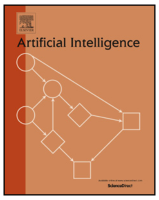
Artificial Intelligence 300 (2021) 103554

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Why bad coffee? Explaining BDI agent behaviour with valuings

Michael Winikoff[a,∗,1], Galina Sidorenko[b,1], Virginia Dignum[c], Frank Dignum[c,d]

[a] Victoria University of Wellington, New Zealand
[b] Halmstad University, Sweden
[c] Umeå University, Sweden
[d] Utrecht University, The Netherlands

ARTICLE INFO

Article history:
Received 7 January 2019
Received in revised form 30 January 2021
Accepted 30 June 2021
Available online 7 July 2021

Keywords:
Explanation
Belief-desire-intention
Valuings

ABSTRACT

An important issue in deploying an autonomous system is how to enable human users and stakeholders to develop an appropriate level of trust in the system. It has been argued that a crucial mechanism to enable appropriate trust is the ability of a system to explain its behaviour. Obviously, such explanations need to be comprehensible to humans. Due to the perceived similarity in functioning between humans and autonomous systems, we argue that it makes sense to build on the results of extensive research in social sciences that explores how humans explain their behaviour. Using similar concepts for explanation is argued to help with comprehensibility, since the concepts are familiar. Following work in the social sciences, we propose the use of a folk-psychological model that utilises beliefs, desires, and "valuings". We propose a formal framework for constructing explanations of the behaviour of an autonomous system, present an (implemented) algorithm for giving explanations, and present evaluation results.

1. Introduction

The deployment of autonomous systems in situations where they interact with people raises the need for these systems to be able to explain their behaviour. Such explanations are important for a range of reasons. They allow humans to better understand the system in order to be able to more effectively coordinate with it and anticipate its behaviour, as well as understand its limitations. Floridi et al. argue that "*It is especially important that AI be explicable, as explicability is a critical tool to build public trust in, and understanding of, the technology*" [1, Section 5.1]. The importance of explanation has also been recognised by the European Union, and is captured in the EU General Data Protection Regulation[2] (http://www.eugdpr.org) as a right to explanation.

Explanations can also play a role in developing an *appropriate* level of trust in a system, including avoiding over-trusting the system [2]. It has been argued [3] that in a range of domains, the system's ability to *explain* why they performed a certain course of action is a key factor in the development of appropriate levels of trust. For example, a recent report

∗ Corresponding author.
E-mail addresses: michael.winikoff@vuw.ac.nz (M. Winikoff), galina.sidorenko@hh.se (G. Sidorenko), virginia@cs.umu.se (V. Dignum), dignum@cs.umu.se (F. Dignum).
1 Michael Winikoff and Galina Sidorenko were at the University of Otago when this work was done.
2 http://tinyurl.com/GDPREU2016 - see Articles 13-15 and 22.

https://doi.org/10.1016/j.artint.2021.103554
0004-3702/© 2021 Elsevier B.V. All rights reserved.

proposes that "...*for users of care or domestic robots a why-did-you-do-that button which, when pressed, causes the robot to explain the action it just took*" [4, Page 20]. Indeed, there is also experimental evidence that explanation, specifically in the form of answers to "why?" questions, supports the development of trust [5].

The problem this paper therefore addresses is how an autonomous system can provide an explanation for why it chose a particular course of action. In other words, answering questions of the form "*why did you do X?*" (we consider other possible forms of questions in the closing discussion of future work). Specifically, we develop a computational mechanism that provides building blocks for explanations that are subsequently assembled into explanations for why a particular action was performed.

In developing such an explanation mechanism, it is important to be mindful that the explanations have to be comprehensible, and useful, to a human listener. In our work we consider in particular the seminal work of Malle [6]. The key aspects of Malle's work that are relevant to us include the following.

Firstly, Malle argues that humans use folk psychological constructs (e.g. beliefs, desires) to explain their behaviour. There is also empirical evidence that humans use these constructs to explain the behaviour of robots [7,8]. This leads us to adopt a model that includes desires and beliefs, specifically the well-known Belief-Desire-Intention (BDI) model [9–11]. Doing so allows explanations to be based on concepts that are the same as those used by humans when explaining their actions. We contend that providing explanations in terms of the same concepts used in human-to-human explanations will help enable explanations to be comprehensible. Note that using a BDI model does not necessarily require the system to be implemented as BDI agents: the BDI model could be used to provide post-hoc explanations of a system's behaviour even if the system does not use BDI concepts. However, this risks differences between the system's behaviour and the BDI module explaining it (we expand on this in the last Section).

Secondly, Malle provides a careful analysis, supported by experimental evidence, that highlights a number of different types of concepts used to explain behaviour. The ones that are most relevant to the context we are considering (software explaining its decisions) are what Malle terms *reasons*.[3] Malle identifies three types of reasons: desires, beliefs, and what he terms *valuings*. He defines valuings as things that "*directly indicate the positive or negative affect toward the action or its outcome*". For example, someone appreciating snow. He goes on to argue that valuings are a third concept, distinct from either beliefs or goals: "*As a group, then, valuings resist being subsumed under either beliefs or desires*" [6, Section 4.2.4]. We therefore extend our BDI model with valuings, following recent work by Cranefield et al. [12] (see Section 2 for how this is done).

This paper uses a running example, which, as suggested by the paper's title, relates to getting coffee. The scenario is the following one: Jo is an academic visiting colleagues at another University. Like many academics, he requires coffee. There are a number of possible sources of coffee. The little kitchen near Ann's office has coffee-like-substance freely available, but this machine requires a staff card to operate. Ann has in her office a coffee machine which converts pods into nice coffee (we assume that Ann has a plentiful supply of coffee pods). There is also a coffee shop a few buildings away, where good coffee can be obtained, at a (financial) cost. Jo prefers coffee to coffee-like substances, which is the over-riding preference. Less-important preferences are to save money, and to use the nearest coffee source. Therefore the three relevant quality attributes are (in order): quality (coffee preferred to coffee-like), money (free preferred to expensive), and location (smallest distance from starting location).

We next discuss related work. The remainder of this paper then briefly presents the formal framework (Section 2), and illustrates it in the context of a running example. The core of the paper is the method for generating explanations based on the formal framework, which we present as formal definitions (Section 3), and then as an efficient algorithm (Section 4). We then empirically evaluate both the efficiency of the algorithm (Section 5) and how the generated explanations are perceived (Section 6). We finish with a discussion of limitations and future work (Section 7).

This paper considerably expands on earlier published work [13] by providing complete detailed definitions, an efficient algorithm (along with its evaluation), and a new user study of the generated explanations.

1.1. Related work

Explainable AI is a very hot topic at this moment due to the advancements in deep learning where neural networks produce results but it is unclear how they are produced. However, explaining the results of rule based expert systems already was discussed in the 1970s and 1980s as can be seen in [14] that discusses the experiences with the MYCIN project. One of the main differences with this early work in expert systems is that the rules in these systems do not (necessarily) have a causal connection such as is the case in goal trees. Thus explanations in these systems were based on which rules were necessary to logically derive a result, but lack a reason for why these rules are used. Another difference is that in explaining the behaviour of autonomous systems we are explaining a *course of action* (taken over time, in an environment),

[3] The other two concepts he identifies are Causal History of Reasons (CHRs), which are general background factors, usually used to explain trends, or to distance oneself from an undesirable action, and Enabling Factors (EFs), which are used to explain "why did this succeed, despite being difficult". Although CHRs are sometimes used to explain behaviour, for both human and robot behaviour [7], as general background factors we consider them less helpful in explaining a specific behaviour. Enabling factors are similar to beliefs in that both are about a condition (e.g. having money). However, EFs are used to answer questions of the form "how was this possible?" whereas beliefs are used to answer questions of the form "why did you do this?", which is the focus of this paper.

not a (static) recommendation. Consequently, we are not dealing with deductive reasoning rules (as in expert systems), but with *practical* reasoning (reasoning about actions to take to achieve a desired outcome).

Despite the huge current interest in explainable AI there are surprisingly few publications on explainable BDI agents. One exception is Harbers [15]. Like us, it assumes that a goal tree is given, and defines a number of templates that can be used to explain observed behaviour. Harbers' templates provide a range of possible explanations, e.g. explaining an action in terms of its parent goal, or its grand-parent goal, or the *next* action (corresponding to our links). She also conducted an empirical evaluation of the different templates in a given scenario. It is worth noting that our approach strictly generalises Harbers' approach, in that we include links, ancestor goals, and relevant beliefs (see Section 3 for details). In other words, every factor that is included in explanations generated using Harbers' templates is included in our explanation function $\mathcal{E}$. Furthermore, Harbers does not take into account other available alternatives, i.e. it explains why X was done solely in terms of what allowed X to be done, and does not mention anything relating to other available options. By contrast, our approach also explains why a particular option was chosen to realise a given goal. Finally, we note that whereas Harbers just outlines the rules as brief templates, we provide full formal definitions and an implementation of them.

Our evaluation (briefly described in Section 6) included explanations that corresponded to the explanation mechanism proposed by Harbers. The evaluation found that this explanation mechanism did not perform well compared to the mechanism proposed in this paper. Additionally, our evaluation found that valuings (which were not included in Harbers' explanation) were the most preferred aspect of explanations.

Other work from workshop series on Explanation aware computing [16] and more recently XAI [17,18] focus on more specific aspects of explanation such as visualisation, explaining neural networks or reinforcement learning systems. Even seemingly very similar work such as [19] (on explaining AI planner decisions) focuses on a case where a user can ask why a particular action is part of a plan and can replace it with another applicable action at that point in the plan. It thus aims for a very specific sort of explanation, where the user already must have a pretty good idea how a particular outcome was generated and wants to check a particular action. Our approach is more general than these publications and could in principle implement most of the ones based on some planning algorithm by adding specific heuristics to our algorithm to limit the explanation to specific parts of the explanation set.

Miller [20] surveys a broad range of work in the social sciences. His key argument is that work on explainable AI should take account of the work that has been done on human explanation of behaviour. He argues that the following three key findings from social sciences literature are especially relevant when developing computational mechanisms for explanations. Firstly, that explanations are *contrastive* i.e. that they answer questions of the form "why did you do X ... instead of Y?" (although the contrastive part "instead of Y" may be implicit). Secondly, that they are *selected*, i.e. humans do not give complete explanations that cover all factors, but they select relevant factors and present those. Thirdly, that explanations are *social*, i.e. they are presented relative to what the explainer believes the listener knows. The second and third finding are, of course, related. Based on the assumed knowledge of a listener those explanations can be selected that fit the model of the listener and provide a (good enough) model for the listener to be able to infer the correct reasons for an action to be performed. However, the paper does not itself propose any specific explanation mechanisms, instead it provides a broad survey. In doing so, it raises questions, poses challenges to the field of explainable AI, and identifies various factors that affect the design of explanation mechanisms. For example, that humans give explanations of the behaviour of intentional entities in terms of folk psychological constructs. Another example is the importance of abnormality in explanations: a factor that is unexpected, or abnormal, is more important in an explanation, even though other explanatory factors might be closer in time (we return to this in our discussion of future work in Section 7).

An approach closer to our work is that of explanation as model reconciliation, where the assumption is that in realistic scenarios humans have domain and task models that differ significantly from that used by the agent [21]. This assumption is supported by psychological studies that observed that explanations are *"typically contrastive... the contrast provides a constraint on what should figure in a selected explanation..."* [22]. However, this approach does not link to the values/valuings, beliefs and desires of the human in the loop and is therefore less adequate to connect to the reasons behind the decisions taken in the process. Additionally, it assumes that we *know* the human's mental model, which is a fairly strong assumption, and one that we do not make.

Finally, it is worth noting that the term "explaining" can be applied to other things. In this paper we tackle the problem of an autonomous system explaining its *behaviour* to a human observer. There is also work (e.g. Milliez et al. [23]) that considers an autonomous system (specifically a robot) explaining a *plan* with a human collaborator. The problem being addressed is different. In the work of Milliez et al. there is a plan, and the explanation is of the next steps that the human needs to perform. In our work the explanation is of the observed behaviour, i.e. the past actions of the software, rather than the immediate future actions of the human.

2. Formal setting

The basic formal structure is that of a goal tree, which indicates how the goals of an agent are achieved through subgoals and in the end by actions performed. The structure of a goal tree is an abstraction of a wide range of BDI agent platforms (e.g. JACK [24,25], JAM [26], dMARS [27], PRS [28,29], UM-PRS [30], Jason [31], SPARK [32], Jadex [33], IRMA [10]) where agents are specified using plans which have a trigger, a context condition, and a plan body.

A *goal tree*[4] is a tree of nodes, where non-leaf nodes (i.e. nodes that have children) are (sub-)goals or plans,[5] and leaves (nodes with no children) are actions. Formally, we model a goal tree GT as being either a tuple (N, A) of a name N and an action[6] A, or a tuple (N, G) where N is the name and G is a combination of n sub-goal-trees GT_i which can be in sequence (Seq) or unspecified order (And), or a choice (Or) of options. An option O_i is defined as being a goal tree GT_i with an added context condition C_i, formally: $O_i = (C_i, GT_i)$. This means that each option O_i has a sub-goal-tree GT_i and a condition C_i (a propositional logic formula[7]) indicating in which situations that sub-goal can be selected to realise the parent goal.

$$
\begin{aligned}
GT &::= (N, A) \mid (N, G)\\
G &::= \textsc{Seq}(GT_{1\text{-}n}) \mid \textsc{And}(GT_{1\text{-}n}) \mid \textsc{Or}(O_{1\text{-}n})\\
O_i &= (C_i, GT_i)\\
C_i &= \text{a formula in propositional logic}
\end{aligned}
$$

Where we use $GT_{1\text{-}n}$ to abbreviate a number of goal-trees $(GT_1, \dots, GT_n)$, and $O_{1\text{-}n}$ to abbreviate the sequence of options $(O_1, \dots, O_n)$. For example, the goal tree that indicates that one can getShopCoffee by the sequence of three actions goto(Shop), pay(Shop), and then getCoffee(shop) would be represented as the following formal structure:

$$(\mathit{getShopCoffee}, \textsc{Seq}((\mathit{goto}(\mathit{Shop}), \mathit{goto}(\mathit{Shop})), (\mathit{pay}(\mathit{Shop}), \mathit{pay}(\mathit{Shop})), (\mathit{getCoffee}(\mathit{Shop}), \mathit{getCoffee}(\mathit{Shop}))))$$

Each action node (N, A) has associated pre-conditions and post-conditions denoted respectively $pre(A)$ and $post(A)$. Note that not all of the information that we use would necessarily need to be provided by the designer. For instance, action post-conditions and pre-conditions could perhaps be learned from observation. By defining actions in terms of their pre- and post conditions we abstract and hide the details of how the action is performed. For example, the action getCoffee(office) has the pre-condition $\mathit{havePod} \wedge \mathit{at}(\mathit{office})$ and the post-condition $\mathit{haveCoffee}$.

Each node in the goal tree has a unique name that can be used to refer to it. When naming a node that is a goal, it is good custom to use the achievement condition of that goal as the name of that node. E.g. "have(coffee)" for the goal node that accomplishes the goal of getting coffee. However, it should be noted that formally the name is just a label of the node and has no logical status as truth condition.

Intuitively, a goal tree is executed as follows. If the tree is simply an action, then the action is performed (assuming its pre-conditions hold). If the tree is an And or Seq decomposition, then all of the sub-goals are executed, either in the specified sequential order (Seq), or in some, unspecified, order. Finally, if the tree is an Or decomposition, then an applicable option (i.e. one whose condition C_i is believed to hold in the current situation) is selected and executed. Many BDI platforms provide a way to handle failure, which we discuss later in the paper.

Note that while there is a general intuition about how behaviour is generated from a BDI model, the operational details differ between different platforms and implementations. For instance, the BDI model does not specify the order in which alternative choices are tried, or at which precise state conditions are checked [38, Section 2.3]. In this paper we assume a BDI model based on goal trees and we also assume that the listener assumes such a goal tree as the deliberation mechanism of the agent. In the remainder of the paper we will comment where the specific details of the assumed model make a difference.

We now turn to valuings. We need to incorporate them as an additional concept, distinct from beliefs or desires, in order to be able to provide explanations of the form "I could have done A or B, and I chose B because I preferred it because it was better". There are a number of ways in which we could choose to represent valuings. For the purposes of this paper we remain agnostic, simply requiring that the agent has a preference between options, and that this preference can depend on the situation. Specifically, following [12], we incorporate valuings by annotating nodes in the goal tree with an abstract evaluation of key aspects of their effects,[8] and defining a preference ordering between value annotations. We need a preference in order to be able to explain not just that an aspect of the outcome of action A was valued, but that we chose A because its outcome was *more valued* than the alternative option B. In other words, we represent valuings using preferences between options, so our valuings are preferences between options (but not all preferences represent valuings). It would also have been possible to represent valuings using rewards and costs, assuming that one could determine numerical

[4] We use "goal" to be consistent with the literature, e.g. [34–37].

[5] Our formalism is more flexible than traditional BDI goal-plan trees in that it does not insist on goals only having plans as children, and plans only having sub-goals and actions as children. Rather, we represent a goal as a node that is OR-decomposed, and a plan as a node that is either AND or SEQ decomposed.

[6] For actions we assume that the name of the node and the name of the action coincide, i.e. that $A = N$. For example, in the running example we would have an action node $(\mathit{getPod}, \mathit{getPod})$.

[7] For our purposes propositional logic is adequate.

[8] By "key aspects" we mean those that are relevant to the agent evaluating which options it prefers. For example, when selecting between coffee from the kitchen and coffee from the shop, it may be that the quality and cost of the coffee are important aspects that can affect which option is preferred. Or it could be that only the coffee quality matters, in which case cost would not be a key aspect.

values for these. The agent's valuings, i.e. which options it appreciates more or less, are specific to a given situation. For example, if the bank account balance is running low, then cost might become a more important factor.

The valuings are founded on the agent's values, which are the underlying drivers. In other words, we distinguish between *values* and *valuings*. The former are specific aspects of an outcome that are preferred (e.g. preferring the quality of the coffee to be high). On the other hand, the latter are preferences between options in a given context, taking into account all relevant values. For example, in a given context we might be choosing between free but bad coffee, and good but expensive coffee. The values are coffee quality and coffee cost. The valuing is a preference between the two options, for instance preferring the cheap coffee, despite the quality being low.

For example, an agent might value good coffee, saving money, and saving time. These aspects are the measurable criteria indicating whether a certain value is promoted by a course of action. However, as already can be seen by the fact that we have multiple aspects (thus creating a kind of multi-criteria optimisation), they do not completely determine the agent's valuing. E.g. an agent might prefer good coffee over bad coffee, but decide to get bad coffee for free at the end of the month when his salary runs out and get good coffee once his salary is in. So, the weighing of the different aspects and thus the resulting valuings might not be fixed, but depend on the context. Also he might prefer the best coffee from the shop, but not want to spend much time to get it when he is finishing a paper for a deadline. Thus, in general a valuing (or preference) for an option is based on the values, but also on the current situation and practical considerations.

In explanations we need to capture both values and valuings (as well, of course, as other aspects, such as goals and conditions). Both are important: to explain why a given course of action was chosen we use the valuing. To provide a more detailed explanation for why one course of action was preferred over another we drill down into the values. We therefore add valuings and values to the action or plan nodes of the goal tree. Specifically, (N_i, A_i, V_i) denotes that action[9] A_i is annotated with value effects V_i (a tuple of key aspects that A_i is influencing), for example (veryGood, high, low) would indicate an option that provides very good coffee, at a location that is close (low distance), but with a high price. We then define $V_1 <_C V_2$ to indicate that tuple V_2 is preferred to tuple V_1 (i.e. the agent is *valuing* V_2 over V_1) when the agent believes condition C holds. For the purposes of this paper, we simply need that there is a known ordering, i.e. that for a given situation, we know whether $V_1 <_C V_2$. As the previous paragraph indicates, the situations and reasons for why this might be the case are complex (we return to this in Section 4.3). For example, we might formalise the following valuings, which capture that when Jo is low on money or busy he prefers the free office coffee which is good quality, but when he has time and money, he prefers the very good coffee from the shop.

$$(\text{veryGood}, \text{high}, x) <_{bank\_balance<\$400 \vee busy} (\text{good}, \text{none}, x)$$

$$(\text{good}, \text{none}, x) <_{bank\_balance\geq\$400 \wedge \neg busy} (\text{veryGood}, \text{high}, x)$$

We next extend this to define $A_1 \prec_C A_2$ to denote that A_2 is preferred over A_1 under condition C. This is defined formally as follows, where $\mathcal{B}(N)$ stands for the beliefs of the agent just *prior* to the execution of the goal tree named N. Note that recording, and retrieving, a history of beliefs during execution can be done efficiently [39].

$$A_1 \prec_C A_2 \textbf{ iff } (\mathcal{B}(N_1) \models C) \wedge (\mathcal{B}(N_2) \models C) \wedge V_1 <_C V_2 \wedge (N_1, A_1, V_1) \wedge (N_2, A_2, V_2)$$

This definition states that an action[10] A_2 is preferred over A_1 iff its value effect annotation V_2 is preferred to the other action's value effect annotation, i.e. $V_1 <_C V_2$ where C is the condition that holds prior to the action's execution.

We generalise this to preferences over sets by defining A being preferred to a set of actions iff it is preferred to each of the actions in the set, formally:

$$\{A_1, \ldots, A_n\} \prec_C A \textbf{ iff } \bigwedge_{1\leq i\leq n} A_i \prec_C A$$

This definition can be used to induce preferences between options in the goal tree based on the valuings of the actions in each option. We will not expand this further in this paper, but refer to [12] for more on this topic. For this paper it is enough to know that we now have valuings available in each node of the goal tree.

Finally we say that V_2 is preferred to V_1 in *all* conditions C as:

$$V_1 < V_2 \textbf{ iff } \forall C : \mathcal{B}(N) \models C \rightarrow V_1 <_C V_2$$

Although this is not used in the examples in this paper, this can be used when generating explanations. For example, using the running scenario, this would be the case if Jo would always go for good coffee (assuming this option is always available). When $V_1 < V_2$ then an explanation does not have to include the condition that happened to be true at that moment, as that condition had no effect on this order of valuings.

[9] We also allow for non-action nodes to be annotated with value effects, written (N_i, G_i, V_i).

[10] This definition, and the others in this section, can be applied to non-action nodes where these have been annotated with value effects.

$$
\begin{array}{rcl}
GT & ::= & (N_i, A_i, V_i) \mid (N_i, G_i, V_i) \\
G & ::= & \textsc{Seq}(GT_{1\text{-}n}) \mid \textsc{And}(GT_{1\text{-}n}) \mid \textsc{Or}(O_{1\text{-}n}) \\
 & & \text{where } GT_{1\text{-}n} \text{ abbreviates } (GT_1, \ldots, GT_n) \\
 & & \text{and } O_{1\text{-}n} \text{ abbreviates } (O_1, \ldots, O_n) \\
O_i & = & (C_i, GT_i) \\
C_i & = & \text{a formula in propositional logic} \\
V_i & = & \text{a tuple of aspects being influenced by the action } A_i \\
V_1 <_C V_2 & & V_1 \text{ is preferred to } V_2 \text{ when } C \text{ is believed to hold} \\
A_1 \prec_C A_2 & \textbf{iff} & (\mathcal{B}(N_1) \models C) \wedge (\mathcal{B}(N_2) \models C) \wedge V_1 <_C V_2 \wedge (N_1, A_1, V_1) \wedge (N_2, A_2, V_2) \\
\{A_1, \ldots, A_n\} \prec_C A & \text{iff} & \bigwedge\limits_{1 \le i \le n} A_i \prec_C A \\
V_1 < V_2 & \text{iff} & \forall C : \mathcal{B}(N) \models C \rightarrow V_1 <_C V_2
\end{array}
$$

Fig. 1. Collected definitions for Section 2.

We now write (N, G_V) to denote a goal tree where the actions have been annotated with value effects, and O_V to denote an annotated option, e.g. $O_{iV} = (C_i, (N_i, G_{iV}))$. Before we give the formal semantics of goal trees we introduce some more notation that will ease the definitions. We write $(G_{V1\text{-}n})$ (resp. $(O_{V1\text{-}n})$) to abbreviate $((N_1, G_{V1}), \ldots, (N_n, G_{Vn}))$ (resp. $(O_{V1}, \ldots, O_{Vn})$). We also sometimes abbreviate (N, G_V) to G_{VN} for readability, and, where the name is not important, just write G_V for G_{VN}.

Fig. 2 shows a goal tree for obtaining coffee in the setting of this running example. The figure also shows the pre- and post-conditions. As noted earlier, we follow [12] in capturing value effects as annotations. In this case each annotation V_i is of the form (coffee quality, cost, distance), respectively drawn from {veryGood, good, bad}, {none, low, high}, and {none, low, medium, high} where the office and kitchen are close to each other ("low" distance), and the shop is far from both kitchen and office ("high" distance). We define the function $dist(L_1, L_2)$ to return the distance between locations L_1 and L_2. Fig. 1 summarises the definitions introduced in this section.

The formal semantics of a goal tree is obtained by mapping it to a sequence of (valued) actions. A goal tree can yield multiple such sequences, so formally $[\![(N, G_V)]\!]$ is a set of sequences of actions with the expectation that each sequence of actions (which we will refer to as *trace* T) achieves the goal denoted by N. The semantics is defined by the following equations.

$$
\begin{aligned}
[\![(N, A, V)]\!] &= \begin{cases} \{\langle (A, V) \rangle\} & \text{if } \mathcal{B}(N) \models pre(A) \\ \{\langle\rangle\} & \text{otherwise} \end{cases} \\
[\![(N, \textsc{And}(G_{V1\text{-}n}))]\!] &= ([\![G_{1V}]\!] \mathbin{|||} \ldots \mathbin{|||} [\![G_{nV}]\!]) \\
&\quad \text{where } X \mathbin{|||} Y = \{z \mid x \in X \wedge y \in Y \wedge z \in x || y\} \\
&\quad \text{and } x || y = \text{set of all interleavings of } x \text{ and } y \\
[\![(N, \textsc{Seq}(G_{V1\text{-}n}))]\!] &= ([\![G_{1V}]\!] \bigcirc \ldots \bigcirc [\![G_{nV}]\!]) \\
&\quad \text{where } X \bigcirc Y = \{x \circ y \mid x \in X \wedge y \in Y\} \\
&\quad \text{and } x \circ y = \text{sequences } x \text{ and } y \text{ concatenated} \\
[\![(N, \textsc{Or}(O_{V1\text{-}n}))]\!] &= \Big(\bigcup_{O_{iV} = (C_i, (N_i, G_{iV})) \wedge \mathcal{B}(N) \models C_i} [\![(N_i, G_{iV})]\!] \Big)
\end{aligned}
$$

The first clause of the above definition of the semantics of the goal tree ensures that the set of action sequences only contains sequences that the agent believes are executable. The consequence of this requirement is that a goal tree does not necessarily achieve the goal as denoted by N. E.g. if the goal tree is the tuple $(open(door), open(door))$ and the pre-condition to open the door is $have(key)$ and $\mathcal{B}(open(door)) \models \neg have(key)$ then the semantics of the goal tree is $\{\langle\rangle\}$ and no action is performed and thus the door is not opened.

The second clause defines the semantics of an unordered AND as the parallel interleaving of the semantics of all of the sub-goals. Similarly, the third clause defines the semantics of an ordered SEQ as the concatenation of the semantics of the (ordered) sub-goals. Finally, the last clause defines the semantics of a choice OR as the union of the semantics of the sub-goals, but limited to only consider those sub-goals whose condition C_i holds, which is specified by $\mathcal{B}(N) \models C_i$.

3. Generating explanations

This section defines how to explain why a given action was selected, given the trace of actions, the goal-plan tree, and the beliefs that were held during execution. We begin (Section 3.1) with some preliminary definitions, and then proceed to

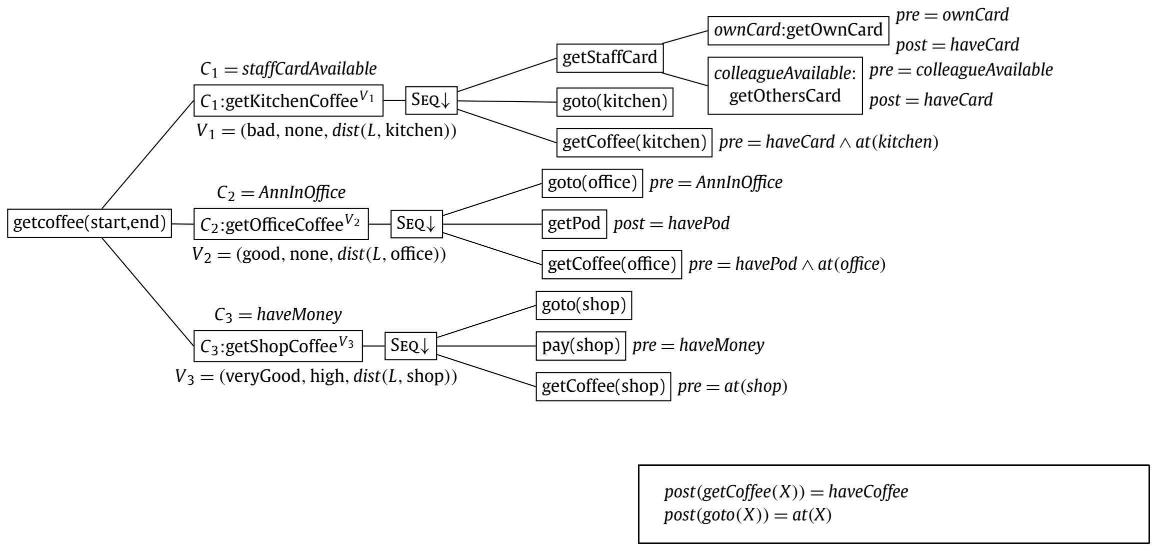


Fig. 2. Running Example. Notation: option nodes are written $C : N$ where C is the context condition and N the node name; V_i are value effect annotations of the form (quality, cost, distance), where $dist(L_1, L_2)$ is the distance between locations L_1 and L_2; and children are OR-refined, except where indicated with a Seq, in which case they are ordered from top to bottom.

define the core explanation function (Section 3.2). We then enhance the core explanation function, extending it with explanatory factors relating to preparatory factors (Section 3.3), and to motivations (Section 3.4). We then add failure handling (Section 3.5) and filtering of the set of explanatory factors (Section 3.6).

3.1. Preliminary definitions

As discussed in the introduction, an explanation is given in terms of reasons which can be desires (goals), beliefs, or valuings. More precisely, an explanation is either $\bot$ (representing that the question does not make sense, e.g. "why did you do X?" when X was not done), or a set of explanatory factors. Factors can be beliefs that were held, desires that were pursued, and valuings. Valuings are explained as "I preferred V to $\{V_1, \ldots, V_n\}$". Note that for the purposes of explanation we do not draw a distinction between desires and goals. While there is a distinction between these terms in the BDI literature, for the purposes of presenting an explanation, we use "I desired X" and "I had the goal to X" interchangeably. Additionally, when presenting explanations we simplify the presentation of the agent's beliefs, eliding the "I believed that ...", since the explanation is given from the agent's perspective. For instance, instead of saying "I believed that Ann was not in her office" we simply would say "Ann was not in her office". We are *not* assuming that the agent's beliefs and reality coincide: when the agent provides an explanation that includes a condition, such as that "Ann was not in her office", this is stated by the agent, and hence is implicitly their belief, not necessarily the reality. This is reflected in the definition below, where we define possible explanatory factors to include *conditions* (rather than beliefs), and *desires* (rather than goals).

In addition to these reasons, which follow Malle, We also have forward-looking explanatory factors of the form "I did N_1 in order to be able to later do N_2" ($N_1 \mapsto N_2$). Finally, as discussed towards the end of this section, one possible type of explanatory factor is an indication that a particular option was attempted but failed. For example, "I chose to get coffee from the kitchen because I tried to buy it from the shop but failed" (e.g. shop was closed). Finally, we also define $\top$ to be an explanation that carries no information. Clearly, $\top$ is not a useful explanation to a user, but it is used in the formal definitions below where some parts of the process do not provide any useful information.

In this paper we focus on generating the set of all building blocks for an explanation. That is why the explanation will look like a set of elements. Of course, this set has to be filtered when it gets very large in order for the explanation to be comprehensible. We do not consider this filtering in the general case, but Section 3.6 presents a number of cases where we can filter out information that is argued to be clearly redundant, i.e. can be removed without risking the loss of information. In addition to filtering, we also have to generate natural language text out of this set, which we discuss in Section 4.3.

Formally an explanation *Exp* can be defined as:

$$
\begin{aligned}
Exp &::= \bot \mid \{Exp'_1, \ldots, Exp'_n\} \\
Exp' &::= Condition \mid \{V_1, \ldots, V_n\} \prec V \mid \mathsf{Desire}\ N \mid N_1 \mapsto N_2 \mid \mathsf{Tried}\ N \mid \top
\end{aligned}
$$

3.2. The core explanation function

The definition of the explanation function E is with respect to the goal-tree. Specifically, $E^T_A(G_{VN'})$ is "explain action A using the tree (N', G_V) and trace T". Note that we assume that explanations are requested for observable actions (which are the leaves of the tree, and which appear in the trace). The definitions can be easily extended to allow explanations to also be provided for (sub-)goals by including sub-goals in the trace.

Notation: We define $n(G_V)$ as denoting the set of all node names occurring in the tree rooted at G_V. We define $T^{\prec A}$ to be the part of the trace T that occurs before A. Note that if $A \notin T$ then we simply define $E^T_A(G) = \bot$, otherwise the rest of the definitions below apply.

$$
\begin{aligned}
E^T_A(G_{VN'}) &= \bot\text{, if } A \notin T \\
E^T_A(A_{VN'}) &= \begin{cases} \{\} & \text{if } pre(A_{VN'}) = \top \\ \{pre(A_{VN'})\} & \text{otherwise} \end{cases} \\
E^T_A(\textsc{And}(G_{iV})_{N'}) &= \bigcup_{G_{iV}: n(G_{iV}) \cap T^{\prec A} \neq \emptyset} E^T_A((N_i, G_{iV})) \\
E^T_A(\textsc{Seq}(G_{iV})_{N'}) &= \bigcup_{G_{iV}: n(G_{iV}) \cap T^{\prec A} \neq \emptyset} E^T_A((N_i, G_{iV})) \\
E^T_A(\textsc{Or}(O_{iV})_{N'}) &= \begin{cases} \displaystyle\bigcup_{G_{iV}: n(G_{iV}) \cap T^{\prec A} \neq \emptyset} E^T_A((N_i, G_{iV})) \cup \Theta & \text{if } A \in n(G_{iV}) \\ \displaystyle\bigcup_{G_{iV}: n(G_{iV}) \cap T^{\prec A} \neq \emptyset} E^T_A((N_i, G_{iV})) & \text{otherwise} \end{cases} \\
&\text{where } \Theta = pref(O_{iV}, \{O_{1V}, \ldots, O_{nV}\})
\end{aligned}
$$

The intuition for this definition is that whenever the execution reaches a point where there is a choice, then the reason why execution took the path it did, is an explanation. For example, suppose that action A is reached, and, since its pre-condition holds, it is performed. This is a choice point: had the action's pre-condition not held, another path would have been taken. So therefore part of the explanation for why things went the way they did is that the action's pre-condition held at that point in time. Note that all of the conditions that are included as explanatory factors (pre-conditions, context conditions) are actively considered (i.e. checked) as part of the agent's reasoning cycle.

The function E realises this by collecting explanation factors by traversing the relevant parts of the goal tree. A part of the goal tree is relevant if it occurs in the execution trace before beginning the process of executing the action A that is being explained. Simply, if something occurs before A, then it can affect A. This relevance condition is checked in the definition $G_{iV} : n(G_{iV}) \cap T^{\prec A} \neq \emptyset$, which finds all sub-goals G_{iV} which contain at least some node that appears in the prefix of the trace T before A.

In the case of an action A the explanation collected is the action's pre-condition. This is because whether the pre-condition holds or not affects the execution of the action, and consequently, whatever comes after it.

In the case for Seq and And the explanation collected is simply the explanation associated with the sub-goals. For Seq this could be refined in case the post-condition of one action implies the pre-condition of the next action. In that case the pre-condition does not have to be mentioned anymore (as we assume it to be true after the execution of the earlier action).

In the case for Or there is an additional explanation relating to why the particular option taken was chosen. This is defined by the function *pref* which provides an explanation for why the selected option, O_{iV}, is preferred to the other options.

The definition of *pref* is complex, reflecting that the choice between options is at the heart of the endeavour of explanation. Intuitively, given a choice-point $(N, \textsc{Or}(O_{1V}, \ldots, O_{nV}))$, where O_{iV} was selected, *pref* first considers the conditions C_j. The explanation consists of three parts:

1. the condition of the selected sub-goal being true ("C_i");
2. for each condition C_j $(j \neq i)$ that is believed to be false at the decision point, the explanation includes that the condition was false (but see below):

$$\bigcup_{C_j : \mathcal{B}(N) \not\models C_j} \neg C_j$$

3. for each condition C_j $(j \neq i)$ that is true at the decision point, the explanation includes an indication that the selected sub-goal was preferred to the other available sub-goals[11] in the current situation:

$$\{V_j \mid j \neq i \wedge \mathcal{B}(N) \models C_j\} \prec V_i$$

Formally we therefore define $pref(O_{iV}, \{O_{1V}, \ldots, O_{nV}\})$ as:

$$\{C_i\} \cup \left(\bigcup_{C_j : \mathcal{B}(N) \not\models C_j} \neg C_j \right) \cup \{\{V_j \mid j \neq i \wedge \mathcal{B}(N) \models C_j\} \prec V_i\}$$

where we define $\{\} \prec V_i$ to be equivalent to $\top$, in other words, we do not generate that part of the explanation if there actually was no alternative option. In this situation, the explanation includes the conditions C_j that explain why the other options were not available (as defined in point 2 above).

Now let us go through a (simple) example to show how explanations are generated from the goal tree and the particular trace that has been followed. Assume in the scenario given in Section 1 the situation in which C_2 is false, and the other C_i are true. Then the preference explanation for why C_3 was chosen[12] is:

$$\{C_3, \neg C_2, \{V_1\} \prec V_3\}$$

in other words: "I chose to get coffee from the shop because I had money, and Ann was not in her office, and I prefer V_3 to V_1 in this situation". On the other hand, in a situation where all C_i are true and C_3 is selected, the explanation would take the form:

$$\{C_3, \{V_1, V_2\} \prec V_3\}$$

[11] This is one place where the details of the BDI model matter. If the children of OR nodes are considered in a particular order and the BDI model is being used post-hoc, then instead of considering $j \neq i$ we only consider options that appear earlier, i.e. $j < i$. If the BDI model is being used to both generate and explain behaviour then, assuming the implementation captures which options were considered when O_i was selected, we would only consider those options that were considered in the explanation.

[12] As noted earlier, an explanation would actually be requested for an observed action, in this case perhaps goto(shop).

in other words: "I chose to get coffee from the shop because I had money, and I prefer V_3 to both V_1 and V_2 in this situation".

Note that these explanations just present the set of annotations, indicating an overall preference between them. However, we could provide more precise explanations by taking into account the known priorities of factors, e.g. that coffee quality is the overriding factor, followed by money, then distance. So, for example, for the first example above, we could explain more precisely that the reason why V_3 was preferred to V_1 is that it yields better quality coffee.

On the other hand, suppose that the second option (office coffee) was selected, even though all three C_i were true. In this situation, in order to explain why $\{V_1, V_3\} \prec V_2$ we would need to use two factors. We could note that V_2 was preferred to V_1 because it had better coffee, and, perhaps, that it was preferred to V_3 because cost was a factor at this point in time (we return to this in Section 4.3).

3.3. Adding preparatory actions

We now extend the definition to also include preparatory actions. For example, an explanation for "why did you go to the kitchen?" could also be "because I need to be in the kitchen in order to get coffee". This is where an action's post condition is (a necessary part of) the pre-condition of a future action. Specifically, a preparatory reason applies to explain an action A when (i) the post-condition of A is required in order for the pre-condition of another action A' to hold, and (ii) A' always occurs after A (to be precise: if A' occurs, then it must be preceded by A).

We now need to formalise these two conditions. The second condition, A' occurring after[13] A, can be straightforwardly formalised by considering the structure of the tree. A simple case of "occurs before" is where A and A' are both direct children of a Seq decomposition, with A before A'. More generally, the only situation where we can guarantee that A and A' can never occur in the wrong order is when they have a common ancestor, that common ancestor node is a Seq decomposition, and, of course, the sub-tree containing A is prior to that containing A'. We therefore define:

$$before(A, A') \equiv \exists T.T = (N, \textsc{Seq}(\ldots T_A \ldots T_{A'} \ldots)) \wedge ancestor(T_A, A) \wedge ancestor(T_{A'}, A')$$

where $ancestor(T, A)$ is true iff A is a child of T, or one of T's children is an ancestor of A.

To see that this definition is correct, consider the following cases (which together exhaust all possibilities). Firstly, if there is no common ancestor (which might be the case if we consider an agent with multiple concurrent goals), then obviously there is no constraint on the relative timing of A and A'. Secondly, if there is a common ancestor that is an And node, then semantically the sub-tree T which contains A and the sub-tree T' which contains A' are pursued (executed) in parallel, so A and A' can occur in either order. Finally, if there is a common ancestor that is an Or node, then it is possible for A' to occur without A.

Turning to the first condition, that the post-condition of A is required for the pre-condition of A' to hold, an obvious formalisation is simply $post(A) \rightarrow pre(A')$. But A's post condition may be only *part* of the pre-condition. For example, the action getPod only achieves havePod, so $post(\text{getPod}) \not\rightarrow pre(\text{getCoffee(office)})$. We therefore formalise "required" as "without it, things don't work", i.e. if A's post-condition fails to hold, then the pre-condition of A' also must fail to hold: $(\neg post(A)) \rightarrow (\neg pre(A'))$. This assumes that $post(A) \neq \top$. In our setting, where pre and post conditions are conjunctions of positive atoms, this is equivalent (viewing the conjunctions as sets) to $post(A) \neq \emptyset \wedge post(A) \subseteq pre(A')$. However, we actually use the condition $post(A) \cap pre(A') \neq \emptyset$. While this condition is weaker (less precise), it is needed to cater for actions that have post-conditions with more than one effect. If $post(A)$ has two propositions, then we not require that both of them are part of $pre(A')$, only that at least one of them is in $pre(A')$.

Combining, we therefore have:

$$link(A, A') \equiv before(A, A') \wedge post(A) \neq \emptyset \wedge post(A) \cap pre(A') \neq \emptyset$$

Having defined the concept of a link, we then extend the explanation function by also including preparatory action explanations. When explaining an action A given goal tree G_V and trace T, we add to $E_A^T(G_V)$ the set of links $A \mapsto A'$ where $A' \in n(G_V) \wedge link(A, A')$. So, for example, an alternative explanation for why the agent performed the action getPod is that it was required for the subsequent getCoffee(office) action.

However, we actually want to add not just direct links, but also transitive links. We therefore define the transitive closure of the link relationship in the obvious way:

$$tlink(A, A') \equiv link(A, A') \vee (\exists C : link(A, C) \wedge tlink(C, A'))$$

The definitions so far only deal with preparatory links between actions. To extend them to deal with links between goals, we need to extend the definitions, following previous work on summary information [34,40,41]. Specifically, we extend pre and post conditions to intermediate goals, inferring them.

[13] Note that "after" does not mean "immediately after". What matters is that the goal tree constrains the order so that A' must occur after A, i.e. A' cannot occur before A.

For an OR-decomposed node, we have a pre (respectively post) condition if the condition is required for all options. Conversely, for an AND-decomposed node, we have a pre (resp. post) condition if the condition is required for any option, since all sub-nodes will need to be achieved. This is an estimate: for instance, depending on the order in which the sub-goals are achieved, a post-condition of one sub-goal might be undone by another goal. Finally, for a SEQ-decomposed node we know the order in which sub-goals are achieved. We therefore can simply define the pre-condition of $\textsc{Seq}(G_{V1\text{-}n})$ as being $pre(G_{1V})$. However, we can also include those parts of the pre-condition of a later G_{iV} that are not post-conditions of earlier G_{jV}. This is because if an earlier G_{jV} has something as a post-condition, then it is not required as a pre-condition for the whole sequence. The following definition views pre/post conditions as being sets (conjunctions) of propositions. For the definition of OR recall that $O_{iV} = (C_i, (N, G_{iV}))$.

$$
\begin{aligned}
pre(\textsc{Seq}(G_{V1\text{-}n})) &= pre(G_{1V}) \cup \bigcup_{1<i\leq n} pre(G_{iV}) \setminus post(\textsc{Seq}((N_1, G_{V1}), \ldots (N_{i-1}, G_{V(i-1)})))\\
post(\textsc{Seq}(G_{V1\text{-}n})) &= \bigcup_{1\leq i\leq n} post(G_{iV})\\
pre(\textsc{And}(G_{V1\text{-}n})) &= \bigcup_{1\leq i\leq n} pre(G_{iV})\\
post(\textsc{And}(G_{V1\text{-}n})) &= \bigcup_{1\leq i\leq n} post(G_{iV})\\
pre(\textsc{Or}(O_{V1\text{-}n})) &= \bigcap_{1\leq i\leq n} pre(G_{iV})\\
post(\textsc{Or}(O_{V1\text{-}n})) &= \bigcap_{1\leq i\leq n} post(G_{iV})
\end{aligned}
$$

We also define $pre(\textsc{Seq}(G_{1V})) = pre(G_{1V})$ for sequences with a single item.

3.4. Adding motivations

Finally, in addition to the explanation function E, which yields beliefs and valuings, and the link function, we also add explanations in terms of parent goals: these explain the agent's motivation for pursuing the current course of action.

This factor is simple: we also include in the explanation all the ancestors of the node being explained. However, we do not include ancestors that are OR refined. The reason is that for an OR refined goal, the child goals convey more information: they capture the specific approach taken to achieve the parent goal. For example, in explaining why the agent did *goto*(*shop*) it would not be helpful to include both Desire: getShopCoffee and Desire: getcoffee. The latter is implied by the former, since getShopCoffee is one particular way of getting coffee. This argument applies in general: an OR refined goal conveys less information than one of its children, and therefore if the child (e.g. getShopCoffee) is included in an explanation, there is no information lost by leaving out the parent (e.g. getcoffee). This can be seen as a form of filtering.

Pulling all the pieces together, the overall explanation function is then:

$$\mathcal{E}_A^T(G_{VN'}) = E_A^T(G_{VN'}) \cup \{A \mapsto N'' \mid N'' \in n(G_V) \wedge link(A, N'')\} \cup \{\text{Desire}(N''') \mid ancestor(N''', A) \wedge \neg isOR(N''')\}$$

For example, given the scenario described, in a situation where C_1 and C_3 hold, but not C_2, the possible factors that could be used to explain why the agent did "*goto*(*shop*)" are: {haveMoney, ¬AnnInOffice, {⟨bad, none, high⟩} ≺ ⟨ veryGood, high, none⟩, goto(shop) ↦ getCoffee(shop), Desire: getShopCoffee}. In English, these are: I had money, Ann was not in her office, I preferred V_3 to V_1 (perhaps because it yields better quality coffee), I needed to go to the shop in order to do getCoffee(shop), and I desired to getShopCoffee.

3.5. Adding failure handling

We now extend the explanation mechanism to handle failure handling. Informally, actions can fail, and the failure of a node is handled by considering its parent.[14] If the parent is a SEQ or an AND then it too is considered to be failed, and failure handling moves to consider that node's parent. When an OR node is reached, failure is handled by trying an alternative plan (if one exists, otherwise the OR node is deemed to have failed).

We assume that we know which actions in the trace are failed (denoted $\mathit{failed}^T(A)$). Then the condition under which a non-leaf node[15] is considered to be failed is defined as:

[14] This mechanism is common to many BDI languages.
[15] Recall that a non-leaf node is one that has children..

$$failed^T(\textsc{And}(G_{V1-n})) = \bigvee_{1 \le i \le n} failed^T(G_{iV})$$

$$failed^T(\textsc{Seq}(G_{V1-n})) = \bigvee_{1 \le i \le n} failed^T(G_{iV})$$

$$failed^T(\textsc{Or}(O_{V1-n})) = \bigwedge_{1 \le i \le n} failed^T(G_{iV})$$

We first note that the definition of the explanation function E is almost unchanged, except that in the definition of the recursive call we exclude failed nodes by adding $\neg failed^T(G_{iV})$:

$$\bigcup_{G_{iV}:n(G_{iV}) \cap T^{\prec A} \neq \emptyset \wedge \neg failed^T(G_{iV})} E^T_A((N_i, G_{iV}))$$

Extending the explanation to account for the possibility of previous failures is done by defining an extended *pref* function. Recall that the definition of *pref* has three components: the condition of the selected sub-goal being true, the conditions of those (other) sub-goals that are false, and, for those other sub-goals that have true conditions, a preference indication.

We modify the second and third components by only considering those sub-goals that have not yet been attempted. So, instead of the second component being

$$\bigcup_{C_j:\mathcal{B}(N) \not\models C_j} \neg C_j$$

we modify it to

$$\bigcup_{C_j:\mathcal{B}(N) \not\models C_j \wedge \neg failed^T(G_{jV})} \neg C_j$$

Similarly, we modify the third component to:

$$\{V_j \mid j \neq i \wedge \mathcal{B}(N) \models C_j \wedge \neg failed^T(G_{jV})\} \prec V_i$$

Finally, we add a fourth component that explains those things that have been previously attempted. Intuitively, this is of the form "... and I already unsuccessfully tried doing X". Formally we have:

$$\{\mathsf{Tried}(G_{jV}) \mid j \neq i \wedge failed^T(G_{jV})\}$$

To illustrate this definition, consider a situation where Jo has decided to getOfficeCoffee, but by the time he reaches Ann's office, Ann has had to leave for a meeting. The plan therefore fails, and Jo then recovers by electing to go to the shop. In response to the query "why did you getShopCoffee?" the explanation given is "{haveMoney, {⟨bad, none, low⟩} ≺ ⟨ veryGood, high, high⟩, Tried:getOfficeCoffee}" which can be rendered in English as "because I have money, I prefer good coffee to bad coffee, and because I tried (and failed) to get pod coffee". One interesting direction for future work (see also Section 7) is extending the explanation mechanism so it can answer the follow up question: "why did you fail to get pod coffee?" with "Ann was not in her office" (and, if the agent is aware of the reason for her absence, this could even be extended with "... because she had a meeting elsewhere.").

3.6. Filtering

We now turn to filtering. As mentioned earlier, as explanations get larger, it becomes more important to be able to *filter* them. This is challenging in that, in general, filtering risks losing information by removing explanatory factors. In this section we present a number of cases where filtering can be done without losing information, since the explanation function defined generates explanations that contain some redundancy. We return to filtering more generally in the discussion of future work (Section 7).

Our first case of redundant information that can be safely filtered concerns the second case of the *pref* function which returns those conditions C_j that are not believed to hold. However, if the conditions C_j from clause 2 in the above definition (thus those that are false) are of the form $C_j \equiv c_{j1} \wedge \ldots \wedge c_{jn}$ then the explanation can be restricted to those c_{ji} that are false. This replaces $\bigcup_{C_j:\mathcal{B}(N) \not\models C_j \wedge \neg failed^T(G_{jV})} \neg C_j$ with $\bigcup_{C_j:\mathcal{B}(N) \not\models C_j \wedge \neg failed^T(G_{jV}) \wedge c \in C_j \wedge \mathcal{B}(n) \not\models c} \neg c$, i.e. instead of including a false condition C_j, we select its constituent false parts (recalling that we view a conjunction as a set of propositions). This gives the definition of $pref(O_{iV}, \{O_{1V}, \ldots, O_{nV}\})$ as:

$$\{C_i\} \cup \left(\bigcup_{C_j : \mathcal{B}(N) \not\models C_j \wedge \neg \mathit{failed}^T(G_{jV}) \wedge c \in C_j \wedge \mathcal{B}(n) \not\models c} \{\neg c\} \right)$$
$$\cup \{\{V_j \mid j \neq i \wedge \mathcal{B}(N) \models C_j \wedge \neg \mathit{failed}^T(G_{jV})\} \prec V_i\}$$
$$\cup \{\mathsf{Tried}(G_{jV}) \mid j \neq i \wedge \mathit{failed}^T(G_{jV})\}$$

To illustrate this, consider a variant of the coffee example where $C_2 = \mathit{AnnInOffice} \wedge \mathit{PodsAvailable}$, and where Ann is in her office, but is temporarily out of pods. In this case the explanation for why the second option was not taken would include the explanatory factor $\neg \mathit{PodsAvailable}$, rather than the less specific $\neg(\mathit{AnnInOffice} \wedge \mathit{PodsAvailable})$, in other words the filtering removes $\neg \mathit{AnnInOffice}$, since in fact Ann is believed to be in her office, so the false conjunct is a more precise explanation.

A second filter that can be applied to the explanation set without losing information concerns the overlap between the condition of the chosen option (which is part of the explanation set) and the pre-condition of actions in that option (which are also part of the generated set). For example, suppose I have the option to get coffee from the coffee shop only when I have (enough) money. Then I don't have to check whether I have money once I am in the coffee shop to pay. So, the explanation of having money to buy coffee only needs to appear once as the condition for the option and not as a pre-condition of the action. For the coffee example, the effect of this filter is that some explanations no longer contain *haveMoney* twice (once from C_3, and once since it is a pre-condition of *pay(shop)*, and similarly for *AnnInOffice*). This filtering can be formalised by modifying the definition of $E_A^T(A_{VN'})$. Specifically, we replace the second case ($E_A^T(A_{VN'}) = \{\mathit{pre}(A_{VN'})\}$, if $\mathit{pre}(A_{VN'}) \neq \top$) as follows:

$$E_A^T(A_{VN'}) = \begin{cases} \{\} & \text{if } \mathit{pre}(A) = \top \\ \Psi & \text{otherwise} \end{cases}$$
$$\text{where } \Psi = \{c \mid c \in \mathit{pre}(A_{VN'}) \wedge \neg \exists N : \mathit{ancestor}(N, A_{VN'}) \wedge c \in \mathit{condition}(N)\}$$

The new condition states that any condition $c \in \mathit{pre}(A_{VN'})$ should be excluded if it also appears in the context condition of an ancestor node N (we define $\mathit{condition}(N)$ to be the context condition of a node, as a set of propositions, so for example, $\mathit{condition}(\text{getShopCoffee}) = \{\mathit{haveMoney}\}$). This filter can also be dealt with, if conditions are conjunctions of atomic propositions, by collecting the conditions as a set of propositions, which naturally avoids duplicate propositions. Of course, things can get more complex if the truth of these conditions changes in the period between the condition being established, and the action being taken that requires the condition to hold. For example, if I have money when I go to the coffee shop but give it to a friend on the way as she needs it for the bus.

A third filter concerns the case of an action A where the explanation collected is the action's pre-condition. The pre-condition is an explanatory factor because whether it holds or not affects the execution of the action, and consequently, whatever comes after it. However, if the condition in question is one that is guaranteed to be brought about by an earlier action, then we can filter it out of the set of explanatory factors without a loss of information. For example, if we wanted to explain why we did getCoffee(office), then we would not include that we had a coffee pod, or that we were in Ann's office, since both these conditions are brought about by previous actions (respectively getPod and goto(office)).

We formalise this by modifying the second case of $E_A^T(A_{VN'})$, specifically, by modifying the definition of Ψ to also include the condition $\neg \exists N : \mathit{link}(N, A_{VN'}) \wedge c \in \mathit{post}(N)$. This condition states that a condition $c \in \mathit{pre}(A_{VN'})$ should be excluded if there is a node N that brings c about ($c \in \mathit{post}(N)$) and that is before $A_{VN'}$. This gives the following complete definition for $E_A^T(A_{VN'})$:

$$E_A^T(A_{VN'}) = \begin{cases} \{\} & \text{if } \mathit{pre}(A_{VN'}) = \top \\ \Psi & \text{otherwise} \end{cases}$$
$$\begin{aligned} \text{where } \Psi = \{c \mid\ & c \in \mathit{pre}(A_{VN'}) \\ & \wedge \neg \exists N : \mathit{link}(N, A_{VN'}) \wedge c \in \mathit{post}(N) \\ & \wedge \neg \exists N : \mathit{ancestor}(N, A_{VN'}) \wedge c \in \mathit{condition}(N)\} \end{aligned}$$

Note that there is a subtle difference between the post-condition of action A being part of the pre-condition of A', and action A being actually required to make the pre-condition of A' true. The difference is that while A can make the pre-condition of A' true, it may not actually be required if the pre-condition already holds. For example, the pre-condition of buying coffee is that I have money, and an alternative plan might include an action to go the ATM to get money. However, in the situation in which I already had enough money to buy coffee this is not a preparatory action (it might be a preparatory action for buying groceries later on), whereas if I did not have money then this would be required in order to buy the coffee. We can extend the definition to cater for this case by adding the condition that $\mathcal{B}(A) \not\models \mathit{pre}(A')$, i.e. that the pre-condition of A' already held before A was done. However, we do not adopt this extension because it is safer to assume that A is actually required. The reason is that even if the pre-condition of A' already held when A was done, it may still be the case that had A not been done, the pre-condition of A' might not have held. For example, we might have had enough money to buy a coffee before going to the ATM, but it might be that other actions spend money, so that had the ATM not been visited, there would not have been sufficient money for a coffee. Furthermore, we do not have enough information available to identify these sorts of cases.

Fig. 3 shows the complete formal definition for explanation, including all the aspects (preparatory actions, motivations, and failure handling), but excluding the calculation of pre/post conditions, which can be found in Section 3.3.

$$
\begin{array}{rcl}
\mathcal{E}_A^T(G_{VN'}) & = & E_A^T(G_{VN'}) \cup \\
 & & \{A \mapsto N'' \mid N'' \in n(G_V) \wedge \mathit{tlink}(A, N'')\} \cup \\
 & & \{\mathsf{Desire}(N''') \mid \mathit{ancestor}(N''', A) \wedge \neg \mathit{isOR}(N''')\} \\
\mathit{link}(A, A') & \equiv & \mathit{before}(A, A') \wedge \mathit{post}(A) \neq \emptyset \wedge \mathit{post}(A) \cap \mathit{pre}(A') \neq \emptyset \\
\mathit{tlink}(A, A') & \equiv & \mathit{link}(A, A') \vee (\exists C : \mathit{link}(A, C) \wedge \mathit{tlink}(C, A')) \\
\mathit{before}(A, A') & \equiv & \exists T . T = (N, \textsc{Seq}(\ldots T_A \ldots T_{A'} \ldots)) \\
 & & \wedge\, \mathit{ancestor}(T_A, A) \wedge \mathit{ancestor}(T_{A'}, A') \\
E_A^T(G_{VN'}) & = & \bot \text{ , if } A \notin T \\
E_A^T(A_{VN'}) & = & \begin{cases} \{\} & \text{if } \mathit{pre}(A) = \top \\ \Psi & \text{otherwise} \end{cases} \\
 & & \text{where } \Psi = \{c \mid c \in \mathit{pre}(A_{VN'}) \\
 & & \qquad \wedge \neg \exists N : \mathit{link}(N, A_{VN'}) \wedge c \in \mathit{post}(N) \\
 & & \qquad \wedge \neg \exists N : \mathit{ancestor}(N, A_{VN'}) \wedge c \in \mathit{condition}(N)\} \\
E_A^T(\textsc{And}(G_{iV})_{N'}) & = & \displaystyle\bigcup_{G_{iV} : n(G_{iV}) \cap T^{\prec A} \neq \emptyset \wedge \neg \mathit{failed}^T(G_{iV})} E_A^T((N_i, G_{iV})) \\
E_A^T(\textsc{Seq}(G_{iV})_{N'}) & = & \displaystyle\bigcup_{G_{iV} : n(G_{iV}) \cap T^{\prec A} \neq \emptyset \wedge \neg \mathit{failed}^T(G_{iV})} E_A^T((N_i, G_{iV})) \\
E_A^T(\textsc{Or}(O_{iV})_{N'}) & = & \begin{cases} \displaystyle\bigcup_{\substack{G_{iV} : n(G_{iV}) \cap T^{\prec A} \neq \emptyset \\ \wedge \neg \mathit{failed}^T(G_{iV})}} E_A^T((N_i, G_{iV})) \cup \Theta & \text{if } A \in n(G_{iV}) \\ \displaystyle\bigcup_{\substack{G_{iV} : n(G_{iV}) \cap T^{\prec A} \neq \emptyset \\ \wedge \neg \mathit{failed}^T(G_{iV})}} E_A^T((N_i, G_{iV})) & \text{otherwise} \end{cases} \\
 & & \text{where } \Theta = \mathit{pref}(O_{iV}, \{O_{1V}, \ldots, O_{nV}\}) \\
\mathit{pref}(O_{iV}, \{O_{1V}, \ldots, O_{nV}\}) & = & \{C_i\} \cup \\
 & & \left(\displaystyle\bigcup_{C_j : \mathcal{B}(N) \not\models C_j \wedge \neg \mathit{failed}^T(G_{jV}) \wedge c \in C_j \wedge \mathcal{B}(n) \not\models c} \{\neg c\} \right) \\
 & & \cup \{\{V_j \mid j \neq i \wedge \mathcal{B}(N) \models C_j \wedge \neg \mathit{failed}^T(G_{jV})\} \prec V_i\} \\
 & & \cup \{\mathsf{Tried}(G_{jV}) \mid j \neq i \wedge \mathit{failed}^T(G_{jV})\} \\
\mathit{failed}^T(\textsc{And}(G_{V1-n})) & = & \displaystyle\bigvee_{1 \le i \le n} \mathit{failed}^T(G_{iV}) \\
\mathit{failed}^T(\textsc{Seq}(G_{V1-n})) & = & \displaystyle\bigvee_{1 \le i \le n} \mathit{failed}^T(G_{iV}) \\
\mathit{failed}^T(\textsc{Or}(O_{V1-n})) & = & \displaystyle\bigwedge_{1 \le i \le n} \mathit{failed}^T(G_{iV})
\end{array}
$$

Fig. 3. Collected Formal Definitions (excluding calculating pre/post conditions for goals).

4. Algorithm for computing explanations

Until now we have shown how we can formally define the set of explanatory factors. However, in order to generate the actual explanations we also need an efficient and scalable algorithm.

The definitions given in the previous section were implemented in Haskell. This gave an inefficient prototype, but had the advantage that, because the definitions were able to be transliterated quite directly to Haskell, it was clear that the Haskell implementation matched the definitions.

The algorithms described in this section have been implemented in Python. Note that the input goal-plan trees are encoded in Haskell, and automatically converted to the Python representation, which uses the `anytree` Python package. This conversion includes computing the links between nodes (based on which nodes occur before other nodes, and the overlaps between their post/pre-conditions).

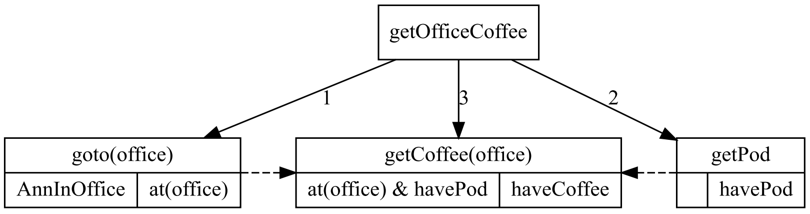


Fig. 4. GetPodcoffee branch with link (dashed arrows) attributes.

The next section (Section 4.1) describes the tree structure. We then describe the algorithms (Section 4.2), including how we generate English text (Section 4.3), and then finish with a discussion of the algorithm's computational complexity (Section 4.4 — the next section covers an empirical evaluation of the algorithm's efficiency).

4.1. Tree structure

The goal-plan tree consists of a set of nodes. Each node has several mandatory and several optional attributes. Mandatory attributes are a "name" attribute, "parent" attribute (which is null for the root), a "type" attribute which can be set to Or, Seq, And or Act, and a "bel" attribute which contains the beliefs that were held at the time the node was reached.

All children of Or nodes also have "condition" and "annot" attributes. The "condition" attribute contains all conditions that must be fulfilled in order for the choice represented by the node to be available. Conditions are represented as a list of propositions, e.g. *["AnnInOffice", "haveMoney"]*. The "annot" attribute captures the value effect annotation. This is represented as a list of numbers where each number reflects the value of each factor for this node. By default, we assume that the bigger number, the more preferable it is. For instance, for the coffee example, possible values of the coffee quality factor are (1,2,3) which correspond to (bad, good, veryGood). For the price factor, the possible values (1,2,3) are equivalent to (high [cost], low [cost], none [i.e. free cost]). And for the distance factor, the possible values (1,2,3,4) are equivalent to (high [distance], medium, low, none [i.e. no distance]), assuming that lower distances are preferred, i.e. we do not want to spend much time to get a cup of coffee. These factors are stored in a fixed, known, order. For example, coffee quality, price, and then distance to the coffee. Then the annotation of the node getShopCoffee $[3, 1, 2]$ can be translated as: very good quality coffee (3) at a high price (1) requiring one to travel a medium distance (2).

Action nodes can have "pre" and "post" attributes which are for pre- and post-conditions respectively. Action nodes also can have a "link" attribute that captures links between nodes. Recall that if (part of) the post-condition of the current node N_1 is a part of pre-condition of some future action N_2 (i.e. $pre(N_1) \cap post(N_2) \neq \emptyset$, where N_2 is always preceded by N_1), then the current action N_1 can be considered as having been done in order to make true the pre-condition of the future action N_2. In this case the current node N_1 has attribute "link" which contains the name of the node N_2 (actually we store a list of names).

In addition to the "link" attribute, we also have an inverse attribute that records when something before the current node was done, and has a post-condition that is (part of) the pre-condition of the current node. This is named "slink".[16] So, for every node N_1 that has attribute $link = [N_2]$, we have node N_2 with attribute $slink = [N_1]$. It should be noted that both "link" and "slink" attributes are represented as lists and can be empty or contain one or several nodes. For example, for the coffee example, necessary pre-conditions to hold for the *getCoffee(office)* action are *havePod & at(office)*. The second one is a post-condition of the *goto(office)* action, and the first one is a post-condition of *getPod* action. Consequently, according to our definitions of "link" and "slink" attributes, nodes *getPod* and *goto(office)* have links to the *getCoffee(office)* node, whereas the *getCoffee(office)* node has an "slink" to both the *goto(office)* and *getPod* actions, represented as a list with two members (see Fig. 5).

Another optional attribute is "sequence" which captures the ordering of children of a Seq node. However, this is not used in the algorithm (it is used in the Haskell code to work out links, which depend on the ordering).

Fig. 4 shows a graphical view of part of a goal-plan tree. Numbers on normal edges indicate the ordering of children of a Seq node, and nodes with multiple parts denote the name (top), pre-condition (bottom left), and post-condition (bottom right). The "link" attributes are represented as dashed arrows.

Fig. 5 shows the Python code corresponding to the tree in Fig. 4. Note that the attributes for node "getOfficeCoffee" also include a link to the parent (for convenience), a condition, and an annotation (the valuing) that are not shown in Fig. 4. The Python code also includes for each node the beliefs $\mathcal{B}(N)$. Otherwise, the structure of the Python-encoded tree corresponds to the tree in Fig. 2. Note that in order to allow node names to be used as Python identifiers, they are rewritten by replacing brackets with underscores, so for example, instead of "getCoffee(office)" we have "getCoffee_office_".

Finally, in addition to representing the goal-plan tree itself, we also define an additional data structure that captures information about value effect annotations. This data structure, represented as a nested dictionary, allows us to interpret value effect annotations such as $[3, 1, 2]$. It includes the names of the value attributes (e.g. coffee quality, price) and their

[16] Short for "subsumption link", since it is used to remove parts of the explanation that are subsumed by earlier parts of the tree.

```
from anytree import Node
getOfficeCoffee = Node("getOfficeCoffee", type="Seq", parent=getcoffee, annot
    ↪ =[2,3,3], condition=["AnnInOffice"], bel = ["haveMoney","
    ↪ staffCardAvailable","ownCard"])
goto_office_ = Node("goto_office_", type="Act", parent=getOfficeCoffee, link=["
    ↪ getCoffee_office_"], sequence=1, pre = ["AnnInOffice"], post = ["at(
    ↪ office)"], bel = ["haveMoney","staffCardAvailable","ownCard","
    ↪ AnnInOffice"])
getPod = Node("getPod", type="Act", parent=getOfficeCoffee, link=["
    ↪ getCoffee_office_"], sequence=2, pre = [], post = ["havePod"], bel = [
    ↪ "haveMoney","staffCardAvailable","ownCard"])
getCoffee_office_ = Node("getCoffee_office_", type="Act", parent=
    ↪ getOfficeCoffee, slink=["goto_office_","getPod"], sequence=3, pre = ["
    ↪ at(office)","havePod"], post = ["haveCoffee"], bel = ["haveMoney","
    ↪ staffCardAvailable","ownCard","at(office)","havePod"])
```

Fig. 5. Python code for getOfficeCoffee tree.

importance. The importance is represented as a list of numbers: the bigger the number, the more important the particular attribute is for making a choice. For example, for the coffee scenario where an annotation is a sequence of coffee quality, price and distance to the coffee, the importance might be represented as the list $[3, 1, 1]$, indicating that perhaps coffee quality (first item) is the most important factor in making a choice, with an importance of 3, followed by the other factors which are equally important to each other (importance of 1). We describe the nested dictionary in more detail in Section 4.3 where it is used.

4.2. Explanation algorithm

Having defined the tree structure, we can now present the algorithm that calculates the explanation (Algorithm 1).

The algorithm takes four parameters as an input: the goal-tree G (represented using the `anytree` package, as discussed above), the trace (i.e. all actions that were executed), the question Q, and the list of failed nodes F. The output from the algorithm is an explanation, which, as defined earlier, is a set of explanatory factors. Each explanatory factor can be one of a number of different types (denoted with a letter prefix): a *pre-condition* of an action (denoted with a “P:”), a condition of a choice (“C”), an indication that a node has been unsuccessfully tried (“T”), a value statement (“V”), a link (“L”), or a desire (“D”). These are explained below.

The algorithm begins by traversing the tree in a postorder from leaves to the root. This traversal marks all nodes to indicate whether they have failed, and whether they are relevant to the explanation, using new attributes `.failed` and `.relevant`, respectively (Algorithm 2, invoked on line 5 of Algorithm 1). The attribute `.failed` is set to True if a node is failed and False otherwise. The attribute `.relevant` is set to True if a node is relevant to the explanation and False otherwise. It is worth mentioning that by definition, a node is relevant to the explanation if any of its children is relevant. Analogically, a node is failed if any of its children is failed (the only exception is an Or node which is failed if *all* of its children are failed). Similarity in these features gives the opportunity to perform these markings in a single post-order traversal. Algorithm 2 starts with marking all leaf nodes from the given list of failed nodes F as being failed, and marking all leaf nodes from given list $trace_Q$ as being relevant (where $trace_Q$ is the part of the trace before and including the question Q). For every non-leaf node we check the markings on its children of the attributes `.failed` and `.relevant` and calculate the marking for the node.

This calculation, which is done before calculating the explanation, is one way in which the algorithm is more efficient than the Haskell-transliterated definitions. The formal definition has a recursive call that iterates over all nodes G_{iV} that satisfy $n(G_{iV}) \cap T^{\prec A} \neq \emptyset$. The Haskell transliteration of this definition yields an inefficient calculation method, since it has to evaluate this for each node, which involves evaluating $n(G_{iV})$ repeatedly. By contrast, Algorithm 2 marks all the nodes' relevance and failed flags in a single traversal of the tree.

In the second part, Algorithm 1 traverses the tree in a preorder from the root to leaves and, for each node which is relevant to the explanation, performs the next steps (lines 6-29), which can seen to correspond to the definitions in Fig. 3:

1. If the current node is an Action, then we add its pre-conditions to the explanation (lines 8-13). The exception is a case when (part of) the pre-condition of the current action is also (part of) the post-condition of some other action that happened before the current node. In this case the condition is not included in the explanation because it is not a causal factor, since it is ensured by an earlier action (and can be filtered out using the “slink” attribute). For example, if action *getCoffee(office)* is relevant to the explanation, the algorithm should add its two pre-conditions *at(office) & havePod*. However, since this node has an “slink” attribute with the *goto(office)* and *getPod* actions, then we should not include any pre-conditions that are part of the post-conditions of the linked actions. Therefore, in this particular case, the algorithm does not add any pre-conditions to the explanation.

Algorithm 1: Explanation algorithm.

```
   input  : $G$, $trace$, $Q$, $F$
   output: Explanation
1  Explanation = ∅ // Initialise empty explanation
2  if $Q \notin trace$ then return "Question does not make sense!"
3  else
4      $trace_Q$ = part of trace before (and including) Q
5      Marking relevant and failed nodes according to Algorithm 2
6      for $N$ in $PreOrder(G)$ do
7          if $N.relevant$ then
8              if $N.type = Act$ then
9                  $PR = N.pre$
10                 if $slink \neq \emptyset$ then
11                     for $A_i \in N.slink$ do
12                         $PR = PR - A_i.post$ // difference of two sets
13                 add P:$PR$ to Explanation if $PR \neq \emptyset$
14             if $N.type = Or$ then
15                 select $choice \in \{ch \mid ch \in N.children \wedge ch.relevant \wedge \neg ch.failed\}$ // find the choice made
16                 add C:$choice.condition$ to Explanation
17                 for $child \in N.children$: $child \neq choice$ do
18                     if $child.failed$ then add T:"Tried child" to Explanation
19                     else
20                         if ($child.condition \subseteq child.bel$) // condition is true then
21                             add V:"$choice : child.annot < choice.annot$" to Explanation
22                         else
23                             for $C_j \in child.condition : C_j \notin child.bel$ // find false parts of condition do
24                                 add C:$\neg C_j$ to Explanation
25                     delete $child$ from $N.children$ // prune current branch of N
26             if $N.type = And \vee N.type = Seq$ then
27                 for $child \in N.children$ do
28                     if $\neg child.relevant$ then
29                         delete $child$ from $N.children$ // prune irrelevant branch of N
30                     else if $child.Failed$ then
31                         delete $child$ from $N.children$ // prune failed branch of N
32     $N_s = \{Q\}$ // 3rd part, add links
33     while $N_s \neq \emptyset$ do
34         select $N$ from $N_s$
35         forall $N_1 \in N.link$ do
36             add L:"$Q \rightarrow N_1$" to Explanation
37             add $N_1$ to $N_s$
38         $N_s = N_s - \{N\}$ //delete $N$ from $N_s$
39     $N = Q$ // 4th part, add motivation
40     repeat
41         $N = N.parent$
42         if $N.type \neq Or$ then add D:"Desire of $N$" to Explanation
43     until $N = root$
44     return Explanation
```

2. If the current node has Or type, then we add to the explanation why the particular choice was made. The choice that was made is the child of the Or node that is relevant and not failed (line 15). The explanation provided for why this choice was made has four parts (following the definition of *pref*):
   (a) the condition of the selected choice being true, i.e. C_i (line 16)
   (b) tried but failed options (line 18)
   (c) the conditions of not selected choices which are false, i.e. $\bigcup_{C_j : C_j \notin bel(N_j)} \neg C_j$ (line 24), specifically, for each option that could not be selected because its condition C_j is false, we select those parts of C_j that are false (recall that C_j is a set of propositions, representing a conjunction). We do not use C_j itself, because C_j may include some factors that in fact were true.

Algorithm 2: Marking relevant and failed nodes.

```
1  for $N$ in $PostOrder(G)$ do
2      $N.relevant = False$
3      $N.failed = False$
4      if $N.isleaf$ then
5          if $N \in trace_Q$ then
6              $N.relevant = True$
7          if $N \in F$ then
8              $N.failed = True$
9      else
10         if $\exists child \in N.children$: $child.relevant$ then
11             N.relevant = True
12         if $(N.type = And \vee N.type = Seq) \wedge \exists child \in N.children$: $child.failed$ then
13             $N.failed = True$
14         if $N.type = Or \wedge \forall child \in N.children$: $child.failed$ then
15             $N.failed = True$
```

(d) for every other possible option which has a true condition, the annotation of this choice which is less preferred to the annotation of the selected choice, i.e.[17] $V_j < V_i$ (line 21).

It should be noted that the part of the algorithm that works with one Or node (lines 15 - 25) can be implemented in one loop over all children. This can be done by storing a list of the "V" explanations to be added, and once the loop is complete, and we know *choice*, we loop over the stored list, adding the explanations relating to each child, and the *choice*.

During the calculations above, the algorithm modifies the tree cutting off branches of Or nodes that are not relevant to the explanation (line 25). It also prunes failed branches which were tried but didn't achieve a desired sub-goal. This can not affect calculating the explanation because the explanation has already taken all the information it needs from the pruned sub-trees. Moreover, irrelevant and failed nodes are not part of the explanation. It is worth mentioning that modification of the tree decreases the running time of the algorithm because it decreases the number of considered nodes in the tree.

The part of the explanation above that corresponds to the "pref" function (defined earlier) can be divided into three groups:

- the group that includes conditions (corresponds to (a) and (c)), indicated with a letter "C" in front.
- the group that contains comparison of values (corresponds to (d)), indicated with a letter "V" in front.
- the group that consist of tried but failed options (corresponds to (b)), indicated with a letter "T" in front.

3. If the current node has And or Seq type, then the algorithm modifies the tree by deleting irrelevant nodes (lines 28-29).

The third and fourth parts of the algorithm add to the explanation preparatory links for the question Q, and desires.

If the question Q has a link to another node N_2, then we want to add to the explanation that the action was done in order to accomplish node N_2: $Q \to N_2$. Furthermore, if linked node N_2 has a link to another node N_3, then the algorithm adds the link $Q \to N_3$ to our explanation too. This part of the explanation consisting of links is indicated with a letter "L" in front. The calculation of these links (lines 32-38) also takes into account that a node can have more than one link, e.g. instead of Q linking to N_2, it could link to both N_2 and N_4, and both links need to be followed and included. This is handled by managing a set N_s of nodes that need to be processed. This is initialised to the singleton set containing just the question Q. Then an element is removed from N_s, and for each link that it has, a link explanatory factor is generated, and the target node is added to the queue N_s. This process is repeated until there are no more nodes in N_s.

The fourth part of the algorithm adds a motivation to the explanation, i.e. why the current course of action was being pursued. The algorithm moves up from the question Q to the root and for every parent which is not of Or type, adds a desire of accomplishing it (lines 39-43). This part of the explanation is indicated with a letter "D" in front.

4.3. Translating explanations to English

Having calculated an explanation, we can translate into English words why the action Q was executed. The most complicated part of the translation is related to Or nodes (Algorithm 4), i.e. why the made choice was preferred to other possible options. For example, if we know that one of the factors is the most important, e.g. coffee quality, and we preferred *getShopCoffee* $[3, 1, 1]$ to *getKitchenCoffee* $[1, 3, 3]$, then we can say: "The made choice (getShopCoffee) has the best quality of coffee and that is the most important attribute". If we know that the most important attribute can be overridden by other factors, then for example when we preferred *getKitchenCoffee* $[1, 3, 3]$ to *getShopCoffee* $[3, 1, 1]$, we can say: "I chose getKitchenCoffee with poorer quality of coffee despite quality of coffee being my most important attribute, because it was cheaper and closer than the other option (getShopCoffee)".

[17] The choice node is also stored, to allow us to distinguish between different places where choices are made.

Before proceeding further we need to explain the additional data structure that is used to interpret value effect annotations. This data structure is domain-specific, so needs to be defined alongside the goal-plan tree for a given system. It defines:

- A mapping that specifies the order in which value effects are indicated, and their associated names. For example, for the coffee scenario we specify that the annotations are given in order: (i) quality of coffee, (ii) price, and (iii) distance to the coffee.
- A mapping (I) that specifies the weighted importance of each attribute, for example we might specify $[5, 1, 1]$ to indicate that the first attribute (coffee quality) is the most important (5) and the remaining two attributes are less important, but equal in importance to each other.
- An English dictionary (*english*) that provides for each attribute the appropriate words to use to describe the attribute, and to describe the relationship between values. For example, for the first attribute we specify that the attribute is "quality", and that the highest value is termed "the best", with relative value preferences being described using the words[18] "better" and "poorer". By contrast, for the second attribute we specify that it is called "price", with highest value termed "lowest", and relative preferences using the terms "a lower" and "a higher". Finally, the third attribute is termed "distance", with the highest value being "the smallest", and relative preferences being described by terms "a smaller" and "a larger".
- Finally, an English mapping (*engmap*) that provides mappings for the conditions that occur in a particular domain, as well as some generic English text mappings.

An example mapping for the coffee example might be:

$$\begin{aligned}
\mathit{name} &= \langle\text{“quality of coffee”}, \text{“price”}, \text{“distance to the coffee”}\rangle \\
I &= \langle 5, 1, 1\rangle \\
\mathit{english} &= \langle(\text{“the best”}, \text{“better”}, \text{“poorer”}), (\text{“lowest”}, \text{“a lower”}, \text{“a higher”}), \\
&\qquad (\text{“the smallest”}, \text{“a smaller”}, \text{“a larger”})\rangle \\
\mathit{engmap} &= \{(\text{“have”} \mapsto \text{“had ”}), (\text{“Available”} \mapsto \text{“ was Available”}), \\
&\qquad (\text{“AnnInOffice”} \mapsto \text{“Ann was in the office”}), \\
&\qquad (\text{“own”} \mapsto \text{“owned ”}), (\text{“not not”} \mapsto \text{“”})\}
\end{aligned}$$

The presentation of Algorithms 3 and 4 uses a number of notational conveniences. Firstly, it uses brackets within strings to indicate interpolation of values. For example, if the variable v has value "best" then the string "this string has the (v) coffee" evaluates to "this string has the best coffee". Secondly, we use "++" to denote string concatenation, and a C-like assignment operator "v ++= e" as shorthand for "v = v ++ e". Finally, we define the function e which takes a set of items $\{e_1, e_2, \dots, e_{n-1}, e_n\}$, and maps them to English of the form "e_1, e_2, …, e_{n-1} and e_n".

The basic part of the translating algorithm is presented in Algorithm 3. This algorithm takes as input a set of explanatory factors, and generates an English rendition of the explanation (a list of English sentences). It performs a straightforward mapping: for each type of factor, it maps the explanatory factor to English text. It then (line 27) applies the English mapping (*engmap*). The one part of the translation that is more complex is, as noted earlier, the handling of "V:" explanatory factors (lines 8-20). The generation collects into sequences V and N the value effects (and corresponding names) relating to a given node, i.e. given $V : n : a \to b$ it collects all other $V : n : a' \to b$ to form $V : \{a, a', \dots\} \preceq b$. Once the algorithm collects all relevant other options it invokes Algorithm 4 (the function *name_of* gives the name associated with a given value effect annotation).

Algorithm 4 attempts to provide a clear English mapping of the preferences in each case. It takes a number of parameters as input: the list of importance weights I, the name n_m and annotations v_m (list of numbers) of the made choice, the names $N[i]$ and annotations $V[i]$ (list of numbers) of the other options, and the names of the attributes $\mathit{name}[j]$. Algorithm 4 operates as follows. Firstly, in line 3 it creates the start of the explanation, which is the name of the selected option n_m. It then computes the most important factors (MI). This is done because if there is a single most-important factor (say, coffee quality), then we can provide a simple explanation in terms of that factor (e.g. "this option has the best quality coffee"). This explanation is complete, in that it does not lose information. Rather, the algorithm recognises that in this case, where there is a single most-important factor, it is possible to provide a compact explanation.

Lines 6-15 deal with the case in which there is a single most-important factor. In this case MI is a singleton set, and j is the index of the most important factor. There are a number of sub-cases here. The first is when the selected option is better in terms of the most important factor than all other options (lines 8-9). In this case we simply explain that the selected option was better than all other options, and stop (e.g. "getShopCoffee has the best quality of coffee (and that is the most

[18] Although the algorithm only uses the first of these.

Algorithm 3: Translating an explanation into English.

```
    input  : $X$ (set of explanatory factors)
    output: Explanation in words (explain)
1   ee = ⟨⟩ // initialise empty explanation (sequence)
2   for $P : c, C : c \in X$ do
3       if c begins with "have" or "own" then
4           ee ++= ⟨ "I " ++ c ⟩
5       else
6           ee ++= ⟨ c ⟩
7   $X_V = \{V : n : v_m < v_o \mid x \in X \wedge x = V : n : v_m < v_o\}$ // collect V: explanatory factors
8   while $len(X\_V) > 0$ do
9       remove $V : n : v_o < v_m$ from $X_V$
10      $n_m = name\_of(v_m)$
11      $V = [v_o]$
12      $N = [name\_of(v_o)]$
13      // Collect into V and N ...
14      $T_V = copy(X_V)$
15      for $V : n' : v_i < v \in T_V$ do
16          if $n = n' \wedge v = v_m$ then
17              remove $V : n' : v_i < v$ from $X_V$
18              add $v_i$ to $V$
19              add $name\_of(v_i)$ to $N$
20      ee ++= ⟨ invoke Algorithm 4($I, name\_of(v_m), v_m, N, V, name$) ⟩
21  for $T : n \in X$ do
22      ee ++= ⟨ "I tried ($n$) but I failed" ⟩
23  for $L : a \to b \in X$ do
24      ee ++= ⟨ "I needed to ($a$) in order to ($b$)" ⟩
25  for $D : n \in X$ do
26      ee ++= ⟨ "I desired to ($n$)" ⟩
27  Return find_and_replace(ee,engmap)
```

important attribute)"). Again, the explanation generated does not lose information, the algorithm simply recognises that a compressed explanation in this situation is complete.

If there is a single most-important factor, but the selected option is not better than the other options in terms of the most important factor, then it is possible that the selected option is better (in terms of the most important factor) than some, but not all, other options. If this is the case (line 10), then we construct a partial explanation that explains that the selected option was better than certain options (line 11), and we then go on to add on to it the exception cases (lines 12-15), yielding a final explanation of the form (e.g.) "getShopCoffee has better quality of coffee than getKitchenCoffee, and is preferred to getOfficeCoffee despite having worse or equal quality of coffee (which is the most important attribute) because it has better distance than getOfficeCoffee".

We begin generating an explanation of the overridden cases by creating a set ($overrode$) of other options which have as good or better quality on the most important factor. These are *overridden*, in the sense that they were not selected, despite being better (or as good) on the most important attribute, because improved performance on other factors was sufficient to override the most important factor. For example, in certain situation one might select an option that did not yield the best quality coffee (most important attribute) because it was both cheaper and closer. Or because a small difference in quality was overridden by a large difference in cost or in distance.

We begin to construct the explanation by adding to the explanation that the selected option was preferred to those overridden options despite having worse (or equal) performance on the most important factor because it has better performance on other factors. The last part (better performance on other factors) is added by lines 20-35 (which allows the same code to be also used to construct the explanation for the case where there is not a single most important factor).

Lines 20-35 create a list of explanations for why overridden options were overridden. It attempts to make the explanation more readable by *grouping* related options. If two options are worse than the selected option on the same factors then we can group them together and say (e.g.) "[the selected option] has better (*factor1*, *factor2*, and *factor3*) than (*option1* and *option2*)". This grouping is done by computing for each option i the things which the selected option is better at ($better[i]$). We then repeatedly select an option i, and retrieve all other options i' that should be grouped with it, i.e. where $better[i] = better[i']$. The collected set of options (tmp) is then processed in one step (line 34). Note that the variable *join* is used to add "and" at the start of each explanation fragment apart from the first.

Finally, in the case where there is not a single most important factor (lines 17-19) we simply note that the selected explanation is preferred because it has better performance than the other options on various factors. We then go on (using the same code discussed in the previous paragraph) to construct this list of "it has better (*factors*) than (*options*)".

In all cases, all available information is provided. In some cases, a simple yet complete explanation can be provided (e.g. "X is preferred because it provides the best V (which is the most important factor)"). In other cases, a more complex explanation is required to convey a complete explanation (e.g. "X is preferred because it provides better V than option Y, and is preferred to option Z despite having worse V because it has better V' and V''"). In other cases, we cannot do better than (e.g.) "X is preferred because it has better (factor set 1) than (option set 1), and better (factor set 2) than (option set 2)".

Algorithm 4: Translating into English words why a particular choice was made.

```
input  : I (importance), $n_m$ (name of selected option), $v_m$ (value annotation of selected option), N (vector of names of other
         options), V (vector of value annotations of other options), name (from mapping)
output: Explanation in words (explain)
1  // Note: $j$ ranges over attributes (e.g. 1-3 for coffee quality, price, distance)
2  // $i$ ranges over the other options, so use $V[i]$ and $N[i]$.
3  explain = "($n_m$) "
4  MI = $\{j \mid \forall j' : j \neq j' \Rightarrow I[j] > I[j']\}$ // most important factor(s)
5  if $|MI| = 1$ then
6      // there is a single most important factor
7      select $j$ from MI // note: singleton set
8      if $\forall i : v_m[j] > V[i][j]$ // selected option is best w.r.t. most important factor then
9          return explain ++ "has ($english[j][1]$) ($name[j]$)"
10     if $\exists i : v_m[j] > V[i][j]$ then
11         explain ++= "has ($english[j][2]$) ($name[j]$) than ($e(\{N[i] \mid v_m[j] > V[i][j]\})$), and "
12     // collect cases that are overridden
13     overrode = $\{i \mid v_m[j] \leq V[i][j]\}$
14     // explain the cases where we overrode the most important attributes
15     explain ++= " is preferred to ($e(\{N[i] \mid i \in overrode\})$) despite having worse or equal ($name[j]$) (which is the most important
         attribute) because it has "
16 else
17     // more than one most important attribute
18     explain ++= " is preferred because it has "
19     overrode = $\{1, \ldots, |V|\}$ // need to handle all other options in this case
20 // find the things that a given option $i$ does worse on – applies in both cases
21 for $i \in overrode$ do
22     better[i] = $\{j \mid v_m[j] > V[i][j]\}$
23 // explain in groups: for a set of options that have the same set of attributes where the selected option does better, explain them
     together
24 join = "" // only add "and" after the first factor
25 while $overrode \neq \emptyset$ do
26     $tmp = \emptyset$
27     select $i \in overrode$
28     $overrode = overrode \setminus \{i\}$
29     $tmp = tmp \cup \{i\}$
30     for $i' \in overrode$ do
31         if $better[i] = better[i']$ then
32             $overrode = overrode \setminus \{i'\}$
33             $tmp = tmp \cup \{i'\}$
34     explain ++= " ($join$) better ($e(\{name[j] \mid j \in better[i]\})$) than ($e(\{N[i] \mid i \in tmp\})$) "
35     join = " and "
36 return explain
```

We now give an example to illustrate the English generation done by the two algorithms. Consider a scenario where the selected option is to go to the shop, and where the starting location was close to the shop (so there was a low distance to the shop, and a higher distance to the office and the kitchen). We represent this with the value effect annotations: $[3, 2, 3]$ for the (selected) shop option, and respectively $[1, 3, 1]$ and $[2, 3, 1]$ for the kitchen and office options. Recall that higher numbers indicate more preferred characteristics (e.g. higher quality coffee, lower price, smaller distance), and that the order of attributes is quality, then cost, then distance. So, for instance $[3, 2, 3]$ represents very good coffee, at a medium price, and low distance.

Given this scenario, let us first assume that there is a single most-important attribute, namely the quality of the coffee. In this case the full generated English explanation is:

"I had Money; *getShopCoffee has the best quality of coffee (and that is the most important attribute)*; I needed to pay shop in order to getCoffee shop; I desired to getShopCoffee"

On the other hand, if we assume that instead the quality and price are both equally important (but that distance is less important), then the italic sentence above would instead be replaced by:

"getShopCoffee is preferred because it has better quality and distance than getOfficeCoffee and getKitchenCoffee"

In a (hypothetical) situation where pod coffee was seen as being of equal quality to shop coffee (represented by changing the value annotation on office coffee to $[3, 3, 1]$), then, if we assume again that quality is the most important factor, the italic sentence would be:

> "getShopCoffee has better quality of coffee than getKitchenCoffee, and is preferred to getOfficeCoffee despite having worse or equal quality of coffee (which is the most important attribute) because it has better distance than getOfficeCoffee"

This is an example of a case where one option is covered by line 11, and another option is overridden.

Finally, in the hypothetical situation where pod coffee is as good as shop coffee, but where quality and price are equally important, the italic sentence would be:

> "getShopCoffee is preferred because it has better quality and distance than getKitchenCoffee and better distance than getOfficeCoffee"

4.4. Algorithm computational complexity

We observe that the explanation algorithm has four components: preparatory marking of the nodes (Algorithm 2), the reasons calculated by the function $E_N^T(G)$ (lines 6-31 of Algorithm 1), the links between nodes (lines 32-38) and the motivations (lines 39-43).

The last is simple to compute (lines 39-43), involving merely traversing the tree upwards from the node being queried (i.e. $O(\log N)$ where N is the number of nodes in the goal tree[19]).

The third, the links, depend *only* on the static structure of the tree (i.e. which nodes precede other nodes), and on the pre- and post-conditions, and therefore can be computed ahead of time. Furthermore, the length of the chain of links can not be more than the number of all action nodes in the tree ($n(Act)$) which is less than N. If we consider the worst case, when every action node has links to all other possible actions in the tree, then the number of links is proportional to[20] $|n(Act)|^2$ which is overall $O(N^2)$. However, for real trees the number of the links is small and, more importantly, usually scales with (at most) the number of nodes, since it is unusual for a node to have a link to many other nodes, since the first node needs to both precede the other nodes, and have an overlap between its post-condition and their pre-condition.

During preparatory marking of failed and relevant nodes (Algorithm 2) we check every node in the tree. For the leaf nodes calculations are proportional to the size of list F of failed nodes and $trace_Q$, i.e. marking *all* leaf nodes (assuming indexing) is $O((|F| + |trace_Q|) \times n(Act))$ where $n(Act)$ is the number of all leaf nodes (actions) in the tree. Since $|F| < |trace_Q|$, i.e. the number of failed nodes can not be more than the number of all executed nodes, this complexity is $O(|trace_Q| \times n(Act))$. If $trace_Q$ is implemented as a hash-table, then the average time for looking up one particular action in $trace_Q$ is $O(1)$. In this case, the complexity of the algorithm's part that works with leaf nodes is $O(n(Act))$ which is overall $O(N)$, since $n(Act) < N$.

For every non-leaf node calculations are proportional to the number of children where for every child we perform $O(1)$ operations. However, during this marking we access every node no more than two times - the first time when we work with the (non-leaf) node and second time when we work with its parent. This means that the number of operations that are performed during marking all non-leaf nodes grows as fast as N.

Finally, we turn to the main part of the algorithm (corresponding to E_A^T) which is lines 6-29 of Algorithm 1. This computation traverses the tree from the root to leaves, pruning at the same time irrelevant branches. Consequently, the number of considered nodes is equal to the number of nodes in the tree marked as relevant: $|relevant|$. For each considered And or Seq node we do calculations that are proportional to the number of children. For each considered Act node we do calculations that are proportional to the size of the pre-conditions of the current node and some previous nodes if attribute "slink" is not empty. For each considered Or node we do calculations that are proportional to the size of the conditions and beliefs and the number of children. It is worth mentioning that the second part of the algorithm accesses every node in the tree at most twice. If we assume that all sizes mentioned above (size of pre-conditions, conditions, beliefs) are effectively constants and do not grow with N, we can conclude that calculating this part of the explanation is restricted by $O(N)$. This is based on the (additional) assumption that the size of a node's "slink" attribute does not grow as the tree grows, since links (i.e. dependencies where one action node's post-conditions overlap with a subsequent action node's pre-conditions) are expected to be local. If this assumption fails to hold, then in the worse case the overall complexity is $O(N^2)$. However, in practice we would expect that even if the size of a node's "slink" list does grow with the tree size, the growth is considerably less than linear in the size of the tree, so overall complexity would be close to $O(N)$.

Therefore, total complexity of the complete explanation algorithm is $O(N)$.

[19] This is assuming that $j > 1$ and $k > 1$, i.e. that we have a tree, rather than a "stick".

[20] We use $|F|$ to denote the size of set (or list) F.

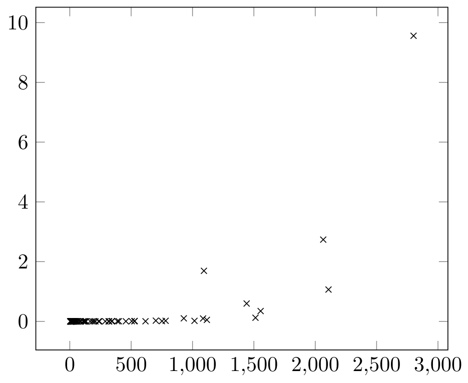


Fig. 6. Time in seconds (Y) vs. number of nodes (X) for Haskell implementation.

5. Evaluation: efficiency

There are two broad questions that concern evaluation of this work. The first (in this section) is whether the approach is sufficiently *efficient*. The second (in the next section) is whether the explanations provided are comprehensible and *useful* to a human user.

In order to empirically assess the actual runtime required, and the algorithm's scalability, we have conducted an experimental evaluation on generated trees. The generated trees have the following structure:

$$T^0 = A$$
$$T^{d+1} = \textsc{Or}(O_{1-j})$$
$$\text{where } O_i = (c, \textsc{Seq}(T^d_{1-k}))$$

In other words, a generated tree of depth 0, denoted T^0, is just an action A (with a new unique name), and a generated tree of depth $d+1$ is a disjunction of j options, where each option O_i has the same fixed condition c, and a sequential composition of k trees of depth d. All nodes have unique names. Note that the number of nodes in a tree with branching factors j and k and depth d can be calculated as: $n(j,k,0) = 1$ and $n(j,k,(d+1)) = 1 + j + (j \times k \times n(j,k,d))$, which can be written in non-recursive form (for $d > 0$) as:

$$n(j,k,d) \ = \ (jk)^d + (1+j) \times \sum_{q=0}^{d-1} (jk)^q$$

Our efficiency evaluation considered the formal definitions (Section 3) implemented by transliterating them into Haskell, and the Python implementation of the algorithm (Section 4).

For the Haskell evaluation various values of j, k and d were systematically generated, and the number of nodes in the tree and the time taken to compute E^T_N were recorded. The experiments were done using the GHC Haskell implementation (version 8.2.1) running on a 3.2 GHz Intel Core i5 iMac with 16 GB RAM running OSX 10.10.3.

The results are in Fig. 6, which shows a scatter plot[21] of runtime (Y axis, in seconds) against the number of nodes in the tree (X axis). As can be seen, for relatively small trees (fewer than 1000 nodes) the explanation generation, even with an unoptimised Haskell prototype, is clearly fast enough to be practical (Fig. 7 shows the timings, excluding trees with more than 1000 nodes). It is worth emphasising that the core of the Haskell implementation is a direct transliteration of the formal definitions given in Section 3. While this ensures that the implementation matches the paper, there are clear, and substantial, opportunities to improve efficiency. It is also worth noting that real goal trees are not necessarily large. For instance, the (real-world) application described by Burmeister et al. [42] has 57 nodes in its goal tree.[22]

For evaluating the Python implementation of the algorithm 179 trees were generated (in Haskell) with various values of j, k and d. For each tree the explanation program was run 30 times, and the time taken recorded. We then removed outliers before computing an average time. Outliers were those timings that were 2 or more standard deviations away from the mean. The experiments were done using the Python 3.6 32 bit implementation running on a 3.30 GHz Intel Core i5-4590 with 8 GB RAM running Windows 10.

[21] The time taken is affected not just by the number of nodes, but also by the shape of the tree, so there can be a number of trees with the same number of nodes and different time taken to compute the explanation.

[22] Burmeister et al. do not provide the full details of their goal-plan tree (it's from an industrial application). It is used simply to gauge how large real-world goal-plan trees are likely to be, and to justify the selection of tree size for our evaluation.

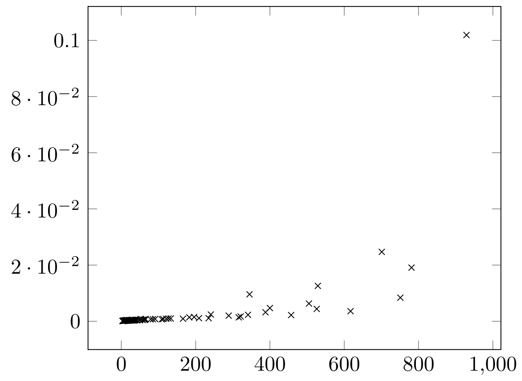


Fig. 7. Time in seconds (Y) vs. number of nodes (X) for Haskell implementation excluding trees with > 1000 nodes.

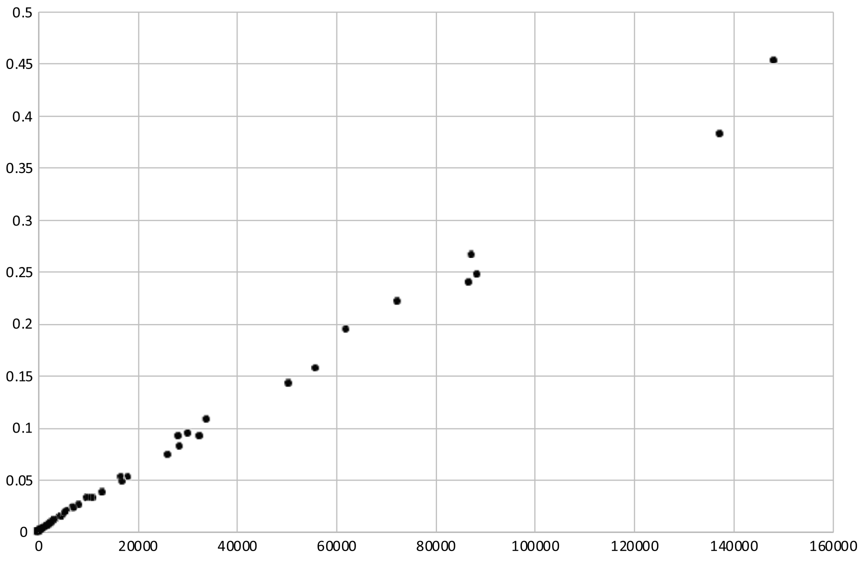


Fig. 8. Running time in seconds (Y) vs. number of nodes (X) for Python implementation.

The results are shown in Fig. 8. These show that the algorithms presented in Section 4 are indeed $O(N)$, consistent with the analysis in Section 4.4. They also show that the time taken is low: with even the largest tree, with over 137,000 nodes, taking less than 0.4 s.

We also analysed the results in order to isolate the effects of each of the three parameters j, k, and d. For instance, for j we grouped together all trees with $k = 1$ and $d = 1$ and plotted a line showing the effects of increasing j while holding $k = d = 1$. This was repeated for each combination of values of k and d. Similarly, we plotted a set of lines showing the effects of varying k while holding j and d constant, and the effects of varying d while holding j and k constant.

These results are shown in Fig. 9. The top panel shows the groupings where, for each line, k and d have a fixed value and j is varied. The middle panel shows for each line varying k, and in the bottom panel each line corresponds to varying d, given particular fixed values for j and k. Each of the panels is split into three sections (left, middle, right). These sections split apart a small number of nodes, medium number, and a large number of nodes. This is done in order to allow the smaller number of nodes to be viewed, rather than occupying a small part of a larger graph.

In the top two panels the pattern is the same: there is some variance for smaller values of N, but with the larger values (right sides) there is a clear and consistent linear scale. For the bottom panel this pattern is also apparent for smaller N.

6. Evaluation: effectiveness

In order to assess the comprehensibility and usability of the explanations generated, as well as provide guidance to future work on selecting explanations, we conducted two human participant evaluations. Since the focus of this paper is on the explanation mechanism, this section briefly reports on the outcomes of these initial studies, giving enough background to understand the outcomes, but omitting methodological and statistical details. For the full details on these studies see [13] for the first study, and [43] for the second study.

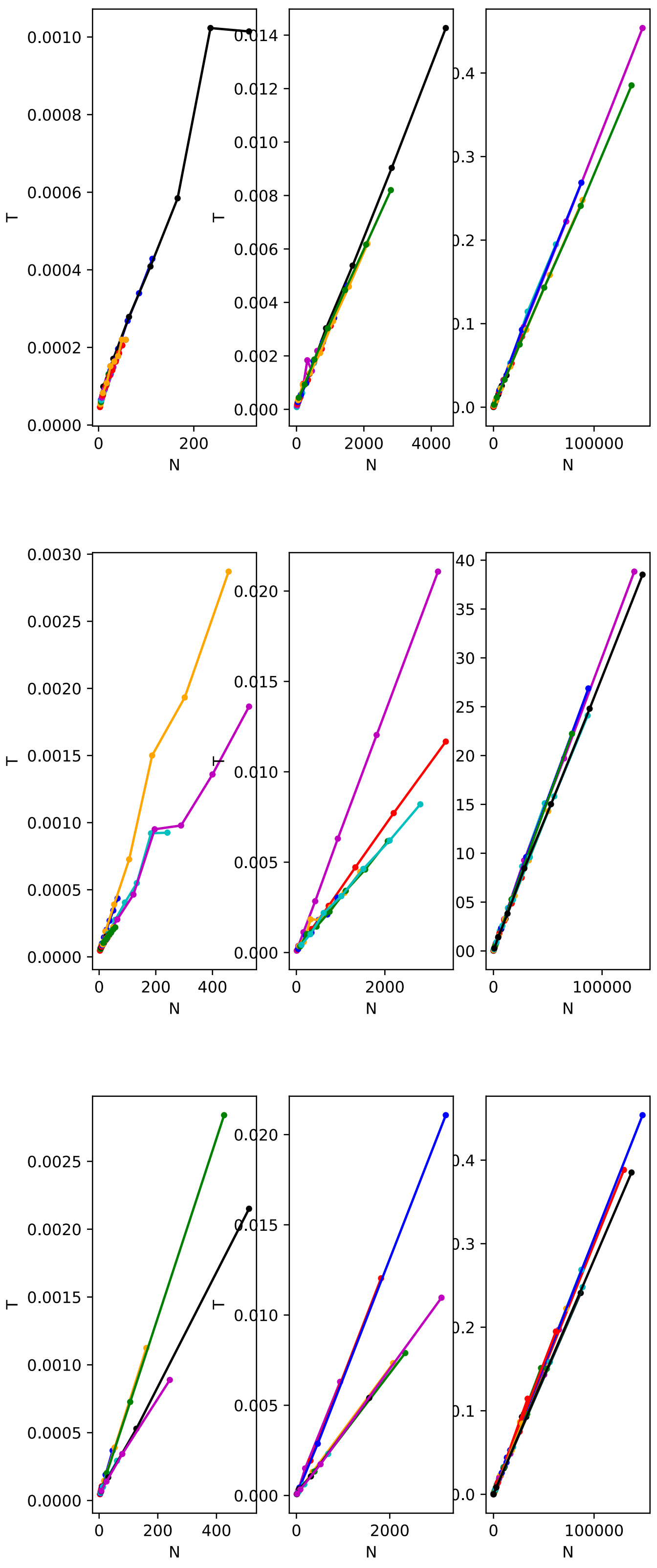


Fig. 9. Running time for varying j (top), k (middle) and d (bottom).

Table 1
Rankings for the three groups and five explanations.

Explanation	Average ranking and implied collective ranking					
	Group 1		Group 2		Group 3	
E1	2.7857	2	2.5135	1	2.0667	1
E2	3.5714	5	2.5676	2	3.7333	5
E3	2.5238	1	2.7297	3	2.5667	2
E4	3.1429	4	4.1622	5	3.1667	3
E5	2.9762	3	3.0270	4	3.4667	4
$p =$	0.011		0.0000007		0.000016	

Our first evaluation took the coffee scenario described and administered a survey. Participants, who were recruited on mechanical Turk, were provided with a brief description of the scenario and an indication of what behaviour was observed (participants were randomly allocated into one of three groups, each of which was given a different observed behaviour). Each of the 109 participants was given five possible explanations for the observed behaviour, and was asked[23] to rank the explanations from most to least preferred. The first explanation combined valuings and beliefs, and corresponds to the E_N^T function defined earlier (in Section 3.2, and indicated with "V+B" below). The second and third explanations are solely in terms of valuings: one is abstract (AV), just saying literally "*This is the best possible coffee available*", and the second is concrete (V), with a specific explanation (see below). The fourth candidate explanation provides only relevant beliefs (B). Finally, the fifth candidate explanation gives the goal, and the beliefs that enabled the specific behaviour that was selected, which is the explanation mechanism proposed by Harbers [15] (G+B).

For example, in the case where the colleague's machine was selected (Group 1), the five explanations given are:

(E1) "This is the best possible coffee available; I had no money." (V+B)
(E2) "This is the best possible coffee available." (AV)
(E3) "This coffee is better than the kitchen and cheaper than in the shop." (V)
(E4) "I've no money; Ann was in her room." (B)
(E5) "I wanted coffee; Ann was in her room." (G+B)

Table 1 shows for each explanation and for each group the *average ranking*, which is the average of each explanation's ranking. The table also shows for each explanation and group the preferred order of explanations implied by the average ranking (the implied collective ranking). For example, for group 1, explanation 3 had the lowest average ranking, and is therefore the most preferred explanation (differences between the explanations' scores for each of the groups is statistically significant with all p values < 0.05).

Overall, these results show that explanations 1 and 3 were considered as being better than the other explanations, and that, except for Group 2, explanation 2 was seen as being the worst.[24] Since explanations 1 and 3 both include valuings, this indicates that valuings are of value in providing effective explanations. Furthermore, for Groups 2 and 3, E1 was preferred to E3, indicating that valuings alone were not sufficient. For Group 1 this was not the case, which could be because for this particular observed behaviour, E1 is somewhat complex, containing two conditions.

The second evaluation also used a survey. Participants were recruited using advertisements in a range of undergraduate lectures within the Otago Business school, by email to students at co-authors' institutions, and by posting on social media.

The scenario used involved a software personal assistant called SAM (see top of Fig. 10). Each participant was presented with five possible explanations (Fig. 10), given in a random order, and was asked to rank the explanations from most to least preferred.[25] The explanations combine different elements of the explanation mechanism described earlier in this paper. Specifically, there are four types of elements that can be included in an explanation: beliefs, valuings, desires, and links. Explanation E1 includes all four elements, explanation E2 includes only valuings and beliefs, E3 includes only valuings, E4 includes only beliefs, and E5 includes only beliefs and desires.

To analyse the ranking of explanations we employed a general discrete choice model, using a ranked-ordered logit model which is also known as an exploded logit [44]. This allows us to analyse which factors contributed to the outcome of a made choice. It is required in this case because each of the five explanations being ranked represented a combination of explanatory factor types. The ranked-ordered logit is used to deal with the fact that the data represents a ranking: after selecting the most preferred explanation, the next selection is made out of the remaining four explanations. This means that the selections are not independent.

Results are shown in Table 2. Each row (e.g. row E2) is in relation to the reference explanation, E1. The column β gives the key parameter, showing the relative likelihood. These estimates indicate that, on average, respondents are most

[23] They were also asked other questions, e.g. about the believability of the explanations, whether they felt that further explanation was required, and also demographic questions.
[24] For Group 2 the behaviour observed was getting coffee from the shop, which meant that E2 was in fact a good explanation.
[25] As in evaluation 1, there were also other questions.

Imagine that you have a smart phone with a new smart software assistant, SAM. Unlike current generations of assistants, this one is able to act proactively and autonomously to support you. One particular afternoon, you are about to leave to go home, when the phone alerts you that SAM has just bought you a ticket to catch the bus home. This surprises you, since you typically walk or cycle home. SAM knows that usually you use one of the following three options to get home: (1) Walking; (2) Cycling, if a bicycle is available; and (3) Catching a bus, if money is available (i.e. there is enough credit on your card). You push the “please explain” button. The next screens give five possible explanations (presented in random order).

E1: A bicycle was not available, money was available, the made choice (catch bus) has the shortest duration to get home (in comparison with walking) and I believe that is the most important factor for you, I needed to buy a bus ticket in order to allow you to go by bus, and I have the goal to allow you to catch the bus. **(beliefs, valuings, desires, links)**
E2: A bicycle was not available, money was available, and the made choice (catch bus) has the shortest duration to get home (in comparison with walking) and I believe that is the most important factor for you. **(beliefs, valuings)**
E3: The made choice (catch bus) has the shortest duration to get home (in comparison with walking) and I believe that is the most important factor for you. **(valuings)**
E4: A bicycle was not available, and money was available. **(beliefs)**
E5: A bicycle was not available, money was available, and I have the goal to allow you to catch the bus. **(beliefs, desires)**

Fig. 10. SAM Scenario and explanations E1-E5.

Table 2
Respondent’s preferences in ranking explanations.

Analysis of maximum likelihood estimates					
Explanation	DF	β	Standard error	Chi-square	Pr > ChiSq
E2	1	0.475	0.166	8.18	0.0042
E3	1	−0.154	0.165	0.878	0.3488
E4	1	−1.077	0.17	40.016	<.001
E5	1	−0.887	0.168	28.034	<.0001

Table 3
Respondent’s preferences in ranking components.

Analysis of maximum likelihood estimates					
Parameter	DF	Parameter estimate (β)	Standard error	Chi-square	Pr > ChiSq
Valuings	1	2.402	0.224	115.28	<.0001
Beliefs	1	0.821	0.176	21.661	0.0001
Desires	1	0.543	0.224	5.88	0.0153
Links	1	−1.164	0.285	16.6224	0.0001

likely to prefer explanation E2 ($\beta_{E2} = 0.475$) and least likely to prefer E4 ($\beta_{E4} = -1.077$). The odds of preferring E2 are $exp^{0.475} = 1.608$ times the odds of preferring E1. We then investigated the effects of explanation components (beliefs, desires, valuings, links) and how they affect ranking. Our analysis (see Table 3) found that respondents prefer explanations which have valuing, belief, and desire components. They are reluctant to prefer explanations that have links. Furthermore, this analysis shows that of the four factors, the presence of valuing components most strongly (and significantly) correlates with higher preference for the explanation. In other words, explanations including valuings are more likely to be preferred. These results are consistent with, and reinforce, the findings from the first evaluation.

7. Discussion

We have argued that explaining the behaviour of autonomous software could be done using the same concepts as are used by humans when explaining their behaviour. Specifically, we have followed the findings of Malle that “*Among the mental states that function as reasons, beliefs and desires are most common, and there is a third class that we might call* valuings” [6, Section 4.2.4].

This paper has proposed a formal framework, using BDI-style goal-trees, augmented with value effect annotations. This formal framework is then used to define an explanation function, which has been implemented. We have assumed that explanations are *honest*, in that they attempt to provide an accurate representation of the reasons for the observed behaviour.

An alternative, which we eschew, is to consider *dishonest* explanations that seek to present the agent in the best possible light.

That our explanations are couched in terms of familiar concepts suggests that the explanations are likely to be comprehensible. This is seen in the presented example explanations, which are not excessively long, nor excessively complex. More importantly, this claim is also supported by the evaluation results, including the finding in both evaluations that valuings are seen as important components of good explanations.

However, there is always scope for future work. One area for future work is conducting further empirical evaluation. This would include using more scenarios, including the other explanatory factors, and assessing not just the believability, acceptability and comprehensibility of explanations, but more broadly assessing their effect on trust in the autonomous system.

Turning to the explanation mechanism itself, there are three broad areas for future work. Firstly, extending to handle other types of questions. This paper has provided a mechanism for answering questions of the form "why did you do X?". However, an answer to a question of this form may include explanatory elements that prompt further questions. For instance, in explaining why the agent chose to get shop coffee rather than pod coffee it might indicate that an explanatory factor is that it believed Ann to be out of her office. This might prompt the follow-up question: "but why did you believe that Ann was not in her office?". Other question types that we would like to be able to answer are: "why did you do X ... rather than Y?" (contrastive questions, as noted in the introduction, and discussed below); "why did you fail at doing X?" (which can arise when a plan unexpectedly fails); and "why did you *not* do X?" (which can be useful in understanding differences between the beliefs of the agent, and the human explainee). A possible starting point is Winikoff [45], which defines these question types, and provides mechanisms for answering them (although in a debugging context where the explainee is a programmer who is familiar with the code, and hence the explanations can, and should, refer to code). Finally, it might be desirable to consider extending explanation to include not just reasons, but also enabling factors and Causal History of Reason explanations, which would involve also adding more types of questions such as "how was X possible?" [46].

A second area for future work that concerns the explanation mechanism is adding a "social layer" that deals with the potential differences between the agent's beliefs and the human explainee's beliefs. This relates to Miller's finding that human explanations are social, i.e. presented relative to what the explainer believes the listener knows [20]. It is, of course, possible for the agent and human's beliefs to differ, which is a potential source of misunderstanding that requires explanation to clarify. For example, if the human believes that Ann is in her office and the software agent does not, then helping the human to understand why the agent did not obtain pod coffee requires this difference in belief to be identified and resolved. This can be done from the human's end: if the system provides an explanation that includes "Ann was not in her office", then the human could update their beliefs, or query why the system believed this. Alternatively, the system could attempt to build up an approximate model of what the human believes, and use this in providing explanations.

The third area for future work relating to the explanation mechanism is filtering. As noted in the introduction, one of the key findings highlighted by Miller [20] is that human explanations are selective, they do not include all factors, but select and present only factors deemed relevant. Therefore, one next step in this research is to define means for *selecting* parts of the possible explanation. This would need to take into account what information is available about the human asking for the explanation, for instance, when a contrastive question of the form "why did you do X rather than Y?" is asked, we can interpret Y as the expected behaviour, and look for factors that specifically explain the difference between the implied expected behaviour of Y and the actual behaviour of X [47]. There is existing research that can be used to guide the identifying of the factors likely to be the most useful to include in an explanation. For example, McClure & Hilton [48] found that the preference between goal (desires in our terminology) and condition explanation depended on whether the course of action was "obfuscated", i.e. whether it was expected to be hard to do.[26] Filtering to provide only the most relevant factors can also be done using the social layer discussed in the previous paragraph: knowing what the user is expecting, and what they believe, can highlight the most relevant explanatory factors. For example, a condition where the agent and human have different beliefs. Another approach to filtering is to make the explanation process interactive: provide an initial (partial) explanation, and provide the ability for the human to pose follow-up questions.

In addition to these three areas for future work, which concern extending the explanation mechanism, there are also two areas for work that concern extending the decision-making process used by the system which is being explained. Firstly, there is scope to extend the representation (goal-plan trees) in various ways, and to consider related notations such as Behaviour Trees [49], Hierarchical Task Networks [50] (which extend Hierarchical Goal Networks [51]), or Geib & Goldman's Plan Trees [52]. One particular area of extension to consider is how the agent deals with failure. The BDI model provides one particular mechanism for failure handling (trying alternative plans), but other options are also possible, and have been investigated in the literature, for instance resorting to look-ahead planning based on action descriptions [53]. Of course, changing or extending the decision-making process used by the system also requires the explanation mechanism to be suitably modified.

Secondly, there is the challenge in explaining the behaviour of systems that do *not* make decisions using BDI plans (or similar). The focus of this paper is on explaining the behaviour of BDI agents. However, autonomous systems are realised using a wide range of techniques, not just BDI-style goal trees. As mentioned in the introduction, the approach proposed

[26] They also found that the preference was affected by the way in which the question was phrased: "why" vs. "explain".

in this paper can be used with non-BDI agents, by separating explanation from action selection. In other words, having one module that explains behaviour (using the approach in this paper), and having a separate module that the agent uses to select its actions. This requires the explanation module to have a goal tree that describes how the agent might have selected its behaviour. This approach is desirable, in that it provides explanations in terms of concepts that are familiar (and desirable). However, it runs the risk that the goal tree is inconsistent with the agent's behaviour. This therefore raises a number of questions for future work. Firstly, is it possible to effectively learn goal trees from a history of behaviour traces? Secondly, given a goal tree and traces of behaviour, is it possible to effectively detect cases where the goal tree is inconsistent with the behaviours and propose changes to the goal tree that improve (or even fix) the inconsistencies? There is a range of work on learning in BDI systems (e.g. [54–58]) that may provide a starting point for this work, bearing in mind that while related, the problem here is different to previous work: we are seeking to learn a goal tree from exhibited behaviour, or to adjust a goal tree to better describe observed behaviour, whereas existing work assumes that the goal tree is correct in structure, and focuses on improving it by learning context conditions, without changing its structure.

Providing explanations of the behaviour of autonomous systems is both challenging and crucial. It is challenging to do in a way that provides useful explanations to humans, and crucial to the use of autonomous systems in a social context, alongside humans. Our paper provides a carefully considered mechanism for explaining autonomous system behaviour. Crucially, the explanatory factors that are the building blocks for our explanations are those concepts that humans use when explaining their behaviour. This choice, which is enabled by the use of BDI-style goal-trees, is well-motivated by literature on how humans explain behaviour, and allows us to provide human-oriented explanations of autonomous system behaviour.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank Dr Damien Mather, at the University of Otago, for statistical advice relating to the second evaluation presented in Section 6. This work was supported by a University of Otago Research Grant (UORG). We would also like to thank the anonymous reviewers for the detailed and insightful comments that helped to improve this paper.

References

[1] L. Floridi, J. Cowls, M. Beltrametti, R. Chatila, P. Chazerand, V. Dignum, C. Luetge, R. Madelin, U. Pagallo, F. Rossi, B. Schafer, P. Valcke, E. Vayena, AI4people—an ethical framework for a good AI society: opportunities, risks, principles, and recommendations, Minds Mach. 28 (4) (2018) 689–707, https://doi.org/10.1007/s11023-018-9482-5.
[2] P. Robinette, W. Li, R. Allen, A.M. Howard, A.R. Wagner, Overtrust of robots in emergency evacuation scenarios, in: C. Bartneck, Y. Nagai, A. Paiva, S. Sabanovic (Eds.), The Eleventh ACM/IEEE International Conference on Human Robot Interaction, HRI 2016, March 7-10, 2016, IEEE/ACM, Christchurch, New Zealand, 2016, pp. 101–108.
[3] M. Winikoff, Towards trusting autonomous systems, in: A. El Fallah Seghrouchni, A. Ricci, T.C. Son (Eds.), Engineering Multi-Agent Systems - 5th International Workshop, EMAS 2017, Sao Paulo, Brazil, May 8-9, 2017, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 10738, Springer, 2017, pp. 3–20.
[4] The IEEE Global Initiative for Ethical Considerations in Artificial Intelligence and Autonomous Systems, Ethically Aligned Design: A Vision For Prioritizing Wellbeing With Artificial Intelligence And Autonomous Systems, Version 1, IEEE, 2016, http://standards.ieee.org/develop/indconn/ec/autonomous_systems.html.
[5] B.Y. Lim, A.K. Dey, D. Avrahami, *Why* and *why not* explanations improve the intelligibility of context-aware intelligent systems, in: D.R. Olsen Jr., R.B. Arthur, K. Hinckley, M.R. Morris, S.E. Hudson, S. Greenberg (Eds.), Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, April 4-9, 2009, ACM, Boston, MA, USA, 2009, pp. 2119–2128.
[6] B.F. Malle, How the Mind Explains Behavior: Folk Explanations, Meaning, and Social Interaction, The MIT Press, ISBN 0-262-13445-4, 2004.
[7] M.M.A. de Graaf, B.F. Malle, People's explanations of robot behavior subtly reveal mental state inferences, in: 14th ACM/IEEE International Conference on Human-Robot Interaction, HRI 2019, Daegu, South Korea, March 11-14, 2019, IEEE, 2019, pp. 239–248.
[8] S. Thellman, A. Silvervarg, T. Ziemke, Folk-psychological interpretation of human vs. humanoid robot behavior: exploring the intentional stance toward robots, Front. Psychol. 8 (2017) 1–14, https://doi.org/10.3389/fpsyg.2017.01962.
[9] A.S. Rao, M.P. Georgeff, An abstract architecture for rational agents, in: C. Rich, W. Swartout, B. Nebel (Eds.), Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann Publishers, San Mateo, CA, 1992, pp. 439–449.
[10] M.E. Bratman, D.J. Israel, M.E. Pollack, Plans and resource-bounded practical reasoning, Comput. Intell. 4 (1988) 349–355.
[11] M.E. Bratman, Intentions, Plans, and Practical Reason, Harvard University Press, Cambridge, MA, 1987.
[12] S. Cranefield, M. Winikoff, V. Dignum, F. Dignum, No pizza for you: value-based plan selection in BDI agents, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 178–184.
[13] M. Winikoff, V. Dignum, F. Dignum, Why bad coffee? Explaining agent plans with valuings, in: B. Gallina, A. Skavhaug, E. Schoitsch, F. Bitsch (Eds.), Computer Safety, Reliability, and Security - SAFECOMP 2018 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Västerås, Sweden, September 18, 2018, Proceedings, in: Lecture Notes in Computer Science, vol. 11094, Springer, 2018, pp. 521–534.
[14] B. Buchanan, E. Shortliffe, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, 1984.
[15] M. Harbers, Explaining agent behavior in virtual training, SIKS dissertation series no. 2011-35, SIKS, Dutch Research School for Information and Knowledge Systems, 2011.
[16] T. Roth-Berghofer, S. Schulz, D. Bahls, D.B. Leake (Eds.), Explanation-Aware Computing, Papers from the 2007 AAAI Workshop, Vancouver, British Columbia, Canada, July 22-23, 2007, AAAI Technical Report, vol. WS-07-06, AAAI Press, 2007.

[17] D. Aha, T. Darrell, M. Pazzani, D. Reid, C. Sammut, P. Stone (Eds.), Proceedings IJCAI-17 Workshop on Explainable AI, XAI, 2017.
[18] D. Aha, T. Darrell, P. Doherty, D. Magazzeni (Eds.), Proceedings IJCAI-18 Workshop on Explainable AI, XAI, 2018.
[19] R. Borgo, M. Cashmore, D. Magazzeni, Towards providing explanations for AI planner decisions, in: D. Aha, T. Darrell, P. Doherty, D. Magazzeni (Eds.), Proceedings IJCAI-18 Workshop on Explainable AI, XAI, 2018, pp. 11–18.
[20] T. Miller, Explanation in artificial intelligence: insights from the social sciences, Artif. Intell. 267 (2019) 1–38, https://doi.org/10.1145/1824760.1824761.
[21] T. Chakraborti, S. Sreedharan, Y. Zhang, S. Kambhampati, Plan explanations as model reconciliation: moving beyond explanation as soliloquy, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 156–163.
[22] T. Lombrozo, Explanation and abductive inference, in: Oxford Handbook of Thinking and Reasoning, 2012, pp. 260–276.
[23] G. Milliez, R. Lallement, M. Fiore, R. Alami, Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring, in: The Eleventh ACM/IEEE International Conference on Human Robot Interaction, HRI '16, IEEE Press, Piscataway, NJ, USA, 2016, pp. 43–50.
[24] P. Busetta, R. Rönnquist, A. Hodgson, A. Lucas, JACK intelligent agents - components for intelligent agents in Java, Tech. rep., Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998, available from http://www.agent-software.com.
[25] M. Winikoff, JACK^{TM} intelligent agents: an industrial strength platform, in: R.H. Bordini, M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.), Multi-Agent Programming: Languages, Platforms and Applications, Springer, 2005, pp. 175–193.
[26] M.J. Huber, JAM: a BDI-theoretic mobile agent architecture, in: Proceedings of the Third International Conference on Autonomous Agents, Agents'99, ACM Press, 1999, pp. 236–243.
[27] M. d'Inverno, D. Kinny, M. Luck, M. Wooldridge, A formal specification of dMARS, in: M. Singh, A. Rao, M. Wooldridge (Eds.), Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, in: LNAI, vol. 1365, Springer-Verlag, 1998, pp. 155–176.
[28] M.P. Georgeff, A.L. Lansky, Procedural knowledge, in: Special Issue on Knowledge Representation, Proc. IEEE 74 (1986) 1383–1398.
[29] F.F. Ingrand, M.P. Georgeff, A.S. Rao, An architecture for real-time reasoning and system control, IEEE Expert 7 (6) (1992).
[30] J. Lee, M.J. Huber, P.G. Kenny, E.H. Durfee, UM-PRS: an implementation of the procedural reasoning system for multirobot applications, in: Proceedings of the Conference on Intelligent Robotics in Field, Factory, Service, and Space, CIRFFSS'94, 1994, pp. 842–849.
[31] R.H. Bordini, J.F. Hübner, M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak Using Jason, Wiley, ISBN 0470029005, 2007.
[32] D. Morley, K. Myers, The SPARK agent framework, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, IEEE Computer Society, Washington, DC, USA, 2004, pp. 714–721.
[33] A. Pokahr, L. Braubach, W. Lamersdorf, Jadex: A BDI reasoning engine, in: R.H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (Eds.), Multi-Agent Programming: Languages, Platforms and Applications, Springer, 2005, pp. 149–174.
[34] J. Thangarajah, L. Padgham, M. Winikoff, Detecting and avoiding interference between goals in intelligent agents, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI, 2003, pp. 721–726.
[35] J. Thangarajah, L. Padgham, Computationally effective reasoning about goal interactions, J. Autom. Reason. 47 (1) (2011) 17–56, https://doi.org/10.1007/s10817-010-9175-0.
[36] Y. Yao, B. Logan, Action-level intention selection for BDI agents, in: C.M. Jonker, S. Marsella, J. Thangarajah, K. Tuyls (Eds.), Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016, ACM, 2016, pp. 1227–1236, http://dl.acm.org/citation.cfm?id=2937103.
[37] Y. Yao, B. Logan, J. Thangarajah, Robust execution of BDI agent programs by exploiting synergies between intentions, in: D. Schuurmans, M.P. Wellman (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, AAAI Press, 2016, pp. 2558–2565, http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12148.
[38] M. Winikoff, An AgentSpeak meta-interpreter and its applications, in: Third International Workshop on Programming Multi-Agent Systems, ProMAS, in: LNCS, vol. 3862, Springer, 2006, pp. 123–138 (post-proceedings, 2006).
[39] Vincent J. Koeman, Koen V. Hindriks, Catholijn M. Jonker, Omniscient debugging for cognitive agent programs, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 265–272.
[40] J. Thangarajah, L. Padgham, M. Winikoff, Detecting and exploiting positive goal interaction in intelligent agents, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, ACM Press, 2003, pp. 401–408.
[41] S. Visser, J. Thangarajah, J. Harland, F. Dignum, Preference-based reasoning in BDI agent systems, Auton. Agents Multi-Agent Syst. 30 (2) (2016) 291–330, https://doi.org/10.1007/s10458-015-9288-2.
[42] B. Burmeister, M. Arnold, F. Copaciu, G. Rimassa, BDI-agents for agile goal-oriented business processes, in: Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems, AAMAS [Industry Track], IFAAMAS, 2008, pp. 37–44.
[43] M. Winikoff, G. Sidorenko, Evaluating a mechanism for explaining BDI agent behaviour, https://profwinikoff.files.wordpress.com/2021/07/evaluating.pdf, 2021.
[44] P.D. Allison, N.A. Christakis, Logit models for sets of ranked items, Sociol. Method. 24 (1994) 199–228, https://www.jstor.org/stable/270983.
[45] M. Winikoff, Debugging agent programs with "Why?" questions, in: Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems, 2017, pp. 251–259.
[46] B.F. Malle, J. Knobe, M.J. O'Laughlin, G.E. Pearce, S.E. Nelson, Conceptual structure and social functions of behavior explanations: beyond person–situation attributions, J. Pers. Soc. Psychol. 79 (2000) 309–326, https://doi.org/10.1037/0022-3514.79.3.309.
[47] T. Miller, Contrastive explanation: a structural-model approach, CoRR, arXiv:1811.03163 [abs].
[48] J. McClure, D.J. Hilton, Are goals or preconditions better explanations? It depends on the question, Eur. J. Soc. Psychol. 28 (1998) 897–911, https://doi.org/10.1002/(SICI)1099-0992(1998110)28:6<897::AID-EJSP902>3.0.CO;2-P.
[49] M. Colledanchise, P. Ögren, Behavior trees in robotics and AI: an introduction, CoRR, arXiv:1709.00084 [abs].
[50] R. Alford, V. Shivashankar, M. Roberts, J. Frank, D.W. Aha, Hierarchical planning: relating task and goal decomposition with task sharing, in: S. Kambhampati (Ed.), Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, IJCAI/AAAI Press, 2016, pp. 3022–3029, http://www.ijcai.org/Abstract/16/429.
[51] V. Shivashankar, Hierarchical goal networks: formalisms and algorithms for planning and acting, Ph.D. thesis, University of Maryland, College Park, MD, USA, 2015, http://hdl.handle.net/1903/16698.
[52] C.W. Geib, R.P. Goldman, A probabilistic plan recognition algorithm based on plan tree grammars, Artif. Intell. 173 (11) (2009) 1101–1132, https://doi.org/10.1016/j.artint.2009.01.003.
[53] S. Sardiña, L. Padgham, A BDI agent programming language with failure handling, declarative goals, and planning, Auton. Agents Multi-Agent Syst. 23 (2011) 18–70, https://doi.org/10.1007/s10458-010-9130-9.
[54] D. Singh, S. Sardiña, L. Padgham, Extending BDI plan selection to incorporate learning from experience, Robot. Auton. Syst. 58 (9) (2010) 1067–1075, https://doi.org/10.1016/j.robot.2010.05.008.
[55] D. Singh, S. Sardiña, L. Padgham, S. Airiau, Learning context conditions for BDI plan selection, in: W. van der Hoek, G.A. Kaminka, Y. Lespérance, M. Luck, S. Sen (Eds.), 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010, Toronto, Canada, May 10-14, 2010, IFAAMAS, 2010, pp. 325–332, https://dl.acm.org/citation.cfm?id=1838252.

[56] D. Singh, S. Sardiña, L. Padgham, G. James, Integrating learning into a BDI agent for environments with changing dynamics, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, IJCAI/AAAI, 2011, pp. 2525–2530.
[57] A. Guerra-Hernández, A. El Fallah-Seghrouchni, H. Soldano, Learning in BDI multi-agent systems, in: J. Dix, J. Leite (Eds.), Computational Logic in Multi-Agent Systems, Springer, Berlin, Heidelberg, 2005, pp. 218–233.
[58] T. Phung, M. Winikoff, L. Padgham, Learning within the BDI framework: an empirical analysis, in: R. Khosla, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Springer, Berlin, Heidelberg, 2005, pp. 282–288.