Self-Assessment
for
Master Degree Course Computer Science

Department of Computer Science
Paderborn University
Germany

November 27, 2018

The following are exercises for a self-assessment in Computer Science (CS), covering **some** of the subjects which a student with a Bachelor degree in Computer Science should know. This self-assessment is intended to be taken by prospective students of Computer Science at Paderborn University aiming for a Master degree. The exercises are taken out of written exams in CS Bachelor courses at Paderborn University. Thanks go to a number of lecturers for preparing the exercises. The exercises are for personal usage only; copyright is with Paderborn University and the lecturers having prepared the exams.

The first part contains the exercises; the second part the solutions. For a self-assessment, it is **highly recommended** to do the exercises *without* looking at the solutions. Solutions are solely ment for checking the correctness of answers. As the exercises are prepared by different persons, they differ in style.

The maths exercises cover linear algebra only. For more exercises in mathematics see e.g. Frank Ayres JR.: Schaum's Outline Series; Theory and Problems of Calculus (2nd edition); McGraw-Hill Book Company, e.g. Chapter 2 on "Limits", Chapter 3 on "Continuity" or Chapter 13 on "Differentiation of inverse trigonometric functions". Additionally, a certain amount of knowledge in probability theory is necessary.

If a prospective student has **difficulties in the majority of exercises** (or even, has never heard of the concepts in the exercises), this is a clear indication that the student does not possess enough knowledge in Computer Science to successfully complete the Master degree course in CS at Paderborn University.

Part I

Exercises

1 Mathematics

Exercise 1. (easy)

Consider the four vectors

$$v_1 = \begin{pmatrix} -1 \\ 3 \\ 1 \end{pmatrix}, \qquad v_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \qquad v_3 = \begin{pmatrix} -2 \\ 3 \\ -1 \end{pmatrix} \qquad \text{and} \qquad v_4 = \begin{pmatrix} 1 \\ t^2 - 2 \\ t \end{pmatrix}$$

in $\mathbb{R}^3$ where $t \in \mathbb{R}$.

(a) Do the vectors v_1, v_2, v_3 form a basis of $\mathbb{R}^3$?

(b) Determine all $t \in \mathbb{R}$ such that v_1, v_2, v_4 is a generating set of $\mathbb{R}^3$.

Exercise 2. (more difficult)

Let $a, b, c \in \mathbb{R}$. Show that the real (2×2)-matrix

$$A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

is diagonalizable.

2 Programming, Programming Languages, Software Engineering, Databases

2.1 Programming

The questions below use Java, Python, or a combination of the two. If you are familiar with other modern programming languages, that is fine as well. We do not insist upon any particular programming language as a prerequisite.

Exercise 3. Methods.

Write a static Java method `countEven` which takes an integer array as argument and returns the number of *even* values in it. Write a `main` method in which `countEven` is called.

Exercise 4. Recursion.

Let the function f be defined for $a \leq b$ by

$$f(a,b) = \begin{cases} f(a, \lfloor \frac{a+b}{2} \rfloor) + f(\lfloor \frac{a+b}{2} \rfloor + 1, b) & \text{if } a < b \\ b & \text{else.} \end{cases}$$

What does f calculate? Is f using *direct* recursion or is it using *indirect* recursion? Give the source code of either a Java or a Python implementation of f which uses recursion.

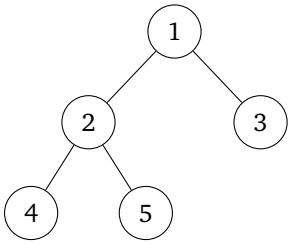
Exercise 5. Trees.

Let the following Java class `BinTree` for binary trees be given.

```
public class BinTree {
    BinTree left;
    BinTree right;
    int data;

    public BinTree (BinTree l, BinTree r, int d) {
```

```
            left = l;
            right = r;
            data = d;
        }

        public BinTree (int d) {
            left = null;
            right = null;
            data = d;
        }
    }
```

(a) Write a Java program which generates the following tree.



(b) Write a method `int numberOfLeaves()` which recursively counts the number of leafs in the tree. A leaf is a node without successors.

Example: The above tree has 3 leafs.

Exercise 6. Closures.

Use Python to implement a function `f` that takes a single number `m` as its parameter. It shall return a function that does the following: Take a list of numbers as its parameter and return a list which has all numbers smaller than `m` removed. Also give a sample invocation of your function.

Exercise 7. Inheritance.

What output does the following Java code produce if `new Line()` is called?

```
public abstract class Chain {
    public int a = 42;
    public Chain() {
        System.out.println("Forty-Two");
    }
}

public class Line extends Chain {
    public Line() {
        System.out.println(a);
    }
}
```

2.2 Programming Languages

Exercise 8. Syntax.

The following productions of a context-free grammar with starting symbol `spec` are given:

```
(p1)  spec       ::= interfaces
(p2)  interfaces ::= interfaces interface
(p3)  interfaces ::=
(p4)  interface  ::= 'ifac' ident ';'
(p5)  interface  ::= 'ifac' ident 'extends' super ';'
(p6)  super      ::= idents
(p7)  idents     ::= ident idents
(p8)  idents     ::= ident
```

(a) Determine the sets of terminals and nonterminals.

(b) Draw the derivation tree for the following sentence:

```
ifac ident ; ifac ident extends ident ident ;
```

(c) We add a production p9 to the above grammar:

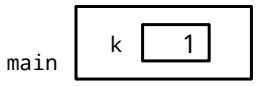
```
(p9)  idents    ::=
```

This modification additionally allows empty lists of `ident` terminals. Unfortunately it turns the grammar into an ambiguous grammar. Show the ambiguity of this extended grammar. Give a short explanation of your proof.

Exercise 9. Runtime stack.

(a) Complete the drawing of the runtime stack (1 snapshot) for the execution of the following program until the point of time where "Halt" is being output. Include the parameter and variable values in the stack frame (like shown in main's frame) and draw the references to the static predecessor frames (*static links*). Mark frames that have been removed by crossing them out.

```
1  proc main() {
2     k = 1;
3     proc a(n) {
4        proc b() {
5          if (n == k) {
6             print "Halt";
7          }
8        }
9
10       if (n == 1) {
11          a(n + 1);
12       }
13       b();
14    }
15    a(k);
16 }
```

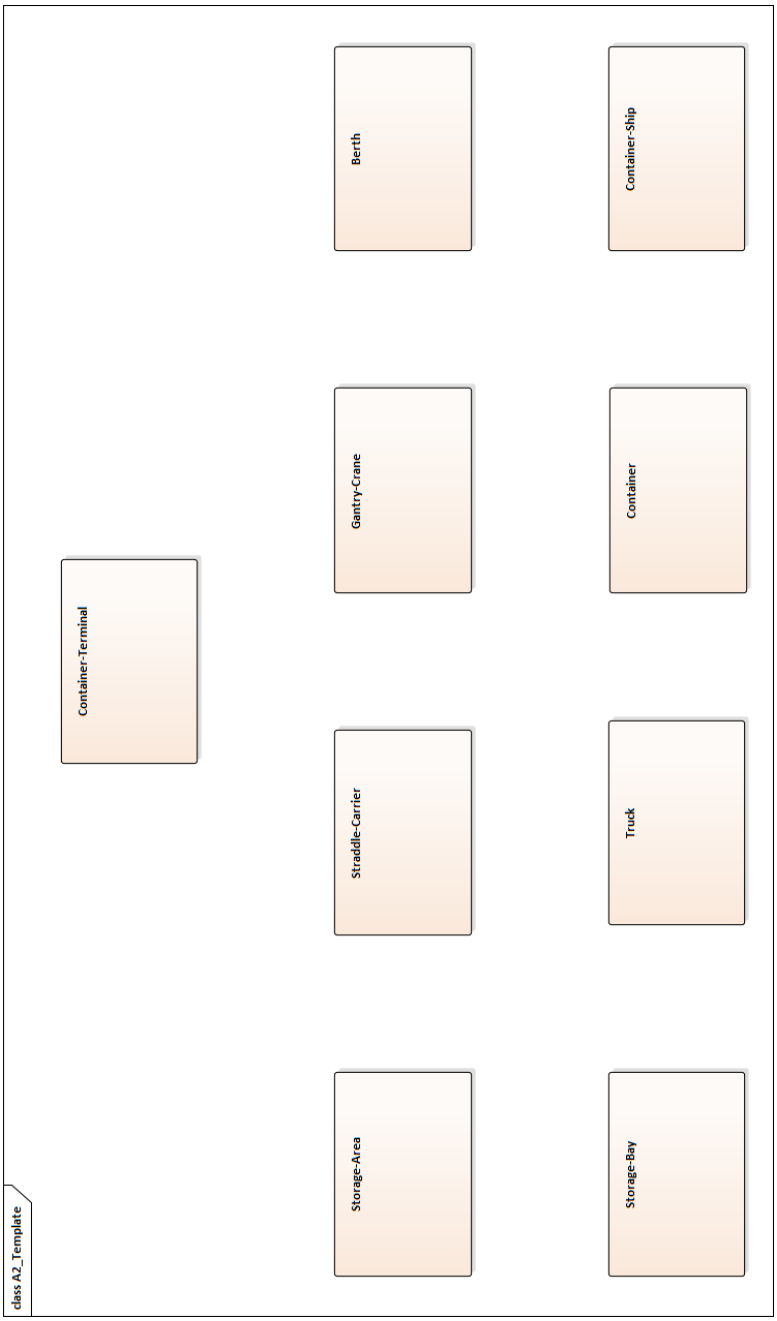


(b) The stack frames contain references to the static predecessors. What are those references used for in general and particularly in the above example?

2.3 Software Engineering

Exercise 10. Model of the problem domain.

Specify a model of the problem domain that models the description of a container terminal contained in the reference as accurately and completely as possible. To do this, use the following predefined class frame.



Reference:

A container terminal consists of at least one storage area for containers, at least one straddle carrier, at least one gantry crane and at least one berth for container ships.

A storage area consists of 50 individual storage bays. On one storage field, up to four containers can be stacked on top of each other. A gantry crane stands at a berth and can transport up to two containers. A straddle carrier can only transport one container and is located either at a berth or a storage area. There is at least one gantry crane at a berth and there can be at most one berth.

Each container ship has a capacity that specifies how many containers can be transported. The maximum capacity is 3000 containers.

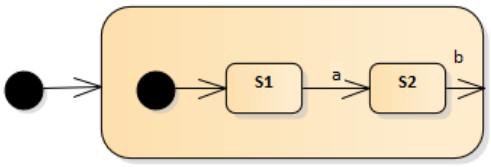
At the storage area, trucks can stand to be loaded. A truck can carry up to two containers.

A container can be empty or full and has a container number.

Furthermore, a container is located either on a container ship, a truck, a straddle carrier, a gantry crane or on a storage bay. Containers can be stacked.

Exercise 11. Statecharts.

(a) Convert the following statechart into a flat state machine without superstates.



(b) The information in the reference describes important functions of a microwave oven. Use it to construct a UML statechart of the system that contains all the properties described as complete and precise as possible. Use superstates to keep the number of transitions low.

Reference:

> We’re looking at a microwave oven. It has a mechanical main powerswitch, with which the microwave oven can be completely switched on and off.
>
> When the microwave is switched on, it goes to standby mode and the power can be adjusted with a push button. When switched on for the first time, it is set to 600W. Press the push button to switch to 200W, 400W and then back again to 600W. The selected power is stored when the microwave oven is turned on and off.
>
> To start the warm-up process, the user can press one of three buttons: If the start button is used, the unit will run until the stop button is pressed. When ssing the "1 minute" button, the oven will run until the time has elapsed or until the user presses the stop button. The "5 minutes" button behaves analogously. Even during the warm-up process, the output power can be adjusted.
>
> Of course, the microwave oven also has a door that can be opened and closed, even when switched off. However, the warm-up process cannot be started when the door is open, and while the warm-up process is running the door cannot be opened.
>
> On delivery, the microwave oven is switched off and the door is closed.

Exercise 12. Design patterns.

In the reference you find Java code that creates a window, draws a black square and creates a quit button. In addition, you find a simplified class diagram of the used Java library Swing (**Attention:** The Java code contains further classes). You will also find the Composite Pattern and the Observer Pattern in the reference as assistance.

(a) How are the classes of the source code and the Swing library related to the Observer Pattern and the Composite Pattern? For each class in the following table, specify the role in the Composite Pattern and the role in the Observer Pattern. If you cannot find a match for a class in a pattern, type "n/a".

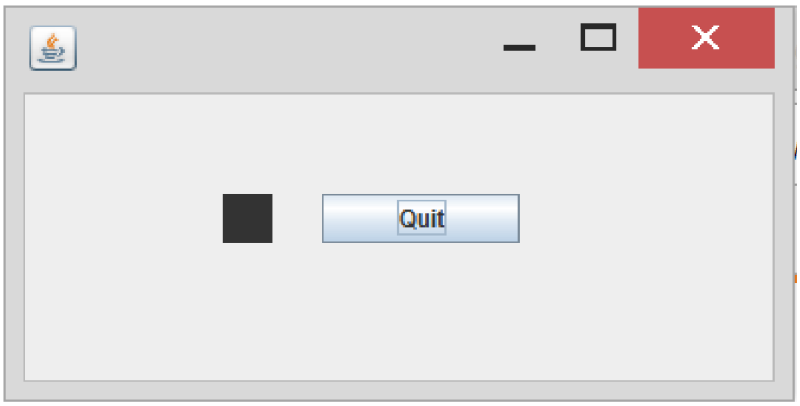
Class in Reference	Role in Composite Pattern	Role in Observer Pattern
AbstractButton		
ActionEvent		
ActionListener		
Application		
Component		
Container		
Graphics		
JButton		
JFrame		
JComponent		
JPanel		
MainPanel		
Window		

(b) How are the methods related to the Observer Pattern and the Composite Pattern? Specify a mapping between the methods from the reference and the methods from the respective pattern.

Method in Reference	Method in Composite Pattern	Method in Observer Pattern
actionPerformed		
add		
addActionListener		
paint		
paintComponents		

(c) Have a look at the class diagram of the Swing library in the reference. How does the Composite Pattern in the Swing library differ from the Composite Pattern in the lecture (see below) regarding leaf nodes?

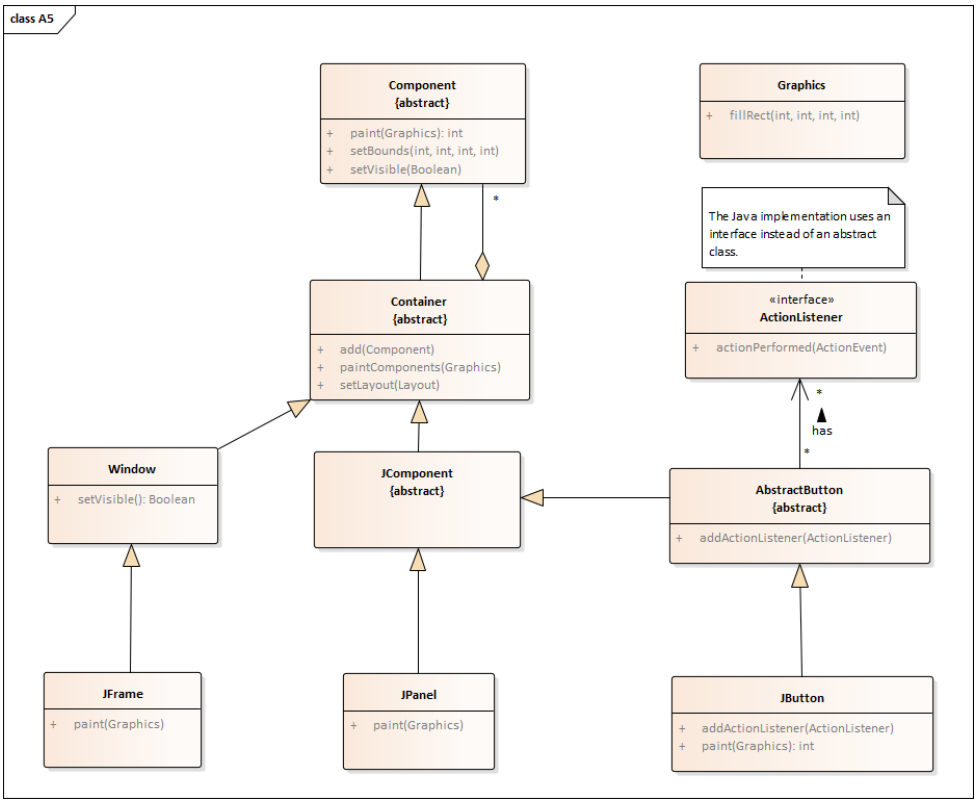
Reference:



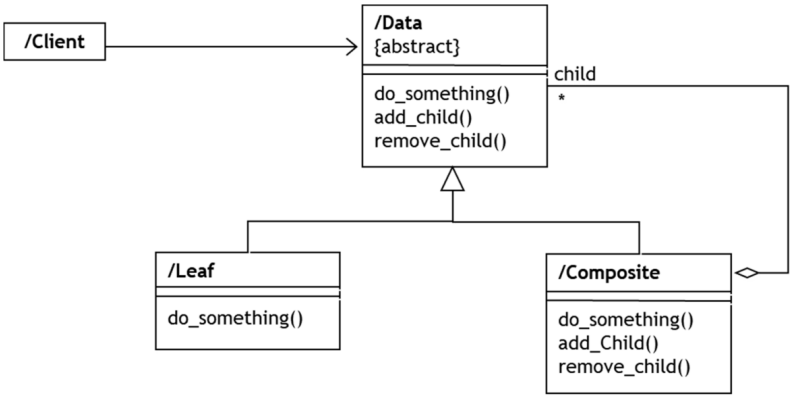

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Graphics;
import java.awt.Window;
import javax.swing.AbstractButton;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

class MainPanel extends JPanel {
    @Override
    public void paint(Graphics g) {
        g.fillRect(100, 50, 25, 25);
```

```
            super.paintComponents(g);
        }
    }

    public class Application implements ActionListener {
        public Window createWindow() {
            AbstractButton quitButton = new JButton("Quit");
            quitButton.setBounds(150, 50, 100, 25);
            quitButton.addActionListener(this);

            JPanel panel = new MainPanel();
            panel.setLayout(null);
            panel.add(quitButton);

            Window window = new JFrame();
            window.setBounds(0, 0, 400, 200);
            window.add(panel);
            return window;
    }

        public void actionPerformed(ActionEvent event) {
            System.exit(0);
        }

        public static void main(String[] args) {
            Application app = new Application();
            Window window = app.createWindow();
            window.setVisible(true);
        }
    }
```

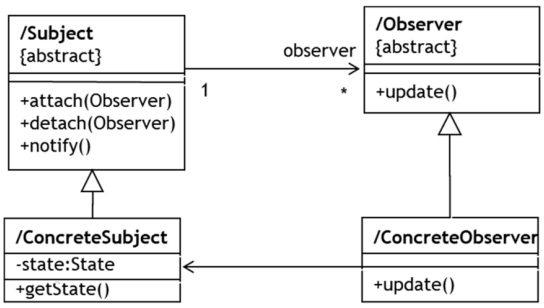
Class Diagram of the Swing library (simplified)



Composite-Pattern from lecture



Observer-Pattern from lecture



2.4 Databases

Exercise 13. Functional dependencies and schema design.

Given is the relation $R_A = \{A, B, C, D, E, F\}$ and the set of functional dependencies $\mathcal{F}_A = \{AF \rightarrow B, A \rightarrow C, EF \rightarrow D, D \rightarrow AF, F \rightarrow E, D \rightarrow C\}$.

(a) Prove or disprove that $\{EF\}$ is a unique key.

Given are $R_N = \{A, B, C, D, E\}$ and $\mathcal{F}_{min} = \{A \rightarrow C, B \rightarrow A, BE \rightarrow D, D \rightarrow B\}$. Unique keys are $\{BE\}$ and $\{DE\}$.

(a) Name all breaches of the 3rd Normal Form that occur within the universal relation R_N.

(b) Compute a schema in 3NF for the given universal relation R_N and the minmal set of functional dependencies $\mathcal{F}_{min}$ using the schema synthesis approach.

Given is $R = \{A, B, C, D, E, F\}$ and the set of functional dependencies $\mathcal{F} = \{A \rightarrow B, FB \rightarrow C, F \rightarrow AD, F \rightarrow B, E \rightarrow BCD\}$.

(a) Compute the minimal functional dependency set for $\mathcal{F}$.

Exercise 14. Recovery.

(a) Given is the following history of transactions. The objects A, B, C and D are stored on pages with page IDs P_A, P_B, P_C or P_D respectively. At the beginning, the values of the objects are $A = 0, B = 0, C = 0$ and $D = 0$.

Hint: $write(C, V)$ means the value V is written to object C.

	$T1$	$T2$
1.	BOT	
2.		BOT
3.	$write(A, 10)$	
4.		$write(B, 20)$
5.	$write(C, 30)$	
6.	commit	
7.		$write(D, 40)$

Assign each of the following disk states a combination of the replacement strategies steal or ¬steal and force or ¬force. Each combination has to be assigned *exactly once*.

Replacement strategy	A	B	C	D
	10	20	30	0
	0	0	30	0
	10	0	30	0
	0	20	30	0

(b) Complete the log entries for the above given history within the following table.

LSN	TA ID	Page ID	Redo	Undo	Prev-LSN
#1	T1	BOT			
#2	T2	BOT			
#3					
#4					
#5					
#6					
#7					
#8					

(c) If we assume that the log is empty before starting the above given exercise. How many lines the log contains *at least* after having performed the operations of line 7 of the history on the database? Justify your answer

(d) Assume that we use the combination of replacement strategies steal and ¬force and the following data is stored on disk:

P_A	P_B	P_C	P_D
LSN $= \#3$ $A = 10$	LSN $= \#0$ $B = 0$	LSN $= \#0$ $C = 0$	LSN $= \#7$ $D = 40$

(a) What are the LSNs of log entries that are redone and undone during the redo and undo phase of recovery. Respect the correct order of each phase.

Redo:

Undo:

(b) Which CLRs are added to log during the undo phase?
Hint: You might not need all lines of the table.

LSN	TA ID	Page ID	Redo	Prev-LSN	Undo-NextLSN

Exercise 15. Logical query optimization.

Given is the following SQL query:

```
SELECT title FROM Newspaper, Authors, Article
    WHERE Newspaper.name=worksfor
    AND Authors.name=author
    AND age > 50
    AND price > 310;
```

(a) Draw the logical query tree for this SQL query. Assume the following relations (prime attributes are printed **bold**, foreign keys are printed *in intalics*.)

Newspaper:

name	price
NY Post	300
LA Times	320
Wall Street Journal	360
USA Today	180
Chicago Tribune	0

Authors:

name	age	gender	*worksfor*
Peter	28	male	NY Post
Lars	46	male	NY Post
Susanne	28	female	LA Times
Elvis	36	male	Wall Street Journal
Sarah	54	female	Wall Street Journal
Anne	28	female	USA Today
Bertie	62	male	USA Today
Rolf	26	male	Chicago Tribune

Article:

id	title	*author*	*publishedin*
10001	Young and successfull	Peter	NY Post
10002	Black Friday Deals	Susanne	LA Times
10003	Costs of Growth	Susanne	LA Times
10004	The Brutal Reality of Brexit	Elvis	Wall Street Journal
10005	DAX rises above 16000	Sarah	Wall Street Journal
10006	No pain, no gain	Anne	USA Today
10007	Cutting corners	Bertie	USA Today
10008	Bad offers	Rolf	Chicago Tribune

(b) Draw the optimal query tree using the rules of logical query optimization. Calculate the number of intermediate results and the selectivity of all selections and joins in relation to the input size.

3 Modelling, Algorithms and Complexity

3.1 Logics and Modelling

Exercise 16. Logic.

- Let the following two propositional logic formulae over the atoms A, B, C be given:

$$\alpha = (A \vee C) \wedge \big(\neg A \vee (\neg B \wedge C)\big)$$
$$\beta = \neg C \rightarrow B$$

  (a) Complete the following truth table:

A	B	C	β	$A \vee C$	$\neg B \wedge C$	$\neg A \vee (\neg B \wedge C)$	α
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

  (b) Is the formula α a tautology, satisfiable or contradictory?

  (c) Are α and β logically equivalent?

  (d) Is β a semantic entailment of α?

- Let p be a unary predicate. Are the predicate logic formulae $\exists x\ \neg p(x)$ and $\neg \forall x\ p(x)$ logically equivalent?

Exercise 17. Sets.

Give the extensional description and the cardinality of the following sets U, V and W. In this, $A = \{a, b\}$ und $B = \{b, c\}$.

(a) $U = \emptyset \cup \{\emptyset\} \cup \{\{\emptyset\}\}$

$U =$ _____ $|U| =$

(b) $V = (A \times A) \cap (A \times B)$

$V =$ _____ $|V| =$

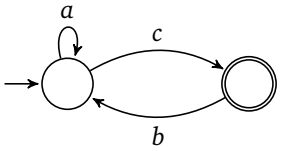
(c) $W = (A \times B) \setminus ((A \times \{\emptyset\}) \cup (B \times A))$

$W =$ _____ $|W| =$

Exercise 18. Regular languages.

(a) Give a finite state automaton accepting the regular language $(a + b)^*.c.d^*$.

(b) Give a regular expression describing the language accepted by the following finite state automaton.



3.2 Computability and Complexity

Exercise 19. The symmetric difference $L_1 \Delta L_2 = (L_1 \vee L_2) \setminus (L_1 \wedge L_2)$ of two languages $L_1, L_2 \subseteq \Sigma^*$ is the set of all words of Σ^* that are contained in exactly one of the two languages. Show the following: If L_1, L_2 are decidable, then $L_1 \Delta L_2$ is also decidable.

Hint: *Start with a deterministic Turing machine (DTM) M_1 that decides L_1 and a second DTM M_2 that decides L_2. Then construct a 2-band DTM $M_{L_1 \Delta L_2}$, that decides $L_1 \Delta L_2$.*

Exercise 20. Can the following languages be enumerated recursively? Justify your answer. If a language can be enumerated recursively, describe a DTM that accepts the language. Otherwise, show that it is not enumerable by doing a reduction from a suitable language.

a) $L_1 := \{\langle M \rangle \mid M \text{ holds for at least one input}\}$

b) $L_2 := \{\langle M \rangle \mid M \text{ holds for infinitely many inputs}\}$

c) $L_3 := \{(\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle) \mid L(M_1) = L(M_2) \cap L(M_3)\}$

Exercise 21. We define the problem *One-in-three-3SAT* (1in3SAT): The problem 1in3SAT is defined as 3SAT, but with the difference that it has to be decided whether a fulfilling assignment of the variables exists, so that in each clause exactly one literal is true.

Show that 3SAT $\leq_p$ 1in3SAT ($\leq_p$ = polynomially reducible).

3.3 Data Structures and Algorithms

Exercise 22. Binary Max Heaps.

A binary max heap is a data structure to store keys. It is defined by the the following two properties:

(a) **Form invariant**
A binary max heap is a *complete binary tree*. That means, all the tree's levels (except possibly the last) are fully filled, i.e., each node has exactly two children. If the tree's last level is not complete, the nodes of that level are filled from left to right.

(b) **Heap invariant**
The key stored in each node is greater or equal to the keys in the node's children.

A binary max heap may be stored using an array A. The keys are stored in the following way:

(a) The root is at $A[0]$.

(b) The children of the element stored in $A[i]$ are stored in $A[2i+1]$ and $A[2i+2]$ respectively.

(c) The parent of the element stored in $A[i]$ is stored in $A\left[\lfloor \frac{i-1}{2} \rfloor\right]$.

Tasks

(a) Observe the array $A = [5, 10, 10, 4, 9, 6, 8, 3, 2, 8, 6]$.
Draw the binary tree implied by this array and the rules given above.

(b) Does array A store a correct binary max heap?
If not, state which of the heap's properties are violated.

(c) Apply the operation MAXHEAPIFY(A,0) given below to array A.
Draw the corresponding tree before each (recursive) execution of MAXHEAPIFY.

Algorithm 1 MAXHEAPIFY(A,i)

```
1:  n ⟵ SIZE(A)                          ▷ Denotes the maximal number of elements in A
2:  l ⟵ 2i + 1
3:  r ⟵ 2i + 2
4:  largest ⟵ i
5:  if l < n and A[l] > A[largest] then
6:      largest ⟵ l
7:  if r < n and A[r] > A[largest] then
8:      largest ⟵ r
9:  if largest ≠ i then
10:     SWAP(A[i],A[largest])            ▷ Swaps the elements in A[i] and A[largest]
11:     MAXHEAPIFY(A, largest)
```

Exercise 23. Dijkstra's Algorithm.

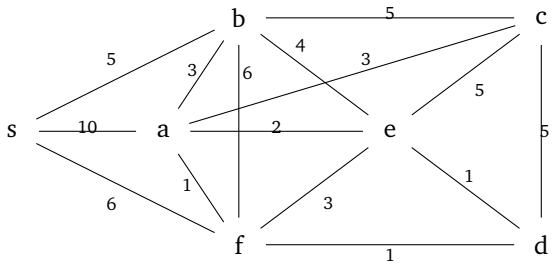
Given an undirected weighted graph $G := (V, E, w)$ and a stating node $s \in V$ Dijkstra's algorithm can be used to find the shortest path from s to all other nodes. The following pseudocode describes the algorithm:

Algorithm 2 DIJKSTRA(G:=(V,E,w),s)

```
1:  for v ∈ V do
2:      d[v] ← ∞
3:      π[v] ← nil
4:  d[s] ← 0
5:  S ⟵ ∅
6:  Q ⟵ V
7:  while Q ≠ ∅ do
8:      u ← EXTRACTMIN(Q)                ▷ Removes element with lowest distance from Q
9:      S ← S ∪ {u}
10:     for (u, v) ∈ E do
11:         if d[u] + w(u, v) < d[v] then
12:             d[v] ← d[u] + w(u, v)
13:             π[v] ⟵ u
```

Task

Observe the following unidirected weighted graph G:



Apply Dijkstra's algorithm to G with s as the starting node.
After each execution of the while loop give the following properties:

(a) The content of the set S.

(b) The length of the most recently found path.

(c) For each node $v \in V \setminus S$ its current distance $d[v]$ to s and its predecessor $\pi[v]$.

4 Operating Systems and Computer Architecture, Security

4.1 Operating Systems

Exercise 24. Scheduling – Round Robin.

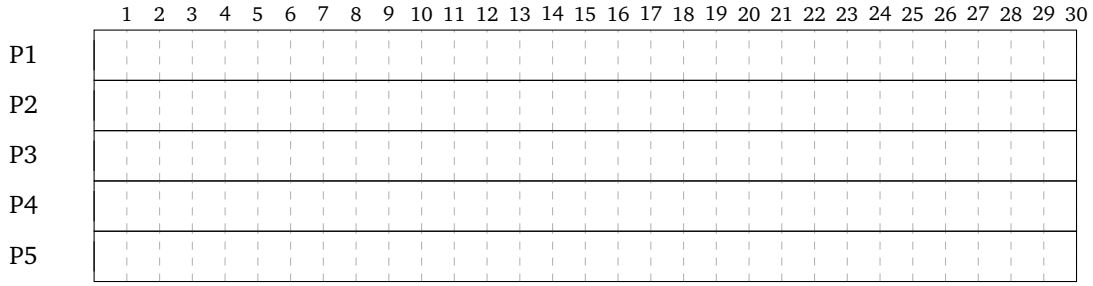
We assume a single CPU and single core system!

Let P1 to P5 be processes with the arrival and service times given in Table 1. A process is selected by Round Robin scheduling and then processed using a time quantum of 2 time units. New processes will be appended at the end of the ready queue. If a process is completed within its time quantum, the next waiting process is scheduled immediately.

Process	Arrival time	Service time
P1	1	5
P2	2	4
P3	4	3
P4	6	8
P5	7	6

Table 1: Properties of the processes P1 to P5.

Fill the following Gantt diagram indicating which process is scheduled at which time unit. Processes that are ready-to-run shall be marked 'o' and the currently scheduled process shall be marked 'X'. To ease the algorithm, we assume that the process switch takes 0 time units.



Calculate the average response time:

Average response time $t_m =$ .....

.....

.....

Exercise 25. Resource allocation – Mutual Exclusion.

Consider the C program on page 16. We assume the C program has been compiled with the following options `-std=c11` and `-O0`, and linked with the option `-lpthread`. The program compiles without errors.

(a) Identify and mark critical sections by listing the respective line numbers.

.....

.....

.....

(b) Which output would be naively expected?

.....

.....

.....

(c) What's the name of the underlying problem?

.....

.....

.....

(d) Which output *could* appear due to this problem?

.....

.....

.....

(e) Modify the number (sticking to the original objective) in order to fix this bug (indicate line numbers and new code lines)!

.....

.....

.....

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int a = 3;
6  int* b;
7  int num = 0;
8
9  pthread_mutex_t calc_mutex;
10
11 void* print(void* id_p) {
12
13         if (num++ == 0) pthread_mutex_lock(&calc_mutex);
14
15         printf("%d\n", a + *b);
16
17         if (--num == 0) pthread_mutex_unlock(&calc_mutex);
18
19         return 0;
20 }
21
22 void* calc(void* id_p) {
23
24         pthread_mutex_lock(&calc_mutex);
25
26         --a;
27         b[0]++;
28
29         pthread_mutex_unlock(&calc_mutex);
```

```
30
31          return 0;
32  }
33
34  int main(int argc, char* argv[]) {
35          pthread_t id[10];
36          b = malloc(sizeof(int));
37          *b = 4;
38
39          for (int i=0; i < 10; ) {
40                  pthread_create(&id[i++], NULL, calc, 0);
41                  pthread_create(&id[i++], NULL, print, 0);
42          }
43          for (int i=0; i < 10; i += 1) {
44                  pthread_join(id[i], NULL);
45          }
46
47          free(b);
48
49          return 0;
50  }
```

Exercise 26. Computer networking – Round trip time.

We assume a client is connecting to a server using TCP. The data rate between client and server is limited to 10 Mbit/s. The distance between both systems is $l = 10\,000$ km and we assume a propagation speed of $c = 3 \times 10^8$ m/s. For simplicity, we also assume that our messages have a length of 0 B.

(a) Calculate the Round Trip Time (RTT) between client and server; provide the complete equation.

(b) What changes if the data rate is increased to 100 Mbit/s?

(c) What changes if the propagation speed is reduced to $c_{\text{neu}} = 2 \times 10^8$ m/s?

4.2 Computer Architecture and Security

Exercise 27. Logic minimization.

(a) Given the circuit in Figure 1.

Determine the set of minterms for the logic function $z = f(a, b, c, d, e)$:

$\mathcal{E} = \{$ _____

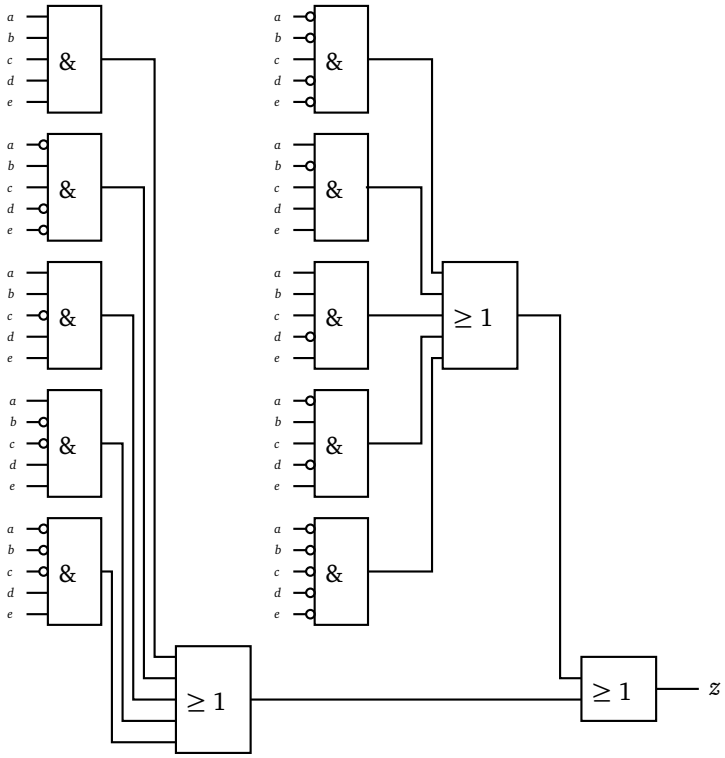


Figure 1: Circuit Netlist

}

(b) Given the set of minterms

$$\mathscr{E} = \{11000, 11100, 01011, 11011, 00011,$$
$$00111, 10011, 00001, 01001, 00000\}$$

Find the set of all prime implicants $\mathscr{P}$ out of $\mathscr{E}$ using the Quine-McCluskey algorithm. Use the given table.

L_0	✓						

$$\mathscr{P} = \{\_\_\_\_\_\}$$

(c) Given the logic function f with four inputs a, b, c and d. The sum-of-products form is given by

$$f = m_2 + m_5 + m_6 + m_7 + m_9 + m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15}$$

Fill out the truth table given in Table 3.

Hint: Each entry in the truth table represents a binary coded number. The entry 0000 *is abbreviated with* m_0.

(d) Assume that the first phase of the Quine-McCluskey algorithm provides the following prime implicants:

$$--10, -1-1, -11-, 1--1, 1-1-, 11--$$

Find a minimum subset of prime implicants covering all minterms using the second phase of the Quine-McCluskey algorithm.

Prime implicant chart 1:

P	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$

a	b	c	d	f
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Table 2: Truth table for f

List all essential prime implicants which can be identified in the first round:

$\{$ _____ $\}$

Prime implicant chart 2:

P	m___	m___	m___	m___	m___	m___	m___	m___	m___	m___	m___

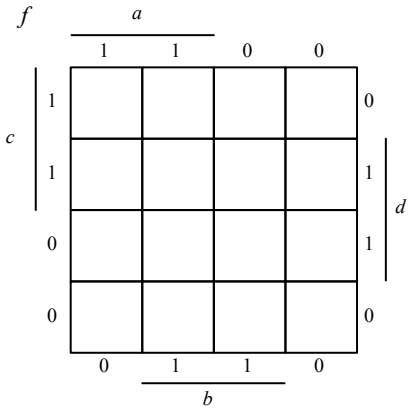
List the minimum subset of prime implicants covering all minterms:

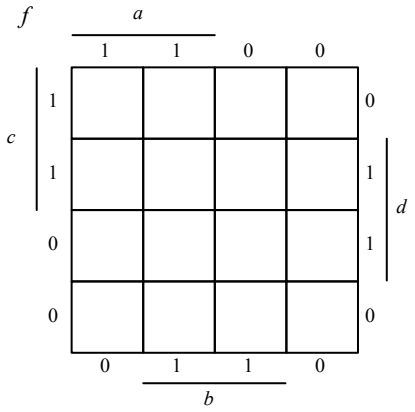
$\{$ _____ $\}$.

(e) Given the set of minterms

$$\mathcal{E} = \{0010, 0101, 0110, 0111, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$$

for the logic function $f(a, b, c, d)$. Find the minimal sum-of-products form for f using the given Karnaugh-map:





Spare K-map. **Cross out invalid solution!**

Minimal sum-of-products form for f:

{ _____ }.

Exercise 28. Multiple choice.

For the following questions, there might be none, one or several correct answers per question. Tick off all correct answers.

(a) What is the result of this MIPS assembly language program:

```
# x in $t0, y in $t1
xor $t1, $t0, $t1
xor $t0, $t1, $t0
xor $t1, $t1, $t0
```

☐ `x in $t0, y in $t1`

☐ `x in $t0, x in $t1`

☐ `0 in $t0, 0 in $t1`

☐ `y in $t0, x in $t1`

(b) What are characteristics of a RISC instruction set architecture?

☐ fixed instruction length

☐ many and complex instructions

☐ load/store architecture

☐ register/memory operations

(c) Which statements hold for a write-back cache?

☐ Cache and main memory are kept consistent.

☐ Each cache block requires a dirty bit.

☐ Write-back works only for set associative caches.

(d) What is the average rotational latency for a hard disk with 15000 rpm?

☐ 0.5 ms

☐ 1 ms

☐ 2 ms

☐ 3 ms

(e) What are similarities of superscalar and VLIW processors?

☐ Both are multiple issue processors.

☐ For both the compiler assigns instructions to execution units.

☐ Both have an ideal CPI of less than one.

☐ Both can profit from instruction caches.

Exercise 29. Pipelining.

Consider a processor with a 5-stage pipeline (IF, ID, EX, ME, WB). The processor does not use forwarding and the register file can either be read or written within a single clock cycle.

The following assembly language program is executed on this processor:

```
1:  lw   $1,  100($3)
2:  sub  $2,  $4,  $1
3:  add  $5,  $2,  $1
4:  mul  $6,  $2,  $1
5:  or   $7,  $2,  $1
6:  add  $8,  $2,  $1
7:  sw   $8,  100($3)
```

(a) Resolve any pipeline hazards by stalling the pipeline and fill in the diagram in Figure 2 to show the pipeline allocation.

(b) Consider an improved processor design that supports forwarding and where writes to the register file occur in the first half of a clock cycle and reads occur in the second half of the clock cycle. Fill in the diagram in Figure 3 to show the pipeline allocation. Indicate forwarding with arrows.

(c) What is the speedup achieved with version (b) over version (a)?

Exercise 30. Consider a 2 GHz processor with a two-level cache. The L1 cache has a hit time of 0,5 ns and a miss rate of 10%. The L2 cache has a hit time of 24 clock cycles, a hit rate of 95% and a miss penalty of 200 ns. Both caches use the same block size.

(a) What is the average memory access time for this two-level cache?

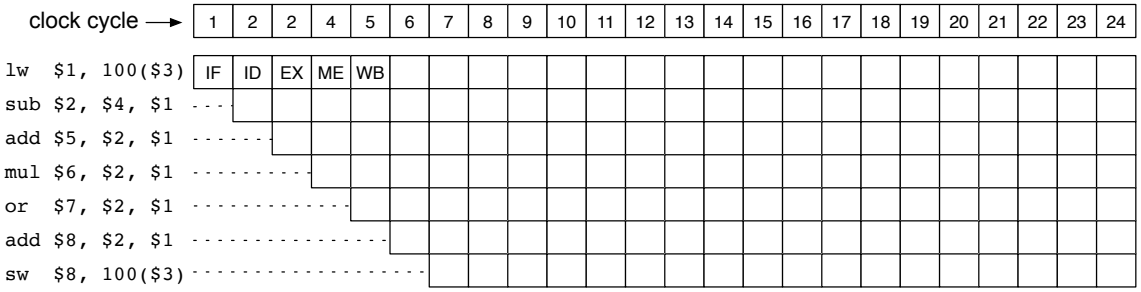


Figure 2: Pipeline allocation (a)

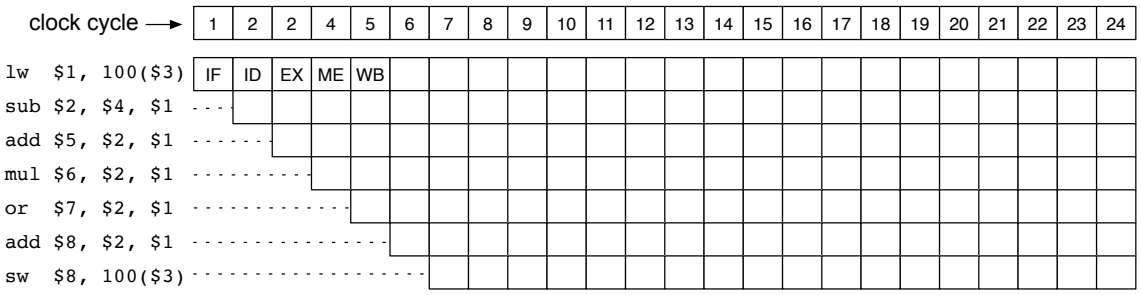


Figure 3: Pipeline allocation (b)

(b) Designers have come up with an alternative solution that employs a one-level cache. This cache has a hit rate of 99%, a miss penalty of 200 ns and a hit time of 0,5 ns. What is the average memory access time for this cache? Should it be preferred over the two-level cache?

Exercise 31. IT-Security.

(a) What is "Salting" in the context of password hashing, and which type of attacks does it prevent?

(b) What is a SYN-Flooding Attack? Which technique can be used to prevent this attack?

(c) Explain "Kerckhoffs' Principle" and its meaning for cryptography!

Part II

Solutions

5 Mathematics

Solution to Exercise 1:

(a) We test whether v_1, v_2, v_3 are linearly independent. Let $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ with

$$\alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = 0. \tag{1}$$

This equality is equivalent to the following three equations

$$\begin{array}{rcrcrcl} -\alpha_1 & & & + & -2\alpha_3 & = & 0, \\ 3\alpha_1 & + & \alpha_2 & + & 3\alpha_3 & = & 0, \\ \alpha_1 & + & \alpha_2 & - & \alpha_3 & = & 0. \end{array}$$

The first equation yields $\alpha_1 = -2\alpha_3$. The second equation then yields $\alpha_2 = -3\alpha_1 - 3\alpha_3 = 6\alpha_3 - 3\alpha_3 = 3\alpha_3$. The third equation yields no further information: it is a consequence of the first two equations by adding twice the first equation to the second equation.

Hence for any $\alpha_3 \in \mathbb{R}$ there is a unique solution $(\alpha_1, \alpha_2, \alpha_3)$ of the equation (1). For example, choosing $\alpha_3 = 1$ yields the linear dependence relation

$$-2v_1 + 3v_2 + v_2 = 0.$$

Hence v_1, v_2, v_3 are not linearly independent. In particular, they do not form a basis of $\mathbb{R}^3$.

(b) From the lecture course we know that v_1, v_2, v_4 form a generating set of $\mathbb{R}^3$ if and only if the determinant of the matrix

$$A := \begin{pmatrix} -1 & 0 & 1 \\ 3 & 1 & t^2 - 2 \\ 1 & 1 & t \end{pmatrix}$$

(whose columns are the given vectors) is non-zero. We compute this determinant using the rule of Sarrus.

$$\det(A) = -t + 0 + 3 - (1 + 0 - (t^2 - 2)) = t^2 - t = t(t-1).$$

This determinant is zero if and only if $t \in \{0, 1\}$. Hence the vectors v_1, v_2, v_4 form a generating set of $\mathbb{R}^3$ if and only if $t \in \mathbb{R} \setminus \{0, 1\}$.

Solution to Exercise 2:

We compute the characteristic polynomial χ_A of A.

$$\chi_A = \det \begin{pmatrix} X - a & -b \\ -b & X - c \end{pmatrix} = (X-a)(X-c) - b^2 = X^2 - (a+c)X + ac - b^2.$$

The zeros of χ_A in $\mathbb{R}$ are the eigenvalues of $A\colon \mathbb{R}^2 \to \mathbb{R}^2$ (known from the lecture course). The solution formula for the zeros of a quadratic polynomial shows that the zeros of χ_A (in $\mathbb{C}$) are

$$\frac{1}{2}\Big(a + c \pm \sqrt{(a+c)^2 - 4(ac - b^2)}\Big).$$

Let D be the term under the square root sign. Then

$$D = a^2 + 2ac + c^2 - 4ac + 4b^2 = a^2 - 2ac + c^2 + 4b^2 = (a-c)^2 + (2b)^2$$

is a sum of squares of real numbers and hence a non-negative real number. We distinguish two cases.

Case $D > 0$: Then χ_A has the two distinct zeros $\frac{1}{2}(a + c + \sqrt{D})$ and $\frac{1}{2}(a + c - \sqrt{D})$, i. e. A has two distinct eigenvalues. From the lecture course it is known that any linear endomorphism of $\mathbb{R}^n$ with n distinct eigenvalues is diagonalizable. Hence A is diagonalizable.

Case $D = 0$: From $0 = D = (a-c)^2 + 4b^2$ we deduce $a - c = 0$ and $b = 0$. Hence $A = \begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}$ is a diagonal matrix and trivially diagonalizable.

6 Programming, Programming Languages, Software Engineering, Databases

Solution to Exercise 3:

```
public class EvenCounter {
    public static int countEven(int[] arr) {
        int count=0;
        for (int i=0; i < arr.length; i++) {
            if (arr[i] % 2 == 0)
                count = count + 1;
        }
        return count;

    }
    public static void main(String[] args){
        int[] arr = {5,56,23,77,-5,99,100,-6};
        System.out.println(countEven(arr));
    }

}
```

Solution to Exercise 4:

The function f calculates the sum of all numbers from a to b. It uses direct recursion. Sample source code (Python version):

```
def f(a, b):
    if a == b:
        return b
    m = int((a + b) / 2)
    return f(a, m) + f(m + 1, b)
```

Solution to Exercise 5:

```
public class BinTree {
      BinTree left;
      BinTree right;
      int data;

      public BinTree (BinTree l, BinTree r, int d) {
          left = l;
          right = r;
          data = d;
      }

      public BinTree (int d) {
          left = null;
          right = null;
          data = d;
      }

      public int countLeafs() {
          int number = 0;
          if (left != null)
              number = number + left.countLeafs();
          if (right != null)
              number = number + right.countLeafs();
          if (left == null && right == null)
              return 1;
          else
              return number;
      }
  }

public class BinTreeExercise {
```

```
    public static void main(String[] args) {
        BinTree bt = new BinTree
                            (new BinTree(new BinTree(4),new BinTree(5),2),
                    new BinTree(3),
                    1);
        System.out.println(bt.countLeafs());
    }
}
```

Solution to Exercise 6:

```
def f(m):
    def g(L):
        return [x for x in L if x >= m]
    return g

g = f(2)
print(g([1, 2, 3]))
```

Solution to Exercise 7:

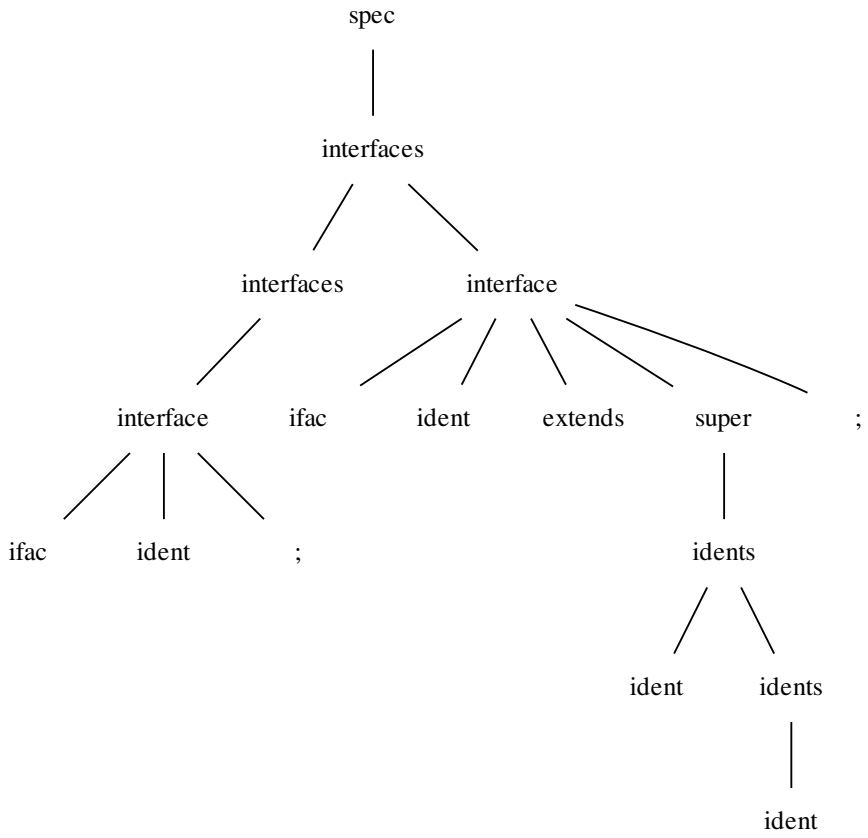
It outputs `Forty-Two` and `42`, in this order. Solution to Exercise 8:

(a) The sets of terminals `T` and nonterminals `N`:

```
T = {'ifac', ident, 'extends', ';'}
NT = {spec, interfaces, interface, super, idents]
```

(b) The derivation tree for the sentence:

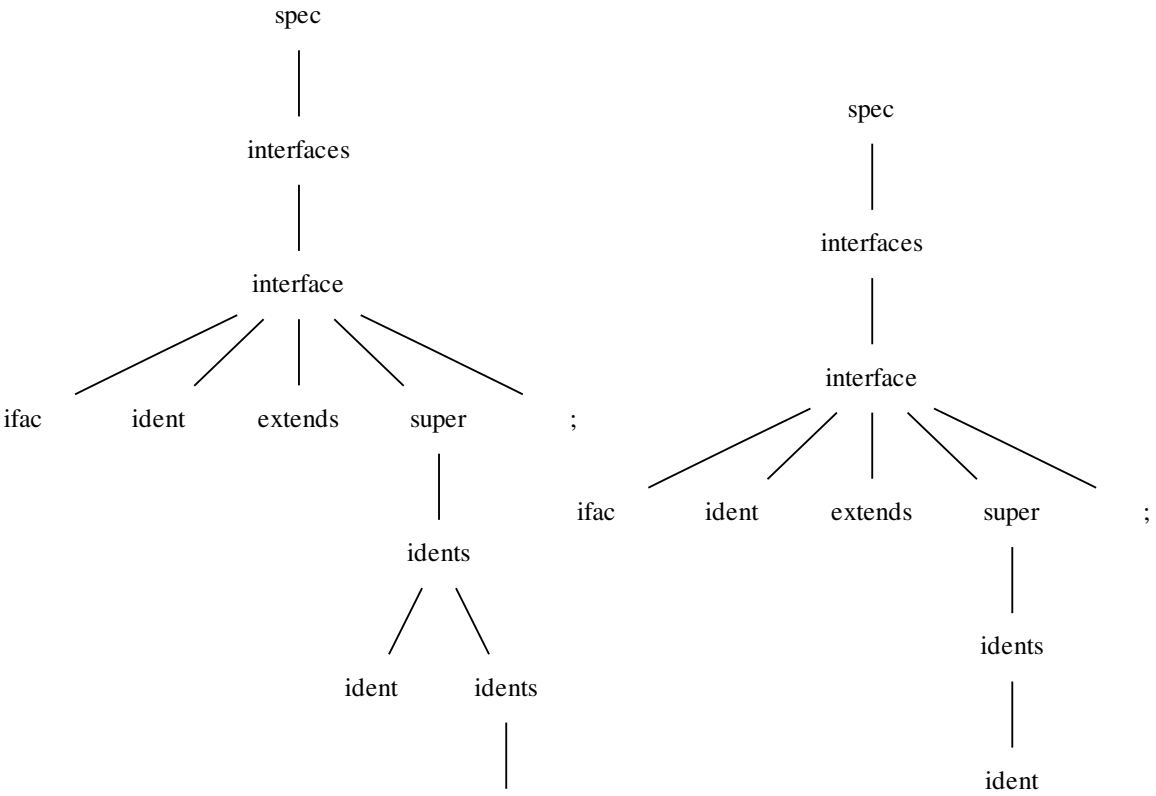
```
ifac ident ; ifac ident extends ident ident ;
```



(c) A grammar is ambiguous if there is a sentence in the language of this grammar for which there are two derivation trees. For our grammar such a sentence is

```
ifac ident extends ident ;
```

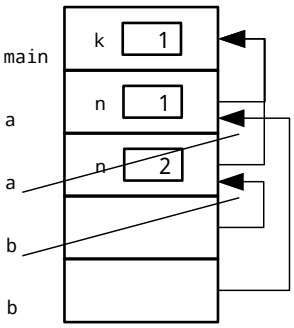
It has two different derivation trees:



Solution to Exercise 9:

(a) Runtime stack snapshot at the point of time where “Halt” is being output.

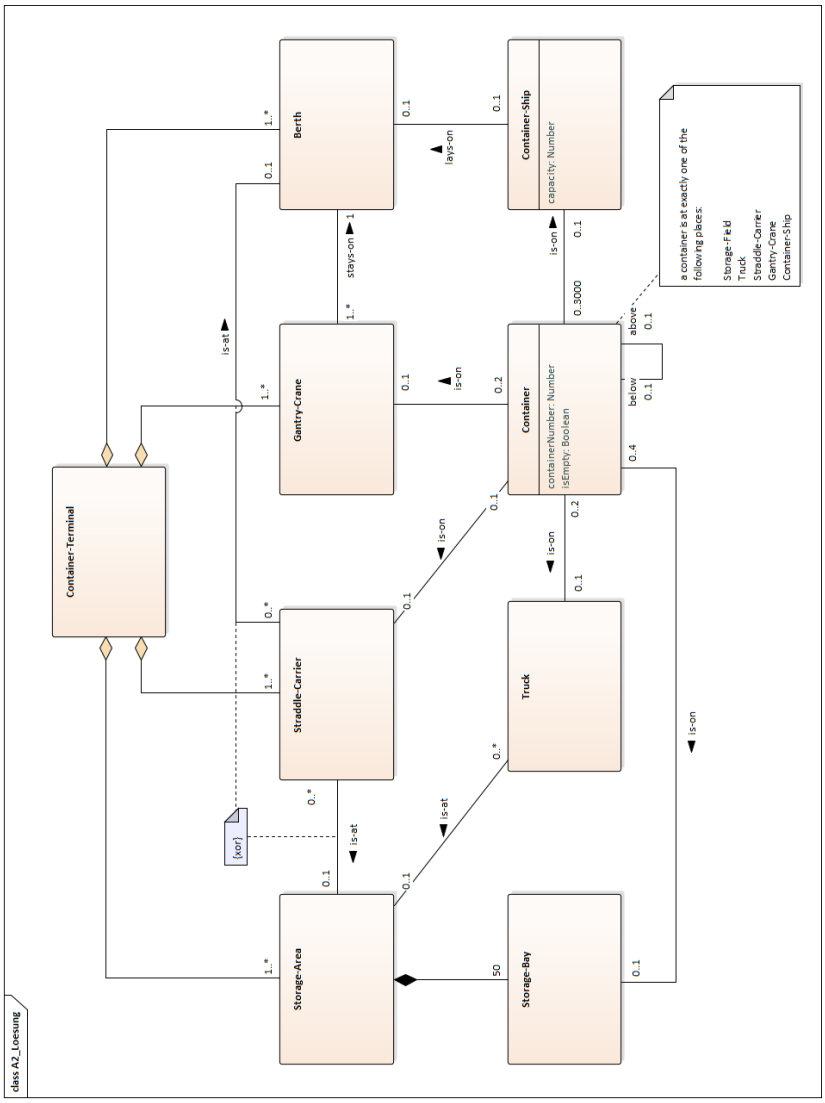
```
1  proc main() {
2     k = 1;
3     proc a(n) {
4        proc b() {
5          if (n == k) {
6             print "Halt";
7          }
8        }
9
10       if (n == 1) {
11          a(n + 1);
12       }
13       b();
14    }
15    a(k);
16 }
```



(b) The static link points to the frame of the function that contains the definition of the current function. It is necessary for languages with nested functions to allow access to values (parameters, variables) stored in the stack frames of surrounding functions.

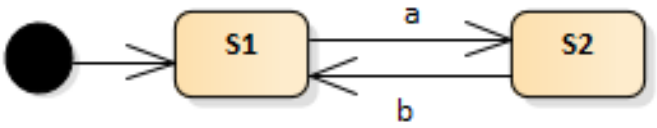
For our example the static links are needed to access the values of `k` and `n` in function `b`.

Solution to Exercise 10:

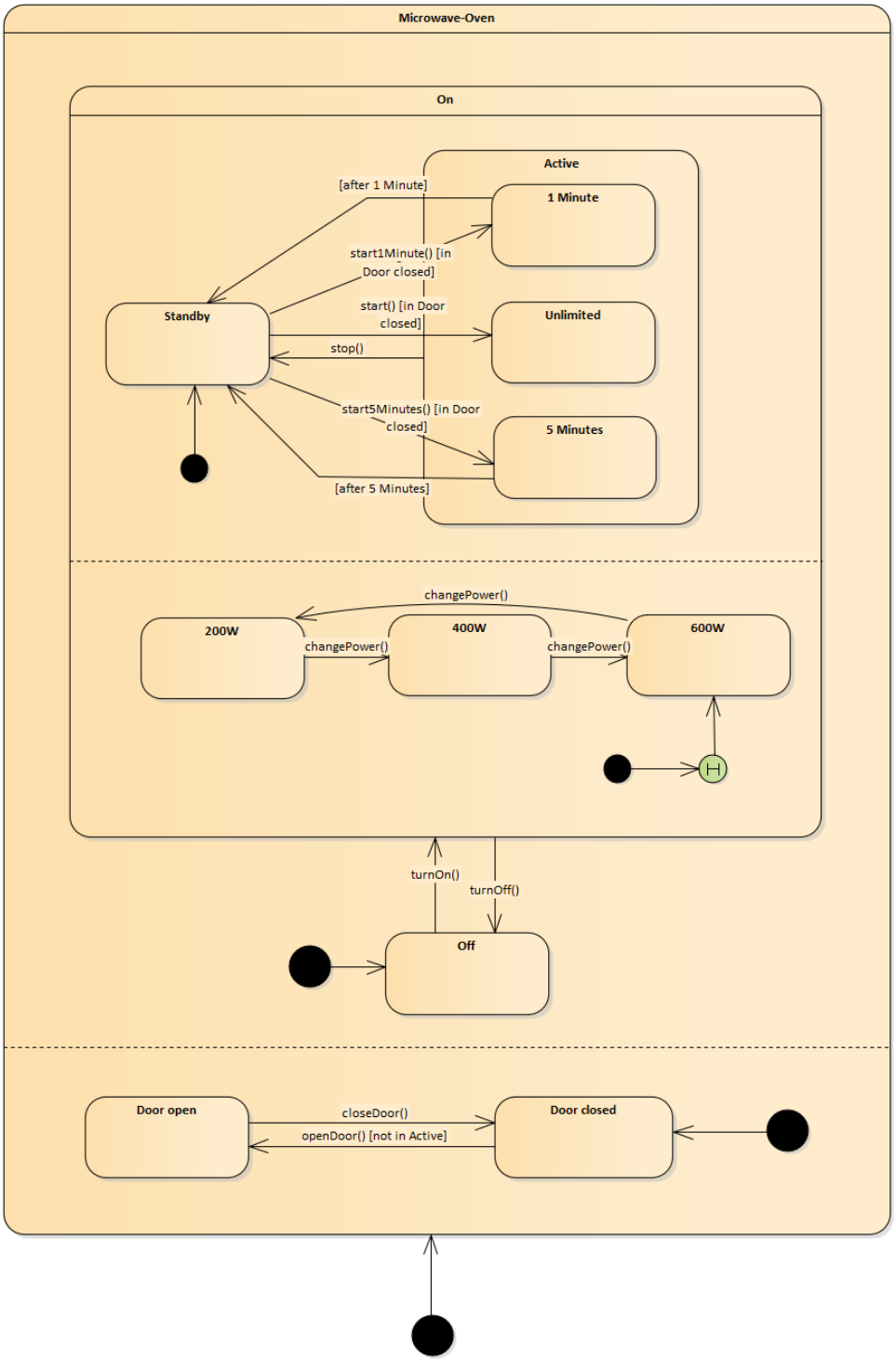


Solution to Exercise 11:

(a)



(b)



Solution to Exercise 12:

(a)

Class in Reference	Role in Composite Pattern	Role in Observer Pattern
AbstractButton	/Composite	/Subject
ActionEvent	n/a	/State
ActionListener	n/a	/Observer
Application	/Client	/ConcreteObserver
Component	/Data	n/a
Container	/Composite	n/a
Graphics	n/a	n/a
JButton	/Composite	/ConcreteSubject
JFrame	/Composite	n/a
JComponent	/Composite	n/a
JPanel	/Composite	n/a
MainPanel	/Composite	n/a
Window	/Composite	n/a

(b)

Method in Reference	Method in Composite Pattern	Method in Observer Pattern
actionPerformed		update
add	add_child	
addActionListener		attach
paint	do_something	
paintComponents	do_something	

(c) There are no Leaf Nodes. Leaf nodes must be a sub-class of component (see class diagram).

Solution to Exercise 13:

(a) $\{EF\}$ is a superkey, because $EF \rightarrow R_A$ (according to the RAP algorithm). But $EF \rightarrow R_A$ ($F \rightarrow E$), $\{F\}$ which is a proper subset of $\{EF\}$ is a superkey as well. Thus, $\{EF\}$ is not minimal and therefore, $\{EF\}$ is *not* a unique key.

Given are $R_N = \{A, B, C, D, E\}$ and $\mathscr{F}_{min} = \{A \rightarrow C, B \rightarrow A, BE \rightarrow D, D \rightarrow B\}$. Unique keys are $\{BE\}$ and $\{DE\}$.

(a) $A \rightarrow C$ breaches 3NF as A is no superkey and C no prime attribute.

$B \rightarrow A$ breaches 3NF as B is no superkey and A is no prime attribute.

(b) For $BE \rightarrow D$ create:
$R_1 = \{B, D, E\}$
$\mathscr{F}_1 = \{BE \rightarrow D\, D \rightarrow B\}$
For $A \rightarrow C$ create:
$R_2 = \{A, C\}$
$\mathscr{F}_2 = \{A \rightarrow C\}$
For $B \rightarrow A$ create:
$R_3 = \{A, B\}$
$\mathscr{F}_3 = \{B \rightarrow A\}$
As R_1 already contains a unique key, no further relation is created.

(a) (a) Simplifying the right-hand side: $F \rightarrow AD$ is replaced by $F \rightarrow A$ and $F \rightarrow D$; $E \rightarrow BCD$ is replaced by $E \rightarrow B$, $E \rightarrow C$ and $E \rightarrow D$

(b) Simplifying the left-hand side: $FB \rightarrow C$ is replaced by $F \rightarrow C$, because $F \rightarrow B$

(c) Remove all redundant FDs: $F \rightarrow B$ is redundant, because $F \rightarrow A$ and $A \rightarrow B$

(d) Combine all FDs with identical left-hand side: $\mathscr{F}_{min} = \{A \rightarrow B, F \rightarrow ACD, E \rightarrow BCD\}$

Solution to Exercise 14:

(a)

Replacement strategy	A	B	C	D
steal, force	10	20	30	0
¬steal, ¬force	0	0	30	0
¬steal, force	10	0	30	0
steal, ¬force	0	20	30	0

(b)

LSN	TA ID	Page ID	Redo	Undo	Prev-LSN
#1	T1	BOT			
#2	T2	BOT			
#3	T1	P_A	A+=10	A-=10	#1
#4	T2	P_B	B+=20	B-=20	#2
#5	T1	P_C	C+=30	C-=30	#3
#6	T1	commit			#5
#7	T2	P_D	D+=40	D-=40	#4
#8					

(c) 6 lines: According to WAL principle, before executing the commit of line 6, all log entries have to be written securely (i.e., to disk). Line 7 might be already written to log but does not need to be.

(d) Assume that we use the combination of replacement strategies steal and ¬force and the following data is stored on disk:

(a) **Redo:** #4, #5

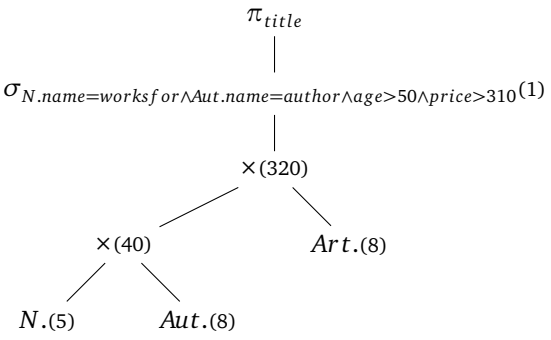
Undo: #7, #4

(b)

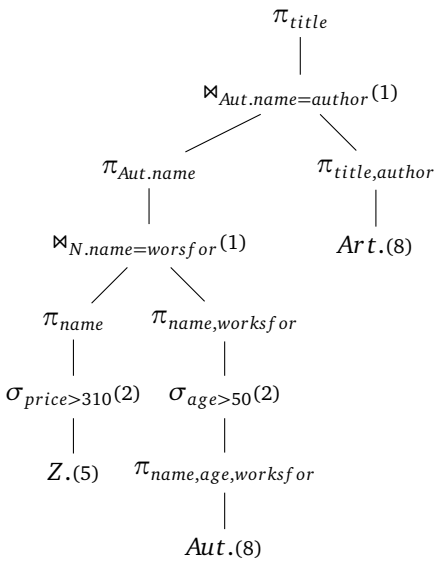
LSN	TA ID	Page ID	Redo	Prev-LSN	Undo-NextLSN
<#8>	T_2	P_D	D-=40	#7	#4
<#9>	T_2	P_B	B-=20	#8	#0

Solution to Exercise 15

(a)



(In total $40 + 320 + 1 = 361$ intermedate results are computed.)



(b)

$2+2+1+1=6$ intermediate results are computed. Selectivities:

- $\frac{2}{5}=0.4$ for $\sigma_{price>310}$
- $\frac{2}{8}=0.25$ for $\sigma_{age>50}$
- $\frac{1}{2\cdot 2}=0.25$ for $\bowtie_{N.name=worksfor}$
- $\frac{1}{1\cdot 8}=0.125$ for $\bowtie_{Aut.name=author}$

7 Modelling, Algorithms and Complexity

Solution to Exercise 16:

- (a)

A	B	C	β	$A\vee C$	$\neg B\wedge C$	$\neg A\vee(\neg B\wedge C)$	α
0	0	0	1	0	0	1	0
0	0	1	0	1	1	1	1
0	1	0	1	0	0	1	0
0	1	1	1	1	0	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	1	1	1
1	1	0	1	1	0	0	0
1	1	1	1	1	0	0	0

  (b) α is satisfiable.

  (c) no

  (d) no

- yes

Solution to Exercise 17:

(a) $U=\emptyset\cup\{\emptyset\}\cup\{\{\emptyset\}\}$

$U=\quad\{\emptyset,\{\emptyset\}\}\qquad|U|=2$

(b) $V = (A \times A) \cap (A \times B)$

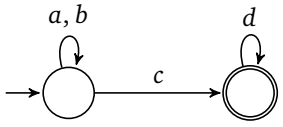
$V = \{(a,b),(b,b)\} \qquad |V| = 2$

(c) $W = (A \times B) \setminus ((A \times \{\emptyset\}) \cup (B \times A))$

$W = \{(b,a),(b,b)\} \qquad |W| = 2$

Solution to Exercise 18:

(a)



(b) $(a^*.c.b)^*.c$

Solution to Exercise 19:

Let M_1 be a DTM that decides L_1 and M_2 a DTM that decides L_2. We construct the following 2-band DTM $M_{L_1 \Delta L_2}$ that decides $L_1 \Delta L_2$.

$M_{L_1 \Delta L_2}$: Input w

(a) Copy w to band 2.

(b) Simulate M_1 with input w on band 1 and M_2 with input w on band 2.

(c) If both simulations accept or both reject, reject. Otherwise, accept.

We have to show that $L_1 \Delta L_2$ is the language that is decided by $M_{L_1 \Delta L_2}$ and that the DTM decides the language.

Let $w \in L_1 \Delta L_2$ be the input. That means that w is exactly of one of the two languages L_1, L_2. Therefore, the input w is accepted by exactly one of the turing machines M_1 and M_2, and rejected by the other one. This causes the machine $M_{L_1 \Delta L_2}$ to accept the input w which yields $w \in L(M_{L_1 \Delta L_2})$.

Let us now assume that $w \notin L_1 \Delta L_2$. That is, w is either in both languages or in none of them. But then the Turing machines M_1 and M_2 will either both accept the input w or both reject the input. So $M_{L_1 \Delta L_2}$ will also reject the input w and therefore, $w \notin L(M_{L_1 \Delta L_2})$.

From the fact that $M_{L_1 \Delta L_2}$ rejects any input that is not in $L_1 \Delta L_2$ follows, that $M_{L_1 \Delta L_2}$ decides the language $L_1 \Delta L_2$.

Solution to Exercise 20:

a) L_1 is recursively enumerable.
Proof by construction of a DTM M', which works as follows with input $w \in \{0,1\}^*$:

(a) If $w \neq \langle M \rangle$ for a DTM M, reject.

(b) Set $i = 1$.

(c) Simulate successively M for all x with $|x| \leq i$ for i steps.

(d) If M holds with input x within the i steps, accept.

(e) Otherwise set $i = i + 1$ and go to step c).

It remains to be shown that the DTM M' accepts the language L_1, i.e., it accepts all words from L_1 and does not accept all words that are not from L_1 (i.e., rejects or does not hold).

Let $w \in L_1$. Then $w = \langle M \rangle$, where M holds for at least one input z. Name s the number of steps after which M with input z holds. At the latest in step $i = \max\{|z|, s\}$, M is simulated with input z for i steps and holds. Thus, also M' *holds.*

Let $w \notin L_1$. Then $w \neq \langle M \rangle$ for a DTM M and M' rejects or $w = \langle M \rangle$, where M does not hold on any input. In particular, there is no i and no input x with $|x| \leq i$ for which M holds within i steps. Thus, also M' *does not hold.*

b) L_2 is not recursively enumerable.
We show $\bar{H} \leq L_2$. Define $f : \{0,1\}^* \to \{0,1\}^*$ for this:

$$f(w) = \begin{cases} \langle M^{accept} \rangle & w \neq \langle M \rangle x \text{ für eine DTM } M \text{ und } x \in \{0,1\}^* \\ \langle M_x \rangle & w = \langle M \rangle x \text{ für eine DTM } M \text{ und } x \in \{0,1\}^* \end{cases}$$

where M^{accept} is a DTM that accepts any input and M_x works as follows for input $z \in \{0,1\}^*$:

(a) Simulate M with input x for $|z|$ steps.

(b) If M with input x does not stop within $|z|$ steps, accept z.

(c) Otherwise, enter an infinite loop.

The f function is computable for the following reasons:

- The check if $w = \langle M \rangle x$ for a DTM M and $x \in \{0,1\}^*$ can be done with a DTM.
- M^{accept} is a DTM, which goes from the start state directly into the accepting state at every input, so M^{accept} is easy to construct and $\langle M^{accept} \rangle$ is also computable.
- M_x can be constructed from $\langle M \rangle$ and x by adding an additional counter to the simulation of M, which is checked after each step to see if a certain limit (length of the input word) has already been exceeded. $\langle M_x \rangle$ can also be computed.

It remains to show: $w \in \bar{H} \Leftrightarrow f(w) \in L_2$.

Let $w \in \bar{H}$.
$\Rightarrow w \neq \langle M \rangle x$ for a DTM M and $x \in \{0,1\}^*$ *or* $w = \langle M \rangle x$, where M is a DTM that does not hold with input $x \in \{0,1\}^*$.
Case 1: $f(w) = \langle M^{accept} \rangle$, where M^{accept} holds with all inputs $\Rightarrow f(w) \in L_2$.
Case 2: $f(w) = \langle M_x \rangle$. Since M with input x does not hold within finite many steps, there is no input z, so M holds within $|z|$ steps on input x. M_x thus accepts any input in the second step.
$\Rightarrow f(w) \in L_2$.

Let $w \notin \bar{H}$.
$\Rightarrow w = \langle M \rangle x$, where M is a DTM that holds with input $x \in \{0,1\}^*$. Let s be the number of steps after which M will hold with input x. Only for inputs $z \in \{0,1\}^*$ with $|z| < s$, M with input x does not hold, i.e., only for these inputs M_x holds. Since the number of $z \in \{0,1\}^*$ with $|z| < s$ is finite, M_x holds only on finite many inputs.
$\Rightarrow f(w) \notin L_2$.

c) L_3 is not recursively enumerable.
We reduce from the emptiness problem (should have been introduced in the lecture): $L_\emptyset = \{\langle M \rangle \mid L(M) = \emptyset\}$.

$$f(w) = \begin{cases} (\langle M_\emptyset \rangle, \langle M \rangle, \langle M_{accept} \rangle) & w = \langle M \rangle \text{ für eine DTM } M \\ (\langle M_\emptyset \rangle, \langle M_{accept} \rangle, \langle M_{accept} \rangle) & otherwise \end{cases}$$

Here M_{accept} is a DTM which accepts every input and $M_\emptyset$ is a DTM which accepts no input, so $L(M_{accept}) = \{0,1\}^*$ and $L(M_\emptyset) = \emptyset$. It is clear that there are corresponding DTMs with a trivial transition function, so f is computable.

Let $w \in L_\emptyset$. Then $w = \langle M \rangle$ with $L(M) = \emptyset$. So also the intersection of this language with every other set is empty and it follows taht$L(M_\emptyset) = \emptyset = L(M) \cap L(M_{accept})$. Thus, $f(w) \in L_3$.

Let $w \notin L_\emptyset$. Then either w is not a valid Gödel number, which obviously implies $f(w) \notin L_3$, or $w = \langle M \rangle$ with $\{x\} \subseteq L(M)$ for a $x \in \{0,1\}^*$. But then $L(M_\emptyset) = \emptyset \subset \{x\} \subseteq L(M) \cap L(M_{accept})$ and $f(w) \notin L_3$.

Solution to Exercise 21:

We prove this by reducing 3SAT into 1in3SAT.

The transformation works as follows: For each of the $c_i = \{x_{i1}, x_{i2}, x_{i3}\}$, we add 4 new variables, a_i, b_i, c_i, d_i and produce three new clauses:

$$\{\bar{x}_{i1}, a_i, b_i\}, \{\bar{x}_{i2}, b_i, c_i\}, \{\bar{x}_{i3}, c_i, d_i\}$$

Suppose the original $c_1 \wedge c_2 \wedge \ldots \wedge c_m$ has n variables and k clauses, our transformation will produce an instance of $n + 4k$ variables and $3k$ clauses. This transformation is obviously carried out in polynomial time.

Then, we need to prove that our transformed instance will have a solution under 1in3SAT condition if and only if the original instance does under 3SAT condition.

($\Rightarrow$) Firstly, suppose our transformed instance has a solution, which means exactly one literal per clause is set to true. We want to show that the original instance will have a solution, too.

Suppose in the original instance clause $c_i, x_{i1}, x_{i2}, x_{i3}$ are all set to false. This means that in the second clause we produce, either b_i or c_i (but not both) must be true. Otherwise it is a contradiction to exactly one literal has to be true. So either in the first or the third transformed clause, there will be at least two variables set to true (either $\bar{x}_{i1}$ and b_i or $\bar{x}_{i3}$ and c_i, but not both). This is a contradiction to our 1in3SAT condition. So in the original clause c_i, at least one of three literals has to be true. Therefore, the original instance must have a solution under 3SAT condition.

($\Leftarrow$) Secondly, suppose the original instance $c_1 \wedge c_2 \wedge \ldots \wedge c_m$ has a solution. Therefore, there must be at least one literal set to be true in each clause.

Suppose in clause c_i,

a) x_{i2} is set to true, we can set b_i and c_i to false in the second transformed clause and set $a_i = x_{i1}$ and $d_i = x_{i3}$.

b) x_{i2} is set to false and both x_{i1} and x_{i3} are set to true. We can set a_i to true, b_i to false, c_i to true and d_i to false.

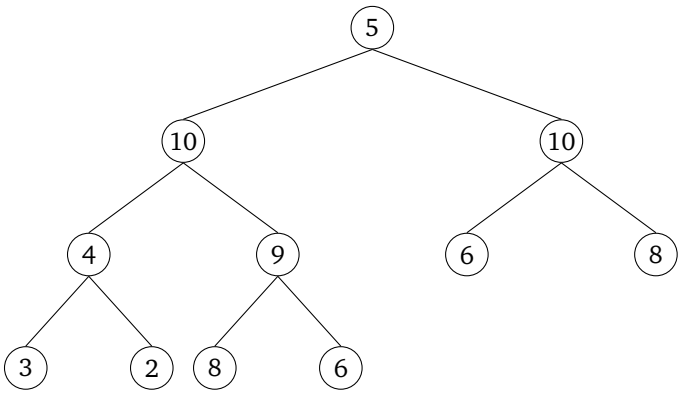
c) only x_{i1} is set to true. We can set b_i to true and a_i, c_i and d_i to false.

d) only x_{i3} is set to true. We can set d_i to true and a_i, b_i and c_i to false.

In all these cases, the three transformed clauses corresponding to c_i will have exactly one literal set to true in each clause. Therefore, the transformed instance will have a solution under 1in3SAT condition.

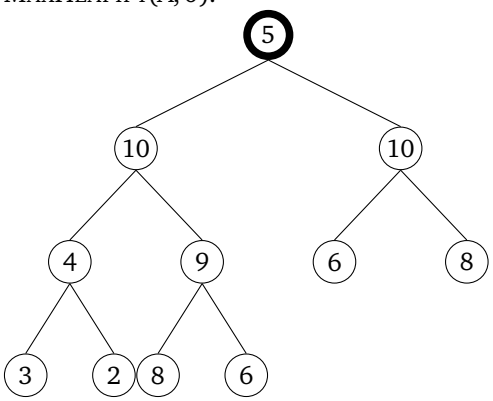
Solution to Exercise 22

(a) The implied tree looks as follows:

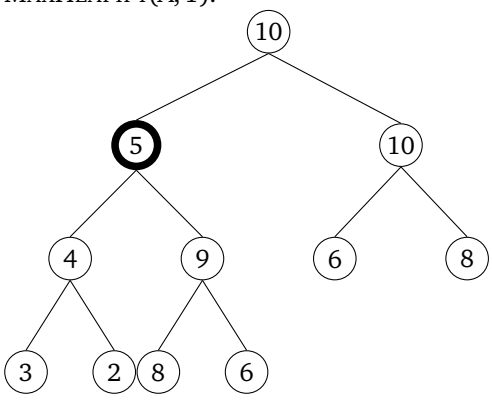


(b) No. The heap invariant is violated as $A[0] = 5$ is the parent of $A[1] = 10$, but it holds $5 < 10$!

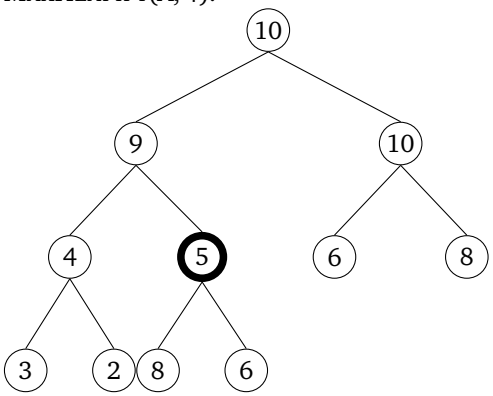
(c) MaxHeapify$(A, 0)$:



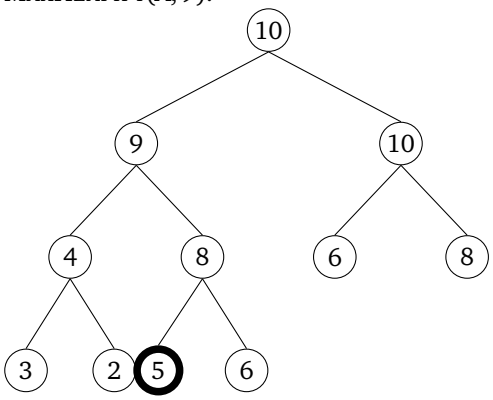
MaxHeapify$(A, 1)$:



MaxHeapify$(A, 4)$:



MaxHeapify$(A, 9)$:



Solution to Exercise 23

(a) $S = \{s\}$, $d(s) = 0$

$V \setminus S$	a	b	c	d	e	f
$d[v]$	10	5	∞	∞	∞	6
$\pi[v]$	s	s	nil	nil	nil	s

(b) $S = \{s, b\}$, $d(b) = 5$

$V \setminus S$	a	c	d	e	f
$d[v]$	8	10	∞	9	6
$\pi[v]$	b	b	nil	b	s

(c) $S = \{s, b, f\}$, $d(f) = 6$

$V \setminus S$	a	c	d	e
$d[v]$	7	10	7	9
$\pi[v]$	f	b	f	b

(d) $S = \{s, b, f, a\}$, $d(a) = 7$

$V \setminus S$	c	d	e
$d[v]$	10	7	9
$\pi[v]$	b	f	b

(e) $S = \{s, b, f, a, d\}$, $d(d) = 7$

$V \setminus S$	c	e
$d[v]$	10	8
$\pi[v]$	b	d

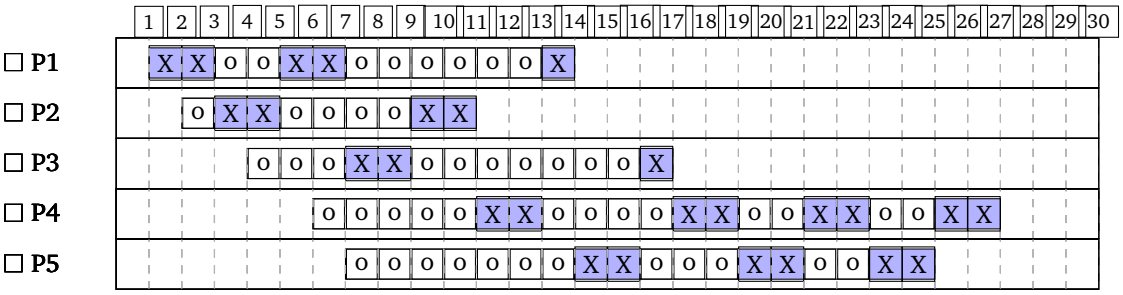
(f) $S = \{s, b, f, a, d, e\}$, $d(e) = 9$

$V \setminus S$	c
$d[v]$	10
$\pi[v]$	b

(g) $S = \{s, b, f, a, d, e, c\}$

8 Operating Systems and Computer Architecture, Security

Solution to Exercise 24:



Average response time $t_m = \frac{14+19+6+6+21}{5} = \frac{66}{5} = 13.2$

Solution to Exercise 25:

(a) 15
26–27
13 and 17

(b) Five times the number seven

(c) Reader-writer-problem

(d) Five times another sum as seven

(e) New global `pthread_mutex_t`, `..._lock` and `..._unlock` before and after modifying `num`.

Solution to Exercise 26:

(a) $\text{RTT} = 2 \times \frac{l}{c} = 2 \times \frac{10 \times 10^6\,\text{m}}{3 \times 10^8\,\text{m/s}} = \frac{1}{15}\text{s} = 0.066\,667\,\text{s}$

(b) Nothing

(c) $\text{RTT} = 2 \times \frac{l}{c_{\text{neu}}} = 2 \times \frac{10 \times 10^6\,\text{m}}{2 \times 10^8\,\text{m/s}} = 0.1\,\text{s}$

Solution to Exercise 27:

(a) $\mathscr{E} = \{$ 11111,01100,11011,10011,00011,
00100,10111,11101,01101,00000 $\}$

(b)

L_0	✓	L_1	✓	L_2	✓		
00000	✓	0000-		0-0-1			
00001	✓	000-1	✓	- -011			
00011	✓	0-001	✓				
01001	✓	00-11					
11000	✓	0-011	✓				
00111	✓	-0011	✓				
01011	✓	010-1	✓				
10011	✓	11-00					
11100	✓	-1011	✓				
11011	✓	1-011	✓				

$$\mathscr{P} = \{0000-, 00-11, 11-00, 0-0-1, --011\}$$

(c)

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(d) Prime implicant chart 1:

P	m_2 0010	m_5 0101	m_6 0110	m_7 0111	m_9 1001	m_{10} 1010	m_{11} 1011	m_{12} 1100	m_{13} 1101	m_{14} 1110	m_{15} 1111
- - 1 0	X		X			X				X	
- 1 - 1		X		X					X		X
- 1 1 -			X	X						X	X
1 - - 1					X		X		X		X
1 - 1 -						X	X			X	X
1 1 - -								X	X	X	X

List all essential prime implicants which can be identified in the first round:

$\{\ --10(c\bar{d}), -1-1(bd), 1--1(ad), 11--(ab)\ \}$

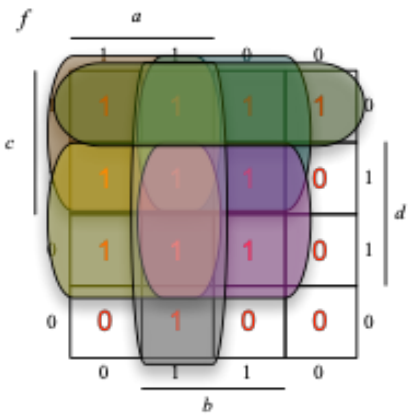
Prime implicant chart 2:

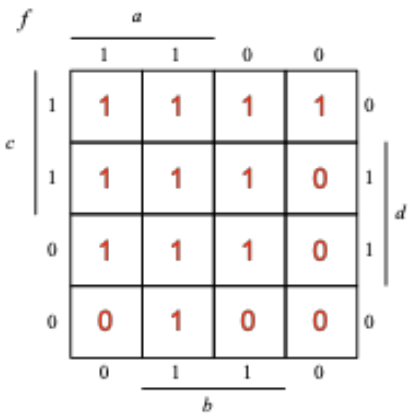
P	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$	$m\_\_$

List the minimum subset of prime implicants covering all minterms:

$\{\ --10(c\bar{d}), -1-1(bd), 1--1(ad), 11--(ab)\ \}$

(e)





Spare K-map. **Cross out invalid solution!**

Minimal sum-of-products form for f:

$\{\ c\bar{d}, bd, ad, ab\ \}$

Solution to Exercise 28

(a) ☐ `x in $t0, y in $t1`

- [ ] `x in $t0, x in $t1`
- [ ] `0 in $t0, 0 in $t1`
- [X] `y in $t0, x in $t1`

(b)
- [X] fixed instruction length
- [ ] many and complex instructions
- [X] load/store architecture
- [ ] register/memory operations

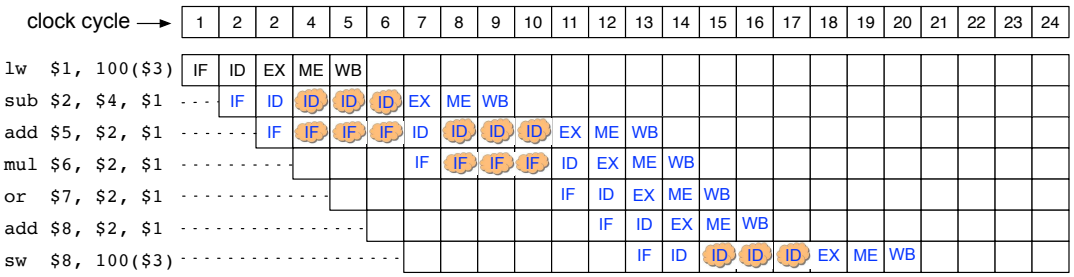
(c)
- [ ] Cache and main memory are kept consistent.
- [X] Each cache block requires a dirty bit.
- [ ] Write-back works only for set associative caches.

(d)
- [ ] 0.5 ms
- [ ] 1 ms
- [X] 2 ms
- [ ] 3 ms

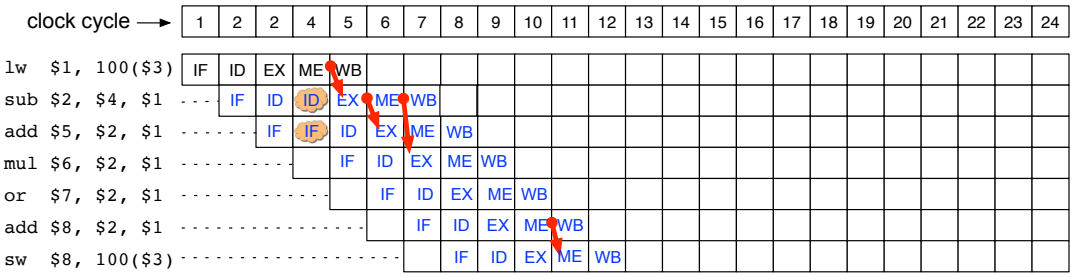
(e)
- [X] Both are multiple issue processors.
- [ ] For both the compiler assigns instructions to execution units.
- [X] Both have an ideal CPI of less than one.
- [X] Both can profit from instruction caches.

Solution to Exercise 29

(a)



(b)



(c) Speedup = clock_cycles$_{(a)}$ / clock_cycles$_{(b)}$ = $\frac{20}{12}$ = 5/3.

Solution to Exercise 30:

Consider a 2 GHz processor with a two-level cache. The L1 cache has a hit time of 0.5 ns and a miss rate of 10%. The L2 cache has a hit time of 24 clock cycles, a hit rate of 95% and a miss penalty of 200 ns. Both caches use the same block size.

(a) average_memory_access_time = average_memory_access_time$_{L1}$.
average_memory_access_time$_{L1}$ = hit_time$_{L1}$ + miss_rate$_{L1}$×miss_penalty$_{L1}$.
miss_penalty$_{L1}$ = average_memory_access_time$_{L2}$.
average_memory_access_time$_{L2}$ = hit_time$_{L2}$ + miss_rate$_{L2}$×miss_penalty$_{L2}$.
1 clock cycle takes 0.5 ns.

average_memory_access_time$_{L2}$ = 24×0.5 ns + (1-0.95)×200 ns = 22 ns.

average_memory_access_time$_{L1}$ = 0.5 ns + 0.1×22 ns = 2.7 ns.

(b) average_memory_access_time = hit_time + miss_rate×miss_penalty =
0.5 + 0.01×200 ns = 2.5 ns

The solution with the one-level cache has a lower average memory access time than the two-level cache and should thus be preferred.

Solution to Exercise 31:

(a) Salting essentially means that a username-password combination is stored as tuple

(username, pwdhash, salt)

where pwdhash = H("Password" || salt). salt is a bit string of sufficient length (typically 16 bits or more), chosen at random and individually for each user.

Salting makes attacks with precomputed hash tables, such as Rainbow Tables, infeasible.

(b) A SYN-Flooding attacker sends many TCP SYN packages to a server. For each packet, the server may then have to reserve memory for a TCP connection (e.g. to store sequence numbers, ports

numbers, IP addresses, etc.). The goal of the attacker is to exhaust the server's memory, and thus make it impossible for honest users to connect.

The standard countermeasure implemented in many modern computer systems are TCP Cookies.

(c) Kerckhoffs' Principle states that the security of a cryptosystem must depend only on the secrecy of keys, but not on the secrecy of the algorithms used. In practice, algorithms are very difficult to keep secret, and often easily recovered by reverse engineering. A prime example where hiding the encryption algorithm has failed dramatically is DVD encryption, which is completely broken today.