Natural Computing 2022

Second Exam Solution* — November 8th, 2022

Please mind the following:

- Fill in your **personal information** in the fields below.
- You may use an unmarked dictionary during the exam. **No** additional tools (like calculators, e.g.) might be used.
- Fill in all your answers **directly into this exam sheet**. You may use the backside of the individual sheets or ask for additional paper. In any case, make sure to mark **clearly** which question you are answering. Do not use pens with green or red color or pencils for writing your answers. You may give your answers in both **German and English**.
- Points annotated for the individual tasks only serve as a preliminary guideline.
- At the end of the exam hand in your **whole exam sheet**. Please make sure you do not undo the binder clip!
- To forfeit your exam ("entwerten"), please cross out clearly this **cover page** as well as **all pages** of the exam sheet. This way the exam will not be evaluated and will not be counted as an exam attempt.
- Please place your LMU-Card or student ID as well as photo identification ("Lichtbildausweis") clearly visible on the table next to you. Note that we need to check your data with the data you enter on this cover sheet.

Time Limit: 90 minutes

First Name:	
Last Name:	
Matriculation Number:	

Topic 1	max. 10 pts.	pts.
Topic 2	max. 14 pts.	pts.
Topic 3	max. 10 pts.	pts.
Topic 4	max. 16 pts.	pts.
Topic 5	max. 15 pts.	pts.
Topic 6	max. 13 pts.	pts.
Topic 7	max. 12 pts.	pts.
Total max. 90 pts.		pts.

*Corrections to the original handout are marked in red in this file.

We will reiterate definitions known from the lecture and introduce some new ones in this exam. Please mind the following notational conventions:

Notation. $\wp(X)$ denotes the power set of X. A vector $\langle x_0, ..., x_{n-1} \rangle$ with length $n \in \mathbb{N}$ can also be written as $\langle x_i \rangle_{0 \leq i \leq n-1}$ for a new iteration variable i. _ denotes unspecified function arguments ($f(\_) = 0$ is the constant function that always returns zero, e.g.). We commonly write set operators ($\in, \subseteq, \wp$ etc.) for multisets where they can be used trivially and we transparently use multisets instead of sets where necessary.

1 General Knowledge / Multiple Choice **10pts**

For each of the following questions select **one** correct answer ('1 of n'). Every correct answer is awarded one point. Multiple answers or incorrect answers will be marked with zero points.

(a) **Which of the following disciplines is a common inspiration for natural computing?** (1pt)

i Romance	ii Biology	iii History	iv Law

(b) In the game of life there exist states that have no predecessor, i.e., they can never be generated from another state via the rules of the game of life, but they can be used as initial states for an instance of the game of life. **Such a pattern is called...?** (1pt)

i Garden of Eden	ii Big Bang	iii Noah	iv Atlantis

(c) According to Stephen Wolfram we can classify 1D cellular automata into roughly four (non-exclusive) groups. **Which of the following is *not* part of Wolfram's classification system?** (1pt)

i uniformity	ii random	iii activity	iv repetition

(d) Optimization algorithm A outperforms optimization algorithm B on problem f. **The No Free Lunch theorem states that there must thus also exist a problem f' so that on problem f'...?** (1pt)

i A outperforms B	ii B outperforms A	iii Neither A nor B find the solution	iv A and B need a table reservation

(e) **The 'cooling down' process in simulated annealing takes direct inspiration from which traditional field?** (1pt)

i Accounting	ii Woodworking	iii Metallurgy	iv Alchemy

(f) We consider a fitness landscape where the global optimum lies within a region of very bad fitness values, which most optimization algorithms will avoid after a first few samples. **Said fitness landscape is thus called...?** (1pt)

i skewed	ii evil	iii distorted	iv deceptive

(g) **In the λ-calculus, the term $(\lambda x.(x\ y))$ has which free variables?** (1pt)

i $\{x\}$	ii $\{y\}$	iii $\{x, y\}$	iv $\{\lambda, x, y\}$

(h) Consider an artificial chemistry where all particles have fixed positions on a 2D grid and can only interact with their direct neighbors on that grid. **What would a soup made out of such particles be called?** (1pt)

i well-stirred	ii shaken	iii topological	iv quantum

(i) **What is necessary for a neural network with a linear activation function to be able to approximate *non-linear* functions?** (1pt)

i one-dimensional inputs	ii 2 or more layers	iii 7 or more neurons	iv strong biases

(j) Two quantum bits (qubits) can assume a joint state so that when we measure only one of these qubits, we only then immediately know the other's value. **These qubits are called...?** (1pt)

i identical	ii similar	iii enhanced	iv entangled

2 Game of Life **14pts**

The following definition was given in the lecture.

Definition 1 (Conway's game of life)**.** Let $G = (V, E)$ be a graph with vertices V and (undirected) edges $E \subseteq V \times V$. We define *neighborhood* $: V \to \wp(V)$ via *neighborhood*$(v) = \{w \mid (v, w) \in E\}$ as a topology for G. A state $x \in \mathcal{X}$ is a mapping of vertices to the labels $\{dead, alive\}$, i.e., the state space $\mathcal{X}$ is given via $\mathcal{X} = (V \to \{dead, alive\})$. Let x_t be a state that exists at time step $t \in \mathbb{N}$. We define $|v|_{x_t} = |\{w \mid w \in neighborhood(v) \wedge x_t(w) = alive\}|$. In the game of life, the evolution of a state x_t to its subsequent state x_{t+1} is given deterministically via

$$x_{t+1}(v) = \begin{cases} dead & \text{if } |v|_{x_t} \leq 1, \\ x_t(v) & \text{if } |v|_{x_t} = 2, \\ alive & \text{if } |v|_{x_t} = 3, \\ dead & \text{if } 4 \leq |v|_{x_t}, \end{cases}$$

for all $v \in V$. A tuple (G, x_S) is called an instance of the game of life for initial state $x_S \in \mathcal{X}$.

We play game of life on an $n \times m$ wrap-around grid of vertices V so that for each $v_{i,j} \in V$ with $0 < i < n$ and $0 < j < m$ the call *neighborhood*$(v_{i,j})$ returns all 8 neighboring vertices

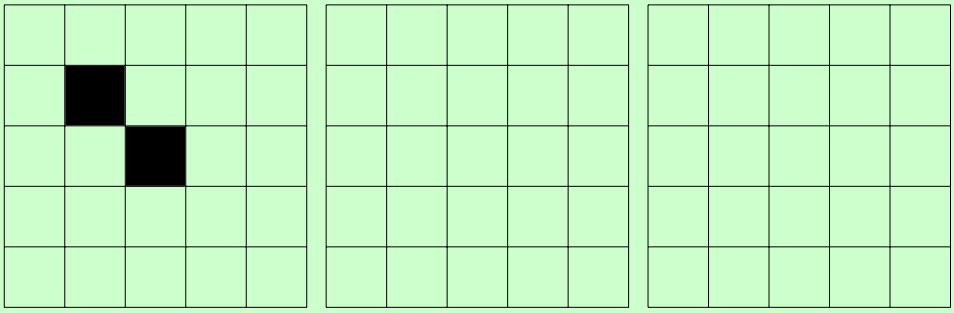
- $v_{(i-1 \mod n),\,(j-1 \mod m)}$,
- $v_{(i-1 \mod n),\,j}$,
- $v_{(i-1 \mod n),\,(j+1 \mod m)}$,
- $v_{i,\,(j-1 \mod m)}$,
- $v_{i,\,(j+1 \mod m)}$,
- $v_{(i+1 \mod n),\,(j-1 \mod m)}$,
- $v_{(i+1 \mod n),\,j}$, and
- $v_{(i+1 \mod n),\,(j+1 \mod m)}$.

We now introduce a new formal definition:

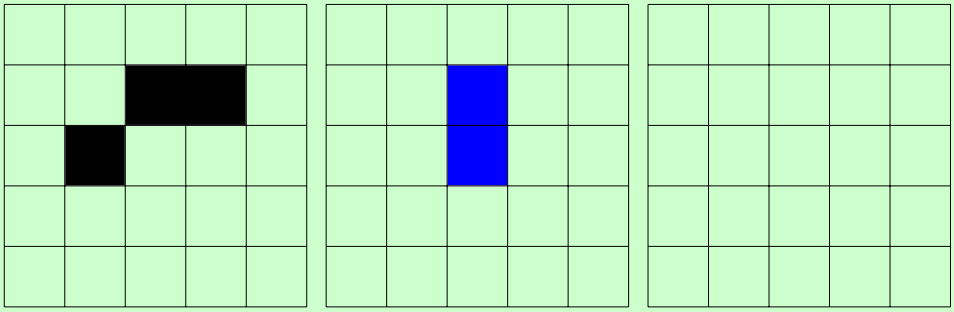
Definition 2 (exam spaceship)**.** Let (G, x_S) be an instance of the game of life with $G = (V, E)$. Let $x_{S+1}, x_{S+2}, \ldots$ be the subsequent states of that instance. (G, x_S) is called an *exam spaceship* if there exists a time frame $\Delta t \in \mathbb{N} \setminus \{0\}$ and there exist displacement values $\Delta x, \Delta y \in \mathbb{Z}$ so that for all $v_{i,j} \in V$ it holds that

$$x_{S+\Delta t}\left(v_{(i+\Delta y \mod n),\,(j+\Delta x \mod m)}\right) = x_S(v_{i,j}).$$

(a) **Show why these two non-empty instances of the game of life are *not* exam spaceships.** You may use the empty grids to the right of the instances to draw the subsequent states of these instances. (5pts)



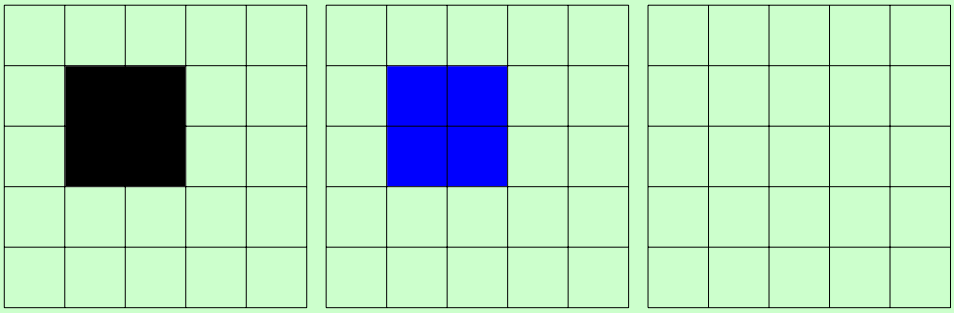
Field dies out at $t = 1$, can never reach initial state or any displacement of it.



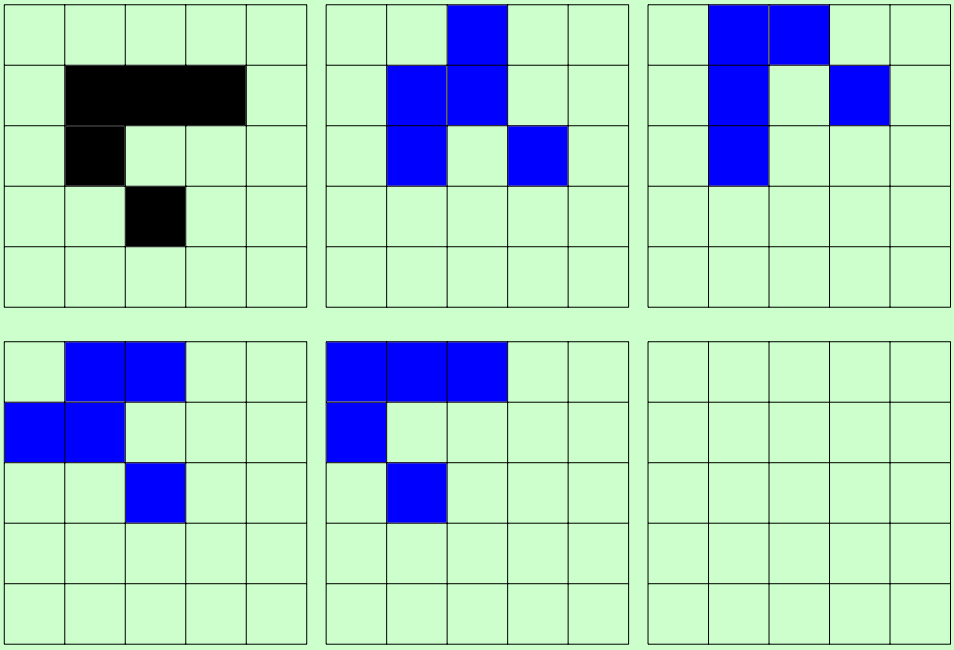
Field dies out at $t = 2$, can never reach initial state or any displacement of it.

(b) **Show why these two non-empty instances of the game of life are *indeed* exam spaceships. Also give the time frame Δt and the respective displacement values $\Delta x, \Delta y$ for which they fulfill Definition 2.** You may use the empty grids to the right and bottom of the instances to draw the subsequent states of these instances. (7pts)

Solution:



Exam spaceship for $\Delta t = 1$ with $\Delta x = 0, \Delta y = 0$



Exam spaceship for $\Delta t = 4$ with $\Delta x = -1, \Delta y = -1$

(c) The definition of an exam spaceship does not match the common definition of a spaceship. Commonly, a spaceship does need to travel across the grid and cannot remain in one place. **Give a constraint on Δt, Δx, and/or Δy from Definition 2 to express this property.** (2pts)

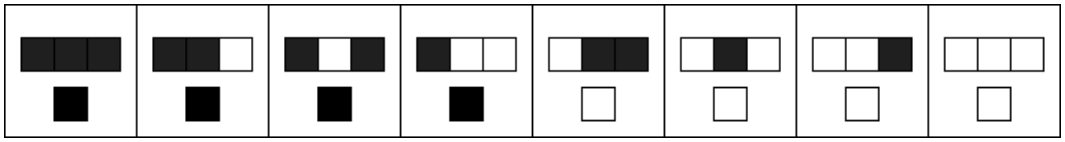
$\neg(\Delta x = 0 \ \wedge \ \Delta y = 0)$

3 Cellular Automata **10pts**

An *exam cellular automaton* is defined by a function $f : \{dead, alive\}^3 \to \{dead, alive\}$ on an initial state $x \in \{dead, alive\}^n$ for some fixed $n \in \mathbb{N}$. Let $x(i)$ denote the ith value of x, i.e., $x = (x(0), ..., x(i), ..., x(n-1))$. Also let $x(-1) = x(n-1)$ and $x(n) = x(0)$. Given a state $x_t \in \{dead, alive\}^n$ at time step $t \in \mathbb{N}$ the next state $x_{t+1} \in \{dead, alive\}^n$ is given via

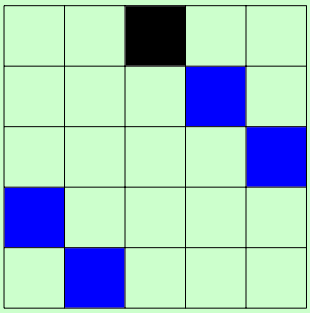
$$x_{t+1}(i) = f\big(x_t(i-1), x_t(i), x_t(i+1)\big).$$

(a) A cellular automaton's function f is often given as the template below, which shows the inputs to f on the top row and the respective output on the bottom row. The template shown below is called *rulestring 240*. **Explain why the rulestring number (240, e.g.) is sufficient to define the function f of an exam cellular automaton.** (2pts)

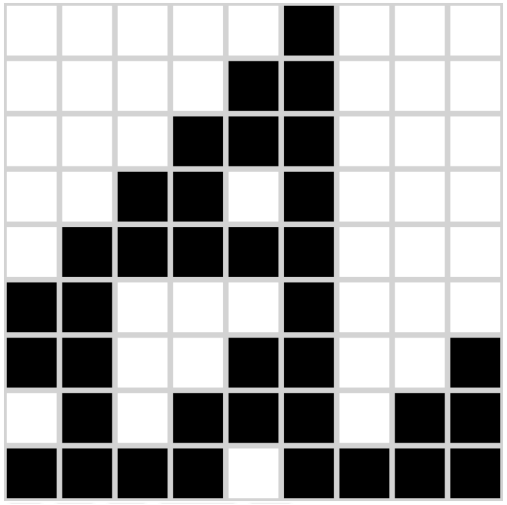


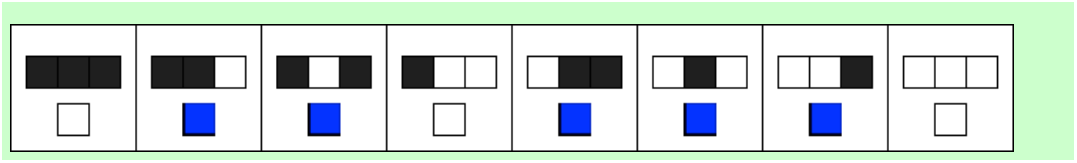
The binary representation of the rulestring number gives f's output for all $2^3 = 8$ possible inputs in order.

(b) **With the function f given in task (a), evolve this initial configuration by hand for four generations. Use one row per generation.** (4pts)



(c) Observe the evolution of a new cellular automaton below, showing one generation per row. **Give a full definition of this automaton's function f.** You may use the empty rule template printed below. (4pts)





4 Optimization 16pts

We give a shortened version of the definition for optimization processes from the lecture and then introduce a new optimization algorithm called *exam search*.

Definition 3 (optimization (shortened))**.** Let $\mathcal{X}$ be an arbitrary set called state space. Let $\mathcal{T}$ be an arbitrary set called target space and $\leq$ be a total order on $\mathcal{T}$. A total function $\tau : \mathcal{X} \to \mathcal{T}$ is called target function. Optimization or minimization is the procedure of searching for a $x \in \mathcal{X}$ so that $\tau(x)$ is optimal or minimal.
An optimization run of length $g+1$ is a sequence of states $\langle x_t \rangle_{0 \leq t \leq g}$ with $x_t \in \mathcal{X}$ for all t. Let e be a possibly randomized or non-deterministic function so that the optimization run is produced by calling e repeatedly, i.e., $x_{t+1} = e(\langle x_u \rangle_{0 \leq u \leq t}, \tau)$, where x_0 is given externally (i.e., $x_0 =_{def} 42$) or chosen randomly (i.e., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Algorithm 1 (exam search)**.** Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. We assume $\mathcal{X} = \mathbb{R}$ and $\mathcal{T} = \mathbb{R}$. The process looks fixed distances to the left and right and remembers the best of these values: Let $\langle b_u \rangle_{0 \leq u \leq t}$ with $b_u \in \mathcal{X}$ for all u be a sequence of current best found solutions so far in the optimization process, i.e., let $b_0 = x_0$ and let

$$b_{u+1} = \underset{a \in \{b_u,\, x_u-2,\, x_u-1,\, x_u,\, x_u+1,\, x_u+2\}}{\arg\min} \tau(a)$$

for $0 \leq u \leq t$.
Note that the subscript reads: $a \in \{b_u,\ x_u - 2,\ x_u - 1,\ x_u,\ x_u + 1,\ x_u + 2\}$, i.e. we mostly subtract and add numbers to x_u.
The process $\mathcal{D}$ continues via exam search iff e is given via

$$e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = 0.5 \cdot (b_{t+1} + x_t).$$

(a) Let target function

$$\tau(x) = \begin{cases} 2x & \text{if } x \geq 0, \\ -x & \text{otherwise.} \end{cases}$$

Let initial state $x_0 = 2.5$. **Apply exam search to this setting for 4 time steps, i.e., compute $x_1, ..., x_4$ by computing $b_1, ..., b_4$. Does exam search reach the global optimum of τ within these time steps?** (9pts)

step 1:
$b_1 = \arg\min_{a \in \{2.5,\ 0.5,\ 1.5,\ 2.5,\ 3.5,\ 4.5\}} \tau(a) = 0.5$
$x_1 = 0.5 \cdot (0.5 + 2.5) = 1.5$

step 2:
$b_2 = \arg\min_{a \in \{0.5,\ -0.5,\ 0.5,\ 1.5,\ 2.5,\ 3.5\}} \tau(a) = -0.5$
$x_2 = 0.5 \cdot (-0.5 + 1.5) = 0.5$

step 3:
$b_3 = \arg\min_{a \in \{-0.5,\ -1.5,\ -0.5,\ 0.5,\ 1.5,\ 2.5\}} \tau(a) = -0.5$
$x_3 = 0.5 \cdot (-0.5 + 0.5) = 0$

step 4:
$b_4 = \arg\min_{a \in \{-0.5,\ -2,\ -1,\ 0,\ 1,\ 2\}} \tau(a) = 0$
$x_4 = 0.5 \cdot (0 + 0) = 0$

Exam search reaches the global optimum at $x_3 = 0$ with $\tau(0) = 0$.

(b) **Give a new target function $\tau' : \mathbb{R} \to \mathbb{R}$ for which exam search *never* reaches the global optimum of τ' when starting from $x_0 = 0$. Briefly explain why.** (3pts)

$$\tau'(x) = \begin{cases} 0 & \text{if } x = 0.42, \\ 1 & \text{otherwise.} \end{cases}$$

Exam search remains at $x_t = 0$ as it only looks for integer values for x and thus never discovers the optimum at $x = 0.42$.

(c) **State if exam search is elitist. Briefly explain why.** (2pts)

Yes. Once x_u is the best value in the computation of b_{u+1}, we remain at x_u, i.e., the current best (or new best) value, meaning the solution always stays the same or improves.

(d) **State if exam search is greedy. Briefly explain why.** (2pts)

No. Even though the algorithm discovers a superior solution when computing b_{t+1}, we only adopt it partially for x_{t+1} and and instead remain at a possibly worse solution x_{t+1}.

5 Lambda Calculus 15pts

Church encoding is a standard encoding for various common data types to λ-expressions.

Definition 4 (Church truth values)**.** A truth value $v \in \{\textsc{True}, \textsc{False}\}$ can be used as a λ-term by defining

$$\textsc{True} = (\lambda a.(\lambda b.a))$$
$$\text{and } \textsc{False} = (\lambda a.(\lambda b.b)).$$

Based on Definition 4 we can further define standard functions of Boolean logic like

$$\textsc{And} = (\lambda x.(\lambda y.((x\ y)\ x)))$$
$$\text{or } \textsc{Or} = (\lambda x.(\lambda y.((x\ x)\ y))).$$

(a) Let $f = (\lambda x.((x\ \textsc{False})\ \textsc{True}))$. **Reduce the λ-expression $(f\ \textsc{True})$ step by step as far as possible.** Imagine the result for $(f\ \textsc{False})$; you do not need to show it. **Which standard Boolean function does f encode?** (7pts)
Hint: Try to substitute definitions as late as possible in order to keep your expressions short. Start by substituting the definition for f only.

$$\begin{aligned}
&(f\ \textsc{True}) \\
&= ((\lambda x.((x\ \textsc{False})\ \textsc{True}))\ \textsc{True}) \\
&\rightarrowtail_\beta ((\textsc{True}\ \textsc{False})\ \textsc{True})) \\
&= (((\lambda a.(\lambda b.a))\ \textsc{False})\ \textsc{True}) \\
&\rightarrowtail_\beta ((\lambda b.\textsc{False})\ \textsc{True}) \\
&\rightarrowtail_\beta \textsc{False}
\end{aligned}$$

f is the Boolean NOT function.

(b) **Reduce the λ-expression** ((And True) False) **step by step as far as possible.** Do not use your knowledge about the result in Boolean algebra to shorten the path for λ-reduction! You may, of course, use said knowledge to check your result. (8pts)
Hint: Try to substitute definitions as late as possible in order to keep your expressions short. Start by substituting the definition for And only.

$$
\begin{aligned}
& ((\text{And } \text{True}) \text{ False}) \\
= & (((\lambda x.(\lambda y.((x\ y)\ x))) \text{ True}) \text{ False}) \\
\rightarrowtail_\beta & ((\lambda y.((\text{True}\ y) \text{ True})) \text{ False}) \\
\rightarrowtail_\beta & ((\text{True False}) \text{ True}) \\
= & (((\lambda a.(\lambda b.a)) \text{ False}) \text{ True}) \\
\rightarrowtail_\beta & ((\lambda b.\text{False}) \text{ True}) \\
\rightarrowtail_\beta & \text{False}
\end{aligned}
$$

6 Artificial Chemistries / Soups **13pts**

We used the following definition in the lecture:

> **Definition 5** (artificial chemistry, soup)**.** Let $\mathcal{X}$ be a state space. Let $\mathcal{R}$ be a set of reaction rules $R \in \mathcal{R}$ where each R is a function $R : \wp(\mathcal{X}) \to \wp(\mathcal{X})$. Let $A : \wp(\mathcal{X}) \to \wp(\mathcal{X}) \times \mathcal{R}$ be a possibly randomized or non-deterministic function that returns a (multi-) set of reactants and a reaction rule to perform so that the reactants are suitable input to the reaction rule; the input to A is a (multi-)set of elements from the state space, i.e., a population. The tuple $(\mathcal{X}, \mathcal{R}, A)$ defines an artificial chemistry. A population $X \subseteq \mathcal{X}$ of an artificial chemistry is also called soup; each element $x \in X$ is also called particle. The evolution of a soup for g generations is a sequence $\langle X_0, .., X_g \rangle$ so that X_0 is given as the initial soup and
>
> $$X_{t+1} = (X_t \setminus X') \cup R(X')$$
>
> where $(X', R) = A(X_t)$. Reaction rules are usually written in the form $R = \sum_i x_i \rightharpoonup \sum_j x_j$ for $x_i, x_j \in \mathcal{X}, i, j \in \mathbb{N}$.

For the following tasks we define an artificial chemistry $\mathcal{B} = (\mathcal{X}, \mathcal{R}, A)$ where $\mathcal{X} = \{\textsc{Egg}, \textsc{Caterpillar}, \textsc{Cocoon}, \textsc{Butterfly}, \textsc{Tree}, \textsc{Squirrel}\}$ and $\mathcal{R} = \{R_1, .., R_6\}$ where

$$R_1 = \textsc{Egg} \rightharpoonup \textsc{Caterpillar},$$
$$R_2 = \textsc{Caterpillar} + \textsc{Tree} \rightharpoonup \textsc{Cocoon} + \textsc{Tree},$$
$$R_3 = \textsc{Cocoon} \rightharpoonup \textsc{Butterfly},$$
$$R_4 = \textsc{Butterfly} + \textsc{Butterfly} \rightharpoonup \textsc{Egg} + \textsc{Egg} + \textsc{Egg} + \textsc{Egg},$$
$$R_5 = \textsc{Squirrel} + \textsc{Egg} \rightharpoonup \textsc{Squirrel},$$
$$R_6 = \textsc{Squirrel} + \textsc{Caterpillar} \rightharpoonup \textsc{Squirrel}.$$

For the function A we assume that it first picks a uniform random rule $R \in \mathcal{R}$ and then checks if all particles necessary to apply R exist within the current soup X_t at time t. If such particles exist, A picks a random set of suitable particles X' to match the left hand side of rule R and then returns (X', R). Whenever suitable particles for R do not exist within X_t, A again picks a uniform random rule $R \in \mathcal{R}$ and tries to use that.

(a) **Give a possible evolution of the soup**

$$X_0 = \{\textsc{Caterpillar}, \textsc{Butterfly}, \textsc{Tree}\}$$

for as many time steps as necessary until an Egg **appears.** (Hint: Instead of random choices, you can decide which rules to apply to make the process quicker. Use the table below.) (3pts)

time step	chosen rule	current soup
0	none	$\{\textsc{Caterpillar}, \textsc{Butterfly}, \textsc{Tree}\}$
1	R_2	$\{\textsc{Tree}, \textsc{Butterfly}, \textsc{Cocoon}\}$
2	R_3	$\{\textsc{Tree}, \textsc{Butterfly}, \textsc{Butterfly}\}$
3	R_4	$\{\textsc{Tree}, \textsc{Egg}, \textsc{Egg}, \textsc{Egg}, \textsc{Egg}\}$
4		
5		

(b) **Give initial populations $X_0^{stay}, X_0^{grow}, X_0^{shrink}$ so that when we evolve these populations separately, i.e., we compute $X_t^{stay}, X_t^{grow}, X_t^{shrink}$, respectively, for increasingly large values of t by applying the artificial chemistry $\mathcal{B}$, it holds that $|X_t^{stay}|$ remains constant for all t, $|X_t^{grow}|$ eventually increases with sufficiently large t, and $|X_t^{shrink}|$ eventually shrinks with sufficiently large t.** (3pts)

$$X_0^{stay} = \emptyset$$
$$X_0^{grow} = \{\textsc{Butterfly}, \textsc{Butterfly}\}$$
$$X_0^{shrink} = \{\textsc{Squirrel}, \textsc{Egg}\}$$

(c) For this task, consider a soup of artificial chemistry $\mathcal{B}$ with the initial population

$$X_0 = \{\text{Egg}, \text{Squirrel}, \text{Tree}\}.$$

What are the chances that anywhere in the future evolution of X_0 a Butterfly appears? Show your reasoning. You can assume that all executions of A eventually terminate. You can draw all possible paths for the evolution of X_0 to derive the chances. (5pts)
Hint: Review exactly which behavior A uses.

$\{\text{Egg}, \text{Squirrel}, \text{Tree}\}$
$\quad | \quad [R_1],\ R_5 \qquad 1/2$ chance
$\{\text{Caterpillar}, \text{Squirrel}, \text{Tree}\}$
$\quad | \quad [R_2],\ R_6 \qquad 1/2$ chance
$\{\text{Cocoon}, \text{Squirrel}, \text{Tree}\}$
$\quad | \quad [R_3] \qquad 1/1$ chance
$\{\text{Butterfly}, \text{Squirrel}, \text{Tree}\}$

Since this is the only path, the overall chances of a Butterfly appearing are $1/2 \cdot 1/2 \cdot 1 = 1/4$.

(d) For this task, consider another soup of artificial chemistry $\mathcal{B}$ with the initial population

$$X_0 = \{\text{Egg}, \text{Squirrel}, \text{Squirrel}, \text{Tree}\},$$

i.e., this time starting with two Squirrels instead of only one. **Does this addition of another Squirrel affect the chances of a Butterfly appearing, which you computed for task (c)? Show your reasoning.** (2pts)
Hint: Review exactly which behavior A uses.

No, the chances are not affected because A always chooses the *rule first*, which does not depend on the amount of particles available. Since we do not change the number of applicable rules, we also do not change the chance of a Butterfly appearing.

7 Evolutionary Computing 12pts

If necessary, you can recollect the definition for evolutionary algorithms as used in the lecture on this page. (Hint: Try the tasks first before reading this lengthy definition again.)

Algorithm 2 (typical evolutionary algorithm)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \phi, E, \langle X_u \rangle_{0 \leq u \leq t})$ be a population-based optimization process.

- Let $\sigma_N^{survivors}, \sigma_N^{parents} : \wp(\mathcal{X}) \to \wp(\mathcal{X})$ be (possibly randomized or non-deterministic) selection functions that return $N \in \mathbb{N}$ individuals, i.e., $|\sigma_N(X)| = N$ and $\sigma_N(X) \subseteq X$ for all X. They may possibly depend on $\mathcal{E}$ (most commonly ϕ). Let $\rho_N^{mutants} : \wp(\mathcal{X}) \to \wp(\mathcal{X})$ be a special selection function that, given a population X, returns a random subset $X' \subseteq X$ so that $|X'| = N$. Note that $\rho_{|X|}(X) = X$ for all X.

- Let $mutate : \mathcal{X} \to \mathcal{X}$ be a (possibly randomized or non-deterministic) single-individual mutation function. We write $mutate[\_] : \wp(\mathcal{X}) \to \wp(\mathcal{X})$ for the per-individual application of $mutate$, i.e., $mutate[X] = \{mutate(x) \ : \ x \in X\}$.

- Let $recombine : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ be a (possibly randomized or non-deterministic) two-parent recombination function. We write $recombine[\_] : \wp(\mathcal{X}) \to \wp(\mathcal{X})$ for the random-pairing application of $recombine$, i.e., $recombine[X] = \{recombine(x_1, x_2)\} \cup recombine[X \setminus \{x_1, x_2\}]$ with $x_1, x_2 \sim X$ and $recombine[\{x\}] = \emptyset$ for all x as well as $recombine[\emptyset] = \emptyset$.

- Let $migrate : \emptyset \to \mathcal{X}$ be a (possibly randomized or non-deterministic) individual creation function. We write $migrate_N : \emptyset \to \wp(\mathcal{X})$ with $N \in \mathbb{N}$ for the iterated application of $migrate$, i.e., $migrate_N() = \{migrate()\} \cup migrate_{N-1}()$ with $migrate_0 = \emptyset$.

The process $\mathcal{E}$ continues via an evolutionary algorithm if E has the form

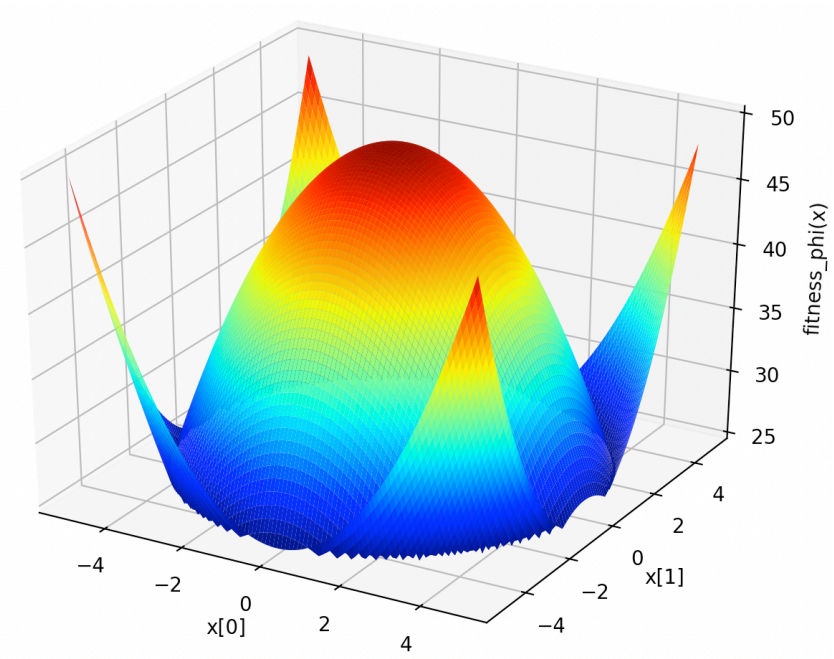
$$\begin{aligned} E(\langle X_u \rangle_{0 \leq u \leq t}, \phi) = X_{t+1} = \sigma_{|X_t|}^{survivors}\big(X_t \\ \cup\ & mutate[\rho_M^{mutants}(X_t)] \\ \cup\ & recombine[\sigma_{2C}^{parents}(X_t)] \\ \cup\ & migrate_H()\big) \end{aligned}$$

where $M \in \mathbb{M}$ is the number of mutants produced per generation, $C \in \mathbb{N}$ is the number of children produced per generation, and $H \in \mathbb{N}$ is the number of migrants (sometimes called hypermutants) generated per generation. Especially M is often expressed as a rate for random choice within the population, i.e., $M = \lceil m \cdot |X_t| \rceil$ where $m \in \mathbb{R}$ is called mutation rate.

We now consider an evolutionary algorithm that searches through individuals within $\mathcal{X} = Q^2$ where $Q = [-5; +5] \subset \mathbb{R}$. It does so by minimizing a fitness function $\phi : Q^2 \to \mathbb{R}$, which is given via the following Python function:

```
def fitness_phi(x):
    return max(x[0]**2 + x[1]**2, 50 - x[0]**2 - x[1]**2)
```

The resulting solution landscape can be seen as shown below:



(a) This fitness function features multiple global optima. **Shortly describe a technique that should help us find multiple different global optima within one single run of the evolutionary algorithm.** (2pts)

We can measure the *diversity* single individuals contribute to the population and prefer individuals with more diversity during selection. This should help to spread out the population and possibly discover multiple optima.

(b) Consider the following two recombination functions given in Python. Note that the Python function `random.random()` returns a random floating point number between `0.0` (inclusive) and `1.0` (exclusive), i.e., [0.0, 1.0).

```
import random

def recombine_avg(parent_a, parent_b):
    child = [0.0, 0.0]
    child[0] = (parent_a[0] + parent_b[0]) / 2.0
    child[1] = (parent_a[1] + parent_b[1]) / 2.0
    return child

def recombine_xover(parent_a, parent_b):
    if random.random() < 0.5:
        return [parent_a[0], parent_b[1]]
    else:
        return [parent_b[0], parent_a[1]]
```

Which of the two recombination functions `recombine_avg` and `recombine_xover` is better suited for the optimization problem given by `fitness_phi`? Explain why. (3pts)

`recombine_avg` has a tendency generate new individuals near the center of the population. This makes it nearly useless during the start of evolution, as individuals will be spread out throughout the whole search space and the center region of the solution landscape has very bad fitness. Thus, `recombine_xover` is better suited.

(c) We define a *nice* mutation function to have the following qualities:

- It does not return solution candidates that lie out of the search space's boundaries.
- Its effects are random and not affected by fitness.
- Its effects work the same way in all the individual's dimensions.
- Its effects *completely* change the original individual *only very rarely at most*.
- There is *no point* within the search space that can *never* be reached via mutation.

Give a *nice* mutation function for the above evolutionary algorithm. (5pts)

```
import random

def mutate(parent):
    index = round(random.random() * 2)
    parent[index] = parent[index] + 0.5 * random.random() - 1
    if parent[index] < -5:
        parent[index] = -5
    if parent[index] > 5:
        parent[index] = 5
    return parent
```

(d) Consider that we now construct an evolutionary algorithm with local search, i.e., we augment the evolutionary algorithm by using a greedy gradient-based search algorithm on each single individual before the fitness evaluation. **Given the fitness function `fitness_phi` (and thus the solution landscape above), what should we expect to happen when running this augmented evolutionary algorithm? Is this setup reasonable in order to optimize for the given fitness function `fitness_phi`?** (2pts)

Greedy gradient-based search on `fitness_phi` should immediately converge towards a global minimum. The setup thus yields results very quickly, but it does not use the evolutionary algorithm at all, which makes the overall setup unreasonable. We should instead use the greedy gradient-based search only.