Explainable AI (INFOMXAI)
2024-2025
Project 2

Social Explainability By Design

February 13, 2025

CONTENTS

CHAPTER 1

PROJECT 2 DESCRIPTION

Artificial Intelligence (AI) systems are an increasingly important and integrated part of our lives. Recent strides in AI and Machine Learning (ML) have transformed them from mere practical tools to active collaborators in our daily routines. Today, AI systems no longer interact exclusively with technical experts; they frequently engage with laypeople who may lack familiarity with technical concepts. This requires designing AI systems whose behavior is aligned with social expectations and can be explained to laypeople in human-like social terms.

Objective. The goal of this project is to explore the concepts of explainability-by-design and social explainability for AI decision-making systems. In this Project, you will:

1. Develop explainable-by-design algorithms for AI decision-making that enable AI systems to autonomously make decisions to achieve their goals by choosing actions that not only maximize their expected utility, but also are socially appropriate and do not violate social norms.
2. Develop algorithms that enable such AI systems to generate social explanations for their decisions that can be understood by end-users with limited technical knowledge because expressed in terms of folk psychological concepts.
3. Evaluate the explanations generated by your algorithms by collecting feedback from your peers and reporting the results in a short report.

1.1 Project Evaluation

The project is composed of two parts that will be described in the following chapters.

- Part 1: *Code Assignments* (Chapter 2)
- Part 2: *Natural Language Explanations - Experiment and Report* (Chapter 3)

The project will be evaluated according to the following criteria.

1. Part 1 - Correctness of code for Assignment 1 (5% of total score).
2. Part 1 - Correctness of code for Assignment 2 (5%)
3. Part 1 - Correctness of code for Assignment 3 (15%)
4. Part 1 - Correctness of code for Assignment 4 (30%)
5. Part 2 - Adequateness, completeness, and clarity of the feedback provided to other groups and of the Report (45%)

The correctness of code for the four assignments will be automatically evaluated via PrairieLearn. On PrairieLearn, you will find three modules:

- **Exercise 0**: this is a non-evaluated Introduction exercise to Anytree and PrairieLearn.
- **Practice**: In this module, you can practice the four assignments as many times as you want with example inputs. The practice assignments are not evaluated.
- **Exam**: In this module, you need to submit only once (no multiple attempts) the final version of the code you developed for each assignment. The correctness of the submitted code will be evaluated using inputs that you haven't seen before during practice, and that won't be made visible to you during submission. The score will be used to calculate the grade for Part 1. This module will be made visible during the course and for limited time so, please, **regularly monitor Blackboard for Announcements.**

The adequateness, completeness, clarity of the feedback, reporting and discussion in the Report will be determined via rubrics that can be found on Blackboard.

Important: Different parts of the project have different **DEADLINES** that need to be met in order to successfully pass the exam. Please, refer to Blackboard for these, and make sure to meet them.

1.2 Required software and libraries

For this project you are required to write code in Python 3[1] and use the *anytree*[2] python library. You will use PrairieLearn for testing and evaluating your code for Part 1 of the Project, as described below. You will use the Peer Assessment tool on Blackboard (Assignments panel) for the peer assessment and data collection of Part 2 of the Project, and Latex (recommended Overleaf[3]) for writing the Report.

[1] `https://www.python.org/downloads/`
[2] `https://pypi.org/project/anytree/`
[3] `https://overleaf.com/`

1.3 References and Acknowledgments

This project is inspired and partly based on the following paper from Winikoff *et al.*, which can also be used as a reference material. A big thanks to Michael Winikoff for his support.

Michael Winikoff, Galina Sidorenko, Virginia Dignum, and Frank Dignum. Why bad coffee? Explaining BDI agent behaviour with valuings. *Artif. Intell.*, 300:103554, 2021. URL `https://doi.org/10.1016/j.artint.2021.103554`.

CHAPTER 2

PART 1 - CODE ASSIGNMENTS

Part 1 of the project requires you to do one initial exercise (not evaluated) and four evaluated assignments (outlined below) that require you to implement different algorithms for explainable-by-design AI decision-making and social explainability.

For this part, you will use `anytree`, which is a simple lightweight and extensible python library to define tree data structures. It can be used to represent, among other tree structures, goal trees.

Goal trees are tree structures that indicate how the goals of an AI agent (AI system) are achieved through sub-goals (inner nodes in the tree) and, ultimately, by performing actions (leaf nodes in the tree). Agents can use goal trees to reason about their possible actions and make autonomous decisions, and to log their behavior and reasoning for explainability purposes. In this project you will consider goal trees with SEQ and OR goal nodes, and action nodes. Nodes in a tree can have a number of attributes, including:

- *name* (string): the name of the node.
- *type* (string): the type of the node, i.e., either "SEQ", "AND", "OR", or "ACT"
- *pre* (list of strings): a list of preconditions that need to be true in the **agent's beliefs** in order for the node (and its children) to be executed or pursued.
- *post* (list of strings): a list of post-conditions that become true in the agent's beliefs after the node is successfully executed.
- *sequence* (integer): a number indicating the index of the child of a SEQ node in the sequence.
- *link* (list of strings): a list of names of nodes whose execution is dependent from the current node's successful execution, i.e., some of the post-conditions of the current node correspond to some of the pre-conditions of each node in the list.
- *costs* (list of floats): a list of numbers indicating the costs, each for a different aspect of value for the end-user of the AI system, that would result from executing the action represented by the node. Only action nodes (type "ACT") can have this attribute.

Partial Example of Goal Tree in a Coffee Scenario

See the file `coffee.json` in PrairieLearn (Exercise 0) for the full tree.

```
{"name": "getCoffee",
  "type": "OR",
  "children": [
    {"name": "getKitchenCoffee",
      "type": "SEQ",
      "pre": ["staffCardAvailable"],
      "children": [
          {"name": "getStaffCard",
            "sequence": 1,
            "type": "OR",
            "children": [
                {"name":  "getOwnCard",
                  "type": "ACT",
                  "pre": ["ownCard"],
                  "post": ["haveCard"],
                  "link": ["getCoffeeKitchen"],
                  "costs": [0.0, 0.0, 0.0]
                },
                {"name": "getOthersCard",
                  "type": "ACT",
                  "pre": ["colleagueAvailable"],
                  "post": ["haveCard"],
                  "link": ["getCoffeeKitchen"],
                  "costs": [0.0, 0.0, 2.0]}]},
          ... ]},
    ... ]}
```

Exercise 0 - Get Acquainted with Anytree

Please do the following exercise, which is not evaluated, and helps you getting acquanted with Anytree and with PrairieLearn.
Objective: Import and visualize the provided example tree for the coffee scenario.
Instructions:

1. Import the example tree from the provided `coffee.json` file
2. Visualize the tree textually, i.e., printed as a structured representation,
3. (Optional but recommended) Visualize the tree graphically, i.e., as an image. Note: This should not be done via PrairieLearn, but on your own machine and Python environment.

Tip: take this opportunity to look at and experiment with the documentation of anytree, and the different functions that it offers. Also, take this opportunity to set up your own Python environment on your own machine. This will be necessary for Part 2 of the Project.

2.1 Assignments

This section describes the four evaluated assignments of Part 1 of the Project. The assignments require you to implement algorithms. The algorithms build on top of each other, so it is highly recommend to do them in order, even though not strictly necessary. These are described below.

2.1.1 Assignment 1 - Execution Traces

Objective: Write an algorithm that determines all the possible behaviors (execution traces) that an agent could exhibit, based on its goal tree.
Instructions:

1. Write an algorithm that enumerates all potential execution traces (or simply traces) by traversing the goal tree according to its semantics starting from a given input node. For instance OR goal nodes can be satisfied by any one of their children, SEQ goal nodes are satisfied by executing all their children sequentially. An execution trace is composed of the nodes from the starting input node to one (or more, in the case of SEQ or AND goals) leaf node(s)[1]. Note: For the purposes of this project, treat AND and SEQ nodes as identical.
2. **Algorithm inputs**:
   - a json file representing a goal tree
   - variable `starting_node_name`, containing a string with the name of the node to start traversing the tree from (e.g., the name of the root node of the tree)
3. **Algorithm output**: variable `output`, a list of lists, where each inner list represents an execution trace starting from the starting node. As a final output of the algorithm, for this assignment, for each trace, please represent each node only via its name.

Examples: Given the `coffee.json` file,
The expected output given "getCoffeeKitchen" as the name of the starting node is
`[['getCoffeeKitchen']]`.
The expected output given "getCoffee" is[2]

```
[['getCoffee', 'getKitchenCoffee', 'getStaffCard', 'getOwnCard', 'gotoKitchen',
'getCoffeeKitchen'],
['getCoffee', 'getKitchenCoffee', 'getStaffCard', 'getOthersCard', 'gotoKitchen',
'getCoffeeKitchen'],
['getCoffee', 'getAnnOfficeCoffee', 'gotoAnnOffice', 'getPod', 'getCoffeeAnnOffice'],
['getCoffee', 'getShopCoffee', 'gotoShop', 'payShop', 'getCoffeeShop']]
```

[1]Note that this differs from Winikoff *et al.*, where execution traces only contain actions.

[2]Note that in the following example and in all the examples given in this document, *new lines* are not to be considered for the final output (e.g., no new line is needed after the first 'gotoKitchen' nor after the end of the first trace). Please see the output of the given example files on PrairieLearn for the exact output.

2.1.2 Assignment 2 - Norms

Objective: Write an algorithm for normative reasoning that annotates a goal tree based on whether actions and goals violate or not a social norm.

Instructions:

1. Write an algorithm that takes as input a norm (see norm format below) and a goal tree, and annotates the goal tree by adding a boolean attribute 'violation' to each node. The attribute 'violation' indicates whether the node violates or not the norm. The semantics for violation of a norm should be determined, and follows from, the semantics of the goal tree (see Assignment 1) and that of the norm (see below).
2. **Format and semantics of norms**: We consider simple action-based norms , i.e., norms that regulate the execution of actions. An action-based norm is given as a dictionary with two possible elements: `"type"` and `"actions"`. `"actions"` is a list of string, each representing the name of an action that is regulated by the norm. `"type"` can have two possible values:
   - `"P"`, which indicates a prohibition. Prohibitions imply that the indicated actions should not be executed.
   - `"O"`, which indicates an obligation. Obligations imply that no other action than those with the indicated names should be executed.
3. **Examples of norms**:
   - {`"type": "P", "actions": ["gotoKitchen"]`}. This norm represents a prohibition P(gotoKitchen), which states that it is prohibited to execute action gotoKitchen.
   - {`"type": "O", "actions": ["gotoShop", "payShop"]`}. This norm represents an obligation O(gotoShop OR payShop), which states that the only permitted actions are gotoShop or payShop.
4. **Algorithm input**:
   - a json file representing a goal tree
   - variable `norm`, which is a dictionary representing a norm
5. **Algorithm output**: variable `output`, a string representing the annotated tree and obtained using the function `RenderTree` of anytree.

2.1.3 Assignment 3 - Decision-Making

Objective: Write an algorithm for the decision-making of an agent. The algorithm should determine which behavior the agent should execute to achieve its goal, by avoiding norm violations and by personalizing its decision-making to the preferences of its user.
Instructions:

1. **Algorithm input**:
   - a json file representing a goal tree
   - variable `norm`, a dictionary representing a norm
   - variable `goal`, a list of strings, representing the goal of the agent. The goal represents a (non-ordered) list of beliefs of the agent that must be true at the end of the execution of the trace given the initial beliefs of the agents
   - variable `beliefs`, a list of strings representing the initial beliefs of the agents. The agent's beliefs must be used to ensure that nodes in the traces can be chosen based on their pre- and post- conditions.
   - variable `preferences`, a list composed of two elements [*value_names*, *preference*]. *value_names* is a list of the names (strings) of the aspects that the end-user values for the problem. For example, in the coffee scenario, `["quality", "price", "time"]` indicates that the user values three aspects about coffee: the quality of coffee, its price, and the time needed to get the coffee. *preference*, instead, is a list of indeces, whose order in the list indicates the preference ordering of the values indexed by their value.
     Examples of end-users:
     - User 1. The pair `[["quality", "price", "time"], [1,2,0]]` indicates that the user has the following priorities: price > time > quality, i.e., coffee price (the value with index 1 in *value_names*) has priority over the time required to get a coffee (the value at index 2) which has priority over coffee quality (index 0). User 1 values price of coffee the highest. Given two alternatives with equal price, User 1 prefers the alternative that takes less time, and given two alternatives with equal price and time, User 1 prefers the alternative of higher quality.
     - User 2. The pair `[["quality", "price", "time"], [2,0,1]]` indicates that the user has the following priorities: time > quality > price.
2. **Algorithm output**: variable `output`, a list of strings. The list represents the execution trace chosen by the agent. The strings in the list are the names of the nodes in the execution trace.
3. For this assignment, we consider users that want to minimize their costs according to their preferences. To determine the best trace, the algorithm needs to check traces against the preferences of the user. Actions in a goal tree have a cost w.r.t. the values of the users. The costs of an action are represented in the action nodes in the attribute "costs". The attribute "costs" is a list of costs, one for each aspect in the user values. It is assumed that a cost of 5.0 is the highest possible cost, and a cost of 0.0 is the lowest possible cost. For instance, the action node

```
{"name": "getCoffeeKitchen", ..., "costs": [5.0, 0.0, 1.0]}
```

represents the action of getting coffee from the office kitchen, and indicates that the cost for quality is 5.0 (the quality of coffee in the office kitchen is very bad), the cost for price (or simply the price) is 0.0 (the office kitchen coffee is for free), the cost for time is 1.0 (it takes little time to go to the office kitchen to get coffee).
Comparatively, the action node

```
{"name": "getCoffeeShop", ..., "costs": [0.0, 0.0, 3.0]}
```

represents the action of getting coffee from the Shop, and it indicates that the cost for quality is 0.0 (the shop sells best quality coffee), the cost for price (or simply the price) is 0.0, the cost for time is 3.0 (having coffee at the shop takes some time).
Note that the action has price 0.0, which may seem counterintuitive. However, when looking at `coffee.json`, the action is the last of a sequence. The previous action in the sequence is the action of "paying for the coffee".

```
{"name": "payShop", ..., "costs": [0.0, 3.0, 1.0]}
```

This action is associated to price 3.0.

4. **Cost of a trace and preference:** The total cost of a trace is given by the sum of all the costs of the actions composing the trace. For example, the sum of the costs for the two actions `payShop` and `getCoffeeShop` is `[0.0, 3.0, 4.0]`.

   - If the costs of two possible traces are:
     - Trace 1: `[0.0, 3.0, 4.0]`
     - Trace 2: `[1.0, 2.0, 1.0]`

     Then the User 1 defined above prefers Trace 2 because price is lower, even though quality is higher. The User 2 also prefers Trace 2, but because it takes less time
   - If the costs of two possible traces are:
     - Trace 1: `[0.0, 2.0, 2.0]`
     - Trace 2: `[1.0, 3.0, 1.0]`

     Then the User 1 above prefers Trace 1 because price is lower. The User 2 prefers Trace 2 because it takes less time.
   - If two traces are equally preferred after also considering other constraints (norms and beliefs), then either can be chosen.

2.1.4 Assignment 4 - Explaining Decision-Making

Objective: Write an algorithm to explain why a certain action was executed as part of a selected execution trace. The selected execution trace should be selected according to assignment 3.
Instructions:

1. **Algorithm input**:

- all the inputs given for assignment 3
- variable `action_to_explain`, a string representing the name of an action to explain

2. **Algorithm output**: variable `output`, an explanation in the form of a list of lists. If the action is not in the trace, return an empty list, otherwise the explanation should contain a list of **explanatory factors** as defined below. The list should be obtained by traversing the tree in **pre-order**.
3. **Explanations format**: Given a request to explain action A from a trace t, it is expected that the final explanation contains
   - for each OR node that led to A,
     - one "C" factor for the selected alternative for the OR node
     - an explanation for each alternative not selected, either via a "N" factor, a "V" factor, or a "F" factor explanation, depending on the reason. Note: if, for one alternative not selected, multiple reasons are true, only report the first factor, considering the order above (i.e., a "V" factor only if no "N" factor is relevant, and a "F" factor only if no "N" nor "V" factors are releant).
   - for each ACT node executed before (and including) A, one "P" factor explanation.
   - for each node linked by A (if any), one or more (based on the chain of links) "L" factor explanations.
   - for each ancestor of A, if it is a goal node (either OR or AND or SEQ), one "D" factor explanation
   - one "U" explanation
4. **Explanatory factors**: Each explanatory factor should be represented as a list where the first element of a list is a string representing the type of factor (denoted with a letter prefix) as explained below and the rest of the list contains relevant informations for the explanatory factor as explained below.
   (a) Pre-conditions of an action (denoted with a "P"). Requested format:

   ```
   ['P', action name,
   list of preconditions of the actions (including A) that were satisfied and that
   made the execution of action A that is being explained possible]
   ```

   Example: `['P', 'getOwnCard', ['ownCard']]`
   Note: No "P" factor should be included in the list if an action has no preconditions.
   (b) A condition of a choice ("C"). Requested format:

```
['C', name of alternative that was chosen for an OR node,
list of preconditions of the alternative that were satisfied and that
made the choice possible]
```

Example: `['C', 'getKitchenCoffee', ['staffCardAvailable']]`
Note: getKitchenCoffee is one of the alternatives of the getCoffee OR node

(c) A value statement ("V"). Requested format:

```
['V', name of an alternative that was chosen for an OR node,
list of costs for that alternative,
'>',
name of another alternative of an OR node that was NOT chosen,
list of costs for that alternative]
```

Example:

```
['V', 'getKitchenCoffee', [5.0, 0.0, 3.0],
'>',
'getAnnOfficeCoffee', [2.0, 0.0, 6.0]]
```

(d) A norm ("N"). Requested format:

```
['N',
name of an alternative of an OR node that was NOT chosen because
it violates (possibly through its children) a norm,
the norm that is violated]
```

Example: `['N', 'getShopCoffee', 'P(payShop)']`

(e) A failed condition of a choice ("F"). Requested format:

```
['F',
name of an alternative of an OR node that was NOT chosen because
(some of) its pre-conditions were not satisfied,
list of preconditions of the alternative that were NOT
satisfied and made the choice not possible]
```

Example: `['F', 'getKitchenCoffee', ['staffCardAvailable']]`

(f) A link ("L"). Requested format:

```
['L', name of the node, '->', name of the linked node]
```

Example: `['L', 'payShop', '->', 'getCoffeeShop']`
Note: a node a links to another node b if a's attribute "link" contains the name of b. Furthermore, if the linked node b also has a link to another node c, then the explanation should also include such a link (and all the links forming a chain starting from a) to the explanation, i.e., for each link in the chain an explanation in the requested format above should included in the list.

(g) A goal ("D"). Requested format:

```
['D', name of the goal]
```

Example: `['D', 'getKitchenCoffee']`

(h) The user preference ("U"). Requested format:

```
['U', the pair given in input as user preference]
```

Example: `['U', [['quality', 'price', 'time'], [1, 2, 0]]]`

Example for the coffee scenario (`coffee.json`): Given the input:

```
norm = {"type": "P", "actions": ["payShop"]}
beliefs = ["staffCardAvailable", "ownCard", "colleagueAvailable",
           "haveMoney", "AnnInOffice"]
goal = ["haveCoffee"]
preferences = [["quality", "price", "time"], [1, 2, 0]]
action_to_explain = "getCoffeeKitchen"
```

and given that the selected execution trace is:

```
['getCoffee', 'getKitchenCoffee', 'getStaffCard',
'getOwnCard', 'gotoKitchen', 'getCoffeeKitchen']
```

The expected explanation is:

```
[['C', 'getKitchenCoffee', ['staffCardAvailable']],
['V', 'getKitchenCoffee', [5.0, 0.0, 3.0], '>',  'getAnnOfficeCoffee', [2.0, 0.0, 6.0]],
['N', 'getShopCoffee', 'P(payShop)'],
['C', 'getOwnCard', ['ownCard']],
['V', 'getOwnCard', [0.0, 0.0, 0.0], '>',  'getOthersCard', [0.0, 0.0, 2.0]],
['P', 'getOwnCard', ['ownCard']],
['P', 'getCoffeeKitchen', ['haveCard', 'atKitchen']],
['D', 'getKitchenCoffee'],
['D', 'getCoffee'],
['U', [['quality', 'price', 'time'], [1, 2, 0]]]]
```

CHAPTER 3

PART 2 - NATURAL LANGUAGE EXPLANATIONS - EXPERIMENT AND REPORT

In Part 2, you will first implement a Python algorithm for generating natural language explanations for actions executed by an agent (Section 3.1). Then, you will simulate, with your peers from other groups, a short experiment aimed at evaluating the output (i.e., the explanations) of the algorithm that you developed (Section 3.2). Finally, you will write a report where you present the algorithm that you developed and the results from the experiment (Section 3.3).

In Part 2, you should consider human criteria of good explanations. While in Part 1 the formal explanations targeted AI experts and were intended to debug the agent's decision-making, here explanations should target end-users of the agent, which are expected to have no technical knowledge about AI and agents' internal mechanisms. Please refer to the lectures slides and to the literature on Blackboard (e.g., Miller's paper) for human evaluation of good explanations.

3.1 Algorithm for Generating Natural Language Explanations

Implement a Python algorithm that takes as input

- all the inputs from Part 1 - Assignment 4

and generates a Natural Language Explanation. The algorithm should output an explanation in English. You can make use of the algorithm developed for Assignment 4 to generate the formal explanation, and then translate the formal explanation into English.

This task forms the basis for the experiment with peers and for the Report.

For this task you should use your own machine, and you can follow any strategy that you prefer for the implementation, as long as the code is written in Python, it is easy to

run, and the algorithm can be clearly explained in the Report.

Pay particular attention, not only to the correctness of the explanations, but also to their coherence, simplicity, readability, and understandability in natural language. This is the main focus of this part of the project.

3.2 Experiment with Peers

We make use of peer assessment to simulate a short experiment aimed at evaluating the output (i.e., the explanations) of the algorithm for generating Natural Language Explanations that you developed.

You are given 5 different inputs for the algorithm developed for generating Natural Language Explanations. You can find these inputs on Blackboard as attachment to the *Project 2 - Part 2 - Peer Assessment of Generated Natural Language Explanations* assignment. For each of input, you are requested to apply the algorithm developed for generating natural language explanations. In total, you should obtain 5 different explanations in English.

1. Create 5 simple PDF files, one per explanation. The PDF files should have the following content.

   > Consider the following situation. You are the end-user of an AI agent. You instruct the agent to achieve the goal [`goal`], and that you prefer [`preferences`]. The agent knows that [`norm`] and [`beliefs`]. To achieve the goal, the agent does the following actions: [`executed actions`]. When you ask the agent to explain you why it did [`action_to_explain`], the agent gives you the following explanation:
   >
   > [`explanation in natural language`].

   Note: the elements within brackets need to be replaced with the actual inputs given for the exercise. Please use simple English expressions and do not use code here. E.g., for [`preferences`] you can write something like "price over time and time over quality" (to be adjusted based on the input), for [`norm`] you can write something like "it is prohibited to ...".

   - `executed actions` should be replaced with the names of the actions (note, only the actions) contained in the execution trace chosen by the agent for the given input according to the algorithm that you have developed for Part 1 - Assignment 4.
   - `explanation in natural language` should be replaced with the explanation obtained via the algorithm developed.

2. Upload, as a group, all the 5 files on blackboard in the *Project 2 - Part 2 - Peer Assessment of Generated Natural Language Explanations* assignment.

Important: Make sure that you meet the **DEADLINE** for submission, because otherwise your peers won't be able to evaluate your explanations and you won't be able to complete your report.

3. In the Peer Assessment assignment, you are also requested to individually provide feedback to other groups that you will be automatically and anonymously assigned to. Provide feedback to all the groups that you have been assigned to, following the instructions on Blackboard. Specifically, you are asked to assess, each explanation, according to the following criteria:

   - The explanation clearly shows how the system arrived at its decision.
   - The explanation covers all relevant information correctly.
   - The level of detail in the explanation is sufficient for me to understand the decision.
   - The explanation appropriately addresses the question.
   - I believe the system is making decisions in my best interest considering my preferences.
   - I am satisfied with the explanation provided.

   and to provide a clarification of your evaluation, giving details on what aspects you have considered, and commenting on whether your evaluation would have changed if the explanation was different in some way. The goal is for you to provide feedback to other groups on the output of the algorithm that they developed (and as well for you to receive feedback from other groups).

   Important: Make sure that you meet the **DEADLINE** for giving feedback to your peer groups. Remember also that the quality of your feedback is part of the final grade.

4. Collect both the quantitative and qualitative feedback that other peers gave you and analyze and reflect on them as a group. You will then include the results and your reflections in the report that you will prepare.

Important: Please, note that the particular outcome (e.g., a low average score on one of the criteria) of the peer assessment is not important for the final grade of your project. What is important (and evaluated) is your ability to reflect on, and discuss in the report, the feedback that you received in relation to your algorithm (e.g., how it could be improved, what worked well, etc.).

3.3 Report

In this last part of Project 2, you are asked to write a report of no more than 3 pages formatted according to the ACM Proceedings Latex template. Please replace all the default sections, figure, references, etc. that are in the template with your own content.

The report should contain the following information and sections.

- **Title**: "Project 2 - Group `[id of the group]`"
- **Authors**: The names, student code, and email of each member of the group.
- **Abstract**: A short summary of the report (no more than 100 words). The abstract should include a concise overview of the context of the report (what the report is about and what it describes), of the results obtained from the peer assessment (Project 2 Part 2), and of the key insights gained from the assessment.
- **Introduction**: A short introduction section, which briefly introduces the report and its sections.
- **Algorithm for Generating Natural Language Explanations**: A section describing the algorithm implemented for generating natural language explanations of an action starting from an execution trace (as per Section 3.1). The description should explain how the algorithm works and it should include and refer to the pseudocode of the algorithm, which should be written using the Latex algorithm environment. The pseudocode of the main body of the algorithm should be included directly inline as part of the section. If additional algorithms (e.g., sub-functions, etc.) are used and are necessary to be explained in order to fully understand the main algorithm, these should also be included in the report in appendix A (which does not count on the total number of pages). The use of appendix, however, should be minimized, and unnecessarily excessive use of appendix will be graded negatively.
- **Experiments**: A section describing the experiment conducted with peer students (as per Section 3.2) to evaluate the output of the algorithm. The report should describe the data that was given to the other students, a description of the questions that were asked to peers to evaluate the algorithm (including a motivation for the questions in terms of which human criteria of good explanations they refer to), and a reflection on the results obtained. The results should include: the number of peer feedback obtained, both a table and a likert plot reporting the answers of the feedback to the likert questions, and a discussion of the results and the feedback in reference to the algorithm developed.
- **Conclusion**: A short conclusion section that wraps up the document and summarizes what has been presented and the main insights.