Computational Intelligence 2022/23

Exam* — February 9th, 2023

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. 17 pts.	pts.
Topic 3	max. 23 pts.	pts.
Topic 4	max. 20 pts.	pts.
Topic 5	max. 20 pts.	pts.
Total max. 90 pts.		pts.

---

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

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 **17pts**

For scientific purposes, we want to deploy a *SquirrelBot*, i.e., a small robotic agent (cf. Definition 1 in the appendix) that is able to drive across soil and dig for nuts. It can observe its exact location $p \in \mathcal{L}$ with $\mathcal{L} = [0; 100] \times [0; 100] \subset \mathbb{R}^2$ on a continuous 2D plane representing the accessible soil. In the same plane it can also observe a marked target location $g \in \mathcal{L}$ that it wants to navigate to. The value of g is provided by a *MemoryAgent* that tries to remember all locations where nuts are buried, but to the *SquirrelBot* that location g (like its own location p) is just part of its observation. The *SquirrelBot* can execute an action a of the form $a = (\delta x, \delta y, dig) \in \mathbb{R} \times \mathbb{R} \times \mathbb{B}$ once per time step. The action is resolved by the environment by updating the robot's own location by $\delta x, \delta y$ and then digging at the new location iff $dig = True$. However, all actions that attempt to drive a distance greater than 1 (i.e., $\sqrt{(\delta x)^2 + (\delta y)^2} > 1$) per time step are completely ignored by the environment.

(i) Assume that the complete state of the system is given by the position p_t of the *SquirrelBot* at time step t, the position of the marked target location g_t, and a flag dug_t marking if the *SquirrelBot* attempted to dig (after driving) at time step t, i.e., the whole system generates a sequence of states $\langle s_t \rangle_{0 \leq t \leq T}$ for some fixed maximum episode length $T \in \mathbb{N}$ and $s_t \in \mathcal{L} \times \mathcal{L} \times \mathbb{B}$. Give a goal predicate $\gamma : \langle \mathcal{L} \times \mathcal{L} \times \mathbb{B} \rangle \to \mathbb{B}$ so that $\gamma(\langle p_t, g_t, dug_t \rangle_{0 \leq t \leq T})$ holds iff the agent has at one point in time attempted to dig at a location nearer than 1 to the target location g. (5pts)
Hint: You can use the function $\texttt{dist} : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$ to compute the Euclidean distance between two points in $\mathcal{L}$, i.e., $\texttt{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

(ii) Assume that the whole plane of soil is without obstacles and thus easily navigable for the *SquirrelBot*. Give a policy π that always fulfills the goal predicate γ eventually regardless of the initial state. Also give π's type signature. (12pts)
Hint: You do not need to construct the fastest such policy.

3 Optimization and Fuzzy Logic **23pts**

We consider the *MemoryAgent*, which should provide a digging *SquirrelBot* with a target location to dig at. Despite its name, however, the *MemoryAgent* has no recollection of where good locations $g \in \mathcal{L}$ with $\mathcal{L} = [0;100] \times [0;100] \subset \mathbb{R}^2$ might be. Thus, we attempt to find locations $g \in \mathcal{L}$ via optimization of the fitness function $\phi : \mathcal{L} \to \mathcal{N}$ where $\mathcal{N} = \{5, 4, 3, 2, 1, 0\}$ is the space of nutrition scores so that $\phi(g)$ is the nutrition score of the items (hopefully nuts) found by digging at location g. A higher nutrition score is better.

(i) We now want to use *simulated annealing* (cf. Algorithm 1 in the appendix) to optimize for ϕ. To complete the algorithm, we need to define a nice *neighbors* function. A nice *neighbors* function has the following properties:

- It does not include candidates whose Euclidean distance to the given candidate is greater than 1.
- It does not include candidates which are not part of the search space.
- Its transitive hull is the whole search space.

Give a nice *neighbors* function. (3pts)
Hint: You can use the function `dist` $: \mathcal{L} \times \mathcal{L} \to \mathbb{R}$ to compute the Euclidean distance between two points in $\mathcal{L}$, i.e., $\texttt{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

(ii) Assume now that for a concrete problem instance the fitness function ϕ is given via

$$\phi(x) = \begin{cases} 5 & \text{if } \texttt{dist}(x, (42, 42)) < 1, \\ 3 & \text{if } \texttt{dist}(x, (1, 99)) < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Briefly explain why we would expect simulated annealing to perform badly when optimizing for this fitness function ϕ? (3pts)

(iii) Now imagine there is an agent called *MrRogers* at a fixed position $m = (45, 46) \in \mathcal{L}$. Being in the neighborhood of that agent is also a good position to be in, i.e., formally there exists a different fitness function $\psi : \mathcal{L} \to \mathbb{R}$ with

$$\psi(x) = 1000 - \texttt{dist}(x, (45, 46)).$$

Now imagine we already know a solution $x^+ \in \mathcal{L}$ so that $\psi(x^+) = 999.2$, but we want to search for $\psi(x) \geq 999.8$ for some x. Briefly explain why simulated annealing as we have defined it within this chapter, regardless of the parameter choice, is not particularly helpful (when compared to random search, e.g.) for this optimization challenge? (3pts)

(iv) A squirrel expert tells us that at a location $x \in \mathcal{L}$ we can compute a squirrel's hunger via

$$\mathit{hungry}(x) = 1 - \frac{\phi(x)}{10}$$

and a squirrel's calmness via

$$\mathit{calm}(x) = \frac{\psi(x)}{1000}$$

where ϕ and ψ are given as stated above. Using fuzzy logic where

$$\mathit{quite}(x) = \min(2x, 1),$$

the expert states that a squirrel's overall happiness at location x depends on it being "calm or not quite hungry" or to be exact

$$\mathit{happy}(x) = \texttt{OR}\Big(\mathit{calm}(x), \texttt{NOT}\big(\mathit{quite}(\mathit{hungry}(x))\big)\Big)$$

where `NOT` and `OR` are used as is common in fuzzy logic. Compute the fuzzy happiness of a squirell at location $x = (42, 42)$, i.e., compute $\mathit{happy}((42, 42))$. What is the global maximum of the combined fitness function $\omega : \mathcal{L} \to \mathbb{R}$ given via $\omega(x) = \mathit{happy}(x)$? (14pts)

4 Game Theory and Coordination **20pts**

We consider two agents fighting to receive a common resource (*MrRogers*' affection, e.g.). If both agents agree on a schedule, one agent goes first and receives payoff 5 and the other agent goes second and receives payoff 4. However, if both agents do not reach an agreement, they will alienate the common resource dispatcher and both receive payoff 0. Formally, we consider the following two-player normal-form game we call *affection schedule*:

$$\begin{array}{cc} & \begin{array}{cc} \text{F} & \text{S} \end{array} \\ \begin{array}{c} \text{F} \\ \text{S} \end{array} & \begin{pmatrix} 0,0 & 5,4 \\ 4,5 & 0,0 \end{pmatrix} \end{array}$$

(i) Compute <u>all</u> pure Nash equilibria (cf. Definitions 6 and 9 in the appendix) for *affection schedule* by denoting all best respones. (4pts)

(ii) Compute <u>all</u> strategies on the Pareto front (cf. Definition 7 in the appendix) for *affection schedule* by denoting all relationships of Pareto-dominance between strategies. (4pts)

(iii) Consider two agents A and B whose behavior is defined by the following processes in π-calculus

$$\begin{aligned} A(\mathit{affection}) &= \overline{\mathit{go\_first}}.\mathit{give\_affection}(x).A(\texttt{add}(\mathit{affection}, x)) \\ B(\mathit{affection}) &= \overline{\mathit{go\_second}}.\mathit{give\_affection}(x).B(\texttt{add}(\mathit{affection}, x)) \end{aligned}$$

where $\texttt{add}(x, y) = x + y$ performs <u>arithmetic</u> addition (not choice in π-calculus). Give a π-process MrR so that the process

$$\mathit{MrR} \mid A(0) \mid B(0)$$

runs indefinitely and A and B accumulate *affection* according to the rules of *affection schedule.* (5pts)
<u>Hint:</u> The processes A and B remain fixed; MrR does <u>not need</u> to work with any arbitrary processes besides exactly A and B as defined above.

(iv) Without any squirrels, evaluate the following π-processes step by step until they cannot be reduced any further. (7pts)

$$\big(a.A + b.(c.C + d.D)\big) \mid \overline{c}.0 \mid \overline{a}.0$$

$$\overline{a}.\overline{b}.Z \mid !(a.X + b.Y)$$

$$a.0 + \big(b.\overline{c}.0 \mid (\overline{b}.c.0 + b.\overline{c}.0)\big)$$

5 Scientific Reading: Inverse Reinforcement Learning **20pts**

Read the following (correct) explanation of the novel technique *inverse reinforcement learning*, which we did not discuss in the lecture, and answer the questions afterwards.

CoIn Team:
Give a full-page explanation of inverse reinforcement learning suitable for computer science graduate students and include historical remarks and mathematical definitions.

ChatGPT Assistant:
Inverse Reinforcement Learning (IRL) is a subfield of Reinforcement Learning (RL) that aims to infer a reward function from observed behavior of an expert in an environment. The idea is to learn a reward function that would have generated the observed behavior, rather than learning a policy directly from interaction with the environment. The history of IRL can be traced back to the 1990s when the concept of reward-based learning was first introduced in the field of RL. However, the field of IRL started gaining traction in the early 2000s with the introduction of algorithms such as Maximum Entropy IRL and Bayesian IRL. These algorithms made it possible to learn reward functions from observed behavior, and they paved the way for further development in the field of IRL.

IRL is motivated by the observation that in many real-world RL problems, it is difficult to specify a reward function that accurately captures the desired behavior. For example, in a robotics problem, it may be difficult to specify a reward function that encourages a robot to move smoothly and avoid obstacles. By observing the behavior of an expert in the environment, IRL allows for the inference of a reward function that would have generated the expert's behavior.

In mathematical terms, let π_E be the policy of the expert and π_θ be a policy parameterized by θ that is learned from interaction with the environment. The goal of IRL is to find the reward function, R, that best explains the observed behavior of the expert, π_E:

$$R^* = \arg\min_R D\big(\pi_E, \pi_\theta(R)\big)$$

where D is a distance metric that measures the difference between the expert's behavior and the behavior of the learned policy. The learned reward function can then be used as a guide for learning a policy that mimics the expert's behavior.

There are several algorithms for IRL, including Maximum Entropy IRL, Bayesian IRL, and Gradient-Based IRL. Each of these algorithms has its own strengths and weaknesses, and the choice of algorithm will depend on the specific problem being solved.

Maximum Entropy IRL is a popular approach that seeks to find the reward function that maximizes the entropy of the policy that is optimized with respect to the reward function. This approach has been shown to be effective in a variety of domains, including robotics and gaming.

Bayesian IRL is another popular approach that uses Bayesian inference to learn a reward function from observed behavior. This approach has the advantage of allowing for uncertainty in the learned reward function, which can be useful in problems where the expert's behavior is ambiguous.

Gradient-Based IRL is an approach that uses gradient-based optimization to learn a reward function from observed behavior. This approach has the advantage of being computationally efficient, and it has been used to learn reward functions in large and complex environments.

IRL has been applied in a variety of domains, including robotics, computer vision, and gaming, and it has the potential to make it easier to apply RL to real-world problems by allowing the use of human-defined reward functions. It is an active area of research with many open questions and opportunities for further development.

In conclusion, IRL is a promising subfield of RL that offers new and innovative ways to learn reward functions from observed behavior. It has the potential to greatly expand the scope of RL by making it easier to apply to real-world problems, and it is an area of ongoing research and development.

CoIn Team:
What basic exam question could you ask about inverse reinforcement learning?

ChatGPT Assistant:
A basic exam question about Inverse Reinforcement Learning could be:

What is Inverse Reinforcement Learning and why is it used in Reinforcement Learning?

[. . .]

CoIn Team:
Alright, let's talk about squirrels...

[. . .]

(i) Answer the question suggested by ChatGPT: What is Inverse Reinforcement Learning and why is it used in Reinforcement Learning? (8pts)

(ii) In the mathematical formula for the optimal reward function R^*, ChatGPT introduces an expert policy π_E that can be used with the type signature we are used to, i.e., $\pi_E : \mathcal{O} \to \mathcal{A}$. It also introduces a trained policy π_θ that is defined by a (network) parameter vector $\theta \in \Theta$ and is parametrized on a given reward function R that it is trained on, i.e., $\pi_\theta : \mathcal{R} \to \mathcal{O} \to \mathcal{A}$ where $\mathcal{R}$ is the space of all possible reward functions.

Why does the expert policy π_E not need parameters (like θ)? Given a set of training observations $\mathbb{O}$, how can the behavior of two policies (like π_E and π_θ) be compared? Describe a possible implementation for a function D as discussed in the text. (8pts)
Hint: You do not need to give a full definition for D but just sketch a possible approach.

(iii) In the text it is briefly mentioned that Gradient-Based IRL might be computationally more efficient compared to other variants of IRL. Give a reasonable explanation for this statement. (4pts)

Appendix: Definitions

Notation. $\wp(X)$ 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.).

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 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 (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 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 (simulated annealing)**.** Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let *neighbors* $: \mathcal{X} \to \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x_u \in \mathcal{X}$. Let $x'_u \sim \textit{neighbors}(x_u)$ be a neighbor state of x_u that was drawn at random. Let $T : \mathbb{N} \to \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \to \mathbb{P}$ be a function that returns an acceptance probability given two target values and a temperature. Commonly, we set

$$A(Q, Q', C) = e^{\frac{-(Q'-Q)}{C}}$$

for $\mathcal{T} \subseteq \mathbb{R}$. Let $r \sim \mathbb{P}$ be a random number drawn from $\mathbb{P} = [0; 1] \subset \mathbb{R}$. The process $\mathcal{D}$ continues via simulated annealing if e is of the form

$$e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = \begin{cases} x'_t & \textit{if } \tau(x'_t) \leq \tau(x_t) \textit{ or } r \leq A\big(\tau(x_t), \tau(x'_t), T(t)\big), \\ x_t & \textit{otherwise.} \end{cases}$$

Definition 3 (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 $P(s'|s, a) = \Pr(s_{t+1} = s' \mid s_t = s \wedge a_t = a)$.
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

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

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

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 $r_t = R(s_t, a_t, s_{t+1})$.

Definition 4 (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 5 (normal-form game)**.** Let $G = \{G^{[1]}, ..., G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times ... \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i. Let $\chi : \mathcal{A} \to \mathcal{T}$ be a utility function for the joint action space $\mathcal{A}$ and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times ... \times \mathcal{T}^{[N]}$ where $\mathcal{T}^{[i]}$ is the target space of agent $G^{[i]}$ for all i. Unless stated otherwise, the utility χ is to be maximized. From χ we can derive a set of single-agent utility functions $\chi^{[i]} : \mathcal{A} \to \mathcal{T}^{[i]}$ for all i. A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game.*

Definition 6 (strategy)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. In a single iteration of the game, an agent $G^{[i]}$'s behavior is given by a (possibly randomized) policy $\pi^{[i]} : () \to \mathcal{A}^{[i]}$. Then, $\pi^{[i]}$ is also called $G^{[i]}$'s *strategy*. If multiple iterations of the game are played, an agent $G^{[i]}$'s behavior can be given by a (possibly randomized) policy $\pi^{[i]} : \mathcal{A}^n \to \mathcal{A}^{[i]}$ where n is a number of previous iterations. The agent's strategy is then conditioned on a list of previous joint actions.
An agent $G^{[i]}$ whose actions are given via a policy $\pi^{[i]}$ of the form $\pi^{[i]}(\_) = a^{[i]}$ for some action $a^{[i]} \in \mathcal{A}^{[i]}$ is playing a pure strategy.
An agent $G^{[i]}$ whose actions are given via a policy $\pi^{[i]}$ of the form $\pi^{[i]}(\_) \sim A^{[i]}$ according to some distribution over $A^{[i]} \subseteq \mathcal{A}^{[i]}$ is playing a mixed strategy.
If a mixed strategy is based on a uniform distribution over actions $A^{[i]} \subseteq \mathcal{A}^{[i]}$, we write $\pi^{[i]} = A^{[i]}$ as a shorthand.

Definition 7 (Pareto front for strategies)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(\_) = (\pi^{[1]}(\_), ..., \pi^{[|G|]}(\_))$ Pareto-dominates another joint strategy $\pi'(\_) = (\pi'^{[1]}(\_), ..., \pi'^{[|G|]}(\_))$ iff for all agents $G^{[i]}$ it holds that

$$\chi^{[i]}(\pi(\_)) \geq \chi^{[i]}(\pi'(\_))$$

and there exists some agent $G^{[j]}$ so that

$$\chi^{[j]}(\pi(\_)) > \chi^{[j]}(\pi'(\_)).$$

A joint strategy π is Pareto-optimal iff there is no other strategy π' so that π' Pareto-dominates π.
The set of all Pareto-optimal strategies is called the Pareto front.

Definition 8 (best response)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, ..., G^{[i-1]}, G^{[i+1]}, ..., G^{[N]}$, i.e., all agents except $G^{[i]}$. Let $\pi^{[i]} \oplus \pi^{[-i]}$ be a joint strategy of all agents then. Given a strategy $\pi^{[-i]}$ for all agents except $G^{[i]}$, $G^{[i]}$'s *best response* is the strategy $\pi^{*[i]}$ so that for all strategies $\pi'^{[i]}$ it holds that

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

Definition 9 (Nash equilibrium)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game played for a single iteration. A joint strategy π is a *Nash equilibrium* iff for all agents $G^{[i]}$ it holds that $\pi^{[i]}$ is the best response to $\pi^{[-i]}$.

Definition 10 (π-process (shortened))**.** 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.

Definition 11 (π-congruence)**.** Two π-processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv : \mathbb{L}_\pi \times \mathbb{L}_\pi \to \mathbb{B}$. 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 12 (π-evaluation)**.** An evaluation of a π-process P is a sequence of π-processes $P_0 \rightarrowtail \ldots \rightarrowtail P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrowtail : \mathbb{L}_\pi \to \mathbb{L}_\pi$ 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 $\equiv : \mathbb{L}_\pi \times \mathbb{L}_\pi \to \mathbb{B}$ is the predicate for structural congruence.