1. Simplify the following Boolean algebra expressions as much as possible.

   (a) $(a \vee 0) \wedge (a' \vee 1)$

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

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

(5+5+5)

(a) $(a \vee 0) \wedge (a' \vee 1) = a \wedge 1 = a$

(b) $a \vee (a' \wedge b) = (a \vee a') \wedge (a \vee b) = 1 \wedge (a \vee b) = a \vee b$

(distr.)

(c) Let $z = (a \vee b) \wedge (a' \vee c)$

$\Rightarrow z' = (a' \wedge b') \vee (a \wedge c')$ (De Morgan)

So the given expression reads $z \wedge z' = 0$.

2. Suppose you evaluate the expression

$$f(x) = \frac{1}{x-1} - \frac{1}{x+1}$$

in floating point arithmetic for large values of x.

(a) Is the absolute error large? Is the relative error large? Explain!

(b) Can you reduce the floating point error when x is large by rewriting the expression? Explain!

(5+5)

(a) When x is large, both terms in $f(x)$ are small, so the absolute error is also small.

However, when x is large, the expression is the difference of almost equal numbers, leading to a loss of significant digits, i.e. a large relative error.

In an extreme case, x is so large (depending on the length of the significant) that

$$x \oplus 1 = x \ominus 1 = x,$$

so that

$$\frac{1}{x \ominus 1} \ominus \frac{1}{x \oplus 1} = 0$$

Then the absolute error is $|f(x)|$ and the relative error is 1, meaning that all digits of the resulting significant are wrong.

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

This avoids taking a difference of almost equal numbers.

Note: Multiplying out the denominator to get $f(x) = \frac{2}{x^2-1}$ is <u>not</u> a good idea, because $x \gg 1$ implies $x^2 \gg x$, which roughly doubles the number of significant digits that are lost when computing x^2-1 vs. $x \pm 1$.

3. Consider the following tasks.

   (i) Take two arbitrary length integers as binary strings written right-to-left and output their sum, also as a binary string right-to-left.

   (ii) Take two arbitrary length integers as binary strings written left-to-right and output their sum, also as a binary string written left-to-right.

   (iii) Count the number of zeros in a binary string or arbitrary length.

   (iv) Read a binary string whose length is divisible by four and output the parity of every group of four bits.

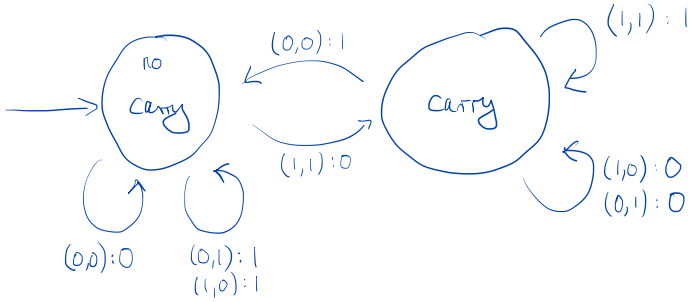
   Answer the following questions:

   (a) Which of the tasks above can be performed by a finite state transducer? Give a concise justification in each case.

   (b) Pick one of the tasks that can be performed by a finite state transducer and draw the transducer that solves the task.

(8+7)

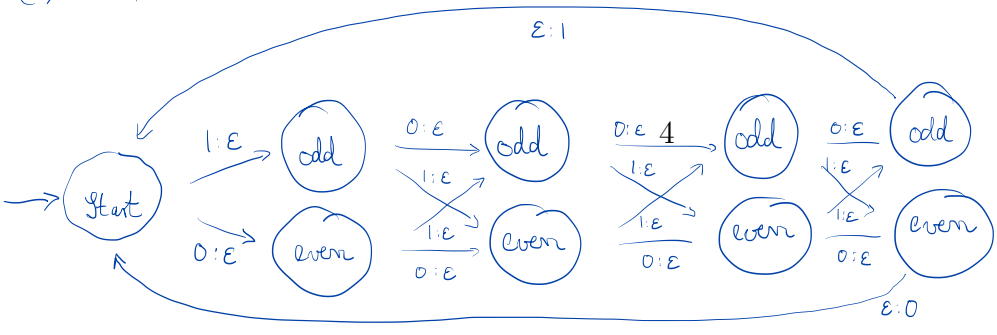
(i) Addition is performed right-to-left (starting with the least significant binary digit). At each state, the operation can be represented by a simple truth table that takes both incoming digits and carry-in, produces the output digit and one bit of carry-out, which can be stored in an FST:



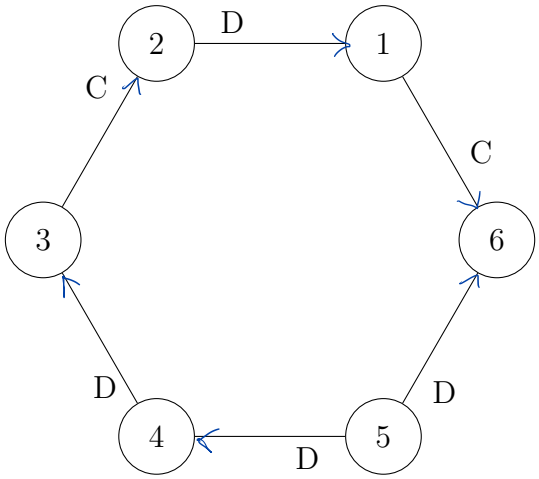
(ii) As the most significant digit may be affected by the least significant digit, the latter needs to be computed first. Thus, this order of data requires unbounded memory, not possible.

(iii) A counter on unbounded input requires unbounded memory, not possible

(iv) Requires storage of a count of at most 4, thus easily implemented:



4. Consider the dining philosopher's problem with six philosophers sitting around a table. They follow the Chandy–Misra protocol. A chopstick is either *dirty* ("D") or *clean* ("C"), and held by the philosopher closest to it in the following initial configuration:



Assume, for simplicity, that the philosophers "are running on the same CPU", i.e., no two actions (starting to eat, requesting a chopstick, honoring the request for a chopstick) can happen at the same time. You have no control on the ordering of events, except for the constraints imposed by the Chandy–Misra protocol.

(a) All philosophers are currently hungry. Which philosopher is the first to eat? If there is ambiguity, state all possibilities.

(b) Argue that Philosopher 3 cannot starve, i.e., eventually gets to eat.

(c) Give a high-level summary of the proof that the Chandy–Misra protocol always works, i.e., that the philosophers cannot deadlock, and that every philosopher who wants to eat will eventually get to eat.

(5+5+5)

(a) 3 and 1 cannot eat first, as one of their neighbors holds a clean chopstick, which they will not yield until they have eaten. All the others may be first to eat, as the order of necessary actions is not constrained.

(b) The arrows indicated above indicate priority.

- 3 has higher priority than 4, so can always get their common chopstick (before or after 4 has eaten)
- to its left, 3 depends on 2 to have eaten. 2 can either eat immediately (and loses priority over 3 afterwards), or has to wait for Philosopher 1 to its left. This pattern repeats, but only a finite number of times as 6 has priority over both its neighbors.
- Thus 3 has to wait, in the worst case, for 5, 6, 1, 2 to finish (in this order).

(c) We say that a philosopher who

- holds a clean chopstick
- or does NOT hold a dirty chopstick

has priority over that resource. Then

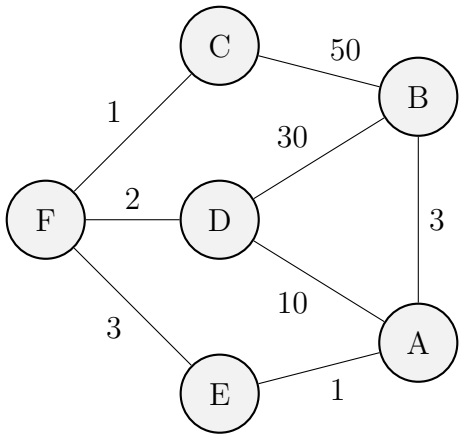
- the graph of priority relations is initially acyclic
- it remains acyclic under all changes of priorities
- A philosopher with highest priority (who must always exist in an acyclic graph) can always eat
- When a philosopher stops eating its priority is lowered and they cannot eat again until all philosophers initially lower in the priority chain have eaten

⇒ every philosopher will eventually eat.

5. How many data bits can you encode in a Hamming code of total length 20 if additional parity is used? (5)

The next-shortest maximal code is Hamming-(16, 11), where 5 parity bits guard 11 data bits. Thus, one additional parity will be enough for 4 more bits in total. This leaves 20 − 6 = 14 bits for data.

6. Consider the following router network:



Use Dijkstra's algorithm to compute the shortest path from router A to every other router in the network. (10)

Step	Todo	d(A)	d(B)	d(C)	d(D)	d(E)	d(F)
0	~~A~~BCDEF	(0)	∞	∞	∞	∞	∞
1	BCDEF	0	3 viaB	∞	10 viaD	(1) viaE	∞
2	~~B~~CDF	0	(3)	∞	10	1	4 viaE
3	CD~~F~~	0	3	53 viaB	10	1	(4)
5	~~C~~D	0	3	(5) viaE	6 viaE	1	4
6	~~D~~	0	3	5	(6)	1	4
7		0	3	5	6	1	4

Routing table (not required)

From A	A	B	C	D	E	F
Next Hop	A	B	E	E	E	E
Cost	0	3	5	6	1	4

7. (a) Design a database schema for a (simplified) employee database. An *employee* has a name and is a member of exactly one *team*. A team has a *team leader* who is an employee and may have additional members that are also employees.

   (b) Write a query, using relational algebra or SQL, to find the name of the leader of the team where employee "Susie Miller" is a member.

(5+5)

(a) Employee (E_No, E_Name, T_No)

Team (T_No , Lead_E_No)

(b) $t\_susie = \sigma_{E\_Name = 'Susie\ Miller'} (Employee \bowtie Team)$

gives a relation containing all teams (should be one) where Susie is a member. Then

$$\pi_{E\_Name}\ \sigma_{t\_susie.Lead\_E\_No = Employee.E\_No} (t\_susie \times Employee)$$

gives the name of the team lead.

Alternative, using SQL:

```
SELECT E2.E_Name FROM Employee AS E1
    JOIN Team ON Team.T_No = E1.T_No
    JOIN Employee AS E2 ON Team.Lead_E_No = E2.E_No
    WHERE E1.E_Name = 'Susie Miller'
```