Ex-Plan: Explaining BDI Agent Behaviour Through Contrastive Plan Analysis

Curtis Davies[1][0009-0006-8412-3893] and Babak Esfandiari[2][0000-0002-4441-3435]

Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada {curtisrdavies,babak}@sce.carleton.ca

Abstract. Belief-Desire-Intention (BDI) agents can make decisions that are difficult to interpret from observed behaviour alone. We present Ex-Plan, a trace-based methodology for generating contrastive explanations of AgentSpeak agent behaviour, answering queries of the form "Why did the agent do X rather than Y?". Given an explanandum event and a foil literal, Ex-Plan reconstructs the relevant decision context through AgentSpeak plan structure (triggers and context guards), together with a linear execution trace, and returns a minimal set of trace events that witness where the foil became infeasible. This provides a post-hoc, trace-based contrastive explanation grounded in an agent's plan structure and logged events. We evaluate the approach on three AgentSpeak scenarios using structured execution logs, demonstrating how an action can be explained in contrast to an alternative using specific, pivotal events in the trace.

Keywords: Explainable AI (XAI) · Belief-Desire-Intention (BDI)

1 Introduction

Autonomous agents are increasingly deployed in settings where understanding why they act in certain ways is crucial for trust, safety and effective collaboration with humans and other agents alike [6]. Explainability has therefore emerged as an important property of AI and multi-agent systems. In particular, agents based on the Belief-Desire-Intention (BDI) model offer a transparent decision-making structure that can be leveraged for explanations. In a BDI agent, behaviour results from the agent's beliefs about the world, its goals (desires) and the plans (intentions) it selects for execution [2]. This cognitive architecture inherently records reasoning steps (for example, which plan was chosen to achieve a goal), suggesting that BDI agents support explainability by design. A developer may be satisfied with a low-level trace of events or a plan ID as an explanation for a phenomenon, but an end-user needs an explanation in terms of domain concepts with minimal unnecessary, implementation-level information [17, 11].

Moreover, human explanation-seeking is often contrastive by nature. People typically ask "Why X instead of Y?" rather than simply "Why X?" In other

words, an explanation is sought relative to a plausible alternative outcome (a *foil*) that did not occur. Cognitive studies show that focusing on the difference between a fact and a foil allows explanations to pinpoint the relevant cause of a discrepancy [12]. Effective explanations are also selective, in that they highlight only the key factors among many that led to the outcome.

These insights from social science have been influential in recent explainable AI research, but are not yet fully realized in many agent systems. In the context of BDI agents, a contrastive question might be: "Why did the robot deliver a package via Route A (which it did), instead of Route B (which I expected)?" A satisfying answer should refer to the agent's knowledge and goals, rather than just citing implementation-specific variables or counters as one might see in traditional debugging software.

In this paper, we propose Ex-Plan: a methodology for contrastive explanation of BDI agent behaviour that combines the interpretability of BDI models with the human-centric notion of contrastive, causal explanations. We specifically focus on AgentSpeak, a well-known agent-oriented programming language [1, 13], though the approach is applicable to any BDI platform with similar constructs.

Our goal is to support human-oriented explanation queries over agent execution logs: given an observed event in a trace (the explanandum) and a user-specified alternative (the foil), we identify the plan that produced the observed behaviour, a plausible foil plan that would realize the foil, and the minimal set of discriminating preconditions that separate the two. We then ground each discriminating condition to trace evidence, returning the specific belief/goal or plan-selection events that witness where the foil became infeasible.

2 Background

This section reviews prior work on contrastive explanation, trace-based debugging, and explainable agent systems that derive explanations from execution logs.

2.1 Contrastive explanations

A central finding in explanation research is that many explanation requests are implicitly *contrastive*. Questions of the form "Why P?" are often intended as "Why P rather than Q?" where Q is a counterfactual alternative, commonly called the *foil* [12]. The contrast arises from a mismatch between the explainee's mental model of the circumstances under which Q would occur and the actual circumstances that produced P. While explanations that provide a broad context may be accurate, they tend to add extra information that does not identify the source of this mismatch.

In operational terms, a contrastive explanation aims to identify factors that differ between the actual and foil situations, and that account for the divergence between P and Q along the relevant causal or decision-making pathway. This is illustrated in Figure 1, showing diverging histories between a real event and its foil. For algorithmic treatments, it is useful to distinguish:

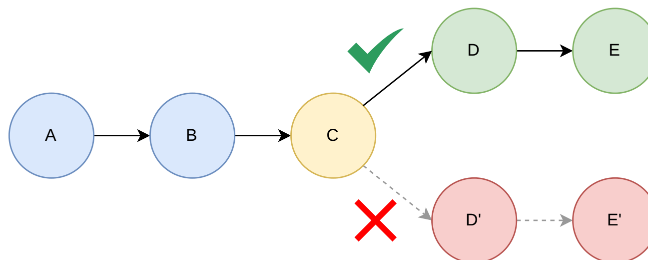

Fig. 1. Real and hypothetical event histories share a prefix (A–B–C) and diverge at C to the realized event E versus the foil E'.

- the *explanandum* P, the observed outcome to be explained,
- the *foil* Q, the alternative outcome of interest,
- the *evidence basis*, the source of truth that grounds the explanation.

In agent settings, contrastive queries commonly arise because internal state and deliberation are only partially observable to an external stakeholder. An explainee may expect Q on the basis of incomplete information, while the agent produces P due to beliefs, events, or constraints not represented in that expectation. Effective explanations therefore aim to surface the discriminating conditions that separate P from Q, while avoiding restating background facts already shared by the explainee.

Recent BDI-specific evaluations support this emphasis on selectivity. Winikoff presents a BDI explanation mechanism to answer explicitly contrastive queries, reporting that they return significantly shorter explanations. A human-subject study further finds scenario-dependent evidence that contrastive answers are preferred and can increase trust, perceived understanding, and confidence in the system's correctness. Notably, the same study reports cases where full explanations reduced confidence relative to providing no explanation, showing a practical risk of overly verbose explanations, indicating the need for a minimal (yet sufficient) information set within an explanation [14]. Since Winikoff's method is the closest prior approach to ours, we return to it in Section 3.9 when discussing how Ex-Plan's output differs from goal-plan trees.

2.2 BDI Agents and AgentSpeak

Belief–Desire–Intention (BDI) agents structure decision-making around explicit symbolic attitudes: a *belief base* representing current information, *goals* and *events* representing objectives and stimuli, and *intentions* representing committed courses of action. In AgentSpeak-style languages, behaviour is encoded as *plans* of the form *trigger* : *context* $\leftarrow$ *body* [13]. A triggering event retrieves candidate plans, the context condition filters for applicability against current beliefs, and the body executes actions and subgoals. Execution proceeds through an event-option-intention cycle in which the agent selects an event, generates

relevant options by trigger matching, filters by context, commits to a plan by adopting or advancing an intention, and executes intention steps as beliefs and goals evolve.

BDI systems are a natural supporting foundation for explanation, because their operational semantics expose meaningful decision points that mimic human reasoning [16, 10]. Plan relevance and applicability, belief conditions that enable or disable alternatives, and explicit goal and intention structure align closely with the abstractions used by developers to discuss behaviour. Compared with opaque learned policies or purely reactive controllers, BDI execution yields symbolic state transitions and explicit alternatives that can be referenced directly when explaining why one course of action was taken and another was not.

2.3 Log-based Explanations and Debugging

Explanations in software systems. and more specifically in agent-based systems, are generated through a range of interaction and analysis paradigms, including dialogue-based justification, trace-driven analysis, narrative abstraction, and question-driven debugging interfaces.

Interrogative debugging systems such as the Whyline demonstrate that explanation can reduce cognitive overhead by returning a focused slice of program context that is sufficient to answer a user query [9]. However, these systems are largely grounded in general-purpose control flow and data flow rather than agent-specific constructs such as beliefs, goals, and intentions, and they do not natively support explicit contrasts of the form "Why P rather than Q?". Other prior works have introduced Whyline-style interrogative debugging methods to agent-specific systems [15, 7], but have yet to introduce a question format that compares an event to an alternative.

Dialogue-oriented explanation frameworks, such as that proposed by Dennis et al., treat explanation as an interactive process. A user poses a query (for example "Why action A at time T?") and receives a minimal justification, often expressed in terms of a state fact such as a belief. The user can then iteratively ask follow-up "Why?" questions to traverse earlier parts of the agent's history until the relevant mismatch is resolved [5]. Such approaches support adaptive depth by allowing the explainee to control how far back the explanation chain proceeds, and they leverage causal links in the event history to justify actions and state changes. A limitation is that the user may need to perform substantial manual exploration, potentially traversing long chains through repeated follow-up questions before reaching the decisive factor.

A different line of work is shown in the multi-level explainability framework of Yan et al., which performs post-hoc analysis external to the agent platform. The approach logs substantial contextual metadata per event to enable interpretation without requiring direct access to the runtime when generating explanations [17]. In this setting, the explainer primarily enriches trace points with surrounding contextual information and higher-level narrative structure, rather than constructing explicit comparisons between behavioural alternatives.

2.4 Summary and Implications

Prior work establishes three complementary points. First, contrastive explanation theory represents an explanation request in terms of an explanandum and a foil, with the goal being to find the differentiating factor between the two [12]. Second, BDI agent programming exposes semantically meaningful decision points, such as plan relevance, applicability, and selection, that naturally support contrasts grounded in symbolic state and deliberation. Third, log-based explanation and debugging methods demonstrate that traces can support retrospective interpretation through interactive justification, contextual enrichment, and question-driven retrieval [5, 17, 9].

Taken together, these findings suggest that an effective explanation mechanism for BDI systems should treat contrast as a first-class input, ground explanations in agent-specific evidence, including beliefs, events, intentions, and the plan library, and localize the divergence between observed and foil behaviours to specific deliberation points in the execution trace. The remainder of this paper builds on these implications by constructing contrastive, log-grounded explanations that refer to the plan alternatives available at a decision point and the conditions that prevented the foil behaviour from being realised.

3 Methodology: Contrastive Trace Analysis

Our methodology takes as input two components: (a) the agent's execution trace, and (b) a contrastive query consisting of an actual event (the explanandum) and a foil literal. The output is a set of events that constitute the explanation. In order to produce an explanation, we do the following:

1. Find the real plan instance π_R responsible for the explanandum event.
2. Determine foil plan π_F, selected from candidate plans that could realize the foil.
3. Find discriminating conditions Δ which are required as preconditions by π_F, but not by π_R.
4. Locate divergence points D, mapping each $\delta \in \Delta$ to the last traced event that made the foil unreachable.

3.1 Inputs and Assumptions

- We assume a linear trace of timestamped events emitted by an AgentSpeak agent. The trace includes, at minimum: belief additions/removals, goal additions/removals, plan selection events and action executions. Crucially, it also includes plan loading events, (e.g. `PlanAdded`) that provide a plan's trigger, context (guard) and body as it is loaded into the agent's plan library at runtime.
- A query is a pair (E, F) where E is an event instance that occurred in the trace (the explanandum) and F is a foil literal representing a plausible alternative.

```
+ask(owner,beer) : true <- !has(owner,beer).

+stock(beer,0) : true <- -available(beer,fridge).

+!has(owner,beer)
    :  not has(beer) & available(beer,fridge) & at(fridge)
    <- .open(fridge); .get(beer); !has(owner,beer).

+!has(owner,beer) : not at(fridge)
    <- .move_towards(fridge); !has(owner,beer).

+!has(owner,beer) : at(fridge) & not available(beer,fridge)
    <- .order(beer,5).

+!has(owner,beer) : has(beer) & not at(owner)
    <- .move_towards(owner); !has(owner,beer).

+!has(owner,beer) : has(beer) & at(owner)
    <- .deliver(owner,beer); -has(beer).
```

Listing 1.1. The Domestic Robot's AgentSpeak code (Adapted from [8]).

- Because plans are added at the beginning of an agent's runtime upon loading them from an initial set of AgentSpeak behaviours (such as from a `.asl` file), we let Π be a static set of plans representing the agent's plan library after each plan is initially loaded. We assume the agent's plan library is not modified after this initialization phase, and Π remains static regardless of the time at which it is observed.
- Each plan in the library is denoted π with subscripts for specific plans. A plan π has a trigger event $te(\pi)$ and a set of literals $guard(\pi)$. We define the precondition set $\mathrm{Pre}(\pi) = \{\mathrm{te}(\pi)\} \cup \mathrm{guard}(\pi)$, where $Pre(\pi)$ includes the triggering condition (which may be a goal or belief) and all context belief literals that must hold for the plan to be eligible for selection.
- The trace is modeled as a time-ordered series of events. While these events can also be bucketed by the reasoning cycles during which they occur, we consider only the linear ordering between them.
- We assume that for any action execution in the trace, we can determine which plan instance (intention) produced it, either by storing this context within the event data or reconstructing the agent's state at the time of evaluation [4].

We now describe each step algorithmically. As a running example, we consider a domestic robot operating in a home environment, similar to Jason's Domestic Robot example environment [8]. The example robot's source code is presented in Listing 1.1. To make the running example concrete, Table 1 summarizes the projected trace events relevant to the beer-ordering decision.

Cycle	Event	Payload (projected)
1	BeliefAdded	`stock(beer,1)`
1	BeliefAdded	`available(beer,fridge)`
3	BeliefAdded	`ask(owner,beer)`
3	EventSelected	`+ask(owner,beer)`
3	PlanSelected	`p1` (trigger `+ask(owner,beer)`)
3	GoalAdded	`has(owner,beer)` (intention 1)
4	EventSelected	`+!has(owner,beer)`
4	PlanSelected	`p4` (trigger `+!has(owner,beer)`)
4	ActionExec	`move_towards(fridge)` (intention 2)
8	BeliefAdded	`at(fridge)`
8	BeliefRemoved	`available(beer,fridge)`
8	BeliefAdded	`stock(beer,0)`
8	PlanSelected	`p5` (trigger `+!has(owner,beer)`)
8	ActionExec	`order(beer,5)` (intention 4)
9	EventSelected	`+stock(beer,0)`
9	PlanSelected	`p2` (trigger `+stock(beer,0)`)

Table 1. Excerpt of the beer scenario trace (compact projection from the logs).

The robot maintains beliefs about its location (`at(fridge)`), the availability of items in the fridge (`available(beer,fridge)`), and whether the owner currently has an item (`has(owner,beer)`). When the owner requests beer, the robot travels to the fridge and can either fetch a beer (`get(beer)`) if one is available, or place an online order for beer (`order(beer,5)`) when the fridge is empty. In this scenario, the robot has placed an online order for beer. The user questions why the robot performed this action (the explanandum E), rather than getting one from the fridge (the foil F).

3.2 Identifying the Real Plan

The first step is to find the plan instance π_R under which the actual event E occurred, as outlined in Algorithm 1. E can be any type of event within the agent's reasoning process, such as an action, belief update or subgoal achievement. Actions are likely to be the most common case, since user queries often center on observable behaviour.

The algorithm first checks if a link exists within the trace G that connects the event E to its parent plan. We use a helper function $\mathsf{ParentPlan}(E, G)$ to denote this, returning the associated plan. If such a link does not exist, the algorithm searches for the most recent plan selection event prior to E's execution that matches its intention and returns this plan.

In our example, Algorithm 1 identifies π_R as the plan related to giving a beer to the owner when the fridge is empty:
`+!has(owner,beer) : at(fridge) & not available(beer,fridge)`.

3.3 Retrieving Candidate Foil Plans

Next, we gather plans that could have produced the foil event F. We do this through static analysis, by searching through the plan library for plans whose bodies contain a literal matching F. This can be represented formally as follows:

$$\Pi_F = \{\pi \in \Pi \mid F \in \mathsf{body}(\pi)\}$$

Algorithm 1 Identify the plan instance responsible for an actual trace event.

Input: event instance E; event trace G
Output: plan instance π_R responsible for E, or $\perp$

```
1: function FindRealPlan(E, G)
2:     if ParentPlan(E, G) ≠ null then
3:         return π_R ← ParentPlan(E, G)
4:     else
5:         t ← time(E)
6:         π_R ← ⊥
7:         for all events e in G in reverse-chronological order starting from t do
8:             if e.type = PlanSelected and Intention(E) = Intention(e) then
9:                 π_R ← SelectedPlan(e)
10:                break
11:            end if
12:        end for
13:        return π_R
14:    end if
15: end function
```

The applicability of these plans is addressed later.

One point worth noting is that this algorithm returns a set Π_F, meaning a foil can have more than one candidate plan. Since our methodology hinges on a direct comparison of a real and foil plan, we address this concern next.

3.4 Selecting the Optimal Foil Plan

If there is only one candidate plan for F, we take that as π_F. However, if there are multiple, we assume the user's implicit foil is the one most analogous to what actually happened. We use the idea that the preconditions for the explanandum and the foil differ minimally, except for some key differentiators. Thus, we select the plan whose context is most similar to π_R.

Let $B = \mathsf{Pre}(\pi_R)$ denote the trigger and guard literals of the real plan. We select the foil plan whose preconditions are the most similar to B:

$$\pi_F = \arg\max_{\pi \in \Pi_F} \mathsf{similarity}(\mathsf{Pre}(\pi),\, B)\,. \tag{1}$$

Here, we use a generic $\mathsf{similarity}(\mathsf{A}, \mathsf{B})$ function to represent any function that returns a value based on the similarity between two plans' sets of preconditions. For example, $\mathsf{similarity}$ could be defined as Jaccard similarity $\frac{|A \cap B|}{|A \cup B|}$. Another approach could be to penalize differences (e.g. $|A \setminus B| + |B \setminus A|$) and choose the plan with the minimum distance. In either case, we rank the candidates and choose the closest one to π_R. It is worth noting that in many cases, the number of relevant conditions in these plans is small, so a simple comparison suffices.

Another point worth keeping in mind is that this method does not check the relevance of each π_F at the selection time of π_R, and does not account for foils with a deeper relation to π_R (which could stem from earlier in the event trace

than π_R's selection). To maintain the scope of this methodology, we leave these foil plan selection methods to future work.

In our domestic robot example, Ex-Plan selects π_F as plan `p3` for the same trigger, whose context includes `available(beer,fridge)` (and `at(fridge)`).

3.5 Finding the Discriminating Conditions

Given π_R and π_F, we compute the set difference of their precondition sets: $\Delta = \mathsf{Pre}(\pi_F) \setminus \mathsf{Pre}(\pi_R)$. This returns the conditions required by the foil plan that were not required by the actual plan. Given that π_R was executed at runtime instead of π_F, we can state that each precondition of π_R was met, while a subset of the preconditions of π_F did not. Therefore, any condition that π_F has and π_R does not, if not true, could have blocked the execution of of π_F.

$$\Delta = \mathsf{Pre}(\pi_F) \setminus \mathsf{Pre}(\pi_R). \tag{2}$$

For the domestic robot, the discriminating condition is the single literal `available(beer,fridge)`, as this precondition is present in π_F, but not π_R.

3.6 Finding Divergence Points

By this stage, we have a set of preconditions for the foil plan that may have acted as blocking conditions, preventing its execution at runtime. While this list of predicates might be appropriate for someone manually debugging the agent's behaviour, it fails to indicate an exact moment in the agent's runtime that caused the preconditions to be unmet.

In order to provide this missing piece of information, we search the trace for the moment or event that determined the satisfaction of each literal δ within Δ. We do this by finding the most recent event in the trace that established the non-satisfaction of δ prior to when π_R was selected. We refer to this as the divergence event c, part of the set of all divergence events C corresponding to Δ.

Formally, if T_R is the selection time of π_R, we search for an event c in the trace at time $t < T_R$, such that:

- The set of literals modified by c includes δ.
- Immediately after c, the condition δ is not satisfied.
- There is no later event prior to T_R that also affects the satisfaction of δ.

We define the divergence map D, such that, for each $d \in D$ and $e \in G$, $d = e$ if and only if e is the most recent event in G prior to T_R which satisfies the prior conditions.

This definition outlines that a divergence point should be the last trace event that "made the foil impossible" (with respect to δ) before the agent selected the real plan.

The predicates $\mathsf{affects}(e, \delta)$ and $\mathsf{UnsatisfiedAfter}(\delta, e, G)$ are trace-dependent. In the minimal case, $\mathsf{affects}$ holds when e is a belief addition/removal mentioning δ (or its negation), and $\mathsf{UnsatisfiedAfter}$ can be checked by replaying belief updates up to just after e.

In our example, we take the condition `available(beer,fridge)` and trace backwards to the latest event removing it:
`(78, BeliefRemoved(available(beer,fridge))`. This indicates that the robot perceived a lack of beer in the fridge, invalidating the belief that a beer is available.

3.7 Assembling Explanation Trails

The divergence map D identifies, for each discriminating condition δ, a single trace event that last established δ's failure before π_R was selected. To present this in a compact, contrastive form, we output a short trail per δ that ties that divergence to (i) the selection of π_R and (ii) the explanandum event E.

When the trace records enough structure, we optionally augment the trail with causal hints around the divergence event. For example, if the divergence event is a belief update, we can search backward for a plan-selection event whose plan body could plausibly have produced that update (e.g., by containing an action or internal event that adds/removes the relevant literal). This yields an implementation-level but still plan-grounded "because" chain.

We use helper functions and predicates for trail augmentation. First, $\mathsf{CanAugment}(c, G)$ holds when the trace contains sufficient provenance structure to connect the divergence event c to an earlier plan-selection event, for example, through intention identifiers. Second, $\mathsf{FindEffectPlanSelection}(c, \delta, G)$ returns the most recent plan selection event sel_c before c such that the selected plan belongs to the same context as c and its body contains a step that could account for the state change in δ observed at c.

In the minimal trace-only setting, CanAugment is false and each trail reduces to $[c, sel_R, E]$. When intention links are available, FindTriggerEvent can follow them to report the event that introduced the intention containing sel_c.

To complete the example, Ex-Plan shows that the reason the robot performed `order(beer,5)` in lieu of `get(beer)` was due to the removal of the `available(beer,fridge)` belief, which led to the selection of the explanandum's plan, `p5`.

3.8 Algorithmic Complexity

We summarize the asymptotic cost of Ex-Plan in terms of the trace length $|G|$ (number of logged events), the number of plan schemas $|\Pi|$, and the average plan body length L. Let $|\mathrm{Pre}(\pi)|$ denote the size of a plan's precondition set (trigger plus context/guard literals), and let $\Pi_F \subseteq \Pi$ be the set of candidate foil plans returned by Section 3.2.

Identifying the real plan. Finding the plan instance π_R responsible for the explanandum event E is a bounded backward scan over the trace until the matching `PlanSelected` event is located (or an equivalent parent link is used when available). This step is $O(|G|)$ in the worst case.

Algorithm 2 Assemble contrastive explanation trails grounded in trace evidence.

Require: Explanandum event E; real plan π_R and its selection event sel_R; divergence map D; trace/event graph G
Ensure: A set of explanation trails S, one per $\delta \in \mathrm{dom}(D)$
1: $S \leftarrow [\,]$
2: **for all** δ such that $\delta \in \mathrm{dom}(D)$ **do**
3: $\quad trail \leftarrow [\,]$
4: $\quad c \leftarrow D[\delta]$
5: $\quad$ append c to $trail$
6: $\quad$ **if** $\mathsf{CanAugment}(c, G)$ **then**
7: $\quad\quad sel_c \leftarrow \mathsf{FindEffectPlanSelection}(c, \delta, G)$
8: $\quad\quad$ **if** $sel_c \neq \bot$ **then**
9: $\quad\quad\quad trig_c \leftarrow \mathsf{FindTriggerEvent}(sel_c, G)$
10: $\quad\quad\quad$ prepend $trig_c$ then sel_c to $trail$
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: $\quad$ append sel_R then E to $trail$
14: $\quad$ append $(\delta, trail)$ to S
15: **end for**
16: **return** S

Retrieving candidate foil plans. Candidate retrieval scans each plan body for a step that unifies with the foil literal F. Without indexing, this is $O(|\Pi| \cdot L)$. In an implementation, a simple index keyed by predicate symbol (or functor) can reduce this to near-constant lookup.

Selecting the optimal foil plan. When multiple candidates exist, Ex-Plan ranks them by similarity of precondition sets. This comparison is $O(|\Pi_F| \cdot |\mathrm{Pre}|)$, and is typically small in practice, since AgentSpeak context guards tend to contain few literals.

Computing discriminating conditions. The discriminating set $\Delta = \mathrm{Pre}(\pi_F) \setminus \mathrm{Pre}(\pi_R)$ is a set difference over small sets and is $O(|\mathrm{Pre}|)$.

Finding divergence points. For each $\delta \in \Delta$, Ex-Plan searches backward from the selection time T_R of π_R for the most recent event that both affects δ and yields δ unsatisfied immediately afterward. In the worst case, this is $O(|\Delta| \cdot |G|)$, though the AFFECTS predicate substantially filters candidate events in typical traces.

Assembling explanation trails. Trail construction is linear in the number of discriminators and the (usually short) retrieved ancestry around each divergence event. In the minimal trace-only setting, each trail reduces to $[c, sel_R, E]$, making the cost effectively $O(|\Delta|)$ plus the cost of retrieving the recorded links.

Overall. The dominant terms are $O(|\Pi| \cdot L)$ for candidate retrieval and $O(|\Delta| \cdot |G|)$ for divergence search. Given that $|\Delta|$ and Π are commonly small, Ex-Plan's runtime is driven primarily by trace length, while avoiding full counterfactual simulation.

3.9 Comparison with Goal-Plan-Tree Explanation

A closely related method is Winikoff's recent account of contrastive explanations for BDI agents [14]; the domestic robot example helps clarify both the overlap and differences with our approach.

Winikoff's approach operates over a goal-plan tree abstraction of a BDI program. In that representation, goal nodes encode the agent's possible courses of action, action nodes encode executable steps and conditions determine which alternatives are available at a decision point. Given a goal-plan tree, an execution trace T, an observed action X and a foil F, the method computes an explanation for X as a set of explanatory factors, composed of beliefs, desires and valuings (context-specific preferences) [14]. Then, factors that are shared by both the action X and the foil F are removed, leaving only those factors that distinguish why X was selected instead of F.

For the query "why did the agent do `order(beer,5)` rather than `get(beer)`?", the local AgentSpeak plan context within our example can be seen as analogous to the corresponding local goal-plan-tree decision in Winikoff's setting. Combined with the presence of the event trace, this suggests that the explanations provided by both approaches are grounded in the agent's behavioural patterns and the series of events that occurred during its runtime.

However, Winikoff's method takes an action and derives an explanation in terms of beliefs, desires and valuings (indicators that one low-level alternative was preferable to another), before applying a filter to retain only the factors that differentiate the explanandum from the foil. An explanation in this form would start as the literal

$$\{D: \mathit{has}(\mathit{owner}, \mathit{beer}), B: \mathit{at}(fridge), B: \mathit{not\ has}(\mathit{beer}), B: \mathit{not\ available}(\mathit{beer}, fridge)\}$$

before being filtered down to just $\{B: \mathit{not\ available}(\mathit{beer}, fridge)\}$; the shared contexts between the explanandum and foil having been removed.

The two methods therefore often arrive at a similar explanatory conclusion, namely, the foil failed because a condition required by the foil was not satisfied. However, the methods differ in how the explanation is structured and grounded. Winikoff's method returns an abstract set of explanatory factors, such as $\{B: \mathit{not\ available}(\mathit{beer}, fridge)\}$, derived from an abstract goal-plan tree and filtered by the foil. Ex-Plan works directly within the AgentSpeak setting, reusing plans, triggers and context guards directly from the agent's runtime. Ex-Plan identifies $\mathit{available}(\mathit{beer}, fridge)$ as the missing condition for the foil plan, but returns the relevant trace events showing when and how that belief was removed.

4 Experimental Evaluation

This section describes a series of experiments investigating the effectiveness of Ex-Plan in generating accurate, contrastive explanations of agent actions. We

evaluate Ex-Plan using MicroASL, a lightweight AgentSpeak/BDI framework designed to execute AgentSpeak-style plans with timed external belief additions and emit structured execution traces from the agent. [1]

MicroASL provides deterministic control over the agent's runtime and a stable log schema (JSONL) capturing plan loading, plan relevance/applicability checks, plan selection, belief updates, goal adoption, and action execution. Mainstream AgentSpeak frameworks such as Jason [1], while versatile for real multi-agent scenarios, make it difficult to inject scripted perceptions into an agent's belief base without the use of a fully-implemented environment class. Despite this, Ex-Plan's methodology operates on higher-level AgentSpeak concepts, making it framework-agnostic, provided that the framework provides adequate traces of an agent's execution.

4.1 Test Scenarios

We evaluate Ex-Plan's explanations on traces gathered from the runtimes of three AgentSpeak programs:

- **Domestic robot.** See Section 3.1; we reuse the same scenario implemented as an agent-based program.
- **Cleaning robot.** A robot cleans each room in its house. The robot normally cleans rooms in sequence, but can skip a room when it is crowded.
- **Autonomous vehicle.** An autonomous vehicle proceeds along a trip to drive its occupant home. The vehicle usually turns right at a specific intersection but may change its route after receiving a traffic update.

In each scenario, the agent performs an action that may be deemed unusual by a third party due to underlying behaviour in the agent's AgentSpeak plans:

- The cleaning robot is notified of the presence of heavy foot traffic in the dining room, causing it to skip the dining room and proceed to clean the kitchen next instead.
- Before arriving at an intersection, the vehicle is notified of traffic to the right of that intersection. Rather than turning right, along its usual route, the vehicle opts to turn left instead.

For each scenario, we make a contrastive query to Ex-Plan, requesting an explanation for the action in comparison to the expected foil behaviour:

- Domestic robot:
  - Explanandum: Event `order(beer,5)`
  - Foil: Literal `get(beer)`
- Cleaning robot:
  - Explanandum: Event `clean(kitchen)`
  - Foil: Literal `clean(dining)`
- Autonomous vehicle:
  - Explanandum: Event `intersection_turn(left)`
  - Foil: Literal `intersection_turn(right)`

[1] Code, traces, and reproduction scripts are available at `https://github.com/110Percent/ex-plan-py` and `https://github.com/110Percent/microasl`.

4.2 Results

Domestic Robot: In our running example, the contrastive query asks why the agent executed `order(beer,5)` instead of `get(beer)`. Qualitatively, Ex-Plan produces a minimal contrast: the foil requires that beer is available in the fridge, but the trace shows that no beer was available, causing the agent to fall back to its behaviour of ordering more. In our implementation, a TUI produces a structured output for this query, including the selected foil plan, unmet discriminating condition and the trace event that invalidated it.

Cleaning Robot: The contrastive query asks why the agent executed `clean(kitchen)` instead of `clean(dining)`.

- **Query.** Explanandum event $E =$ `clean(kitchen)` with foil $F =$ `clean(dining)`.
- **Real plan π_R.**
  - π_R set to the plan associated with cleaning the kitchen, with the header `+!clean : next_room(kitchen)`.
  - $\text{Pre}(\pi_R) = \{$`!clean, next_room(kitchen)`$\}$.
- **Foil plan π_F.**
  - π_F identified as the plan associated with cleaning the dining room, assuming no foot traffic:
    `+!clean : next_room(dining) & not foot_traffic(dining,high)`
  - $\text{Pre}(\pi_F) = \{$`!clean`, `next_room(dining)`, `not foot_traffic(dining,high)`$\}$.
- **Discriminators $\Delta = \text{Pre}(\pi_F) \setminus \text{Pre}(\pi_R)$.**
  - $\delta_1 =$ `next_room(dining)`.
  - $\delta_2 =$ `not foot_traffic(dining,high)`.
- **Divergence events.**
  - For $\delta_1 =$ `next_room(dining)`:
    - Divergence event: `BeliefRemoved next_room(dining)`.
  - For $\delta_2 =$ `not foot_traffic(dining,high)`:
    - Divergence event: `BeliefAdded foot_traffic(dining,high)`.

The foil plan requires both that the dining room is next and that it is not crowded, demonstrating the need for Ex-Plan to handle foil differences with multiple discriminators. The trace contains an explicit crowding update and the consequent revision of the next-room belief, which grounds both discriminators and makes the dining foil plan inapplicable before π_R is selected and `clean(kitchen)` executes. The returned trail indicates that the cleaning robot skipped the dining room as a result of being notified of heavy foot traffic in the area.

Autonomous Vehicle: The contrastive query asks why the vehicle executed `intersection_turn(left)` instead of `intersection_turn(right)`.

- **Query.** Explanandum event `E` = `intersection_turn(left)` with foil F = `intersection_turn(right)`.
- **Real plan π_R.**
  - π_R set to the left-turn plan: `+!handle_intersection(I) : route_choice(I,left)`
  - $\text{Pre}(\pi_R) = \{$`handle_intersection(I)`, `route_choice(I,left)`$\}$.
- **Foil plan π_F.**
  - π_F identified as the right-turn plan: `+!handle_intersection(I) : route_choice(I,right)`.
  - $\text{Pre}(\pi_F) = \{$`handle_intersection(I)`, `route_choice(I,right)`$\}$.
- **Discriminator $\Delta = \text{Pre}(\pi_F) \setminus \text{Pre}(\pi_R)$.**
  - δ = `route_choice(I,right)`.
- **Divergence event.**
  - Divergence event: `BeliefRemoved route_choice(i42,right)`.

The right-turn foil requires `route_choice(I,right)`. The trace records an upstream heavy-traffic update that triggers reconsideration and removes the right-route choice; later, π_R becomes applicable and the agent executes the left turn.

5 Conclusion

We presented Ex-Plan, a post-hoc, trace-based framework for answering contrastive "why" questions about AgentSpeak BDI agent behaviour. The experiments demonstrate that Ex-Plan can recover contrastive, plan-grounded explanations from logs in a way that remains faithful to the BDI decision model. In each scenario, Ex-Plan highlights minimal differences between the observed behaviour and a plausible alternative, and it anchors these differences to specific prior events that made the foil unattainable at the decision point. This provides a compact bridge from a user's question to concrete evidence in the event trace and plan library, providing an appropriately contextualized response.

5.1 Limitations

Ex-Plan assumes that the conditions relevant to plan applicability can be reconstructed from events explicitly recorded in the agent's log trace, such as belief additions, removals, percept updates, and plan-selection events. This assumption can fail when an AgentSpeak interpreter supports rules (Horn clauses) in the agent program, as in supersets such as Jason [1]. In such cases, a condition may hold or fail as the result of a derived rule value rather than a directly logged state change. To support these settings faithfully, rule-level updates would need to be recorded in the trace, either by extending the runtime framework to log them explicitly or by inserting equivalent update events at the time of evaluation.

A second limitation involves how Ex-Plan identifies and compares foil plans. The current approach selects an optimal foil from candidate plans using a heuristic based on overlap in triggering events and guard conditions. This is suitable for decision-local contrasts, but it does not fully capture deeper notions of relevance that depend on the goal-plan tree or on relationships involving nested intended means. As such, Ex-Plan currently assumes that the real plan π_R and foil plan π_F can be treated as local alternatives for the queried contrast. Cases in which the apparent foil is related only indirectly to π_R, or through a more distant context, are therefore not handled completely by this method.

Furthermore, Ex-Plan shows counterfactual contrasts through foil inapplicability at the point where π_R is selected. If multiple plans, including the foil, are applicable at that decision point, then the contrast depends not only on which conditions held, but also on the agent platform's plan selection function. Accounting for this requires that the framework record the reason a plan was selected, such as ordering, priority, or other tie-breaking behaviour. Without such information, the explanation can identify why a foil was blocked, but not always why one applicable plan was preferred over another.

Finally, Ex-Plan takes input as a contrastive query (E, F) and therefore assumes that the foil F is given by the explainee. In practice, the relevant foil may be implicit, underspecified, or unknown to the user, especially when several plausible alternatives exist at the decision point or when the agent's vocabulary is not transparent to the explainee. This limits Ex-Plan to explicitly contrastive queries. An extended approach would require inferring or proposing candidate foils so that the framework could answer open-ended "Why E?" questions directly, which we leave to future work.

5.2 Future Work

A principal direction for future work is automatic foil generation for open-ended queries. Ex-Plan currently assumes the foil literal is provided; in practice, many users may instead ask open-ended "why not" questions. Foils can be proposed from decision-local alternatives at the relevant deliberation point (e.g., other relevant/applicable plans and their salient effects), expectation-driven baselines learned from typical plan choices under the same trigger, and other potential user-model constraints that capture what the explainee expected to hold in the state.

While the post-hoc explanation capabilities provided by Ex-Plan are useful for debugging behaviour after it has occurred, it may also be useful for a user or other agent to perform a query "online," while the agent is still running. In a similar fashion to another approach by Broekens et al, [3] Ex-Plan could produce explanations on the fly, using a snapshot of the event trace as it is generated. However, because queries need to point to specific event instances, performing online queries may be difficult for a human to do in real time, hence the need for a harness or interface which facilitates quick and easy event selection.

References

1. Bordini, R., Hübner, J., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley (2007)
2. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational Intelligence **4**(3), 349–355 (1988). https://doi.org/10.1111/j.1467-8640.1988.tb00284.x
3. Broekens, J., Harbers, M., Hindriks, K., van den Bosch, K., Jonker, C., Meyer, J.-J.: Do You Get It? User-Evaluated Explainable BDI Agents. In: Dix, J., Witteveen, C. (eds.) Multiagent System Technologies, pp. 28–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
4. Davies, C., Esfandiari, B.: Event Sourcing in Jason: Event-Driven State Reconstruction for BDI Agents. In: 2024 IEEE International Conference on Agents (ICA), pp. 70–75 (2024). https://doi.org/10.1109/ICA63002.2024.00023
5. Dennis, L.A., Oren, N.: Explaining BDI Agent Behaviour through Dialogue. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '21, pp. 429–437. International Foundation for Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom (2021)
6. Haynes, S.R., Cohen, M.A., Ritter, F.E.: Designs for explaining intelligent agents. International Journal of Human-Computer Studies **67**(1), 90–110 (2009). https://doi.org/10.1016/j.ijhcs.2008.09.008
7. Hindriks, K.V.: Debugging Is Explaining. In: Rahwan, I., Wobcke, W., Sen, S., Sugawara, T. (eds.) PRIMA 2012: Principles and Practice of Multi-Agent Systems, pp. 31–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
8. Hübner, J.: domestic-robot – Jason, https://github.com/jason-lang/jason/tree/main/examples/domestic-robot
9. Ko, A.J., Myers, B.A.: Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '04, pp. 151–158. Association for Computing Machinery, Vienna, Austria (2004). https://doi.org/10.1145/985692.985712
10. Malle, B.F.: How the mind explains behavior: Folk explanations, meaning, and social interaction. MIT press (2006)
11. Mauri, M., Minor, M.: Towards Explainable BDI Agents for End Users. In: Rodriguez, S., Feng, L., Müller, J.P. (eds.) Engineering Multi-Agent Systems, pp. 221–237. Springer Nature Switzerland, Cham (2026)
12. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence **267**, 1–38 (2019). https://doi.org/10.1016/j.artint.2018.07.007
13. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Agents Breaking Away: Agents Breaking Away. MAAMAW '96, pp. 42–55. Springer-Verlag, Einhoven, The Netherlands (1996)
14. Winikoff, M.: Contrastive explanations of BDI agents. In: Proceedings of the 25th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, Paphos, Cyprus (2026)
15. Winikoff, M.: Debugging Agent Programs with Why? Questions. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. AAMAS

'17, pp. 251–259. International Foundation for Autonomous Agents and Multiagent Systems, São Paulo, Brazil (2017)
16. Winikoff, M., Sidorenko, G., Dignum, V., Dignum, F.: Why bad coffee? Explaining BDI agent behaviour with valuings. Artificial Intelligence **300**, 103554 (2021). https://doi.org/10.1016/j.artint.2021.103554
17. Yan, E., Burattini, S., Hübner, J.F., Ricci, A.: A multi-level explainability framework for engineering and understanding BDI agents. Autonomous Agents and Multi-Agent Systems **39**(1) (2025). https://doi.org/10.1007/s10458-025-09689-6