Computational Intelligence 2023/24

Mock Exam – February 8th, 2024

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") 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. 22 pts.	pts.
Topic 3	max. 28 pts.	pts.
Topic 4	max. 10 pts.	pts.
Topic 5	max. 20 pts.	pts.
Total max. 90 pts.		pts.

1 General Knowledge / Single 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) The *imitation game* is meant to test whether an artificial intelligence can communicate like a human person. Today it is mostly referred to as ...

i Chomsky Challenge	ii Erasmus Exam	iii Turing Test	iv Sutton Stint

(b) Complex knowledge-based systems with lots of hand-coded rules have been prominent in artificial intelligence research in the 1970s. They are called ...

i genius systems	ii expert systems	iii intelligence systems	iv agent systems

(c) A programming language that is especially designed to program such large logic-based systems is called ...

i Dialog	ii Prolog	iii Interlog	iv Epilog

(d) Given a fixed architecture, a neural network's behavior given some input x is sufficiently defined by its ...

i derivations	ii colors	iii weights	iv angles

(e) Sufficiently large neural networks can approximate ...

i only constant functions	ii only positive functions	iii only differentiable functions	iv any arbitrary function

(f) Which of these is not a standard component for an evolutionary algorithm?

i selection	ii imagination	iii mutation	iv recombination

(g) In 2016, a team of researchers first managed to build an artificial intelligence to beat famous player Lee Sedol in a game of Go. The software was called ...

i AlphaGo	ii BetaGo	iii PhiGo	iv OmegaGo

(h) When building multi-agent systems (such as swarms), it is often difficult to translate goals for the system as a whole to goals for single agents. This is called ...

i the high-low problem	ii the front-back problem	iii the left-right problem	iv the micro-macro problem

(i) In *evolutionary game theory*, mixed strategies are usually interpreted as ...

i distributions within a population	ii ancestors in a genealogical tree	iii target functions in selection	iv patterns in the genome

(j) Some well-known people in the artificial intelligence community entertain the idea that at some point in time technological advancement will outpace the human ability to learn. This fictitious future point in time is commonly referred to as ...

i singularity	ii nexus	iii world between worlds	iv matrix

2 Agents and Goals **22pts**

Consider a power grid consisting of 100 power nodes located on a 10×10 grid. Per default, each power node is assigned the state `working`. If a cell detects a problem, it sounds an alarm and switches to the state `alarming`. After a problem has not been treated for an indeterminate amount of time, the power node may switch to the state `defect` and thereby stop functioning. The state of the power node at position (x, y) can be checked by calling a function $\texttt{state\_of} : \mathcal{C}^2 \rightarrow \mathcal{M}$ where $\mathcal{C} = [0; 9] \subset \mathbb{N}$ is the space of grid positions and $\mathcal{M} = \{\texttt{working}, \texttt{alarming}, \texttt{defect}\}$ is the space of power node states, i.e., states that a single power node can be in.

To keep the grid in good shape, we deploy a small drone called *RepairBot*. It can be present at exactly one power node at the same time. It can observe its current position coordinates $(x, y) \in \mathcal{C}^2$ as well as the nearest `alarming` power node's coordinates $(x^\dagger, y^\dagger) \in \mathcal{C}^2$. With one action (which also means within one time step), the *RepairBot* can move to any of the 8 neighboring power nodes (horizontally, vertically, and diagonally), do nothing, or repair the node it is currently at; the *RepairBot* thus has an action space $\mathcal{A} = \{\texttt{move\_northwest}, \texttt{move\_north}, ..., \texttt{move\_southeast}, \texttt{do\_nothing}, \texttt{repair}\}$ with $|\mathcal{A}| = 10$. The move actions place the *RepairBot* at the corresponding neighboring power node in the next time step, `do_nothing` (obviously) does not change the robot's position, and `repair` puts the power node where the robot is at in the power node state `working`, regardless of what its previous state was. If the *RepairBot* attempts to move in a direction where there is no neighboring power node (which happens at the edges of the grid), nothing happens.

Assume that the complete state of the whole system (consisting of the power grid and the *RepairBot*) is given by one instance of a function fulfilling the definition of the `state_of` function as given above as well as the position of the *RepairBot*. The whole system then generates a sequence of system states $\langle s_t \rangle_{0 \leq t \leq T}$ for some fixed maximum episode length $T \in \mathbb{N}$ and every system state $s_t \in \mathcal{S}$ with $\mathcal{S} = (\mathcal{C}^2 \rightarrow \mathcal{M}) \times \mathcal{C}^2$ in accordance with all above specifications.

(i) As stated above, the *RepairBot* can observe $(x^\dagger, y^\dagger)$, which is the nearest `alarming` power node. Why is this coordinate data not needed as a separate part of the system state definition? (2pts)

(ii) Give a goal predicate $\gamma : \langle \mathcal{S} \rangle \to \mathbb{B}$ so that $\gamma(\langle \texttt{state\_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T})$ holds iff no power node has ever been `defect` and any `alarming` power node is `working` again (at least once) within at most 5 time steps. (12pts)

(iii) Now consider that a policy for *RepairBot* is generated via standard optimization and/or reinforcement learning techniques to maximize the undiscounted accumulated reward given by the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ defined via

$$R\big((\texttt{state\_of}_t, (x_t, y_t)), a_t, (\texttt{state\_of}_{t+1}, (x_{t+1}, y_{t+1}))\big) = \begin{cases} +10 & \text{if } a_t = \texttt{repair} \text{ and } \texttt{state\_of}_t((x_t, y_t)) = \texttt{defect}, \\ +5 & \text{if } a_t = \texttt{repair} \text{ and } \texttt{state\_of}_t((x_t, y_t)) = \texttt{alarming}, \\ 0 & \text{otherwise.} \end{cases}$$

Explain why this reward function might easily lead to unwanted behavior, i.e., might facilitate the violation of goal γ as given above, and how this behavior looks like. Add to the definition of R (i.e., do not alter the given if-cases but feel free to add new ones) so that R now incentivizes the *RepairAgent* to actually fulfill γ. (8pts)

3 Fuzzy Logic and Optimization **28pts**

We still consider a power grid consisting of 100 power nodes located on a 10×10 grid. The state of the power node at position (x, y) can be checked by calling a function $\texttt{state\_of} : \mathcal{C}^2 \to \mathcal{M}$ where $\mathcal{C} = [0; 9] \subset \mathbb{N}$ and $\mathcal{M} = \{\texttt{working}, \texttt{alarming}, \texttt{defect}\}$. A *RepairBot* travels through that grid by performing actions from the action space $\mathcal{A} = \{\texttt{move\_northwest}, \texttt{move\_north}, ..., \texttt{move\_southeast}, \texttt{do\_nothing}, \texttt{repair}\}$ with $|\mathcal{A}| = 10$.

(i) We now want to evaluate how well the power nodes of the power grid are treated. We ask power grid whisperers and they tell us that the happiness of a power grid node at position $p = (x, y)$ can be computed via

$$\mathit{happy}(p) = \texttt{OR}\big(\mathit{very}(\mathit{central}(p)), \mathit{cared}(p)\big)$$

where

$$\mathit{central}((x, y)) = \frac{10 - |5 - x| - |5 - y|}{10}$$

only depends on the node's position and

$$\mathit{cared}((x, y)) = \frac{H(x, y)}{100}$$

can change over time where $H(x, y)$ is the number of times that the *RepairBot* visited this power node's position during the last 100 time steps, i.e., the number of time steps for which the *RepairBot*'s position was (x, y).
Furthermore, OR is the commonly known fuzzy logic function and *very* is given via

$$\mathit{very}(\mu) = \mu^2.$$

Assume that the *RepairBot* follows a policy that makes it stand still at node position $(4, 2)$ infinitely. Compute $\mathit{happy}((4, 2))$ and $\mathit{happy}((5, 5))$ including all intermediate results, say which of these two nodes is a global optimum (or say if both of them are), and give a brief argument why no other nodes are global optima. (11pts)

(ii) We now want to find a policy for the *RepairBot* via an evolutionary algorithm. To this end, we consider only policies which run for exactly 1000 time steps and are fully defined by a parameter $\theta \in \mathcal{A}^{1000}$, i.e., a list of actions with length 1000.

We define that a nice mutation function $\texttt{mutate} : \mathcal{A}^{1000} \to \mathcal{A}^{1000}$ for such an evolutionary algorithm has the following properties:

- It changes exactly one action within the policy. That action can be any other action afterwards.
- The mutated policy is not fully determined by the parent policy but depends on random effects in some way.
- Through iterative application we can produce any arbitrary policy.

Give a nice mutation function `mutate`. (9pts)
Hint: You may use a probabilistic function $\texttt{random} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by calling $\texttt{random}(i, j)$, which returns a random integer within $[i; j) \subset \mathbb{N}$ according to a uniform distribution.

(iii) But why think ourselves when we can simply ask a large language model? (Rhetorical question!) Assume that $\mathcal{D}$ is a dictionary of every word in the English language. $\mathcal{D}^+$ is the space of all sequences of words that contain at least one word, i.e., $x \in \mathcal{D}^+$ iff there is an $n \in \mathbb{N} \setminus \{0\}$ so that $x = \langle d_1, ..., d_n \rangle$ and for all i, $1 \leq i \leq n$, it holds that $d_i \in \mathcal{D}$, as well as a function `ask` : $\mathcal{D}^+ \to \mathcal{D}^+$ that sends an input text (prompt) to our favorite chat bot and returns the chat bot's answer. (Also assume that we can freely convert between elements from $\mathcal{D}^+$, which have the type `Prompt` in the code, and typical `string` objects as they are found in Python. Note that the operator `+` performs string concatenation and that `int`(s) converts string s to an integer).

```
def mystery_a(x1: Prompt, x2: Prompt) -> Prompt:
    x_new = "Think about all the following expressions
             at the same time: " + x1 + ". " + x2 + "."
    return x_new

def mystery_b(x: Prompt) -> int:
    question = "On a scale from 1 (least) to 10 (most),
                return only the number as your response!
                How much inner peace do you estimate
                that the following prompt brings
                the average reader: " + x + "."
    return int(ask(question))
```

Answer the following questions about an evolutionary algorithm that uses these mystery functions (cf. task (ii)). (8pts)

- As which typical function used in the definition of evolutionary algorithms can the function `mystery_a` be used? Briefly explain why.

- As which typical function used in the definition of evolutionary algorithms can the function `mystery_b` be used? Briefly explain why.

- Are there words $d \in \mathcal{D}$ that you would expect to occur in individuals from this evolutionary algorithm much more frequently than in common English writing? Why?

- Which trend for the average length of individuals in this evolutionary algorithm should we expect?

4 The π-Calculus **10pts**

(i) Evaluate the following π-processes step by step until they cannot be reduced any further. (6pts)

$$\overline{a}\langle 42\rangle.b(x).\overline{c}\langle x\rangle.0 \mid a(y).\overline{b}\langle y\rangle.0 \mid c(z).S(z)$$

$$\overline{a}.\overline{c}.\overline{b}.0 \mid (\overline{a}.A + \overline{c}.C) \mid !(a.A + b.B)$$

$$a(x).b(y).S(x,y) + (a(x).A(x) \mid \overline{a}\langle Y\rangle.0)$$

(ii) Now, thinking back one final time to the power node setting of the previous tasks: When the *RepairBot* finally goes up to an `alarming` power node, both follow a fixed protocol of interaction that is specified for the power node as follows in π-calculus:

$$\begin{aligned} \mathit{PowerNode}(\mathit{name}, \mathit{state}) = & \mathit{announce}(\mathit{bot\_capability}). \\ & \overline{\mathit{introduce}}\langle \mathit{name} \rangle. \\ & (\mathit{do\_nothing}.\mathit{PowerNode}(\mathit{name}, \mathit{state}) \\ & + \ \mathit{repair}.\mathit{PowerNode}(\mathit{name}, \mathit{bot\_capability})) \end{aligned}$$

Give a π-process *RepairBot* so that the process

$$\mathit{RepairBot} \mid \mathit{PowerNode}(\texttt{1}, \texttt{alarming}) \mid \mathit{PowerNode}(\texttt{2}, \texttt{defect})$$

always evaluates to the process

$$\mathit{RepairBot} \mid \mathit{PowerNode}(\texttt{1}, \texttt{working}) \mid \mathit{PowerNode}(\texttt{2}, \texttt{working}) \mid \mathit{Bill}(\texttt{1}) \mid \mathit{Bill}(\texttt{2})$$

where $\mathit{Bill}(\mathit{name})$ is a named (black-box) process that cannot be evaluated any further for any *name* and can be called freely within your namespace. (4pts)
Hint: It is perfectly fine to repair a `working` *PowerNode* as well.

5 Scientific Reading: The Spi-Calculus **20pts**

Read the adapted paper below[1] and answer the following questions.
Note: This excerpt is self-contained.

Abstract
We introduce the *spi-calculus*, an extension of the pi-calculus designed for the description and analysis of cryptographic protocols. We show how to use the spi calculus, particularly for studying authentication protocols. The pi-calculus (without extension) suffices for some abstract protocols; the spi calculus enables us to consider cryptographic issues in more detail. We represent protocols as processes in the spi-calculus and state their security properties in terms of coarse-grained notions of protocol equivalence. [...]

2.1 Basics

The pi-calculus is a small but extremely expressive programming language. It is an important result of the search for a calculus that could serve as a foundation for concurrent computation, in the same way in which the lambda calculus is a foundation for sequential computation. Pi-calculus programs are systems of independent, parallel processes that synchronize via message-passing handshakes on named channels. The channels a process knows about determine the communication possibilities of the process. Channels may be restricted, so that only certain processes may communicate on them. [...]

What sets the pi-calculus apart from earlier calculi is that the scope of a restriction—the program text in which a channel may be used—may change during computation. When a process sends a restricted channel as a message to a process outside the scope of the restriction, the scope is said to *extrude*, that is, it enlarges to embrace the process receiving the channel. Processes in the pi-calculus are mobile in the sense that their communication possibilities may change over time; they may learn the names of new channels via scope extrusion. Thus, a channel is a transferable capability for communication. A central technical idea of this paper is to use the restriction operator and scope extrusion from the pi-calculus as a formal model of the possession and communication of secrets, such as cryptographic keys. These features of the pi-calculus are essential in our descriptions of security protocols.

2.2 Outline of the Pi-Calculus

[...] We assume an infinite set of names, to be used for communication channels, and an infinite set of variables. We let m, n, p, q, and r range over *names*, and let z, y, and z range over *variables*. The set of *terms* is defined by the grammar:

$L, M, N ::=$	terms
n	name
(M, N)	pair
$\mathbf{0}$	zero
$suc(M)$	successor
x	variable

In the standard pi-calculus, names are the only terms. For convenience we have added constructs for pairing and numbers, (M, N), 0, and suc(M), and have also distinguished variables from names. The set of *processes* is defined by the grammar:

$P, Q, R ::=$	processes
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input
$P \mid Q$	composition
$(\nu n)\ P$	restriction
!P	replication
$[M\ is\ N]\ P$	match
$\mathbf{0}$	nil
$let\ (x, y) = M\ in\ P$	pair splitting
$case\ M\ of\ 0 : P\ suc(x) : Q$	integer case

[1] from: Abadi and Gordon. A Calculus for Cryptographic Protocols – The Spi Calculus. ACM, 1997. https://dl.acm.org/doi/pdf/10.1145/266420.266432

In $(\nu n)\ P$, the name n is bound in P. In $M(x).P$, the variable x is bound in P. In *let* $(x, y) = M$ *in* P, the variables x and y are bound in P. In *case* M *of* $0 : P$ *suc*$(x) : Q$, the variable x is bound in the second branch, Q. We write $P[M/x]$ for the outcome of replacing each free occurrence of x in process P with the term M, and identify processes up to renaming of bound variables and names. We adopt the abbreviation $\overline{M}\langle N\rangle.P$ for $\overline{M}\langle N\rangle.P.\mathbf{0}$. [. . .]

Since we added pairs and integers, we have two new process forms:

- A pair splitting process *let* $(x, y) = M$ *in* P behaves as $P[N/x][L/y]$ if term M is the pair (N, L), and otherwise it is stuck.
- An integer case process *case* M *of* $0 : P$ *suc*$(x) : Q$ behaves as P if term M is 0, as $Q[N/x]$ if M is *suc*(N), and otherwise is stuck.

We write $P \simeq Q$ to mean that the behaviors of the processes P and Q are indistinguishable. [. . .]

2.3 A first example

Our first example is extremely basic. In this example, there are two principals A and B that share a channel, c_{AB}; only A and B can send data or listen on this channel. The protocol is simply that A uses c_{AB} for sending a single message M to B. In informal notation, we may write this protocol as follows:

$$\text{Message 1} \quad A \to B : M \quad \text{on } c_{AB}$$

A first pi-calculus description of this protocol is:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle M\rangle \\ B &\triangleq c_{AB}(x).\mathbf{0} \\ \mathit{Inst}(M) &\triangleq (\nu c_{AB})\ (A(M) \mid B) \end{aligned}$$

The processes $A(M)$ and B describe the two principals, and $\mathit{Inst}(M)$ describes (one instance of) the whole protocol. The channel c_{AB} is restricted; intuitively, this achieves the effect that only A and B have access to c_{AB}.

In these definitions, $A(M)$ and $\mathit{Inst}(M)$ are processes parameterised by M. More formally, we view A and *Inst* as functions that map terms to processes, called abstractions, and treat the M's on the left of $\triangleq$ as bound parameters. Abstractions can of course be instantiated (applied); for example, the instantiation $A(0)$ yields $c_{AB}\langle 0\rangle$. The standard rules of substitution govern application, forbidding parameter captures; for example, expanding $\mathit{Inst}(c_{AB})$ would require a renaming of the bound occurrence of c_{AB} in the definition of *Inst*.
The first pi-calculus description of the protocol may seem a little futile because, according to it, B does nothing with its input. A more useful and general description says that B runs a process F with its input. We revise our definitions as follows:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle M\rangle \\ B &\triangleq c_{AB}(x).F(x) \\ \mathit{Inst}(M) &\triangleq (\nu c_{AB})\ (A(M) \mid B) \end{aligned}$$

Informally, $F(x)$ is simply the result of applying F to x. More formally, F is an abstraction, and $F(x)$ is an instantiation of the abstraction. We adopt the convention that the bound parameters of the protocol (in this case, M, c_{AB}, and x) cannot occur free in F. This protocol has two important properties:

- Authenticity (or integrity): B always applies F to the message M that A sends; an attacker cannot cause B to apply F to some other message.
- Secrecy: The message M cannot be read in transit from A to B: if F does not reveal M, then the whole protocol does not reveal M.

The secrecy property can be stated in terms of equivalences: if $F(M) \simeq F(M')$, for any M, M', then $\mathit{Inst}(M) \simeq \mathit{Inst}(M')$. This means that if $F(M)$ is indistinguishable from $F(M')$, then the protocol with message M is indistinguishable from the protocol with message M'.

[. . .]

(i) In the pi-calculus definition, as described above, the authors include some additional distinctions to the canonical grammar. Name *two* of these additions and briefly explain their intended purpose. (5pts)

(ii) The described spi-calculus is built *upon* the basics of the pi-calculus. In particular: (a) What 'feature' of the pi-calculus does the spi-calculus extension rely on? (b) What two properties are gained as such? Briefly describe their purpose. (10 pts)

(iii) The authors clarify the definitions of the spi-calculus running example from their first description in pi-calculus notation. What is the change they made? Give an argument why the protocol makes more sense in the second iteration. (5pts)

Appendix: Definitions

Notation. $\mathbb{B}$ is the set of truth values or Booleans, i.e., $\mathbb{B} = \{0, 1\}$. $\mathbb{N}$ is the set of natural numbers starting from zero, i.e., $\mathbb{N} = \{0, 1, 2, ...\}$ so that it holds that $\mathbb{B} \subset \mathbb{N} \subset \mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$. $\mathbb{P}$ denotes the space of probabilities and $\mathbb{F}$ denotes the space of fuzzy values with $\mathbb{P} = \mathbb{F} = [0; 1] \subset \mathbb{R}$ being only discerned for semantic but not mathematical reasons. $\wp(X)$ denotes the power set of X. $\mathbb{E}$ denotes the expected value. $+\!\!+$ denotes vector or sequence concatenation, i.e., given two vectors $\mathbf{x} = \langle x_1, ..., x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, ..., y_{|\mathbf{y}|} \rangle$, $\mathbf{x} +\!\!+ \mathbf{y} = \langle x_1, ..., x_{|\mathbf{x}|}, y_1, ..., y_{|\mathbf{y}|} \rangle$. 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.). For any finite set $X = \{x_0, ..., x_n\}$, $|X| = n$ denotes the number of elements in X. For infinite sets, $|X| = \infty$.

Definition 1 (agent)**.** Let $\mathcal{A}$ be a set of actions. Let $\mathcal{O}$ be a set of observations. An agent A can be given via a policy function $\pi : \mathcal{O} \to \mathcal{A}$. Given a time series of observations $\langle o_t \rangle_{t \in \mathcal{Z}}$ for some time space $\mathcal{Z}$ the agent can thus generate a time series of actions $\langle a_t \rangle_{t \in \mathcal{Z}}$ by applying $a_t = \pi(o_t)$.

Definition 2 (optimization)**.** Let $\mathcal{X}$ be an arbitrary state space. Let $\mathcal{T}$ be an arbitrary set called target space and let $\leq$ be a total order on $\mathcal{T}$. A total function $\tau : \mathcal{X} \to \mathcal{T}$ is called target function. Optimization (minimization/maximization) is the procedure of searching for an $x \in \mathcal{X}$ so that $\tau(x)$ is optimal (minimal/maximal). Unless stated otherwise, we assume minimization.
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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \to \mathcal{T}) \to \mathcal{X}$ be a possibly randomized or non-deterministic function so that the optimization run $\langle x_t \rangle_{0 \leq t \leq g}$ is produced by calling e repeatedly, i.e., $x_{t+1} = e(\langle x_u \rangle_{0 \leq u \leq t}, \tau)$ for all t, $1 \leq t \leq g$, where x_0 is given externally (e.g., $x_0 =_{def} 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Definition 3 (population-based optimization (alternate))**.** An optimization process $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_t \rangle_{0 \leq t \leq g})$ is called population-based iff $\mathcal{X}$ has the form $\mathcal{X} = \wp^*(\mathcal{Y})$ for some other state space $\mathcal{Y}$.

Algorithm 1 (basic evolutionary algorithm)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be a population-based optimization process. The process $\mathcal{E}$ continues via an evolutionary algorithm if E has the form

$$E(\langle X_u \rangle_{0 \leq u \leq t}, \tau) = X_{t+1} = \mathit{selection}\big(X_t \uplus \mathit{variation}(X_t)\big)$$

where *selection* and *variation* are possibly randomized or non-deterministic functions so that for any $X \in \wp^*(\mathcal{X})$ it holds that $|\mathit{selection}(X)| \leq |X|$ and $\big|\mathit{selection}\big(X \uplus \mathit{variation}(X)\big)\big| = |X|$.

Definition 4 (optimization (policy))**.** Let $\mathcal{X} = \Pi$ be a policy space. Let $\mathcal{D} = (\Pi, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$ be an optimization process according to Definition 2. $\mathcal{D}$ is called a policy optimization process.

Definition 5 (Markov decision process (MDP))**.** Let $\mathcal{A}$ be a set of actions. Let $\mathcal{S}$ be a set of states. Let A be an agent given via a policy function $\pi : \mathcal{S} \to \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{P}$ is the *transition probability function* often written as $P(s'|s, a) = \Pr(s_{t+1} = s' \,|\, s_t = s \wedge a_t = a)$.
A Markov decision process run for policy π is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \tau, \pi, \langle s_t \rangle_{t \in \mathcal{Z}}, \langle a_t \rangle_{t \in \mathcal{Z}})$ where

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

and where τ is to be minimized (maximized) and usually has a form similar to

$$\textit{accumulated cost (reward)}\ \tau(\pi) =_{\mathit{def}} \sum_{t \in \mathcal{Z}} R(s_t, a_t, s_{t+1})$$

$$\text{or } \textit{discounted expected cost (reward)}\ \tau(\pi) =_{\mathit{def}} \mathbb{E}\Big[\sum_{t \in \mathcal{Z}} \gamma^t \cdot R(s_t, a_t, s_{t+1})\Big]$$

where $\gamma \in [0; 1] \subset \mathbb{R}$ is called a discount factor.
A decision process generates a (possibly infinite) series of rewards $\langle r_t \rangle_{t \in \mathcal{Z}}$ with $r_t = R(s_t, a_t, s_{t+1})$.

Algorithm 2 (optimal policy)**.** Let $V^* : \mathcal{S} \to \mathcal{T}$ be the *true value function* of a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$. The optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ is given via

$$\pi^*(s_t) = \arg\max_{a \in \mathcal{A}} V^*(s')$$

where $s' \sim P(s'|s_t, a)$ is the follow-up state when executing action a in state s_t.

Definition 6 (training of a neural network (shortened))**.** Let $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ be a neural network with n weights $\overline{\mathcal{N}} = \mathbf{w} +\!\!+ \mathbf{b} \in \mathbb{R}^n$.

- Let $(\mathcal{O}, \mathcal{A}, \mathcal{T}, e, R)$ be a decision process (cf. Definition **??**) for which policy $\pi_{\overline{\mathcal{N}}} : \mathcal{O} \to \mathcal{A}$ yields (possibly randomized or non-deterministic) rewards $\langle r_t \rangle_{t \in \mathcal{Z}}$. Note that $\pi_{\overline{\mathcal{N}}}$ in some way calls $\mathcal{N}$ to produce its output, for example

  $$\pi_{\overline{\mathcal{N}}}(o) = \mathcal{N}(o)$$

  for $\mathcal{O} \subseteq \mathbb{R}^p, \mathcal{A} \subseteq \mathbb{R}^q$ or if suitable translations exist.
  If τ is of the form

  $$\tau(\overline{\mathcal{N}}) = -\mathbb{E}\Big[\sum_{t \in \mathcal{Z}} \gamma^t \cdot r_t\Big]$$

  or a similar form, the process of training $\mathcal{N}$ is called policy-based reinforcement learning.

- Let $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ be a Markov decision process (cf. Definition 5) for which we run policy $\pi_{\overline{\mathcal{N}}} : \mathcal{S} \to \mathcal{A}$. Note that $\pi_{\overline{\mathcal{N}}}$ in some way calls $\mathcal{N}$ to produce its output, for example

  $$\pi_{\overline{\mathcal{N}}}(s) = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim P(s'|s,a)} \left[\mathcal{N}(s') \right]$$

  for $\mathcal{S} \times \mathcal{A} \subseteq \mathbb{R}^p$ with $q = 1$ or if suitable translations exist.
  Let $R : \mathcal{S} \times \mathcal{A} \to \mathcal{T}$ be the expected reward of executing an action in a given state, i.e., $R(s, a) = \mathbb{E}[R(s, a, s')]$ where $s' \sim P(s'|s, a)$. Let $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ be a (possibly randomized or non-deterministic) transition function, i.e., $T(s, a) = s'$ where $s' \sim P(s'|s, a)$. Let $\gamma \in [0; 1]$ be a discount factor. Let $V_{\pi_{\overline{\mathcal{N}}}} : \mathcal{S} \to \mathbb{R}$ be the total discounted reward that policy $\pi_{\overline{\mathcal{N}}}$ generates when starting in state s, i.e.,

  $$V_{\pi_{\overline{\mathcal{N}}}}(s) = R\big(s, \pi_{\overline{\mathcal{N}}}(s)\big) + \gamma \cdot V_{\pi_{\overline{\mathcal{N}}}}\big(T(s, \pi_{\overline{\mathcal{N}}}(s))\big).$$

  Note that for $\gamma < 1$ we can abort this recursive computation once the effect of the further recursive part is sufficiently small. Note that we may also have a fixed recursion depth or that $T(s^\dagger, \_)$ might not be defined for all $s^\dagger \in \mathcal{S}$, which are then called terminal states and also cause the recursion to end.
  Let $\mathbb{S} = \{\mathbf{s}_i : i = 1, ..., N\} \subseteq S$ be a set of training states. If τ is of the form

  $$\tau(\overline{\mathcal{N}}) = -\frac{1}{N} \cdot \sum_{i=1}^{N} V_{\pi_{\overline{\mathcal{N}}}}(\mathbf{s}_i)$$

  or a similar form, the process of training $\mathcal{N}$ is called value-based reinforcement learning.

Definition 7 (multi-agent system)**.** Let $G = \{G^{[1]}, ..., G^{[N]}\}$ be a set of $|G| = N$ agents with observation spaces $\mathcal{O}^{[i]}$ and action spaces $\mathcal{A}^{[i]}$ controlled by policies $\pi^{[i]}$ for all $i = 1, ..., N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times ... \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times ... \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, ..., o^{[N]}) = (a^{[1]}, ..., a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i.

Definition 8 (evolutionary stable strategy)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game played by two players $i, -i$ with the same action space $\mathcal{A}^{[i]} = \mathcal{A}^{[-i]}$. A strategy $\pi^{[i]}$ for agent $G^{[i]}$ is an *evolutionary stable strategy* iff

- $\pi = \pi^{[i]} \oplus \pi^{[-i]}$ with $\pi^{[-i]} = \pi^{[i]}$ is a Nash equilibrium and
- for every other strategy $\pi'^{[i]} = \pi'^{[-i]} \neq \pi^{[i]}$ it holds that

$$\chi^{[i]}((\pi^{[i]} \oplus \pi'^{[-i]})(\_)) > \chi^{[i]}((\pi'^{[i]} \oplus \pi'^{[-i]})(\_)).$$

Definition 9 (π-process)**.** Let N be a set of names ($N = \{\text{“a”}, \text{“b”}, \text{“c”}, ...\}$, e.g.). $\mathbb{L}_\pi$ is the set of valid processes in the π-calculus given inductively via:

(null process) $0 \in \mathbb{L}_\pi$,

(τ prefix) if $P \in \mathbb{L}_\pi$, then $\tau.P \in \mathbb{L}_\pi$,

(receiving prefix) if $a, x \in N$ and $P \in \mathbb{L}_\pi$, then $a(x).P \in \mathbb{L}_\pi$,

(sending prefix) if $a, x \in N$ and $P \in \mathbb{L}_\pi$, then $\overline{a}\langle x\rangle.P \in \mathbb{L}_\pi$,

(choice) if $P, Q \in \mathbb{L}_\pi$, then $(P + Q) \in \mathbb{L}_\pi$,

(concurrency) if $P, Q \in \mathbb{L}_\pi$, then $(P \mid Q) \in \mathbb{L}_\pi$,

(scoping) if $x \in N$, $P \in \mathbb{L}_\pi$, then $(\nu x)\ P \in \mathbb{L}_\pi$,

(replication) if $P \in \mathbb{L}_\pi$, then $!P \in \mathbb{L}_\pi$.

Any element $P \in \mathbb{L}_\pi$ is called a π-process. If the binding order is clear, we leave out parentheses.
The free names of a π-process $P \in \mathbb{L}_\pi$, written $\mathfrak{F}(P)$ with $\mathfrak{F} : \mathbb{L}_\pi \to \wp(N)$ are given inductively via:

- $\mathfrak{F}(0) = \emptyset$,
- $\mathfrak{F}(\tau.P) = \mathfrak{F}(P)$,
- $\mathfrak{F}(a(x).P) = \{a\} \cup (\mathfrak{F}(P) \setminus \{x\})$,
- $\mathfrak{F}(\overline{a}\langle x\rangle.P) = \{a, x\} \cup \mathfrak{F}(P)$,
- $\mathfrak{F}(P + Q) = \mathfrak{F}(P) \cup \mathfrak{F}(Q)$,
- $\mathfrak{F}(P \mid Q) = \mathfrak{F}(P) \cup \mathfrak{F}(Q)$,
- $\mathfrak{F}((\nu x)\ P) = \mathfrak{F}(P) \setminus \{x\}$,
- $\mathfrak{F}(!P) = \mathfrak{F}(P)$,

for any $a, x \in N$ and any $P, Q \in \mathbb{L}_\pi$.

Definition 10 (π-congruence)**.** Two π-processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv\colon \mathbb{L}_\pi \times \mathbb{L}_\pi \to \mathbb{B}$. We define inductively:

(α-conversion) $P \equiv Q$ if both only differ by the choice of bound names,

(choice rules) $P + Q \equiv Q + P$, and $(P + Q) + R \equiv P + (Q + R)$, and $P \equiv P + P$

(concurrency rules) $P \mid Q \equiv Q \mid P$, and $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, and $P \equiv P \mid 0$,

(scoping rules) $(\nu x)\ (\nu y)\ P \equiv (\nu y)\ (\nu x)\ P$, and $(\nu x)\ (P \mid Q) \equiv P \mid ((\nu x)\ Q)$ if $x \notin \mathfrak{F}(P)$, and $(\nu x)\ 0 \equiv 0$,

(replication rules) $!P \equiv P \mid !P$,

for any names $x, y \in N$ and processes $P, Q, R \in \mathbb{L}_\pi$.

Definition 11 (π-substitution)**.** For a π-process P we write $P[y := z]$ for the π-process where every free occurrence of name y is replaced by name z. Formally, we define:

- $0[y := z] = 0$,
- $(\tau.P)[y := z] = \tau.(P[y := z])$,
- $(a(x).P)[y := z] = z(x).P$ for $a = y$ and $x = y$,
- $(a(x).P)[y := z] = a(x).P$ for $a \neq y$ and $x = y$,
- $(a(x).P)[y := z] = z(x).(P[y := z])$ for $a = y$ and $x \neq y$,
- $(a(x).P)[y := z] = a(x).(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{z}\langle z\rangle.(P[y := z])$ for $a = y$ and $x = y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{a}\langle z\rangle.(P[y := z])$ for $a \neq y$ and $x = y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{z}\langle x\rangle.(P[y := z])$ for $a = y$ and $x \neq y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{a}\langle x\rangle.(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(P + Q)[y := z] = (P[y := z]) + (Q[y := z])$,
- $(P \mid Q)[y := z] = (P[y := z]) \mid (Q[y := z])$,
- $((\nu x)\ P)[y := z] = ((\nu x)\ P)$ for $x = y$,
- $((\nu x)\ P)[y := z] = (\nu x)\ (P[y := z])$ for $x \neq y$,
- $(!P)[y := z] = !(P[y := z])$,

for any names $a, x, y, z \in N$ and processes $P, Q \in \mathbb{L}_\pi$.

Definition 12 (π-evaluation)**.** An evaluation of a π-process P is a sequence of π-processes $P_0 \rightarrowtail ... \rightarrowtail P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrowtail: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i. We define the following evaluation rules:

(reaction) $(a(x).P + P') \mid (\overline{a}\langle y\rangle.Q + Q') \rightarrowtail_{\text{REACT}} (P[x := y]) \mid Q$,

(τ transition) $\tau.P + P' \rightarrowtail_{\text{TAU}} P$,

(parallel execution) $P \mid R \rightarrowtail_{\text{PAR}} Q \mid R$ if it holds that $P \rightarrowtail Q$,

(restricted execution) $(\nu x)\ P \rightarrowtail_{\text{RES}} (\nu x)\ Q \mid R$ if it holds that $P \rightarrowtail Q$,

(structural congruence) $P' \rightarrowtail_{\text{STRUCT}} Q'$ if it holds that $P \rightarrowtail Q$ and $P \equiv P'$ and $Q \equiv Q'$,

for any names $a, x, y \in N$ and processes $P, P', Q, Q' \in \mathbb{L}_\pi$ where $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$ is the predicate for structural congruence.