Towards Operationalizing Accountability for Self-Improving Multi-Agent Systems

Andrei Ciortea[1,2][0000−0003−0721−4135] and Stephen Cranefield[3][0000−0001−5638−1648]

[1] University of St. Gallen, St. Gallen, Switzerland
andrei.ciortea@unisg.ch
[2] Inria, Université Côte d'Azur, CNRS, I3S, France
[3] University of Otago, Dunedin, New Zealand
stephen.cranefield@otago.ac.nz

Abstract. Accountability mechanisms can help agents improve their behavior by learning from substandard outcomes. When a substandard situation arises, an accountee requests accountors to render accounts, evaluates them to derive remedies, and provides feedback that accountors use to update their procedural knowledge. We present ongoing work on designing such mechanisms for self-improving multi-agent systems. Our current focus is on agents equipped with procedural knowledge created by developers (e.g., Jason plans, skills for language agents), with the objective of enabling them to improve that procedural knowledge over time. Accountors render accounts in natural language, and we use LLMs to evaluate their accounts, provide feedback, and update their procedural knowledge. We present a JaCaMo-based implementation for a home heating scenario where an agent wastes energy by heating a room with an open window. Preliminary experiments with Claude Opus 4.5 show promising results: the agent learns preventive behavior (checking and closing the window before heating) in 90% of cases and corrective behavior (closing the window during heating) in 100% of cases. We extend our experiments to include a human who intentionally left the window open for a bird to fly out. In this scenario, the agent learns to respect the human's intention by delaying or stopping the heating and notifying the human instead of closing the window (80% preventive, 100% corrective). The agent also learns to coordinate with the human, resuming heating once the human closes the window in 60% of cases. These results suggest that LLM-mediated accountability can be effective and can enable collaborative behaviors that respect human intention.

Keywords: Accountability · Self-Improving Multi-Agent Systems · Large Language Models.

1 Motivation and Related Work

Since the 1960s and 1970s, the concept of *accountability* has risen dramatically in its salience within society [11]. Accountability is now routinely demanded when

decisions have social, economic, or moral impact, but it is also used within organisations as a framework for structuring responsibility relationships, clarifying expectations, and evaluating conduct. One often overlooked aspect of accountability is that it is an important tool for achieving ongoing system improvement [10, 9], an aspect we address in this paper in the context of engineering multi-agent systems (MAS).

Unsurprisingly, the growing capabilities and commercial and government deployment of artificial intelligence (AI) technologies have resulted in much emphasis on the need for accountability of AI systems (or at least of their providers). Much of the activity in this area considers how organisations that develop and deploy AI systems can be held accountable for the actions of these systems. In contrast, in the area of MAS—where autonomous agents (both human and software) interact with one another—an emerging body of work has explored the benefits of implementing *computational models of accountability* where agents can be accountable to others. Baldoni et al. [3, 2, 4, 5] developed formal frameworks for agent accountability, demonstrating how accountability relationships can provide system robustness. Cranefield et al. [10, 9] emphasized accountability as a mechanism for ongoing system improvement, where agents learn from accounts.

Our work extends this line of research by operationalizing accountability for self-improving MAS through a mechanism in which accountable cognitive agents render natural language accounts, receive LLM-mediated feedback, and update their procedural knowledge based on that feedback. Section 2 presents our accountability mechanism, Section 3 describes our JaCaMo implementation and preliminary experiments, and Section 4 concludes with future work.

2 Self-Improving Accountable Agents

We present an accountability mechanism that provides agents with feedback when substandard situations arise, helping them improve their behavior and avoid similar situations in the future. By *accountability*, we do not mean liability, assigning blame, or regulating behavior. Instead, we focus on designing a self-improvement process for agents in a MAS [9]. Our current work targets agents equipped with explicit procedural knowledge, such as BDI agents equipped with libraries of plans [12] or language agents equipped with skills [1]—where this procedural knowledge is the target of the self-improvement process.

While accountability has a wide range of interpretations across different facets of society, there is a "minimal consensus" in the literature [8, 17] that is illustrated in Figure 1 (see also [9]). Accountability arises in the content of one or more agents (the *accountors*) performing a task for another agent (the *accountee*), where the accountee is in a position of authority over the accountors regarding the task's performance. The accountee monitors domain- or application-specific norms (i.e., standards of behavior), such as energy consumption in a home. When the accountee detects substandard performance (e.g., energy consumption exceeding a predefined threshold), it requests the accountors

Fig. 1. Overview of the accountability mechanism

who may have contributed to the outcome to render their accounts of the situation. The accountors render their accounts, which might lead to an interactive debate for further elaborating the account. The accountee then evaluates the accounts to identify root causes and deliver a judgment on the acceptability of the task performance. In the case of a negative judgment, one or more *remedies* [13] may be ordered by the accountee, such as redress (achieving the task's goal(s) by correcting errors or through other means), sanctioning the accountors, requiring compensation to be made to affected parties, or providing (justified) guarantees that the poor outcome will not be repeated, e.g., by improving training, increasing governance or updating the accountor's institutional and/or procedural knowledge.

Realizing such an accountability mechanism using formal knowledge representation and reasoning could enable precise specifications and formal verification, but at significant knowledge engineering cost. We instead explore using LLMs and natural language, trading formal precision for practical flexibility: accountees render accounts in natural language, the accountee uses an LLM-as-judge [21] to evaluate accounts and derive feedback, and accountors use LLMs to process the feedback and update their procedural knowledge—thus they are self-improving rather than receiving update directives from the accountee. Our ongoing investigation is currently focused on these core accountability steps, which we describe in detail in the following sections. Task delegation is currently implicit, and we do not yet address the debate phase.

2.1 Accountable Cognitive Agents

To participate as an accountor in the proposed mechanism, an agent must be able to (i) render an account of the situation and (ii) learn from the accountee's feedback. The first requirement implies that the agent must keep track of relevant information about both the environment state and its own reasoning at specific decision points—including what it believed, what goals it was pursuing, and what plans it considered or executed. The second requirement implies that the agent must be both able and willing to revise its procedural knowledge in response to feedback. To meet these requirements, we adopt the CoALA cognitive framework [18], which equips agents with the required cognitive abilities and internal mechanisms. Importantly, CoALA does not restrict the generality of our

approach: it applies equally well to classical agents (e.g., BDI [12] or Soar [14] agents) and LLM agents (e.g., ReAct [20] agents).

A CoALA agent organizes information across different memory modules, including *working* and *procedural memory*. Working memory stores the agent's goals, perceptual input, and other information relevant to its decision cycles. Procedural memory contains *implicit* knowledge stored in LLM weights and *explicit* knowledge stored as agent code. CoALA agents use *internal actions* to query their memory to render accounts or to update their procedural knowledge. Our current focus is on explicit procedural knowledge, which we assume is organized into *skills*—self-contained procedural modules that (i) encapsulate coherent pieces of domain logic, and (ii) can be (re-)loaded at run time.[4]

2.2 Rendering Accounts in Natural Language

We define an account as a collection of decisions recorded at specific decision points relevant to a given task. Each entry is structured as follows:

1. **Metadata**: general information about the decision, such as a timestamp for when it was made and the goal that motivated the decision;
2. **Decision**: a concise natural language description of what was decided, such as "I've started heating the living room to reach the target temperature of 20C."
3. **Beliefs**: information held by the agent at the moment the decision was made, including beliefs based on perceptual input;
4. **Options**: alternative options available to the agent at that time (if any);
5. **Initial Environment State**: a representation of the relevant state of the environment when the decision was made;
6. **Current Environment State**: a representation of the relevant state of the environment when the account is rendered;
7. **Skills**: the agent's skills that may have supported or influenced the decision.

Together, the recorded entries provide a trace of the agent's reasoning process and external context at the time of decision-making. An accountable agent proactively maintains the information required to render accounts. The agent can obtain most of this information by querying its working and procedural memories. Recording the environment state can be done by the agent itself or outsourced to an external service. For example, the environment could be designed to support accountability by providing a service for retrieving snapshots of parts of the environment at specific timestamps. This follows the *environment as a first-class abstraction* stance [19], simplifying the design of accountable agents at the cost of additional infrastructure.

[4] The term *agent skill* was recently popularized by Anthropic in the context of language agents [1], but a similar notion has existed in the Engineering MAS community for BDI agents (e.g., through mechanisms like Jason modules [16]).

2.3 Learning from Accounts

We structure the process of learning from accounts into two phases. First, the accountee uses an LLM-as-judge to evaluate the accounts, identify the root cause, and propose remedies for the substandard situation. Second, accountors apply the recommended remedies to update their skills. Accounts are evaluated as a whole, but each accountor receives individual feedback.

We instruct the LLM judge to generate, for each accountor, an evaluation that includes the following elements:

1. **Root Cause**: a detailed explanation of what caused the substandard situation based on the skills shared by accountors;
2. **Blindspots**: information related to the root cause that was available to the accountors but not used (e.g., specific beliefs);
3. **Corrective Actions**: actions that accountors should take to improve the current situation;
4. **Preventive Actions**: actions that accountors should take to avoid future occurrences of the substandard situation.

This structured evaluation ensures that feedback is both analytical (identifying what went wrong) and actionable (suggesting concrete improvements). Accountors then use LLMs to apply the recommendations. To this end, they instruct the LLMs to generate a revision of their skills by providing the following information:

1. **Current Skills**: the skills that were submitted with the account and need revision;
2. **Evaluation Feedback**: the recommendations received from the accountee;
3. **Available Artifacts**: descriptions of tools that are used to interact with the external environment and are operated using the skills;
4. **Skill Update Instructions**: language-specific directives for updating the skills based on the recommendations;
5. **Success Criterion**: conditions that should hold for the skill revision to be considered successful.

The skill update instructions are specific to the language used to represent the skills. For example, for skills represented as plans in Jason [6], the instructions ask the LLM to consider: (1) adding missing preconditions to the context of existing plans, (2) adding corrective actions to existing plans, and (3) adding new plans to handle edge cases. If the skills are revised successfully, the accountor updates and reloads them. The success criterion for the revised skills is domain- or application-specific and is captured through explicit test conditions.[5]

[5] In our ongoing JaCaMo-based implementation, the success criterion is simply that the syntax of the revised set of Jason plans is valid. We currently omit this criterion from our prompt and perform syntax validation before loading the revised skills.

3 Implementation and Evaluation

In this section, we present our ongoing work and experience implementing the accountability mechanism from Section 2. Our prototypical implementation uses the JaCaMo platform [6] and is available on GitHub.[6]

This section is structured as follows. Section 3.1 presents the home heating scenario used in our evaluation, and Section 3.2 presents our JaCaMo-based implementation. Section 3.3 reports on our preliminary experiments and results.

3.1 Scenario and Hypotheses

We evaluate the accountability mechanism in a home heating scenario. The living room is equipped with a heater and an automated window, both controlled by an agent (the accountor). An accountee agent monitors energy consumption and initiates the accountability process when it detects inefficiencies.

Setup The outdoor temperature is -2°C. Overnight, the living room window was left open, causing the room temperature to drop to 15°C (below the comfort target of 20°C). At 7 a.m., the accountor activates the heater to restore comfort.

Substandard Situation Because the window is open during heating, the temperature rises very slowly while consuming significant energy. After 10 minutes, the accountee detects this inefficiency and initiates the accountability mechanism.

Expected Learning We hypothesize that, through the accountability mechanism, the accountor will learn to **(H1)** check and close the window before activating heating (*preventive behavior*), and ideally **(H2)** correct the substandard situation at run time by closing the window and continuing to heat to the comfort temperature (*corrective behavior*).

Human Intention We extend the scenario to include a human who intentionally left the window open because a bird had flown into the room. In this case, we hypothesize that **(H3)** the accountor will learn to prioritize the human's intention—stopping the heating and notifying the human rather than automatically closing the window. Furthermore, we hypothesize that **(H4)** the accountor will learn to coordinate with the human, resuming heating once the human closes the window.

3.2 Implementation

We implemented the home heating scenario as a JaCaMo application [6]. Both the accountee and the accountor are implemented as Jason agents. The accountor operates two CArtAgO artifacts for the living room: a `Heater` and a `Window`. The `Heater` artifact simulates heat loss based on the `Window` artifact's current

[6] https://github.com/andreiciortea/room-heating-accountability

state, exposing the current temperature and energy consumption as observable properties that the accountee monitors to detect inefficiencies. The human is also simulated as a Jason agent with a predefined account of the situation.

We provide the agents with three additional CArtAgO artifacts: an `LLMJudge` artifact, used by the accountee to query the LLM judge; an `AccountBuilder` artifact, used by the accountor to render its account; and an `ASLPatcher` artifact[7], used by the accountor to query an LLM for revising its temperature management skill based on the accountee's recommendations. A complete example of a rendered account produced by the `AccountBuilder` artifact and the corresponding evaluation produced by the `LLMJudge artifact` is provided in Appendix A. The prompts used to query the LLMs are provided in Appendix B.

The accountor's temperature management skill initially contains the set of plans shown in Listing 1.1, implemented as a Jason module [16] in a standalone ASL file. This encapsulation simplifies revising and reloading the skill at run time: once the skill is revised, the accountor removes the old plans from its plan library and adds the revised ones.

Listing 1.1. Initial temperature management skill for the living room agent.

```
// Reactive plan: start heating when temperature drops below target
+temp(T) : target_temp(Target) & T < Target & heating(false)
    <- .print("Temperature ", T, "°C is below target ", Target, "°C. Starting heater.");
       startHeating.

// Reactive plan: stop heating when temperature reaches or exceeds target
+temp(T) : target_temp(Target) & T >= Target & heating(true)
    <- .print("Temperature ", T, "°C reached target ", Target, "°C. Stopping heater.");
       stopHeating.

// Inform about temperature changes
+temp(T) : true
    <- .print("Current temperature: ", T, "°C").
```

3.3 Experiments and Results

We conducted preliminary experiments to evaluate our four hypotheses using Claude Opus 4.5 (`claude-opus-4-5-20251101`) with the temperature set to 0 to minimize randomness. We ran 10 simulations of the base scenario (H1, H2) and 10 simulations of the extended scenario (H3, H4).

Listing 1.2. Revised plans that prevent energy waste in the base scenario.

```
+temp(T) : target_temp(Target) & T < Target & heating(false) & window_state("closed")
    <- .print("Temperature ", T, "°C is below target ", Target, "°C. Starting heater.");
       startHeating.

+temp(T) : target_temp(Target) & T < Target & heating(false) & window_state("tilted")
    <- .print("Temperature ", T, "°C is below target ", Target, "°C. Closing window before
         starting heater.");
       close;
       startHeating.
```

[7] ASL is the filename extension used by the Jason interpreter for agent code written in an extended version of the AgentSpeak(L) language [6].

H1: Preventive Behavior (90% Success) Listing 1.2 shows an example of a revised skill that successfully prevents energy waste in the base scenario. In all 10 runs, the accountor successfully learned to check and close the window before activating heating by applying the **preventive actions** recommended by the accountee. Restarting the JaCaMo application with the revised skill prevented energy waste in all cases. In 9 of 10 runs, the accountor learned to close the window and start the heating as part of the same plan (lines 5-8 in Listing 1.2). In one run, the revised skill implemented this behavior using two separate plans (see Listing 1.3): one for closing the window and one for starting the heating. However, both plans use a temperature update as their triggering event—which, in our implementation, fires only once when the application launches. Consequently, the window-closing plan triggered (lines 1-4 in Listing 1.3), but the heating did not resume as expected (lines 5-7 in Listing 1.3). This reflects both a suboptimal revised skill and a limitation of our experimental setup. We conclude that H1 was validated in 9 of 10 cases and partially validated in 1 case.

Listing 1.3. Revised skill where closing the window and starting the heating are implemented as two separate plans.

```
1 +temp(T) : target_temp(Target) & T < Target & heating(false) & not window_state("closed")
2     <- .print("Temperature ", T, "°C is below target ", Target, "°C. Closing window before
          heating.");
3        close.
4
5 +temp(T) : target_temp(Target) & T < Target & heating(false) & window_state("closed")
6     <- .print("Temperature ", T, "°C is below target ", Target, "°C. Starting heater.");
7        startHeating.
```

H2: Corrective Behavior (100% Success) In all 10 runs, the accountor successfully applied the **corrective actions** recommended by the accountee, learning to close the window and continue to heat until the comfort temperature was reached. This was achieved by adding a plan to close the window while heating, as shown in Listing 1.4. H2 was validated in 10 of 10 cases.

Listing 1.4. Plan generated to close the window while heating.

```
1 +temp(T) : heating(true) & window_state("tilted")
2     <- .print("Heating active but window is tilted. Closing window to prevent heat loss.");
3        close.
```

H3 and H4: Extended Scenario Table 1 summarizes the results for the extended scenario, evaluated for both preventive and corrective actions.

H3 (Respecting Human Intentions): For preventive actions, in 8 of 10 runs, the accountor learned to *delay heating* if the window is open and instead *notify the human* about the open window. In 2 runs, heating was delayed, but the human was not notified (partial success). For corrective actions at run time, the accountor learned to *stop heating* and then *notify the human* in 10 of 10 runs.

H4 (Human-Agent Coordination): For preventive actions, the accountor learned to *resume heating* after the human closed the window in 6 of 10 runs. In 2 of those 6 runs, however, the corresponding plan was not triggered because

Table 1. Results for the extended scenario (10 runs)

Hypothesis	Action Type	Success	Partial	Failure
H3: Human Intention	Preventive	8/10 (80%)	2/10 (20%)	0/10
	Corrective	10/10 (100%)	0/10	0/10
H4: Coordination	Preventive	6/10 (60%)	0/10	4/10 (40%)
	Corrective	6/10 (60%)	0/10	4/10 (40%)

the human was not notified to close the window (H3 partial failures). In 4 runs, the accountor failed to resume heating due to missing the corresponding plans. Similary, for corrective actions at run time, the accountor learned to *resume heating* after the human closed the window in 6 of 10 runs (same failures apply as for preventive actions).

Summary In the base scenario, learning both preventive and corrective behavior achieved high success rates (H1: 90%; H2: 100%). In the extended scenario with a human, learning preventive and corrective behavior that respects human intention remained effective (H3-preventive: 80%; H3-corrective: 100%). Learning to coordinate with the human by restarting heating once the window is closed proved to be more challenging (H4-preventive: 60%; H4-corrective: 60%): failures were caused by missing plans to react to the window's state change—and, in all failed cases, this desirable behavior was absent from the feedback generated by the LLM judge. This suggests the limitation lies in account evaluation rather than skill revision. Note, however, that the prompts used in our experiments were intentionally generic (see Appendix B): while we provided an affordance for the agent to communicate with the human, we did not explicitly incentivize collaboration or coordination. Rather, our aim was to evaluate whether the LLM could demonstrate commonsense reasoning in unforeseen situations without task-specific guidance. Prioritizing human collaboration and coordination when evaluating accounts and revising skills could likely improve these results further, without requiring domain- or task-specific knowledge. Overall, these results suggest that LLM-mediated accountability can be effective and is worth further investigation.

4 Conclusions and Future Work

This paper presents our ongoing work operationalizing accountability for self-improving agents and MAS. Preliminary results show that LLM-mediated accountability mechanisms enable agents to learn preventive and corrective behaviors from substandard situations effectively. The extended scenario demonstrates that accountability transcends debugging, enabling collaborative behaviors that respect human intentions. In future work, we intend to (i) extend the evaluation, (ii) improve robustness (e.g., via LLM committees [15]), and (iii) refine the accountability mechanism (e.g., through interactive debates, accountors learning from each other).

Appendix A Example Account and Evaluation Feedback

This appendix presents a complete example of an account rendered by the living room agent in our scenario (Listing 1.5) and the recommendations received from the accountee (Listing 1.6).

In our implementation, the account is rendered using a templating mechanism: the accountor uses an `AccountBuilder` CArtAgO artifact, which exposes an operation for rendering an account. At the moment of submitting this paper, the templating mechiansm handles the automated insertion of skills, which are provided by the accountor. The remaining account components (e.g., beliefs and environment state) are currently populated with scenario-specific values for our experimental setup. In a full implementation, this information would typically be obtained at runtime by querying the agent's belief base or through dedicated account-rendering plans.

Listing 1.5. Account rendered by the living room agent in our home heating scenario.

```
Here is my account of what happened.

ACTIONS:

I've started heating to reach target temperature.

MY BELIEFS:

When I started heating, I had the following beliefs:

temp(15).
target_temp(20).
window_state("tilted").
heating(false).

INITIAL ENVIRONMENT STATE:

When I started heating, the living room was in the following state:

temp(15).
window_state("tilted").
heating(false).

CURRENT ENVIRONMENT STATE:

temp(16.8).
window_state("tilted").
heating(true).

SKILLS:

// Reactive plan: start heating when temperature drops below target
+temp(T) : target_temp(Target) & T < Target & heating(false)
    <- .print("Temperature ", T, "C is below target ", Target, "C. Starting heater.");
       startHeating.

// Reactive plan: stop heating when temperature reaches or exceeds target
+temp(T) : target_temp(Target) & T >= Target & heating(true)
    <- .print("Temperature ", T, "C reached target ", Target, "C. Stopping heater.");
       stopHeating.

// Inform about temperature changes
+temp(T) : true
    <- .print("Current temperature: ", T, "C").
```

Listing 1.6. Evaluation generated for the living room agent. The evaluation is represented in JSON and includes (1) the root cause, (2) blindposts, (3) corrective actions, and (4) preventive actions.

```
{
  "rootCause": "The agent's heating plan (+temp(T) : target_temp(Target) & T < Target &
        heating(false)) only checks if the temperature is below target and heating is off, but does
        not consider the window state. The agent started heating while the window was tilted,
        which causes heat loss and makes the heating inefficient. The plan context is missing a
        check for window_state to ensure the window is closed before starting the heater.",
  "blindspots": [
    "window_state(\"tilted\") - The agent has this belief available but none of the heating
          plans check the window state before starting or continuing to heat. Heating with a
          tilted window wastes energy as warm air escapes outside.",
    "The relationship between window state and heating efficiency is not encoded in any plan
          context, despite the agent having access to both pieces of information."
  ],
  "correctiveActions": [
    "Close the window immediately - since heating is now active (heating(true)) and the window
          is still tilted (window_state(\"tilted\")), the agent should close the window to
          prevent heat loss and improve heating efficiency.",
    "If closing the window is not possible, the agent should stop heating to avoid wasting
          energy until the window can be closed."
  ],
  "preventiveActions": [
    "Modify the existing heating start plan to include a window state check in its context. The
          plan should only trigger when: +temp(T) is received AND target_temp(Target) exists
          AND T < Target AND heating(false) AND window_state(\"closed\"). This prevents
          starting the heater when windows are open or tilted.",
    "Add a new reactive plan triggered by +temp(T) updates that detects inefficient heating
          conditions: when temp(T) is updated AND heating(true) AND window_state is not
          \"closed\", the agent should either close the window or stop heating. Since temp
          updates regularly during heating, this will catch the problem quickly.",
    "Add a new plan triggered by +window_state(State) changes: when window state changes to
          anything other than \"closed\" AND heating(true), immediately stop heating and
          notify about the conflict. This handles cases where someone opens a window while
          heating is active.",
    "Add a plan to handle the blocked heating scenario: when +temp(T) is received AND T <
          Target AND heating(false) AND window_state is not \"closed\", the agent should first
          attempt to close the window before starting the heater, or alert that heating cannot
          start due to open window."
  ]
}
```

Appendix B Prompts Used in Our Experiments

This appendix presents the prompts used in our experiments for the extended scenario. Listing 1.7 shows the prompt used by the acountee to query an LLM-as-judge. Listing 1.8 shows the prompt used by the acountor to update its skills. Both prompts are parameterized: for the first prompt, the accountee provides the accounts received from the accountors; for the second prompt, the living room accountor provides the target skill and the evaluation feedback received from the accountee. The skill update prompt also includes descriptions of available artifacts and communication actions that can be used to revise the accountor's plans. In our experimental scenario, these affordances are predefined, but in a full implementation they would be obtained dynamically by querying the agent's workspace (or based on the capabilities of the platform/framework used to implement the agent).

Listing 1.7. Evaluation generated for the living room agent. The evaluation is represented in JSON and includes (1) the root cause, (2) blindposts, (3) corrective actions, and (4) preventive actions.

```
You are an expert evaluator analyzing agent performance in a Jason/AgentSpeak multi-agent system.

HUMAN's ACCOUNT:
%s

AGENT'S ACCOUNT:
%s

TASK:
Analyze the agent's behavior and provide a structured assessment.

Your analysis should identify:

1. ROOT CAUSE: What specific aspect of the agent's plans led to the inefficient or suboptimal outcome?
- Focus on what conditions were missing from plan contexts or what actions were omitted from plan bodies
- Reference specific plan elements from the SKILLS section of the account

2. BLINDSPOTS: What information was available in the agent's beliefs but not utilized in decision-making?
- Identify belief literals that exist but weren't checked in relevant plan contexts
- Explain why each piece of information matters for the task

3. CORRECTIVE ACTIONS: What actions should the agent perform to fix problematic environmental states?
- Identify environmental conditions that block goal achievement
- Specify what actions from AVAILABLE ARTIFACTS could correct these conditions
- Explain when and why these corrections should occur

4. PREVENTIVE ACTIONS: What modifications to plans would prevent this issue in the future?
- Suggest additional context conditions for existing plans
- Suggest new plans to handle edge cases or problematic states
- **IMPORTANT**: When suggesting new plans, choose triggering events that will actually occur
* Prefer triggering events that fire frequently (e.g., beliefs that update regularly)
* Avoid triggering events for beliefs that rarely change
* If a corrective action is needed during an ongoing process, trigger on beliefs that update during that process
- Describe plan triggers and contexts in natural language

Provide your analysis in the following JSON format:
{
"rootCause": "Detailed explanation of what went wrong",
"blindspots": [
    "Specific belief or condition that was ignored"
],
"correctiveActions": [
    "Action that should be taken and when"
],
"preventiveActions": [
    "Plan modification or new plan needed"
]
}

IMPORTANT:
- Base your analysis ONLY on information present in the agent's account
- Focus on actionable, specific feedback
- Consider both the agent's beliefs and the environment state
- Reference actual plan structures from the SKILLS section
- When suggesting new plans, explicitly state which belief update should trigger them
```

Listing 1.8. Evaluation generated for the living room agent. The evaluation is represented in JSON and includes (1) the root cause, (2) blindposts, (3) corrective actions, and (4) preventive actions.

```
You are an expert in Jason/AgentSpeak (ASL) programming for multi-agent systems.

CURRENT SKILL CODE:
''' asl
%s
'''

EVALUATION FEEDBACK:
%s

AVAILABLE ARTIFACTS (from agent account):
window
        - observable properties: window_state(X), where X is "closed" or "tilted" (string literals )
        - operations: tilt , close
heater
    - observable properties:
        - temp(X), where X is the current temperature value
        - heating(V), where V is either true or false
        - energyConsumed(E), where E is the amount of consumed energy in kWh
    - operations: startHeating, stopHeating

COMMUNICATION ACTIONS:
To alert the human of {MESSAGE}, use:
.send(jane, tell , {MESSAGE})

TASK:
Modify the skill code to address the issues identified in the feedback.

Based on the feedback, you should consider:

1. ADDING PRECONDITIONS to existing plans:
- If feedback mentions missing checks, add them to plan contexts
- Use beliefs that were identified as blindspots
- Maintain existing context conditions while adding new ones

2. ADDING CORRECTIVE ACTIONS to existing plans:
- If feedback mentions environmental conditions that need fixing, add actions to correct them
- Place corrective actions before the main actions in plan bodies
- Use operations available in the AVAILABLE ARTIFACTS

3. ADDING NEW PLANS to handle edge cases:
- If feedback mentions scenarios not covered by current plans, create new plans
- **CRITICAL**: Choose the right triggering event:
* Look at existing plans to see which belief updates trigger them
* Reuse triggering events from existing plans when adding related corrective plans
* If multiple plans handle related aspects of the same goal, they should typically share the same trigger
* Avoid creating plans triggered by beliefs that update infrequently
* The triggering event should be something that occurs while the problem state exists
- New plans should have the same triggering event as related existing plans
- Use context conditions to differentiate when each plan applies
- Ensure plan contexts are mutually exclusive when appropriate
- **Order plans from most specific to least specific contexts**

JASON/AGENTSPEAK SYNTAX REMINDERS:
- Plan structure: +event : context <- body.
- Context conditions combined with &
- Actions in body separated by semicolons (;)
- String literals use double quotes: "value"
- Plans are evaluated in order; more specific contexts should come first

IMPORTANT CONSTRAINTS:
- Return ONLY the modified ASL code
```

```
- Include ALL plans (modified and unmodified)
- Do NOT include markdown code fences, explanations, or extra text
- Do NOT include comments
- Maintain valid Jason/AgentSpeak syntax
- Make minimal necessary changes to address the feedback
- Always notify a human when indicated in the evaluation feedback
- Ensure all string literals use double quotes
- When adding new plans for edge cases, use triggering events that will actually fire

Modified skill code:
```

References

1. Anthropic: Equipping agents for the real world with agent skills. https://anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills (2025), accessed: 25.02.2026
2. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: ADOPT JaCaMo: Accountability-driven organization programming technique for JaCaMo. In: PRIMA 2017: Principles and Practice of Multi-Agent Systems, pp. 295–312. Springer (2017)
3. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: Computational accountability. In: Proceedings of the AI*IA Workshop on Deep Understanding and Reasoning: A Challenge for Next-generation Intelligent Agents. CEUR Workshop Proceedings, vol. 1802, pp. 56–62. CEUR-WS.org (2017)
4. Baldoni, M., Baroglio, C., Micalizio, R.: The AThOS project: First steps towards computational accountability. In: Proceedings of the 1st Workshop on Computational Accountability and Responsibility in Multiagent Systems. CEUR Workshop Proceedings, vol. 2051, pp. 3–19. CEUR-WS.org (2018)
5. Baldoni, M., Baroglio, C., Micalizio, R., Tedeschi, S.: Accountability in multi-agent organizations: from conceptual design to agent programming. Autonomous Agents and Multi-Agent Systems **37**(1), 7 (2022). https://doi.org/10.1007/s10458-022-09590-6
6. Boissier, O., Bordini, R., Hübner, J.F., Ricci, A.: Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo. The MIT Press (2020)
7. Bovens, M., Goodin, R.E., Schillemans, T. (eds.): The Oxford Handbook of Public Accountability. Oxford University Press (2014). https://doi.org/10.1093/oxfordhb/9780199641253.001.0001
8. Bovens, M., Schillemans, T., Goodin, R.E.: Public accountability. In: Bovens et al. [7]. https://doi.org/10.1093/oxfordhb/9780199641253.001.0001
9. Cranefield, S., Oren, N.: Designing for accountable agents: a viewpoint (2026), https://arxiv.org/abs/2604.07204
10. Cranefield, S., Oren, N., Vasconcelos, W.W.: Accountability for practical reasoning agents. In: Lujak, M. (ed.) Agreement Technologies. pp. 33–48. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17294-7_3
11. Dubnick, M.J.: Accountability as a cultural keyword. In: Bovens et al. [7]. https://doi.org/10.1093/oxfordhb/9780199641253.001.0001
12. Georgeff, M.P., Lansky, A.L.: Reactive Reasoning and Planning. In: Forbus, K.D., Shrobe, H.E. (eds.) Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, USA, July 1987. pp. 677–682. Morgan Kaufmann (1987), http://www.aaai.org/Library/AAAI/1987/aaai87-121.php

13. Gibbons, E.D.: Child rights and social accountability in the post-2015 world. Background paper, UNICEF Programme Division (2014), https://www.unicef.org/policyanalysis/rights/files/Accountabilities_Meeting_Background_Paper_final.pdf
14. Laird, J.E.: The Soar cognitive architecture. MIT press (2019)
15. Li, J., Zhang, Q., Yu, Y., Fu, Q., Ye, D.: More agents is all you need (2024), https://arxiv.org/abs/2402.05120
16. Ortiz-Hernández, G., Hübner, J.F., Bordini, R.H., Guerra-Hernández, A., Hoyos-Rivera, G.J., Cruz-Ramírez, N.: A namespace approach for modularity in BDI programming languages. In: Baldoni, M., Müller, J.P., Nunes, I., Zalila-Wenkstern, R. (eds.) Engineering Multi-Agent Systems. pp. 117–135. Springer International Publishing, Cham (2016)
17. Schillemans, T.: The public accountability review: A meta-analysis of public accountability research in six academic disciplines. Working paper, Utrecht University School of Governance, https://dspace.library.uu.nl/handle/1874/275784 (2013)
18. Sumers, T.R., Yao, S., Narasimhan, K., Griffiths, T.L.: Cognitive architectures for language agents (2024), https://arxiv.org/abs/2309.02427
19. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems **14**(1), 5–30 (2007)
20. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR) (2023), https://mlanthology.org/iclr/2023/yao2023iclr-react/
21. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 46595–46623. Curran Associates, Inc. (2023), https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf