

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
---------------	----------	--------------------------	-----------

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 <u>Turing Test</u>	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 <u>expert systems</u>	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 <u>Prolog</u>	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 <u>weights</u>	iv angles
---------------	-----------	--------------------	-----------

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

i only constant functions	ii only positive functions	iii only differentiable functions	iv <u>any arbitrary function</u>
---------------------------	----------------------------	-----------------------------------	----------------------------------

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

i selection	ii <u>imagination</u>	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 <u>AlphaGo</u>	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 <u>the micro-macro problem</u>
------------------------	---------------------------	----------------------------	--------------------------------------

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

i <u>distributions within a population</u>	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 <u>singularity</u>	ii nexus	iii world between worlds	iv matrix
-------------------------	----------	--------------------------	-----------

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) According to Alan Turing, the goal of the *Imitation Game*, with a party A and B and a judge C , is for party A to convince ...

i B that $A \approx C$	ii C that $B \approx A$	iii C that $A \approx A$	iv A that $B = B$
--------------------------	---------------------------	----------------------------	---------------------

(b) Which of the following is *not* commonly in use as a policy *encoding*?

i Python code	ii behavior trees	iii random execution	iv pseudo-code
---------------	-------------------	----------------------	----------------

(c) *Simulated annealing* is an...?

i optimization algorithm	ii game strategy	iii agent policy	iv type of neural network
--------------------------	------------------	------------------	---------------------------

(d) For *evolutionary algorithms* we usually do *not* associate more of which of the following techniques with *increased variation*?

i mutation	ii crossover	iii selection	iv recombination
------------	--------------	---------------	------------------

(e) As measured by sample efficiency, all optimization algorithms perform the same when averaged over all possible target functions. What is this theorem called?

i always cheap breakfast	ii no free lunch	iii sometimes expensive dinner	iv never available midnight snack
--------------------------	------------------	--------------------------------	-----------------------------------

(f) The *Pareto front* denotes the set of samples which we consider as ...

i non-optimal	ii fully randomized	iii dominated	iv non-dominated
---------------	---------------------	---------------	------------------

(g) In the reading exercise we have seen Prolog used to implement an *expert system*, emulating the decision-making ability of a human (expert). For their inference advantage these systems mainly rely on large bodies of... ?

i goal classes	ii knowledge	iii probability	iv uncertainty
----------------	--------------	-----------------	----------------

(h) Which of the following properties are *not* describing an evolutionary (robotic) swarm as we have defined it in the lecture?

i relatively simple	ii fully interdependent	iii mass producible	iv mostly identical
---------------------	-------------------------	---------------------	---------------------

(i) The quote '[a] physical agent that performs tasks by manipulating the physical world' (Russel and Norvig (2003)) best describes a ... ?

i robot	ii PhD student	iii neural network	iv swarm collective
---------	----------------	--------------------	---------------------

(j) Which of the following is *not* commonly considered a swarm behavior task?

i autonomous assembly	ii parallel computation	iii coordinated motion	iv decision making
-----------------------	-------------------------	------------------------	--------------------

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) According to Alan Turing, the goal of the *Imitation Game*, with a party A and B and a judge C , is for party A to convince ...

i B that $A \approx C$	ii <u>C that $B \approx A$</u>	iii C that $A \approx A$	iv A that $B = B$
--------------------------	---	----------------------------	---------------------

(b) Which of the following is *not* commonly in use as a policy *encoding*?

i Python code	ii behavior trees	iii <u>random execution</u>	iv pseudo-code
---------------	-------------------	--------------------------------	----------------

(c) *Simulated annealing* is an...?

i <u>optimization algorithm</u>	ii game strategy	iii agent policy	iv type of neural network
------------------------------------	------------------	------------------	---------------------------

(d) For *evolutionary algorithms* we usually do *not* associate more of which of the following techniques with *increased variation*?

i mutation	ii crossover	iii <u>selection</u>	iv recombination
------------	--------------	-------------------------	------------------

(e) As measured by sample efficiency, all optimization algorithms perform the same when averaged over all possible target functions. What is this theorem called?

i always cheap breakfast	ii <u>no free lunch</u>	iii sometimes expensive dinner	iv never available midnight snack
--------------------------	----------------------------	--------------------------------	-----------------------------------

(f) The *Pareto front* denotes the set of samples which we consider as ...

i non-optimal	ii fully randomized	iii dominated	iv <u>non-dominated</u>
---------------	---------------------	---------------	----------------------------

(g) In the reading exercise we have seen Prolog used to implement an *expert system*, emulating the decision-making ability of a human (expert). For their inference advantage these systems mainly rely on large bodies of... ?

i goal classes	ii <u>knowledge</u>	iii probability	iv uncertainty
----------------	------------------------	-----------------	----------------

(h) Which of the following properties are *not* describing an evolutionary (robotic) swarm as we have defined it in the lecture?

i relatively simple	ii fully interdependent	iii mass producible	iv mostly identical
---------------------	-------------------------	---------------------	---------------------

(i) The quote ‘[a] physical agent that performs tasks by manipulating the physical world’ (Russel and Norvig (2003)) best describes a . . . ?

i robot	ii PhD student	iii neural network	iv swarm collective
---------	----------------	--------------------	---------------------

(j) Which of the following is *not* commonly considered a swarm behavior task?

i autonomous assembly	ii parallel computation	iii coordinated motion	iv decision making
-----------------------	-------------------------	------------------------	--------------------

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) Turing designed the Turing Test to test an artificial intelligence's ability to...

i simulate a Turing machine.	ii fit inside a robot.	iii communicate like a human being.	iv produce the technological singularity.
------------------------------	------------------------	-------------------------------------	---

(b) Let $\text{equal}(x, y) \iff (\forall p : p(x) \iff p(y))$. This approach to give an equality predicate is called...

i emotional	ii intentional	iii extensional	iv logical
-------------	----------------	-----------------	------------

(c) In accordance with the main focus of the programming language Prolog, its name is derived from...

i producing log files	ii programming logic	iii probabilistic logarithms	iv pranking old ogres
-----------------------	----------------------	------------------------------	-----------------------

(d) Which of these is not a standard component of any neural network?

i weights	ii concentrations	iii biases	iv activation functions
-----------	-------------------	------------	-------------------------

(e) The optimization algorithm *simulated annealing* compares its current solution candidate to a randomly generated solution candidate from its neighborhood. The chance of accepting a worse solution candidate...

i is virtually always 100%.	ii increases with computational effort spent on the new candidate.	iii decreases according to a temperature parameter.	iv is computed by an artificial intelligence.
-----------------------------	--	---	---

(f) A programming language that is especially designed to program interaction-based systems and has found usage in large-scale communication applications is called ...

i Münch	ii Augsbu	iii Erlang	iv Neu-U
---------	-----------	------------	----------

(g) One of the first breakthroughs in AI image generation around 2015 was achieved by the GAN model. Its key concept involved network training that is...

i adversarial	ii absolute	iii agnostic	iv astonishing
---------------	-------------	--------------	----------------

(h) In a single-iteration game of Prisoner's Dilemma with actions *cooperate* (C) and *defect* (D), the Nash equilibrium is the joint action...

i (C, C)	ii (C, D)	iii (D, C)	iv (D, D)
----------	-----------	------------	-----------

(i) When we deploy a large amount of simple robots without any centralized control unit, we call these robots...

i a swarm	ii a school	iii a pack	iv an empire
-----------	-------------	------------	--------------

(j) The idea of technological singularity is based on ever accelerating...

i warp speeds	ii scientific progress	iii force abilities	iv agent cloning
---------------	------------------------	---------------------	------------------

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) Turing designed the Turing Test to test an artificial intelligence's ability to...

i simulate a Turing machine.	ii fit inside a robot.	iii communicate like a human being.	iv produce the technological singularity.
------------------------------	------------------------	-------------------------------------	---

(b) Let $\text{equal}(x, y) \iff (\forall p : p(x) \iff p(y))$. This approach to give an equality predicate is called...

i emotional	ii intentional	iii extensional	iv logical
-------------	----------------	-----------------	------------

(c) In accordance with the main focus of the programming language Prolog, its name is derived from...

i producing log files	ii programming logic	iii probabilistic logarithms	iv pranking old ogres
-----------------------	----------------------	------------------------------	-----------------------

(d) Which of these is not a standard component of any neural network?

i weights	ii concentrations	iii biases	iv activation functions
-----------	-------------------	------------	-------------------------

(e) The optimization algorithm *simulated annealing* compares its current solution candidate to a randomly generated solution candidate from its neighborhood. The chance of accepting a worse solution candidate...

i is virtually always 100%.	ii increases with computational effort spent on the new candidate.	iii decreases according to a temperature parameter.	iv is computed by an artificial intelligence.
-----------------------------	--	---	---

(f) A programming language that is especially designed to program interaction-based systems and has found usage in large-scale communication applications is called ...

i Münch	ii Augsbu	iii Erlang	iv Neu-U
---------	-----------	------------	----------

(g) One of the first breakthroughs in AI image generation around 2015 was achieved by the GAN model. Its key concept involved network training that is...

i adversarial	ii absolute	iii agnostic	iv astonishing
---------------	-------------	--------------	----------------

(h) In a single-iteration game of Prisoner's Dilemma with actions *cooperate* (C) and *defect* (D), the Nash equilibrium is the joint action...

i (C, C)	ii (C, D)	iii (D, C)	iv (D, D)
----------	-----------	------------	-----------

(i) When we deploy a large amount of simple robots without any centralized control unit, we call these robots...

i a swarm	ii a school	iii a pack	iv an empire
-----------	-------------	------------	--------------

(j) The idea of technological singularity is based on ever accelerating...

i warp speeds	ii scientific progress	iii force abilities	iv agent cloning
---------------	------------------------	---------------------	------------------

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) If you train a large language model to imitate how you text with people and then ask your fellow students to discern if they are writing with you or with the large language model, you are effectively running an instance of...

i the Turing Test	ii Prolog	iii the Singularity	iv FunSearch
-------------------	-----------	---------------------	--------------

(b) In accordance with the main focus of the programming language Prolog, its name is derived from ...

i producing log files	ii programming logic	iii probability logarithms	iv pranking old ogres
-----------------------	----------------------	----------------------------	-----------------------

(c) Given a fixed graph structure, a neural network is fully defined by three categories of entities. Which is not one of them?

i weights	ii biases	iii curvatures	iv activation functions
-----------	-----------	----------------	-------------------------

(d) In a standard evolutionary algorithm, *variation* operators are usually employed to increase which property of the optimization?

i selection	ii exploration	iii exploitation	iv production
-------------	----------------	------------------	---------------

(e) Compared to the game of Go, which was only considered artificially 'mastered' as of 2016, the game of Chess was solvable much earlier. The software that was able to beat then world champion Garri Kasparow in the late 1990s was called ...

i Deep Blue	ii AlphaGo	iii DeepMind	iv ChatGPT
-------------	------------	--------------	------------

(f) A multi-agent system made up of many similar agents that have a shared interest but little communication and only locally executed algorithms is called ...

i market	ii evolutionary stable	iii goal class	iv swarm
----------	------------------------	----------------	----------

(g) One of the main difficulties for finding effective strategies in the game *Lemonade Stand* (sometimes also called *Glühwein Stand*) lies in the opponent strategies' ...

i action space	ii non-stationarity	iii beverage quality	iv policy model
----------------	---------------------	----------------------	-----------------

(h) In a multi-iteration tournament of the game *Prisoner's Dilemma* where you play against each opponent for an unknown number of games, you are allowed to adopt any of these four strategies. Which one of these is expected to be the most successful over the course of such a tournament?

i always Cooperate	ii always <i>Defect</i>	iii give up	iv tit-for-tat
--------------------	-------------------------	-------------	----------------

(i) The *Hawk–Dove* game was modeled as a possible explanation for the ecological balance of ...

i aggression between peers	ii differences in size	iii animal intelligence	iv CO2 emissions
----------------------------	------------------------	-------------------------	------------------

(j) The outlook of machine learning expert Richard Sutton on the eventual superiority of using search-based algorithms instead of human-experience-based knowledge was summarized in his famous 2019 publication titled ...

i “The Bitter Lesson”	ii “The Bitter Lemon”	iii “The Barren Line”	iv “The Better LISP”
-----------------------	-----------------------	-----------------------	----------------------

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) If you train a large language model to imitate how you text with people and then ask your fellow students to discern if they are writing with you or with the large language model, you are effectively running an instance of...

i <u>the Turing Test</u>	ii Prolog	iii the Singularity	iv FunSearch
-----------------------------	-----------	---------------------	--------------

(b) In accordance with the main focus of the programming language Prolog, its name is derived from ...

i producing log files	ii <u>programming logic</u>	iii probability logarithms	iv pranking old ogres
-----------------------	-----------------------------	----------------------------	-----------------------

(c) Given a fixed graph structure, a neural network is fully defined by three categories of entities. Which is not one of them?

i weights	ii biases	iii <u>curvatures</u>	iv activation functions
-----------	-----------	-----------------------	-------------------------

(d) In a standard evolutionary algorithm, *variation* operators are usually employed to increase which property of the optimization?

i selection	ii <u>exploration</u>	iii exploitation	iv production
-------------	-----------------------	------------------	---------------

(e) Compared to the game of Go, which was only considered artificially 'mastered' as of 2016, the game of Chess was solvable much earlier. The software that was able to beat then world champion Garri Kasparow in the late 1990s was called ...

i <u>Deep Blue</u>	ii AlphaGo	iii DeepMind	iv ChatGPT
--------------------	------------	--------------	------------

(f) A multi-agent system made up of many similar agents that have a shared interest but little communication and only locally executed algorithms is called ...

i market	ii evolutionary stable	iii goal class	iv swarm
----------	------------------------	----------------	-----------------

(g) One of the main difficulties for finding effective strategies in the game *Lemonade Stand* (sometimes also called *Glühwein Stand*) lies in the opponent strategies' ...

i action space	ii non-stationarity	iii beverage quality	iv policy model
----------------	----------------------------	----------------------	-----------------

(h) In a multi-iteration tournament of the game *Prisoner's Dilemma* where you play against each opponent for an unknown number of games, you are allowed to adopt any of these four strategies. Which one of these is expected to be the most successful over the course of such a tournament?

i always Cooperate	ii always <i>Defect</i>	iii give up	iv tit-for-tat
--------------------	-------------------------	-------------	-----------------------

(i) The *Hawk–Dove* game was modeled as a possible explanation for the ecological balance of ...

i aggression between peers	ii differences in size	iii animal intelligence	iv CO2 emissions
-----------------------------------	------------------------	-------------------------	------------------

(j) The outlook of machine learning expert Richard Sutton on the eventual superiority of using search-based algorithms instead of human-experience-based knowledge was summarized in his famous 2019 publication titled ...

i "The Bitter Lesson"	ii "The Bitter Lemon"	iii "The Barren Line"	iv "The Better LISP"
------------------------------	-----------------------	-----------------------	----------------------

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
---------------	----------	--------------------------	-----------

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 <u>Turing Test</u>	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 <u>expert systems</u>	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 <u>Prolog</u>	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 <u>weights</u>	iv angles
---------------	-----------	--------------------	-----------

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

i only constant functions	ii only positive functions	iii only differentiable functions	iv <u>any arbitrary function</u>
---------------------------	----------------------------	-----------------------------------	----------------------------------

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

i selection	ii <u>imagination</u>	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 <u>AlphaGo</u>	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 <u>the micro-macro problem</u>
------------------------	---------------------------	----------------------------	--------------------------------------

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

i <u>distributions within a population</u>	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 <u>singularity</u>	ii nexus	iii world between worlds	iv matrix
-------------------------	----------	--------------------------	-----------

1 General Knowledge / Fill in the Blanks

10pts

For each of the following gaps in the text, fill in the missing word or short phrase with specific expressions from the field of computational intelligence. There might be more than one possible correct answer, but you only need to give one. Every correctly filled gap is awarded one point. Incorrect or empty gaps will be marked with zero points.

In the field of collective intelligence, we sometimes consider large groups of agents showing complex behavior even though each single agent only follows a rather simple ruleset. By analogy to similarly behaving groups of animals, these groups are called _____ . In nature, we can find such group behavior in many animals such as _____ , for example. This phenomenon — that a group of agents can exhibit a seemingly much more complex behavior than what the single agents are capable of — is referred to as _____. Which behavior patterns tend to evolve in large groups of randomly interacting agents is explained by _____ game theory. A behavior that such a group tends to evolve towards and not change from (once it has evolved) is thus called an _____ .

Alternatively, truly cooperative agents with multi-agent goals can be found in, e.g., _____ (MARL). When each agent makes different, distinct observations about its environment, that environment is clearly characterized by _____ observability. However, the behavior of the environment can still be fully known and fully encoded in that environment's state, which means that the environment fulfills the _____ property. In addition to the state space, action space, and target value space, we need the _____ and the _____ to fully define a decision process for such an environment.

1 General Knowledge / Fill in the Blanks

10pts

For each of the following gaps in the text, fill in the missing word or short phrase with specific expressions from the field of computational intelligence. There might be more than one possible correct answer, but you only need to give one. Every correctly filled gap is awarded one point. Incorrect or empty gaps will be marked with zero points.

In the field of collective intelligence, we sometimes consider large groups of agents showing complex behavior even though each single agent only follows a rather simple ruleset. By analogy to similarly behaving groups of animals, these groups are called swarms. In nature, we can find such group behavior in many animals such as fish / birds / etc., for example. This phenomenon — that a group of agents can exhibit a seemingly much more complex behavior than what the single agents are capable of — is referred to as emergence. Which behavior patterns tend to evolve in large groups of randomly interacting agents is explained by evolutionary game theory. A behavior that such a group tends to evolve towards and not change from (once it has evolved) is thus called an evolutionary stable strategy.

Alternatively, truly cooperative agents with multi-agent goals can be found in, e.g., multi-agent reinforcement learning (MARL). When each agent makes different, distinct observations about its environment, that environment is clearly characterized by partial observability. However, the behavior of the environment can still be fully known and fully encoded in that environment's state, which means that the environment fulfills the Markov property. In addition to the state space, action space, and target value space, we need the cost / reward function and the transition probability function to fully define a decision process for such an environment.

1 General Knowledge / Fill in the Blanks

10pts

For each of the following gaps in the text, fill in the missing word or short phrase with specific expressions from the field of computational intelligence. There might be more than one possible correct answer, but you only need to give one. Every correctly filled gap is awarded one point. Incorrect or empty gaps will be marked with zero points.

Throughout the decades, the approaches to computational intelligence have changed quite drastically. Early systems were trying to encode human knowledge and intelligence into clear rules; these were called _____ systems. ELIZA from 1966 was an early chatbot and an attempt to trick humans into thinking they are chatting with another human, i.e., passing the _____ test. Later attempts often used Prolog, which — also in name — was meant to combine _____ and _____. Prolog was also used within IBM Watson to beat human contestants at a _____ called “Jeopardy!”. Since the early days of computational intelligence, some visionaries have predicted that technological progress would follow an _____ growth pattern and thus someday out-grow all other aspects of human life, a moment in time called technological _____.

Most recently, modern _____ (LLMs) do not have explicit reasoning capabilities like the rule-based approaches of the past. Instead, they use variations of _____ networks to model language, which can capture complex relationships between words and phrases. However, LLMs may sometimes produce outputs that are less reliable or consistent, as they generate responses based on probabilities learned from data. Thus, both have their strengths and are important steps on the way towards the milestone of _____ (AGI).

1 General Knowledge / Fill in the Blanks

10pts

For each of the following gaps in the text, fill in the missing word or short phrase with specific expressions from the field of computational intelligence. There might be more than one possible correct answer, but you only need to give one. Every correctly filled gap is awarded one point. Incorrect or empty gaps will be marked with zero points.

Throughout the decades, the approaches to computational intelligence have changed quite drastically. Early systems were trying to encode human knowledge and intelligence into clear rules; these were called expert systems. ELIZA from 1966 was an early chatbot and an attempt to trick humans into thinking they are chatting with another human, i.e., passing the Turing test. Later attempts often used Prolog, which — also in name — was meant to combine programming and logic. Prolog was also used within IBM Watson to beat human contestants at a quiz show called “Jeopardy!”. Since the early days of computational intelligence, some visionaries have predicted that technological progress would follow an exponential growth pattern and thus someday out-grow all other aspects of human life, a moment in time called technological singularity.

Most recently, modern large language models (LLMs) do not have explicit reasoning capabilities like the rule-based approaches of the past. Instead, they use variations of neural networks to model language, which can capture complex relationships between words and phrases. However, LLMs may sometimes produce outputs that are less reliable or consistent, as they generate responses based on probabilities learned from data. Thus, both have their strengths and are important steps on the way towards the milestone of artificial general intelligence (AGI).

Feedback Aufgabe 1

2022-Exam-Coln.pdf

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf

2025-MockExam1-Coln.pdf

2025-MockExam2-Coln.pdf

Gesamtpunkte (ueber alle Aufgaben)

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 $\gamma : \langle \mathcal{L} \times \mathcal{L} \times \mathbb{B} \rangle \rightarrow \mathbb{B}$ 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 $dist : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}$ to compute the Euclidean distance between two points in \mathcal{L} , i.e., $dist((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

- (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.

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 $a = (\delta x, \delta y, \text{dig}) \in \mathbb{R} \times \mathbb{R} \times \mathbb{B}$ 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 $\text{dig} = \text{True}$. 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 $\gamma : \langle \mathcal{L} \times \mathcal{L} \times \mathbb{B} \rangle \rightarrow \mathbb{B}$ 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 $\text{dist} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}$ to compute the Euclidean distance between two points in \mathcal{L} , i.e., $\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

$$\gamma(\langle p_t, g_t, dug_t \rangle_{0 \leq t \leq T}) \iff \exists t : \text{dist}(p_t, g_t) < 1 \wedge dug_t = \text{True}$$

- (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.

$$\pi : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R} \times \mathbb{R} \times \mathbb{B}$$

$$\pi((x_p, y_p), (x_g, y_g)) = \begin{cases} (+0.1, 0, \text{False}) & \text{if } x_g - x_p > 0.1, \\ (-0.1, 0, \text{False}) & \text{else if } x_g - x_p < -0.1, \\ (0, +0.1, \text{False}) & \text{else if } y_g - y_p > 0.1, \\ (0, -0.1, \text{False}) & \text{else if } y_g - y_p < -0.1, \\ (0, 0, \text{True}) & \text{otherwise.} \end{cases}$$

2 Agents, Environments, Goals

10pts

Consider a beaver in the final steps of building a dam. It just needs to add one more log in precisely the right angle. If it hits the right angle, the dam is now leak-proof and the water level (of the river flowing from behind the dam) rises. The beaver's behavior is thus given by a policy $\pi : \text{Level} \rightarrow \text{Angles}$ where $\text{Level} = [0; \infty) \subset \mathbb{R}$ is the water level in cm and $\text{Angles} = [0; 180] \times [0; 180] \subset \mathbb{R}^2$ are the horizontal and vertical angles in degrees at which the beaver tries to insert the final log into the dam.

(i) Assume that the environment is updated in discrete time steps and thus produces a sequence of states $\langle s_t \rangle_{1 \leq t \leq T}$ where $T \in \mathbb{N}$ is the episode length of the environment and $s_t \in \text{Level} \times \text{Angles}$ for all t . The beaver deems its work successful iff the water level keeps rising from some point in time on for at least three consecutive time steps and all future time steps afterwards. Give a goal predicate $\gamma : \langle \text{Level} \times \text{Angles} \rangle \rightarrow \mathbb{B}$ so that $\gamma(\langle s_t \rangle_{1 \leq t \leq T})$ holds iff the beaver deems its work successful. (4pts)

(ii) We still assume that our environment's current state s is of the form $s \in \text{Level} \times \text{Angles}$. We cannot define a Markov Decision Process (MDP) so that its reward function R is optimal iff the goal predicate γ as given above is fulfilled. Briefly explain why. (2pts)

Definition 1 (Markov decision process (MDP), *shortened definition*). 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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost (reward)* function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ is the *transition probability function* often written as $P(s'|s, a) = \Pr(s_{t+1} = s' | s_t = s \wedge a_t = a)$.

(iii) Now assume that we know that the perfect angles to put the log into the dam are $a^* = (42, 42)$. In order to train the beaver to use exactly these angles, we want to define a reward function $R' : \text{Angles} \rightarrow \mathbb{R}$ so that the beaver achieves the highest reward iff it uses the angles a^* . For all other angles, we want a smooth reward landscape with a gradient pointing towards a^* . Give a possible definition for R' . (4pts)

2 Agents, Environments, Goals

10pts

Consider a beaver in the final steps of building a dam. It just needs to add one more log in precisely the right angle. If it hits the right angle, the dam is now leak-proof and the water level (of the river flowing from behind the dam) rises. The beaver's behavior is thus given by a policy $\pi : \text{Level} \rightarrow \text{Angles}$ where $\text{Level} = [0; \infty) \subset \mathbb{R}$ is the water level in cm and $\text{Angles} = [0; 180] \times [0; 180] \subset \mathbb{R}^2$ are the horizontal and vertical angles in degrees at which the beaver tries to insert the final log into the dam.

- (i) Assume that the environment is updated in discrete time steps and thus produces a sequence of states $\langle s_t \rangle_{1 \leq t \leq T}$ where $T \in \mathbb{N}$ is the episode length of the environment and $s_t \in \text{Level} \times \text{Angles}$ for all t . The beaver deems its work successful iff the water level keeps rising from some point in time on for at least three consecutive time steps and all future time steps afterwards. Give a goal predicate $\gamma : \langle \text{Level} \times \text{Angles} \rangle \rightarrow \mathbb{B}$ so that $\gamma(\langle s_t \rangle_{1 \leq t \leq T})$ holds iff the beaver deems its work successful. (4pts)

$$\gamma(\langle (l_t, -) \rangle_{1 \leq t \leq T}) \iff \exists t : t \leq T - 3 \wedge \forall t' > t : l_{t'} > l_{t'-1}$$

- (ii) We still assume that our environment's current state s is of the form $s \in \text{Level} \times \text{Angles}$. We cannot define a Markov Decision Process (MDP) so that its reward function R is optimal iff the goal predicate γ as given above is fulfilled. Briefly explain why. (2pts)

Definition 1 (Markov decision process (MDP), *shortened definition*). 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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized cost (reward) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ is the *transition probability function* often written as $P(s'|s, a) = \Pr(s_{t+1} = s' | s_t = s \wedge a_t = a)$.

The goal predicate requires a hidden state as the history of water levels is necessary to compute its fulfillment but said history is not given within the state. Thus, it is inherently non-Markovian.

- (iii) Now assume that we know that the perfect angles to put the log into the dam are $a^* = (42, 42)$. In order to train the beaver to use exactly these angles, we want to define a reward function $R' : \text{Angles} \rightarrow \mathbb{R}$ so that the beaver achieves the highest reward iff it uses the angles a^* . For all other angles, we want a smooth reward landscape with a gradient pointing towards a^* . Give a possible definition for R' . (4pts)

$$R'((x, y)) = -(x - 42)^2 - (y - 42)^2$$

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 `state_of : C^2 → M` where $C = [0; 9] ⊂ N$ is the space of grid positions and $M = \{\text{working}, \text{alarming}, \text{defect}\}$ 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 $(x, y) ∈ C^2$ as well as the nearest `alarming` power node's coordinates $(x^\dagger, y^\dagger) ∈ C^2$. 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 $A = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ with $|A| = 10$. 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 $T ∈ N$ and every system state $s_t ∈ S$ with $S = (C^2 → M) × C^2$ 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 $\gamma : \langle \mathcal{S} \rangle \rightarrow \mathbb{B}$ so that $\gamma(\langle \text{state_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T})$ 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 $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ defined via

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

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)

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 `state_of : C^2 → M` where $C = [0; 9] ⊂ N$ is the space of grid positions and $M = \{\text{working}, \text{alarming}, \text{defect}\}$ 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 $(x, y) ∈ C^2$ as well as the nearest `alarming` power node's coordinates $(x^\dagger, y^\dagger) ∈ C^2$. 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 $A = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ with $|A| = 10$. 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 $T ∈ N$ and every system state $s_t ∈ S$ with $S = (C^2 → M) × C^2$ 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)

When all positions are given, that data can easily be deducted when the environment generates the observation from the state.

- (ii) Give a goal predicate $\gamma : \langle \mathcal{S} \rangle \rightarrow \mathbb{B}$ so that $\gamma(\langle \text{state_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T})$ 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)

$$\begin{aligned}\gamma(\langle \text{state_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T}) \iff & \forall t : \forall X, Y : \\ & \text{state_of}_t((X, Y)) \neq \text{defect} \\ & \wedge \text{state_of}_t((X, Y)) = \text{alarming} \implies \\ & \exists t' : t' - t \leq 5 \wedge \text{state_of}_{t'}((X, Y)) = \text{working}\end{aligned}$$

- (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 $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ defined via

$$R((\text{state_of}_t, (x_t, y_t)), a_t, (\text{state_of}_{t+1}, (x_{t+1}, y_{t+1}))) = \begin{cases} +10 & \text{if } a_t = \text{repair} \text{ and } \text{state_of}_t((x_t, y_t)) = \text{defect}, \\ +5 & \text{if } a_t = \text{repair} \text{ and } \text{state_of}_t((x_t, y_t)) = \text{alarming}, \\ -3 & \text{else if } \exists X, Y : \text{state_of}_t((X, Y)) \in \{\text{defect}, \text{alarming}\}, \\ 0 & \text{otherwise.} \end{cases}$$

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)

Since the *RepairAgent* is not punished for waiting around, it will gain a higher reward when it lets nodes go **defect** and only then repairs them. This is likely to allow more time steps to pass, even when the agent *could* have acted quicker. For an improved reward function R , see above.

2 Agents and Goals

18pts

Scenario 1. We consider a multi-agent system made up of $n = 20$ cows and a mowing robot called *MowBot*. All these 21 agents live on an infinite meadow which we model as an infinite 2d plane of positions $l \in \mathbb{R}^2$. The agent $G^{[0]}$ is the *MowBot* and all other agents $G^{[1]}, \dots, G^{[20]}$ are cows.

The evolution of the multi-agent system is encoded as a series of states $\langle s_t \rangle_t$ of unspecified length for states $s_t \in \mathcal{S}$ at time t , where $\mathcal{S} = \mathbb{N} \times (\mathbb{R}^2)^{21}$ is the (in total 43-dimensional) state space, as well as series of joint observations $\langle o_t \rangle_t = \langle (o_t^{[0]}, \dots, o_t^{[20]}) \rangle_t$ made by the agents and joint actions $\langle a_t \rangle_t = \langle (a_t^{[0]}, \dots, a_t^{[20]}) \rangle_t$ executed by the agents.

Our system is fully observable. Thus, each agent $G^{[i]}$ makes observations

$$o_t^{[i]} = s_t = (t, l_t^{[0]}, \dots, l_t^{[20]}) \in \mathcal{S}$$

at each time step t where $l_t^{[i]} \in \mathbb{R}^2$ is the position of the agent $G^{[i]}$ at time $t \in \mathbb{N}$. Furthermore, any agent $G^{[i]}$ uses the identical action space $\mathcal{A}^{[i]} = \{\text{do_nothing}, \text{go_northeast}, \text{go_east}, \text{go_southeast}, \text{go_south}, \text{go_southwest}, \text{go_west}, \text{go_northwest}, \text{go_north}, \text{eat/mow}\}$ with $|\mathcal{A}^{[i]}| = 10$ where `do_nothing` has the agent not do anything, all actions starting with `go_` cause the agent to move for a certain length in the respective direction, and `eat/mow` causes any cow agent to eat the grass and any *MowBot* agent to mow the grass.

You can use a 2d geometric distance function $dist(l, l') = \sqrt{(x - x')^2 + (y - y')^2}$ where $l = (x, y) \in \mathbb{R}^2$ and $l' = (x', y') \in \mathbb{R}^2$.

- (i) Give a goal predicate $\gamma_1 : \langle \mathcal{S} \rangle \times \langle \mathcal{A} \rangle \rightarrow \mathbb{B}$ so that γ_1 holds iff the *MowBot* has never (at full time steps) been closer than a distance of 1 to any of the cows. (4pts)

$$\gamma_1 \left(\langle (t, l_t^{[0]}, \dots, l_t^{[20]}) \rangle_t, \langle (a_t^{[0]}, \dots, a_t^{[20]}) \rangle_t \right) \iff$$

- (ii) Let $(\mathcal{S}, \mathcal{A}, \mathbb{R}, P, R)$ be a Markov decision process (MDP, cf. Definition 3 in the appendix) with state space \mathcal{S} from Scenario 1, joint action space \mathcal{A} from Scenario 1, target space $\mathcal{T} = \mathbb{R}$, and a given transition probability function P . Give — from our *MowBot*'s perspective — a reward function R that both incentivizes to (a) fulfill γ_1 and (b) execute `mow/eat` as often as possible. (8pts)

- (iii) Now, in any run at time step $t = 26$ all cows suddenly change their policy to a specific new (from then on again fixed) policy.

Can R still be learned as the reward function of an MDP? Briefly state your reasoning. (3pts)

- (iv) Let $\gamma_2 : \langle \mathcal{S} \rangle \times \langle \mathcal{A} \rangle \rightarrow \mathbb{B}$ be goal predicate so that γ_2 holds iff the *MowBot* has never executed the action `eat/mow` within a radius of 1 from any point where any cow has ever executed `mow/eat`. Assume that the cows' policies are fixed.

Can the *MowBot* learn to fulfill γ_2 effectively for an MDP with the given state space \mathcal{S} for some reward function R' ? Briefly state your reasoning. (3pts)

2 Agents and Goals

18pts

Scenario 1. We consider a multi-agent system made up of $n = 20$ cows and a mowing robot called *MowBot*. All these 21 agents live on an infinite meadow which we model as an infinite 2d plane of positions $l \in \mathbb{R}^2$. The agent $G^{[0]}$ is the *MowBot* and all other agents $G^{[1]}, \dots, G^{[20]}$ are cows.

The evolution of the multi-agent system is encoded as a series of states $\langle s_t \rangle_t$ of unspecified length for states $s_t \in \mathcal{S}$ at time t , where $\mathcal{S} = \mathbb{N} \times (\mathbb{R}^2)^{21}$ is the (in total 43-dimensional) state space, as well as a series of joint observations $\langle o_t \rangle_t = \langle (o_t^{[0]}, \dots, o_t^{[20]}) \rangle_t$ made by the agents and a series of joint actions $\langle a_t \rangle_t = \langle (a_t^{[0]}, \dots, a_t^{[20]}) \rangle_t$ executed by the agents.

Our system is fully observable. Thus, each agent $G^{[i]}$ makes observations

$$o_t^{[i]} = s_t = (t, l_t^{[0]}, \dots, l_t^{[20]}) \in \mathcal{S}$$

at each time step t where $l_t^{[i]} \in \mathbb{R}^2$ is the position of the agent $G^{[i]}$ at time $t \in \mathbb{N}$. Furthermore, any agent $G^{[i]}$ uses the identical action space $\mathcal{A}^{[i]} = \{\text{do_nothing}, \text{go_northeast}, \text{go_east}, \text{go_southeast}, \text{go_south}, \text{go_southwest}, \text{go_west}, \text{go_northwest}, \text{go_north}, \text{eat/mow}\}$ with $|\mathcal{A}^{[i]}| = 10$ where `do_nothing` has the agent not do anything, all actions starting with `go_` cause the agent to move for a certain length in the respective direction, and `eat/mow` causes any cow agent to eat the grass and any *MowBot* agent to mow the grass.

You can use a 2d geometric distance function $dist(l, l') = \sqrt{(x - x')^2 + (y - y')^2}$ where $l = (x, y) \in \mathbb{R}^2$ and $l' = (x', y') \in \mathbb{R}^2$.

- (i) Give a goal predicate $\gamma_1 : \langle \mathcal{S} \rangle \times \langle \mathcal{A} \rangle \rightarrow \mathbb{B}$ so that γ_1 holds iff the *MowBot* has never (at full time steps) been closer than a distance of 1 to any of the cows. (4pts)

$$\gamma_1\left(\langle (t, l_t^{[0]}, \dots, l_t^{[20]}) \rangle_t, \langle (a_t^{[0]}, \dots, a_t^{[20]}) \rangle_t\right) \iff \forall t : \forall j \in [1; 20] \subset \mathbb{N} : dist(l_t^{[0]}, l_t^{[j]}) \geq 1$$

- (ii) Let $(\mathcal{S}, \mathcal{A}, \mathbb{R}, P, R)$ be a Markov decision process (MDP, cf. Definition 3 in the appendix) with state space \mathcal{S} from Scenario 1, joint action space \mathcal{A} from Scenario 1, target space $\mathcal{T} = \mathbb{R}$, and a given transition probability function P . Give — from our *MowBot*'s perspective — a reward function R that both incentivizes to (a) fulfill γ_1 and (b) execute **mow/eat** as often as possible. (8pts)

$$R(s_t, a_t, s_{t+1}) = \begin{cases} -10 & \text{if } \exists j \in [1; 20] : \text{dist}(l_{t+1}^{[0]}, l_{t+1}^{[j]}) < 1, \\ +1 & \text{else if } a_t = \text{eat/mow}, \\ 0 & \text{else.} \end{cases}$$

- (iii) Now, in any run at time step $t = 26$ all cows suddenly change their policy to a specific new (from then on again fixed) policy.

Can R still be learned as the reward function of an MDP? Briefly state your reasoning. (3pts)

Yes, because the time is part of the state space and time dependent changes can thus be recognized when learning.

- (iv) Let $\gamma_2 : \langle \mathcal{S} \rangle \times \langle \mathcal{A} \rangle \rightarrow \mathbb{B}$ be goal predicate so that γ_2 holds iff the *MowBot* has never executed the action **eat/mow** within a radius of 1 from any point where any cow has ever executed **mow/eat**. Assume that the cows' policies are fixed.

Can the *MowBot* learn to fulfill γ_2 effectively for an MDP with the given state space \mathcal{S} for some reward function R' ? Briefly state your reasoning. (3pts)

No, because past eaten patches are not encoded in the state, so if the *MowBot* implements an MDP, it can never be certain where cows might have previously eaten grass.

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 `state_of : C2 → M` where $C = [0; 9] ⊂ N$ is the space of grid positions and $M = \{\text{working}, \text{alarming}, \text{defect}\}$ 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 $(x, y) ∈ C^2$ as well as the nearest `alarming` power node's coordinates $(x^†, y^†) ∈ C^2$. 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 $A = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ with $|A| = 10$. 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 $T ∈ N$ and every system state $s_t ∈ S$ with $S = (C^2 → M) × C^2$ in accordance with all above specifications.

- (i) As stated above, the *RepairBot* can observe $(x^†, y^†)$, 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 $\gamma : \langle \mathcal{S} \rangle \rightarrow \mathbb{B}$ so that $\gamma(\langle \text{state_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T})$ 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 $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ defined via

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

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)

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 `state_of : C2 → M` where $C = [0; 9] ⊂ N$ is the space of grid positions and $M = \{\text{working}, \text{alarming}, \text{defect}\}$ 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 $(x, y) ∈ C^2$ as well as the nearest `alarming` power node's coordinates $(x^†, y^†) ∈ C^2$. 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 $A = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ with $|A| = 10$. 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 $T ∈ N$ and every system state $s_t ∈ S$ with $S = (C^2 → M) × C^2$ in accordance with all above specifications.

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

When all positions are given, that data can easily be deducted when the environment generates the observation from the state.

- (ii) Give a goal predicate $\gamma : \langle \mathcal{S} \rangle \rightarrow \mathbb{B}$ so that $\gamma(\langle \text{state_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T})$ 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)

$$\begin{aligned}\gamma(\langle \text{state_of}_t, (x_t, y_t) \rangle_{0 \leq t \leq T}) \iff & \forall t : \forall X, Y : \\ & \text{state_of}_t((X, Y)) \neq \text{defect} \\ & \wedge \text{state_of}_t((X, Y)) = \text{alarming} \implies \\ & \exists t' : t' - t \leq 5 \wedge \text{state_of}_{t'}((X, Y)) = \text{working}\end{aligned}$$

- (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 $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ defined via

$$R((\text{state_of}_t, (x_t, y_t)), a_t, (\text{state_of}_{t+1}, (x_{t+1}, y_{t+1}))) = \begin{cases} +10 & \text{if } a_t = \text{repair} \text{ and } \text{state_of}_t((x_t, y_t)) = \text{defect}, \\ +5 & \text{if } a_t = \text{repair} \text{ and } \text{state_of}_t((x_t, y_t)) = \text{alarming}, \\ -3 & \text{else if } \exists X, Y : \text{state_of}_t((X, Y)) \in \{\text{defect}, \text{alarming}\}, \\ 0 & \text{otherwise.} \end{cases}$$

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)

Since the *RepairAgent* is not punished for waiting around, it will gain a higher reward when it lets nodes go `defect` and only then repairs them. This is likely to allow more time steps to pass, even when the agent *could* have acted quicker. For an improved reward function R , see above.

2 Agents and Goals

20pts

Scenario 1. In the depths of LMU's research centers, brilliant scientists have created the *Dipus*, a computationally intelligent robotic creature that is thrice as efficient as its natural counterpart, the octopus, by condensing all its abilities into just two arms instead of eight. Similar to octopuses, *Dipuses* have a central brain located in their torso, which is responsible for overall decision-making, and a distributed nervous system; each arm contains large clusters of nerve cells that allow for sophisticated local control. Thus, the arms can perform many tasks (like exploring, grasping, and reacting to stimuli) practically independently of the central brain.

We thus define the *Dipus* as a multi-agent system made up of $n = 2$ arms, where we model both arms as agents $G^{[1]}, G^{[2]}$. The evolution of the multi-agent system is encoded as a series of states $\langle s_t \rangle_t$ of unspecified length for states $s_t \in \mathcal{S}$ at time t , where $\mathcal{S} = \{s' \in \mathcal{S}' \mid \text{valid}(s')\}$ is the state space, for a predicate `valid` and raw state space \mathcal{S}' described later. There is also a series of joint observations $\langle o_t \rangle_t = \langle (o_t^{[1]}, o_t^{[2]}) \rangle_t$ made by the agents and a series of joint actions $\langle a_t \rangle_t = \langle (a_t^{[1]}, a_t^{[2]}) \rangle_t$ executed by the agents.

The position of the *Dipus*'s torso is given via a center point $x \in \mathbb{R}^3$ in 3d space. Mechanically, each *Dipus* arm consists of 16 segments. For each segment i , the following information is given:

- the center point $x_i \in \mathbb{R}^3$ of the segment in 3d space;
- a flag $g_i \in \mathbb{B}$ so that $g_i = 1$ holds iff that segment's suckers are attached to something.

It is important that the state of the system is only *valid* if it fulfills a given predicate `valid` : $\mathcal{S} \rightarrow \mathbb{B}$ that checks if the center points of the segments and the torso have the right relative position so that the segment can bend accordingly without breaking and if no two center points are so close to each other that the segments would end up within each other. In short, `valid` checks for physical plausibility of a potential state.

Our environment also contains two objects for the *Dipus* to play with: a hammer and a nail. Since their shapes are fixed, both are fully defined by their center points $h, l \in \mathbb{R}^3$, respectively.

Furthermore, we are also given a predicate `touching` that is given two center points $a, b \in \mathbb{R}^3$ in 3d space so that `touching(a, b)` looks up the respective objects for the center points a, b (which each might be a torso, segment, hammer, or nail) and returns true iff their corresponding shapes are touching in the 3d world.

(i) Define the raw state space \mathcal{S}' , which includes potentially non-valid states, based on the description above. What are the dimensionalities of \mathcal{S}' and \mathcal{S} , counting every single dimension of 3d space positions separately? (7pts)

(ii) Our LMU research colleagues tell us that they want to use a predicate $\text{grasping}(a, b)$ that for two center points a, b checks if at least one of them belongs to a *Dipus* arm segment and for that arm segment i the suckers are attached, i.e., $g_i = 1$, and both center points a, b are **touching**. Note that to check all these properties, you are given a state $s \in \mathcal{S}$ as you defined above and possibly need to deconstruct the parts of s to access them appropriately. Define grasping . (9pts)

$$\text{grasping}(a, b) \iff$$

where $s =$

(iii) Assume a goal predicate γ that holds iff the *Dipus* has picked up both the hammer and the nail at a single point in time, one with each arm, and the hammer and nail have also been **touching** at that point in time. Using this goal predicate γ , one of the *Dipus*'s designers implemented the following reward function R to be maximized via

$$R(s_t, a_t, s_{t+1}) = \begin{cases} +100 & \text{if } \gamma(s_{t+1}), \\ 0 & \text{otherwise.} \end{cases}$$

Should we expect standard techniques of reinforcement learning to succeed in learning a behavior that is reliably fulfilling γ by optimizing for the reward function R ? Briefly discuss your reasoning. (4pts)

2 Agents and Goals

20pts

Scenario 1. In the depths of LMU's research centers, brilliant scientists have created the *Dipus*, a computationally intelligent robotic creature that is thrice as efficient as its natural counterpart, the octopus, by condensing all its abilities into just two arms instead of eight. Similar to octopuses, *Dipuses* have a central brain located in their torso, which is responsible for overall decision-making, and a distributed nervous system; each arm contains large clusters of nerve cells that allow for sophisticated local control. Thus, the arms can perform many tasks (like exploring, grasping, and reacting to stimuli) practically independently of the central brain.

We thus define the *Dipus* as a multi-agent system made up of $n = 2$ arms, where we model both arms as agents $G^{[1]}, G^{[2]}$. The evolution of the multi-agent system is encoded as a series of states $\langle s_t \rangle_t$ of unspecified length for states $s_t \in \mathcal{S}$ at time t , where $\mathcal{S} = \{s' \in \mathcal{S}' \mid \text{valid}(s')\}$ is the state space, for a predicate `valid` and raw state space \mathcal{S}' described later. There is also a series of joint observations $\langle o_t \rangle_t = \langle (o_t^{[1]}, o_t^{[2]}) \rangle_t$ made by the agents and a series of joint actions $\langle a_t \rangle_t = \langle (a_t^{[1]}, a_t^{[2]}) \rangle_t$ executed by the agents.

The position of the *Dipus*'s torso is given via a center point $x \in \mathbb{R}^3$ in 3d space. Mechanically, each *Dipus* arm consists of 16 segments. For each segment i , the following information is given:

- the center point $x_i \in \mathbb{R}^3$ of the segment in 3d space;
- a flag $g_i \in \mathbb{B}$ so that $g_i = 1$ holds iff that segment's suckers are attached to something.

It is important that the state of the system is only *valid* if it fulfills a given predicate `valid` : $\mathcal{S} \rightarrow \mathbb{B}$ that checks if the center points of the segments and the torso have the right relative position so that the segment can bend accordingly without breaking and if no two center points are so close to each other that the segments would end up within each other. In short, `valid` checks for physical plausibility of a potential state.

Our environment also contains two objects for the *Dipus* to play with: a hammer and a nail. Since their shapes are fixed, both are fully defined by their center points $h, l \in \mathbb{R}^3$, respectively.

Furthermore, we are also given a predicate `touching` that is given two center points $a, b \in \mathbb{R}^3$ in 3d space so that `touching(a, b)` looks up the respective objects for the center points a, b (which each might be a torso, segment, hammer, or nail) and returns true iff their corresponding shapes are touching in the 3d world.

- (i) Define the raw state space \mathcal{S}' , which includes potentially non-valid states, based on the description above. What are the dimensionalities of \mathcal{S}' and \mathcal{S} , counting every single dimension of 3d space positions separately? (7pts)

$$\mathcal{S}' = \mathbb{R}^3 \times (\mathbb{R}^3 \times \mathbb{B})^{16 \cdot 2} \times \mathbb{R}^3 \times \mathbb{R}^3$$

for $s \in \mathcal{S}'$ with $s = (\text{torso_position}, \text{segments}, \text{hammer_position}, \text{nail_position})$

$$\dim \mathcal{S}' = \dim \mathcal{S} = 3 + 4 \cdot 16 \cdot 2 + 3 + 3 = 137$$

- (ii) Our LMU research colleagues tell us that they want to use a predicate `grasping(a, b)` that for two center points a, b checks if at least one of them belongs to a *Dipus* arm segment and for that arm segment i the suckers are attached, i.e., $g_i = 1$, and both center points a, b are `touching`. Note that to check all these properties, you are given a state $s \in \mathcal{S}$ as you defined above and possibly need to deconstruct the parts of s to access them appropriately. Define `grasping`. (9pts)

$$\text{grasping}(a, b) \iff ((\exists i : a = x_i \wedge g_i) \vee (\exists i : b = x_i \wedge g_i)) \wedge \text{touching}(a, b)$$

where $s = (_, \langle x_i, g_i \rangle_{i \in \{0, \dots, 31\}}, _, _)$

(iii) Assume a goal predicate γ that holds iff the *Dipus* has picked up both the hammer and the nail at a single point in time, one with each arm, and the hammer and nail have also been **touching** at that point in time. Using this goal predicate γ , one of the *Dipus*'s designers implemented the following reward function R to be maximized via

$$R(s_t, a_t, s_{t+1}) = \begin{cases} +100 & \text{if } \gamma(s_{t+1}), \\ 0 & \text{otherwise.} \end{cases}$$

Should we expect standard techniques of reinforcement learning to succeed in learning a behavior that is reliably fulfilling γ by optimizing for the reward function R ? Briefly discuss your reasoning. (4pts)

No, because the reward is extremely sparse: we only get a positive reward feedback if the agents have by chance found a state fulfilling the full goal and we get no feedback at all along the way there.

2 Agents and Goals

17pts

Scenario 1. Beneath the grounds of LMU's most obscure research laboratory, a mole burrow, i.e., a network of tunnels dug by moles (German: "Maulwürfe"), has been discovered. To make sure that no top secret research has been compromised, the researchers immediately developed a group of three *MoleBots* that mimic real-life moles and can thus traverse the moles' network of tunnels.

Since the moles are seemingly fond of computer science, their burrow can be precisely modeled as a graph $G = (V, E)$ with vertices V and edges E ; here, each vertex $v \in V$ corresponds to an underground chamber that can hold arbitrarily many moles and each edge $e \in E$ corresponds to a tunnel connecting two chambers. Each chamber is uniquely identified by its coordinates in \mathbb{R}^3 . The burrow G may change due to actions of the moles or *MoleBots*.

Any mole or *MoleBot* can be at a single position within G at a given point in time. These positions include any chamber $v \in V$ and any tunnel $e \in E$.

While exploring the burrow, *MoleBots* can mark chambers they have visited. Any chamber can only be marked or unmarked, which can be seen and changed by any agent equally.

- (i) For now, assume a fixed burrow $G = (V, E)$ with chambers V and tunnels $E \subseteq V \times V$. Further assume that our system contains exactly 2 moles and 3 *MoleBots*. We ask the moles about their burrow and they tell us that the burrow's state s_t at time t is best described by $s_t \in \mathcal{S}$ for state space

$$\mathcal{S} = \mathbb{B}^{|V|} \times (V \cup E)^5.$$

Briefly explain what information is likely contained in the burrow's state space \mathcal{S} . (3pts)

(ii) From here on, we want to allow moles and *MoleBots* to dig new tunnels. When doing so, they have a position $d = (d_0, d_1) \in V \times \mathbb{R}^3$ while digging a tunnel from a chamber $d_0 \in V$ to a (possibly new) chamber $d_1 \in \mathbb{R}^3$. We want to adapt the definition of the state space \mathcal{S}^+ to include these new possibilities and a subterranean expert recommended that we use a raw state space

$$\mathcal{S}' = \wp(\mathbb{R}^3 \times \mathbb{B}) \times \wp(\mathbb{R}^3 \times \mathbb{R}^3) \times (\mathbb{R}^3 \cup \mathbb{R}^3 \times \mathbb{R}^3)^5$$

and a validity predicate **valid** by defining

$$\mathcal{S}^+ = \{s \in \mathcal{S}' \mid \text{valid}(s)\}.$$

Briefly explain why this formulation is helpful for modelling possible change in the burrow structure G . Define $\text{valid} : \mathcal{S}' \rightarrow \mathbb{B}$. (7pts)

$$\text{valid}\left(\{(v_0, m_0), \dots, (v_N, m_N)\}, \{(e_0, e'_0), \dots, (e_M, e'_M)\}, \langle p_0, \dots, p_4 \rangle\right) \iff$$

(iii) Investigative LMU staff wants to quickly use the *MoleBots* to explore the burrow. They propose two goals γ_1 and γ_2 with the following descriptions:

- γ_1 shall hold iff at time T every single chamber is marked.
- γ_2 shall hold iff at time T the *MoleBots* have visited every single chamber, i.e., any single chamber that currently exists has been the position of a *MoleBot* at some previous point in time.

Give some formal representations of γ_1 and γ_2 based on the setting of task (ii). (7pts)

$$\gamma_1(\langle s_0, \dots, s_T \rangle) \iff$$

$$\gamma_2(\langle s_0, \dots, s_T \rangle) \iff$$

2 Agents and Goals

17pts

Scenario 1. Beneath the grounds of LMU's most obscure research laboratory, a mole burrow, i.e., a network of tunnels dug by moles (German: "Maulwürfe"), has been discovered. To make sure that no top secret research has been compromised, the researchers immediately developed a group of three *MoleBots* that mimic real-life moles and can thus traverse the moles' network of tunnels.

Since the moles are seemingly fond of computer science, their burrow can be precisely modeled as a graph $G = (V, E)$ with vertices V and edges E ; here, each vertex $v \in V$ corresponds to an underground chamber that can hold arbitrarily many moles and each edge $e \in E$ corresponds to a tunnel connecting two chambers. Each chamber is uniquely identified by its coordinates in \mathbb{R}^3 . The burrow G may change due to actions of the moles or *MoleBots*.

Any mole or *MoleBot* can be at a single position within G at a given point in time. These positions include any chamber $v \in V$ and any tunnel $e \in E$.

While exploring the burrow, *MoleBots* can mark chambers they have visited. Any chamber can only be marked or unmarked, which can be seen and changed by any agent equally.

- (i) For now, assume a fixed burrow $G = (V, E)$ with chambers V and tunnels $E \subseteq V \times V$. Further assume that our system contains exactly 2 moles and 3 *MoleBots*. We ask the moles about their burrow and they tell us that the burrow's state s_t at time t is best described by $s_t \in \mathcal{S}$ for state space

$$\mathcal{S} = \mathbb{B}^{|V|} \times (V \cup E)^5.$$

Briefly explain what information is likely contained in the burrow's state space \mathcal{S} . (3pts)

The state space $\mathcal{S} = \mathbb{B}^{|V|} \times (V \cup E)^5$ consists of two components. The term $\mathbb{B}^{|V|}$ encodes which chambers are currently marked, with one Boolean value per chamber. The term $(V \cup E)^5$ specifies the positions of the 2 moles and 3 *MoleBots*, where each agent is located either in a chamber ($v \in V$) or in a tunnel ($e \in E$).

(ii) From here on, we want to allow moles and *MoleBots* to dig new tunnels. When doing so, they have a position $d = (d_0, d_1) \in V \times \mathbb{R}^3$ while digging a tunnel from a chamber $d_0 \in V$ to a (possibly new) chamber $d_1 \in \mathbb{R}^3$. We want to adapt the definition of the state space \mathcal{S}' to include these new possibilities and a subterranean expert recommended that we use a raw state space

$$\mathcal{S}' = \wp(\mathbb{R}^3 \times \mathbb{B}) \times \wp(\mathbb{R}^3 \times \mathbb{R}^3) \times (\mathbb{R}^3 \cup \mathbb{R}^3 \times \mathbb{R}^3)^5$$

and a validity predicate **valid** by defining

$$\mathcal{S}^+ = \{s \in \mathcal{S}' \mid \text{valid}(s)\}.$$

Briefly explain why this formulation is helpful for modelling possible change in the burrow structure G . Define $\text{valid} : \mathcal{S}' \rightarrow \mathbb{B}$. (7pts)

Since the graph can now change, we need to include chambers and tunnels in the state space. The check if they really match (for example, if a tunnel is connected to a chamber) is then performed dynamically by the predicate **valid**.

$$\begin{aligned} & \text{valid}\left(\{(v_0, m_0), \dots, (v_N, m_N)\}, \{(e_0, e'_0), \dots, (e_M, e'_M)\}, \langle p_0, \dots, p_4 \rangle\right) \\ \iff & \forall i \in \{0, \dots, 4\} : \exists j \in \{0, \dots, N\} : p_i = v_j \vee \exists j \in \{0, \dots, M\} : p_i = (e_j, e'_j) \\ & \wedge \forall j \in \{0, \dots, M\} : e_j \in V \end{aligned}$$

This checks if all positions p_i are either a chamber or a tunnel and also checks if any tunnel at least starts from a known chamber.

(iii) Investigative LMU staff wants to quickly use the *MoleBots* to explore the burrow. They propose two goals γ_1 and γ_2 with the following descriptions:

- γ_1 shall hold iff at time T every single chamber is marked.
- γ_2 shall hold iff at time T the *MoleBots* have visited every single chamber, i.e., any single chamber that currently exists has been the position of a *MoleBot* at some previous point in time.

Give some formal representations of γ_1 and γ_2 based on the setting of task (ii). (7pts)

$$\begin{aligned}\gamma_1(\langle s_0, \dots, s_T \rangle) &\iff \forall(c_0, c_1, c_2, m) \in V_T : m \\ \gamma_2(\langle s_0, \dots, s_T \rangle) &\iff \forall v \in V_T : \exists t \leq T : \exists m \in \{2, 3, 4\} : p_{m,t} = v \\ \text{where } s_t &= (V_t, E_t, \langle p_{0,t}, \dots, p_{4,t} \rangle)\end{aligned}$$

Feedback Aufgabe 2

2022-Exam-Coln.pdf

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf

2025-MockExam1-Coln.pdf

2025-MockExam2-Coln.pdf

Gesamtpunkte (ueber alle Aufgaben)

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 $\phi : \mathcal{L} \rightarrow \mathcal{N}$ 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 $\text{dist} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}$ to compute the Euclidean distance between two points in \mathcal{L} , i.e., $\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

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

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

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 $\psi : \mathcal{L} \rightarrow \mathbb{R}$ with

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

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

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

and a squirrel's calmness via

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

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

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

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

$$\text{happy}(x) = \text{OR}\left(\text{calm}(x), \text{NOT}(\text{quite}(\text{hungry}(x)))\right)$$

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

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 $\phi : \mathcal{L} \rightarrow \mathcal{N}$ 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 $\text{dist} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}$ to compute the Euclidean distance between two points in \mathcal{L} , i.e., $\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

$$\text{neighbors}(x) = \{x' \in \mathcal{L} : \text{dist}(x, x') \leq 1\}$$

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

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

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

Most of the solution landscape is a barren plateau without structure, i.e., it is very likely that all candidates simulated annealing can generate always have the same target value. Simulated annealing thus has no indication where to evolve to.

(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 $\psi : \mathcal{L} \rightarrow \mathbb{R}$ with

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

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)

The granularity of the search, i.e., range of generated candidates, is given by the *neighbors* function and does not change over time. On a scale within the *neighbors* range, our simulated annealing degenerates to random search.

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

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

and a squirrel's calmness via

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

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

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

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

$$\text{happy}(x) = \text{OR}\left(\text{calm}(x), \text{NOT}(\text{quite}(\text{hungry}(x)))\right)$$

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

$$\text{hungry}((42, 42)) = 1 - \frac{5}{10} = 0.5$$

$$\text{quite}(\text{hungry}((42, 42))) = \min(1, 1) = 1$$

$$\text{NOT}(\text{quite}(\text{hungry}((42, 42)))) = 1 - 1 = 0$$

$$\text{calm}((42, 42)) = \frac{995}{1000} = 0.995$$

$$\begin{aligned} \text{happy}((42, 42)) &= \text{OR}\left(\text{calm}((42, 42)), \text{NOT}(\text{quite}(\text{hungry}((42, 42))))\right) \\ &= \max(0.995, 0) = 0.995 \end{aligned}$$

The global maximum is (naturally) at *MrRogers'* location $(45, 46)$, where $\text{calm}((45, 46)) = 1$ and thus $\text{happy}((45, 46)) = 1$.

3 Optimization and Search

10pts

Assume that a beaver's policy for putting logs into dams is encoded by a parameter vector $\theta \in \Theta = \{A, C, G, T\}^5$ where A, C, G, T are arbitrary fixed symbols. We are given a fitness function $\phi : \Theta \rightarrow \mathbb{R}$ so that $\phi(\theta)$ encodes the time it takes a beaver with policy π_θ to put a log into a dam successfully, i.e., lower values of $\phi(\theta)$ are better.

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

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

where *selection* and *variation* are possibly randomized or non-deterministic functions so that $|\text{selection}(X)| \leq |X|$ and $|\text{variation}(X)| \geq |X|$ and $|\text{selection}(X_t \cup \text{variation}(X_t))| = |X|$.

(i) We initialize a population $X \subseteq \Theta$ with population size $|X| = 10$. A nice variation function should

- construct 5 new individuals based on random candidates from the original population X ,
- not always generate completely new individuals, but have them based on the given population X , and
- be able to reach every point in the search space through iterated application.

Give a complete definition for a nice variation function. (6pts)

(ii) Consider the following two definitions of a *selection* function:

$$\begin{aligned} \text{selection}_1(X) &= \begin{cases} \text{selection}_1(X \setminus \arg \max_{x \in X} \phi(x)) & \text{if } |X| > 10, \\ X & \text{otherwise,} \end{cases} \\ \text{selection}_2(X) &= \begin{cases} \text{selection}_2(X \setminus x) & \text{for } x \sim X \text{ if } |X| > 10, \\ X & \text{otherwise.} \end{cases} \end{aligned}$$

Briefly explain why both functions fulfill the definition for a *selection* function in Algorithm 1. Which of these two functions exerts the stronger selection pressure? Which of these two functions is more usable in the context of optimization? State your argument. (4pts)

3 Optimization and Search

10pts

Assume that a beaver's policy for putting logs into dams is encoded by a parameter vector $\theta \in \Theta = \{A, C, G, T\}^5$ where A, C, G, T are arbitrary fixed symbols. We are given a fitness function $\phi : \Theta \rightarrow \mathbb{R}$ so that $\phi(\theta)$ encodes the time it takes a beaver with policy π_θ to put a log into a dam successfully, i.e., lower values of $\phi(\theta)$ are better.

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

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

where *selection* and *variation* are possibly randomized or non-deterministic functions so that $|\text{selection}(X)| \leq |X|$ and $|\text{variation}(X)| \geq |X|$ and $|\text{selection}(X_t \cup \text{variation}(X_t))| = |X|$.

(i) We initialize a population $X \subseteq \Theta$ with population size $|X| = 10$. A nice variation function should

- construct 5 new individuals based on random candidates from the original population X ,
- not always generate completely new individuals, but have them based on the given population X , and
- be able to reach every point in the search space through iterated application.

Give a complete definition for a nice variation function. (6pts)

$$\begin{aligned} \text{variation}(\emptyset) &= \emptyset \\ \text{variation}(X) &= \{\text{mutate}(x)\} \cup \text{variation}(X \setminus \{x, x'\}) \\ \text{where } X &\neq \emptyset \\ \text{and } x, x' &\sim X \\ \text{with } \text{mutate}(x) &= (x'_0, x'_1, x'_2, x'_3, x'_4) \\ \text{where } x'_i &= \begin{cases} g \sim \{A, C, G, T\} & \text{if } i = j \\ x_i & \text{otherwise} \end{cases} \\ \text{and } j &\sim \{0, \dots, 4\} \end{aligned}$$

(ii) Consider the following two definitions of a *selection* function:

$$\begin{aligned} \text{selection}_1(X) &= \begin{cases} \text{selection}_1(X \setminus \arg \max_{x \in X} \phi(x)) & \text{if } |X| > 10, \\ X & \text{otherwise,} \end{cases} \\ \text{selection}_2(X) &= \begin{cases} \text{selection}_2(X \setminus x) & \text{for } x \sim X \text{ if } |X| > 10, \\ X & \text{otherwise.} \end{cases} \end{aligned}$$

Briefly explain why both functions fulfill the definition for a *selection* function in Algorithm 1. Which of these two functions exerts the stronger selection pressure? Which of these two functions is more usable in the context of optimization? State your argument. (4pts)

Both functions reduce the given population (recursively) until $|X| = 10$ is reached. selection_1 exerts the stronger selection pressure as selection_2 exerts no directed pressure at all. Thus, only selection_1 is usable for optimization.

3 Fuzzy Logic and Optimization 20pts

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 $\text{state_of} : \mathcal{C}^2 \rightarrow \mathcal{M}$ where $\mathcal{C} = [0; 9] \subset \mathbb{N}$ and $\mathcal{M} = \{\text{working}, \text{alarming}, \text{defect}\}$. A *RepairBot* travels through that grid by performing actions from the action space $\mathcal{A} = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ 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

$$\text{happy}(p) = \text{OR}(\text{very}(\text{central}(p)), \text{cared}(p))$$

where

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

only depends on the node's position and

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

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 and NOT are the commonly known fuzzy logic functions and *very* is given via

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

Assume that the *RepairBot* follows a policy that makes it stand still at node position $(4, 2)$ infinitely. Compute $\text{happy}((4, 2))$ and $\text{happy}((5, 5))$ 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 $\text{mutate} : \mathcal{A}^{1000} \rightarrow \mathcal{A}^{1000}$ 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 `random` : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ by calling `random(i, j)`, which returns a random integer within $[i; j] \subset \mathbb{N}$ according to a uniform distribution.

3 Fuzzy Logic and Optimization 20pts

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 $\text{state_of} : \mathcal{C}^2 \rightarrow \mathcal{M}$ where $\mathcal{C} = [0; 9] \subset \mathbb{N}$ and $\mathcal{M} = \{\text{working}, \text{alarming}, \text{defect}\}$. A *RepairBot* travels through that grid by performing actions from the action space $\mathcal{A} = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ 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

$$\text{happy}(p) = \text{OR}(\text{very}(\text{central}(p)), \text{cared}(p))$$

where

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

only depends on the node's position and

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

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 and NOT are the commonly known fuzzy logic functions and *very* is given via

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

Assume that the *RepairBot* follows a policy that makes it stand still at node position $(4, 2)$ infinitely. Compute $\text{happy}((4, 2))$ and $\text{happy}((5, 5))$ 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)

$$\text{central}((4, 2)) = \frac{10 - 1 - 3}{10} = 0.6$$
$$\text{very}(\text{central}((4, 2))) = 0.6^2 = 0.36$$

$$\text{cared}((4, 2)) = \frac{100}{100} = 1$$
$$\text{happy}((4, 2)) = \max(0.36, 1) = 1$$

$$\text{central}((5, 5)) = \frac{10 - 0 - 0}{10} = 1$$
$$\text{very}(\text{central}((5, 5))) = 1^2 = 1$$
$$\text{cared}((5, 5)) = \frac{0}{100} = 0$$
$$\text{happy}((5, 5)) = \max(1, 0) = 1$$

Both these nodes are global optima and no other node can be, because the *central* part of the OR is clearly centered around (5,5) and the *cared* part is only non-zero for this one node.

(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 $\text{mutate} : \mathcal{A}^{1000} \rightarrow \mathcal{A}^{1000}$ 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 `random` : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ by calling `random(i, j)`, which returns a random integer within $[i; j] \subset \mathbb{N}$ according to a uniform distribution.

$$\text{mutate}(\theta) = \theta'$$

$$\text{where for all } i \in [0, 1000) : \theta'_i = \begin{cases} a & \text{if } i = r, \\ \theta_i & \text{otherwise,} \end{cases}$$

where $a \sim \mathcal{A} \setminus \{\theta_r\}$
and $r = \text{random}(0, 1000)$

3 Fuzzy Logic

10pts

To evaluate the impact of the *MowBot* in Scenario 1 on the lives of our cows, we performed a field study and measured a cow's happiness at various positions $(x, y) \in \mathbb{R}^2$ on the infinite meadow. For our experiments, we firmly placed the *MowBot* at position $(3, -1)$ and never let it change that position.

From that study, we found that a cow's happiness can be described by the following formula in fuzzy logic:

$$\begin{aligned} \text{happy}((x, y)) &= \text{AND}(\text{green}((x, y)), \text{NOT}(\text{mowbot_nearby}((x, y)))) \\ \text{where } \text{green}((x, y)) &= \begin{cases} 1.0 & \text{if } |x| < 3 \text{ and } |y| < 3, \\ 0.8 & \text{otherwise,} \end{cases} \\ \text{and } \text{mowbot_nearby}((x, y)) &= 0.5 \cdot \max \{2 - \text{dist}((x, y), (3, -1)), 0\}. \end{aligned}$$

- (i) What is the happiness of a cow at position $(3, 0)$? Show your computation steps. (4pts)

- (ii) We now consider that cows can walk one step in any direction by adding one of the eight vectors $\{(+1, +1), (+1, 0), (+1, -1), (0, +1), (0, -1), (-1, +1), (-1, 0), (-1, -1)\}$ to their position. What is the most drastic increase in happiness a cow can achieve with one step? What is one possible corresponding start position and one possible corresponding end position of such a drastic step? (6pts)

3 Fuzzy Logic

10pts

To evaluate the impact of the *MowBot* in Scenario 1 on the lives of our cows, we performed a field study and measured a cow's happiness at various positions $(x, y) \in \mathbb{R}^2$ on the infinite meadow. For our experiments, we firmly placed the *MowBot* at position $(3, -1)$ and never let it change that position.

From that study, we found that a cow's happiness can be described by the following formula in fuzzy logic:

$$\begin{aligned} \text{happy}((x, y)) &= \text{AND}(\text{green}((x, y)), \text{NOT}(\text{mowbot_nearby}((x, y)))) \\ \text{where } \text{green}((x, y)) &= \begin{cases} 1.0 & \text{if } |x| < 3 \text{ and } |y| < 3, \\ 0.8 & \text{otherwise,} \end{cases} \\ \text{and } \text{mowbot_nearby}((x, y)) &= 0.5 \cdot \max\{2 - \text{dist}((x, y), (3, -1)), 0\}. \end{aligned}$$

- (i) What is the happiness of a cow at position $(3, 0)$? Show your computation steps. (4pts)

$$\begin{aligned} \text{green}((3, 0)) &= 0.8 \\ \text{mowbot_nearby}((3, 0)) &= 0.5 \cdot \max(2 - 1, 0) = 0.5 \\ \text{NOT}(\text{mowbot_nearby}((3, 0))) &= 1 - 0.5 = 0.5 \\ \text{happy}((3, 0)) &= \min(0.8, 0.5) = 0.5 \end{aligned}$$

- (ii) We now consider that cows can walk one step in any direction by adding one of the eight vectors $\{(+1, +1), (+1, 0), (+1, -1), (0, +1), (0, -1), (-1, +1), (-1, 0), (-1, -1)\}$ to their position. What is the most drastic increase in happiness a cow can achieve with one step? What is one possible corresponding start position and one possible corresponding end position of such a drastic step? (6pts)

Since it was not directly specified, depending on the following assumptions:

- cows and the *MowBot* can stand on the same position: Maximum happiness increase $+0.7$, from *MowBot* $(3, -1)$ (happiness 0.0) to e.g. $(2, -2)$ 0.7, or
- cows and the *MowBot* cannot stand on the same position: Maximum happiness increase $+0.5$ from close position e.g., position $(3, 0)$ with (happiness 0.5) moving further away to e.g., position $(2, 1)$ with (happiness 1.0)

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 $\text{state_of} : \mathcal{C}^2 \rightarrow \mathcal{M}$ where $\mathcal{C} = [0; 9] \subset \mathbb{N}$ and $\mathcal{M} = \{\text{working}, \text{alarming}, \text{defect}\}$. A *RepairBot* travels through that grid by performing actions from the action space $\mathcal{A} = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ 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

$$\text{happy}(p) = \text{OR}(\text{very}(\text{central}(p)), \text{cared}(p))$$

where

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

only depends on the node's position and

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

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

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

Assume that the *RepairBot* follows a policy that makes it stand still at node position $(4, 2)$ infinitely. Compute $\text{happy}((4, 2))$ and $\text{happy}((5, 5))$ 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 $\text{mutate} : \mathcal{A}^{1000} \rightarrow \mathcal{A}^{1000}$ 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 `random` : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ by calling `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 $x = \langle d_1, \dots, d_n \rangle$ and for all i , $1 \leq i \leq n$, it holds that $d_i \in \mathcal{D}$, as well as a function `ask : $\mathcal{D}^+ \rightarrow \mathcal{D}^+$` 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?

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 $\text{state_of} : \mathcal{C}^2 \rightarrow \mathcal{M}$ where $\mathcal{C} = [0; 9] \subset \mathbb{N}$ and $\mathcal{M} = \{\text{working}, \text{alarming}, \text{defect}\}$. A *RepairBot* travels through that grid by performing actions from the action space $\mathcal{A} = \{\text{move_northwest}, \text{move_north}, \dots, \text{move_southeast}, \text{do_nothing}, \text{repair}\}$ 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

$$\text{happy}(p) = \text{OR}(\text{very}(\text{central}(p)), \text{cared}(p))$$

where

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

only depends on the node's position and

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

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

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

Assume that the *RepairBot* follows a policy that makes it stand still at node position $(4, 2)$ infinitely. Compute $\text{happy}((4, 2))$ and $\text{happy}((5, 5))$ 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)

$$\text{central}((4, 2)) = \frac{10 - 1 - 3}{10} = 0.6$$
$$\text{very}(\text{central}((4, 2))) = 0.6^2 = 0.36$$

$$\text{cared}((4, 2)) = \frac{100}{100} = 1$$
$$\text{happy}((4, 2)) = \max(0.36, 1) = 1$$

$$\text{central}((5, 5)) = \frac{10 - 0 - 0}{10} = 1$$
$$\text{very}(\text{central}((5, 5))) = 1^2 = 1$$
$$\text{cared}((5, 5)) = \frac{0}{100} = 0$$
$$\text{happy}((5, 5)) = \max(1, 0) = 1$$

Both these nodes are global optima and no other node can be, because the *central* part of the OR is clearly centered around (5,5) and the *cared* part is only non-zero for this one node.

(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 $\text{mutate} : \mathcal{A}^{1000} \rightarrow \mathcal{A}^{1000}$ 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 `random` : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ by calling `random(i, j)`, which returns a random integer within $[i; j) \subset \mathbb{N}$ according to a uniform distribution.

$$\begin{aligned}\text{mutate}(\theta) &= \theta' \\ \text{where for all } i \in [0, 1000) : \theta'_i &= \begin{cases} a & \text{if } i = r, \\ \theta_i & \text{otherwise,} \end{cases} \\ &\text{where } a \sim \mathcal{A} \setminus \{\theta_r\} \\ &\text{and } r = \text{random}(0, 1000)\end{aligned}$$

(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 $x = \langle d_1, \dots, d_n \rangle$ and for all i , $1 \leq i \leq n$, it holds that $d_i \in \mathcal{D}$, as well as a function `ask` : $\mathcal{D}^+ \rightarrow \mathcal{D}^+$ 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 a recombination function, because it takes two individuals and returns one.

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

As a fitness/target/evaluation function, because it maps individuals to values.

- 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?

Yes, we force words like "Think", "following", etc. into every recombined child.

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

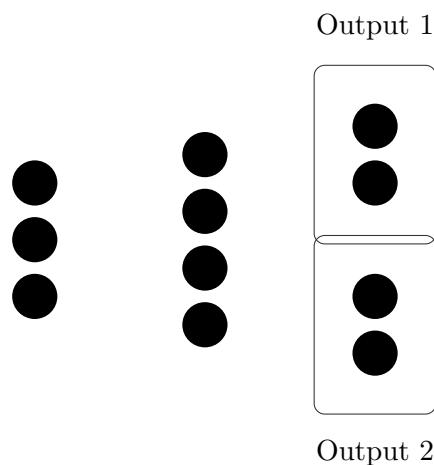
Individuals become longer and longer because our recombination just adds the parents together (plus some glue).

3 Optimization

10pts

We are now tasked to program the *Dipus* from Scenario 1. Obviously, we use a neural network to implement the *Dipus*'s behavior so that we can train the *Dipus* for best performance. For a first prototype, we choose a *centralized* architecture where both arms' actions $a^{[1]}$ and $a^{[2]}$ are the output of a single neural network \mathcal{N} . Furthermore, we assume a shared observation space, i.e., $\mathcal{O}^{[1]} = \mathcal{O}^{[2]}$ so that we can share observations $o \in \mathbb{R}^3$ for both arm agents. To control the arms, we output both $a_1 \in \mathcal{A}^{[1]}$ and $a_2 \in \mathcal{A}^{[2]}$ with $\mathcal{A}^{[1]} = \mathcal{A}^{[2]} = \mathbb{R}^2$ concatenated into one vector $a = a^{[1]} \uparrow a^{[2]} \in \mathbb{R}^4$.

For our neural network $\mathcal{N} : \mathbb{R}^3 \rightarrow \mathbb{R}^4$ we choose one hidden layer with 4 neurons, resulting in the architecture illustrated below.



- (i) Calculate how many edges the neural network \mathcal{N} has, i.e., calculate $|\overline{\mathcal{N}}|$. (3pts)

- (ii) Very briefly discuss pros and cons of having a centralized, “shared brain” architecture for the *Dipus*. Why might this approach be advantageous, compared to modeling the two-agent system as two completely independent agents? (4pts)

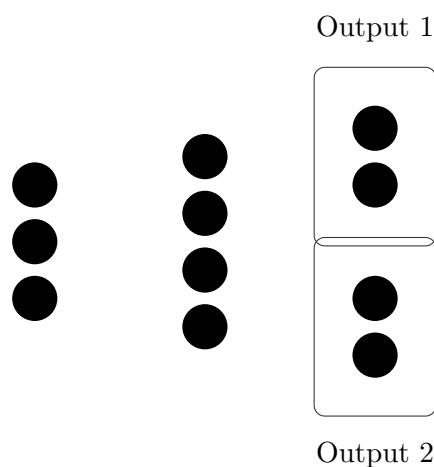
(iii) Assume that we are given two different reward functions $R^{[1]}$ and $R^{[2]}$ to train the *Dipus* arms $G^{[1]}$ and $G^{[2]}$, respectively. Using the architecture of the neural network \mathcal{N} above, we thus use the reward from $R^{[1]}$ to compute the gradient starting from the first two outputs of \mathcal{N} and we use the reward from $R^{[2]}$ to compute the gradient starting from the last two outputs of \mathcal{N} . Assuming backpropagation still computes the gradient throughout the whole network \mathcal{N} , how well should we expect this setup to work? Briefly explain your reasoning. (3pts)

3 Optimization

10pts

We are now tasked to program the *Dipus* from Scenario 1. Obviously, we use a neural network to implement the *Dipus*'s behavior so that we can train the *Dipus* for best performance. For a first prototype, we choose a *centralized* architecture where both arms' actions $a^{[1]}$ and $a^{[2]}$ are the output of a single neural network \mathcal{N} . Furthermore, we assume a shared observation space, i.e., $\mathcal{O}^{[1]} = \mathcal{O}^{[2]}$ so that we can share observations $o \in \mathbb{R}^3$ for both arm agents. To control the arms, we output both $a_1 \in \mathcal{A}^{[1]}$ and $a_2 \in \mathcal{A}^{[2]}$ with $\mathcal{A}^{[1]} = \mathcal{A}^{[2]} = \mathbb{R}^2$ concatenated into one vector $a = a^{[1]} + a^{[2]} \in \mathbb{R}^4$.

For our neural network $\mathcal{N} : \mathbb{R}^3 \rightarrow \mathbb{R}^4$ we choose one hidden layer with 4 neurons, resulting in the architecture illustrated below.



- (i) Calculate how many edges the neural network \mathcal{N} has, i.e., calculate $|\overline{\mathcal{N}}|$. (3pts)

$$|\overline{\mathcal{N}}| = 3 \times 4 + 4 \times 4 = 28$$

- (ii) Very briefly discuss pros and cons of having a centralized, “shared brain” architecture for the *Dipus*. Why might this approach be advantageous, compared to modeling the two-agent system as two completely independent agents? (4pts)

Advantages are in this case increased computational efficiency, as the whole system can produce actions with one single forward pass and use shared observations with a shared precomputed informational common ground. This also allows for the sharing of training experiences as, e.g., general skills (e.g. motor skills) only have to be learned once and can be retained in the shared brain independent of the arm's specialization.

Disadvantages on the other hand are increased complexity in training separate skills (cf. task (iii)) or the fact that common skills might not be desirable for both arm's (hindering specialization), but are still occupying network capacity.

Thus, compared to individual networks, the shared network architecture generally allows for better coordinated actions and more efficient (but not necessarily simpler) training.

(iii) Assume that we are given two different reward functions $R^{[1]}$ and $R^{[2]}$ to train the *Dipus* arms $G^{[1]}$ and $G^{[2]}$, respectively. Using the architecture of the neural network \mathcal{N} above, we thus use the reward from $R^{[1]}$ to compute the gradient starting from the first two outputs of \mathcal{N} and we use the reward from $R^{[2]}$ to compute the gradient starting from the last two outputs of \mathcal{N} . Assuming backpropagation still computes the gradient throughout the whole network \mathcal{N} , how well should we expect this setup to work? Briefly explain your reasoning. (3pts)

Due to two separate reward functions generally inducing different gradient signals onto not only the individual arms, but also the shared portion of the network, a naive approach most likely would not converge to any functional behavior, since gradient updates cancel out / might interfere with each other. In the case that the reward functions are different, but still aligned for a common goal, the weights may eventually converge, but are still considerably slower the training procedure.

3 Optimization

13pts

After some talks with LMU's security people, the moles from Scenario 1 agree to reduce their extensive burrow underneath LMU grounds. Recall from the previous task that a mole burrow is represented as a graph $G = (V, E)$ with chambers represented by vertices $v \in V$ and tunnels represented by edges $e \in E \subseteq V \times V$. The moles insist that they need to maintain their current set of entry points to the burrow $V^* \subseteq V$ as well as a disjunct set of important command chambers $V^\dagger \subseteq V$. We define $V? = V \setminus (V^* \cup V^\dagger)$. Note that the reduced burrow (including chambers V^* and V^\dagger) needs to be a connected graph still.

To help the moles find an ideal burrow configuration, we use simulated annealing. We represent possible burrow configurations as a bitmask $x \in \mathcal{X} = \mathbb{B}^N$ where $N = |V?| = |V| - |V^*| - |V^\dagger|$ is the number of chambers that may be up for reduction. Note that we assume that the set $V? = \{v_1, \dots, v_N\}$ can always be sorted and its elements numbered accordingly. Thus, any solution candidate $x = (b_1, \dots, b_N)$ tells us to close down all chambers $v_i \in V?$, $1 \leq i \leq N$, where $b_i = 0$. When a chamber is closed down, all tunnels going from and to it are closed down as well.

Luckily, cooperative LMU researchers could provide us with a predicate `is_connected` so that `is_connected(V, E)` returns true iff the graph consisting of all vertices V and all edges from E between elements of V is connected. LMU security provided us with a simple function to measure the size of a burrow defined via `size_of(V) = |V|`. As expert interior architects, the moles provided us with a preference function ϕ where $\phi(V, E) \in \mathbb{R}$ tells us how much the moles would like to live in the burrow defined by V and E (higher values are better). We can use the notation $V|_x = \{v_i \in V \mid b_i = 1\}$ for $x = (b_1, \dots, b_N)$, which defines $V|_x$ as the set of remaining elements of V after applying the bitmask given by x .

- (i) For this setup of simulating annealing give a computationally feasible and correctly typed `neighbors` function which only returns connected solution candidates. (8pts)

- (ii) The moles suggest that we optimize our multi-objective optimization problem based on the following combined target function

$$\tau(x) = \phi(V', E) - \text{size_of}(V') \quad \text{where } V' = V^* \cup V|_x \cup V^\dagger$$

to respect both the goal of size reduction as well as the goal of maximizing mole preference. Without any further knowledge of ϕ , what is the problem with this approach? Briefly explain how it could be fixed. (5pts)

3 Optimization

13pts

After some talks with LMU's security people, the moles from Scenario 1 agree to reduce their extensive burrow underneath LMU grounds. Recall from the previous task that a mole burrow is represented as a graph $G = (V, E)$ with chambers represented by vertices $v \in V$ and tunnels represented by edges $e \in E \subseteq V \times V$. The moles insist that they need to maintain their current set of entry points to the burrow $V^* \subseteq V$ as well as a disjunct set of important command chambers $V^\dagger \subseteq V$. We define $V? = V \setminus (V^* \cup V^\dagger)$. Note that the reduced burrow (including chambers V^* and V^\dagger) needs to be a connected graph still.

To help the moles find an ideal burrow configuration, we use simulated annealing. We represent possible burrow configurations as a bitmask $x \in \mathcal{X} = \mathbb{B}^N$ where $N = |V?| = |V| - |V^*| - |V^\dagger|$ is the number of chambers that may be up for reduction. Note that we assume that the set $V? = \{v_1, \dots, v_N\}$ can always be sorted and its elements numbered accordingly. Thus, any solution candidate $x = (b_1, \dots, b_N)$ tells us to close down all chambers $v_i \in V?$, $1 \leq i \leq N$, where $b_i = 0$. When a chamber is closed down, all tunnels going from and to it are closed down as well.

Luckily, cooperative LMU researchers could provide us with a predicate `is_connected` so that `is_connected(V, E)` returns true iff the graph consisting of all vertices V and all edges from E between elements of V is connected. LMU security provided us with a simple function to measure the size of a burrow defined via `size_of(V) = |V|`. As expert interior architects, the moles provided us with a preference function ϕ where $\phi(V, E) \in \mathbb{R}$ tells us how much the moles would like to live in the burrow defined by V and E (higher values are better). We can use the notation $V|_x = \{v_i \in V \mid b_i = 1\}$ for $x = (b_1, \dots, b_N)$, which defines $V|_x$ as the set of remaining elements of V after applying the bitmask given by x .

- (i) For this setup of simulating annealing give a computationally feasible and correctly typed `neighbors` function which only returns connected solution candidates. (8pts)

```
define neighbors(x) :  $\mathbb{B}^N \rightarrow \wp(\mathbb{B}^N)$ 
  set X =  $\emptyset$ 
  for 5 times do :                                # make set of some size
    set x' := x
    do                                         # do ... while
      sample i ~ {1, ..., N}
      set x'_i :=  $\neg x'_i$ 
      while  $\neg \text{is\_connected}(V?|_{x'}, V^* \cup V^\dagger, E)$  :
        set X := X  $\cup \{x'\}$ 
  return X
```

- (ii) The moles suggest that we optimize our multi-objective optimization problem based on the following combined target function

$$\tau(x) = \phi(V', E) - \text{size_of}(V') \quad \text{where } V' = V^* \cup V|_x \cup V^\dagger$$

to respect both the goal of size reduction as well as the goal of maximizing mole preference. Without any further knowledge of ϕ , what is the problem with this approach? Briefly explain how it could be fixed. (5pts)

When ϕ returns very large values it overcomes any goal of size reduction. We could use both goals in an alternating manner or simply normalize their values.

Feedback Aufgabe 3

2022-Exam-Coln.pdf

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf

2025-MockExam1-Coln.pdf

2025-MockExam2-Coln.pdf

Gesamtpunkte (ueber alle Aufgaben)

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*:

$$\begin{array}{cc} & \text{F} & \text{S} \\ \text{F} & \left(\begin{array}{cc} 0, 0 & 5, 4 \\ 4, 5 & 0, 0 \end{array} \right) \\ \text{S} & & \end{array}$$

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

(ii) Compute all 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

$$A(\text{affection}) = \overline{\text{go_first}. \text{give_affection}(x). A(\text{add}(\text{affection}, x))}$$
$$B(\text{affection}) = \overline{\text{go_second}. \text{give_affection}(x). B(\text{add}(\text{affection}, x))}$$

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

$$MrR \mid A(0) \mid B(0)$$

runs indefinitely and A and B accumulate *affection* according to the rules of *affection schedule*. (5pts)

Hint: The processes A and B remain fixed; MrR does not need 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)

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

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

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

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*:

F	S
F	(0, 0 5, 4)
S	(4, 5 0, 0)

- (i) Compute all pure Nash equilibria (cf. Definitions 6 and 9 in the appendix) for *affection schedule* by denoting all best responses. (4pts)

The best responses are....

F	S
F	(0, 0 <u>5</u> , <u>4</u>)
S	(<u>4</u> , <u>5</u> 0, 0)

Thus, the pure Nash equilibria are the joint strategies (F, S) and (S, F) .

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

$$\begin{aligned}(F, S) &\xrightarrow{\text{Pareto-dominates}} (F, F) \\ (F, S) &\xrightarrow{\text{Pareto-dominates}} (S, S) \\ (S, F) &\xrightarrow{\text{Pareto-dominates}} (F, F) \\ (S, F) &\xrightarrow{\text{Pareto-dominates}} (S, S)\end{aligned}$$

The non-dominated strategies and thus the joint strategies on the Pareto front are, by happenstance again, (F, S) and (S, F) .

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

$$A(\text{affection}) = \overline{\text{go_first}. \text{give_affection}(x)}. A(\text{add}(\text{affection}, x))$$
$$B(\text{affection}) = \overline{\text{go_second}. \text{give_affection}(x)}. B(\text{add}(\text{affection}, x))$$

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

$$MrR \mid A(0) \mid B(0)$$

runs indefinitely and A and B accumulate *affection* according to the rules of *affection schedule*. (5pts)

Hint: The processes A and B remain fixed; MrR does not need to work with any arbitrary processes besides exactly A and B as defined above.

$$MrR = \text{go_first}. \overline{\text{give_affection}}\langle 5 \rangle. \text{go_second}. \overline{\text{give_affection}}\langle 4 \rangle. MrR$$

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

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

$$\begin{aligned} &\rightarrow (A) \mid \bar{c}.0 \mid 0 \\ &\rightarrow A \mid \bar{c}.0 \end{aligned}$$

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

$$\begin{aligned} &\rightarrow X \mid \bar{b}.Z \mid !(a.X + b.Y) \\ &\rightarrow X \mid Y \mid Z \mid !(a.X + b.Y) \end{aligned}$$

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

$$\begin{aligned} &\rightarrow a.0 + (\bar{c}.0 \mid c.0) \\ &\rightarrow a.0 + 0 \\ &\rightarrow a.0 \end{aligned}$$

Note: The last step is optional since it does not follow directly from our axioms.

4 Reinforcement Learning

10pts

Recall the Vacuum World from the lecture. It consists of two rooms A, B , which can be dirty or clean. An agent is positioned in exactly one of these rooms and can execute one of two actions: switch the room (action *switch*) or vacuum the room (action *vacuum*) with the obvious result. If the agent chooses to vacuum, it receives a reward of +10 if the room it is in was dirty or -1 if it was not dirty. Other actions inflict no reward nor cost. The initial state of the Vacuum World is given as the agent being positioned in room A , room A being clean, and room B being dirty.

(i) Recall Definition 1 from the lecture (which is printed in this exam under task 2(ii)) and provide suitable definitions for

- the state space \mathcal{S} ,
- the action space \mathcal{A} ,
- the target space \mathcal{T} , and
- the reward function R .

You do not need to give the state transition probability function P . However, note that state transition is deterministic in this case. (5pts)

(ii) Consider the fixed policy π which executes the actions *vacuum*, *switch*, *vacuum*, *vacuum* in that order and then stops execution. Execute π and for each generated state s during the execution (including the initial state $s = s_0$) compute its value $V^\pi(s)$ given a fully deterministic state transition function P and $\gamma = 1$. To this end, recall the Bellman equation as given in the lecture:

Theorem 1 (Bellman equation). Let $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ be a Markov decision process. Let $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{T}$ 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 $\gamma \in [0; 1] \subseteq \mathbb{R}$ be a temporal discount factor.

The expected reward of a policy π being executed starting from state s is given via π 's value function

$$V^\pi(s) = R(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) \cdot V^\pi(s').$$

The value function of the optimal policy π^* is given via

$$V^{\pi^*}(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s'|s, a) \cdot V^{\pi^*}(s') \right).$$

(5pts)

4 Reinforcement Learning

10pts

Recall the Vacuum World from the lecture. It consists of two rooms A, B , which can be dirty or clean. An agent is positioned in exactly one of these rooms and can execute one of two actions: switch the room (action *switch*) or vacuum the room (action *vacuum*) with the obvious result. If the agent chooses to vacuum, it receives a reward of +10 if the room it is in was dirty or -1 if it was not dirty. Other actions inflict no reward nor cost. The initial state of the Vacuum World is given as the agent being positioned in room A , room A being clean, and room B being dirty.

(i) Recall Definition 1 from the lecture (which is printed in this exam under task 2(ii)) and provide suitable definitions for

- the state space \mathcal{S} ,
- the action space \mathcal{A} ,
- the target space \mathcal{T} , and
- the reward function R .

You do not need to give the state transition probability function P . However, note that state transition is deterministic in this case. (5pts)

$$\mathcal{S} = \{A, B\} \times \{\text{dirty}, \text{clean}\} \times \{\text{dirty}, \text{clean}\},$$

$$\mathcal{A} = \{\text{switch}, \text{vacuum}\},$$

$$\mathcal{T} = \mathbb{R},$$

$$R(s, a) = \begin{cases} +10 & \text{if } s \in \{(A, \text{dirty}, _), (B, _, \text{dirty})\} \text{ and } a = \text{vacuum}, \\ -1 & \text{if } s \in \{(A, \text{clean}, _), (B, _, \text{clean})\} \text{ and } a = \text{vacuum}, \\ 0 & \text{otherwise} \end{cases}$$

(ii) Consider the fixed policy π which executes the actions *vacuum*, *switch*, *vacuum*, *vacuum* in that order and then stops execution. Execute π and for each generated state s during the execution (including the initial state $s = s_0$) compute its value $V^\pi(s)$ given a fully deterministic state transition function P and $\gamma = 1$. To this end, recall the Bellman equation as given in the lecture:

Theorem 1 (Bellman equation). Let $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ be a Markov decision process. Let $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{T}$ 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 $\gamma \in [0; 1] \subseteq \mathbb{R}$ be a temporal discount factor.

The expected reward of a policy π being executed starting from state s is given via π 's value function

$$V^\pi(s) = R(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) \cdot V^\pi(s').$$

The value function of the optimal policy π^* is given via

$$V^{\pi^*}(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s'|s, a) \cdot V^{\pi^*}(s') \right).$$

(5pts)

$$s_0 = \{A, clean, dirty\}$$

$$s_1 = \{A, clean, dirty\}$$

$$s_2 = \{B, clean, dirty\}$$

$$s_3 = \{B, clean, clean\}$$

$$s_4 = \{B, clean, clean\}$$

$$V^\pi(s_4) = 0$$

$$V^\pi(s_3) = -1$$

$$V^\pi(s_2) = 9$$

$$V^\pi(s_1) = 9$$

$$V^\pi(s_0) = 8$$

4 Game Theory and Coordination

18pts

Consider a power grid in which all nodes can choose between two different strategies: Strategy E signals alarming as early as possible while strategy L signals alarming as late as possible. When all power nodes signal early, the *RepairBot* gets overwhelmed and, when all nodes signal late, the chance of the grid breaking down increases. Thus, when two power nodes need to decide on a strategy of how to attract the *RepairBot*, formally, they play the following two-player normal-form game we call *alarm signaling*:

$$\begin{array}{cc} & \begin{matrix} E & L \end{matrix} \\ \begin{matrix} E \\ L \end{matrix} & \begin{pmatrix} 2, 2 & 3, 1 \\ 1, 3 & 1, 1 \end{pmatrix} \end{array}$$

- (i) Show that (E, E) is the only pure Nash equilibrium (cf. Definitions 6 and 8 in the appendix) for *alarm signaling* by denoting all best responses. (4pts)
- (ii) Show that the strategy E is an evolutionary stable strategy (cf. Definition 9 in the Appendix) in a population of power nodes. (6pts)

(iii) 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:

$$\begin{aligned} PowerNode(name, state) = & \text{announce(bot_capability).} \\ & \overline{\text{introduce}}(name). \\ & (do_nothing.PowerNode(name, state) \\ & + repair.PowerNode(name, bot_capability)) \end{aligned}$$

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

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

always evaluates to the process

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

where *Bill(name)* is a named (black-box) process that cannot be evaluated any further for any *name* and can be called freely within your namespace. (8pts)

Hint: It is perfectly fine to repair a **working** *PowerNode* as well.

4 Game Theory and Coordination

18pts

Consider a power grid in which all nodes can choose between two different strategies: Strategy E signals alarming as early as possible while strategy L signals alarming as late as possible. When all power nodes signal early, the *RepairBot* gets overwhelmed and, when all nodes signal late, the chance of the grid breaking down increases. Thus, when two power nodes need to decide on a strategy of how to attract the *RepairBot*, formally, they play the following two-player normal-form game we call *alarm signaling*:

	E	L
E	(2, 2 3, 1)	
L	(1, 3 1, 1)	

- (i) Show that (E, E) is the only pure Nash equilibrium (cf. Definitions 6 and 8 in the appendix) for *alarm signaling* by denoting all best responses. (4pts)

The best responses are....

	E	L
E	(2, <u>2</u> <u>3</u> , 1)	
L	(1, <u>3</u> 1, 1)	

- (ii) Show that the strategy E is an evolutionary stable strategy (cf. Definition 9 in the Appendix) in a population of power nodes. (6pts)

We already know that $E \otimes E = (E, E)$ is a Nash equilibrium. It also holds that

$$\begin{aligned}\chi^{[1]}(E \otimes L) &> \chi^{[1]}(L \otimes L) \\ 3 &> 1\end{aligned}$$

so E is an evolutionary stable strategy.

(iii) 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:

$$\begin{aligned} PowerNode(name, state) = & \text{announce(bot_capability).} \\ & \overline{\text{introduce}}(name). \\ & (do_nothing.PowerNode(name, state) \\ & + repair.PowerNode(name, bot_capability)) \end{aligned}$$

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

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

always evaluates to the process

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

where *Bill(name)* is a named (black-box) process that cannot be evaluated any further for any *name* and can be called freely within your namespace. (8pts)

Hint: It is perfectly fine to repair a **working** *PowerNode* as well.

$$\begin{aligned} RepairBot = & \overline{\text{announce}}(\text{working}). \\ & \overline{\text{introduce}}(name). \\ & \overline{\text{repair}}. \\ & (RepairBot \mid Bill(name)) \end{aligned}$$

4 Optimization

14pts

Scenario 2. The policy $\pi : \mathcal{S} \rightarrow \mathcal{A}^{[-]}$ of a *math cow* is given by a real number $0 \leq u < 1$, i.e.,

$$\pi((t, -, \dots, -)) = \text{map}(\lfloor 10^t \cdot u \mod 10 \rfloor)$$

where state space \mathcal{S} and action space \mathcal{A} are given as in Scenario 1, $\lfloor w \rfloor$ for $w \in \mathbb{R}$ is the nearest integer from w rounded down, $w \mod 10$ for positive real numbers $w \in \mathbb{R}$ is given recursively via

$$w \mod 10 = \begin{cases} w & \text{if } w < 10, \\ w - 10 \mod 10 & \text{otherwise,} \end{cases}$$

and $\text{map} : \{0, \dots, 9\} \rightarrow \mathcal{A}^{[-]}$ is given exhaustively via

$$\begin{aligned} \text{map}(0) &= \text{do_nothing}, \\ \text{map}(1) &= \text{go_northeast}, \\ &\dots \\ \text{map}(8) &= \text{go_north}, \\ \text{map}(9) &= \text{eat/mow}. \end{aligned}$$

In short, the *math cow* policy encoding described in Scenario 2 uses a single real number u to encode a (potentially very long or even infinitely long) behavior by interpreting every single decimal digit of u as a subsequent action to be taken by a corresponding policy. We now give two examples if you need them:

- If a *math cow*'s policy π is given via the number $u = 0.199$, then that cow's action at time point $t = 1$ is given via

$$\begin{aligned} \pi((1, -, \dots, -)) &= \text{map}(\lfloor 10^1 \cdot 0.199 \mod 10 \rfloor) \\ &= \text{map}(\lfloor 1.99 \mod 10 \rfloor) \\ &= \text{map}(\lfloor 1.99 \rfloor) \\ &= \text{map}(1) \\ &= \text{go_northeast}. \end{aligned}$$

Analogously, $\pi((2, -, \dots, -)) = \text{eat/mow}$, then $\pi((3, -, \dots, -)) = \text{eat/mow}$, and then $\pi((t, -, \dots, -)) = \text{do_nothing}$ for all $t \geq 4$.

- If a *math cow*'s policy π is given via the number $u = \frac{8}{9} = 0.\overline{8} = 0.888\dots$, then that cow's action at time point $t = 1$ is given via

$$\begin{aligned} \pi((1, -, \dots, -)) &= \text{map}(\lfloor 10^1 \cdot 0.\overline{8} \mod 10 \rfloor) & (1) \\ &= \text{go_north}. & (2) \end{aligned}$$

This action is then also chosen for any other time point $t \geq 1$.

We now want to optimize one cow's policy for any given target function τ via *simulated annealing*, a special form of simulated annealing (cf. Algorithm 1 in the appendix) with a specific acceptance probability function

$$A(Q, Q', K) = \frac{1}{K}$$

for temperature schedule

$$\kappa(t) = t.$$

- (i) What is the effect of this definition for A and κ in *simulated annealing* compared to more typical or more general variants of simulated annealing? How does this effect vary with the quality of the newly generated solution candidate? (4pts)

(ii) A friendly *neighbors* function, through which we generate from a policy π a new policy $\pi' \sim \text{neighbors}(\pi)$, has the following properties:

1. On average, substantial parts of the information in π remain within π' .
2. π' always differs in at least one action within the first 1000 actions from π .
3. *neighbors* is symmetric in the sense that generating π' from π has the same chance as generating π from π' would have.
4. For a given π , the value of π' is non-deterministic.

Give a friendly *neighbors* function. Give a brief explanation why your function fulfills each of the four properties enumerated above. (10pts)

Hint: You can use policies π, π' and their corresponding real numbers u, u' interchangeably, without any need to type cast from π to u or vice versa.

Note: You can write $x \sim X$ to sample a (uniformly) random element x from a finite set X . You can also write $x = \text{random}()$ to sample a random real number x with $x \in [0; 1] \subset \mathbb{R}$.

4 Optimization

14pts

Scenario 2. The policy $\pi : \mathcal{S} \rightarrow \mathcal{A}^{[-]}$ of a *math cow* is given by a real number $0 \leq u < 1$, i.e.,

$$\pi((t, -, \dots, -)) = \text{map}(\lfloor 10^t \cdot u \mod 10 \rfloor)$$

where state space \mathcal{S} and action space \mathcal{A} are given as in Scenario 1, $\lfloor w \rfloor$ for $w \in \mathbb{R}$ is the nearest integer from w rounded down, $w \mod 10$ for positive real numbers $w \in \mathbb{R}$ is given recursively via

$$w \mod 10 = \begin{cases} w & \text{if } w < 10, \\ w - 10 \mod 10 & \text{otherwise,} \end{cases}$$

and $\text{map} : \{0, \dots, 9\} \rightarrow \mathcal{A}^{[-]}$ is given exhaustively via

$$\begin{aligned} \text{map}(0) &= \text{do_nothing}, \\ \text{map}(1) &= \text{go_northeast}, \\ &\dots \\ \text{map}(8) &= \text{go_north}, \\ \text{map}(9) &= \text{eat/mow}. \end{aligned}$$

In short, the *math cow* policy encoding described in Scenario 2 uses a single real number u to encode a (potentially very long or even infinitely long) behavior by interpreting every single decimal digit of u as a subsequent action to be taken by a corresponding policy. We now give two examples if you need them:

- If a *math cow*'s policy π is given via the number $u = 0.199$, then that cow's action at time point $t = 1$ is given via

$$\begin{aligned} \pi((1, -, \dots, -)) &= \text{map}(\lfloor 10^1 \cdot 0.199 \mod 10 \rfloor) \\ &= \text{map}(\lfloor 1.99 \mod 10 \rfloor) \\ &= \text{map}(\lfloor 1.99 \rfloor) \\ &= \text{map}(1) \\ &= \text{go_northeast}. \end{aligned}$$

Analogously, $\pi((2, -, \dots, -)) = \text{eat/mow}$, then $\pi((3, -, \dots, -)) = \text{eat/mow}$, and then $\pi((t, -, \dots, -)) = \text{do_nothing}$ for all $t \geq 4$.

- If a *math cow*'s policy π is given via the number $u = \frac{8}{9} = 0.\overline{8} = 0.888\dots$, then that cow's action at time point $t = 1$ is given via

$$\begin{aligned} \pi((1, -, \dots, -)) &= \text{map}(\lfloor 10^1 \cdot 0.\overline{8} \mod 10 \rfloor) & (1) \\ &= \text{go_north}. & (2) \end{aligned}$$

This action is then also chosen for any other time point $t \geq 1$.

We now want to optimize one cow's policy for any given target function τ via *simulated annealing*, a special form of simulated annealing (cf. Algorithm 1 in the appendix) with a specific acceptance probability function

$$A(Q, Q', K) = \frac{1}{K}$$

for temperature schedule

$$\kappa(t) = t.$$

- (i) What is the effect of this definition for A and κ in *simulated annealing* compared to more typical or more general variants of simulated annealing? How does this effect vary with the quality of the newly generated solution candidate? (4pts)

Here, the acceptance probability decreases with time independent of solution quality. That means that (in contrast to simulated annealing) acceptance probability does not vary with the quality of the solution candidates.

(ii) A friendly *neighbors* function, through which we generate from a policy π a new policy $\pi' \sim \text{neighbors}(\pi)$, has the following properties:

1. On average, substantial parts of the information in π remain within π' .
2. π' always differs in at least one action within the first 1000 actions from π .
3. *neighbors* is symmetric in the sense that generating π' from π has the same chance as generating π from π' would have.
4. For a given π , the value of π' is non-deterministic.

Give a friendly *neighbors* function. Give a brief explanation why your function fulfills each of the four properties enumerated above. (10pts)

Hint: You can use policies π, π' and their corresponding real numbers u, u' interchangeably, without any need to type cast from π to u or vice versa.

Note: You can write $x \sim X$ to sample a (uniformly) random element x from a finite set X . You can also write $x = \text{random}()$ to sample a random real number x with $x \in [0; 1] \subset \mathbb{R}$.

$$\begin{aligned} &\text{sample } t \sim [1; 1000] \subset \mathbb{N} \\ &\text{neighbors}(u) = \{u \pm 10^{-t}\} \end{aligned}$$

1. In most cases, only one digit of u' is changed from u , so the rest of the information remains.
2. At least one digit within the first 1000 (see t) of u' is changed from u .
3. Choosing the plus case from \pm in $\text{neighbors}(u)$ is reverted by sampling the same t and choosing the minus case starting from u' .
4. We sample t (and then again sample from the set $\text{neighbors}(u)$).

4 The π -Calculus

10pts

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

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

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

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

(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:

$$\begin{aligned} PowerNode(name, state) = & announce(bot_capability). \\ & \overline{introduce}(name). \\ & (do_nothing.PowerNode(name, state) \\ & + repair.PowerNode(name, bot_capability)) \end{aligned}$$

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

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

always evaluates to the process

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

where *Bill(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.

4 The π -Calculus

10pts

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

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

$$\begin{aligned} &\rightarrow b(x).\overline{c}\langle x \rangle.0 \mid \overline{b}\langle 42 \rangle.0 \mid c(z).S(z) \\ &\rightarrow \overline{c}\langle 42 \rangle.0 \mid 0 \mid c(z).S(z) \\ &\rightarrow 0 \mid 0 \mid S(42) \\ &\rightarrow S(42) \end{aligned}$$

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

$$\begin{aligned} &\rightarrow \overline{c}.\overline{b}.0 \mid (\overline{a}.A + \overline{c}.C) \mid A \mid !(a.A + b.B) \\ &\rightarrow \overline{c}.\overline{b}.0 \mid A \mid A \mid A \mid !(a.A + b.B) \end{aligned}$$

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

$$\begin{aligned} &\rightarrow a(x).b(y).S(x,y) + (A(Y) \mid 0) \\ &\rightarrow a(x).b(y).S(x,y) + A(Y) \end{aligned}$$

(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:

$$\begin{aligned} PowerNode(name, state) = & \overline{\text{announce}}(bot_capability). \\ & \overline{\text{introduce}}(name). \\ & (do_nothing.PowerNode(name, state) \\ & + repair.PowerNode(name, bot_capability)) \end{aligned}$$

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

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

can evaluate to the process

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

where *Bill(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.

$$\begin{aligned} RepairBot = & \overline{\text{announce}}(\text{working}). \\ & \overline{\text{introduce}}(name). \\ & \overline{\text{repair}}. \\ & (RepairBot \mid Bill(name)) \end{aligned}$$

4 Fuzzy Logic

15pts

Like octopuses, the *Dipus* from Scenario 1 — when threatened — can eject a cloud of ink to confuse predators. To accommodate LMU branding guidelines, this ink naturally needs to be a greenish color. The exact tone of greenness is then judged very critically, and only the most color-accurate of *Dipuses* are allowed to be released into the harsh nature of university mascots. LMU scientists sample the color of the ink by its *RGB* value, with $r, g, b \in [0, 255]$, and then decide the $\text{brand-conformity}(r, g, b)$.

Directly from LMU's head of color theory, we were told about the following formulae in fuzzy logic:

$$\text{brand-conformity}(r, g, b) = \text{AND}(\text{pretty}(\text{eccentric}(r, g)), \text{pretty}(\text{accurate}(g, b))),$$

$$\text{where } \text{pretty}(u) = \min\{2u, 1\},$$

$$\text{eccentric}(v, x) = \begin{cases} 0 & \text{if } 42 - v - x \geq 0, \\ 0.42 & \text{otherwise,} \end{cases}$$

$$\text{and } \text{accurate}(y, z) = \max\left\{\frac{126 - y}{255}, \frac{58 - z}{255}, 0\right\}.$$

- (i) For an ink evaluation of the color $(r, g, b) = (100, 100, 100)$ compute the $\text{brand-conformity}(r, g, b)$ and show all computation steps. (7pts)

- (ii) Apparently, an overworked color scientist previously proposed the following formula to use instead of *accurate*:

$$\text{accurate}'(y, z) = \min \{126 - y, 58 - z\}$$

to test for color accuracy more directly. Why did the head color scientist change the fuzzy variable definition *accurate'* to the version *accurate* you see above? Explain your reasoning. (3pts)

- (iii) To decide, finally, whether a *Dipus* is adequate as a mascot or not, the head of color theory tasks you with designing a *defuzzifier*, i.e., a function that takes in the value of *brand-conformity*(r, g, b) and returns a decision variable

$$\text{useful} \in \{\text{use-as-mascot}, \text{dont-use-as-mascot}\}.$$

Give your *defuzzifier* for the decision variable *useful*, give its type signature, and briefly explain your reasoning. (5pts)

4 Fuzzy Logic

15pts

Like octopuses, the *Dipus* from Scenario 1 — when threatened — can eject a cloud of ink to confuse predators. To accommodate LMU branding guidelines, this ink naturally needs to be a greenish color. The exact tone of greenness is then judged very critically, and only the most color-accurate of *Dipuses* are allowed to be released into the harsh nature of university mascots. LMU scientists sample the color of the ink by its *RGB* value, with $r, g, b \in [0, 255]$, and then decide the $\text{brand-conformity}(r, g, b)$.

Directly from LMU's head of color theory, we were told about the following formulae in fuzzy logic:

$$\text{brand-conformity}(r, g, b) = \text{AND}(\text{pretty}(\text{eccentric}(r, g)), \text{pretty}(\text{accurate}(g, b))),$$

$$\text{where } \text{pretty}(u) = \min\{2u, 1\},$$

$$\text{eccentric}(v, x) = \begin{cases} 0 & \text{if } 42 - v - x \geq 0, \\ 0.42 & \text{otherwise,} \end{cases}$$

$$\text{and } \text{accurate}(y, z) = \max\left\{\frac{126 - y}{255}, \frac{58 - z}{255}, 0\right\}.$$

- (i) For an ink evaluation of the color $(r, g, b) = (100, 100, 100)$ compute the $\text{brand-conformity}(r, g, b)$ and show all computation steps. (7pts)

$$\text{brand-conformity}(r, g, b) = \text{AND}(\text{pretty}(\text{excentric}(100, 100)), \text{pretty}(\text{accurate}(100, 100)))$$
$$\text{pretty}(\text{excentric}(100, 100)) = \text{pretty}(0.42) = 2 \cdot 0.42 = 0.84$$

$$\text{pretty}(\text{accurate}(100, 100)) = \text{pretty}(\max\left\{\frac{26}{255}, -\frac{42}{255}, 0\right\}) = 2 \cdot \frac{26}{255} = \frac{52}{255}$$

$$\text{AND}(0.84, \frac{52}{255}) = \min\left\{0.84, \frac{52}{255}\right\} = \frac{52}{255}$$

- (ii) Apparently, an overworked color scientist previously proposed the following formula to use instead of *accurate*:

$$\text{accurate}'(y, z) = \min \{126 - y, 58 - z\}$$

to test for color accuracy more directly. Why did the head color scientist change the fuzzy variable definition *accurate'* to the version *accurate* you see above? Explain your reasoning. (3pts)

Because the function did not conform to principles of fuzzy variables, which must be bounded by $\mathbb{F} \in [0; 1]$, i.e., must not be negative and must not be larger than 1. Both cases were possible with the previous function *accurate'*, but are fixed in *accurate* by the max operator and the percentage scaling.

- (iii) To decide, finally, whether a *Dipus* is adequate as a mascot or not, the head of color theory tasks you with designing a *defuzzifier*, i.e., a function that takes in the value of *brand-conformity*(r,g,b) and returns a decision variable

$$\text{useful} \in \{\text{use-as-mascot}, \text{dont-use-as-mascot}\}.$$

Give your *defuzzifier* for the decision variable *useful*, give its type signature, and briefly explain your reasoning. (5pts)

Defuzzifiers are responsible for converting our fuzzy variables back into real world decisions. In our case, we could map the values of the function *brand-conformity* to the decision variable with a desired threshold value like 0.8, e.g., with:

$$\text{useful} : \mathbb{F} \rightarrow \{\text{use-as-mascot}, \text{dont-use-as-mascot}\}$$

and

$$c = \text{brand-conformity}(r, g, b),$$

$$\text{useful}(c) = \begin{cases} \text{use-as-mascot} & \text{if } c > 0.8, \\ \text{dont-use-as-mascot} & \text{otherwise.} \end{cases}$$

4 Fuzzy Logic

15pts

The LMU researchers need to assess the risk associated with each chamber discovered in the mole burrow. A high-risk chamber might indicate recent mole activity near sensitive lab equipment or structural instability requiring immediate attention. They decide to use a fuzzy logic system to calculate the *ChamberRisk* based on sensor readings from the *MoleBots*.

The system considers three factors for a chamber v : Its *distance* $d \in [0; 50]$ to the nearest known sensitive lab area (in some units), a sensor reading $p \in [0; 100]$ indicating the mole *presence* (higher means more signs of recent mole activity), and a sensor reading $w \in [0; 1]$ indicating wall *weakness* (higher means the chamber walls are less stable).

The LMU fuzzy systems expert provides the following formulae:

$$\begin{aligned} ChamberRisk(d, p, w) &= \text{AND}(ProximityAlert}(d), ActivityLevel(p, w)) \\ \text{where } ProximityAlert}(d) &= \max \left\{ 0, 1 - \frac{d}{50} \right\}, \\ ActivityLevel(p, w) &= \text{OR}(MoleTraffic}(p), Instability(w)), \\ MoleTraffic}(p) &= \frac{p^2}{10000}, \\ Instability(w) &= \sqrt{w}. \end{aligned}$$

(i) One of our *MoleBots* surveys a chamber v and reports the following readings:

- distance to sensitive area: $d = 5$
- presence of moles sensor: $p = 80$
- wall weakness sensor: $w = 0.36$

Calculate the *ChamberRisk* for this chamber using the provided formulae. Show all computation steps. (7pts).

(ii) Assume that a different *MoleBot* reported readings p' and w' for a chamber v' such that the corresponding fuzzy variable was estimated at $ActivityLevel = 0.80$. What is the highest distance value d' that the *MoleBot* could have reported where the resulting final *ChamberRisk* for v' is still 0.80? Briefly explain your answer. (3pts)

(iii) Based on the calculated *ChamberRisk*, the researchers need to decide whether to dispatch further *MoleBot* agents for a detailed inspection. Design a *defuzzifier* function named *DispatchDecision* that takes the *ChamberRisk* value (a fuzzy value in range $[0; 1]$) and outputs a crisp decision from the set {*Dispatch*, *Monitor*, *Abort*}, where each of these decisions occurs depending on the readings d , p , or w . Provide the type signature and a brief reasoning behind your defuzzifier function. (5pts)

4 Fuzzy Logic

15pts

The LMU researchers need to assess the risk associated with each chamber discovered in the mole burrow. A high-risk chamber might indicate recent mole activity near sensitive lab equipment or structural instability requiring immediate attention. They decide to use a fuzzy logic system to calculate the *ChamberRisk* based on sensor readings from the *MoleBots*.

The system considers three factors for a chamber v : Its *distance* $d \in [0; 50]$ to the nearest known sensitive lab area (in some units), a sensor reading $p \in [0; 100]$ indicating the mole *presence* (higher means more signs of recent mole activity), and a sensor reading $w \in [0; 1]$ indicating wall *weakness* (higher means the chamber walls are less stable).

The LMU fuzzy systems expert provides the following formulae:

$$\begin{aligned} ChamberRisk(d, p, w) &= \text{AND}(ProximityAlert}(d), ActivityLevel(p, w)) \\ \text{where } ProximityAlert}(d) &= \max \left\{ 0, 1 - \frac{d}{50} \right\}, \\ ActivityLevel(p, w) &= \text{OR}(MoleTraffic}(p), Instability(w)), \\ MoleTraffic}(p) &= \frac{p^2}{10000}, \\ Instability(w) &= \sqrt{w}. \end{aligned}$$

(i) One of our *MoleBots* surveys a chamber v and reports the following readings:

- distance to sensitive area: $d = 5$
- presence of moles sensor: $p = 80$
- wall weakness sensor: $w = 0.36$

Calculate the *ChamberRisk* for this chamber using the provided formulae. Show all computation steps. (7pts).

We need to compute $ChamberRisk(5, 80, 0.36)$.

$$\begin{aligned}
 ProximityAlert(15) &= \max \left\{ 0, 1 - \frac{5}{50} \right\} \\
 &= \max \{0, 1 - 0.1\} = \max \{0, 0.9\} = 0.9 \\
 MoleTraffic(80) &= \frac{80^2}{10000} = \frac{6400}{10000} = 0.64 \\
 Instability(0.36) &= \sqrt{0.36} = 0.6 \\
 ActivityLevel(80, 0.36) &= \text{OR}(MoleTraffic(80), Instability(0.36)) \\
 &= \text{OR}(0.64, 0.6) = \max \{0.64, 0.6\} = 0.64 \\
 ChamberRisk(5, 80, 0.36) &= \text{AND}(ProximityAlert(5), ActivityLevel(80, 0.36)) \\
 &= \text{AND}(0.9, 0.64) = \min \{0.9, 0.64\} = 0.64
 \end{aligned}$$

The ChamberRisk is 0.64.

- (ii) Assume that a different *MoleBot* reported readings p' and w' for a chamber v' such that the corresponding fuzzy variable was estimated at $ActivityLevel = 0.80$. What is the highest distance value d' that the *MoleBot* could have reported where the resulting final *ChamberRisk* for v' is still 0.80? Briefly explain your answer. (3pts)

For a ChamberRisk of 0.8, it needs to hold for a distance $d(v')$ that

$$\text{AND}(d(v'), 0.84) = \min \{d(v'), 0.80\} = 0.80,$$

or, simply put, that $d(v') \geq 0.80$. This is the case for values of $d(v') \in [0; 10]$ such that $ProximityAlert(d(v')) \geq 0.8$. The largest distance value for a final ChamberRisk of 0.8 is therefore $d(v') = 10$.

- (iii) Based on the calculated *ChamberRisk*, the researchers need to decide whether to dispatch further *MoleBot* agents for a detailed inspection. Design a *defuzzifier* function named *DispatchDecision* that takes the *ChamberRisk* value (a fuzzy value in range $[0; 1]$) and outputs a crisp decision from the set $\{\text{Dispatch}, \text{Monitor}, \text{Abort}\}$, where each of these decisions occurs depending on the readings d , p , or w . Provide the type signature and a brief reasoning behind your defuzzifier function. (5pts)

$$DispatchDecision : [0; 1] \rightarrow \{\text{Dispatch}, \text{Monitor}, \text{Abort}\}$$

$$DispatchDecision(\text{ChamberRisk}(d, p, w)) = \begin{cases} \text{Dispatch} & \text{if } d < 15 \text{ or } p > 80 \text{ or } w > 0.7 \\ \text{Monitor} & \text{if } d < 35 \text{ or } p > 40 \text{ or } w > 0.4 \\ \text{Abort} & \text{otherwise} \end{cases}$$

This rule-based defuzzification reflects the underlying intuition of the fuzzy model:

- **Dispatch** is triggered by close proximity to sensitive areas, strong mole activity, or severe structural instability.
- **Monitor** is chosen for moderate risk levels in any of the three inputs, where ongoing observation may be sufficient.
- **Abort** is the default for low-risk chambers, which are far from sensitive areas, stable, and show little to no mole presence.

Feedback Aufgabe 4

2022-Exam-Coln.pdf

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf

2025-MockExam1-Coln.pdf

2025-MockExam2-Coln.pdf

Gesamtpunkte (ueber alle Aufgaben)

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 :

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

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 sub-field 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., $\pi_E : \mathcal{O} \rightarrow \mathcal{A}$. 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., $\pi_\theta : \mathcal{R} \rightarrow \mathcal{O} \rightarrow \mathcal{A}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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., $x_0 = \text{def } 42$) 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 $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x_u \in \mathcal{X}$. Let $x'_u \sim \text{neighbors}(x_u)$ be a neighbor state of x_u that was drawn at random. Let $T : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ 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

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

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

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 ($N = \{\text{``a''}, \text{``b''}, \text{``c''}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

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

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

(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 $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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 $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

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 :

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

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 sub-field 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...

ChatGPT Assistant:

Sure, I'd be happy to provide more information about squirrels! What specific aspect of squirrels would you like to know more about? Their behavior, habitat, biology, or something else?

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

As ChatGPT elaborates:

This question tests a student's understanding of the basic concept of IRL and its motivation. The answer should reflect an understanding of the definition of IRL and its purpose, which is **to infer a reward function from observed behavior of an expert in an environment**. The answer should also briefly mention **the limitations of specifying a reward function in RL problems and the potential benefits of using IRL to address these limitations**.

- (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., $\pi_E : \mathcal{O} \rightarrow \mathcal{A}$. 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., $\pi_\theta : \mathcal{R} \rightarrow \mathcal{O} \rightarrow \mathcal{A}$ 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.

The expert policy π_E is not given via an algorithm (with possible parameters) but via a data set $\mathbb{D} \subseteq \mathcal{O} \times \mathcal{A}$ of expert actions $a \in \mathcal{A}$ given observations $o \in \mathcal{O}$ with $(o, a) \in \mathbb{D}$.

We could define a distance function D on two policies, e.g., by counting the times they suggest different actions given the same observations. (As a concrete suggestion, for some action similarity relation \approx , let

$$D(\pi, \pi') = \sum_{o \in \mathbb{O}} \begin{cases} 0 & \text{if } \pi(o) \approx \pi'(o), \\ 1 & \text{otherwise,} \end{cases}$$

if it is possible to iterate through all of \mathbb{O} efficiently.)

- (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)

There are several learning and optimization techniques (cf. gradient descent, e.g.) that can use (helpful) gradient information for more efficient computation by gaining a rough estimate on the right direction in the search space to look into next. (This is reflected in gradient-based goals sitting in a higher level of the goal hierarchy introduced in the lecture. ;D)

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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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., $x_0 = \text{def } 42$) 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 $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x_u \in \mathcal{X}$. Let $x'_u \sim \text{neighbors}(x_u)$ be a neighbor state of x_u that was drawn at random. Let $T : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ 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

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

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

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 ($N = \{\text{``a''}, \text{``b''}, \text{``c''}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

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

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

(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 $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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 $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

5 Fuzzy Logic

10pts

A home buyer has to decide between the houses A and B. Thus, he makes his decision considering the criteria *price* and *location*. The houses are similar with respect to their location — which the buyer is familiar with — and statements regarding their pricing are already assigned with ‘truth values’ (see table below). For the price, however, the buyer has to consult a survey, here pictured in Figure 1.

Statement		Truth value
ψ_1	Price of house A (400k EUR) is good	?
ψ_2	Location of house A is good	0.6
ψ_3	Price of house B (600k EUR) is good	?
ψ_4	Location of house B is good	0.7

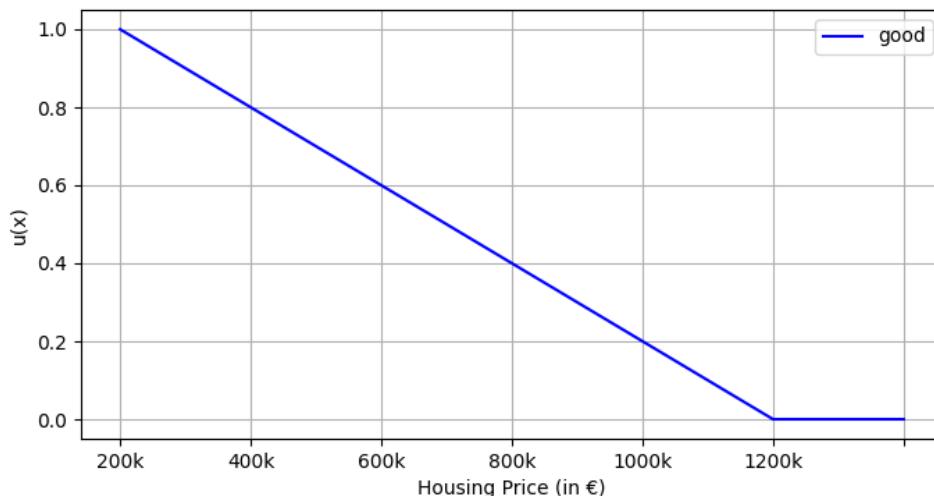


Figure 1: Housing Prices — Fuzzy Membership Function

- (i) Now consider that our buyer gets more picky and now wants a *very* nice house. We have seen linguistic modifiers defined in the lecture before; we again define the operator *very* as a modification of the truth value membership function as the following:

$$\text{very}(a) = a^2$$

In the Figure 1, add the graph for this modification. Your function should be defined at

least for the points

$$\begin{aligned}x_1 &= 400k, \\x_2 &= 600k, \\x_3 &= 1200k.\end{aligned}$$

(2pts)

- (ii) In case the buyer changes his mind, define a linguistic modifier *somewhat* with the only constraint

$$\mu(x) > \text{somewhat}(\mu(x))$$

Also add the graph for the membership function of *somewhat* in the Figure 1, again, defined for (at least) the points x_1, x_2, x_3 (2pts).

- (iii) Finally, compute the truth value for the buyer considering the proposition ‘*The price of house x is very good OR the location of house x is somewhat good*’. In favor of which house does the buyer decide? (6pts)

5 Fuzzy Logic

10pts

A home buyer has to decide between the houses A and B. Thus, he makes his decision considering the criteria *price* and *location*. The houses are similar with respect to their location — which the buyer is familiar with — and statements regarding their pricing are already assigned with ‘truth values’ (see table below). For the price, however, the buyer has to consult a survey, here pictured in Figure 1.

Statement		Truth value
ψ_1	Price of house A (400k EUR) is good	?
ψ_2	Location of house A is good	0.6
ψ_3	Price of house B (600k EUR) is good	?
ψ_4	Location of house B is good	0.7

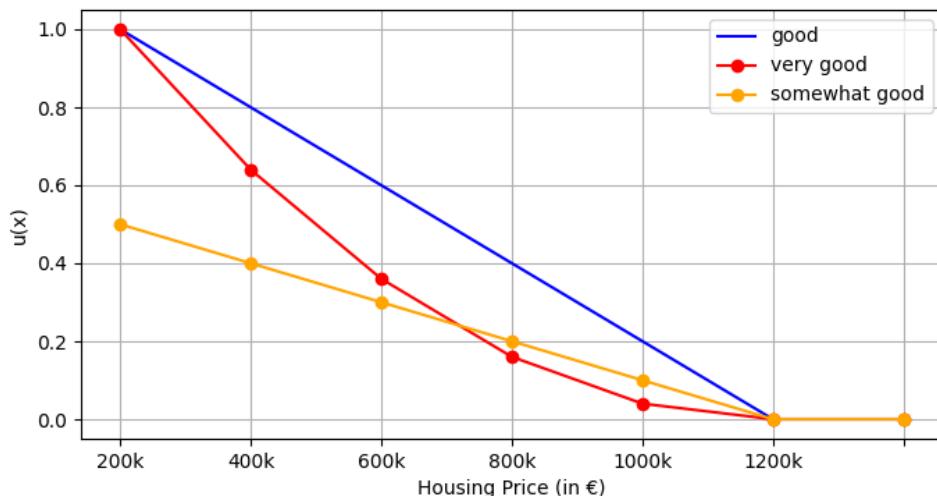


Figure 1: Housing Prices — Fuzzy Membership Function [Solution]

- (i) Now consider that our buyer gets more picky and now wants a *very* nice house. We have seen linguistic modifiers defined in the lecture before; we again define the operator *very* as a modification of the truth value membership function as the following:

$$\text{very}(a) = a^2$$

In the Figure 1, add the graph for this modification. Your function should be defined at

least for the points

$$\begin{aligned}x_1 &= 400k, \\x_2 &= 600k, \\x_3 &= 1200k.\end{aligned}$$

(2pts)

Cf. Figure 1 solution.

$$\begin{aligned}\text{very } \mu(x_1)^2 &= 0.8^2 = 0.64 \\ \text{very } \mu(x_2)^2 &= 0.6^2 = 0.39 \\ \text{very } \mu(x_3)^2 &= 0^2 = 0\end{aligned}$$

(ii) In case the buyer changes his mind, define a linguistic modifier *somewhat* with the only constraint

$$\mu(x) > \text{somewhat}(\mu(x))$$

Also add the graph for the membership function of *somewhat* in the Figure 1, again, defined for (at least) the points x_1, x_2, x_3 (2pts).

E.g.

$$\text{somewhat } (a) = a/2$$

$$\begin{aligned}\text{somewhat } \mu(x_1)/2 &= 0.8/2 = 0.4 \\ \text{somewhat } \mu(x_2)/2 &= 0.6/2 = 0.3 \\ \text{somewhat } \mu(x_3)/2 &= 0/2 = 0\end{aligned}$$

(iii) Finally, compute the truth value for the buyer considering the proposition ‘*The price of house x is very good OR the location of house x is somewhat good*’. In favor of which house does the buyer decide? (6pts)

very(good price A 400k) = $0.8^2 = 0.64$

somewhat(good location A) = $0.6/2 = 0.3$

very(good price A) \vee somewhat(good location A) = $\max(0.64, 0.3) = 0.64$

very(good price B 600k) = $0.6^2 = 0.36$

somewhat(good location B) = $0.7/2 = 0.35$

very(good price B) \vee somewhat(good location B) = $\max(0.36, 0.35) = 0.36$

$\rightarrow A > B$

5 Scientific Reading: Empowerment

20pts

Read the following (correct) explanation of the novel idea of *empowerment*, which we did not discuss in the lecture, and answer the questions afterwards.

The CoIn Team:

Generate a one-page explanation of empowerment in reinforcement learning. Include and explain relevant formulas in LaTeX format. Make the text suitable for computer science graduate students.

The ChatGPT Assistant:

Empowerment in reinforcement learning refers to the ability of an agent to maximize its influence over the environment. It is an intrinsic motivation that encourages agents to explore and take actions that can lead to long-term benefits, even if they do not result in immediate rewards.

One way to measure empowerment is through information-theoretic concepts, such as the mutual information between the agent's actions and the resulting state of the environment. Mutual information is a measure of the degree of dependence between two variables, and in this case, it can be used to capture how much the agent's actions influence the environment. The formula for mutual information is:

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Where X and Y represent two random variables, and $p(x,y)$, $p(x)$, and $p(y)$ represent their joint and marginal probabilities, respectively.

To apply empowerment in reinforcement learning, the agent maximizes the mutual information between its actions and the resulting state of the environment. This can be achieved through various algorithms, such as empowerment-based exploration or empowerment-driven reinforcement learning.

Empowerment has several benefits in reinforcement learning. First, it encourages agents to explore their environment more broadly, which can lead to the discovery of new strategies and solutions. Second, it can help agents to learn more robust and adaptable policies, as they focus on the long-term effects of their actions rather than just maximizing immediate rewards.

In conclusion, empowerment is a valuable concept in reinforcement learning that can help agents to become more effective at achieving their goals. By maximizing the mutual information between its actions and the environment, an agent can become more empowered and navigate complex and changing environments more efficiently.

The CoIn Team:

Give a very short overview of common problems when using empowerment.

The ChatGPT Assistant:

One common problem when using empowerment in reinforcement learning is that it can be difficult to estimate the mutual information between the agent's actions and the environment, which is required to compute empowerment. Another challenge is that maximizing empowerment can sometimes lead to suboptimal behavior if the agent becomes overly focused on exploring at the expense of exploiting known high-reward actions. Finally, the computational complexity of empowerment-based algorithms can be high, which may limit their practical applicability in some settings.

(i) Answer the following question partly generated by ChatGPT: Explain the concept of empowerment in reinforcement learning, and give an example of how it can be used to encourage more efficient exploration. (8pts)

(ii) Discuss one potential challenge in using empowerment as a learning signal for reinforcement learning agents. (4pts)

(iii) Again imagine a single *RepairBot* traveling through a grid of power nodes as we considered throughout this exam. In a dynamic environment, where nodes can become **alarming** in a chaotic way, what kind of behavior of the *RepairBot*, which is rewarded for not letting nodes be **defect** for long, would we expect when the *RepairBot* is trained to maximize empowerment as well? If the *RepairBot* could teleport to any arbitrary power node at will, how would that change its empowerment? (8pts)

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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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., $x_0 = \text{def } 42$) 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 $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x_u \in \mathcal{X}$. Let $x'_u \sim \text{neighbors}(x_u)$ be a neighbor state of x_u that was drawn at random. Let $T : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ 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

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

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 (best response). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

Definition 8 (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 9 (evolutionary stable strategy). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game played by two players for a single iteration. 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

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

Definition 10 (π -process (shortened)). Let N be a set of names ($N = \{\text{``a''}, \text{``b''}, \text{``c''}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

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

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

(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 $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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 $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

5 Scientific Reading: Empowerment

20pts

Read the following (correct) explanation of the novel idea of *empowerment*, which we did not discuss in the lecture, and answer the questions afterwards.

The CoIn Team:

Generate a one-page explanation of empowerment in reinforcement learning. Include and explain relevant formulas in LaTeX format. Make the text suitable for computer science graduate students.

The ChatGPT Assistant:

Empowerment in reinforcement learning refers to the ability of an agent to maximize its influence over the environment. It is an intrinsic motivation that encourages agents to explore and take actions that can lead to long-term benefits, even if they do not result in immediate rewards.

One way to measure empowerment is through information-theoretic concepts, such as the mutual information between the agent's actions and the resulting state of the environment. Mutual information is a measure of the degree of dependence between two variables, and in this case, it can be used to capture how much the agent's actions influence the environment. The formula for mutual information is:

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Where X and Y represent two random variables, and $p(x,y)$, $p(x)$, and $p(y)$ represent their joint and marginal probabilities, respectively.

To apply empowerment in reinforcement learning, the agent maximizes the mutual information between its actions and the resulting state of the environment. This can be achieved through various algorithms, such as empowerment-based exploration or empowerment-driven reinforcement learning.

Empowerment has several benefits in reinforcement learning. First, it encourages agents to explore their environment more broadly, which can lead to the discovery of new strategies and solutions. Second, it can help agents to learn more robust and adaptable policies, as they focus on the long-term effects of their actions rather than just maximizing immediate rewards.

In conclusion, empowerment is a valuable concept in reinforcement learning that can help agents to become more effective at achieving their goals. By maximizing the mutual information between its actions and the environment, an agent can become more empowered and navigate complex and changing environments more efficiently.

The CoIn Team:

Give a very short overview of common problems when using empowerment.

The ChatGPT Assistant:

One common problem when using empowerment in reinforcement learning is that it can be difficult to estimate the mutual information between the agent's actions and the environment, which is required to compute empowerment. Another challenge is that maximizing empowerment can sometimes lead to suboptimal behavior if the agent becomes overly focused on exploring at the expense of exploiting known high-reward actions. Finally, the computational complexity of empowerment-based algorithms can be high, which may limit their practical applicability in some settings.

- (i) Answer the following question partly generated by ChatGPT: Explain the concept of empowerment in reinforcement learning, and give an example of how it can be used to encourage more efficient exploration. (8pts)

The ChatGPT Assistant:

Empowerment in reinforcement learning refers to the agent's ability to maximize its influence over the environment, by taking actions that lead to long-term benefits even if they do not result in immediate rewards. One way to measure empowerment is through information-theoretic concepts, such as the mutual information between the agent's actions and the resulting state of the environment. Maximizing empowerment can encourage more efficient exploration, as the agent focuses on actions that have a high potential to influence the environment, rather than just maximizing immediate rewards.

- (ii) Discuss one potential challenge in using empowerment as a learning signal for reinforcement learning agents. (4pts)

The ChatGPT Assistant:

One potential challenge in using empowerment as a learning signal for reinforcement learning agents is that it can be difficult to estimate the mutual information between the agent's actions and the environment. This can result in high computational complexity, which may limit the practical applicability of empowerment-based algorithms. One possible solution to address this issue is to use approximations or simplified models to estimate mutual information, or to combine empowerment with other learning signals such as reward-based reinforcement learning to balance exploration and exploitation more effectively.

- (iii) Again imagine a single *RepairBot* traveling through a grid of power nodes as we considered throughout this exam. In a dynamic environment, where nodes can become alarming in a chaotic way, what kind of behavior of the *RepairBot*, which is rewarded for not letting nodes be defect for long, would we expect when the *RepairBot* is trained to maximize empowerment as well? If the *RepairBot* could teleport to any arbitrary power node at will, how would that change its empowerment? (8pts)

A *RepairBot* trained for empowerment would prefer to visit locations that make it easy to reach the whole grid, thus maximizing the effect of its actions by re-positioning near the center of the grid, for example.

If the *RepairBot* could teleport, its empowerment would be increased as it can reach every position via a single action. Consequently, its positioning within the grid would no longer matter.

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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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., $x_0 = \text{def } 42$) 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 $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x_u \in \mathcal{X}$. Let $x'_u \sim \text{neighbors}(x_u)$ be a neighbor state of x_u that was drawn at random. Let $T : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ 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

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

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 (best response). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

Definition 8 (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 9 (evolutionary stable strategy). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game played by two players for a single iteration. 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

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

Definition 10 (π -process (shortened)). Let N be a set of names ($N = \{\text{``a''}, \text{``b''}, \text{``c''}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

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

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

(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 $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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 $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

5 The π -Calculus

10pts

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

$!(\overline{\text{panic}}.\text{beep}.\mathbf{0} + \overline{\text{beep}}.\text{MowBot} + \text{panic}.\text{beep}.\text{Cow} + \overline{\text{brr}}.\text{moo}.\text{Cow}) \mid \overline{\text{moo}}.\text{Cow} \mid \overline{\text{panic}}.\text{moo}.\mathbf{0}$

- (ii) Evaluate the following π -process step by step until it cannot be reduced any further.
(4pts)

$!(\overline{\text{beep}}.\text{MowBot}) \mid \overline{\text{beep}}.\text{beep}.(\text{panic}.\overline{\text{moo}}.\text{beep}.\mathbf{0} + \text{beep}.\text{panic}.\text{MowBot}) \mid \text{beep}.\mathbf{0} \mid \overline{\text{beep}}.\mathbf{0}$

5 The π -Calculus

10pts

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

$!(\overline{\text{panic}}.\text{beep.0} + \overline{\text{beep}}.\text{MowBot} + \text{panic}.\text{beep.Cow} + \overline{\text{brr}}.\text{moo.Cow}) \mid \overline{\text{moo}}.\text{Cow} \mid \overline{\text{panic}}.\text{moo.0}$

$\rightarrow !(\overline{\text{panic}}.\text{beep.0} + \overline{\text{beep}}.\text{MowBot} + \text{panic}.\text{beep.Cow} + \overline{\text{brr}}.\text{moo.Cow}) \mid \overline{\text{moo}}.\text{Cow} \mid \text{beep.Cow} \mid \text{moo.0}$
 $\rightarrow !(\overline{\text{panic}}.\text{beep.0} + \overline{\text{beep}}.\text{MowBot} + \text{panic}.\text{beep.Cow} + \overline{\text{brr}}.\text{moo.Cow}) \mid \text{Cow} \mid \text{beep.Cow} \mid \mathbf{0}$
 $\rightarrow !(\overline{\text{panic}}.\text{beep.0} + \overline{\text{beep}}.\text{MowBot} + \text{panic}.\text{beep.Cow} + \overline{\text{brr}}.\text{moo.Cow}) \mid \text{Cow} \mid \text{Cow} \mid \text{MowBot} \mid \mathbf{0}$
 $\rightarrow !(\overline{\text{panic}}.\text{beep.0} + \overline{\text{beep}}.\text{MowBot} + \text{panic}.\text{beep.Cow} + \overline{\text{brr}}.\text{moo.Cow}) \mid \text{Cow} \mid \text{Cow} \mid \text{MowBot}$

- (ii) Evaluate the following π -process step by step until it cannot be reduced any further.
(4pts)

$!(\overline{\text{beep}}.\text{MowBot}) \mid \overline{\text{beep}}.\text{beep}.(\text{panic}.\overline{\text{moo}}.\text{beep.0} + \text{beep}.\text{panic.MowBot}) \mid \text{beep.0} \mid \overline{\text{beep}}.\mathbf{0}$

one possible path (out of several):

$\rightarrow !(\overline{\text{beep}}.\text{MowBot}) \mid \overline{\text{beep}}.\text{beep}.(\text{panic}.\overline{\text{moo}}.\text{beep.0} + \text{beep}.\text{panic.MowBot}) \mid \mathbf{0} \mid \text{MowBot} \mid \overline{\text{beep}}.\mathbf{0}$
 $\rightarrow !(\overline{\text{beep}}.\text{MowBot}) \mid \overline{\text{beep}}.\text{beep}.(\text{panic}.\overline{\text{moo}}.\text{beep.0} + \text{beep}.\text{panic.MowBot}) \mid \text{MowBot} \mid \overline{\text{beep}}.\mathbf{0}$

5 Scientific Reading: The Spi-Calculus

20pts

Read the adapted paper below¹ 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
$[M \text{ is } N] P$	match
$\mathbf{0}$	nil
$\text{let } (x, y) = M \text{ in } P$	pair splitting
$\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$	integer case

¹from: 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 $\text{let } (x, y) = M \text{ in } P$, the variables x and y are bound in P . In $\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$, 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 $\text{let } (x, y) = M \text{ in } P$ behaves as $P[N/x][L/y]$ if term M is the pair (N, L) , and otherwise it is stuck.
- An integer case process $\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$ behaves as P if term M is 0, as $Q[N/x]$ if M is $\text{suc}(N)$, 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:

Message 1 $A \rightarrow B : M$ on c_{AB}

A first pi-calculus description of this protocol is:

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

The processes $A(M)$ and B describe the two principals, and $Inst(M)$ 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 $Inst(M)$ 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 $Inst(c_{AB})$ 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:

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

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 $Inst(M) \simeq Inst(M')$. 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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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 $X = \{x_0, \dots, x_n\}$, $|X| = n$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ 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., $x_0 = \text{def } 42$) 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

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

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

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost (reward)* function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $V^* : \mathcal{S} \rightarrow \mathcal{T}$ be the *true value function* of a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$. The optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ 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 $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ be a neural network with n weights $\bar{\mathcal{N}} = \mathbf{w} \# \mathbf{b} \in \mathbb{R}^n$.

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

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

for $\mathcal{O} \subseteq \mathbb{R}^p, \mathcal{A} \subseteq \mathbb{R}^q$ or if suitable translations exist.

If τ is of the form

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

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 $\pi_{\bar{\mathcal{N}}} : \mathcal{S} \rightarrow \mathcal{A}$. Note that $\pi_{\bar{\mathcal{N}}}$ in some way calls \mathcal{N} to produce its output, for example

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

for $\mathcal{S} \times \mathcal{A} \subseteq \mathbb{R}^p$ with $q = 1$ or if suitable translations exist.

Let $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{T}$ 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 $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ 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 $V_{\pi_{\bar{\mathcal{N}}}} : \mathcal{S} \rightarrow \mathbb{R}$ be the total discounted reward that policy $\pi_{\bar{\mathcal{N}}}$ generates when starting in state s , i.e.,

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

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 $T(s^\dagger, \cdot)$ 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 $\mathbb{S} = \{\mathbf{s}_i : i = 1, \dots, N\} \subseteq S$ be a set of training states. If τ is of the form

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

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

Definition 7 (multi-agent system). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ 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 ($N = \{\text{``a''}, \text{``b''}, \text{``c''}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

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

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

(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 $\mathfrak{F} : \mathbb{L}_\pi \rightarrow \wp(N)$ are given inductively via:

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

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 $\equiv : \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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 $x \neq y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{z}\langle z \rangle.(P[y := z])$ for $a = y$ and $x = y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{a}\langle z \rangle.(P[y := z])$ for $a \neq y$ and $x = y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{z}\langle x \rangle.(P[y := z])$ for $a = y$ and $x \neq y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{a}\langle x \rangle.(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(P + Q)[y := z] = (P[y := z]) + (Q[y := z])$,
- $(P | Q)[y := z] = (P[y := z]) | (Q[y := z])$,
- $((\nu x) P)[y := z] = ((\nu x) P)$ 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 $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

5 Scientific Reading: The Spi-Calculus

20pts

Read the adapted paper below¹ 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
$[M \text{ is } N] P$	match
$\mathbf{0}$	nil
$\text{let } (x, y) = M \text{ in } P$	pair splitting
$\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$	integer case

¹from: 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 $\text{let } (x, y) = M \text{ in } P$, the variables x and y are bound in P . In $\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$, 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 $\text{let } (x, y) = M \text{ in } P$ behaves as $P[N/x][L/y]$ if term M is the pair (N, L) , and otherwise it is stuck.
- An integer case process $\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$ behaves as P if term M is 0, as $Q[N/x]$ if M is $\text{suc}(N)$, 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:

Message 1 $A \rightarrow B : M$ on c_{AB}

A first pi-calculus description of this protocol is:

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

The processes $A(M)$ and B describe the two principals, and $Inst(M)$ 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 $Inst(M)$ 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 $Inst(c_{AB})$ 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:

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

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 $Inst(M) \simeq Inst(M')$. 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)

Any two of the following examples:

- added constructs for pairings and numbers like (M,N) and 0
- added a successor function that works with numbers
- added matches, pair-splitting and integer case to the process notation
- (obviously) introduced the spi-calculus extension

(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. (10pts)

(a) The ‘restricted channel’ property $(\nu n) P$ for a process P (to model cryptographically secure channel communication).

(b 1) Authenticity: The secure F process always receives the same input parameter as the bound variable of the c_{AB} channel, making sure that the message always gets ‘passed on’ to the following (secure) process.

and

(b 2) Secrecy: Due to the priority and exclusivity of the c_{AB} channel and the authenticity property, if message M does not get ‘leaked’ (revealed/sent) from process F , then M is always secret to F .

(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)

Changed from $c_{AB}(x).\mathbf{0}$ to $c_{AB}(x).F(x)$ to signify that the information x received over secure channel c_{AB} then actually gets used in / bound to some process F and only F (authenticity / integrity property).

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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 $X = \{x_0, \dots, x_n\}$, $|X| = n$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ 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., $x_0 =_{\text{def}} 42$) 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

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

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

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $V^* : \mathcal{S} \rightarrow \mathcal{T}$ be the *true value function* of a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$. The optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ 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 $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ 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 $\pi_{\overline{\mathcal{N}}} : \mathcal{O} \rightarrow \mathcal{A}$ 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

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

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 $\pi_{\overline{\mathcal{N}}} : \mathcal{S} \rightarrow \mathcal{A}$. Note that $\pi_{\overline{\mathcal{N}}}$ in some way calls \mathcal{N} to produce its output, for example

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

for $\mathcal{S} \times \mathcal{A} \subseteq \mathbb{R}^p$ with $q = 1$ or if suitable translations exist.

Let $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{T}$ 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 $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ 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 $V_{\pi_{\overline{\mathcal{N}}}} : \mathcal{S} \rightarrow \mathbb{R}$ be the total discounted reward that policy $\pi_{\overline{\mathcal{N}}}$ generates when starting in state s , i.e.,

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

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 $T(s^\dagger, \cdot)$ 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 $\mathbb{S} = \{\mathbf{s}_i : i = 1, \dots, N\} \subseteq S$ 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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ 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

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

Definition 9 (π -process). Let N be a set of names ($N = \{\text{``a''}, \text{``b''}, \text{``c''}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

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

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

(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 $\mathfrak{F} : \mathbb{L}_\pi \rightarrow \wp(N)$ are given inductively via:

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

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 $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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 $x \neq y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{z}\langle z \rangle.(P[y := z])$ for $a = y$ and $x = y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{a}\langle z \rangle.(P[y := z])$ for $a \neq y$ and $x = y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{z}\langle x \rangle.(P[y := z])$ for $a = y$ and $x \neq y$,
- $(\bar{a}\langle x \rangle.P)[y := z] = \bar{a}\langle x \rangle.(P[y := z])$ 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 $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

5 Game Theory

15pts

We (finally) want our *Dipus* from Scenario 1 to hammer a nail. However, since each of the *Dipus*'s arms has a brain of its own, we end up with the two-player normal form game called the *hammer game* that the arms play with each other. The following matrix describes how the hammer arm and the nail arm might interact, with the hammer-holding arm being the row player and the nail-holding arm being the column player. Note that utilities in the matrix cells are denoted H, L where H is the utility for the row player and L is the utility for the column player.

		hold steady	hold loosely	drop	flinch
hit hard	2, 3	4, 0	2, 3	3, 0	
hit carefully	3, 1	4, 2	1, 3	4, 2	
drop	0, 4	0, 0	2, 3	0, 4	
feint	0, 0	2, 1	2, 0	3, 4	

(i) Compute all pure Nash equilibria (cf. Definitions 6 and 9 in the appendix) for the *hammer game* by explicitly denoting all best responses in the matrix above. State all pure Nash equilibria separately below. (6pts)

(ii) Give all pure strategies on the Pareto front (cf. Definition 7 in the appendix) for the *hammer game*. (3pts)

(iii) Change the utilities of exactly one joint strategy so that there is now exactly one more Pareto-optimal strategy than before. Briefly explain your reasoning. (2pts)

(iv) Our examination has led us to the conviction that we need to design a different game for *Dipus* arm interaction. Give your own 2×2 matrix for the *coordinated hammer game* by setting utilities for the joint strategies resulting from the column player's actions $\{hold\ steady, drop\}$ and the row player's actions $\{hit\ precisely, drop\}$ such that the game can be regarded as a pure coordination game. (4pts)

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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.) or other variables that are not used. For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ denotes the number of elements in X . For infinite sets, $|X| = \infty$. For any vector \vec{x} , $|\vec{x}|$ denotes the dimensionality of \vec{x} .

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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 (neural network). A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers L_1, \dots, L_r where each layer L_l consists of $|L_l|$ cells $C_{l,1}, \dots, C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, \dots, |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}$, $e = 1, \dots, |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \rightarrow \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|, e=1, \dots, |L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} \# \mathbf{b}$ is called the network \mathcal{N} 's parameters.

A network's output given an input $\mathbf{x} \in \mathbb{R}^p$ is given via

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \langle O(r, c) \rangle_{c=1, \dots, |L_r|} \in \mathbb{R}^q$$

$$\text{where } O(l, c) = \begin{cases} x_c & \text{if } l = 0, \\ f(B_{l,c} + \sum_{i=1}^{|L_{l-1}|} E_{l,c,i} \cdot O(l-1, i)) & \text{otherwise.} \end{cases}$$

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

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 $\pi^{[-i]}$.

5 Game Theory

15pts

We (finally) want our *Dipus* from Scenario 1 to hammer a nail. However, since each of the *Dipus*'s arms has a brain of its own, we end up with the two-player normal form game called the *hammer game* that the arms play with each other. The following matrix describes how the hammer arm and the nail arm might interact, with the hammer-holding arm being the row player and the nail-holding arm being the column player. Note that utilities in the matrix cells are denoted H, L where H is the utility for the row player and L is the utility for the column player.

	hold steady	hold loosely	drop	flinch
hit hard	2, 3	4, 0	2, 3	3, 0
hit carefully	3, 1	4, 2	1, 3	4, 2
drop	0, 4	0, 0	2, 3	0, 4
feint	0, 0	2, 1	2, 0	3, 4

- (i) Compute all pure Nash equilibria (cf. Definitions 6 and 9 in the appendix) for the *hammer game* by explicitly denoting all best responses in the matrix above. State all pure Nash equilibria separately below. (6pts)

	hold steady	hold loosely	drop	flinch
hit hard	2, <u>3</u>	4, 0	2, <u>3</u>	3, 0
hit carefully	3, 1	4, 2	1, <u>3</u>	4, 2
drop	0, <u>4</u>	0, 0	2, 3	0, <u>4</u>
feint	0, 0	2, 1	2, 0	3, <u>4</u>
	hold steady	hold loosely	drop	flinch
hit hard	2, 3	<u>4</u> , 0	<u>2</u> , 3	3, 0
hit carefully	<u>3</u> , 1	<u>4</u> , 2	1, 3	<u>4</u> , 2
drop	0, 4	0, 0	<u>2</u> , 3	0, <u>4</u>
feint	0, 0	2, 1	<u>2</u> , 0	3, 4
	hold steady	hold loosely	drop	flinch
hit hard	2, <u>3</u>	<u>4</u> , 0	<u>2</u> , 3	3, 0
hit carefully	<u>3</u> , 1	<u>4</u> , 2	1, <u>3</u>	<u>4</u> , 2
drop	0, <u>4</u>	0, 0	<u>2</u> , 3	0, <u>4</u>
feint	0, 0	2, 1	<u>2</u> , 0	3, <u>4</u>

There is exactly one pure Nash equilibrium for the joint strategy (hit hard, drop).

- (ii) Give all pure strategies on the Pareto front (cf. Definition 7 in the appendix) for the *hammer game*. (3pts)

All non-Pareto-dominated strategies in the *hammer game* are Pareto-optimal on the Pareto front, here: (hit carefully, hold loosely) and (hit carefully, flinch) with utilities (4, 2) each, and (feint, flinch) with utility (3, 4).

(iii) Change the utilities of exactly one joint strategy so that there is now exactly one more Pareto-optimal strategy than before. Briefly explain your reasoning. (2pts)

We could simply copy the joint utilities of one, currently non-dominated joint strategy for one dominated one. For example: (hit carefully, drop) \mapsto (3, 4).

(iv) Our examination has led us to the conviction that we need to design a different game for *Dipus* arm interaction. Give your own 2×2 matrix for the *coordinated hammer game* by setting utilities for the joint strategies resulting from the column player's actions $\{hold\ steady, drop\}$ and the row player's actions $\{hit\ precisely, drop\}$ such that the game can be regarded as a pure coordination game. (4pts)

A simple pure *coordinated hammer game* could look like this:

$$\begin{array}{cc} & \begin{matrix} hold\ steady & drop \end{matrix} \\ \begin{matrix} hit\ precisely \\ drop \end{matrix} & \left(\begin{array}{cc} 1, 1 & 0, 0 \\ 0, 0 & 0, 0 \end{array} \right) \end{array}$$

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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.) or other variables that are not used. For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ denotes the number of elements in X . For infinite sets, $|X| = \infty$. For any vector \vec{x} , $|\vec{x}|$ denotes the dimensionality of \vec{x} .

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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 (neural network). A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers L_1, \dots, L_r where each layer L_l consists of $|L_l|$ cells $C_{l,1}, \dots, C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, \dots, |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}$, $e = 1, \dots, |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \rightarrow \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|, e=1, \dots, |L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} \# \mathbf{b}$ is called the network \mathcal{N} 's parameters.

A network's output given an input $\mathbf{x} \in \mathbb{R}^p$ is given via

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \langle O(r, c) \rangle_{c=1, \dots, |L_r|} \in \mathbb{R}^q$$

$$\text{where } O(l, c) = \begin{cases} x_c & \text{if } l = 0, \\ f(B_{l,c} + \sum_{i=1}^{|L_{l-1}|} E_{l,c,i} \cdot O(l-1, i)) & \text{otherwise.} \end{cases}$$

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

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 $\pi^{[-i]}$.

5 Game Theory

15pts

In the case that our *MoleBot* agents from Scenario 1 do happen to meet in the burrow, they can never be too sure if they are interacting with their LMU counterparts, or if they perhaps found a suspicious foreign entity. To resolve this dilemma, these agent play the following *Identification Game*. The following matrix (printed twice for convenience) describes how the agents might interact, with agent 1 being the row player and agent 2 being the column player. Note that utilities in the matrix cells are denoted H, L where H is the utility for the row player and L is the utility for the column player (this matches the notation in the lecture).

	Greet	Observe	Use Codeword	Ignore
Greet	2, 4	2, 4	2, 1	4, 2
Observe	0, 2	1, 2	2, 1	3, 0
Use Codeword	3, 3	0, 1	0, 3	1, 1
Ignore	0, 0	2, 4	3, 3	0, 0

	Greet	Observe	Use Codeword	Ignore
Greet	2, 4	2, 4	2, 1	4, 2
Observe	0, 2	1, 2	2, 1	3, 0
Use Codeword	3, 3	0, 1	0, 3	1, 1
Ignore	0, 0	2, 4	3, 3	0, 0

- (i) Compute all pure Nash equilibria (cf. Definitions 7 and 10 in the appendix) for the *Identification Game* by explicitly denoting all best responses in one (clearly marked) matrix above. State all pure Nash equilibria separately below. (6pts)

(ii) Give all pure strategies on the Pareto front (cf. Definition 8 in the appendix) for the *Identification Game*. (3pts)

(iii) Upon encounter, the *MoleBots* need some way to pass each other in the narrow burrow. They may execute these actions:

- *turn back*: The executing *MoleBot* turns back in the tunnel and goes back to the last chamber it came from.
- *advance*: The executing *MoleBot* tries to push on, but fails if the path is still blocked (because the other mole has not turned back).
- *teleport*: Through technology beyond imagination, the two *MoleBots* switch their position instantly, iff both choose to execute that action.

Model your own 3×3 symmetric payoff matrix using utilities $(H, L) \in \{0, 1\}^2$ for this *Traversal Game* that allows for a cooperative traversal of the burrow. (6pts)

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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.) or other variables that are not used. For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ denotes the number of elements in X . For infinite sets, $|X| = \infty$. For any vector \vec{x} , $|\vec{x}|$ denotes the dimensionality of \vec{x} .

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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ 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., $x_0 =_{\text{def}} 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Algorithm 1 (simulated annealing). Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x \in \mathcal{X}$. Let $\kappa : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ with $\mathbb{P} = [0; 1] \subset \mathbb{R}$ be a function that returns an acceptance probability given two target values and a temperature. Commonly, we use

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

for $\mathcal{T} \subseteq \mathbb{R}$ and Euler's number e . The process \mathcal{D} continues via simulated annealing if e is of the form

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

where $x'_t \sim \text{neighbors}(x_t)$ and $r \sim \mathbb{P}$ are drawn at random for each call to e .

Definition 3 (neural network). A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers L_1, \dots, L_r where each layer L_l consists of $|L_l|$ cells $C_{l,1}, \dots, C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, \dots, |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}$, $e = 1, \dots, |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \rightarrow \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|, e=1, \dots, |L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} \# \mathbf{b}$ is called the network \mathcal{N} 's parameters.

A network's output given an input $\mathbf{x} \in \mathbb{R}^p$ is given via

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \langle O(r, c) \rangle_{c=1, \dots, |L_r|} \in \mathbb{R}^q$$

$$\text{where } O(l, c) = \begin{cases} x_c & \text{if } l = 0, \\ f(B_{l,c} + \sum_{i=1}^{|L_{l-1}|} E_{l,c,i} \cdot O(l-1, i)) & \text{otherwise.} \end{cases}$$

Definition 4 (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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 5 (multi-agent system). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 6 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ for all i . A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game*.

Definition 7 (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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 8 (Pareto front for strategies). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 9 (best response). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

Definition 10 (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 $\pi^{[-i]}$.

5 Game Theory

15pts

In the case that our *MoleBot* agents from Scenario 1 do happen to meet in the burrow, they can never be too sure if they are interacting with their LMU counterparts, or if they perhaps found a suspicious foreign entity. To resolve this dilemma, these agent play the following *Identification Game*. The following matrix (printed twice for convenience) describes how the agents might interact, with agent 1 being the row player and agent 2 being the column player. Note that utilities in the matrix cells are denoted H, L where H is the utility for the row player and L is the utility for the column player (this matches the notation in the lecture).

	Greet	Observe	Use Codeword	Ignore
Greet	2, 4	2, 4	2, 1	4, 2
Observe	0, 2	1, 2	2, 1	3, 0
Use Codeword	3, 3	0, 1	0, 3	1, 1
Ignore	0, 0	2, 4	3, 3	0, 0

	Greet	Observe	Use Codeword	Ignore
Greet	2, 4	2, 4	2, 1	4, 2
Observe	0, 2	1, 2	2, 1	3, 0
Use Codeword	3, 3	0, 1	0, 3	1, 1
Ignore	0, 0	2, 4	3, 3	0, 0

- (i) Compute all pure Nash equilibria (cf. Definitions 7 and 10 in the appendix) for the *Identification Game* by explicitly denoting all best responses in one (clearly marked) matrix above. State all pure Nash equilibria separately below. (6pts)

	Greet	Observe	Use Codeword	Ignore
Greet	2, <u>4</u>	2, <u>4</u>	2, 1	4, 2
Observe	0, <u>2</u>	1, <u>2</u>	2, 1	3, 0
Use Codeword	<u>3</u> , <u>3</u>	0, 1	0, <u>3</u>	1, 1
Ignore	0, 0	2, <u>4</u>	3, 3	0, 0
	Greet	Observe	Use Codeword	Ignore
Greet	2, 4	<u>2</u> , 4	2, 1	<u>4</u> , 2
Observe	0, 2	1, 2	2, 1	3, 0
Use Codeword	<u>3</u> , 3	0, 1	0, <u>3</u>	1, 1
Ignore	0, 0	<u>2</u> , 4	<u>3</u> , 3	0, 0
	Greet	Observe	Use Codeword	Ignore
Greet	2, <u>4</u>	<u>2</u> , <u>4</u>	2, 1	<u>4</u> , 2
Observe	0, <u>2</u>	1, <u>2</u>	2, 1	3, 0
Use Codeword	<u>3</u> , <u>3</u>	0, 1	0, <u>3</u>	1, 1
Ignore	0, 0	<u>2</u> , 4	<u>3</u> , 3	0, 0

There are exactly three pure Nash equilibria:

(Greet, Observe), (Ignore, Observe), (Use Codeword, Greet).

- (ii) Give all pure strategies on the Pareto front (cf. Definition 8 in the appendix) for the *Identification Game*. (3pts)

All non-Pareto-dominated strategies in the *Identification Game* are Pareto-optimal on the Pareto front, here: (Greet, Greet), (Greet, Observe), (Ignore, Observe) with utilities (2, 4) each, (Greet, Ignore) with utility (4, 2), as well as (Use Codeword, Greet) and (Ignore, Use Codeword) with utilities (3, 3).

- (iii) Upon encounter, the *MoleBots* need some way to pass each other in the narrow burrow. They may execute these actions:

- *turn back*: The executing *MoleBot* turns back in the tunnel and goes back to the last chamber it came from.
- *advance*: The executing *MoleBot* tries to push on, but fails if the path is still blocked (because the other mole has not turned back).
- *teleport*: Through technology beyond imagination, the two *MoleBots* switch their position instantly, iff both choose to execute that action.

Model your own 3×3 symmetric payoff matrix using utilities $(H, L) \in \{0, 1\}^2$ for this *Traversal Game* that allows for a cooperative traversal of the burrow. (6pts)

A simple pure *Traversal Game* could look like this:

	<i>switch</i>	<i>advance</i>	<i>turn back</i>
<i>switch</i>	1, 1	0, 0	0, 0
<i>advance</i>	0, 0	0, 0	1, 0
<i>turn back</i>	0, 0	0, 1	1, 1

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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.) or other variables that are not used. For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ denotes the number of elements in X . For infinite sets, $|X| = \infty$. For any vector \vec{x} , $|\vec{x}|$ denotes the dimensionality of \vec{x} .

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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ 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., $x_0 =_{\text{def}} 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Algorithm 1 (simulated annealing). Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x \in \mathcal{X}$. Let $\kappa : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ with $\mathbb{P} = [0; 1] \subset \mathbb{R}$ be a function that returns an acceptance probability given two target values and a temperature. Commonly, we use

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

for $\mathcal{T} \subseteq \mathbb{R}$ and Euler's number e . The process \mathcal{D} continues via simulated annealing if e is of the form

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

where $x'_t \sim \text{neighbors}(x_t)$ and $r \sim \mathbb{P}$ are drawn at random for each call to e .

Definition 3 (neural network). A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers L_1, \dots, L_r where each layer L_l consists of $|L_l|$ cells $C_{l,1}, \dots, C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, \dots, |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}$, $e = 1, \dots, |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \rightarrow \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|, e=1, \dots, |L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{l=1, \dots, r, c=1, \dots, |L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} \# \mathbf{b}$ is called the network \mathcal{N} 's parameters.

A network's output given an input $\mathbf{x} \in \mathbb{R}^p$ is given via

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \langle O(r, c) \rangle_{c=1, \dots, |L_r|} \in \mathbb{R}^q$$

$$\text{where } O(l, c) = \begin{cases} x_c & \text{if } l = 0, \\ f(B_{l,c} + \sum_{i=1}^{|L_{l-1}|} E_{l,c,i} \cdot O(l-1, i)) & \text{otherwise.} \end{cases}$$

Definition 4 (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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 5 (multi-agent system). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 6 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ for all i . A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game*.

Definition 7 (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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 8 (Pareto front for strategies). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 9 (best response). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

Definition 10 (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 $\pi^{[-i]}$.

Feedback Aufgabe 5

2022-Exam-Coln.pdf

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf

2025-MockExam1-Coln.pdf

2025-MockExam2-Coln.pdf

Gesamtpunkte (ueber alle Aufgaben)

6 Game Theory

10pts

In the lecture we have defined the common notion of a normal-form game like this:

Definition 2 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ for all i . A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game*.

Now let us consider the following new definition:

Definition 3 (mock exam game). A *mock exam game* is a two-player normal-form game written as pair of $m \times n$ matrices (A, B) , where m is the number of rows and n the number of columns. If agent $G^{[1]}$ (the row player) plays row i and $G^{[2]}$ (the column player) plays column j , then $\chi^{[1]} = a_{ij}$ and $\chi^{[2]} = b_{ij}$, where a_{ij}, b_{ij} are the corresponding entries of A and B respectively.

(i) Give one example of a zero-sum, two-player, two-action normal-form game. (1pts)

(ii) Give one example of a zero-sum, two-player, two-action mock exam game. (2pts)

(iii) What are the two key differences between the definitions? (2pts)

(iv) Remember the definition of the key concepts of Pareto-optimality from the lecture:

Definition 4 (Pareto front for strategies). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{[|G|]}(\cdot))$ iff for all agents $G^{[i]}$ it holds that $\chi^{[i]}(\pi(\cdot)) \geq \chi^{[i]}(\pi'(\cdot))$ and there exists some agent $G^{[j]}$ so that $\chi^{[j]}(\pi(\cdot)) > \chi^{[j]}(\pi'(\cdot))$.

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.

Can symmetrical mock exam games of the form

$$A := \begin{array}{cc} X & Y \\ \begin{matrix} X \\ Y \end{matrix} & \left(\begin{matrix} a & b \\ c & d \end{matrix} \right) \end{array} \quad B := \begin{array}{cc} X & Y \\ \begin{matrix} X \\ Y \end{matrix} & \left(\begin{matrix} a & c \\ b & d \end{matrix} \right) \end{array}$$

contain strategies that are also *Pareto-optimal*? If so, give an example for a symmetrical mock exam game with a payoff (defined by $a, b, c, d \in \mathbb{R}$) so that it contains at least one *Pareto-optimal* strategy. If not, show why. (5pts)

6 Game Theory

10pts

In the lecture we have defined the common notion of a normal-form game like this:

Definition 2 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ for all i . A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game*.

Now let us consider the following new definition:

Definition 3 (mock exam game). A *mock exam game* is a two-player normal-form game written as pair of $m \times n$ matrices (A, B) , where m is the number of rows and n the number of columns. If agent $G^{[1]}$ (the row player) plays row i and $G^{[2]}$ (the column player) plays column j , then $\chi^{[1]} = a_{ij}$ and $\chi^{[2]} = b_{ij}$, where a_{ij}, b_{ij} are the corresponding entries of A and B respectively.

- (i) Give one example of a zero-sum, two-player, two-action normal-form game. (1pts)

$$\begin{array}{cc} & \text{L} & \text{R} \\ \text{L} & \left(\begin{array}{cc} 0, 0 & 1, -1 \end{array} \right) \\ \text{R} & \left(\begin{array}{cc} -1, 1 & 0, 0 \end{array} \right) \end{array}$$

- (ii) Give one example of a zero-sum, two-player, two-action mock exam game. (2pts)

For the row player $G^{[1]}$:

$$A := \begin{array}{cc} & \text{L} & \text{R} \\ \text{L} & \left(\begin{array}{cc} 0 & 1 \end{array} \right) \\ \text{R} & \left(\begin{array}{cc} -1 & 0 \end{array} \right) \end{array}$$

For the column player $G^{[2]}$:

$$B := \begin{array}{cc} & \text{L} & \text{R} \\ \text{L} & \left(\begin{array}{cc} 0 & -1 \end{array} \right) \\ \text{R} & \left(\begin{array}{cc} 1 & 0 \end{array} \right) \end{array}$$

(iii) What are the two key differences between the definitions? (2pts)

- Two-players are explicitly assigned their roles as row- and column-player.
- The payoff matrices of these agents are written separately.

(iv) Remember the definition of the key concepts of Pareto-optimality from the lecture:

Definition 4 (Pareto front for strategies). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{[|G|]}(\cdot))$ iff for all agents $G^{[i]}$ it holds that $\chi^{[i]}(\pi(\cdot)) \geq \chi^{[i]}(\pi'(\cdot))$ and there exists some agent $G^{[j]}$ so that $\chi^{[j]}(\pi(\cdot)) > \chi^{[j]}(\pi'(\cdot))$.

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.

Can symmetrical mock exam games of the form

$$A := \begin{array}{cc} X & Y \\ \begin{matrix} X \\ Y \end{matrix} & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \end{array}, \quad B := \begin{array}{cc} X & Y \\ \begin{matrix} X \\ Y \end{matrix} & \begin{pmatrix} a & c \\ b & d \end{pmatrix} \end{array}$$

contain strategies that are also *Pareto-optimal*? If so, give an example for a symmetrical mock exam game with a payoff (defined by $a, b, c, d \in \mathbb{R}$) so that it contains at least one *Pareto-optimal* strategy. If not, show why. (5pts)

Yes, they can. We have already seen such an example in the Hawk–Dove Game, with

$$\begin{array}{cc} & \text{Hawk} \quad \text{Dove} \\ \text{Hawk} & \left(\begin{array}{cc} 0, 0 & 3, \underline{1} \\ \underline{1}, 3 & 2, 2 \end{array} \right) \\ \text{Dove} & \end{array}$$

where (Hawk,Dove), (Dove,Hawk), and (Dove,Dove) are symmetrical *Pareto-Optimal* strategies (two of which also happen to be both Nash equilibria).

As exam game matrices:

$$A := \begin{array}{cc} & \text{X} \quad \text{Y} \\ \text{X} & \left(\begin{array}{cc} 0 & 3 \\ 1 & 2 \end{array} \right) \\ \text{Y} & \end{array} \qquad B := \begin{array}{cc} & \text{X} \quad \text{Y} \\ \text{X} & \left(\begin{array}{cc} 0 & 1 \\ 3 & 2 \end{array} \right) \\ \text{Y} & \end{array}$$

Note: The simplest solution is probably something like

$$a = 100, b = c = d = 0$$

with a Pareto-optimum at (X, X) .

6 Game Theory

10pts

Since cows (and their farmer) can be terribly dangerous animals, the *MowBot* has to think twice how it should behave around the meadow, depending on which behavior it encounters. Within the two-player normal-form game for our true-to-life scenario, we can observe these dynamics unfold in the following payoff matrix for the *Cowmunication Game*:

	Cool Cow	Shy Cow	Default Cow	Brave Cow	Farmer
‘moo?’	2, 2	3, 0	4, 0	4, 2	2, 1
‘brrrr’	4, 1	3, 1	3, 0	4, 4	2, 4
stop	1, 0	0, 1	3, 4	4, 1	2, 4
ignore	3, 0	0, 3	1, 3	0, 0	4, 4

- (i) Compute all pure Nash equilibria (cf. Definitions 6 and 9 in the appendix) for the *Cowmunication Game* by denoting all best responses. (6pts)
- (ii) Give all strategies on the Pareto front (cf. Definition 7 in the appendix) for the *Cowmunication Game*. Give your reasoning! (4pts)

6 Game Theory

10pts

Since cows (and their farmer) can be terribly dangerous animals, the *MowBot* has to think twice how it should behave around the meadow, depending on which behavior it encounters. Within the two-player normal-form game for our true-to-life scenario, we can observe these dynamics unfold in the following payoff matrix for the *Cowmunication Game*:

	Cool Cow	Shy Cow	Default Cow	Brave Cow	Farmer
'moo?'	2, 2	3, 0	4, 0	4, 2	2, 1
'brrrr'	4, 1	3, 1	3, 0	4, 4	2, 4
stop	1, 0	0, 1	3, 4	4, 1	2, 4
ignore	3, 0	0, 3	1, 3	0, 0	4, 4

- (i) Compute all pure Nash equilibria (cf. Definitions 6 and 9 in the appendix) for the *Cowmunication Game* by denoting all best responses. (6pts)

The best responses are....

	Cool Cow	Shy Cow	Default Cow	Brave Cow	Farmer
'moo?'	2, 2	3, 0	4, 0	4, 2	2, 1
'brrrr'	4, 1	3, 1	3, 0	4, 4	2, 4
stop	1, 0	0, 1	3, 4	4, 1	2, 4
ignore	3, 0	0, 3	1, 3	0, 0	4, 4
	Cool Cow	Shy Cow	Default Cow	Brave Cow	Farmer
'moo?'	2, 2	3, 0	4, 0	4, 2	2, 1
'brrrr'	4, 1	3, 1	3, 0	4, 4	2, 4
stop	1, 0	0, 1	3, 4	4, 1	2, 4
ignore	3, 0	0, 3	1, 3	0, 0	4, 4
	Cool Cow	Shy Cow	Default Cow	Brave Cow	Farmer
'moo?'	2, 2	3, 0	4, 0	4, 2	2, 1
'brrrr'	4, 1	3, 1	3, 0	4, 4	2, 4
stop	1, 0	0, 1	3, 4	4, 1	2, 4
ignore	3, 0	0, 3	1, 3	0, 0	4, 4

Thus, the pure Nash equilibria are the joint strategies: (ignore, Farmer) \mapsto (4, 4), ('moo?', Brave Cow) \mapsto (4, 2), ('brrrr?', Brave Cow) \mapsto (4, 4)

- (ii) Give all strategies on the Pareto front (cf. Definition 7 in the appendix) for the *Cowmunication Game*. Give your reasoning! (4pts)

All strategies are Pareto-dominated by the (4,4)-payoff pairs (ignore, Farmer) and ('brrrr?', Brave Cow), which are therefore the only two remaining strategies on the Pareto front.

Feedback Aufgabe 6

2022-Exam-Coln.pdf (nicht in Klausur)

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf (nicht in Klausur)

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf (nicht in Klausur)

2025-MockExam1-Coln.pdf (nicht in Klausur)

2025-MockExam2-Coln.pdf (nicht in Klausur)

Gesamtpunkte (ueber alle Aufgaben)

7 The π -Calculus

10pts

Recall the basic rules of π -calculus from the lecture:

Definition 5 (π -congruence). Two π -processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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

Definition 6 (π -evaluation). An evaluation of a π -process P is a sequence of π -processes $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

(parallel execution) $P | R \xrightarrow{\text{PAR}} Q | R$ if it holds that $P \rightarrow Q$,

(restricted execution) $(\nu x) P \xrightarrow{\text{RES}} (\nu x) Q | R$ if it holds that $P \rightarrow Q$,

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

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

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

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

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

(ii) Consider the following definition for π -processes to model the auction for buying a house:

$$\begin{aligned}
 \text{Bidder}(\text{current}) &= \text{give_house}. \overline{\text{be_happy}}.0 \\
 &\quad + \overline{\text{ask_more}} \langle \text{current} \rangle . (\\
 &\quad \quad \text{approved}(\text{new}). \overline{\text{bid}} \langle \text{new} \rangle . \text{Bidder}(\text{new}) \\
 &\quad \quad + \text{denied}.0 \\
 &\quad) \\
 \text{Seller} &= \text{bid}(\text{"low"}). \text{Seller} \\
 &\quad + \text{bid}(\text{"high"}). \overline{\text{give_house}}.0
 \end{aligned}$$

Give a definition for the π -process *Bank* so that

$$\text{Bidder}(\text{"low"}) \mid \text{Seller} \mid \text{Bank}$$

evaluates to the process 0. Give a short explanation of the behavior of your *Bank* in English or German (4pts).

Note: The behavior of *Bank* does not need to be plausible in a real-world scenario but merely needs to fulfill the requirements by the above definition and the rules of the π -calculus.

7 The π -Calculus

10pts

Recall the basic rules of π -calculus from the lecture:

Definition 5 (π -congruence). Two π -processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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 6 (π -evaluation). An evaluation of a π -process P is a sequence of π -processes $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \rightarrow_{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

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

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

$$\begin{aligned} &\rightarrow b(x).\bar{c}\langle x \rangle.0 \mid \bar{b}\langle 42 \rangle.0 \mid c(z).S(z) \\ &\rightarrow \bar{c}\langle 42 \rangle.0 \mid 0 \mid c(z).S(z) \\ &\rightarrow 0 \mid 0 \mid S(42) \\ &\rightarrow S(42) \end{aligned}$$

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

$$\begin{aligned} &\rightarrow \bar{c}.\bar{b}.0 \mid (\bar{a}.A + \bar{c}.C) \mid A \mid !(a.A + b.B) \\ &\rightarrow \bar{c}.\bar{b}.0 \mid A \mid A \mid A \mid !(a.A + b.B) \end{aligned}$$

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

$$\begin{aligned} &\rightarrow a(x).b(y).S(x,y) + (A(Y) \mid 0) \\ &\rightarrow a(x).b(y).S(x,y) + A(Y) \end{aligned}$$

(ii) Consider the following definition for π -processes to model the auction for buying a house:

$$\begin{aligned} \text{Bidder}(\text{current}) &= \text{give_house}. \overline{\text{be_happy}}.0 \\ &\quad + \overline{\text{ask_more}} \langle \text{current} \rangle . (\\ &\quad \quad \text{approved}(\text{new}). \overline{\text{bid}} \langle \text{new} \rangle . \text{Bidder}(\text{new}) \\ &\quad \quad + \text{denied}.0 \\ &\quad) \\ \text{Seller} &= \text{bid}(\text{"low"}). \text{Seller} \\ &\quad + \text{bid}(\text{"high"}). \overline{\text{give_house}}.0 \end{aligned}$$

Give a definition for the π -process *Bank* so that

$$\text{Bidder}(\text{"low"}) \mid \text{Seller} \mid \text{Bank}$$

evaluates to the process 0. Give a short explanation of the behavior of your *Bank* in English or German (4pts).

Note: The behavior of *Bank* does not need to be plausible in a real-world scenario but merely needs to fulfill the requirements by the above definition and the rules of the π -calculus.

Possibly one of the simplest banks receives an ask for more, always approves "high" and then gives a target for the buyer's happiness before vanishing.

$$\text{Bank} = \text{ask_more}(-). \overline{\text{approved}}(\text{"high"}). \text{be_happy}.0$$

7 Scientific Reading: Model-Based Reinforcement Learning

18pts

After some tweaking, a large language model generated the following (correct) explanation of the established technique *model-based reinforcement learning*, which we did not discuss in the lecture. Read this explanation and answer the questions afterwards!

In the context of model-based reinforcement learning (MBRL), a “model” refers to an abstract representation or approximation of the environment in which the agent operates. This model captures the dynamics of the environment, including how states transition to other states and the rewards associated with these transitions, based on actions taken by the agent.

Model-based reinforcement learning is a sophisticated approach in the field of artificial intelligence, where the system explicitly constructs this model of the environment. Unlike model-free methods, which learn solely from trial-and-error interactions with the environment, MBRL leverages its internal model to simulate outcomes, enabling it to anticipate future states and make more informed decisions. The internal model can be either learned from interactions with the real environment or provided *a priori*.

In MBRL, the model interacts with other components of the learning process in a pivotal way. It serves as a predictive engine that forecasts the next state and potential rewards for given actions. The agent uses this model for planning by simulating actions in a “mental rehearsal” before executing them in the real environment. This ability to predict and evaluate potential future scenarios allows the agent to optimize its policy with fewer real-world interactions, which is particularly valuable in scenarios where experimentation is costly or dangerous.

The core components of an MBRL system include the model of the environment, the policy that dictates the agent’s behavior, and the value function, which estimates the expected return from states or state-action pairs. The environment model is central, providing a bridge between the policy and value function by predicting the outcomes of actions. Learning can occur in two key areas: improving the environment model to better predict future states and rewards, and optimizing the policy and value function based on predictions from the model.

The distinction between model-based RL and simpler value or Q-value approaches (model-free methods) lies in their strategies for improving the reward. Model-free methods learn a direct mapping from states or state-action pairs to values or Q-values, respectively, relying heavily on extensive interaction with the environment to iteratively improve their estimates. They excel in environments where the model is hard to learn, but they often require a large amount of experience to converge to an optimal policy. On the other hand, MBRL seeks to understand the dynamics of the environment through its model, which allows it to anticipate and plan for future states. This foresight enables MBRL to potentially achieve higher rewards with fewer interactions by using its model to explore outcomes of actions without physically executing them, making it more efficient in environments where building a reasonably accurate model is feasible.

(i) In Scenario 1, a cow agent $G^{[7]}$ might want to mirror all actions performed by the much cooler cow agent $G^{[6]}$ except that $G^{[7]}$ wants to execute `eat/mow` whenever $G^{[6]}$ executes `do_nothing` and vice versa. Give a policy $\pi^{[7]}$ for $G^{[7]}$ that uses a *model* $\mu^{[6]}$ of $G^{[6]}$'s behavior. Explain how the usage of a *model* matters in your example. (12pts)

(ii) In Scenario 2, imagine an “irrational math cow” agent whose policy is given by the irrational number $u = \frac{\pi}{10} = 0.314\dots$, i.e., π (the policy) is given by $\frac{\pi}{10}$ (the number). Recall that, for any irrational number, there is no repeating pattern in its decimal digits. Will model-based reinforcement learning be helpful for the other cows to model the behavior of the “irrational math cow” agent? Explain your reasoning. (6pts)

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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.) or other variables that are not used. For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ 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., $x_0 = \text{def } 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Algorithm 1 (simulated annealing). Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x \in \mathcal{X}$. Let $\kappa : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ with $\mathbb{P} = [0; 1] \subset \mathbb{R}$ be a function that returns an acceptance probability given two target values and a temperature. Commonly, we use

$$A(Q, Q', K) = e^{\frac{-(Q' - Q)}{K}} \text{ with Euler's number } e$$

for $\mathcal{T} \subseteq \mathbb{R}$. The process \mathcal{D} continues via simulated annealing if e is of the form

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

where $x'_t \sim \text{neighbors}(x_t)$ and $r \sim \mathbb{P}$ are drawn at random for each call to e .

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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 $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \pi, \langle s_t \rangle_{t \in \mathcal{Z}}, \langle a_t \rangle_{t \in \mathcal{Z}})$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi(_) = (\pi^{[1]}(_), \dots, \pi^{[|G|]}(_))$ Pareto-dominates another joint strategy $\pi'(_) = (\pi'^{[1]}(_), \dots, \pi'^{|G|}(_))$ 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 $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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). Let N be a set of names ($N = \{\text{"a"}, \text{"b"}, \text{"c"}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

(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 $\mathfrak{F} : \mathbb{L}_\pi \rightarrow \wp(N)$ are given inductively via:

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

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

Definition 11 (π -congruence). Two π -processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv : \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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

Definition 12 (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 $x \neq y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{z}\langle z \rangle .(P[y := z])$ for $a = y$ and $x = y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{a}\langle z \rangle .(P[y := z])$ for $a \neq y$ and $x = y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{z}\langle x \rangle .(P[y := z])$ for $a = y$ and $x \neq y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{a}\langle x \rangle .(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(P + Q)[y := z] = (P[y := z]) + (Q[y := z])$,
- $(P | Q)[y := z] = (P[y := z]) | (Q[y := z])$,
- $((\nu x) P)[y := z] = ((\nu x) P)$ 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 13 (π -evaluation). An evaluation of a π -process P is a sequence of π -processes $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

7 Scientific Reading: Model-Based Reinforcement Learning

18pts

After some tweaking, a large language model generated the following (correct) explanation of the established technique *model-based reinforcement learning*, which we did not discuss in the lecture. Read this explanation and answer the questions afterwards!

In the context of model-based reinforcement learning (MBRL), a “model” refers to an abstract representation or approximation of the environment in which the agent operates. This model captures the dynamics of the environment, including how states transition to other states and the rewards associated with these transitions, based on actions taken by the agent.

Model-based reinforcement learning is a sophisticated approach in the field of artificial intelligence, where the system explicitly constructs this model of the environment. Unlike model-free methods, which learn solely from trial-and-error interactions with the environment, MBRL leverages its internal model to simulate outcomes, enabling it to anticipate future states and make more informed decisions. The internal model can be either learned from interactions with the real environment or provided *a priori*.

In MBRL, the model interacts with other components of the learning process in a pivotal way. It serves as a predictive engine that forecasts the next state and potential rewards for given actions. The agent uses this model for planning by simulating actions in a “mental rehearsal” before executing them in the real environment. This ability to predict and evaluate potential future scenarios allows the agent to optimize its policy with fewer real-world interactions, which is particularly valuable in scenarios where experimentation is costly or dangerous.

The core components of an MBRL system include the model of the environment, the policy that dictates the agent’s behavior, and the value function, which estimates the expected return from states or state-action pairs. The environment model is central, providing a bridge between the policy and value function by predicting the outcomes of actions. Learning can occur in two key areas: improving the environment model to better predict future states and rewards, and optimizing the policy and value function based on predictions from the model.

The distinction between model-based RL and simpler value or Q-value approaches (model-free methods) lies in their strategies for improving the reward. Model-free methods learn a direct mapping from states or state-action pairs to values or Q-values, respectively, relying heavily on extensive interaction with the environment to iteratively improve their estimates. They excel in environments where the model is hard to learn, but they often require a large amount of experience to converge to an optimal policy. On the other hand, MBRL seeks to understand the dynamics of the environment through its model, which allows it to anticipate and plan for future states. This foresight enables MBRL to potentially achieve higher rewards with fewer interactions by using its model to explore outcomes of actions without physically executing them, making it more efficient in environments where building a reasonably accurate model is feasible.

- (i) In Scenario 1, a cow agent $G^{[7]}$ might want to mirror all actions performed by the much cooler cow agent $G^{[6]}$ except that $G^{[7]}$ wants to execute `eat/mow` whenever $G^{[6]}$ executes `do_nothing` and vice versa. Give a policy $\pi^{[7]}$ for $G^{[7]}$ that uses a *model* $\mu^{[6]}$ of $G^{[6]}$'s behavior. Explain how the usage of a *model* matters in your example. (12pts)

$$\pi^{[7]}(\dots) = \begin{cases} \text{eat/mow} & \text{if } \mu^{[6]}(t) = \text{do_nothing}, \\ \text{do_nothing} & \text{if } \mu^{[6]}(t) = \text{eat/mow}, \\ \mu^{[6]}(t) & \text{otherwise} \end{cases}$$

with $\mu^{[6]} : \mathbb{N} \rightarrow \mathcal{A}^{[6]}$

The model $\mu^{[6]}$ is what allows $\pi^{[7]}$ to predict agent $G^{[6]}$'s actions, so it is essential to adapt specifically to $G^{[6]}$ in a given situation instead of just learning a policy that “accidentally” matches the desired behavior. This is important, when training data on $G^{[6]}$ is easy to come by.

- (ii) In Scenario 2, imagine an “irrational math cow” agent whose policy is given by the irrational number $u = \frac{\pi}{10} = 0.314\dots$, i.e., π (the policy) is given by $\frac{\pi}{10}$ (the number). Recall that, for any irrational number, there is no repeating pattern in its decimal digits. Will model-based reinforcement learning be helpful for the other cows to model the behavior of the “irrational math cow” agent? Explain your reasoning. (6pts)

Since decimals of π (the number) cannot be predicted from previous decimals, let alone from the position of the digit (as it is done with $\mu^{[6]}$ above), we will have a hard time modeling the crazy cow with traditional models (assuming finite memory, e.g.). Thus, we can assume that model-based reinforcement learning will have a hard time modeling an “irrational math cow” and thus will probably not be helpful.

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., $\mathbb{N} = \{0, 1, 2, \dots\}$ 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 $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ 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.) or other variables that are not used. For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ 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 $\pi : \mathcal{O} \rightarrow \mathcal{A}$. 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 $\tau : \mathcal{X} \rightarrow \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ 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., $x_0 = \text{def } 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Algorithm 1 (simulated annealing). Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\text{neighbors} : \mathcal{X} \rightarrow \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x \in \mathcal{X}$. Let $\kappa : \mathbb{N} \rightarrow \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \rightarrow \mathbb{P}$ with $\mathbb{P} = [0; 1] \subset \mathbb{R}$ be a function that returns an acceptance probability given two target values and a temperature. Commonly, we use

$$A(Q, Q', K) = e^{\frac{-(Q' - Q)}{K}} \text{ with Euler's number } e$$

for $\mathcal{T} \subseteq \mathbb{R}$. The process \mathcal{D} continues via simulated annealing if e is of the form

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

where $x'_t \sim \text{neighbors}(x_t)$ and $r \sim \mathbb{P}$ are drawn at random for each call to e .

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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ 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 $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \pi, \langle s_t \rangle_{t \in \mathcal{Z}}, \langle a_t \rangle_{t \in \mathcal{Z}})$ 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

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \sum R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &= \underset{t \in \mathcal{Z}}{\text{def}} \mathbb{E} \left[\sum \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

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 $G = \{G^{[1]}, \dots, G^{[N]}\}$ 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 $i = 1, \dots, N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times \dots \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, \dots, o^{[N]}) = (a^{[1]}, \dots, a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i .

Definition 5 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ 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 $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ 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 $\pi^{[i]}(\cdot) = a^{[i]}$ 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 $\pi^{[i]}(\cdot) \sim A^{[i]}$ 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 $\pi(\cdot) = (\pi^{[1]}(\cdot), \dots, \pi^{[|G|]}(\cdot))$ Pareto-dominates another joint strategy $\pi'(\cdot) = (\pi'^{[1]}(\cdot), \dots, \pi'^{|G|}(\cdot))$ iff for all agents $G^{[i]}$ it holds that

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

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

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

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 $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, 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

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

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). Let N be a set of names ($N = \{\text{"a"}, \text{"b"}, \text{"c"}, \dots\}$, e.g.). \mathbb{L}_π 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 $\bar{a}\langle x \rangle.P \in \mathbb{L}_\pi$,

(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 $\mathfrak{F} : \mathbb{L}_\pi \rightarrow \wp(N)$ are given inductively via:

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

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

Definition 11 (π -congruence). Two π -processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv : \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$. 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) $P | Q \equiv Q | P$, and $(P | Q) | R \equiv P | (Q | R)$, and $P \equiv P | 0$,

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

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

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

Definition 12 (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 $x \neq y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{z}\langle z \rangle .(P[y := z])$ for $a = y$ and $x = y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{a}\langle z \rangle .(P[y := z])$ for $a \neq y$ and $x = y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{z}\langle x \rangle .(P[y := z])$ for $a = y$ and $x \neq y$,
- $(\bar{a}\langle x \rangle .P)[y := z] = \bar{a}\langle x \rangle .(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(P + Q)[y := z] = (P[y := z]) + (Q[y := z])$,
- $(P | Q)[y := z] = (P[y := z]) | (Q[y := z])$,
- $((\nu x) P)[y := z] = ((\nu x) P)$ 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 13 (π -evaluation). An evaluation of a π -process P is a sequence of π -processes $P_0 \rightarrow \dots \rightarrow P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrow: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i . We define the following evaluation rules:

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

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

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

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

(structural congruence) $P' \xrightarrow{\text{STRUCT}} Q'$ if it holds that $P \rightarrow Q$ 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.

Feedback Aufgabe 7

2022-Exam-Coln.pdf (nicht in Klausur)

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf (nicht in Klausur)

2023-Exam-Coln.pdf

2023-MockExam-Coln.pdf (nicht in Klausur)

2025-MockExam1-Coln.pdf (nicht in Klausur)

2025-MockExam2-Coln.pdf (nicht in Klausur)

Gesamtpunkte (ueber alle Aufgaben)

8 Scientific Reading: The Spi-Calculus

20pts

Read the adapted paper below¹ 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
$\bar{M} \langle N \rangle . P$	output
$M(x).P$	input
$P \mid Q$	composition
$(\nu n) P$	restriction
$!P$	replication
$[M \text{ is } N] P$	match
$\mathbf{0}$	nil
$\text{let } (x, y) = M \text{ in } P$	pair splitting
$\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$	integer case

¹from: 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 $\text{let } (x, y) = M \text{ in } P$, the variables x and y are bound in P . In $\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$, 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 $\bar{M}\langle N \rangle.P$ for $\bar{M}\langle N \rangle.P.\mathbf{0}$. [...]

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

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

Message 1 $A \rightarrow B : M$ on c_{AB}

A first pi-calculus description of this protocol is:

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

The processes $A(M)$ and B describe the two principals, and $Inst(M)$ 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 $Inst(M)$ 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 $Inst(c_{AB})$ 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:

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

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 $Inst(M) \simeq Inst(M')$. 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)

8 Scientific Reading: The Spi-Calculus

20pts

Read the adapted paper below¹ 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
$\bar{M} \langle N \rangle . P$	output
$M(x).P$	input
$P \mid Q$	composition
$(\nu n) P$	restriction
$!P$	replication
$[M \text{ is } N] P$	match
$\mathbf{0}$	nil
$\text{let } (x, y) = M \text{ in } P$	pair splitting
$\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$	integer case

¹from: 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 $\text{let } (x, y) = M \text{ in } P$, the variables x and y are bound in P . In $\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$, 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 $\bar{M}\langle N \rangle.P$ for $\bar{M}\langle N \rangle.P.\mathbf{0}$. [...]

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

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

Message 1 $A \rightarrow B : M$ on c_{AB}

A first pi-calculus description of this protocol is:

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

The processes $A(M)$ and B describe the two principals, and $Inst(M)$ 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 $Inst(M)$ 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 $Inst(c_{AB})$ 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:

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

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 $Inst(M) \simeq Inst(M')$. 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)

Any two of the following examples:

- added constructs for pairings and numbers like (M,N) and 0
- added a successor function that works with numbers
- added matches, pair-splitting and integer case to the process notation
- (obviously) introduced the spi-calculus extension

(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)

(a) The ‘restricted channel’ property $(\nu n) P$ for a process P (to model cryptographically secure channel communication).

(b 1) Authenticity: The secure F process always receives the same input parameter as the bound variable of the c_{AB} channel, making sure that the message always gets ‘passed on’ to the following (secure) process.

and

(b 2) Secrecy: Due to the priority and exclusivity of the c_{AB} channel and the authenticity property, if message M does not get ‘leaked’ (revealed/sent) from process F , then M is always secret to F .

(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)

Changed from $c_{AB}(x).\mathbf{0}$ to $c_{AB}(x).F(x)$ to signify that the information x received over secure channel c_{AB} then actually gets used in / bound to some process F and only F (authenticity / integrity property).

Feedback Aufgabe 8

2022-Exam-Coln.pdf (nicht in Klausur)

2022-MockExam-Coln.pdf

2022-SecondExam-Coln.pdf (nicht in Klausur)

2023-Exam-Coln.pdf (nicht in Klausur)

2023-MockExam-Coln.pdf (nicht in Klausur)

2025-MockExam1-Coln.pdf (nicht in Klausur)

2025-MockExam2-Coln.pdf (nicht in Klausur)

Gesamtpunkte (ueber alle Aufgaben)

COMPUTATIONAL INTELLIGENCE – COMPLETE TERMINOLOGY ENCYCLOPEDIA

A

Term	Full Explanation	Category
Acceptance Probability	Function $A : T \times T \times \mathbb{R} \rightarrow [0, 1]$ in simulated annealing that determines probability of accepting a worse solution. Standard: $A(Q, Q', K) = e^{-(Q' - Q)/K}$	Optimization
Accumulated Reward	Total reward summed over all time steps: $\sum_t R(s_t, a_t, s_{t+1})$	Reinforcement Learning
Action	Element $a \in A$ from action space executed by agent at time t	Agent Theory
Action Space	Set A of all possible actions available to an agent	Agent Theory
Action-Value Function	See Q-Function. $Q : S \times A \rightarrow \mathbb{R}$	RL
Activation Function	Nonlinearity σ applied to neuron outputs (ReLU, sigmoid, tanh)	Neural Networks
Adversarial Training	GAN training where generator and discriminator compete	Deep Learning
Agent	Entity with policy $\pi : O \rightarrow A$ that maps observations to actions	Agent Theory
Aggression	Hawk-Dove game models ecological balance of aggression between peers	Game Theory
AlphaGo	DeepMind AI that beat Lee Sedol at Go in 2016 using MCTS + deep RL	AI History
AND (Fuzzy)	$\min(x, y)$ for fuzzy values $x, y \in [0, 1]$	Fuzzy Logic

B

Term	Full Explanation	Category
Barren Plateau	Flat region in optimization landscape where gradient/signal vanishes	Optimization
Best Response	Strategy π_i^* that maximizes agent i 's payoff given others' strategies π_{-i}	Game Theory
Bias	Offset term b added to weighted sum in neural network neuron	Neural Networks
The Bitter Lesson	Richard Sutton's 2019 essay: search-based methods > human-coded knowledge	AI Philosophy
Boolean	Set $\mathbb{B} = \{0, 1\}$ of truth values	Notation

C

Term	Full Explanation	Category
Channel	Named communication medium in π -calculus (e.g., $a, beep, moo$)	π -Calculus
ChatGPT	Large language model by OpenAI (2022); often used as Turing Test example	AI History
Choice	π -calculus: $P + Q$ nondeterministically executes P or Q	π -Calculus
Concentrations	NOT a neural network component (distractor in exams)	Neural Networks
Concurrency	π -calculus: $P \mid Q$ runs P and Q in parallel	π -Calculus
Congruence	Structural equivalence \equiv in π -calculus (e.g., $P \mid Q \equiv Q \mid P$)	π -Calculus
Cooperate (C)	Action in Prisoner's Dilemma that leads to mutual benefit if both choose	Game Theory
Cost Function	See Reward Function (when minimizing instead of maximizing)	RL
Crossover	Recombination operator in evolutionary algorithms combining two parents	Evolutionary Algorithms
Curvatures	NOT a neural network component (exam distractor)	Neural Networks

D

Term	Full Explanation	Category
Deep Blue	IBM chess AI that beat Garry Kasparov in 1997	AI History
Defect (D)	Non-cooperative action in Prisoner's Dilemma	Game Theory
DeepMind	AI research lab (created AlphaGo, AlphaZero)	AI History
Deterministic	Transition function ($P(s, a) \in \{0, 1\}$): outcomes fully determined	MDP
Discount Factor	$\gamma \in [0, 1]$: weighs future rewards. $\gamma = 0$ =myopic, $\gamma \rightarrow 1$ =farsighted	RL

Term	Full Explanation	Category
Discounted Reward	$\mathbb{E}[\sum_t \gamma^t R(s_t, a_t, s_{t+1})]$	RL
Distance Function	$\text{dist}((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2}$ (Euclidean)	Math

E

Term	Full Explanation	Category
Eat/Mow	Action in MowBot/Cow scenarios (cows eat, bots mow)	Exam Scenario
Emergence	Complex global patterns arising from simple local agent rules	Swarm Intelligence
Empowerment	Agent's influence over environment; measured by $I(A; S')$ (mutual information)	RL / Intrinsic Motivation
Environment Model	In MBRL: learned approximation of $P(s' s, a)$ and $R(s, a, s')$	s,a)) and $R(s, a, s')$ Model-Based RL
Episode	Single run of fixed or variable length from start to termination	RL
Episodic	Task with fixed-length episodes (e.g., $T = 1000$ steps)	RL
Erlang	Programming language for concurrent/distributed systems	Programming
ESS	Evolutionarily Stable Strategy: Nash equilibrium that resists invasion by mutants	Game Theory
Euler's Number	$e \approx 2.718$, base of natural logarithm	Math
Evaluation (π)	Sequence of reductions $P_0 \rightarrow P_1 \rightarrow \dots P_n$ in π -calculus	π -Calculus
Evolutionary Algorithm (EA)	Population-based optimization using selection, mutation, crossover	Optimization
Expected Value	$\mathbb{E}[X]$: probabilistic average of random variable X	Probability
Expert System	1970s-80s AI with hand-coded rules (e.g., MYCIN, DENDRAL)	AI History
Exploitation	Refining known good solutions in optimization/RL	Optimization/RL
Exploration	Searching for new regions of state/solution space	Optimization/RL
Extensional Equality	$\text{equal}(x, y) = \forall p. p(x) \leftrightarrow p(y)$	Logic

F

Term	Full Explanation	Category
Fitness	Objective value $\phi(x)$ in evolutionary algorithms	Evolutionary Algorithms
Flocking	Coordinated movement in swarms (boids: separation, alignment, cohesion)	Swarm Intelligence
Free Names	Names in π -process not bound by input/restriction: $F(P)$	π -Calculus
Friendly Neighbors	Neighbors function with properties: preserves info, differs in actions, symmetric, random	Optimization
Full Observability	Each agent observes complete state: $o_i(t) = s_t$	Multi-Agent
FunSearch	Google DeepMind 2023: LLM + evolutionary search for math/CS discoveries	AI History
Fuzzy Logic	Logic with truth values in $[0, 1]$ instead of $\{0, 1\}$	Fuzzy Logic
Fuzzy Value	Real number in $[0, 1]$ representing degree of membership	Fuzzy Logic

G

Term	Full Explanation	Category
GAN	Generative Adversarial Network: adversarial training (~2015 breakthrough)	Deep Learning
Genetic Algorithm	EA using binary encoding + crossover + mutation	Evolutionary Algorithms
Goal Class	Hierarchy 0-5: Boolean predicates \rightarrow scalar values \rightarrow Q-functions \rightarrow gradients	Agent Theory
Goal Class 0	No explicit goal (random)	Agent Theory
Goal Class 1	Boolean predicate on state/action history	Agent Theory
Goal Class 1.5	Multiple predicates (multi-objective)	Agent Theory
Goal Class 2	Scalar valuation (target function)	Agent Theory
Goal Class 2.5	State-value function $V(s)$	RL
Goal Class 3	Q-function $Q(s, a)$	RL
Goal Class 4	Gradient-based (policy gradient)	RL
Goal Class 5	State-value with gradient	RL
Goal Predicate	$\phi : S^* \rightarrow \mathbb{B}$ or $\phi : S \times A^* \rightarrow \mathbb{B}$	Agent Theory

H

Term	Full Explanation	Category
Happiness	Fuzzy evaluation function in exam (e.g., cow happiness)	Fuzzy Logic
Hawk-Dove Game	Aggression model: two asymmetric NE (Hawk, Dove) and (Dove, Hawk)	Game Theory
Hidden Layer	Intermediate layer in neural network between input and output	Neural Networks

I

Term	Full Explanation	Category
Infinite Meadow	\mathbb{R}^2 continuous 2D plane in MowBot scenarios	Exam Scenario
Input (π-calculus)	$a(x). P$: receive value on channel a , bind to x , continue as P	π -Calculus
Intrinsic Motivation	Reward signal not from environment (curiosity, empowerment, novelty)	RL
Inverse RL (IRL)	Infer reward function from expert demonstrations	RL
Irrational Number	Real number with non-repeating decimal expansion (e.g., $\pi/10$)	Math

J

Term	Full Explanation	Category
Joint Action	Tuple (a_1, \dots, a_N) of all agents' simultaneous actions	Multi-Agent
Joint Observation	Tuple (o_1, \dots, o_N) of all agents' observations	Multi-Agent
Joint Policy	$(o_1, \dots, o_N) \mapsto (a_1, \dots, a_N)$	Multi-Agent

K

Term	Full Explanation	Category
Kasparov, Garry	World chess champion beaten by Deep Blue (1997)	AI History

L

Term	Full Explanation	Category
Large Language Model (LLM)	Neural network trained on text (GPT, Claude, etc.)	Deep Learning
Lee Sedol	Go world champion beaten by AlphaGo (2016)	AI History
Lemonade Stand	Game with non-stationary opponent strategies	Game Theory
Local Optima	Non-global peaks/valleys in optimization landscape	Optimization

M

Term	Full Explanation	Category	
Map Function	$\text{map} : \{0, \dots, 9\} \rightarrow A$ encoding actions by digits	Exam Scenario	
MARL	Multi-Agent Reinforcement Learning	RL	
Markov Decision Process	Tuple (S, A, T, P, R)	RL	
Markov Property	Next state depends only on current state+action, not history	MDP	
Math Cow	Agent whose policy π is encoded by real number $u \in [0, 1]$	Exam Scenario	
MBRL	Model-Based Reinforcement Learning: learns P and R to plan	RL	
Mental Rehearsal	Simulating actions internally using learned model	MBRL	
Micro-Macro Problem	Translating system-level goals into individual agent behaviors	Swarm Intelligence	
Mixed Strategy	Randomized policy over actions according to probability distribution	Game Theory	
Model	Internal representation of environment dynamics ($P(s' s, a)$) and R	s,a)) and R	MBRL
Model-Based RL	See MBRL	RL	
Model-Free RL	Learn value/policy directly from experience (Q-Learning, SARSA)	RL	
MowBot	Robot agent in exam scenarios that mows grass	Exam Scenario	
Multi-Agent System (MAS)	N agents with joint actions, observations, and policies	Multi-Agent	
Mutation	Random modification of individual in evolutionary algorithm	Evolutionary Algorithms	
Mutual Information	$I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x,y)}{p(x)p(y)}$	Information Theory	

N

Term	Full Explanation	Category
Nash Equilibrium (NE)	Joint strategy where every player plays best response to others	Game Theory
Natural Numbers	$\mathbb{N} = \{0, 1, 2, \dots\}$	Notation
Neighbors Function	$N : X \rightarrow \mathcal{P}(X)$ generates nearby solution candidates	Optimization
Neural Network	Computational graph with weights, biases, activation functions	Neural Networks
Non-Stationarity	Opponent/environment changes over time (Lemonade Stand)	Game Theory
Normal-Form Game	Tuple (G, A, T, u) : players, actions, targets, utility	Game Theory

Term	Full Explanation	Category
NOT (Fuzzy)	$1 - x$ for fuzzy value $x \in [0, 1]$	Fuzzy Logic
Null Process	0 in π -calculus: does nothing	π -Calculus

O

Term	Full Explanation	Category
Observation	$o_t \in O$ what agent perceives at time t	Agent Theory
Observation Space	Set O of all possible observations	Agent Theory
Optimization	Searching for $x^* = \arg \min_{x \in X} \phi(x)$ (or $\arg \max$)	Optimization
Optimization Process	Tuple $(X, T, \preceq, e, (x_t))$	Optimization
Optimization Run	Sequence (x_0, x_1, \dots, x_g) of solution candidates	Optimization
OR (Fuzzy)	$\max(x, y)$ for fuzzy values $x, y \in [0, 1]$	Fuzzy Logic
Output (π-calculus)	$\bar{a}(v). P$: send value v on channel a , continue as P	π -Calculus

P

Term	Full Explanation	Category
Parallel (π-calculus)	$P \mid Q$: run P and Q concurrently	π -Calculus
Pareto-Dominates	Strategy dominates if no worse for anyone, better for someone	Game Theory
Pareto Front	Set of all Pareto-optimal strategies	Game Theory
Pareto-Optimal	Strategy not Pareto-dominated by any other	Game Theory
Partial Observability	Agents see different/incomplete views: $o_i(t) \neq s_t$	Multi-Agent
Planning	Using model to simulate/evaluate actions before execution	MBRL
Policy	Function $\pi : O \rightarrow A$ or $\pi : S \rightarrow A$	Agent Theory
Policy Encoding	Representing policy as genotype (e.g., real number for Math Cow)	Optimization
Policy Gradient	Gradient-based RL optimizing $\nabla_\theta \mathbb{E}[R]$	RL
Population	Set of candidate solutions in evolutionary algorithm	Evolutionary Algorithms
Power Set	$\mathcal{P}(X)$: set of all subsets of X	Notation
Prefix (π-calculus)	$\tau. P$: execute silent action, continue as P	π -Calculus
Premature Convergence	EA converges too early to suboptimal solution	Evolutionary Algorithms
Prisoner's Dilemma	Game where (Defect, Defect) is NE but (Cooperate, Cooperate) is better	Game Theory

Term	Full Explanation	Category
Prolog	"Programming Logic": logic programming language	Programming
Pure Strategy	Deterministic policy: always chooses same action	Game Theory

Q

Term	Full Explanation	Category
Q-Function	Action-value function $Q : S \times A \rightarrow \mathbb{R}$	RL
Q-Learning	Model-free RL algorithm learning $Q(s, a)$	RL
Quite (Fuzzy)	Hedge: $\text{quite}(x) = \min(2x, 1)$	Fuzzy Logic

R

Term	Full Explanation	Category
Reaction (π)	$a(x). P \mid \bar{a}\langle v \rangle. Q \rightarrow P\{v/x\} \mid Q$	π -Calculus
Receiving Prefix	$a(x). P$: wait for input on channel a	π -Calculus
Recombination	See Crossover	Evolutionary Algorithms
ReLU	Rectified Linear Unit: $\max(0, x)$	Neural Networks
Replication (π)	$!P$: infinite copies of P . $!P \equiv P \mid !P$	π -Calculus
Restriction (π)	$(\nu x)P$: scope channel x to P	π -Calculus
Return	Total (discounted) reward from time t onward	RL
Reward Function	$R : S \times A \times S \rightarrow T$	RL
Reward Hacking	Agent exploits loopholes in poorly designed reward function	RL
Richard Sutton	RL pioneer, author of "The Bitter Lesson"	AI History

S

Term	Full Explanation	Category
Sample Efficiency	How much data needed to learn (MBRL > model-free)	RL
SARSA	On-policy TD learning: State-Action-Reward-State-Action	RL
Scoping	See Restriction	π -Calculus
Selection	Choosing parents based on fitness in EA	Evolutionary Algorithms
Sending Prefix	$\bar{a}\langle v \rangle. P$: output value v on channel a	π -Calculus
Sigmoid	Activation function $\sigma(x) = 1/(1 + e^{-x})$	Neural Networks

Term	Full Explanation	Category	
Simulated Annealing	SA variant with $A(Q, Q', K) = 1/K$, $T(t) = t$	Optimization	
Simulated Annealing (SA)	Optimization accepting worse solutions with decreasing probability	Optimization	
Singularity	Hypothetical point where AI progress outpaces human understanding	AI Philosophy	
spi-calculus	Extension of π -calculus with cryptographic primitives	π -Calculus	
State	$s_t \in S$: complete description of environment at time t	MDP	
State Space	Set S of all possible states	MDP	
State-Value Function	$V : S \rightarrow \mathbb{R}$: expected return from state	RL	
Stigmergy	Indirect coordination via environment (ant pheromones)	Swarm Intelligence	
Stochastic	Transition function ($P(s')$) s,a) \in [0,1]: probabilistic outcomes		MDP
Strategy	See Policy (in game theory context)	Game Theory	
Structural Congruence	$P \equiv Q$ in π -calculus	π -Calculus	
Substitution (π)	$P\{v/x\}$: replace free x with v in P	π -Calculus	
Swarm	Many similar agents, local rules, shared interest, minimal communication	Swarm Intelligence	
Symmetric Neighbors	$P(\pi' \pi) = P(\pi \pi')$	Optimization	

T

Term	Full Explanation	Category	
τ (Tau)	Silent internal action in π -calculus	π -Calculus	
Target Function	$\phi : X \rightarrow T$ mapping solutions to objective values	Optimization	
Target Space	Set T of objective values (often \mathbb{R})	Optimization	
TD-Error	Temporal Difference error: $\delta = r + \gamma V(s') - V(s)$	RL	
Temperature Schedule	$T : \mathbb{N} \rightarrow \mathbb{R}$ controlling SA exploration	Optimization	
Tit-for-Tat	Prisoner's Dilemma strategy: cooperate first, then mirror opponent	Game Theory	
Transition Probability	$P : S \times A \times S \rightarrow [0, 1]$ or $(P(s'))$		MDP

Term	Full Explanation	Category
Turing Test	Test whether AI can communicate indistinguishably from human	AI History

U

Term	Full Explanation	Category
Uniform Distribution	$x \sim \text{Uniform}(A)$: each $a \in A$ equally likely	Probability
Universal Approximation	Theorem: NNs can approximate any continuous function	Neural Networks
Utility Function	$u : A \rightarrow T$ in normal-form game	Game Theory

V

Term	Full Explanation	Category
Value Function	See State-Value Function	RL
Variation Operators	Mutation + crossover in evolutionary algorithms	Evolutionary Algorithms
Very (Fuzzy)	Intensifier: $\text{very}(x) = x^2$	Fuzzy Logic

W

Term	Full Explanation	Category
Weight	Connection strength w_{ij} between neurons	Neural Networks

X, Y, Z

Term	Full Explanation	Category
α-conversion	Renaming bound variables in π -calculus	π -Calculus
γ	See Discount Factor	RL
π-calculus	Process algebra for modeling concurrent communicating systems	π -Calculus

Additional Specialized Terms from 2023/24 Exam

Term	Full Explanation
43-dimensional state space	$S \subseteq \mathbb{N} \times \mathbb{R}^{2 \times 21}$: time + 21 agent positions (2D each)
Dist function	$\text{dist}((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2}$
Free names $F(P)$	Names not bound by input/restriction in π -process P
Simulated annealing	SA with constant acceptance $1/K$ (no exponential decay)

Extended π -Calculus Evaluation Rules

Rule	Formula
REACT	$a(x).P \mid \bar{a}\langle v \rangle.Q \rightarrow P\{v/x\} \mid Q$
TAU	$\tau.P \rightarrow P$
PAR	If $P \rightarrow P'$ then $P \mid Q \rightarrow P' \mid Q$
RES	If $P \rightarrow Q$ then $(\nu x)P \rightarrow (\nu x)Q$
STRUCT	If $P \equiv P'$ and $P' \rightarrow Q'$ and $Q' \equiv Q$ then $P \rightarrow Q$

Formula Reference Card

Concept	Formula
Accumulated Reward	$\sum_{t=0}^T R(s_t, a_t, s_{t+1})$
Discounted Reward	$\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right]$
Mutual Information	$I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$
SA Acceptance	$A(Q, Q', T) = e^{-(Q' - Q)/T}$
Fuzzy NOT	$1 - x$
Fuzzy AND	$\min(x, y)$
Fuzzy OR	$\max(x, y)$
Fuzzy very	x^2
Fuzzy quite	$\min(2x, 1)$

Notation Cheat Sheet (Complete)

$\mathbb{N} = \{0, 1, 2, \dots\}$	Natural numbers
$\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$	Integers
\mathbb{R}	Real numbers
\mathbb{C}	Complex numbers
$\mathbb{B} = \{0, 1\}$	Booleans
$\mathcal{P}(X)$	Power set of X
$\mathbb{E}[X]$	Expected value
\oplus	Vector/sequence concatenation
$\lfloor w \rfloor$	Floor (nearest integer $\leq w$)
$w \bmod 10$	w modulo 10
\forall	For all
\exists	There exists
\equiv	Structural congruence (π)
\rightarrow	Evaluation step (π)

This **encyclopedia** now contains:

- ✓ **Every term** from all exams (2022-2024)
- ✓ **Every definition** from appendices
- ✓ **Every MCQ concept** ever tested
- ✓ **Extended π -calculus** rules and substitution
- ✓ **All fuzzy operators** (AND, OR, NOT, very, quite)
- ✓ **All game theory** concepts (NE, ESS, Pareto, best response)
- ✓ **All RL variants** (MBRL, MARL, IRL, empowerment)
- ✓ **All optimization** terms (SA, EA, neighbors, temperature)
- ✓ **AI history** (Turing Test, Deep Blue, AlphaGo, GAN, Bitter Lesson)
- ✓ **Exam-specific jargon** (MowBot, Math Cow, Cowmunication Game, simulated annealing)

This is the definitive reference. Print it, memorize it, dominate the exam! ☺