A \lor B
\]

\[
A \rightarrow B \models \neg A \lor B
\]

\[
A \leftrightarrow B \models (\neg A \lor B) \land (\neg B \lor A)
\]

All the five standard connectives can be expressed in terms of \( \neg, \land, \lor \), therefore \( \{\neg, \land, \lor\} \) is an adequate set of connectives.

Corollary \( \{\neg, \land\} \), \( \{\neg, \lor\} \), and \( \{\neg, \rightarrow\} \) are adequate.
Schroder showed in 1880 that each of the standard connectives is definable in terms of a single binary connective \( \downarrow \), where the truth table associated with \( \downarrow \) is

\[
\begin{array}{ccc}
p & q & p \downarrow q \\
1 & 1 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{array}
\]
Proof of adequacy

We can express the standard connectives in terms of $\downarrow$ by

\[\neg p \equiv p \downarrow p\]
\[p \land q \equiv (p \downarrow p) \downarrow (q \downarrow q)\]
\[p \lor q \equiv (p \downarrow q) \downarrow (p \downarrow q)\]
\[p \rightarrow q \equiv ((p \downarrow p) \downarrow q) \downarrow ((p \downarrow p) \downarrow q)\]
\[p \leftrightarrow q \equiv ((p \downarrow p) \downarrow q) \downarrow ((q \downarrow p) \downarrow p).\]

Thus it follows that a single connective $\downarrow$ (also called NOR) is adequate.

Consequently, to test a given $S$ for being adequate it suffices to test if $\downarrow$ can be expressed by $S$. 
Sheffer stroke

In 1913 Sheffer showed that the Sheffer stroke with associated truth table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

is another single binary connective (also called NAND) in terms of which the standard connectives can be expressed.
How do we show that a given set $S$ of connectives is not adequate? Inadequacy is proven by showing that some standard connective cannot be expressed by using the connectives in $S$.

Example. The set $S = \{ \land \}$ is not adequate.

Proof. To see this, note that a formula depending on only one variable and which uses only the connective $\land$ has the property that its truth value for a value assignment that makes $p = 0$ is always 0.

In order to define the negation $\neg p$ in terms of $\land$, there should exist a formula $f$ depending on the variable $p$ and using only the connective $\land$ such that $\neg p \models f$.

However, for a value assignment $\nu$ such that $\nu(\neg p) = 1$, we have $\nu(p) = 0$ and therefore $\nu(f) = 0$, which shows that $\neg p$ and $f$ cannot be tautologically equivalent.
A ternary connective

Let us use the symbol $\tau$ for the ternary connective whose truth table is given by

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$r$</th>
<th>$\tau(p, q, r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
A ternary connective

Note that, for any value assignment $\nu$, we have

$$\nu(\tau(p, q, r))$$

equals $\nu(q)$ if $\nu(p) = 1$, and equals $\nu(r)$ if $\nu(p) = 0$.

This is the familiar if-then-else connective from computer science, namely

$$\text{if } p \text{ then } q \text{ else } r$$
We can consider now propositional logic based not upon the five common connectives, but upon any adequate set of connectives, for instance $\{\neg, \land\}$.

Let $\mathcal{L}_0^p$ be a sublanguage of $\mathcal{L}^p$ obtained by deleting from $\mathcal{L}^p$ the three connectives $\lor, \rightarrow, \leftrightarrow$, and let $\text{Form}(\mathcal{L}_0^p)$ be the set of formulas of $\mathcal{L}_0^p$.

**Theorem.** $\text{Form}(\mathcal{L}_0^p) = \text{Form}(\mathcal{L}^p)$. 
Proof.

Obviously, $\text{Form}(\mathcal{L}_0^p) \subseteq \text{Form}(\mathcal{L}^p)$.

Conversely, for $A \in \text{Form}(\mathcal{L}^p)$ we define (by recursion) its translation $A_0$ into $\mathcal{L}_0^p$ as follows:

$A_0 = A$ for atomic variable $A$,

$(\neg A)_0 = \neg A_0$,

$(A \land B)_0 = A_0 \land B_0$,

$(A \lor B)_0 = \neg (\neg A_0 \land \neg B_0)$,

$(A \rightarrow B)_0 = \neg (A_0 \land \neg B_0)$,

$(A \leftrightarrow B)_0 = (A \rightarrow B)_0 \land (B \rightarrow A)_0 = \neg (A_0 \land \neg B_0) \land \neg (\neg A_0 \land B_0)$.
Boolean Algebra and Logic Circuits

- George Boole (1815 - 1864)

- Author of the book “The Laws of Thought”: introduced propositional logic, algebra of logic

  “The design of the following treatise is to investigate the fundamental laws of those operations of the mind by which reasoning is performed; to give expression to them in the symbolical language of Calculus, and upon this foundation to establish the science of Logic and construct its method.”

- Boolean algebra has been fundamental in the development of digital electronics.
Boolean Algebra

A Boolean algebra is a set $B$ with two binary operations $+$ and $\cdot$, elements 0 and 1, and a unary operation $\overline{\cdot}$, such that the following properties hold for all $x, y, z$ in $B$:

- **Identity laws:** $x + 0 = x$ and $x \cdot 1 = x$.
- **Domination laws:** $x + \overline{x} = 1$, $x \cdot \overline{x} = 0$.
- **Associative laws:** $(x + y) + z = x + (y + z)$, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
- **Commutative laws:** $x + y = y + x$, $x \cdot y = y \cdot x$
- **Distributive laws:** $x + (y \cdot z) = (x + y) \cdot (x + z)$ and $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

Using the laws given in this definition, it is possible to prove many other laws that hold for every Boolean algebra.
Boolean algebra examples

• The set of propositional formulas in \( n \) variables, with the \( \lor \) and \( \land \) operators, \( 1 \) and \( 0 \), and the \( \neg \) operator, satisfy all the properties of a Boolean algebra.

• The set of subsets of a universal set \( U \), with the union and intersection operations, the empty set and the universal set, and the set complementation operator, is a Boolean algebra.

• So, to establish results about propositional logic, and sets, we need only prove results about abstract Boolean algebras.
Boolean algebra and computers

- Boolean algebra is used to model the circuitry of electronic devices, including electronic computers.
- Each input and output of such a device can be thought of as a member of the set \( \{0, 1\} \).
- An electronic computer is made up of a number of circuits.
- Each circuit can be designed using the rules of Boolean algebra.
- The basic elements of circuits are called logic gates.
- A gate is an electronic device that operates on a collection of binary inputs and produces a binary output.
The transistor - a simplified model

- Logic gates are physically implemented by transistors.
- A **transistor** is simply a **switch**, it can be in an **off** state, which does not allow electricity to flow, or in an **on** state, in which electricity can pass unimpeded.
- Each transistor contains three lines: **two input lines** and **one output line**. The first input line, called the **control line**, is used to open or close the switch inside the transistor.
How the transistor works

Adequate set of connectives, logic gates and circuits
How the transistor works

- If we set the input value on the control line to a 1 by applying a sufficient amount of voltage, the switch closes and the transistor enters the ON state. In this state, voltage coming from the in line, called the collector, goes directly to the out line, called the emitter, and this voltage can be detected by a measuring device.
- This ON state can be used to represent the binary 1.
- If, instead, we set the input value of the control line to a 0 by not applying voltage, the switch opens and the transistor enters the OFF state. In this state, no voltage can get through the transistor, and none would be detected on the output line.
- The OFF state can be used to represent the binary 0.
Binary computers = transistors?

- This solid-state switching device, the *transistor*, forms the basis of construction of virtually all computers built today, and it is thus the fundamental building block for all high-level computers.

- However, there is no theoretical reason why we must use *transistors* as our *elementary particles* when designing computer systems.
Binary storage devices

Binary computers can be built out of any bistable device. Formally, it is possible to construct a binary computer using any device that meets the following four conditions:

- It has two stable energy states (one for a 0, one for a 1).
- These two states are separated by a large energy barrier (so a 0 does not accidentally become a 1, or the reverse).
- It is possible to sense what state the device is in (to see if it is storing a 0 or a 1) without permanently destroying the stored value.
- It is possible to switch from a 0 to a 1 and vice versa by applying a sufficient amount of energy.
Binary computers ≠ electronic computers!

- There are many devices that meet the four conditions, including some unexpected ones such as the familiar on-off light switch!
- Thus it would be possible to build a reliable (albeit very slow and bulky) binary computer out of ordinary light switches!
- Just as magnetic cores were replaced by transistors, it is possible that transistors will ultimately be replaced by some newer technology: optical, biological, quantum
- These new computation implementations may be faster, smaller, cheaper, and/or more versatile.
Basic logic gates: The inverter

- The inverter accepts a Boolean value (0 or 1) as input and produces the complement of this value as its output.

(a) Inverter
The opposite of OR

- To construct the negation of OR, we use two transistors connected in parallel.
The opposite of OR

- If either or both of the lines Input-1 and Input-2 are set to 1, then the corresponding transistor is in the ON state, and the output is connected to the ground, producing an output value of 0.
- Only if both input lines are 0, effectively shutting off both transistors, will the output line contain a 1.
- This is the exact opposite of the definition of OR, and this gate is called NOR gate.
The OR gate

- The OR gate can be implemented using a NOR gate and an inverter.
- The input to this gate are the values of two or more Boolean variables. The output is the Boolean sum ($\lor$) of their values.
The opposite of AND

To construct the negation of AND, we use two transistors connected in series.
The opposite of AND

- The collector line of transistor 1 is connected to the power supply (logical 1).
- The emitter line of transistor 2 is connected to ground (logical 0).
- If both input lines, Input-1 and Input-2, are set to 1, then both transistors are in the ON state, and the output will be connected to the ground, resulting in a value of 0 on the output line.
- If either (or both) Input-1 or Input-2 is 0, the corresponding transistor is in the OFF state and does not allow the current to pass, resulting in a 1 on the output line.
- Thus the output of this gate is the opposite of AND and represented a gate called NAND gate.
The **AND gate**

- The **AND gate** can be implemented using a NAND gate and an inverter.
- The inputs to this gate are the values of two or more Boolean variables. The output is the Boolean product ($\land$) of their values.
Gates with multiple inputs

We sometime permit multiple inputs to AND and OR gates, as below.
Toffoli Gate

It has a 3-bit input and 3-bit output: If the first two bits are both 1, it inverts the 3rd bit, otherwise all bits stay the same.
Toffoli Gates and Quantum Computing

- The Toffoli gate is a universal, reversible logic gate
- **Universal** = any Boolean function can be realized by Toffoli gates
- **Reversible** = given the output one can uniquely reconstruct the input (¬ is reversible, but ∧ is not)
- The Toffoli gate can be realized by five two-qubit quantum gates
- This implies that a quantum computer can implement all possible classical computations
- A quantum-mechanics-based Toffoli gate has been successfully realized in January 2009 at the University of Innsbruck, Austria
Combinatorial circuits can be constructed using a combination of inverters, OR gates and AND gates.

When combinations of circuits are formed, some gates may share inputs. This is shown in one of two ways in depictions of circuits.

One method is to use branchings to indicate all the gates that use a given input.

The other method is to indicate this input separately for each gate.

Note that output from a gate may be used as input by one or more elements.
How to draw circuits

Both drawings below depict the circuit with output $xy + \overline{xy}$.
Example

- **Example:** Construct the circuits that produce the following outputs:

  \[(a) \ (x + y)\overline{x}\]

  \[(b) \ \overline{x}(y + \overline{z})\]

  \[(c) \ (x + y + z)(\overline{x} \ \overline{y} \ \overline{z})\].
Example answers

(a) \((x + y)\overline{x}\)
(b) \( \overline{x(y + \overline{z})} \)
(c) \((x + y + z)\overline{x} \overline{y} \overline{z}\).
Example 1. A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.

Solution: Let $x = 1$ be 1 if the first individual says yes and $x = 0$ if this individual says no. Similarly for $y$ and $z$.

Then a circuit must be designed that produces output 1 from the inputs $x, y, z$ when two or more of $x, y, z$ are 1.

One representation of the Boolean function that has these output values is $xy + xz + yz$. 
Circuit for majority voting

\[ xy + xz + yz \]
Design a circuit for a 3-switch light fixture

Example 2. Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off and turns the light off when it is on. Design circuits that accomplish this when there are three switches.

Solution:
We have three variables, \(x, y, z\), one for each switch. Let \(x = 1\) when the first switch is closed and \(0\) when it is open, and similarly for \(y\) and \(z\).

Let \(F(x, y, z) = 1\) when the light is on and \(0\) when it is off.

We can arbitrarily specify that the light be on when all three switches are closed, so that \(F(1, 1, 1) = 1\).

This determines all the other values of \(F\).
The function $F$ can be represented as

$$xyz + x\overline{y} \overline{z} + \overline{x} \overline{y} \overline{z} + \overline{x} \overline{y} \overline{z}.$$
Example 3: Adders

- Logic circuits can be used to carry out addition of two positive integers from their binary expansions.
- We will build up the circuitry to do this addition from some component circuits.
- First we build a circuit that can be used to find $x + y$ when $x$ and $y$ are two bits.
- The input to our circuit will be $x$ and $y$, since each of these have value 0 or 1.
- The output will consist of two bits, namely $s$ and $c$ where $s$ is the sum bit and $c$ is the carry bit.
The half-adder

- This circuit is called a multiple output circuit.
- The circuit we are designing is called the half-adder since it adds two bits, without considering a carry from the previous addition.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$s$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- From the truth table we see that $c = xy$ and $s = x\overline{y} + \overline{x}y$
- If we use the fact that $x\overline{y} + \overline{x}y = (x + y)(\overline{xy})$ we obtain a circuit with fewer gates
Circuit for half-adder

\[ \text{Sum} = (x + y)(\overline{xy}) \]

\[ \text{Carry} = xy \]
The full adder (3 inputs, 2 outputs)

Input: Bits \(x\) and \(y\) and the carry bit \(c_i\).

Output: The sum bit \(s\) and the carry bit \(c_{i+1}\).

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>(y)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Formulas for outputs of the full adder

- From the truth table we obtain

\[ s = xyc_i + x\overline{y} \overline{c_i} + \overline{x} \overline{y}c_i + x y c_i. \]

\[ c_{i+1} = xyc_i + xy\overline{c_i} + x\overline{y}c_i + \overline{x} \overline{y}c_i. \]
Circuit for full adder using half adders

However, instead of building the full adder from the scratch, we will use half adders to produce the desired outputs.

\[ s = xyC_i + x\overline{y}\overline{C_i} + \overline{xy}\overline{C_i} + \overline{x}\overline{y}C_i \]

\[ c_{i+1} = xyc_i + x\overline{y}\overline{c_i} + \overline{x}\overline{y}c_i + \overline{xy}\overline{c_i} \]

Exercise: Verify that the output of the above circuit (based on definition of the outputs of the half-adder circuit and of the OR gate) is indeed \( s \) and \( c_{i+1} \) as determined by the truth table for a full-adder.
Adding three-bit integers

Full and half adders can be used to add the three-bit integers \((x_2x_1x_0)_2\) and \((y_2y_1y_0)_2\) to produce the sum \((s_3s_2s_1s_0)_2\).

Note that \(s_3\), the highest-order bit in the sum, is given by the carry \(c_2\).
Circuit minimization through logic formula simplification

- Consider the circuit that has output 1 iff \( x = y = z = 1 \) or \( x = z = 1 \) and \( y = 0 \).
- The formula corresponding to this truth table is \( xyz + x\bar{y} z \).
- \( xyz + x\bar{y} z = (y + \bar{y})(xz) = 1 \cdot (xz) = xz \).
- Hence, \( xz \) is a Boolean expression with fewer operators that represents the circuit, and the corresponding simplified circuit will have fewer logic gates.
- Thus, one can use the essential laws for propositional logic (De Morgan, law of excluded middle, law of contradiction, etc) to minimize circuits.
A circuit and its minimized version

Adequate set of connectives, logic gates and circuits
Circuit minimization

Build a circuit for the formula with the following truth table. Use the laws of propositional calculus (De Morgan, commutativity,..) to simplify the formula and minimize its circuit.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Analyzing and simplifying code through logic formula simplification

\[
\text{if } (C_1 \text{ or not } C_2) \text{ then } \\
\quad \text{if } (\text{not } (C_2 \text{ and } C_3)) \text{ then } P_1 \\
\quad \text{else } \\
\quad \quad \text{if } (C_2 \text{ and not } C_3) \text{ then } P_2 \\
\quad \quad \text{else } P_3 \\
\quad \text{else } P_4
\]
Analyzing and simplifying code

Analyzing code

**Hint:** Write the truth table for the conditions for each of the if-statements. Use this truth table to determine, for each value assignment for $C_1$, $C_2$, $C_3$, which code ($P_1$, $P_2$, $P_3$ or $P_4$) is executed.

Prove that $P_2$ is a dead code (without truth tables)

**Hint:** Determine what is the logical formula involving propositional variables $C_1$, $C_2$, $C_3$ that would lead to the execution of $P_2$. If that formula is a contradiction (always false), then $P_2$ is a dead code.

Is $P_3$ a dead code?

**Hint:** Determine what is the logical formula involving propositional variables $C_1$, $C_2$, $C_3$ that would lead to the execution of $P_3$. Find out whether that formula is a tautology, a contradiction, or a contingent formula. If it is not a contradiction, then $P_3$ is not a dead code.
Simplified Code

- Simplified code
  if \((C_1 \text{ and } C_2 \text{ and } C_3)\) then
    \(P_3\)
  else
    if \((\text{not } C_1 \text{ and } C_2)\) then
      \(P_4\)
    else
      \(P_1\)

- One can use the laws of propositional calculus to verify that the simplified code is equivalent to the original code.
  This is proved by showing that the formula with variables \(C_1, C_2, C_3\) that leads to the execution of \(P_1\) in the original code is logically equivalent to the formula that leads to the execution of \(P_1\) in the simplified code, and the same holds for \(P_2, P_3\) and \(P_4\).

  (example from A.Lubiw, W15 notes)
Acknowledgements

The figures in this set of notes are from