1. Determine whether the following statements are true or false. If true, state the name of the respective Boolean algebra axiom or elementary theorem. If false, state a counterexample.

   (a) $a \wedge b = b \wedge a$

   (b) $a \vee 0 = 0$

   (c) $a \wedge (b \vee c) = (a \vee b) \wedge (a \vee c)$

   (d) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

   (e) $(a \vee b)' = a' \wedge b'$

(2+2+2+2+2)

(a) True: commutative law

(b) False: $1 \vee 0 = 1$ (not 0)

(c) False: $1 \wedge (0 \vee 0) = 1 \wedge 0 = 0$ but $(1 \vee 0) \wedge (1 \vee 0) = 1 \wedge 1 = 1$

(d) True: distributive law

(e) True: De Morgan's law

2. (a) Interpret the hexadecimal number 0xCB as an 8-bit two's complement signed integer and convert to decimal.

(b) Perform the following binary addition:

$$0110\,1111 + 0110\,1010$$

(c) Does the result of the computation in (b) cause an overflow if the two numbers are interpreted as (i) 8-bit unsigned integers and (ii) as 8-bit two's complement signed integers?

(4+3+3)

(a) $0x\,CB = (1100\ 1011)_2$

Take two's complement: $\begin{array}{r} 0011\ 0100 \\ +\qquad\quad 1 \\ \hline 0011\ 0101 \end{array}$

Finally: $(00110101)_2 = 2^0 + 2^2 + 2^4 + 2^5$

$= 1 + 4 + 16 + 32$

$= 53$

Thus, $0x\,CB$ represents $-53$ when interpreted as a two's complement signed integer.

(b) $\begin{array}{r} 0110\ 1111 \\ +\ 0110\ 1010 \\ \hline 1101\ 1001 \end{array}$

(c) As 8-bit unsigned integers, the result of this computation is valid.

As 8-bit two's complement signed integers, the correct result is outside of the representable range $-128, \dots, 127$, so an overflow has occurred.

3. (a) Perform the following computation in decimal floating point with five significant digits:

$$(1.0000 \cdot 10^0 + 3.3333 \cdot 10^{-3}) - 1.0000 \cdot 10^0$$

Suggest a different ordering of arithmetic operations that improves the accuracy of the computation.

(b) Consider the evaluation of the following expression in floating point arithmetic:

$$\frac{1-(1-x)^3}{x}$$

Identify values of $x$ for which there is a substantial growth of relative error and suggest an alternate formula that improves accuracy for the problematic range of $x$.

(5+5)

(a) Computation in steps:

$$\begin{array}{r} 1.0000 \\ +\quad 3.3333 \\ \hline 1.0033 \end{array}$$

(rest is rounded down)

$$\begin{array}{r} 1.0033 \\ -\ 1.0000 \\ \hline 0.0033 \end{array} = 3.3000 \cdot 10^{-3}$$

Computing the result as

$$(1.0000 \cdot 10^0 - 1.0000 \cdot 10^0) + 3.3333 \cdot 10^{-3}$$

would not lose any significant digits.

(b) When $x \approx 0$, there is a diffence between almost equal numbers, causing potential loss of significant digits. Better write

$$\frac{1-(1-x)^3}{x} = \frac{1-(1-3x+3x^2-x^3)}{x} = \frac{3x-3x^2+x^3}{x} = 3-3x+x^2$$

(Or use Horner's scheme and write $\ldots = 3 - x(-3+x)$, not required.)

4. Consider the following FSA:

$$\begin{tikzcd}
\arrow[r] & A \arrow[loop above, "0,1"] \arrow[r, "1"] & B \arrow[r, "0"] & C \arrow[r, "1"] & D \arrow[loop above, "0,1"]
\end{tikzcd}$$

(a) Which strings are recognized by this FSA? Give a plain language answer and also state the corresponding regular expression.

(b) Convert this nondeterministic FSA into a deterministic FSA.

(5+5)

(a) All binary strings that contain the substring "101".

Regular expression: [01]* 101 [01]*

(b)

$$\begin{tikzcd}
\arrow[r] & \text{Start} \arrow[loop above, "0"] \arrow[r, "1"] & \text{"1"} \arrow[loop above, "1"] \arrow[r, "0"] & \text{"10"} \arrow[r, "1"] \arrow[ll, bend left, "0"] & \text{"101"} \arrow[loop above, "0,1"]
\end{tikzcd}$$

Note: We can also label the states via reference to the states of the original FSA:

$$\begin{tikzcd}
\arrow[r] & A \arrow[loop above, "0"] \arrow[r, "1"] & A \text{ or } B \arrow[loop above, "1"] \arrow[r, "0"] & A \text{ or } C \arrow[r, "1"] \arrow[ll, bend left, "0"] & A \text{ or } B \text{ or } D \arrow[loop above, "0,1"]
\end{tikzcd}$$

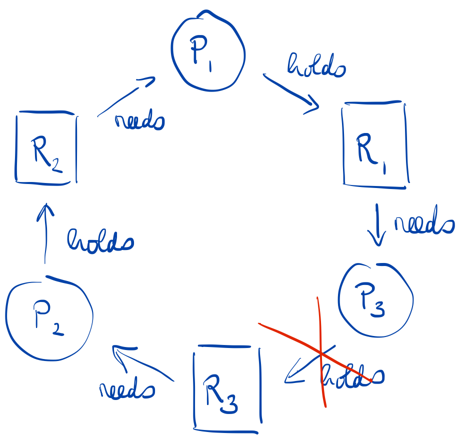
5. (a) Process $P_1$ holds resource $R_1$ and needs resource $R_2$. Process $P_2$ holds resource $R_2$ and needs resource $R_3$. Process $P_3$ holds resource $R_3$ and needs resource $R_1$. Draw a resource allocation graph and argue that this is a deadlock situation.

(b) Now impose the following ordering constraint: *A process may hold resource $R_i$ only if it already holds all $R_j$ with $j < i$ that it needs.*

Illustrate that this rule breaks the deadlock situation in (a). Does this rule prevent deadlock in general? Does it prevent starvation in general?

(5+5)

(a)



dependency cycle → deadlock

(b) According to the rule, $P_3$ must not hold $R_3$ as it does not yet hold $R_1$.

→ The cycle is broken (red above), so $P_2$ can obtain $R_3$ and finish, then $P_1$, finally $P_3$.

In general, the rule prevents dependency cycles as any cycle would contain a "hold" - reference to the resource of highest index which is forbidden by the rule.

The rule does not guarantee any ordering on the use of free resources, so starvation is possible.

6. On a filesystem that allows soft links and hard links, you create `file_1`, hard-link `file_2` to `file_1`, and soft-link `file_3` to `file_1`. Then you delete `file_1`. What happens if you access `file_2`? What happens if you access `file_3`? Explain. (5)

An inode is only deleted if its reference count is zero.

Since file_2 still holds a reference to the inode, the data is still there and accessible via file_2.

However, the soft link file_3 resolves via the file name file_1, which is deleted, so the soft link becomes stale even though the data is still there.

7. (a) In a RAID-5 array, one disk out of 4 has failed. The others contain the bit sequences 0000 1111 . . ., 0101 0101 . . ., 0100 0010 . . .. Reconstruct the beginning of the bit sequence of the failed disk.

(b) An IBAN is checked by verifying that, after suitable rearrangement, replacement of letters by number digits, and conversion to integer, the given number modulo 97 equals 1. What is the rationale behind the convention that 1 rather than 0 indicates correctness?

(5+5)

(a) We need to XOR the three bitstreams, which can be seen as addition without carry:

```
  0000  1111
  0101  0101
  0100  0010
 ------------
  0001  1000
```

(b) Often, missing data is represented by a string of 0s.

The convention ensures that a string of 0s is not a valid IBAN, thus protecting from interpreting missing data as valid.

8. The (3, 1)-Hamming code is the shortest code in the Hamming family of codes.

   (a) Explain the construction of the 3-bit codeword from the single data bit. Then explain how a single-bit error is corrected.

   (b) Draw a truth table which describes the mapping from arbitrary 3-bit ~~data words~~ messages (which may contain single-bit errors) to the *corrected* single data bit.

   (c) Convert the truth table from part (b) into a Boolean algebra expression, simplify if possible, then draw the resulting Boolean function as an array of logic gates.

   (d) *(Extra credit.)* Draw an FSA that reads three data bits of a (3, 1)-Hamming-encoded message and has two final states, 1 and 0, representing the *corrected* single data bit.

(5+5+5+5)

(a) Following the standard bit ordering from class, the bit pattern is

|  | $p_1$ | $p_2$ | $d$ |
|---|---|---|---|
| $p_1$ | — |  | — |
| $p_2$ |  | — | — |

where $p_1$ and $p_2$ check only themselves and the data bit $d$.

The check procedure simplifies as follows:

- If both parity bits agree, the corrected data bit is identical to parity.
- If the parity bits disagree, one must be wrong, so take $d$ as received.

Alternatively: The only correct codewords are 000 and 111, so the role of the parity bits and the data bits is symmetric. The corrected output is determined by the majority of inputs.

(b)

| $p_1$ | $p_2$ | $d$ | corrected |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

(Solution ctd./scratch paper)

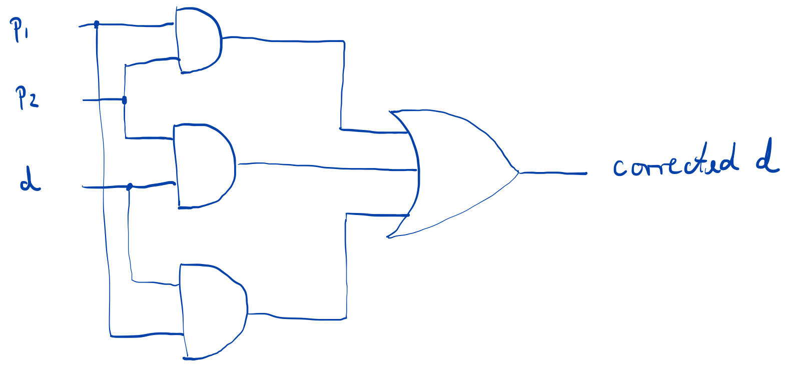
(c) $p_1' \wedge (p_2 \wedge d) \vee (p_1 \wedge (p_2 \vee d))$

$= p_1' \wedge p_2 \wedge d \vee p_1 \wedge p_2 \vee p_1 \wedge d$

$= p_1' \wedge p_2 \wedge d \vee (p_1 \wedge p_2 \wedge d \vee p_1 \wedge p_2 \wedge d') \vee (p_1 \wedge p_2 \wedge d \vee p_1 \wedge p_2' \wedge d) \vee p_1 \wedge p_2 \wedge d$

$= (p_1' \wedge p_2 \wedge d \vee p_1 \wedge p_2 \wedge d) \vee (p_1 \wedge p_2 \wedge d' \vee p_1 \wedge p_2 \wedge d) \vee (p_1 \wedge p_2' \wedge d \vee p_1 \wedge p_2 \wedge d)$

$= p_2 \wedge d \vee p_1 \wedge p_2 \vee p_1 \wedge d$ $(*)$

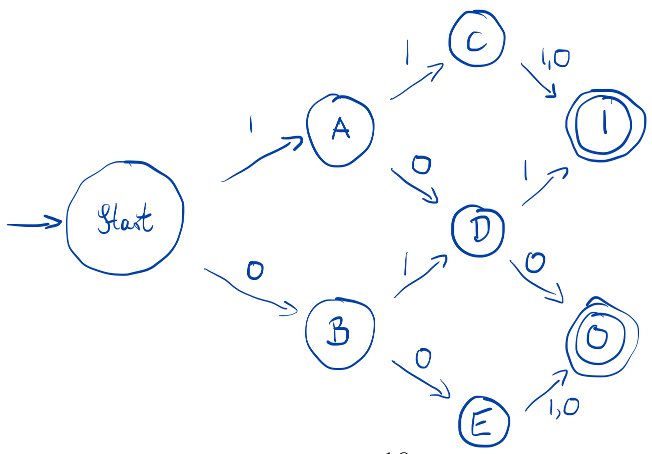


Note: There are many equivalent expressions, this is arguably the shortest.

Grading was focussing on correctness, not on being strictly the shortest.

Also, it is possible to spot $(*)$ directly from part (a) or (b).

(d)



Start
A
B
C
D
E
1
0
1,0