

22nd International Joint Conference on Artificial Intelligence IJCAI-11

Thomas Roth-Berghofer, Nava Tintarev, David B. Leake (Eds.)

Explanation-aware Computing ExaCt 2011

IJCAI-11 Workshop, Barcelona, Catalonia (Spain)

July 16–22, 2011

Proceedings of the 6th International ExaCt workshop

Preface

Both within AI systems and in interactive systems, the ability to explain reasoning processes and results can have substantial impact. Within the field of knowledge-based systems, explanations have been considered as an important link between humans and machines. There, their main purpose has been to increase the confidence of the user in the system's result (persuasion) or the system as a whole (satisfaction), by providing evidence of how it was derived (transparency). Additional AI research has focused on how computer systems can themselves use explanations, for example to guide learning (education).

Current interest in mixed-initiative systems provides a new context in which explanation issues may play a crucial role. When knowledge-based systems are partners in an interactive socio-technical process, with incomplete and changing problem descriptions, communication between human and software systems is a central part. For example, in recommender systems good explanations can help to inspire user trust and loyalty, and make it quicker and easier (efficiency) for users to find what they want (effectiveness). Explanations exchanged between human agents and software agents thus may play an important role in mixed-initiative problem solving.

Other disciplines such as cognitive science, linguistics, philosophy of science, psychology, and education have investigated explanation as well. They consider varying aspects, making it clear that there are many different views of the nature of explanation and facets of explanation to explore. Relevant examples of these include, but are not limited to, open learner models in education, and dialog management and planning in natural language generation.

This volume contains the papers presented at the IJCAI 2011 workshop on Explanation-aware Computing (ExaCt2011) held on July 16–17, 2011 in Barcelona. The major goal of the workshop was to bring researchers, scientists from both industry and academia, and representatives from diverse communities and fields as Informatics, Philosophy, and Sociology, together to study, understand, and explore the aspects of explanation in IT-applications. The papers presented in this volume illustrate some of the variety of perspectives on explanation and recent progress.

There were 12 submissions to ExaCt 2011. Each submission was reviewed by three program committee members. The committee decided to accept 8 papers for oral presentation (of which 1 was retracted) and 3 for poster presentation (of which 1 was retracted). This volume was produced using the EasyChair system.¹

ExaCt 2011² continued a series of workshops begun with a AAAI Fall symposium in 2005 in Washington followed by yearly workshops at such conferences as AAAI, ECAI, and IJCAI. The workshop series aims to draw on multiple perspectives on explanation, to examine how explanation can be applied to further the development of robust and dependable systems and to illuminate system processes to increase user acceptance and feeling of control. The presentations and

¹ <http://www.easychair.org>

² <http://exact2011.workshop.hm>

discussions at this workshop series have shown the value of sharing perspectives to clarify and advance research in explanation-aware computing.

Readers who would like to participate in further discussions on this topic or like to receive further information about future workshops might consider joining the Yahoo!-group `explanation-research`³. More information on explanation research is available on a dedicated website⁴.

May 2011

Thomas Roth-Berghofer
Nava Tintarev
David Leake

³ <http://tech.groups.yahoo.com/group/explanation-research/>

⁴ <http://on-explanation.net>

Workshop Organisation

Chairs

Thomas Roth-Berghofer, University of Hildesheim / DFKI GmbH, Germany
Nava Tintarev, University of Aberdeen, United Kingdom
David B. Leake, Indiana University, USA

Programme Committee

Agnar Aamodt	Department of Computer and Information Science, Norwegian University of Science and Technology
David Aha	Naval Research Laboratory
Martin Atzmüller	University of Kassel
Patrick Brézillon	University of Paris 6
Giuseppe Carenini	University of British Columbia
Jörg Cassens	IMIS, University of Lübeck
Ashok Goel	Georgia Institute of Technology
Pierre Grenon	KMI; IFOMIS
Anders Kofod-Petersen	SINTEF ICT
David Mcsherry	University of Ulster
Hector Munoz-Avila	Lehigh University
Paulo Pinheiro Da Silva	University of Texas at El Paso
Enric Plaza	IIIA-CSIC
Christophe Roche	Equipe Condillac - Universit de Savoie
Olga C. Santos	aDeNu Research Group (UNED)
Sven Schwarz	DFKI GmbH
Gheorghe Tecuci	George Mason University
Douglas Walton	University of Windsor

Additional Reviewer

Björn Forcher, DFKI GmbH

Table of Contents

Papers.

A Natural Language Argumentation Interface for Explanation Generation in Markov Decision Processes	1
<i>Thomas Dodson, Nicholas Mattei and Judy Goldsmith</i>	
Justifying Semantic Search Results By Means Of Semantic Frequency Classes	11
<i>Björn Forcher, Thomas Roth-Berghofer and Stefan Agne</i>	
Learning Event Models that Explain Anomalies	21
<i>Matthew Molineaux, David Aha and Ugur Kuter</i>	
What Just Happened? Explaining the Past in Planning and Execution ...	31
<i>Matthew Molineaux, Ugur Kuter and Matthew Klenk</i>	
Scientific Explanation and Computation	41
<i>James Overton</i>	
Monitoring and explaining reasoning processes in a dialogue system's input interpretation step	51
<i>Daniel Sonntag and Christian Schulz</i>	
Improving User and Decision Support System Teamwork: An Approach Based on Shared Mental Models	61
<i>Iris Van De Kieft, Catholijn Jonker and M. Birna Van Riemsdijk</i>	

Posters.

Consistent Explanations of Answers to Queries in Inconsistent Knowledge Bases	71
<i>Hendrik Decker</i>	
Explanation and Corrective Feedback in Grammar Checking Systems	81
<i>Marie Garnier</i>	

A Natural Language Argumentation Interface for Explanation Generation in Markov Decision Processes ^{*}

Thomas Dodson, Nicholas Mattei, and Judy Goldsmith

University of Kentucky
Lexington, KY 40506, USA

Abstract. A Markov Decision Process (MDP) policy presents, for each state, an action, which preferably maximizes the expected reward accrual over time. In this paper, we present a novel system that generates, in real time, natural language explanations of the optimal action recommended by an MDP while the user interacts with the MDP policy. We rely on natural language explanations in order to build trust between the user and the explanation system, leveraging existing research in psychology in order to generate salient explanations for the end user. Our explanation system is designed for portability between domains and uses a combination of domain specific and domain independent techniques. The system automatically extracts implicit knowledge from an MDP model and accompanying policy. This policy-based explanation system can be ported between applications without additional effort by knowledge engineers or model builders. Our system separates domain-specific data from the explanation logic, allowing for a robust system capable of incremental upgrades. Domain-specific explanations are generated through case-based explanation techniques specific to the domain and a knowledge base of concept mappings for our natural language model.

1 Introduction

A Markov decision process (MDP) is a mathematical formalism which allows for long range planning in probabilistic environments [9]. The work reported here uses fully observable, factored MDPs [1]. A policy for an MDP is a mapping of states to actions that defines a tree of possible futures, each with a probability and a utility. Unfortunately, this policy is a large object, with many potential branches, and is difficult to understand even for sophisticated users.

This complexity prevents many end users from trusting, understanding, and implementing the plans generated from MDP policies [5]. Recommendations and plans generated by computers are not always trusted or implemented by end users of decision support systems. Distrust and misunderstanding are two of the most often user cited reasons for not following a recommended plan or action [7]. For a user unfamiliar with stochastic planning, the most troublesome part of existing explanation systems is the explicit use of probabilities, as humans are demonstrably bad at reasoning with probabilities [11]. Additionally, it is our intuition that the concept of a preordained probability of success or failure at a given endeavor discomforts the average user.

^{*} This work is supported by NSF EAGER grant CCF-1049360. We would like to thank Robert Crawford, Joshua Guerin, and Daniel Michler for their support and helpful discussions.

In this paper we present an explanation system for MDP policies. Our system produces natural language explanations, generated from domain specific and domain independent information, to convince end users to implement the recommended actions. Our system generates arguments [6]; that are designed to convince the user of the “goodness” of the recommended action. While the logic of our arguments is generated in a domain independent way, there are domain specific data sources included. These are decoupled from the explanation interface, to allow a high degree of customization. This allows our base system to be deployed on different domains without additional information from the model designers. If an implementation calls for it, our system is flexible enough to incorporate domain specific language and cases to augment its generated arguments. We implement this novel, argument based approach with natural language text in order to closely connect with the user. Building this trust is essential in convincing the user to implement the policy set out by the MDP [7]. We wish to avoid exposing the user to the specifics of stochastic planning, though we cannot entirely avoid language addressing the inherent probabilistic nature of our planning system.

Our system has been developed as a piece of a larger program working with advising college students about what courses to take and when to take them. It was tested on a subset of a model developed to predict student grades based on anonymized student records, as well as capture student preferences, and institutional constraints [4]. Our system presents, as a paragraph, an argument as to why a student should take a specified set of courses in the next semester. The underlying policy is based on the student’s preferences and abilities. This domain is interesting because it involves users who need to reason in discrete time steps about their long term benefits. Beginning students at a university will have limited knowledge about utility theory and represent a good focus population for studying the effectiveness of different explanations.

2 Background and Related Work

Markov Decision Processes A Markov Decision Process (MDP) is a formal model for planning, when actions are modeled as having probabilistic outcomes. We focus here on factored MDPs [1]. MDPs are used in many areas, including robotics, economics and manufacturing.

Definition 1. An MDP is a tuple, $\langle S, A, T, R \rangle$, where S is a set of states and A is a set of actions, and $T(s'|s, a)$ is the probability that state s' is reached if a is taken in state s , and $R(s)$ is the reward, or utility, of being in state s . If states in S are represented by variable (attribute) vector, we say that the MDP is factored.

A policy for an MDP is a mapping $\pi : S \rightarrow A$. The best policy for an MDP is one that maximizes the expected value (Definition 2) [9]. In our system, policies are computed with respect to the expected total discounted reward, where the discount rate γ is such that $0 \leq \gamma < 1$. The optimal policy with respect to discount γ is one that maximizes the total discounted expected value of the start state (see Definition 2) [9].

Definition 2. The expected value of state s with respect to policy π and discount γ is

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s' | \pi(s), s) * V^\pi(s'). \quad (1)$$

The optimal value function V^* is the value function of any optimal policy π^* [9]. We use the optimal policy, and other domain and model information, to generate natural language explanations for users with no knowledge of probability or utility theory.

Explanation Systems Prior work on natural language explanation of MDP policies is sparse, and has focused primarily on what could be called “policy-based explanation,” whereby the explanation text is generated solely from the policy. The nature of such systems limits the usefulness of these explanations for users who are unfamiliar with stochastic planning, as the information presented is probabilistic in nature. However, these algorithms have the advantage of being entirely domain-independent. A good example of such a system is Khan et al.’s minimal sufficient explanations [5], which chooses explanatory variables based on the occupation frequency of desired future states. Note that, while the algorithms used in policy-based explanation systems are domain-independent, the explanations generated by such systems often rely on the implicit domain-specific information encoded into the model in the form of action and variable names. Other work has focused on finding the variable which is most influential to determining the optimal action at the current state [2], while using an extensive knowledge-base to translate these results into natural language explanations.

Case-based and model-based explanation systems rely, to different extents, on domain specific information. To find literature on such systems, it is necessary to look beyond stochastic planning. Case-based explanation, which uses a database of prior decisions and their factors, called a case base, is more knowledge-light, requiring only the cases themselves and a model detailing how the factors of a case can be generalized to arbitrary cases. Care must be taken in constructing a case base in order to include sufficient cases to cover all possible inputs. Nugent et al.’s KLEF [8] is an example of a case-based explanation system. A model-based explanation system, however, relies on domain-specific information, in the form of an explicit explanation model.

An explanation interface provides explanations of the reasoning that led to the recommendation. Sinha and Swearingen [10] found that, to satisfy most users, recommendation software employing collaborative filtering must be *transparent*, i.e., must provide not only good recommendations, but also the logic behind a particular recommendation. Since stochastic planning methods are generally not well understood by our intended users, we do not restrict our explanations to cover, for example, some minimum portion of the total reward [5], and instead choose explanation primitives that, while still factual, will be most convincing to the user.

Model For this project we focus on an academic advising domain. We use a restricted domain for testing which focuses on completing courses to achieve a computer science minor focus at the University of Kentucky. Our group is also working on generating advising domains that capture all classes in a university [4]; our long-term goal is an end-to-end system to aid academic advisors that build probabilistic grade predictors, model student preferences, plan, and explain the offered recommendations.

The variables in our factored domain are the required courses for a minor focus in computer science: Intro Computer Programming (ICP), Program Design and Problem Solving (PDPS), Software Engineering (SE), Discrete Mathematics (DM), and Algorithm Design and Analysis (ALGO). We include Calculus II (CALC2) as a predictor

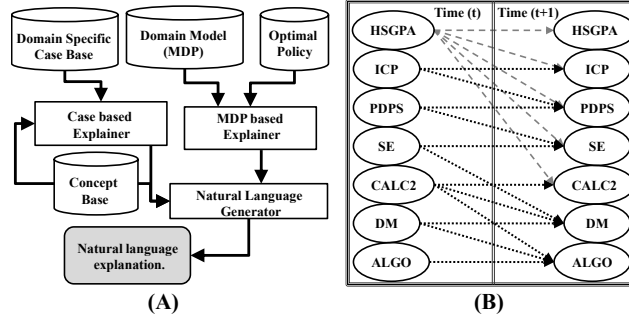


Fig. 1. System organization and data flow (A) and the dynamic decision network (temporal dependency structure) for the academic advising model (B).

course for DM and ALGO due to their strong mathematical components. Each class variable can have values: (G)ood, (P)ass, (F)ail, and (N)ot Taken. An additional variable is high school grade point average, HSGPA; this can have values: (G)ood, (P)ass, (L)ow. The model was hand coded with transition probabilities derived from historic course data at the University of Kentucky.

Each action in our domain is of the form, “Take Course X ,” and only affects variable X . Figure 1-B, shows the temporal dependencies between classes, and implicitly encodes the set of prerequisites due to the near certain probability of failure if prerequisite courses are not taken first. Complex conditional dependences exist between courses due to the possibility of failing a course. CALC2 is not required and we do not place reward on its completion. Taking it correlates with success in DM and ALGO; we want to ensure our model can explain situations where unrewarded variables are important. Most courses in the model have HSGPA, the previous class, and the current class as the priors (except ICP and CALC2 which only have HSGPA as a prior).

The reward function is additive and places a value of 4.0 and 2.0 on Good and Passing grades respectively. Failure is penalized with a 0.0. A discount factor of 0.9 is used to weight early success more than later success. While our current utility function only focuses on earning the highest grades possible as quickly as possible we stress that other utility functions could be used and, in fact, are being developed as part of our larger academic advising project.

3 System Overview

Our explanation system integrates a policy-based approach with case-based and model-based algorithms. However, the model-based system is constructed so the algorithm itself is not domain-specific. Rather, the explanation model is constructed from the MDP and resulting policy and relies on domain-specific inputs and a domain-specific language, in the natural language generation module. Thus, we separate the model dependent factors from the model independent methods. This gives our methods high portability between domains.

Figure 1-A illustrates the data flow through our system. All domain specific information has been removed from the individual modules. We think of each of the modules as generating points of our argument while the natural language generator assimilates all these points into a well structured argument to the user. The assimilated argument is stronger than any of the individual points. However, we can remove modules that are not necessary for specific domains, e.g., when a case base cannot be procured. This allows our system to be flexible with respect to a single model and across multiple domains. In addition, system deployment can happen early in a development cycle while other “points” of the argument are brought online. The novel combination of a case-based explainer, which makes arguments from empirical past data, with a model-based explainer, which makes arguments from future predicted data, allows our system to generate better arguments than either piece alone.

Our system differs from existing but similar systems such as the one designed by Elizalde et al. [2] in several important ways. First, while an extensive knowledge base will improve the effectiveness of explanations, the knowledge base required by our system to generate basic explanations is minimal, and limited to variables which can be determined from the model itself. Second, our model-based module decomposes recommendations from the MDP in a way that is more psychologically grounded in many domains, focusing on user actions instead of variables [3].

Model Based Explanation

The model-based module extracts information from the MDP model and a policy of recommended actions on that model. This module generates explanations based on what comes next — specifically, information about why, in terms of next actions, the recommended action is best. We compare actions in terms of a set of values, called *action factored differential values (AFDVs)* for each possible action in the current state. AFDVs allow us to explain the optimal action in terms of how much better the set of actions at the next state are. E.g., we can model that taking ICP before PDPS is better because taking ICP first improves the expected value of taking PDPS in the next step. We can also highlight how the current action can affect multiple future actions and rewards. This allows our method to explain complex conditional policies without explicit knowledge of the particular conditional. Through the computation of the AFDVs we are able to extract how the current best action improves the assignment of one or more variables under future actions.

This method of explanation allows for a salient explanation which focuses on how the current best action will improve actions and immediate rewards in the next state (the next decision point). Many studies have shown empirically that humans use a hyperbolic discounting function when making long term plans [3]. This discount function places much more value on rewards realized in the short term. In contrast to human reasoning, an MDP uses an exponential discount function when computing optimal policies. The combined effects of human inability to think rationally in probabilistic terms and opposite discounting schemes means there is a fundamental disconnect between the human user and the rational policy [3, 11]. The disconnect between the two reasoning methods must be reconciled in order to communicate MDP policies to human users in terms that they will more readily understand and trust. This translation is achieved through explaining the long term plan in terms of short term gains with AFDV sets.

To generate a usable set of AFDVs from some state s , we define a method for measuring the value of taking an arbitrary two action sequence and then continuing to follow the given policy, π . This is accomplished by modifying the general expression for V^π to accommodate deviation from the policy in the current state and the set of next states:

$$V_2^\pi(s, a_1, a_2) - R(s) = \gamma \sum_{s' \in S} T(s'|s, a_1) \cdot [R(s') + \gamma \sum_{s'' \in S} T(s''|s', a_2) \cdot V^\pi(s'')]. \quad (2)$$

Using V_2^π , we can then compute a single AFDV object for the action to be explained, $\pi(s)$, by computing the value of the two step sequence $\{\pi(s), a\}$ and the value of another two step sequence $\{a_i, a\}$ and taking the difference,

$$\Delta^\pi(s, \pi, a_i, a) = V_2^\pi(s, \pi(s), a) - V_2^\pi(s, a_i, a). \quad (3)$$

To compute a full set of AFDVs for the explanation action, $\pi(s)$, this computation is done $\forall a_i \in A \setminus \pi(s)$ and $\forall a \in A$.

In order to choose variables for explanation, we compute, for each i , $\Delta^\pi(s, \pi, a_i, a)$, to find which actions' utilities will increase after having taken the recommended action. This set of counts gives the number of actions in the current state which cause a greater increase in utility of the action a than the recommended action.

$$x_s^\pi(a) = |\{i : \Delta^\pi(s, \pi, a_i, a) < 0\}| \quad (4)$$

Note that we may have $\forall a \in A : x_s^\pi(a) > 0$, since only the sum of the AFDV set over a_i for the optimal action is guaranteed to be greater than or equal to the sum for any other action. We choose the subset of A for which $x_s^\pi(a)$ is minimal as our explanation variables, and explain $\pi(s)$ in terms of its positive effects on those actions. We can also decompose the actions into corresponding variable assignments and explain how those variables change, leading to higher reward. By focusing on actions we reduce the overall size of the explanation in order to avoid overwhelming the user, while still allowing the most salient variables of the recommended action to be preserved. If more variables are desired, another subset of A can be chosen for which $x_s^\pi(a)$ is greater than the minimum, but less than any other value. While the current method of choosing explanation variables relies on knowledge of the *optimal* policy, the AFDV objects are meaningful for any policy. However, our particular method for choosing the subset of AFDVs for explanation relies on the optimality of the action $\pi(s)$, and would have to be adapted for use with a heuristic policy.

For example, the explanation primitive for a set of future actions with $\pi(s) = \text{act_PDPS}$, $x_s^\pi(\text{act_SE}) = x_s^\pi(\text{act_DM}) = 0$, $x_s^\pi(\text{act_ALGO}) = 1$, and $x_s^\pi(a) = 2$ for all other a is: *The recommended action is act_PDPS, generated by examining long-term future reward. It is the optimal action with regards to your current state and the actions available to you. Our model indicates that this action will best prepare you for act_SE and act_DM in the future. Additionally, it will prepare you for act_ALGO.*

It is possible to construct pathological domains where our domain independent explainer fails to select a best action. In these rare cases, the explainer will default to stating that the action prescribed by the given policy is the best because it leads to the greatest expected reward; this prevents contradictions between the explanation and

policy. The AFDV method will break down if domains are constructed such that the expected reward is 0 within the horizon (2 time steps). This can happen when there are balanced positive and negative rewards. For this reason, we currently restrict our domain independence claims to those domains with only non-negative rewards.

Case-Based Explanation Case-based explanation (CBE) uses past performance in the same domain in order to explain conclusions at the present state. It is advantageous because it uses real evidence, which enhances the transparency of the explanation, and analogy, a natural form of explanation in many domains [8]. This argument from past data combined with our model-based argument from predicted future outcomes creates a strong complete argument for the action recommended by the optimal policy. Our case base consists of 2693 distinct grade assignments in 6 distinct courses taken by 955 unique students. This the University of Kentucky, about all courses taken by students who began their academic tenure between 2001 and 2004.

In a typical CBE system, such as KLEF [8], *a fortiori* argumentation is used in the presentation of individual cases. This presents evidence of a strong claim in order to support a weaker claim. In terms of academic achievement, one could argue that if there is a case of a student receiving a "Fair" in PDPS and a "Good" in SE, then a student who has received a "Good" in PDPS should expect to do at least as well.

In our system, a single case takes the form of: $[scenario1 \rightarrow action \rightarrow scenario2]$, where a scenario is a partial assignment of state variables, and scenario2 occurs immediately after action, which occurs at any time after scenario1. In particular, we treat a single state variable assignment, followed by an action, followed by an assignment to single state variable, usually differing from the first, as a single case. For example, a student having received an A in ICP and a B in PDPS in a later semester comprises a single case with $[scenario1 = \{var_ICP = A\} \rightarrow action = take_PDPS \rightarrow scenario2 = \{var_PDPS = B\}]$. If the same student had also taken CALC2 after having taken ICP, that would be considered a distinct case.

In general, the number of state variables used to specify a case depends on the method in which the case base is used. Two such methods of using a case base are possible: case aggregation and case matching. When using case aggregation, which is better suited to smaller scenarios, the system combines all matching cases into relevant statistics in order to generate arguments. When using case matching, a small number of cases, whose scenarios match the current state closely, would be selected to generate arguments [8]. Case matching methods are more suited to larger scenarios, and ideally use full state assignments.

Our system uses a case aggregation method, as our database does not have the required depth of coverage of our state-space. There are some states which can be reached by our MDP which have few or no cases. With a larger case base, greater specificity in argumentation is possible by considering an individual case to be the entirety of a single student's academic career. However, presenting individual cases still requires that the case base be carefully pruned to generate relevant explanations. Our system instead presents explanations based on dynamically generated statistics over all relevant cases (i.e., assignments of the variables affected by the recommended action). We select the relevant cases and compute the likelihood of a more rewarding variable assignment under a given action. This method allows more freedom to chose the action for which we

present aggregated statistics; the system can pick the most convincing statistics from the set of all previous user actions instead of attempting to match individual cases.

Our method accomplishes this selection in a domain-independent way using the ordered variable assignments stored in the concept base. In our system, there is a single required component of the concept base which must be defined by the system implementer; an ordering in terms of reward value over the assignments for each variable, with an extra marker for a valueless assignment that allows us to easily generate meaningful and compelling case-based explanations. The mapping could also be computed from the model on start-up, but explicitly enumerating the ordering in the concept base allows either the system designer to tweak the case-based explanations in response to user preferences by reordering the values and repositioning the zero-value marker.

For a given state, s , for each variable v_i affected by $\pi(s)$, we consider the naïve distribution, $\phi(v_i)$, over the values of v_i from cases in the database. We compute the conditional distribution, $\phi(v_i|s)$ over the values of v_i given the values to all other variables in s . Then, for each conditional distribution, we examine the probability of a rewarding assignment. We then sort the distributions in order from most rewarding to least, by comparing it to the probability of receiving the assignment from any of the naïve distributions. Conditional distributions which have increased probability of rewarding assignments over the naïve distributions are then chosen to be used for explanation.

For a student in a state such that $var_ICP = Good$, $var_CALC2 = Good$, and $\pi(s_e) = act_PDPS$: since act_PDPS influences only var_PDPS , three grade distributions will be generated over its values: one distribution for all pairs with $var_ICP = Good$, one with $var_CALC2 = Good$, and one over all cases which have some assignment for var_PDPS . If, in the case base, 200 students had $var_ICP = Good$ and $var_PDPS \neq NotTaken$ with 130 “Good” assignments, 40 “Fair”, and 30 “Poor”, giving a [0.65, 0.20, 0.15] distribution; 150 students had $var_CALC2 = Good$ and $var_PDPS \neq NotTaken$ with 100 “Good”, 30 “Fair”, and 20 “Poor”, giving a [0.67, 0.20, 0.13] distribution; while 650 students had $var_PDPS \neq NotTaken$ with 300 “Good”, 250 “Fair”, and 100 “Poor”, giving a [0.47, 0.38, 0.15] distribution, then the distributions indicate that such assignments increase the probability of receiving $var_PDPS = Good$, and the generated explanation primitive is: *Our database indicates that with either $var_ICP = Good$ or $var_CALC2 = Good$, you are more likely to receive $var_PDPS = Good$ in the future.*

Natural Language Generator

In explanations generated by our system, particular emphasis is placed on displaying probabilities in terms that are more comfortable to the target user base, undergraduate students. The subject of such language has been explored in the design of probability elicitation methods, particularly for medical decision making [12]. Even users expressing a preference for numbers don’t necessarily reason well with them [11]. Our explanations replace numerical probabilities with a system of intuitively ordered adverb phrases to express underlying probabilities.

While MDP-based explanations can be generated in a domain-independent way, producing domain-independent natural language explanations is more problematic. The only domain semantics available from the MDP are the names of the actions, variables, and values. These labels, however, tend to be abbreviated or otherwise distorted to conform to technical limitations. Increasing the connection between the language and do-

main increases the user trust and relation to the system by communicating in language specific to the user [7, 10]. Our system uses a relatively simple concept base which provides mappings from variable names and assignments to noun phrases, and action names to verb phrases. This is an optional system component; the domain expert should be able to produce this semantic mapping when constructing the MDP model.

4 Discussion and Study Proposal

Our system successfully generates natural language explanations in real time using domain-independent methods, while incorporating domain language for the final explanation. The concept base also allows designers to insert custom language as a preamble to any or all of the recommendations. This allows the user interface designer flexibility as to how much domain, modeling, and computational information to reveal to the end user. Our concept base includes text stating that recommendations depend on grades (outcomes) so far, and on the user’s preferences. In many applications we expect that users do not want to know how every decision in the system is made; we are building convincing arguments for a general population, not computer scientists. While technically inclined people may want more information regarding the model construction and planning, it is our feeling that most users want to understand what they should do now. Thus, our example explanation does not explain or exhibit the entire policy. The important concept for our end users is not the mathematical structure of a policy, but that future advice will depend on current outcomes. After language substitution, the generated explanations look like: *The recommended action is taking Introduction to Program Design and Problem Solving, generated by examining possible future courses. It is the optimal course with regards to your current grades and the courses available to you. Our model indicates that this action will best prepare you for taking Introduction to Software Engineering and taking Discrete Mathematics in the future. Additionally, it will prepare you for taking Algorithm Design and Analysis. Our database indicates that with either a grade of A or B in Introductory Computer Programming or a grade of A or B in Calculus II, you are more likely to receive a grade of A or B in Introduction to Program Design and Problem Solving, the recommended course.*

This form of explanation offers the advantage of using multiple approaches. The first statement explains the process of generating an MPD policy, enhancing the transparency of the recommendation in order to gain the trust of the user [10]. It makes clear that the planning software is considering the long-term future, which may inspire confidence in the tool. The second statement relies solely on the optimal policy and MDP model. It offers data about expected future performance in terms of the improvement in value of possible future actions, the AFDVs. This part of the explanation focuses on the near future to explain actions which may only be preferable because of far future consequences, thereby leveraging the user’s inherent bias towards hyperbolic discounting of future rewards [3]. The last statement focuses on the student’s past performance in order to predict performance at the current time step and explains that performance in terms of variable assignments. This paragraph makes an analogy between the user’s performance and the aggregated performance of past students. Argument from analogy is very relevant to our domain — academic advisors often suggest, for example, that

advisees talk to students who have taken the course from a particular professor. Additionally, the case-based explanation module can be adapted to take into account user preferences, and therefore make more precise analogies.

We have recently received intuitional approval for a large, multi-staged user study. We informally piloted the system with computer science students at our university, but this informal test fails to address the real issues surrounding user interfaces. Our study will use students from disciplines including psychology, computer science, and electrical engineering, and advisors from these disciplines. We will compare the advice generated by our system and its “most convincing” approach to other systems which use a “most coverage” (with respect to rewards) approach. We will survey both students and advisors to find what, if any, difference exists between these two approaches.

5 Conclusion and Future Work

In this work we have presented a system and design which generates natural language explanations for actions generated by MDPs. This system uses a novel mix of case-based and model-based techniques to generate highly salient explanations. The system design abstracts the domain dependent knowledge from the explanation system, allowing it to be ported to other domains with minimal work by the domain expert. The generated explanations are grounded both psychologically and mathematically for maximum impact, clarity, and correctness. The system operates in real time and is scalable based on the amount of domain specific information available.

References

1. Boutilier, C., Dean, T., Hanks, S.: Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11, 1–94 (1999)
2. Elizalde, F., Sucar, E., Noguez, J., Reyes, A.: Generating explanations based on Markov decision processes. In: *Mexican International Conf. on Artificial Intelligence*. pp. 51–62 (2009)
3. Frederick, S., Loewenstein, G., O’Donoghue, T.: Time discounting and time preference: A critical review. *JEL* 40, 351–401 (2002)
4. Guerin, J.T., Crawford, R., Goldsmith, J.: Constructing dynamic bayes nets using recommendation techniques from collaborative filtering. Tech report, University of Kentucky (2010)
5. Khan, O., Poupart, P., Black, J.: Minimal sufficient explanations for factored Markov decision processes. In: *Proc. ICAPS* (2009)
6. Moore, B., Parker, R.: *Critical Thinking*. McGraw-Hill (2008)
7. Murray, K., Häubl, G.: Interactive consumer decision aids. In: Wierenga, B. (ed.) *Handbook of Marketing Decision Models*, pp. 55–77. Springer (2008)
8. Nugent, C., Doyle, D., Cunningham, P.: Gaining insight through case-based explanation. *JGIS* 32, 267–295 (2009)
9. Puterman, M.: *Markov Decision Processes*. Wiley (1994)
10. Sinha, R., Swearingen, K.: The role of transparency in recommender systems. In: *CHI ’02 Conference Companion*. pp. 830–831 (2002)
11. Tversky, A., Kahneman, D.: Judgement under uncertainty: Heuristics and biases. *Science* 185, 1124–1131 (1974)
12. Witteman, C., Renooij, S., Koele, P.: Medicine in words and numbers: A cross-sectional survey comparing probability assessment scales. *BMC Med. Informatics and Decision Making* 7(13) (2007)

Justifying Semantic Search Results By Means Of Semantic Frequency Classes

Björn Forcher¹, Thomas Roth-Berghofer^{1,3},
Stefan Agne¹, and Andreas Dengel^{1,2}

¹ Knowledge Management Department,
German Research Center for Artificial Intelligence (DFKI) GmbH
Trippstadter Straße 122, 67663 Kaiserslautern, Germany

² Knowledge-Based Systems Group, Department of Computer Science,
University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern

³ Institute of Computer Science
University of Hildesheim, Germany

{first.lastname}@dfki.de

Abstract. The research project *Emergent* aims at developing concepts, technologies and business processes for emergent software. One subgoal is to evolve the semantic search engine *KOIOS* that will be used for different domains such as medicine and agriculture. Since semantic search results are not always self-explanatory, an explanation facility is integrated into the engine revealing connections between search and result concepts. The constructed explanations are depicted as semantic networks containing various domain specific concepts. To provide good explanations, concepts must be labeled with adequate terms regarding the level of expertise of their users. In this paper, we present a user experiment and, based on that, define a method that predicts the understandability of terms by means of semantic frequency classes. Regarding the mentioned domains and levels of expertise the vision is to select the most understandable label for a concept if there is more than one label available.

Key words: explanation generation, justification, semantic search, semantic frequency classes

1 Introduction

The research project *Emergent*⁴ aims at developing concepts, technologies and business processes for emergent software combining a wide range of components from different manufacturers in a dynamic and flexible manner. Qualities of the emergent system are not directly traceable to the isolated features of the system's components, but rather to how those components interact. One subgoal is to evolve the explanation-aware semantic search engine *KOIOS* that can be used for

⁴ <http://www.software-cluster.com/index.php?lang=en>

different domains such as medicine and agriculture. Here, explanation-awareness means, that the engine ‘understands’ the search problem of the user and that it is capable of justifying the search result.

KOIOS [3] enables keyword-based search on RDFS ontologies and is designated for users with different levels of expertise in the respective domains, for instance, experts, advanced users or laypersons. Representational constructs of formal ontologies are used to annotate and retrieve all kinds of information. For instance, *KOIOS* is used to realize a keyword-based search for medical documents. In this case, *KOIOS* employs parts of the Foundational Model of Anatomy (FMA) [6] and a representation of the International Classification of Diseases, Version 10 (ICD-10) [5]. For example, the Wikipedia article about the heart can be annotated with the anatomical concept *heart* or *ventricle*. These concepts can then be used for retrieving the respective article. In this context, the search algorithm leverages the structure of the ontology for finding articles that are annotated with the same or adjacent concepts.

Since semantic search results are not always self-explanatory, an explanation facility is integrated into *KOIOS* that reveals a connection between search and annotation concepts. The constructed explanations are depicted as semantic networks containing various domain specific concepts. To provide good explanations concepts must be labeled with adequate ontology terms regarding the level of expertise of users. For instance, suppose an explanation that contains the concept *cardiac*. This label is appropriate for medical experts but not for medical laypersons who rather prefer *heart* as concept label. For this purpose, we developed a method to predict the understandability of terms by considering their usage frequency of terms in natural language regarding different levels of expertise [2].

In this paper, we present an experiment that evaluates the understandability of terms by means of *semantic frequency classes*, a measure we introduce here, that denotes the frequency of terms with specific meaning. We will show that semantic frequencies provide better results compared to common frequencies.

This paper is structured as follows. The next section gives a short overview about relevant research on explanations. Section 3 presents the functionality of *KOIOS* and its explanation component with respect to the medical domain. Section 4 contains our conceptual work on improving explanations by means of semantic frequency classes. Section 5 describes the user experiment and discusses its results in order to realize a method to generate adapted explanation with respect to the level of expertise of users. We conclude the paper with a brief summary and outlook.

2 Related Work

Explanation facilities were important components of Expert Systems (ES) supporting the user’s needs and decisions. Swartout and Moore formulated five desiderata for ES explanations, among them *Fidelity* and *Understandability* [8]. Fidelity means that the explanation must be an accurate representation of what

the ES really does. Explanations should build on the same knowledge the system uses for reasoning. As this is sometimes not possible due to, e.g., efficiency reasons, Wick and Thompson [11] developed the Reconstructive Explainer (REX). REX transforms a trace, *i. e.*, a line of reasoning, into a plausible explanation story, *i. e.*, a line of explanation. The degree of coupling between the trace and the explanation is controlled by a filter that can be set to one of four states regulating the transparency of the filter. The more information of the trace is let through the filter, the more closely the line of explanation follows the line of reasoning. *KOIOS* takes up the theme of (re-)constructing explanations.

Understandability is the second desideratum which we focus on in here. We developed an easy-to-compute, but effective method to predict the understandability of semantic search result labels [2, 7]. However, we found some shortcomings that we address in the following.

3 The Semantic Search Engine *KOIOS*

KOIOS is a semantic search engine that enables keyword-based search on graph-shaped RDF data. *KOIOS* first computes a set of relevant SPARQL queries allowing users to select the most appropriate ones. Consequently, a selected query is used to search in a respective triple store. The main advantage is that users do not need explicit knowledge about the query syntax or the underlying ontologies. The algorithm of *KOIOS* is based on the work of [10] and [9].

For computing queries, the keywords are mapped to elements of input RDF data in a first step. Based on these *keyword elements*, a search is performed on the graph data to determine a *connecting element*. As its name implies, it is a particular graph element connecting the keyword elements. The paths between the keyword elements and the connecting elements are analyzed to create a *matching subgraph*. For each subgraph, a conjunctive query is constructed by mapping the graph elements to query elements. To achieve a scalable search, the input data is preprocessed obtaining two data indices. The *keyword index* is used to realize the keyword mapping. For searching, a *graph index* is derived which represents a kind of summary of the input data containing structural patterns only. At runtime, this index is extended with the keyword elements obtained by means of the keyword index. Therefore, the extended graph comprises sufficient information to compute the SPARQL query.

The presentation of search results in common search engines always includes some hint why a particular result is relevant with respect to the keywords. Google, for instance, shows title, text snippets and URI of retrieved documents and highlights keywords contained therein. This information represents some kind of explanation or justification which describes a connection between search and result. In the end, it can help the user to open only relevant documents.

In case of *KOIOS* explaining search results for end-users is bit more complicated. The justification has to contain the information how keywords are mapped to keyword elements and how the elements are connected. Figure 1 illustrates a simple example in which a user searches for Wikipedia articles about the hu-

man brain. Therefore, the user types the keywords *brain* and *document* into the textfield at the very top. After computing the SPARQL queries, the left panel describes the keyword-element mapping. The center panel contains the highest ranked SPARQL query, whereas the right panel presents the respective result.

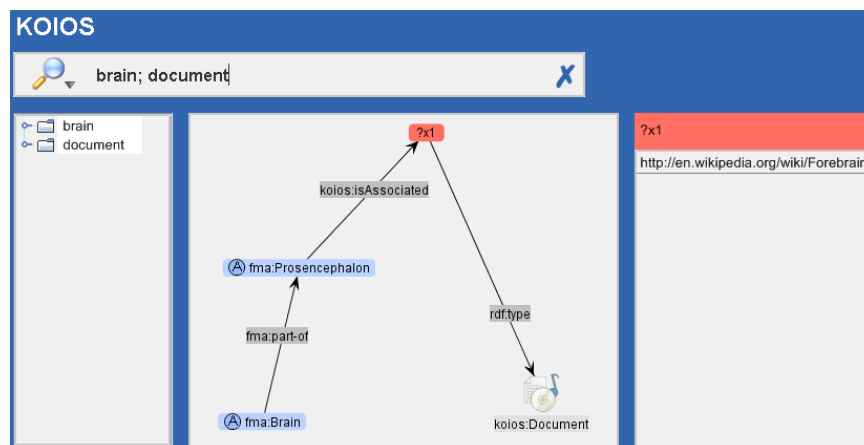


Fig. 1. Screenshot of the KOIOS GUI

The presentation of SPARQL queries is used to select the most relevant queries and to justify the search result as well. In this case, the query justifies why the Wikipedia article is returned: The article is about the *forebrain* which is annotated (associated) with the concept *prosencephalon* that is, in turn, a part of the *brain*. However, the justification may not be the best explanation for all kinds of users because especially medical laypersons may not know that prosencephalon is the Latin term for the forebrain. On the contrary, medical experts may consider the explanation good because the prosencephalon fits their daily language.

Fortunately, the FMA provides both labels. The contribution of this work is to provide a method that helps to select the most appropriate label for concepts in an explanation depicted as semantic network. The method is based on the frequency of terms in the daily language that is explained in the following section.

4 Semantic Frequency Classes

We developed a simple method to improve understandability of semantic search results which is based on a past user experiment as presented in [2, 7]. The method emanates from the hypothesis that the degree of knowledge about scientific terms correlates with the usage frequency in daily language [4]. The more often a (medical or agricultural) term is used in natural language the more users ‘know’ that term, and, thus, the better understandable it is. In order to handle

the usage frequency of terms in daily language a useful statistical measure are frequency classes. The frequency class of a term t is defined as follows [1]:

Definition 1. Let C be a text corpus and let $f(t)$ denote the frequency of a term $t \in C$. The frequency class $c(t)$ of a term $t \in C$ is defined as $\lfloor \log_2(f(t^*)/f(t)) \rfloor$, where t^* denotes the most frequently used term in C .

In many English corpora, t^* denotes the term ‘the’ which corresponds to frequency class 0. Thus, a more uncommonly used term has a higher frequency class. In the following, we refer to any frequency class $c(t) = i$ as c_i , whereas the maximum frequency class is denoted with c_{max} .

For evaluating the personal estimation of medical knowledge only German terms of FMA and ICD-10 consisting of one word were selected. For each frequency class $c_{th}, \dots, c_{max}$ we selected a random set of terms of the same size. We considered only frequency classes at certain threshold c_{th} because all terms below are generally known. We determined the frequency classes with the help of a web service of the project *Deutscher Wortschatz*⁵ regarding a huge representative German text corpus. All selected terms were randomly subdivided into four tests each containing a varying number of frequency classes. Every test person had to estimate their knowledge about each term of exactly one test on a scale from 1 to 5 indicating their *Personal Knowledge Estimation* (PKE). Two kinds of users participated in the experiment: medical experts and laypeople. Regarding laypeople, the experiment result confirms the hypothesis as described above. In contrast, the average PKE per term of medical experts is generally very high so that a trend cannot be observed.

The main objective of the past experiment was not to verify a correlation between users’ degree of knowledge and frequency classes. In fact, the intention was primarily to define a method based on frequency classes as a means of prognosis whether user with a certain level of expertise probably know a term or require supporting information. For that purpose, we defined for medical laypersons the general function $p(t)$ that predicts the familiarity of terms by considering its frequency class $c(t) = c_i$. The function is based on three intervals of frequency classes representing a generalization of the PKE scale. If $n, m \in \mathbb{N}$ and if $n < m$, the function $p(t)$ is defined as follows:

$$p(t) = \begin{cases} \text{well known,} & \text{iff } c_0 \leq c(t) < c_n \\ \text{in need of support,} & \text{iff } c_n \leq c(t) < c_m \\ \text{completely unknown,} & \text{iff } c_m \leq c(t) \leq c_{max} \end{cases}$$

With respect to the experiment results, the parameters for medical laypeople are $n = 15$ and $m = 21$. In this case, the term ‘Gehirn’ (brain) is a *well known* term, in contrast to the term ‘Prosencephalon’ (prosencephalon) which is probably a *completely unknown* term for most people. Based on that function we could select understandable labels for medical laypersons as described in Section 3.

The result of the experiment revealed some interesting outliers. The first can be traced to the term *Atlas*. It is an ambiguous term whose meaning in

⁵ <http://wortschatz.uni-leipzig.de/>

a geological context is quite common. In contrast, its meaning as first cervical vertebra is relatively unknown. The second kind of outliers can be traced to some compounds which are quite common for the German language. The meaning of those terms can easily be derived but their occurrence in daily language is rare. The third problem concerns the inflection of words in the German language (and in the English language as well). The term ‘Zecken’ (ticks) has a significantly lower frequency class as the base word ‘Zecke’ (tick) and thus, it is more often used.

In this work we try to improve that method by handling the problems as described above. As compounds require more complex natural language processing, we only tried to handle the inflection and the ambiguity problem in our current work. For that reason, we propose *semantic frequency classes* which can be defined as follows:

Definition 2. Let C be a text corpus, let t_x be a lexeme of the lexicon L with unambiguous meaning, and let $t_{x_1}, \dots, t_{x_n}$ be all inflection forms of t_x . The semantic frequency class $c_s(t_x)$ of $t_x \in C$ is defined as $\lfloor \log_2(f(t^*)/(f(t_{x_1}) + \dots + f(t_{x_n}))) \rfloor$, where t^* denotes the most frequently used term in C .

In other words, semantic frequency classes denote how often words and their flexion forms are used in a predefined meaning. For instance, it describes how often the word *Atlas* and its flexion forms *Atlasse* and *Atlanten* are really used as first cervical vertebra in the text corpus. Thus, the semantic frequency class of *Atlas* is probably much higher as the common frequency class of it. Consequently, we can define the method $p_s(t)$ that predicts the familiarity of terms by considering its semantic frequency class $c_s(t) = c_{s_i}$:

$$p_s(t) = \begin{cases} \text{well known,} & \text{iff } c_0 \leq c_s(t) < c_{s_n} \\ \text{in need of support,} & \text{iff } c_{s_n} \leq c_s(t) < c_{s_m} \\ \text{completely unknown,} & \text{iff } c_{s_m} \leq c_s(t) \leq c_{max} \end{cases}$$

In the next section we will describe an experiment in which we will show that $p_s(t)$ provides better results than $p(t)$.

5 Evaluation

We assume that the degree of knowledge of domain specific terms can be correlated with semantic frequency classes. The more often a term is used in natural language the more users know about that term. In order to verify the applicability of this assumption we conducted a user experiment. In this experiment the test persons should estimate their knowledge concerning several medical and agricultural terms. Here, we use the German Wikipedia⁶ articles as text corpus since Wikipedia is free available, it can be downloaded and used offline. In addition, it is a kind of lexicon whose articles and respective URLs can be used to compute the semantic frequency classes.

⁶ <http://de.wikipedia.org/wiki/Wikipedia:Hauptseite>

5.1 Experiment Setup

For evaluating the personal estimation of medical knowledge we determined a random list of 300 terms for the medical and for the agricultural domain as well. For each domain, we chose as set of 50 domain specific Wikipedia articles and computed a ranked list of all terms contained therein. The rank is based on a simple TF/IDF weight whereas the chosen articles represented one abstract document and all German Wikipedia articles the complete document corpus. Only thirty-three percent of the best rated terms were taken into account for the experiment in order to get rid of stop words and other domain irrelevant terms. Domain experts reviewed the remaining terms and removed further irrelevant terms. That list we call the domain specific reference list l_r . In addition, all terms were removed for which there was no corresponding article in the Wikipedia. Subsequently, we computed semantic frequency classes for all remaining terms. We randomly selected twenty five terms for each frequency class $c_{11}, \dots, c_{22}$ so that there were altogether 12 lists $l_{r_{11}}, \dots, l_{r_{22}}$ with 25 terms (300 terms altogether) for each domain in the test.

The experiment is provided on the Internet and can be accessed at the following URL: <http://projects.dfki.uni-kl.de/tfccUI/faces/index.jsp>. The URL links to a start page where test persons have to specify some basic information and where they can get some basic information about the test. At first, the test person has to select a domain and the respective level of expertise. After that, the test persons have to estimate their knowledge about 15 terms on a scale from 1 to 5 (see Table 1) indicating their *Personal Knowledge Estimation* (PKE). Exactly one term is randomly selected from each semantic frequency class list $l_{r_{11}}, \dots, l_{r_{22}}$.

Table 1. Personal Knowledge Estimation [2]

(1)	term is completely unknown;
(2)	term has been heard of, but cannot be properly integrated into a medical context;
(3)	the meaning of the term is known or it can be derived. In addition, the term can be vaguely integrated into a medical context;
(4)	the meaning of the term is known and it can be associated with further medical terms;
(5)	term is completely clear and comprehensive knowledge can be associated.

For determining the semantic frequency class of a term t , we first determined all Wikipedia articles $a_{t_1}, \dots, a_{t_q}$ whose title is equal to the term. For instance, for the term *Atlas* we got the following Wikipedia articles representing the different lexemes for that term:

– [http://de.wikipedia.org/wiki/Atlas_\(Gebirge\)](http://de.wikipedia.org/wiki/Atlas_(Gebirge))

- [http://de.wikipedia.org/wiki/Atlas_\(Kartografie\)](http://de.wikipedia.org/wiki/Atlas_(Kartografie))
- [http://de.wikipedia.org/wiki/Atlas_\(Halswirbel\)](http://de.wikipedia.org/wiki/Atlas_(Halswirbel))
- *etc.*

Now, the correct article (lexeme) must be identified which we call reference article a_{t_r} . For this purpose, the most similar article to the reference list is computed using the cosine similarity: $a_{t_r} = \max_{i=1}^m (\cos(l_d, a_{t_q}))$. In a second step, we determined the frequency of the inflection forms of a lexeme in the Wikipedia corpus. For that purpose, we implemented a Java based connector to the German Wiktionary containing many inflection forms of the German language. Then we retrieved all Wikipedia articles $a_{w_1}, \dots, a_{w_t}$ containing at least one of the flexion forms. Again, every retrieved article was compared with the reference $a_{t_1}, \dots, a_{t_q}$ using the cosine similarity. Now we counted only flexion forms in those articles which were most similar to a_{t_r} . The result is the semantic frequency of the article a_{t_r} with the title t . In the third step the semantic frequency class for the lexeme is calculated. Therefore, the most frequent term has to be determined which is a trivial task.

When comparing common and semantic frequency classes we find that 24 percent of all terms have different values. This difference indicates that inflection and ambiguity is a significant problem.

5.2 Evaluation

At the time of writing 130 tests were made. 55 tests by medical laypersons, nine tests by medical advanced persons, four tests by medical experts, 50 tests by agricultural laypersons, ten tests by agricultural advanced persons and two tests by agricultural experts. As there are too few evaluations for advanced persons and experts so far we will focus only on medical and agricultural laypersons here. Figure 2 depicts an average value of the PKE as function of the semantic frequency classes for medical (left chart) and agricultural (right) laypersons.

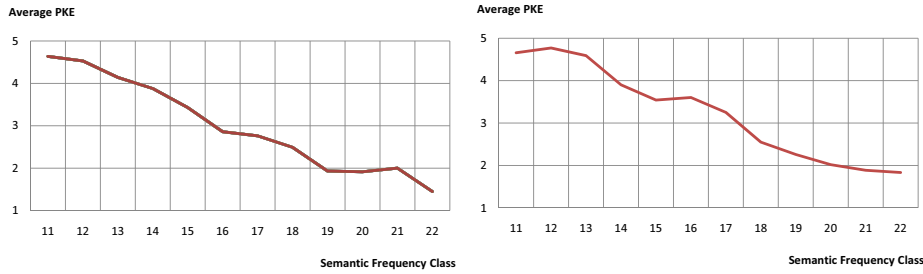


Fig. 2. Experiment results for medical (left) and agricultural (right) laypersons

As in the past experiment, the result for the medical and agricultural laypersons confirms the hypotheses that the semantic frequency of terms correlates

with PKE of users. What remains is the verification whether semantic frequency classes provide better results than common frequency classes or that $p_s(t)$ performs better than $p(t)$. For this purpose, we generalized the collected experiment ratings to obtain ground truth data by mapping the PKE of each experiment term to the values *known*, *in need of support* and *unknown* of the mentioned methods. Subsequently, we conducted a stratified fivefold cross validation for each method using the ground truth data for training and testing as well. In total, we considered three different methods for each domain. The first one is derived from Wikipedia and common frequency classes (FCW), the second one depends on Wikipedia and semantic frequency classes (SFCW), whereas the third relies on common frequency classes of the German corpus of the project *Deutscher Wortschatz* (FCDW). Table 2 aggregates the accuracy values of the validation for both domains.

Table 2. Accuracy values of the considered methods

	Medicine	Agriculture
FCW	61.86%	68.05%
SFCW	65.18%	70.67%
FCDW	68.88%	73.67%

The table above shows that the use of semantic frequency classes only provides slightly better results in comparison to common frequency classes with respect to the Wikipedia corpus. In general, the semantic frequency class of a term t is higher than the other one and thus, it may be used in several meanings. A subsequent interview showed that people rate their knowledge about familiar terms higher as in comparison to terms they never heard of (even if they do not know anything about that term). However, the usage of common frequency classes of the German corpus of the project *Deutscher Wortschatz* provides the best results. This may be due to the fact that this corpus fits more the natural language of laypersons in contrast to the more scholarly language of Wikipedia.

We can expect that the usage of semantic frequency classes and the above mentioned German corpus would increase the accuracy. Unfortunately, we cannot compute semantic frequency classes for this corpus since it consists of a collection of disjoint and disordered sentences.

6 Summary and Outlook

In this paper we presented the explanation facility of the semantic search engine *KOIOS*, focussing on the understandability desideratum. The explanation facility employs the same RDFS ontologies as *KOIOS* and uses reconstructive explanations for justifying semantic search results. Explanations are depicted as semantic networks revealing a connection between search and result. To improve

justifications we conducted an experiment dedicated to experts, advanced users and laypersons in the medical and agricultural domain. The objective of the experiment was to determine a correlation between users' degree of knowledge and semantic frequency classes of terms in the mentioned domains. Based on that evaluation we described a method to label explanation concepts with one of several possible terms in order to give understandable and adequate explanations with respect to the mentioned levels of expertise.

Next steps of our research are to increase the amount of test data and to further refine the proposed method for evaluating not only single terms but also multi-word expressions.

Acknowledgment

This work was funded by the German Federal Ministry of Education and Research (BMBF) in the EMERGENT project under grant nr. 01IC10S01.

References

1. zu Eissen, S.M., Stein, B.: Intrinsic plagiarism detection. In: ECIR. pp. 565–569 (2006)
2. Forcher, B., Schumacher, K., Sintek, M., Roth-Berghofer, T.: Evaluating the intelligibility of medical ontological terms. In: Proceedings of the 5th Workshop on Knowledge Engineering and Software Engineering (KESE-2009) (2009)
3. Forcher, B., Sintek, M., Roth-Berghofer, T., Dengel, A.: Explanation-aware system design of the semantic search engine koios. In: Proceedings of the ECAI-10 workshop on Explanation-aware Computing (ExACt 2010) (2010)
4. Hayes, D.P.: The growing inaccessibility of science. *Nature* 356, 739–740 (1992)
5. Möller, M., Ernst, P., Sintek, M., Biedert, R., Dengel, A., Sonntag, D.: Representing the international classification of diseases version 10 in OWL. In: Proc. of the International Conference on Knowledge Engineering and Ontology Development (KEOD). Valencia, Spain (25-28 October 2010)
6. Rosse, C., Mejino, J.L.V.: *Anatomy Ontologies for Bioinformatics: Principles and Practice*, vol. 6, chap. The Foundational Model of Anatomy Ontology, pp. 59–117. Springer (December 2007), http://sigpubs.biostr.washington.edu/archive/00000204/01/FMA_Chapter_final.pdf
7. Roth-Berghofer, T., Forcher, B.: Improving the understandability of semantic search explanations. *Int. J. Knowledge Engineering and Data Mining* 1(3), 216–234 (2011)
8. Swartout, W.R., Moore, J.D.: Explanation in second generation expert systems. *Second generation expert systems* pp. 543–585 (1993)
9. Tran, D.T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In: Proc. of the 25th Int. Conf. on Data Engineering (ICDE'09). Shanghai, China (March 2009)
10. Wang, H., Zhang, K., Liu, Q., Tran, T., Yu, Y.: Q2semantic: A lightweight keyword interface to semantic search. In: In Proceedings of the 5th International Semantic Web Conference (ESWC'08 (2008)
11. Wick, M.R., Thompson, W.B.: Reconstructive expert system explanation. *Artif. Intell.* 54(1-2), 33–70 (1992)

Learning Event Models that Explain Anomalies

Matthew Molineaux¹, David W. Aha², and Ugur Kuter³

¹Knexus Research Corporation; 9120 Beachway Lane; Springfield, VA 22153; USA

²Naval Research Laboratory, Code 5514; Washington DC; USA

³University of Maryland, Institute for Advanced Computer Studies,
College Park, MD 20742; USA

¹matthew.molineaux@knexusresearch.com

²david.aha@nrl.navy.mil

³ukuter@cs.umd.edu

Abstract. In this paper, we consider the problem of improving the goal-achievement performance of an agent acting in a partially observable, dynamic environment, which may or may not know all events that can happen in that environment. Such an agent cannot reliably predict future events and observations. However, given event models for some of the events that occur, it can improve its predictions of future states by conducting an explanation process that reveals unobserved events and facts that were true at some time in the past. In this paper, we describe the DISCOVERHISTORY algorithm for discovering an explanation for a series of observations in the form of an event history and a set of assumptions about the initial state. When knowledge of one or more event models is not present, we claim that the capability to learn these unknown event models would improve performance of an agent using DISCOVERHISTORY, and provide experimental evidence to support this claim. We provide a description of this problem, and suggest how the DISCOVERHISTORY algorithm can be used in that learning process.

Keywords: Abductive explanation, planning and execution, relational learning

1 Introduction

Recent research in robotics has demonstrated the need for intelligent agents that can recognize dangers to themselves and respond proactively, before humans can communicate their needs. For example, the Mars rovers must navigate for many hours at a time without a human in the loop, and the current generation of autonomous underwater vehicles can operate for similar periods of time. To robustly recognize emerging dangers, these types of agents must reason about the state information that they can observe, and continuously learn new information about how the world works.

Most commonly, agents that reason about many possible states use a *belief state* representation (Russell and Norvig, 2003), in which an agent plans over the set of “belief states”, which is the power set of standard states. While effective in small domains, the size of the belief state in use grows exponentially with the number of hidden literals. We consider instead a model in which the agent begins by making assumptions about hidden facts in the world (most often a closed world assumption), and then *explains*

what has happened when unexpected state observations occur. This is made possible by modeling the types of exogenous *events* that can occur, and determining what sequences of such events are consistent with the observations. Unlike work in diagnosis (e.g., McIlraith 1998; Iwan and Lakemeyer 2003), we treat these events as *uncontrollable*; that is, the rules of the world *require* them to happen when certain conditions are met. This treatment is more realistic for many environments and, by narrowing the set of possible explanations, it allows an agent to correct initial assumptions about the world. An agent can use these events in planning as well as in explanation, which allows explanations to improve predictions made in the future.

We previously created an algorithm, DISCOVERHISTORY, that corrects assumptions in dynamic, partially observable environments (Molineaux et al., in submission). It assumes that the space of possible events is fully available to the agent. However, this assumption is too strong to meet the needs of real-world agents that must operate on their own. How should a Mars Rover respond, for example, if it unexpectedly encounters a Martian parasite? Clearly its designers cannot plan for all possible events, so the robot must respond without a good description of how interstellar parasites behave. Ideally, it should observe the parasite’s behaviors and explain what events cause them, thus learning its own model of environment events.

This paper is organized as follows. Section 2 surveys related work in explanation and diagnosis. Section 3 reviews a formalism for explanation of the past and DISCOVERHISTORY. Section 4 introduces an environment designed to examine its behavior, while Section 5 describes an empirical study that examines the benefits of improving the environment model used by DISCOVERHISTORY. Section 6 sketches an algorithm for learning event models, and Section 7 concludes.

2 Related Work

Our work extends the work described by Molineaux et al (in submission), who introduced DISCOVERHISTORY but, unlike here, assumed that the agent is given knowledge on the complete space of possible events. Here we focus on explaining unexpected states by learning the models of *undetected* events. That is, the role of explanations in this paper is to help an agent to understand its environment, rather than communicate its reasoning to humans. This differs from other visions on the role of explanations in AI systems, such as explanation-aware computing, which concerns software-user interactions (e.g., Atzmeuller and Roth-Berghofer, 2010). Likewise, self-explaining agents (e.g., Harbers et al., 2010) and explainable AI systems (van Lent et al., 2004) can generate explanations of their behavior, again for human consumption. While several methods have been investigated that allow agents to explain their observations (e.g., Schank and Leake (1989) describe a case-based approach), we are not aware of similar approaches that learn event models by abducting explanations from unexpected state changes.

While many automated planners can operate in partially observable dynamic environments, most ignore this challenging learning problem. Rather than attempt to discover the causes of the unexpected state, they instead dynamically replan from it (e.g., Myers, 1999; Ayan et al., 2007; Yoon et al., 2007). Those few exceptions that *do* learn tend to focus on other learning tasks, such as selecting which plans to apply (Corchado

et al., 2008), reducing the time to generate optimal plans when replanning (Rachelson et al., 2010), or reducing the need for future replanning tasks (Jimenez et al., 2008).

Reinforcement learning techniques can be used to learn policies for solving planning problems in partially-observable environments, such as by modeling tasks as Partially Observable Markov Decision Processes (POMDPs) (Russell and Norvig, 2003). Our work differs because we use a relational representation for environment models that permits greater scalability and greater generalization.

Using an extension of the situation calculus, McIlraith (1998) describes an approach that uses goal regression and theorem proving to generate diagnostic explanations, defined as conjectured sequences of actions that caused an observed but unexpected behavior. Iwan and Lakemeyer (2003) addressed related problems using a similar approach, though they did not distinguish actions from events, they focused on single observations, and they did not empirically test an implementation of their formalization. Sohrabi et al. (2010) followed up with an empirical analysis of planning techniques to perform diagnosis, and extended the implementation to also find a consistent set of initial assumptions. Their problem is similar to ours, except that they treat events as actions, which allows them to occur without a strict causal relationship to a prior event. Furthermore, these systems are not incorporated into a plan-executing agent.

ASAS (Josephson and Bharathan 2006) is a “dynamic abducer” that can continuously maintain an explanation of its environment and revise its beliefs. However, it is not clear whether this process of belief revision is used to improve future predictions.

Research on learning goals (Cox and Ram, 1999; Ram and Leake, 1995) is also related in that explanation failures are used to trigger learning. However, these goals are not related to the problem of learning event models or choosing actions.

3 Planning and Explanation

We are interested in algorithms that achieve goals in partially observable, dynamic domains by executing a sequence of actions. We model the task of finding a sequence of actions to accomplish a goal as a planning problem, and employ a Hierarchical Task Network planner, SHOP2, for this purpose (Nau et al, 2003). However, since our domains are partially observable and dynamic, the world is not entirely predictable. Rather than enumerating all possible states resulting from its actions, which is highly computationally expensive, our agent makes assumptions about the environment and creates a plan that will accomplish its goals if those assumptions hold. During execution, if the agent’s assumptions are incorrect, the agent will observe facts that do not match its predictions. We call this difference between predictions and observations a *discrepancy* or *anomaly*. Section 3.1 presents our definition of explanations for these observations.

To recover from anomalies, we use DISCOVERHISTORY, which corrects the agent’s assumptions about the environment by searching for an explanation of the anomaly. This explanation will include a revised set of assumptions about the world, which can then be used to improve the next planning iteration. Having been informed more about the environment, the planner can now create a better plan. Section 3.2 summarizes the DISCOVERHISTORY algorithm.

3.1 Definitions

We use the standard definitions from classical planning for variable and constant symbols, logical predicates and atoms, literals, groundings of literals, propositions, and actions (Nau et al, 2003). We assume a state is encoded by a set of logical propositions.

Let $\mathcal{P}$ be the finite set of all possible propositions for describing an environment. An environment is partially observable if an agent only has access to it through *observations* that do not cover the complete state, where an observation is encoded as a set of propositions. We let $\mathcal{P}_{\text{obs}}$ be the set of all propositions that the agent can observe in the world. An observation associates a truth value with each of these propositions. Further, let $\mathcal{P}_{\text{hidden}}$ be the set of *hidden* (state) propositions that an agent *cannot* observe; for example, a robot’s exact location may be hidden to it if it has no GPS contact.

An *event template* is defined syntactically the same as a classical planning operator: (name, preconds, effects), where name, the *name* of the event, is a literal, and preconds and effects, the *preconditions* and *effects* of the event, are sets of literals. We assume that an event always occurs immediately when all of its preconditions are met in the world.

We formalize the planning agent’s knowledge about the changes in its environment as an *explanation* of the world. We define a finite set of symbols $\mathcal{T} = \{t_0, t_1, t_2, \dots, t_n\}$, called *occurrence points*. An *ordering relation* between two occurrence points is denoted as $t_i \prec t_j$, where $t_i, t_j \in \mathcal{T}$.

There are three types of *occurrences*. An *observation occurrence* is a pair of the form (obs, t) where obs is an observation. An *action occurrence* is a pair of the form (a, t) where a is an action (performed by the agent). Finally, an *event occurrence* is a pair (e, t) where e is an event. In all of the occurrence forms, t is an occurrence point. Given an occurrence o , we define occ as a function such that $\text{occ}(o) \mapsto t$; that is, occ refers to the occurrence point t of any observation, action, or event.

An *execution history* is a finite sequence of observations and actions $\text{obs}_0, a_1, \text{obs}_1, a_2, \dots, a_k, \text{obs}_{k+1}$. A planning agent’s *explanation* of the world given an execution history is a tuple $\chi = (C, R)$ such that C is a finite set of occurrences that includes the execution history and zero or more event occurrences that happened according to that explanation. R is a partial ordering over a subset of C , described by ordering relations $\text{occ}(o_i) \prec \text{occ}(o_j)$ such that $o_i, o_j \in C$. As a shorthand, we sometimes will say $o_i \prec o_j$, which is true when $\text{occ}(o_i) \prec \text{occ}(o_j)$.

We use the definitions $\text{knownbefore}(p, o)$ and $\text{knownafter}(p, o)$ to refer to the truth of a proposition p before or after an occurrence o occurs. Let o be an action or event occurrence. Then, the relation $\text{knownbefore}(p, o)$ is true when $p \in \text{preconds}(o)$. Similarly, the relation $\text{knownafter}(p, o)$ is true when $p \in \text{effects}(o)$. If o is an observation occurrence and $p \in \text{obs}$, then both $\text{knownbefore}(p, o)$ and $\text{knownafter}(p, o)$ hold.

We say that an occurrence o is *relevant* to a proposition p if the following holds: $\text{relevant}(p, o) \equiv \text{knownafter}(p, o) \vee \text{knownafter}(\neg p, o) \vee \text{knownbefore}(p, o) \vee \text{knownbefore}(\neg p, o)$. We use the predicates $\text{prior}(o, p)$ and $\text{next}(o, p)$ to refer to the prior and next occurrence relevant to a proposition p . That is to say, $\text{prior}(o, p) = \{o' \mid \text{relevant}(p, o') \wedge \neg \exists o'' \text{ s.t. } \text{relevant}(p, o'') \wedge o' \prec o'' \prec o\}$. Similarly, $\text{next}(o, p) = \{o' \mid \text{relevant}(p, o') \wedge \neg \exists o'' \text{ s.t. } \text{relevant}(p, o'') \wedge o \prec o'' \prec o'\}$.

The *proximate cause* of an event occurrence (e, t) is an occurrence o that satisfies the following three conditions with respect to some proposition p : (1) $p \in \text{preconds}(e)$, (2) $\text{knownafter}(p, o)$, and (3) there is no other occurrence o' such that $o \prec o' \prec (e, t)$. Every event occurrence (e, t) , must have at least one proximate cause, meaning that by condition 3, every event occurrence (e, t) must occur immediately after its preconditions are satisfied.

An *inconsistency* is a tuple (p, o, o') where o and o' are two occurrences in χ such that $\text{knownafter}(\neg p, o)$, $\text{knownbefore}(p, o')$, and there is no other occurrence o'' such that $o \prec o'' \prec o' \in R$ and p is relevant to o'' .

An explanation $\chi = (C, R)$ is *plausible* if and only if the following holds:

1. There are no inconsistencies in χ ;
2. Every event occurrence $(e, t) \in \chi$ has a proximate cause in χ ;
3. Simultaneous occurrences cannot contradict one other: for each pair $o, o' \in C$ such that $\text{occ}(o) = \text{occ}(o')$, and for all p , $\text{knownafter}(p, o) \implies \neg \text{knownafter}(\neg p, o')$, and $\text{knownbefore}(p, o) \implies \neg \text{knownbefore}(\neg p, o')$.
4. If $\text{preconds}(e)$ of an event e are all met at occurrence point t , e must be in χ at t .

3.2 DISCOVERHISTORY algorithm

The DISCOVERHISTORY algorithm (Molineaux et al., 2011) finds a plausible explanation that is consistent with an execution history. At first, our agent makes a set of “optimistic” closed-world assumptions, assuming that the obstacles it knows about are not present, and makes a plan based on those assumptions. Later, when an anomaly occurs, it can revisit these assumptions and replan. An anomaly occurs as a result of one of two problems: (1) there are hidden facts and/or event types present in the environment for which the agent has no model, or (2) the agent’s initial optimistic assumptions are false. The first problem requires learning, which we address in Section 6. The second problem means that the agent can improve its environment model by correcting its explanation, which will require changing its assumptions.

The agent maintains a current explanation at all times. As it executes actions, the agent makes predictions about what events will occur, according to its observations and assumptions, and adds those events to its explanation. However, in a partially-observable, dynamic world, events occur that the agent is unable to predict, which cause anomalies. The agent recognizes an anomaly because one or more inconsistencies appear in its explanation. As described in Section 3.1, an inconsistency occurs when a fact is true after a relevant occurrence, and false before the next relevant occurrence. When these inconsistencies appear, DISCOVERHISTORY resolves them, using one of the following three inconsistency resolution methods.

1. **Event addition.** This searches for a situation where an event that was not predicted caused a literal’s value to change. The new event must occur before the later occurrence, so as to explain the value at that time, but not before the earlier occurrence.
2. **Event removal:** This searches for a situation where one of the predicted events did not happen as predicted.

3. **Assumption change.** Assumptions are only made about the initial state in our explanations, since all following states can be determined from the initial state in a deterministic world. Any event prior to the inconsistency would provide more information about the truth of the literal in question. Therefore, this resolution method is only applied when no relevant prior event is present in the explanation.

Each of these resolution methods can introduce new inconsistencies. Thus, DISCOVER-HISTORY continues resolving inconsistencies until a consistent explanation is found, or until a maximum number M of changes have been made. This search exhaustively enumerates all explanations within M applications of resolution methods.

When a plausible explanation is found, the assumptions made by that explanation are adopted and can be used in prediction. This results in fewer surprises. Since many such surprises are deleterious, goals can be achieved more often, as shown in Section 5. A more detailed explanation of DISCOVERHISTORY is in (Molineaux et al, 2011).

4 The Hazardous Rover Domain

Inspired by the Mars rovers,¹ which operate in a very hazardous environment, we constructed a domain for testing that is predictable enough to make planning tractable, but requires significant explanation to perform well. In our domain, the only actions are to move in one of the four cardinal directions, and the only goals are to reach a location on a six-by-six grid. However, complications can prevent a rover from reaching its goal:

1. Sandstorms, which obscure visibility and can blow a moving rover off course
2. Hidden sand pits, which trap a rover and prevent it from moving
3. Compass malfunctions, which reverse a rover's navigational system, causing it to move north when it intended to move south

Each of these obstacles provides a different challenge for learning. Consider the problem of a rover that has not been programmed to recognize a sandstorm. When the windy season on Mars starts, the rover's sensor mechanisms become covered in sand, and it can no longer recognize its location. It can predict where it should be, but its predictions may deviate more from reality when strong sandstorm winds blow it off course. When the storm clears, the rover observes that it has not reached its desired location. This is an opportunity to learn; the rover could explain that the unexpected change in its position occurs only *when the rover is covered in sand*.

Next consider the case of a rover that finds itself in a hidden sand pit. The rover observes that it can not move any longer when it reaches a certain location. Later on, another rover gets stuck nearby. Due to the same results occurring to both rovers at the same location, it could explain that rovers get stuck *at a certain location*.

Finally, consider the case of compass malfunctions. In early afternoon one day, the rover attempts to move north, and finds that it ends up in a different location than the one it expected, which could have been reached by going to the south. It tries moving north repeatedly, and repeatedly observes a similar result, a location one to the south

¹ <http://marsrover.nasa.gov/home/>

of its last location. By finding the similarities between the transitions, the rover could explain that an event occurred *in early afternoon* that affected its navigation.

With foreknowledge of these types of events, we have shown in prior work (Molineaux et al, 2011) that an agent employing DISCOVERHISTORY outperforms one that only performs replanning and execution monitoring. We now investigate whether an agent that can learn these event models would outperform one that could not.

5 Experiment: Potential for Event Learning

To examine the potential effects of learning in this domain, we compare the performance of several explanation-forming agents with different levels of knowledge about the domain. We measure performance in this domain as the percentage of goals achieved. As explained earlier, the Hazardous Rovers Domain includes three types of obstacles. For each obstacle, one event related to understanding that obstacle is considered as a target for learning. With regard to sandstorms, the event that causes a rover to blow off course in a sandstorm is *sandstorm-blows*; for sand pits, the event that causes the area around a hidden sand-pit to appear abnormally rough is *pit-causes-rough*; the event that causes a compass to malfunction is called *compass-malfunction-occurs*. Eight agents using the DISCOVERHISTORY algorithm are compared. The first agent has knowledge of none of these events; three agents know about one; three agents know about two; and the last agent has knowledge of all three. If learning is effective in this domain, we expect improvement from an agent with less knowledge to an agent with more.

To compare the agents, we randomly generated 25 scenarios in the hazardous rover domain. In each scenario, three rovers are each assigned a random initial location and a random goal that requires at least four moves to reach from the starting location. Each location on the six-by-six grid contains a sand pit with probability 0.3, and with probability 0.66 the sand pit will be hidden. At each possible time point, a storm may begin with probability 0.2, unless a storm is ongoing at that time. Finally, each rover has a compass malfunction within the first 100 time steps. The time step is chosen by taking the floor of the real number found by squaring a number randomly distributed between 0 and 10. This biases the event to occur early in a scenario execution.

Table 1 shows the difference in the average percentage of goals achieved between each pair of agents which differ only by the knowledge of a single event. When all other events are known, the difference averages 13.8%, meaning that the additional knowledge of a single event was sufficient to achieve success instead of failure in 13.8% of cases. The performance advantage caused by all 3 events together is 32.0%, with statistical significance at $p < .0001$. 9 out of 12 comparisons showed that knowledge of an additional event improved goal achievement at a statistically significant level, indicating that learning event models is a promising avenue.

6 Discovering Events

Now that we've shown the utility of event learning in improving the performance of an explanation discovery agent in a partially observable, dynamic planning domain, we consider the problem of inducing those events. This can be neatly divided into two

Table 1: Increase in goals achieved after adding event models. (N/S): non-significant.

Known Events	Perf.	Known events	Perf.	Known events	Perf.
compass-malfunctions	18.6%, p<.01	storm-blows	12.0%, p<.01	compass-malfunctions	10.7%, p<.05
pit-causes-rough		pit-causes-rough		storm-blows	
compass-malfunctions	14.7%, p<.01	storm-blows	5.33%, p<.05	compass-malfunctions	6.66%, (N/S)
pit-causes-rough	16.0%, p<.01	pit-causes-rough	9.32%, p<.05	storm-blows	4.00%, (N/S)
none	16.0%, p<.01	none	6.66%, p<.05	none	4.00%, (N/S)

(a) Effect of adding storm-blows (b) Effect of adding compass-malfunctions (c) Effect of adding pit-causes-rough

separate problems: discovering when something unexplained happens, and finding an effect that explains it. In a first-order logic domain, the first problem results in a set of literal *preconditions*, and the second in a set of literal *effects*. If found, such an event could be used to explain the anomalies encountered by the agent. However, if all of the preconditions of an event are hidden, and no evidence is available to determine their values, then that event cannot be discovered. Therefore, we assume that at least one precondition is observable. Similarly, since hidden effects are difficult to verify, we assume that all events cause at least one observable effect. We discuss these two problems, discovery of preconditions and effects, in Sections 6.1 and 6.2 respectively.

When events are discovered, we need metrics to use in evaluating and adopting those events. We propose to use *predictive power* and *explanatory power*. If an event is usually predicted before it occurs, and rarely predicted before it does not occur, it has high predictive power. Similarly, if knowledge of an event enables explanation after it occurs, and does not lead to false explanations when it has not occurred, it has high explanatory power. While both are useful as metrics of success, they cannot be easily evaluated *in situ* by an agent. However, an agent can approximate them by maintaining a history of the conditions and results of prior predictions and explanations. By re-attempting each prior prediction, the agent can evaluate how many predictions would have been made incorrect by the inclusion of the event in the agent’s knowledge and how many would have been made correct. Similarly, explanations can be re-attempted with the new knowledge; failed explanations that become successful are successes, and successful explanations for which no explanation is now possible are failures. By adopting a confidence metric over predictive and explanatory power, an agent can decide when to include a discovered event in its knowledge base for use in future explanation attempts.

6.1 Discovering Event Preconditions

In the simplest case, all anomalies can be attributed to a single event. This may be a useful assumption in many cases (e.g., when conditions gradually change an agent might be exposed to new types of events incrementally). When the windy season starts on Mars, for example, the rover may already be prepared with knowledge of equipment malfunctions and hidden sand pits. Its problem is to determine what is common among all the anomalies that are encountered. The classification of anomaly-causing states is a relational multi-instance learning task (Dietterich et al, 1997) over the complete

history of observations made by the agent; any sequence of occurrences that contains an anomaly is represented as a “bag” of positive examples, and all other sequences are included in a bag of negative examples. Each example consists of a set of literals known to be true at one time.

As an example, suppose the rover moves to an area on Mars that is covered with hidden sand pits, and surrounded by rough terrain. The rover frequently gets caught in these pits, and each time the experience is anomalous, because it doesn’t know how to predict. The rover builds up a memory of anomalies and the states that preceded them. The rover solves the learning task described above to discover that each anomaly was preceded by states in which rough ground was present and a sand pit was not nearby. These two facts become the preconditions for a new generated event, G1.

6.2 Discovering Event Effects

After discovering the conditions that caused the recognized anomalies, we can assume an event model exists that is triggered by those conditions. We can now predict when this event will occur, but its effects are still unknown. To find the effects of this event, we can add it to the knowledge base as an event model with no effects, and apply a modified version of DISCOVERHISTORY with a new inconsistency resolution method. This method will resolve inconsistencies whose durations overlap with the new event; it will resolve them by assuming that the event has an effect that causes the inconsistency. All such assumptions made in a plausible explanation constitute one possible set of effects for the event model. This modified DISCOVERHISTORY could be used to find a set of plausible explanations for each recognized anomaly. Therefore, using DISCOVERHISTORY to explain all recognized anomalies will result in a set of possible event models, which can be evaluated in terms of their predictive and explanatory power.

Moving on with the hidden sand pits example (Section 6.1), the rover applies DISCOVERHISTORY to resolve inconsistencies. Suppose that the rover observes a literal (`wheels-stuck`) which prevents the move action from working. The rover’s explanation now includes a G1 event that occurred after moving to a rocky area. Now, the inconsistency can be explained: (`wheels-stuck`) is an effect of the G1 event.

7 Conclusion

Our empirical studies showed that learning event models can improve the performance of an agent using DISCOVERHISTORY in a partially-observable dynamic environment. We proposed a framework for acquiring these models from experience. Our next steps will be to implement this framework and evaluate its ability to (1) discover unknown event models and (2) improve its goal-achievement performance over time.

Acknowledgments

This research is partly supported by the Office of Naval Research and Naval Research Laboratory, as well as DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037. The content does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

1. Atzmueller, M., & Roth-Berghofer, T. (2010). Towards explanation-aware social software: Applying the mining and analysis continuum of explaining. In *Proceedings of the Fifth International Workshop on Explanation-aware Computing*. Lisbon, Portugal.
2. Ayan, N.F., Kuter, U., Yaman F., & Goldman R. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. In F. Ingrand, & K. Rajan (Eds.) *Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop*. Providence, RI.
3. Bharathan, V., & Josephson, J. (2006). *Logic Jnl IGPL*. 14 (2): 271-285.
4. Corchado, J.M., Glez-Bedia, M., De Paz, Y., Bajo, J., & De Paz, J.F. (2008). Replanning mechanism for deliberative agents in dynamic changing environments. *Computational Intelligence*, 24(2), 77-107.
5. Cox, M.T., & Ram, A. (1999). Introspective Multistrategy Learning: On the Construction of Learning Strategies, *Artificial Intelligence*, 112:1-55.
6. Dietterich, T.G., Lathrop, R.H., & Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2) 31-71.
7. Harbers, M., Meyer, J.J., & van den Bosch, K. (2010). Explaining simulations through self explaining agents. *Journal of Artificial Societies and Social Simulation*, 13(1)4.
8. Iwan, G., & Lakemeyer, G. (2003). What observations really tell us. *Proceedings of the Twenty-Sixth Annual German Conference on Artificial Intelligence* (pp. 192-208). Hamburg, Germany: Springer.
9. Jimenez, S., Fernandez, F. & Borrajo, D. (2008). The PELA architecture: Integrating planning and learning to improve execution. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (pp. 1294-1299). Chicago, IL: AAAI Press.
10. van Lent, M., Fisher, W., & Mancuso, M. (2004). An explainable artificial intelligence system for small-unit tactical behavior. *Proceedings of the Sixteenth Conference on Innovative Applications of Artificial Intelligence* (pp. 900-907). San Jose, CA: AAAI Press.
11. McIlraith, S.A. (1998). Explanatory diagnosis: Conjecturing actions to explain observations. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 167-179). Trento, Italy: Morgan Kaufmann.
12. Molineaux, M., Kuter, U., and Klenk, M. (2011). What Just Happened? Explaining the Past in Planning and Execution. In *Proceedings of the Sixth International Workshop on Explanation-Aware Computing*.
13. Myers, K.L. (1999). A continuous planning and execution framework. *AI Magazine*, 20(4).
14. Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *JAIR* 20:379-404.
15. Rachelson, E., Abbes, A.B., & Diemer, S. (2010). Combining mixed integer programming and supervised learning for fast re-planning. *Proceedings of the Twenty-Second International Conference on Tools with Artificial Intelligence* (pp. 63-70). Arras, France: IEEE Press.
16. Ram, A., & Leake, D. (eds.). (1995). *Goal-Driven Learning*. MIT Press/Bradford Books.
17. Russell, S. J. & Norvig, P. (2002), *Artificial Intelligence: A Modern Approach (2nd Edition)*, Prentice Hall .
18. Schank, R.C., & Leake, D.B. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40(1-3), 353-383.
19. Sohrabi, S., Baier, J. A., & McIlraith, S. A. (2010). Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning*.
20. Yoon, S., Fern, A., & Givan, B. (2007). FF-replan: A baseline for probabilistic planning. *Proceedings for the Seventeenth International Conference on Automated Planning Systems* (pp. 352-360). Providence, RI: AAAI Press.

What Just Happened? Explaining the Past in Planning and Execution

Matthew Molineaux¹, Ugur Kuter^{2,*}, and Matthew Klenk^{3,**}

¹Knex Research Corporation; 9120 Beachway Lane; Springfield, VA 22153; USA

²Smart Information Flow Technologies, 211 North 1st Street Minneapolis, MN 55401, USA

³Naval Research Laboratory, Code 5514; Washington DC; USA

¹matthew.molineaux@knexusresearch.com

²ukuter@sift.net

³Matthew.Klenk@parc.com

Abstract. We consider the problem of automated planning in partially-observable dynamic environments, where exogenous events that cannot be directly observed affect the state of the world. In these environments, a planner’s knowledge of the world is limited, and state transitions can be both ambiguous and difficult to predict due to that lack of knowledge. We describe a new formalism and new algorithms that enable a planner to proactively expand its knowledge of the environment during planning and execution, by modeling the exogenous events that can occur and forming explanations that reveal information about the world. We have implemented our new algorithms in a variant of the well-known SHOP2 planner that can replan when a failure occurs during plan execution. We have conducted an ablation study in two planning domains to examine the effects of explanation on execution. The results demonstrate that our algorithm successfully increases the performance of an agent using it in two planning domains. This improvement results from the agent having increased knowledge of the environment, which allows it to more accurately predict future events and ultimately make better plans.

Keywords: planning and execution, abductive explanation, autonomous agents

1 Introduction

In real-world environments, unobserved events underlie causal link failures, and understanding these events can be necessary for strong performance. For example, during its mission in 2009, NASA’s Spirit rover was exploring the region known as “Home Plate” on the surface of Mars when its human operators noticed an *inconsistency* with their expectations: Spirit was not moving as much as expected [6]. The operators were not able to observe the surroundings of the rover fully and precisely, but they nevertheless *explained* this inconsistency by *assuming* that the rover was stuck in loose soil. This explanation enabled the operators to formulate a new plan to escape from the unobserved loose soil and continue the mission.

* Author completed this work while at University of Maryland, Institute for Advanced Computer Studies, College Park, MD 20742; USA.

** Now at Palo Alto Research Center (PARC).

Existing research on planning and execution has typically focused on replanning when individual causal links in the plan fail, without reasoning explicitly about the causes of that failure [10]. Work in planning under uncertainty avoids the issue of failure causes by considering them as part of the uncertainty model of the planner, and does not reason explicitly about them [2].

Determining the causes of failures during planning and execution through abductive explanation is the focus of our paper. We describe an algorithm that reasons about a partially-observable, dynamic environment, and generates explanations that describe the cause of inconsistencies between expectations and outcomes. When those causes become known, a planning agent can use this causal knowledge in replanning to better predict future states. In particular:

- We describe a formalism for reasoning about the causes of unexpected observations during planning and execution. This formalism includes a model for exogenous events and their effects on the world.
- We describe an algorithm that generates abductive explanations for what might be happening in the world during planning and execution. This algorithm is designed to improve the performance of a planning agent by increasing its knowledge of hidden state. When replanning, a planning agent will therefore (1) discard plans which would be invalidated by events caused by future event and (2) take advantage of opportunities afforded by future events.
- We have implemented two planning and execution agents: one uses our explanation-generation algorithms and the other does not. Experiments in two planning domains, modified versions of the Rovers and Satellite domains from past International Planning Competitions, showed that the explanation-based agent outperformed the other by a statistically significant margin in these domains.

2 Definitions and Notation

We use the standard definitions from classical planning for variable and constant symbols, logical predicates and atoms, literals, groundings of literals, propositions, planning operators and actions [2, Chapter 2].

Let $\mathcal{P}$ be the finite set of all possible propositions for describing a planning environment. A planning environment is partially observable if an agent only has access to the environment through *observations* which do not cover the complete state. We let $\mathcal{P}_{\text{obs}}$ be the set of all propositions that the agent can observe in the world. An observation associates a truth value with each of these propositions. Further, let $\mathcal{P}_{\text{hidden}}$ be a set of *hidden* propositions representing aspects of the world an agent cannot observe; for example, the exact location may be hidden to a robot with no GPS contact.

An *event template* is defined syntactically the same as a classical planning operator: (name, preconds, effects), where name, the *name* of the event, preconds and effects, the *preconditions* and *effects* of the event, are sets of literals. We use effects^- and effects^+ to denote the negative and positive literals in effects, respectively. An *event* is a ground instance of an event template. We assume that an event always occurs immediately when all of its preconditions are met in the world.

We formalize the planning agent's knowledge about the changes in its environment as an *explanation* of the world. We define a finite set of symbols $\mathcal{T} = \{t_0, t_1, t_2, \dots, t_n\}$, called *occurrence points*. An *ordering relation* between two occurrence points is denoted as $t_i \prec t_j$, where $t_i, t_j \in \mathcal{T}$.

There are three types of *occurrences*. An *observation occurrence* is a pair of the form (obs, t) where obs is an observation. An *action occurrence* is a pair of the form (a, t) where a is an action. Finally, an *event occurrence* is a pair (e, t) where e is an event. In all of the occurrence forms, t is an occurrence point. Given an occurrence o , we define occ as a function such that $\text{occ}(o) \mapsto t$; that is, occ refers to the occurrence point t of any observation, action, or event.

An *execution history* is a finite sequence of observations and actions $\text{obs}_0, a_1, \text{obs}_1, a_2, \dots, a_k, \text{obs}_{k+1}$. A planning agent's *explanation* of the world given an execution history is a tuple $\chi = (C, R)$ such that C is a finite set of occurrences that includes each obs_i for $i = 0, \dots, k-1$ and each action a_j for $j = 1, \dots, k$ for some number k . C may also include zero or more event occurrences that happened according to that explanation. R is a partial ordering over a subset of C , described by ordering relations $\text{occ}(o_i) \prec \text{occ}(o_j)$ such that $o_i, o_j \in C$. As a shorthand, we sometimes will say $o_i \prec o_j$ if and only if $\text{occ}(o_i) \prec \text{occ}(o_j)$.

We use the definitions $\text{knownbefore}(p, o)$ and $\text{knownafter}(p, o)$ to refer to the truth of a proposition p before or after an occurrence $o \in C$ occurs. Let o be an action or event occurrence. Then, the relation $\text{knownbefore}(p, o)$ is true when $p \in \text{preconds}(o)$. Similarly, the relation $\text{knownafter}(p, o)$ is true when $p \in \text{effects}(o)$. If o is an observation occurrence and $p \in \text{obs}$, then both $\text{knownbefore}(p, o)$ and $\text{knownafter}(p, o)$.

We say that an occurrence o is *relevant* to a proposition p if the following holds:

$$\begin{aligned} \text{relevant}(p, o) \equiv & \text{knownafter}(p, o) \vee \text{knownafter}(\neg p, o) \vee \\ & \text{knownbefore}(p, o) \vee \text{knownbefore}(\neg p, o). \end{aligned}$$

We use the predicates $\text{prior}(o, p)$ and $\text{next}(o, p)$ to refer to the prior and next occurrence relevant to a proposition p . That is to say, $\text{prior}(o, p) = \{o' \mid \text{relevant}(p, o') \wedge \neg \exists o'' \text{ s.t. } \text{relevant}(p, o'') \wedge o' \prec o'' \prec o\}$. Similarly, $\text{next}(o, p) = \{o' \mid \text{relevant}(p, o') \wedge \neg \exists o'' \text{ s.t. } \text{relevant}(p, o'') \wedge o \prec o'' \prec o'\}$.

The *proximate cause* of an event occurrence (e, t) is an occurrence o that satisfies the following three conditions with respect to some proposition p : (1) $p \in \text{preconds}(e)$, (2) $\text{knownafter}(p, o)$, and (3) there is no other occurrence o' such that $o \prec o' \prec (e, t)$. Every event occurrence (e, t) , must have at least one proximate cause, so by condition 3, every event occurrence must occur immediately after its preconditions are satisfied.

An *inconsistency* is a tuple (p, o, o') where o and o' are two occurrences in χ such that $\text{knownafter}(\neg p, o)$, $\text{knownbefore}(p, o')$, and there is no other occurrence o'' such that $o \prec o'' \prec o' \in R$ and p is relevant to o'' .

An explanation $\chi = (C, R)$ is *plausible* if and only if the following holds:

1. There are no inconsistencies in χ ;
2. Every event occurrence $(e, t) \in \chi$ has a proximate cause in χ ;
3. For every pair of simultaneous occurrences such that $o, o' \in C$ and $\text{occ}(o) = \text{occ}(o')$, there may be no conflicts before or after: for all p , $\text{knownafter}(p, o) \implies \neg \text{knownafter}(\neg p, o')$, and $\text{knownbefore}(p, o) \implies \neg \text{knownbefore}(\neg p, o')$.

	(rover-at r L0)	(rover-at r L1)	
t_i	true	false	Observation
t_{i+1}			Navigate Action
t_{i+2}	true false		Rover-Moves Event
t_{i+3}	true	false	Observation

← Inconsistency

```

(:action navigate
:parameters (?r - rover ?dir - dir)
:precondition
  (and (available ?r) (>= (energy ?r) 8))
:effect
  (and (attempting-move ?r ?dir) (decrease (energy ?r) 8)))

(:event rover-moves
:parameters (?r - rover)
:precondition
  (and (attempting-move ?r ?dir) (not (blocked ?r))
        (rover-at ?r ?src) (can_traverse ?r ?src ?dest)
        (visible ?src ?dest))
:effect
  (and (not (rover-at ?r ?src)) (at ?r ?dest)
        (not (attempting-move ?r ?dir))))

```

Fig. 1. Example of an inconsistent explanation, with occurrence points ordered on the left hand side. Relevant action and event descriptions are given on the right. Boolean values are the known-before and knownafter relations; for example, the value “false” at top right indicates that the relation $\text{knownbefore}((\neg(\text{rover-at } r \text{ L1})), o_i)$ holds.

4. If $\text{preconds}(e)$ of an event e are all satisfied at an occurrence point t , e is in χ at t ;

Example 1. Suppose that a rover r attempts to move after its wheel has, unobserved, become stuck. Figure 1 illustrates part of an inconsistent explanation with the prior observation occurrence at t_i , in which the rover is observed at location L0, followed by a navigate action occurrence at t_{i+1} directing the rover to location L1, followed by an event occurrence o_{i+2} at t_{i+2} that changes the rover’s location from L0 to L1, followed by the succeeding observation occurrence o_{i+3} at t_{i+3} stating the rover is at L0.

Two inconsistencies then exist between the event occurrence and observation occurrence: $\langle(\text{rover-at } r \text{ L0}), o_{i+2}, o_{i+3}\rangle$ and $\langle\neg(\text{rover-at } r \text{ L1}), o_{i+2}, o_{i+3}\rangle$.

3 Generating Abductive Explanations

This section describes DISCOVERHISTORY, our search algorithm for generating plausible explanations by recursively applying refinements to implausible explanations, based on a series of observations and actions. DISCOVERHISTORY is designed to be used by a planning agent in a partially-observable dynamic environment. The planning agent takes actions, predicts events, and receives observations. Each time an observation is received, the agent creates an explanation consisting of all these occurrences and calls DISCOVERHISTORY to refine that explanation as necessary. DISCOVERHISTORY generates successive explanations by perceiving and attempting to resolve inconsistencies in the current explanation given its observations.

Algorithm 1 shows a high-level description of DISCOVERHISTORY. The base case occurs when the current explanation is plausible or no refinements can be made. In the recursive case, REFINES generates a set of explanations for each inconsistency, and each explanation is recursively considered by DISCOVERHISTORY. Cycles are prevented by maintaining a list of changes to the explanation (not shown) and cutting off branches which cause repetitions of the same inconsistency.

Algorithm 1: A high-level description DISCOVERHISTORY.

```

1 Procedure DISCOVERHISTORY( $\chi$ )
2 begin
3   if Inconsistencies( $\chi$ ) =  $\emptyset$  then
4      $\chi \leftarrow \text{FINDEXTRAEVENTS}(\chi)$ 
5     if Inconsistencies( $\chi$ ) =  $\emptyset$  then return  $\{\chi\}$ 
6   arbitrarily select an  $i \in \text{Inconsistencies}(\chi)$ 
7    $X \leftarrow \text{REFINE}(\chi, i)$ 
8   foreach  $\chi_{new} \in X$  do  $X \leftarrow X \cup \text{DISCOVERHISTORY}(\chi_{new})$ 
9   return  $X$ 
10 end

```

Let χ be a planner's current explanation of the world which includes obs, the most recent observation received by the agent. If no inconsistencies are present in χ , then the first condition for plausibility (see Section 2) is met. Plausibility conditions 2 and 3 are assumed to be true of the initial explanation, and are maintained throughout the resolution process. Once no inconsistencies exist, therefore, DISCOVERHISTORY updates χ for plausibility condition 4 by adding any events that have been caused by the changes to the explanation in FINDEXTRAEVENTS. The new events that are added to the explanation may cause new inconsistencies. Otherwise, all four conditions for a plausible explanation are met and DISCOVERHISTORY returns the explanation χ in Line 5.

If there are inconsistencies between any pair of occurrences in χ , then the function Inconsistencies(χ) will return them. In Line 6, DISCOVERHISTORY selects an inconsistency i from Inconsistencies(χ) and attempts to resolve it.¹ The REFINE subroutine (Line 7 of Algorithm 1) finds all explanations that result from resolving each individual inconsistency and recursively calls itself on each. We further describe REFINE below.

We refer to the initial explanation constructed by the agent as χ_0 . We use this explanation as the basis for our definition of goodness. An explanation χ_a is *better than* χ_b if DISCOVERHISTORY requires fewer changes to transform χ_0 into χ_a than χ_b . We use an iterative deepening search to find all explanations at a minimum depth. Since each invocation of REFINE adds a single change to its base explanation, these explanations are the *best* explanations by our goodness definition. To prevent searches from continuing indefinitely, we employ a maximum depth bound (not shown in pseudocode). The search conducted by DISCOVERHISTORY has a maximum depth of $|C| * (2^{|E|})^2$ and a maximum branching factor of $|E|$, where E is the finite set of all possible events.

There are three possible ways for REFINE to resolve an inconsistency in an explanation, each of which has different conditions for applicability. They are: (1) adding a new occurrence, (2) removing an occurrence, and (3) hypothesizing a different initial occurrence. All applicable methods must be applied, resulting in multiple explanations. Each resolution may create or resolve other inconsistencies, so the inconsistencies found in refined explanations are not necessarily a subset of those found in the parent. To save space, we omit the pseudo-code for REFINE. We detail the resolution strategies below.

¹ Any inconsistency may be selected; it does not affect the correctness of the algorithm. It may affect the algorithm's efficiency, but we do not discuss this topic further.

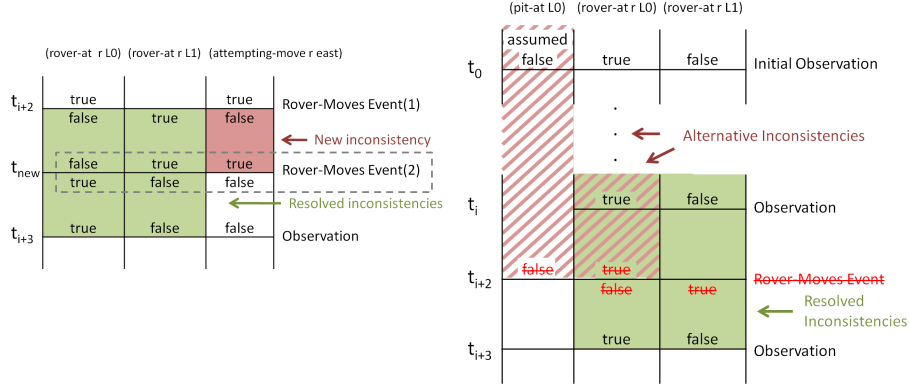


Fig. 2. (a) Example of adding an occurrence (left). (b) Example of removing an occurrence (right).

Adding a New Event Occurrence. Let $\chi = (C, R)$ be an explanation with an inconsistency $i = (p, o, o')$. One way to resolve an inconsistency is to show that some occurrence changed the value of a literal in between the preceding event o and the following event o' . An event o'' relevant to p must be found such that $o \prec o'' \prec o'$.

To add a new occurrence, REFINE starts by generating the set S of possible consecutive ordered pairs of occurrence points between $occ(o)$ and $occ(o')$, given the partial-ordering R . Then, for every e such that $effects(e) \models p$ and each $(t, t') \in S$, REFINE creates a new occurrence point t and a new event occurrence $o'' = (e, t)$. Then the algorithm adds o'' into C and updates the partial ordering R with $o \prec o'' \prec o'$. This results in a new explanation that does not contain the inconsistency i . Each resulting explanation is returned in a set to DISCOVERHISTORY for further refinement.

Example 2. Continuing Example 1, the event Rover-Moves causes the rover to enter a different location, so it could be added to resolve the inconsistency $(\neg (\text{rover-at } r \text{ L1}), o_{i+2}, o_{i+3})$. In order for this to work, a new occurrence must be added between t_{i+2} and t_{i+3} (see Figure 2(a)). REFINE creates a new explanation with added occurrence $o_{new} = (e, t_{new})$ and new ordering relations $t_{i+2} \prec t_{new} \prec t_{i+3}$. A new inconsistency is also generated (see Figure 2(a)). The new inconsistency occurs because another precondition of the Rover-Moves event is the literal $(\text{attempting-move } r \text{ east})$, which is false after such an event occurs. The new inconsistency, $((\text{attempting-move } r \text{ east}), o_{i+2}, o_{new})$, will need to be eliminated in a recursive call to DISCOVERHISTORY.

Removing an occurrence. Another possible way to resolve (p, o, o') , where o and/or o' is an event, is to refine the current explanation to generate new explanations in which either o or o' is removed.

REFINE accomplishes this as follows. The algorithm first creates a new set of occurrences, C' , by removing o from C . Then, in order to explain why the occurrence does not happen, one of the $preconds(e)$ must be found not to hold at occurrence point t . Therefore, REFINE creates a new explanation for each precondition p' of e by creating a *removal occurrence* $o_r = (e_r, occ(o))$. The occurrence o_r occurs at the same time,

and instead of o , and the new event e_r in this occurrence is the reserved removal event e_r , which has no effects and satisfies $\prec(e_r) = \{\neg p'\}$. Due to the presence of the removal event, no new occurrence can be added to the resulting explanation which would cause o to recur. REFINER returns the entire set of resulting explanations, each of which resolves the inconsistency.

Example 3. REFINER can use the above mechanism for removing occurrences to resolve the same inconsistency resolved in example 2 by removing o_{i+2} . Note that no inconsistencies are generated by removing it (recall from example 1 that the rover really didn't move). Finally, the preconditions are examined to look for a precondition which could explain why o_{i+2} might not have occurred. Two preconditions on o_{i+2} are shown in Figure 2(b): (rover-at r L0) and $(\neg (\text{pit-at L0}))$. A new explanation is created for each, with a removal occurrence corresponding to the negated precondition. Each dummy occurrence will cause a new inconsistency, as shown in Figure 2(b).

Hypothesizing an initial value. Given an inconsistency (p, o, o') , where o refers to the initial observation in the execution history, and p is not observable, a different initial occurrence o may be hypothesized. When this is the case, we generate a new explanation by adding to χ an event occurrence $o_p = (e_p, t_0)$. The event e_p in this occurrence is the reserved initially-true event e_p , which has no preconditions or negative effects, and satisfies $\text{effects}^+(e_p) = \{p\}$. This operation has no side effects to any other literal, and thus will never cause new inconsistencies.

Example 4. One of the alternate inconsistencies found in example 3, $(\neg (\text{pit-at L0}), occ_0, occ_{i+2})$, has the characteristics required for hypothesizing an initial occurrence. A pit-at literal is not observable, and the prior occurrence of the inconsistency is occ_0 . Therefore, the discrepancy can be resolved by adding the event occurrence $o_{(\text{pit-at L0})}$.

4 Experimental Evaluation

We examined the performance of DISCOVERHISTORY in the context of planning and execution in two partially-observable domains. These domains have been engineered to include events that are triggered by hidden facts. Thus, events will occur at execution time and can not be predicted due to lack of knowledge during planning time.

Rovers-With-Compass (RWC) is a navigation domain with hidden obstacles inspired by the difficulties encountered by the Mars Rovers. Specifically, individual locations may be windy, sandy, and/or contain sand pits, which the rover cannot observe directly. Sandy locations cause the rover to be covered in sand; while covered in sand, the rover cannot perceive its location or recharge. Sand pits stop the rover from moving; the rover can dig itself out at a high energy cost. Windy locations clear the sand off of the rover, but due to a malfunction, may confuse the rover's compass, causing it to move in the wrong direction. When rovers run out of energy, they stop moving.

Satellite-With-Malfunctions (SWM) is based on the Satellites domain from the 2002 International Planning Competition. The objective in each scenario is to acquire images of various phenomena and transmit them to earth. Our additions to this domain include

Table 1. Statistical t -test results, comparing our explanation-based and non-explanation based systems in the RWC and SWM domains.

Domain	Non-DHAgent	DHAgent	t -test
Rovers	65.3%	78.7%	0.001
Satellite	52.5%	76.0%	< 0.001

various causes of satellite malfunction: supernova explosions, which can damage sensitive instruments if pointed toward them; fuel leaks, which cause fuel reserves to diminish rapidly; and motor malfunctions, which delay a satellite’s turn to a new perspective. When fuel reserves are depleted, no further goals can be accomplished.

We compared the performance of two agents, an explanation-based agent, called DHAgent (short for DISCOVERHISTORYAgent) and a non-explanation-based agent, called Non-DHAgent; each uses a modified version of the SHOP2 planner [9] to create plans that take into account events that are known to occur, and they use the same representation of the domain and the events that occur in it. During execution, when an observation contradicts the state predicted by SHOP2, each agent generates a new planning state and replans, then continues executing.

The two agents differ in how they construct the new planning state. Non-DHAgent removes all observable facts from the SHOP2 predictions, and adds in instead the facts in the environment observation. DHAgent first runs DISCOVERHISTORY. If DISCOVERHISTORY generates a plausible explanation, the explanation-based agent then generates a new planning state based on the most recent truth value for each predicate found in that explanation. If none is found, it falls back on the behavior of Non-DHAgent.

We wrote a problem generator for each domain that randomly creates an initial state including both observable and hidden facts and goals. For the RWC domain, each starting state contained 3 rovers, and a goal for each rover that required it to move to a new destination. Goal destinations were generated randomly such that the rover must cross at least 3 distinct locations to accomplish the goal. Each scenario took place on a 6×6 grid of locations connected in the four compass directions. Hidden state was assigned independently for each location and condition with probability p .

Each randomly-generated SWM problem had 3 satellites and required the attainment of 8 image acquisition goals. Each image target was chosen randomly from a set of 20. Targets were associated with supernovae, fuel leaks, and motor malfunctions based on a probability p .

We randomly generated 25 problems in each domain; Table 1 shows a comparison of the performance of DHAgent and Non-DHAgent. Here, performance is defined as a percentage of goals completed. For the RWC domain, the probability of hidden difficulties was $p = 0.1$; in the SWM domain, the probability of hidden state that induces malfunctions was $p = 0.3$. We used a depth bound of 7 in our experiments, i.e., the search for explanations could not include more than 7 recursive refinements.

We compared the mean performances of DHAgent and Non-DHAgent using a two-tailed t -test with paired samples, which showed that DHAgent statistically outperformed Non-DHAgent in both domains. As the only difference between the two agents is the use of explanation, it’s clear that the use of DISCOVERHISTORY improved perfor-

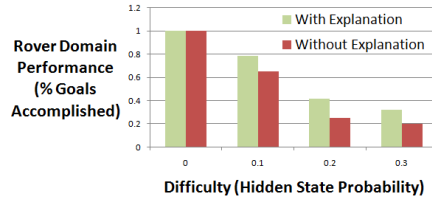


Fig. 3. Comparison at various difficulty levels.

mance. This shows that abductive explanation of state events can improve performance over replanning alone in partially-observable dynamic environments.

To further examine the impact of hidden state on performance, we increased the difficulty of the RWC domain by varying the probability of hidden states. Figure 3 compares the performance of the two agents at 4 difficulty levels: $p = 0.0, 0.1, 0.2$, and 0.3 . At $p = 0.0$, there is no hidden state; as expected, we see perfect performance from both agents since no explanation is necessary. As the probability of obstacles rises, both agents perform more poorly, but DHAgent continued to statistically outperform Non-DHAgent. At $p = 0.3$, DHAgent accomplished goals 50% more often than Non-DHAgent. Differences between them were statistically significant for all $p > 0.0$.

5 Related Work

Planning and execution has been focus of several papers. A survey of existing replanning techniques is described in [10]. Continuous Planning and Execution Framework (CPEF) is introduced in [8]. CPEF assumes that plans are dynamic, that is, that they must be evolving in response to the changes in the environment. Over the years, there has been a large body of research on replanning and plan repair during execution in dynamic environments. These works focus only on execution-time failures as discrepancies; that is, an unexpected state observed during execution triggers a re-planning or plan repair process [13, 4, 1, 12, 11, 8, 7].

In all of the previous works mentioned above, a discrepancy is defined as based on causal links between the preconditions and effects of different actions in the plan. In particular, when the observed state of the world violates such causal-links in the plan, a discrepancy occurs and triggers the re-planning or plan-repair process. In contrast, our formalism and algorithms are designed to generate more expressive and informative explanations of the world, including causal-link failures as well as other changes that may occur due to exogenous events.

McIlraith [5] focuses on an extension to the situation calculus theory for generation of *explanatory diagnoses* for aberrant behaviors given an observation, an initial state and a history of actions. McIlraith’s work has been extended to consider actions that occur during the history as well as actions that do not perform as expected [3]. Unlike ours, their formalization does not differentiate between the concepts of action and event; they only consider a single observation, rather than an execution history. This paper describes the first application of such ideas in a planning and execution system.

6 Conclusions and Future Work

We have described a formalism and algorithm to abductively reason about unexpected event occurrences during planning and execution. The explanations generated by our algorithm can be used by a planning and execution system to proactively expand its knowledge of the exogenous events and hidden state in the world, and thereby improve the performance of its re-planning process. Our experiments in two planning domains showed that the percentage of goals achieved was significantly higher when using our abductive-explanation generation algorithms, compared to an identical system that did not use them. We have shown that this algorithm can improve performance in environments with repeated exposure to hidden events and discrepancies.

There are several tasks that we would like to accomplish in the future based on our results. First, we'll investigate the theoretical properties of abductive reasoning in planning and execution in general. Based on this theory, we intend to generalize our work to investigate explanatory diagnosis in planning with incomplete action and event models, and in temporal planning, where actions and events may have durative effects.

Acknowledgments. This research is partly supported by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

1. Ayan, F., Kuter, U., Yaman, F., Goldman, R.P.: HOTRiDE: Hierarchical Ordered Task Re-planning in Dynamic Environments. In: ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems (2007)
2. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (May 2004), <http://www.laas.fr/planning>
3. Iwan, G., Lakemeyer, G.: What observations really tell us. In: KI-03 (2003)
4. Kambhampati, S., Hendler, J.A.: A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55, 193–258 (1992)
5. McIlraith, S.A.: Explanatory diagnosis: Conjecturing actions to explain observations. In: KR-98 (1998)
6. McKee, M.: Mars rover may not escape sand trap for weeks (2009)
7. Myers, K.: Advisable planning systems. In: Tate, A. (ed.) *Advanced Planning Technology*. AAAI Press. (1996)
8. Myers, K.L.: A continuous planning and execution framework. *AI Magazine* (1999)
9. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. *JAIR* 20, 379–404 (Dec 2003), <http://www.jair.org/papers/paper1141.html>
10. Russell, S., Norvig, P.: *Artificial Intelligence, A Modern Approach* (Second Edition). Prentice-Hall, Upper Saddle River, NJ (2003)
11. Wang, X., Chien, S.: Replanning using hierarchical task network and operator-based planning. In: ECP (1997)
12. Warfield, I., Hogg, C., Lee-Urban, S., Munoz-Avila, H.: Adaptation of hierarchical task network plans. In: FLAIRS-2007 (2007)
13. Yoon, S., Fern, A., Givan, R.: FF-Replan: A baseline for probabilistic planning. In: ICAPS-07 (2007)

Scientific Explanation and Computation

James A. Overton

The University of Western Ontario, joverto2@uwo.ca

Abstract. In this paper I present a philosophical approach to scientific explanation designed to work well with information technologies. The starting-point is an analysis of explanations as answers to why-questions. I argue that scientific explanations of phenomena using models, an important class of explanations, can be implemented using functions written in a programming language. I discuss the benefits of making explanations tractable to information technologies, and conclude with an example showing how such functions can be embedded into larger systems of explanation.

One of the goals of science is explanation. Simply browse the latest issue of *Nature* or *Science* and you will find the words “explain”, “explanation”, the related word “cause” and various causal language, used to describe how things interact, how concepts are related, and how models are justified. Advances in information technology, which are revolutionizing modern science, have not removed this goal. But how should we understand scientific explanation in an era of computerized science?

In this paper I use insights from the philosophy of science to develop an approach to scientific explanation that is friendly to computerized science. I start with an analysis of explanations as answers to why-questions, and then consider the *types* of the questions and the answers. I argue that the *type signature* of an explanation parallels the type signature of a function, and that sometimes the function is the explanation. This is the case for a particularly important class of scientific explanations: explanations of phenomena by models. In short, for an important class of scientific explanations, the explanation can be implemented as a function in a programming language, where the type signature links the type of a why-question to the type of its answer. I conclude with some simple examples.

Before proceeding, I should clarify how this approach relates to others in the literature. Philosophers of science have developed numerous accounts of scientific explanation: deductive-nomological [12,11]; inductive-statistical [11]; statistical-relevance [21]; unification [10,14]; pragmatic [29]; equilibrium [28]; causal-mechanical [22,9]; mechanistic [16,2]; interventionist [31]; asymptotic [1]; model-based [3]; and more. These accounts address the question *what is a scientific explanation?* Almost without exception, they set aside questions of how explanations are generated and understood by human beings. A successful philosophical account helps us to distinguish genuine scientific explanations from illegitimate ones.

The artificial intelligence literature on explanation, with which I am less familiar, seems to be addressing the question *how does one explain?* The key problem seems to be generating acceptable explanations by searching a vast space of possible explanations [15, p.58]. In order to navigate the search space, researchers use abduction [13], case-based reasoning [15,24], defeasible logics [6], interactive dialogues [5], insights from human explanation [23], argumentation capabilities [18], and other techniques. Success often means building a system that can explain in some domain.

My account is firmly rooted in the philosophy of science tradition, but I am reaching for connections with artificial intelligence research. I am trying to say what scientific explanations are in such a way that they can be processed by machines. By starting with an analysis of why-questions I am seeking common-ground for a wide range of philosophical approaches. And by developing a typology of scientific explanations I am trying to fit insights from these diverse approaches into a more unified account.

My long-term interest is in helping human scientists to build better explanations. This is not the same as building automated systems that can explain. But there is an interesting intersection here, where computational tools are designed specifically to help human scientists: explanations implemented as functions in a computer programming language can be versioned, reused, and packaged as libraries; automated systems can quickly test proposed explanations against large datasets; automated systems can generate a dependency tree, revealing the full complexity of the explanation; and automated systems can more effectively search and analyze explanations than current publishing methods allow. I believe that biomedical ontologies are a success story from this intersection [27]. Although their projects are distinct, philosophy of science and artificial intelligence have much to offer each other.

1 Questions, Answers, and Explanations

What is a scientific explanation? While there are many competing accounts in the philosophy of science literature, a widely accepted criterion is that explanations are answers to why-questions. I start with a classic analysis of why-questions, then extend it to how-questions, and propose a working definition of “explanation”.

1.1 Why-Questions

Sylvain Bromberger offers a valuable analysis of the kind of why-question we are concerned with, worth quoting at length:

We will mean by a why-question a question that can be put in English in the form of an interrogative sentence of which the following is true: (1) the sentence begins with the word *why*; (2) the remainder of the sentence has the (surface) structure of an interrogative sentence designed to ask a

whether-question – i.e., a question whose right answer in English, if any, must be either “yes” or “no”; (3) the sentence contains no parenthetical verbs, in Urmson’s sense. A why-question put as an English sentence that satisfies (1), (2), and (3) will be said to be in *normal form*. By the *inner question* of a why-question we will mean the question alluded to in (2) above – i.e., the question reached by putting the why-question in normal form, then deleting the initial “why” and uttering the remaining string as a question. By the *presupposition* of a why-question we will mean that which one would be saying is the case if, upon being asked the inner question of the why-question through an affirmative interrogative sentence, one were to reply “yes,” or what one would be saying is the case if, upon being asked the inner question through a negative sentence, one were to reply “no.” [4, p.86]

The why-questions we are interested in have a nested structure: there is a presupposition about a state of affairs, an inner yes/no question about whether that state of affairs holds, and the outer why-question asking for some account of the state of affairs. Bromberger offers an example. The why-question is “Why does copper turn green when exposed to air?” The inner question is “Does copper turn green when exposed to air?” The presupposition is “Copper turns green when exposed to air.”

I accept Bromberger’s analysis of why-questions. Although it only covers a special case of why-questions, I think this is the core case when it comes to scientific explanations. In the rest of this discussion, all of the why-questions we consider will be of this sort. However I disagree with Bromberger’s further analysis of the correct answers to why-questions, which relies on Hempel and Oppenheim’s deductive-nomological account of scientific explanation [12,11].

I propose that a is the answer to a why-question q with presupposition b if and only if a explains b . So an explanation is not an answer to a why question, but rather the pair of the answer and the presupposition.

1.2 How-Questions

Some how-questions are requests for facts, e.g. “How tall is the flagpole?” Some how-questions are requests for explanations, e.g. “How does copper turn green when exposed to air?” The latter can easily be rephrased as “Why does copper turn green when exposed to air?” while the former cannot: “Why tall is the flagpole?” It is not clear, however, that the original how-question and the rephrased why-question are requests for the same answer.

I define a how-question in exact parallel with Bromberger’s definition of a why-question, substituting *how* for *why* in condition (1). By a why/how-question I mean a question that is either a why-question or a how-question.

1.3 Working Definition

With these considerations in mind, I propose the following working definition of “explanation”:

Definition 1. *An explanation is the pair of an explanans a and an explanandum b , such that there exists a why/how-question q with b as its presupposition, and a explains b .*

The terms “explanans” and “explanandum” are standard in philosophy of science, but for the current purposes “answer” and “presupposition” will suffice.

Although this definition is a simple one, and may not capture all the subtleties of the usage of the English word “explanation”, it is enough to get us some interesting results.

2 Kinds of Explanation

Definition 1 leads us to consider various kinds of questions, their presuppositions, their answers, and the resulting explanations.

2.1 Questions

A simple pendulum sits on a table. A student and his teacher enter into a dialogue. Consider these four questions that the student might ask in the course of the discussion:

1. Why is the arm of this pendulum 1 m long?
2. Why does this pendulum have a period of 2 s?
3. Why does any pendulum with an arm length of 1 m and acceleration 9.8 m/s^2 have a period of 2 s?
4. Why is the period of a pendulum $T = 2\pi\sqrt{\frac{l}{g}}$?

2.2 Presuppositions

Bromberger’s analysis tells us what the presuppositions of these four why-questions are. But we can ask the more general question: What are the presuppositions of these questions about?

1. a property of a particular pendulum
2. a property of a particular pendulum
3. a property of all pendulums with arm length of 1 m and acceleration 9.8 m/s^2
4. an equation describing all pendulums

Although this usage requires a philosophical defence that I cannot give here, I want to say that (3) is a *phenomenon*, i.e. a pattern among events, or among properties of processes or objects, such that a particular pendulum instantiates the phenomenon. I also want to say that (4) is a *model* of the phenomenon (3). There are many different kinds of phenomena and kinds of models, and phenomena and models can be more specific or more general by degrees.

2.3 Answers

Consider these four answers that the teacher in our scenario might give in the course of the discussion:

1. I made the arm 1 m long.
2. This pendulum has an arm length of 1 m, and an acceleration of 9.8 m/s^2 . Any pendulum with arm length of 1 m and acceleration of 9.8 m/s^2 has a period of 2 s.
3. The period of any pendulum is $T = 2\pi\sqrt{\frac{l}{g}}$.
4. We can derive the equation from $F = ma$ and the geometry of forces, along with the small angle approximation, as follows ... [I omit the textbook derivation for the sake of brevity.]

Even when we do not have an answer to a why/how-question, we can sometimes specify the type of answer we are looking for.

2.4 Explanations

The presuppositions and the answers involve different kinds. Here I summarize, using “ $\rightarrow$ ” to denote the *explains* relation: *a* explains *b*.

1. a history $\rightarrow$ a property of a particular
2. a phenomenon $\rightarrow$ a property of a particular
3. a model $\rightarrow$ a phenomenon
4. (a law, definitions, an approximation, ...) $\rightarrow$ a model

I call these four statements the “type signatures” of the four explanations. Type signatures help us distinguish between explanations with the same presupposition (i.e., explanandum) but different kinds of answer (i.e., different explanans type).

So we have four kinds of explanation, distinguished by the kinds of their explanans and explananda. We are now in a position to characterize these four kinds of explanation (in very rough terms):

1. design/causation
2. syllogism/instantiation
3. modelling
4. argumentation/justification

In other words, each of the four pairs of presupposition and answer stands in a different relationship, but all four relationships fall under the general label “explains”. Although the distinction is not always recognized, different accounts of scientific explanation in the philosophy of science literature are often addressing different kinds of explanation. Despite the way I have presented the examples, I believe that this typology is distinct from “levels of explanation” accounts in

cognitive science [26,17,19,7,8]. And I recognize that I am disputing the standard distinction between explanation and argumentation, such as [30, p.30].

I want to draw attention to the similarity between the type signatures for explanations and the type signatures for functions in the typed lambda calculus, or in a typed functional programming language such as Haskell.¹ Benjamin Pierce offers this definition of a type system in a programming language: “A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute” [20, p.1]. In our case, we are classifying explanations according to the kinds of their presuppositions and answers. Although not strictly syntactic, I believe that assessing these kinds is more tractable than alternative classifications of explanations. In the next section I try to complete the analogy by showing how types can help to block certain explanatory errors. Types also help to draw semantic connections between explanations.

3 Explanations of Phenomena

The February 3, 2011 issue of *Nature* includes a letter entitled “Alternative stable states explain unpredictable biological control of *Salvinia molesta* in Kakadu” [25]. Kakadu is a National Park in Australia. *Salvinia molesta* is a species of invasive plant. Invasions of the plant can usually be controlled by the salvinia weevil, *Cyrtobagous salviniae*, but in this case the biological control is not always effective. Why is the biological control of *Salvinia molesta* in Kakadu unpredictable? Because the ecology of the system has alternative stable states: “By fitting a process-based nonlinear model to thirteen-year data sets from four billabongs, here we show that incomplete control can be explained by alternative stable states – one state in which salvinia is suppressed and the other in which salvinia escapes weevil control” [25, p.86].

I take this to be a core case of scientific explanation. A phenomenon is recognized, scientists collect data about instances of the phenomenon, and try to account for the general pattern using a mathematical model. As in case (3) above, a model explains the phenomenon because it stands in the modelling relation to the phenomenon. Since this kind of explanation is so important in science, it is worth investigating further.

3.1 Example: Models of the Period of a Pendulum

Here is a simple example of a scientific explanation of a phenomenon with a model implemented in a computer programming language. I continue with the pendulum example, begun above, and use the Haskell programming language, a functional programming language with a strong static type system well suited to the current approach. I present a series of three implementations of the same mathematical model, in order to show how the code can express increasing degrees of explanatory depth and block certain errors.

¹ <http://www.haskell.org>

Mathematical Model In 1602 Galileo discovered that the period of a simple pendulum is independent of the amplitude of the swing or the mass of the bob. He saw that the period varies with the square root of the length of the arm, and later scientists including Huygens and Hooke realized that the period is also proportional to the force of gravitation. Huygens gave the formula for the period of the ideal pendulum as:

$$T = 2\pi\sqrt{\frac{l}{g}}$$

where l is the length of the arm and g is the gravitational acceleration. Note that this is an idealization, ignoring friction, air resistance, etc., and relying on the small angle approximation. But I take Huygens' formula to be a mathematical model for the period of a pendulum.

Model 1 We can write a simple program in the Haskell programming language to calculate Huygens' formula:

```
pendulum_period_1 :: (Float, Float) -> Float
pendulum_period_1 (x, y) = 2 * pi * sqrt(y/x)
```

The first line of code defines a function called `pendulum_period_1`, that takes a pair of `Floats` as input and returns a `Float` as output. A `Float` is a 32 bit floating point number – a standard computer representation of a real number. The second line defines the mathematical relationship: given a pair of numbers, divide one by the other, take the square root, and multiply by 2π . We run the code by calling the function on some values:

```
result_1 = pendulum_period_1(9.8, 1.0)
```

The result is “2.0070899”.

What does this code tell us? It states the mathematical relationship between three variables. But Huygens' formula gave us more than this, simply because of the convention of understanding T, l, g as a duration, a length, and an acceleration.

Model 2 We can encode these conventions in Haskell by declaring types:

```
type Acceleration = Float
type Length = Float
type Period = Float
```

Here we declare a type called `Acceleration` that is a floating point number, and likewise for `Length` and `Period`. We can define a new function in these terms:

```
pendulum_period_2 :: (Acceleration, Length) -> Period
pendulum_period_2 (acceleration, arm_length) = 2 * pi *
    sqrt(arm_length / acceleration)
```

The function maps the pair of an `Acceleration` and a `Length` to a `Period`. (In Haskell, type names are capitalized, while function and variable names start with a lower-case letter, so `Acceleration` is a type while `acceleration` is a variable.) The mathematical relationship is, of course, the same. A more elaborate system could ensure that we're entering the right kinds of units and block that kind of error: metres per second squared, and metres, respectively. We run the code in the same way as before:

```
result_2 = pendulum_period_2(9.8, 1.0)
```

The result is “2.0070899”.

This implementation is better than the previous one, not because it gets a better result (it does not), but because it is embedded in a larger system of types and functions. The type `Float` tells us very little, but the type `Acceleration` says more about what kinds of values are allowed, and how the period of a pendulum is related to other models in physics. The computer does not understand the meaning of `Acceleration`, but other functions that take `Acceleration` as an input or an output type will thereby stand in a relationship with `pendulum_period_2`. The network of relationships between functions and their types begins to form a web that carries valuable information – an insight more common in artificial intelligence than philosophy of science. This information can be used to block errors.

We can embed the model further by allowing more connections:

```
earth_gravity = 9.8
result_2B = pendulum_period_2(earth_gravity, 1.0)
```

The variable `earth_gravity` can be shared by other models, connecting them together.

Model 3 We can improve the embedding even further:

```
data PendulumSystem = PendulumSystem {
    acceleration :: Acceleration,
    arm_length   :: Length
}
```

This code defines a new complex type called a `PendulumSystem`. A pendulum system is something with an `acceleration` and an `arm_length`. We could add other properties: amplitude, bob mass, air resistance, friction, etc. The point is that we bundle the types together, connecting them in tighter relationships, and allow the same complex type to be reused in different situations.

Now we redefine the period function to use the new `PendulumSystem` type:

```
pendulum_period_3 :: PendulumSystem -> Period
pendulum_period_3 ps = 2 * pi * sqrt(arm_length(ps) / acceleration(ps))
```

Here we use the `arm_length()` and `acceleration()` functions to extract information from the `PendulumSystem` instance. To make a calculation we create an instance of a `PendulumSystem` called `ps_1` and apply the function to it:

```

ps_1 = PendulumSystem {
    acceleration = earth_gravity,
    arm_length = 1.0
}
result_3 = pendulum_period_3(ps_1)

```

The result is “2.0070899”.

Any of these three models can be given as an answer to question (3) above about the periods of pendulums, and then question (2) about the period of a particular pendulum. Although they can answer the same question, with the same presupposition, each provides a different answer, and thus a different explanation. Each explanation is deeper than the last, in the sense that it is embedded more deeply into a system of other types and functions that can be used to model different phenomena. The system of types also helps prevent explanatory errors, such as unit errors and mistaking the type of the answer.

4 Conclusion

The philosophical account of scientific explanation presented here shows how explanations of phenomena by models, an important class of scientific explanations, can be implemented using computer programs. I have drawn on programming techniques familiar to artificial intelligence researchers in order to address the philosophical question *what is a scientific explanation?* I have not directly addressed the question *how does one explain?* But by exploring the intersection of these fields, I hope we are a small step closer to having scientific explanations managed using the information technologies that are so rapidly changing the collection, curation, publication, dissemination, and analysis of scientific data.

References

1. Batterman, R.W.: The Devil in the Details: Asymptotic Reasoning in Explanation, Reduction, and Emergence. Oxford University Press, Oxford (2002)
2. Bechtel, W., Abrahamsen, A.: Explanation: A mechanist alternative. *Studies in History and Philosophy of Biological and Biomedical Sciences* 36, 421–441 (2005)
3. Bokulich, A.: How scientific models can explain. *Synthese* 180(1), 33–45 (2009)
4. Bromberger, S.: Why-questions. In: Colodny, R.G. (ed.) *Mind and Cosmos: Essays in Contemporary Science and Philosophy*, pp. 68–111. Pittsburgh University Press, Pittsburgh (1966)
5. Cawsey, A.: Explanation and interaction: The computer generation of explanatory dialogues. The MIT Press, Cambridge, MA (1992)
6. Chesñevar, C.I., Maguitman, A.G., Loui, R.P.: Logical models of argument. *ACM Computing Surveys* 32(4), 337–383 (December 2000)
7. Dawson, M.R.: Understanding Cognitive Science. Blackwell, Oxford (1998)
8. Dennett, D.C.: The Intentional Stance. Bradford Books, Cambridge, MA (1987)
9. Dowe, P.: Physical causation. Cambridge University Press, Cambridge (2000)
10. Friedman, M.: Explanation and scientific understanding. *The Journal of Philosophy* 71(1), 5–19 (1974)

11. Hempel, C.G.: Aspects of scientific explanation. In: *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science*, pp. 331–496. Free Press, New York (1965)
12. Hempel, C.G., Oppenheim, P.: Studies in the Logic of Explanation. *Philosophy of Science* 15(2), 135–175 (1948)
13. Josephson, J.R., Josephson, S.G.: *Abductive inference: Computation, philosophy, technology*. Cambridge University Press, Cambridge (1996)
14. Kitcher, P.: Explanatory unification and the causal structure of the world. In: Kitcher, P., Salmon, W.C. (eds.) *Scientific Explanation*, pp. 410–505. Minnesota Studies in the Philosophy of Science, University of Minnesota Press, Minneapolis (1989)
15. Leake, D.B.: *Evaluating Explanations: A Content Theory*. Lawrence Erlbaum Associates, Hillsdale, NJ (1992)
16. Machamer, P.K., Darden, L., Craver, C.F.: Thinking About Mechanisms. *Philosophy of Science* 67(1), 1–25 (2000)
17. Marr, D.: *Vision: A computational investigation into the human representation and processing of visual information*. W.H. Freeman, San Francisco (1982)
18. Moulin, B., Irandoust, H., Bélanger, M., Desbordes, G.: Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review* 17(3), 169–222 (2002)
19. Newell, A.: The Knowledge Level. *Artificial Intelligence* 18, 87–127 (1982)
20. Pierce, B.C.: *Types and programming languages*. MIT Press, Cambridge, MA (2002)
21. Salmon, W.C.: Statistical explanation. In: Salmon, W.C. (ed.) *Statistical Explanation and Statistical Relevance*, pp. 29–87. University of Pittsburgh Press, Pittsburgh (1971)
22. Salmon, W.C.: *Scientific explanation and the causal structure of the world*. Princeton University Press, Princeton (1984)
23. Schank, R.C.: *Explanation patterns: Understanding mechanically and creatively*. Lawrence Erlbaum Associates, Hillsdale, NJ (1986)
24. Schank, R.C., Kass, A., Riesbeck, C.K.: *Inside case-based explanation*. Lawrence Erlbaum Associates, Hillsdale, NJ (1994)
25. Schooler, S., Salau, B., Julien, M., Ives, A.: Alternative stable states explain unpredictable biological control of *Salvinia molesta* in Kakadu. *Nature* 470(7332), 86–89 (2011)
26. Simon, H.A.: The architecture of complexity. *Proceedings of the American Philosophical Society* 106(6), 467–482 (1962)
27. Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L.J., Eilbeck, K., Ireland, A., Mungall, C.J., the OBI Consortium, Leontis, N., Rocca-Serra, P., Ruttenberg, A., Sansone, S.A., Scheuermann, R.H., Shah, N., Whetzel, P.L., Lewis, S.: The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology* 25(11), 1251–1255 (2007)
28. Sober, E.: Equilibrium explanation. *Philosophical Studies* 43, 201–210 (1983)
29. van Fraassen, B.C.: *The Scientific Image*. Clarendon Press, Oxford (1980)
30. Walton, D.: *Argument Structure: A Pragmatic Theory*. University of Toronto Press, Toronto (1996)
31. Woodward, J.: *Making things happen: A theory of causal explanation*. Oxford University Press, Oxford (2003)

Monitoring and explaining reasoning processes in a dialogue system’s input interpretation step

Daniel Sonntag and Christian Schulz

German Research Center for AI (DFKI),
Stuhlsatzenhausweg 3, 66123 Saarbruecken, Germany
`sonntag@dfki.de`
`christian.schulz@dfki.de`

Abstract. We implemented a generic speech-based dialogue shell that can be configured for and applied to domain-specific dialogue applications. A toolbox for ontology-based dialogue engineering provides a technical solution for the two challenges of engineering domain extensions for new question and answer possibilities and debugging functional modules. In this paper, we address the process of debugging and maintaining rule-based input interpretation modules. While supporting a rapid implementation cycle until the dialogue systems works robustly for a new domain (e.g., the dialogue-based retrieval of medical images), production rules for input interpretation have to be monitored, configured, and maintained. We implemented a special graphical user interface to monitor and explain reasoning processes for the input interpretation phase of multimodal dialogue systems. A particular challenge was the presentation of the software system’s ontology-based interaction rules in a way that they were accessible to and editable for humans for maintenance, and, at the same time, allowed a real-time monitoring of their application in the running dialogue system.

1 Introduction

Dialogue systems combine many individual natural language processing components into a single, complex, artificial intelligence system. Because of the complexity, these systems cannot easily be constructed and maintained. Over the last several years, we therefore build up a new discourse and dialogue infrastructure to ease the task of dialogue construction for a specific domain thereby allowing for a rapid dialogue system engineering process. A dialogue runtime environment, the ontology-based dialogue platform (ODP) framework and its platform API (the DFKI spin-off company SemVox, see www.semvox.de, offers a commercial version), translates between the speech user input/output and conventional query/answer data structures in ontology-based representations (in the case of, e.g., a SPARQL backend repository) [16]. In order to create the needed interaction rules in ontological form for processing user requests, we implemented an integrated toolbox which builds upon the industry standard Eclipse and also integrates other established open source software development tools to support

dialogue application development, automated testing, and interactive debugging. The main challenges we encountered in supporting a rapid dialogue system engineering process, i.e., implementing a new dialogue for a new domain, can be summarised as follows:

- Engineering ontological domain extensions;
- Debugging functional modules such as natural language understanding (NLU), input fusion, dialogue management, and external text-to-speech (TTS) synthesis.

In this paper, we discuss the input fusion step in particular and show how the production rules that are used for input interpretation can be monitored, configured, and maintained. We basically provide two new contributions. First, we discuss a graphical user interface to monitor and explain reasoning processes for the input interpretation phase of multimodal dialogue systems. Second, we show the usefulness of the interface by a demonstration of its usage in a particular real-world industrial application example, the dialogue-based access to medical image data.

2 Background and Related Work

We use a distributed, ontology-based dialogue system architecture, where every major component can be run on a different host, increasing the scalability of the overall system. In earlier projects [20,10] we integrated different sub-components to multimodal interaction systems which we then extended to the consistent usage of ontology data structures [5]. Thereby, the dialogue system also acts as the middleware between the clients and the backend services that hide complexity from the user by presenting aggregated ontological data. Prominent examples of integration platforms include OOA [7], TRIPS [1], and Galaxy Communicator [12]; the W3C consortium also proposes inter-module communication standards like the Voice Extensible Markup Language VoiceXML¹ or the Extensible MultiModal Annotation markup language EMMA², with products from industry supporting these standards³. We will use an EMMA-related XML format we called PreML which is also used throughout the examples in [14]. A comprehensive overview of ontology-based dialogue processing, which is the basis of our integration platform, can be found in [13], pp.71-131. Figure 1 shows the ontology components the user works with. The graphical user interfaces (GUIs) for editing ontologies, speech recognition grammars, and interaction rules are implemented as Eclipse plugins in the ODP workbench (see screenshots in [17]) using the open source toolkit JUnit. The results of the interactive processes where the dialogue engineers are involved, are stored in RDF repositories as ontology data, domain-dependent specifications, and ontology-based rules sets for interaction and input interpretation rules of the specific application domain.

¹ <http://www.w3.org/TR/voicexml20/>

² <http://www.w3.org/TR/emma/>

³ <http://www.voicexml.org>

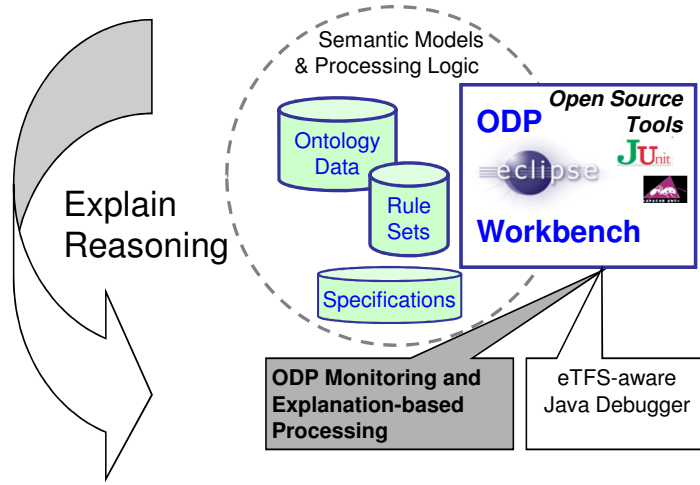


Fig. 1. Ontology components the user works with for explaining reasoning processes.

In [18], we tackled the challenge of how an application backend can automatically infer previously unknown knowledge (facts) and provide explanations for the inference steps involved. (Proper explanations are a main factor for increasing the level of user trust in end-to-end human-computer interaction systems.) In the medical application domain [16], we use a medical backend database system that stores RDF image meta data, e.g., patient names, dates of issue, and the like. The dominant query language for these RDF⁴ repositories is the W3C recommendation SPARQL⁵. Examples of how SPARQL can be used in the context of integrating Linked Data for semantic dialogue and backend access can be found in [15]. A Java debugger has also been implemented; under the hood, it uses an application programming interface for the efficient representation of ontology-based data using extended typed feature structures (eTFS). As described in [11], the eTFS API is tightly integrated into a production rule system which enables a declarative specification of the processing logic in terms of production rules. This is the precondition for the ODP monitoring and explanation capabilities for internal, component-based, reasoning steps as described in the rest of the paper. Figure 2 provides rough sketch of the basic processing chain within the typical interaction cycle. The dialogue manager uses reasoning processes at each of the prominent processing stages, namely input interpretation, dialogue interaction, data requesting, the provision of answers in the application backend, the retrieval of data, and its presentation (which normally incorporates a NLU generation step [4]). In the context of explanations and production rule

⁴ See <http://www.w3.org/TR/rdf-primer/> and <http://www.w3.org/TR/rdf-schema/>.

⁵ See <http://www.w3.org/TR/rdf-sparql-query/>.

reasoning for dialogue systems, in this contribution, we will focus on the input interpretation step which is highlighted in figure 2.

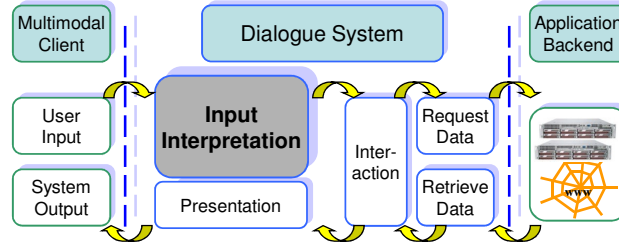


Fig. 2. Typical interaction cycle.

3 Explanation-based Interface for Production Rules

In this section, we explain the technical background of a GUI to monitor and explain reasoning processes for the input interpretation phase of multimodal dialogue systems. Essentially, the explanation-based GUI helps to deal with complex rules in a production rule system.

The production rule system provides rule-based fusion of different input modalities such as text, speech, and pointing gestures, and allows us to solve ambiguities in co-references. For example, when the user opens the patient records of several persons and utters “also the latest CT images” later in the dialogue, the production rule system is used to reason about the correct patient instance the user is referring to. We use the production-rules-based fusion and discourse engine which follows the implementation in [8].

Within the dialogue infrastructure, this component plays a major role since it provides basic and configurable dialogue processing capabilities. More processing robustness is achieved through the application of a special robust parsing feature in the context of RDF graphs as a result of the input parsing process and through the possibility to debug, edit, and maintain the production rules in a specific GUI. In addition, the semantic relationships between several catchwords from the spoken user input can be guessed (following [19]) according to the ontological domain model of the industrial (medical) application domain. For example, the query “any experts available?” searches for other radiologists with a specific portfolio that matches with the semantic disease annotation on the radiology images.

4 An Application Scenario for Explanations

In this section, we will describe an application scenario where the dialogue engineer can use our GUI to edit input fusion rules in a medical application scenario. The rules can be inspected and maintained because the engineer gets many explanations about how new rules behave in the *running* dialogue system. We focus on the aspect that the presentation of the software system’s ontology-based interaction rules is in a way that they are accessible to humans for maintenance, and, at the same time, allow a real-time monitoring of their application in the running dialogue system.

4.1 Dialogue Example

Consider a radiologist who treats a patient which suffers from a specific type of cancer, e.g., lymphoma. The treatment of such severe diseases requires a lot of follow-up examinations where previous diagnoses have to be taken into account. The dialogue engineer has to provide the domain user, i.e., a radiologist in the medical domain, with the desired dialogue competence (also cf. the THESEUS MEDICO use case) to access the desired patient records, the corresponding image data, and comparative cases. In addition, the doctor should be able to complete a new finding while interacting with a multimodal dialogue system.

The following medical dialogue illustrates the doctor’s practical interest in using a dialogue interface on top of an ontology-based search database for medical images. The dialogue concentrates around the questions of this user (U) about the media contents, i.e., the body parts and the anatomy shown in computer tomography (CT) picture series (also see DICOM, medical.nema.org) and magnetic resonance (MR) videos, and the the answers of the system (S).

Figure 3 shows the medical dialogue example and multiple CT image results which are displayed on an iPad. The user can freely speak to the iPad which is used as the interaction device in the distributed dialogue system architecture (also cf. <http://www.dfki.de/RadSpeech>).

4.2 Exploring Multiple Applicable Rules

We chose the OWL formalism recommended by W3C and identified a subset that suits the underlying eTFS representation for the specification of the input fusion ontology. The eTFS format unifies the properties of RDF/RDFS [6] and typed feature structures [3].

For the runtime knowledge processing step, we have implemented an eTFS API that provides useful methods to compare two types and access related concepts within the inheritance hierarchy quickly.

The API is essential for the reasoning engine of the ontology-based production rule system (PATE) [9]. PATE’s architecture is centered around the idea of three separated data storage facilities: (i) the goal stack, (ii) the working memory, and (iii) the long-term memory. The working memory is responsible for the activated instances, the so-called working memory elements (WMEs), which are

- 1 **U**: "Show me the patient file, last examination, Gerda Meier."
- 2 **U**: "Also the images, ... CT"
- 3 **S**: "Shows the patient file and corresponding patient CT studies in DICOM pictures."
- 4 **U**: "Show me the internal organs: lungs, liver, then spleen and colon."
- 5 **S**: Shows corresponding patient image data according to referral record.
- 6 **U**: "Summarise the patient's findings."
- 7 **S**: Synthesises a summary of the patient's findings.

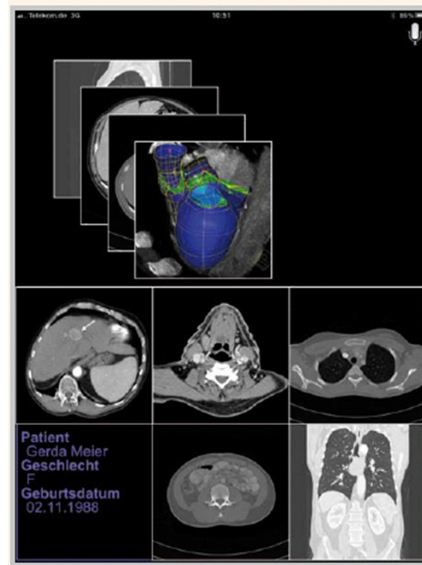


Fig. 3. Medical dialogue and system results on an iPad.

accessible for applying production rules. The long-term memory is responsible for the persistent storage for all instances of the type hierarchy the system has in the background. The purpose of the goal stack is to mimic the attentional focus during the dialogue process that is inspired by the cognitive models introduced in [2]. On which of the three stacks mentioned above the current WMEs are put, is determined by an activation value (which can also change after a rule has fired). In every processing state, there is always one single WME in focus (which has an impact on the rules accessible for firing). Only rules whose goal condition matches the pattern of the WME on top of the goalstack have a chance to fire. Without such editor functionalities, dialogue engineers would have to edit complex OWL files manually which is impractical.

Rule Inspection The background knowledge of the dialogue system determines the relations of different concepts within the hierarchy. Our ontology says that the main condition in the rule represented by the concept *comet#QueryTask* subsumes the concept *medico#RetrievePatientImages*, which represents the ontological concept for the task in the second turn of the dialogue example in 3. If the dialog state has the attentional focus on an that eTFS structure, then the condition is sufficient to cause the rule to fire, because the unification would succeed in this constellation.

Further refinements of rule firing is provided by the possibility to involve the unification of side conditions in the rule with structures located inside the

working memory. This mechanism is especially beneficial in the context of tuning and debugging rules at runtime, which we will illustrate in more detail.

We implemented a *rule inspector*, a tool that provides debugging options along the process pipeline from the interpretation of an utterance to the adaption of ontological representations of commands, to actual retrieval actions in the backend, to the reallocation of the retrieved results in the presentation step. The main view of the rule inspector GUI (figure 4) shows the state of the production rule system after the calculation of the firing rule before applying the actions coded in the head of the production rule (action part). The two lists, the working memory contents and the goal stack that correspond to the *working memory* box and the *goal stack* box of the upper part in figure 4, respectively, show the rule’s basic type and activation score. Additionally, a syntax-highlighted XML viewer allows us to inspect the WMEs during the dialogue process by clicking on the entries in the list, see the *wme inspector* in 4. The same debugging support is provided in the lower part of the GUI, where matching rules indicated in the *conflict set*’ box, once selected, are displayed in an XML format together inside the *rule inspector* box with their probabilities to fire successfully. Rule firing can be invoked manually by the dialogue engineer while clicking on the *Step!* button.

New Rule Creation and Debugging Moreover, rule inspection is supported for all (other) production rules assigned to a processing module that are listed in the *Rule base* box in figure 4. This is crucial when the dialogue engineer adds a new rule to the already existing collection of rules. In this specific scenario, the engineer expects to enrich the existing interaction possibilities covered by the dialogue application.

However, often the expected behavior at a target state along the dialogue process fails to appear. At this point, the dialogue engineer can invoke in the rule inspector to get support in the process of finding out why a certain rule, here the newly created rule, does not fire.

The rule inspector performs the reasoning step to deduce and display the conflicting part of the new rule that does not semantically fit to the set of valid rules with regard to the current state of dialogue process.

Figure 4 captures the state where the dialogue process step shifts to ‘request data’ in the backend system after successful interpretation. This snapshot shows the “pause” in the processing stage after the utterance “Also the images” (cf. the example dialogue in figure 3) when the radiologist requests the patient’s image data. The modality fusion component keeps track of the ongoing discourse context and resolves the correct patient reference ‘Gerda Meier’ shown in the *WME inspector* box of figure 4. The ontological concept *medico#RetrievePatientImages* represents the system task to be executed on the backend service, this means, the query instance will be transformed into the corresponding SPARQL query. In this particular case, however, the rule which would trigger the request to the desired backend service, is not an option for the rule engine because the conditions do not match. Here, the developer has the opportunity to navigate and select the desired rule and receive the information where unification fails, indicated in the

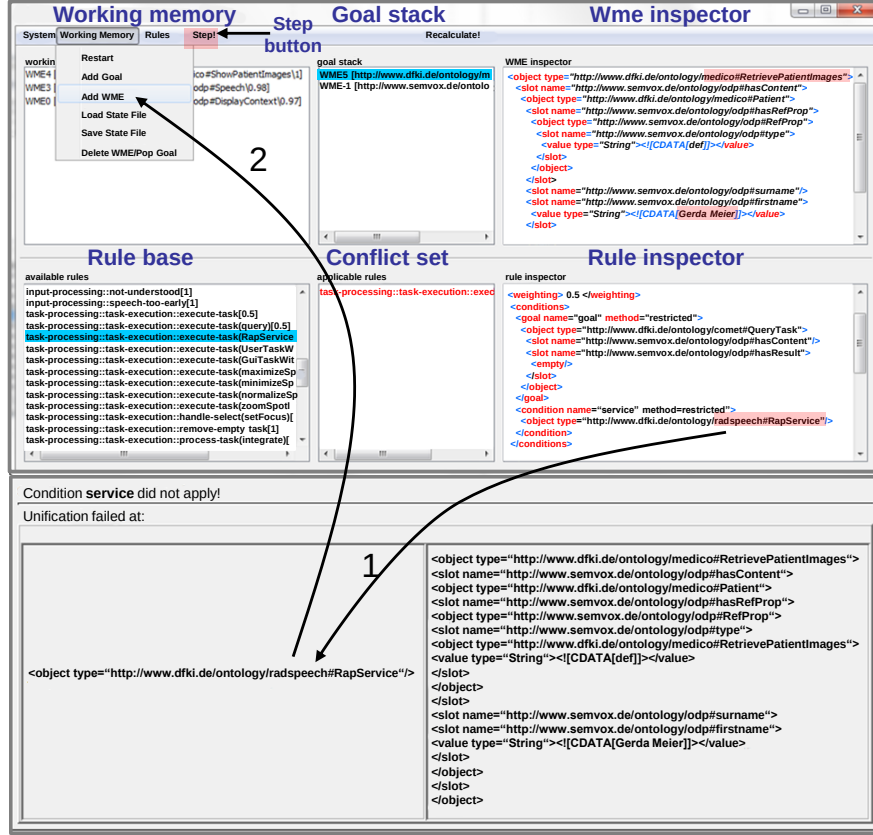


Fig. 4. Display, explanation, and editing in case of a unification conflict.

lower box in figure 4. Therein the dialogue engineer may consult the rule inspector for individual parts of the eTFS structures that collide. In particular, the newly added rule requires an instance of the concept *radspeech#RapService* on the working memory in order to match the current configuration of the dialogue state, indicated by the arrow (1). The GUI support of detecting the conflicting parts during unification provides the necessary explanations to the developer about the changes in the dialogue state and the actions he or she can perform (at runtime!) in order to cause the newly added rule to fire.

In addition to the functionality to give advice to the developer at runtime, the rule inspector offers the opportunity to add the required instance on the working memory manually. Thus, by adding the *radspeech#RapService* instance to the working memory, see arrow (2), the debugging process will be resumed towards the newly created and correct interaction possibility. Previously, the

new backend for the query “also the images” could not be accessed because the rule for the *RapService* did not fire. Instead, we inspected the more general rule (`executeTask(query)`) to fire (with the help of the GUI) and successfully adapted the conditions to rerank the firing rules so that the desired rule is included and the *RapService* could be accessed in this dialogue state. In the course of adding this new behaviour to an already existing dialogue application, the explanation possibilities of the inspection editor offer powerful support to view low-level data, and edit/reason about new rules which can be done at runtime with the help of only one graphical user interface tool.

5 Conclusion and Future Work

Based on an integration platform for off-the-shelf dialogue solutions and internal dialogue modules (ODP platform), we described the parts of the discourse and dialogue infrastructure that allow for a explaining reasoning processes in the (ontology-based) input interpretation step. We focussed on a medical application scenario where we demonstrated how to maintain rules, explore multiple applicable rules, inspect rules, and create new rules that can be reasoned about and therefore explained at runtime. In this way, we provided a solution to the particular challenge to present ontology-based interaction rules in a way that they are accessible to and editable for humans for maintenance, and, at the same time, allow a real-time monitoring of their application in the running dialogue system. In future work, we plan to include reasoning processes that suggest rules to the user in addition to explaining why user-generated rules do not behave correctly.

Acknowledgements Thanks go out to Robert Nesselrath, Yajing Zang, Markus Löckelt, Matthieu Deru, Simon Bergweiler, Alassane Ndiaye, Norbert Pfleger, Alexander Pfalzgraf, Jan Schehl, Jochen Steigner and Colette Weihrauch for the implementation and evaluation of the dialogue infrastructure. This research has been supported by the THESEUS Programme funded by the German Federal Ministry of Economics and Technology (01MQ07016).

References

1. Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A.: An Architecture for a Generic Dialogue Shell. *Natural Language Engineering* 6(3), 1–16 (2000)
2. Anderson, J.R., Lebiere, C.J. (eds.): *The Atomic Components of Thought*. Lawrence Erlbaum Associates, Mahwah, NJ (1998)
3. Carpenter, B.: *The Logic of Typed Feature Structures*. No. 32 in *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, Cambridge, UK (1992)
4. Engel, R.: SPIN: A Semantic Parser for Spoken Dialog Systems. In: *Proceedings of the 5th Slovenian and First International Language Technology Conference (IS-LTC 2006)* (2006)

5. Fensel, D., Hendler, J.A., Lieberman, H., Wahlster, W. (eds.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press (2003)
6. Manola, F., Miller, E.: *RDF primer*. W3C recommendation, W3C (February 2004), published online on February 10th, 2004 at <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>
7. Martin, D., Cheyer, A., Moran, D.: The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence* 13(1/2), 91–128 (1999), citeseer.ist.psu.edu/martin99open.html
8. Pflieger, N.: FADE - An Integrated Approach to Multimodal Fusion and Discourse Processing. In: *Proceedings of the Doctoral Spotlight at ICMI 2005*. Trento, Italy (2005)
9. Pflieger, N., Schehl, J.: Development of advanced dialog systems with PATE. In: *Proceedings of Interspeech 2006—ICSLP: 9th International Conference on Spoken Language Processing*, Pittsburgh, PA, USA. pp. 1778–1781 (2006), http://www.isca-speech.org/archive/interspeech_2006/i06_1598.html
10. Reithinger, N., Fedeler, D., Kumar, A., Lauer, C., Pecourt, E., Romary, L.: MI-AMM - A Multimodal Dialogue System Using Haptics. In: van Kuppevelt, J., Dybkjaer, L., Bernsen, N.O. (eds.) *Advances in Natural Multimodal Dialogue Systems*. Springer (2005)
11. Schehl, J., Pfalzgraf, A., Pflieger, N., Steigner, J.: The BabbleTunes System. Talk to Your iPod! In: *Proceedings of the 10th International Conference on Multimodal Interfaces (ICMI)* (2008)
12. Seneff, S., Lau, R., Polifroni, J.: Organization, Communication, and Control in the Galaxy-II Conversational System. In: *Proceedings of Eurospeech'99*. pp. 1271–1274. Budapest, Hungary (1999)
13. Sonntag, D.: *Ontologies and Adaptivity in Dialogue for Question Answering*. AKA and IOS Press, Heidelberg (2010)
14. Sonntag, D., Deru, M., Bergweiler, S.: Design and Implementation of Combined Mobile and Touchscreen-Based Multimodal Web 3.0 Interfaces. In: *Proceedings of the International Conference on Artificial Intelligence (ICAI)*. pp. 974–979 (2009)
15. Sonntag, D., Kiesel, M.: Linked data integration for semantic dialogue and backend access. In: *AAAI Spring Symposium on Linked Data Meets Artificial Intelligence* (2010)
16. Sonntag, D., Möller, M.: Unifying semantic annotation and querying in biomedical image repositories. In: *Proceedings of International Conference on Knowledge Management and Information Sharing (KMIS)* (2009)
17. Sonntag, D., Sonnenberg, G., Nesselrath, R., Herzog, G.: Supporting a rapid dialogue engineering process. In: *Proceedings of the First International Workshop On Spoken Dialogue Systems Technology (IWSDS)* (2009)
18. Sonntag, D., Theobald, M.: Explanations in dialogue systems through uncertain RDF knowledge bases. In: Roth-Berghofer, T., Tintarev, N., Leake, D.B., Bahls, D. (eds.) *Proceedings of the Fifth International Workshop on Explanation-aware Computing (ExaCt 2010)*. CEUR Workshop Proceedings, vol. 650, pp. 1–12. CEUR-WS.org, Lisbon, Portugal (2010)
19. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In: *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*. pp. 405–416. IEEE Computer Society, Washington, DC, USA (2009)
20. Wahlster, W.: SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell. In: Krah, R., Günther, D. (eds.) *Proceedings of the Human Computer Interaction Status Conference 2003*. pp. 47–62. DLR, Berlin, Germany (2003)

Improving User and Decision Support System Teamwork

An Approach Based on Shared Mental Models

Iris van de Kieft, Catholijn M. Jonker, and M. Birna van Riemsdijk

Man Machine Interaction Group, Delft University of Technology, The Netherlands

Abstract. Social psychology research on teams has shown that team performance improves when team members have a so-called shared mental model (SMM). We observe that a decision support system (DSS) and its user can be viewed as a team whose task is to make a (good) decision. Therefore we argue that it is important for good user-DSS team performance that they have a SMM. In this paper we make precise what it means for a user and DSS to have a SMM. This can be used as the basis for analyzing to what extent a DSS supports the achievement of a SMM, or to design a DSS such that it does this. We apply our framework to a particular negotiation support system, using it to analyze to what extent this DSS facilitates the achievement of a SMM. We propose that the addition of explanation facilities to the DSS can improve sharedness in the user-DSS team.

1 Introduction

Decision support systems (DSSs) are interactive computer systems that aid users in judgement and choice activities [5]. The user and the DSS cooperate, to structure the decision problem and make a choice. We observe that the user and the DSS can be viewed as a *team* whose goal is to make a (good) decision.

It is known from social psychology research that team performance improves if team members have a shared understanding of the task that is to be performed and of the involved teamwork, i.e., if team members have a so-called *shared mental model* (SMM). SMMs are defined in [3] as:

knowledge structures held by members of a team that enable them to form accurate explanations and expectations for the task, and, in turn, coordinate their actions and adapt their behavior to demands of the task and other team members.

We argue that research on SMMs is not only relevant for human teams, but also for human-DSS teams. In other words, we expect that improving user-DSS teams' SMM will improve their team performance. User-DSS teams form a specific kind of team. Most notably, a user-DSS team consists of a human team member and a software agent, while human-only teams have been the focus of SMM research. This also results in each team member having its own, complementary capabilities. For example, compared to the DSS, the user possesses more general world knowledge, whereas, compared to the user, the DSS has superior computational abilities. Moreover, it is a hierarchical team

in which the DSS supports the user. The user makes the final decision, for which she is responsible¹.

In this paper we make precise what it means for a user and DSS to have a SMM. This can be used as the basis for analyzing to what extent a DSS supports the achievement of a SMM, or to design a DSS such that it does this. To be more specific, we analyze the user-DSS task and team, based on existing categories of mental models from SMM theory [3] (Section 2). This allows us to determine what knowledge is relevant for this task and team. We use this analysis to make precise what a SMM is in the context of user-DSS teams, based on the identification of two types of sharedness that we consider relevant in user-DSS teams (Section 3). We discuss how sharedness can be achieved and maintained in Section 4. In Section 5 we apply our framework to a particular negotiation support system by using it to analyze to what extent this DSS facilitates the achievement of a SMM. We propose that the addition of explanation facilities to the DSS can improve sharedness in the user-DSS team, where our framework helps determine the requirements for such explanations. We conclude the paper in Section 6.

2 Analysis of User-DSS Task and Team

In this section we analyse user-DSS teamwork, based on the mental model categories defined in [3]: task, equipment, team members² and team interaction. For each mental model, we discuss its contents, given the user-DSS teamwork context.

2.1 Task

The knowledge content of the task model concerns task procedures, strategies, likely contingencies, and environmental conditions [3]. In user-DSS teamwork, the task is *decision making*, and thus involves decision making knowledge. H.A. Simon identified three phases in the decision making process: intelligence, design and choice [16]. In the intelligence phase, the problem or opportunity is identified. In this phase, the decision problem and its domain are defined.

In the design phase, a decision model is constructed. This involves identifying the important criteria for the decision process and for the decision result, and constructing the available options. A decision model and its variables represent preferences over decision criteria, available options and a measure of uncertainty over the variables influencing decision process and result [5]. We note that the preferences in the decision model are often those of the user, as she is the decision maker. However, the preferences may also be determined by external parties. For example, in managerial decisions, the preferences may be determined by the user's superiors.

In the choice phase, the available options are evaluated, taking into account preferences over decision criteria and measures of uncertainty, after which the best decision option is selected. In this phase a strategy or choice method is needed, to make

¹ Here, we assume the user always remains in control of the final decision. We do not discuss cases of adjustable autonomy [15].

² We refer to the team mental model as the team member mental model, to avoid confusion with the concept of team mental model as used in, for example: [12].

a choice amongst the available options. Thus, the knowledge about ‘task procedures’ and ‘strategies’ consists of the above described decision making knowledge. ‘Likely contingencies’ are taken into account by the measures of uncertainty mentioned above. Furthermore, we take ‘environmental conditions’ to mean the relevant knowledge about the current state of the environment, as well as relevant previous states.

2.2 Equipment

The knowledge content of this model concerns equipment functioning, operating procedures, system limitations, and likely failures [3]. For any equipment that the user-DSS team makes use of, these four knowledge elements are needed. Although the DSS can be viewed as technology or equipment, we view it as a team member, and thus do not consider knowledge about the DSS to be part of this category.

2.3 Team Members

The knowledge content of this mental model concerns knowledge, skills, strengths, limitations, preferences and tendencies of teammates [3]. We combine the ‘skills’ and ‘strengths’ elements, as for our purposes, these are quite similar. Additionally, for now we leave out the ‘tendencies’, as this is a more subtle aspect that is relevant for human-human interaction, whereas for human-agent interaction it is less clear how this applies. In the following we discuss these knowledge content elements in the context of user-DSS teams.

Knowledge: The user and the DSS typically possess partially distinct knowledge. For example, the user has a wealth of knowledge about world, about interacting with other humans and about her own preferences. The user likely possesses certain decision making knowledge such as knowledge about the domain and the available options. The DSS has specific decision making knowledge about how to structure the decision problem, and has one or more choice methods.

Skills and strengths: The user and the DSS have different, complementary skills. For example, the user can interact easily with the world, and gather external knowledge about current state of the environment. The user has emotions that in some cases can help decision making [4]. The DSS is a specialist in the decision making task it was built for. It has good memory, superior computational abilities and remains rational.

Limitations: The user need not be a specialist in the task, has emotions and thus may not always be rational, has limited (working) memory and limited computational power. The reasoning of the DSS is limited to the task it is built for. It may not be able to adapt to new domains or contingencies. The user and the DSS thus, to a large extent, have complementary skills and limitations.

Preferences: The user has preferences that may be key to the decision making process. In this case, they should be part of the task mental model. The user may also have other preferences that relate more to the interaction style she prefers, in other words, preferences related to team interaction, not task. These should then be part of the team member mental model. The DSS does not have preferences.

2.4 Team interaction

The knowledge content of this model concerns roles/responsibilities, communication channels, interaction patterns, information flow, role interdependencies and information sources [3]. For each element of the knowledge content, we discuss what its contents are in the context of user-DSS teams.

Roles/responsibilities: Based on their complementary capabilities, each team member has a specific role and corresponding responsibilities. The role of the user is to provide the DSS with the required information, and other tasks such as interaction with other humans. The user is responsible for making the final decision and for its consequences. The role of the DSS is to assist in structuring the decision, provide choice advice, request necessary information from the user and help the user understand the process and construct domain and preferences.

Communication channel: An important characteristic of user-DSS teams is that the interaction takes place via a user interface. Thus, the set of interactions is limited to those that are allowed by the user interface. This sets this type of teamwork apart from human teamwork, where interactions are not necessarily as constrained.

Interaction patterns and information flow: The interaction between user and DSS starts when the user has a decision to make, and requests assistance from the DSS. Together, the DSS and the user identify the decision problem and its domain (intelligence phase), construct the decision model (design phase) and make a choice (choice phase).

In the intelligence phase, the user informs the DSS of relevant domain and decision knowledge she has access to. The DSS provides information to the user about structuring the decision problem, about the domain and decision knowledge it has, why and when it requires information from the user, extra information that helps the user understand the decision problem, etc.

In the design phase, the user provides information on, for example, preferences, environmental conditions and available options. The DSS provides information about the decision model it constructs, and again, any additional information that helps the user understand the decision problem and the DSS functioning and interaction.

In the choice phase, we come across an interesting characteristic of this type of decision making, due to the hierarchical nature of the team. Because the DSS has superior computational abilities, it can often more easily evaluate the options and choose what it believes to be the best one. However, because the user remains responsible for the decision making, the DSS ‘decision’ is merely advice that is presented to the user. The user must then evaluate this advice, and make the final decision to either accept the advice, or choose another option.

Role interdependencies: The user relies on the DSS to help her structure the decision and provide advice, in other words, to take over some of the reasoning. The DSS usually requires user input on, for example, domain, preferences, environmental conditions. It also relies on the user to make the final decision.

Information Sources: The user and the DSS may use additional information sources. For example, some DSSs assist decision making by integrating various sources of information such as external databases.

3 Sharedness in User-DSS Teams

In this section, we use the analysis of user-DSS teamwork as performed in the previous section, to make precise what a SMM is in the context of user-DSS teams. This can be used as the basis for analyzing to what extent a DSS supports the achievement of a SMM, or to design a DSS such that it does this.

First we identify two types of sharedness, based on the observation that DSSs often do not have explicit mental models about certain aspects of the teamwork (Section 3.1). Second, we discuss for each mental model category which type of sharedness applies and which knowledge should be shared and what can remain distributed (Section 3.2).

3.1 Types of Sharedness

As mentioned previously, we expect appropriate shared knowledge between user and DSS to improve team performance. However, as different definitions of ‘shared’ are possible, it is important to define what is meant by this term [2]. For user-DSS teams, we distinguish two types of sharedness.

In Type 1, the mental models of team members are compared directly and sharedness is defined as the extent to which these mental models overlap (see Fig. 1: Arrow 1). Type 2 stems from the observation that in user-DSS teams, we cannot always directly compare the mental models of these team members, because the DSS does not always have explicit mental models. For example, the DSS does not always have a mental model of itself and the interaction; in this case, the relevant knowledge is implicitly present in the DSS design. In such situations, we compare the mental model of the user directly to the DSS design (see Fig. 1: Arrow 2). This comparison determines if the user’s mental model is an accurate (or at least, compatible) representation of the DSS.

These two ways of comparing mental models are similar to the approach taken in [1], where a distinction is made between comparing mental models directly, and comparing a mental model of one agent with the beliefs another agent has about this mental model.

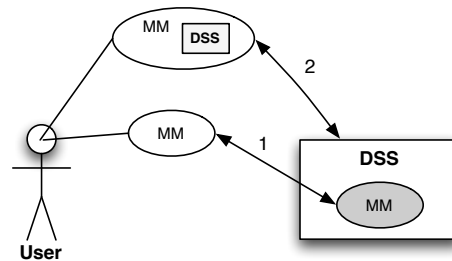


Fig. 1. Two types of sharedness. Type 1 involves comparing the user’s mental model to the DSS’s mental model. Type 2 involves comparing the mental model the user has of the DSS (or the user-DSS interaction), to the DSS itself.

3.2 Desired Shared Mental Model

It cannot be expected that user and DSS have identical knowledge, nor is it desirable, since user and DSS have distinct roles and distinct skills (see Section 2.3). Within the context of heterogeneous teams with distinct roles, of which user-DSS teams are a good example, too much overlap in knowledge can result in redundancy and inefficiency, and contribute to sub-optimal use of resources [14]. Nevertheless, because of their role interdependencies, some knowledge must be shared, to ensure that the user and the DSS have a general shared understanding about the task.

The idea that some, but not all, knowledge should be shared between user and system, is in line with the findings of a study on explanation in knowledge based systems (KBSs) [13]. The authors found that users requested explanation only when deemed necessary and essential for task completion. Such explanation requests were triggered by perceived anomalies in KBS output. In other words, users usually only requested information when they did not understand KBS output that was important for task completion.

We will now discuss, for each mental model category, which type of sharedness applies, and what knowledge should be shared and what can remain distributed. We call the knowledge of a mental model that should be shared the *desired SMM*.

Task mental model: Both user and DSS have a mental model of the task, thus, sharedness Type 1 applies here. Which knowledge elements of the task mental model should be shared or remain distributed depends largely on the specific domain and context.

In general, the DSS will have more detailed knowledge about the task, as it is built specifically for it. However, because the user is responsible for some of the DSS input, and for the final decision, some general task knowledge should be shared. For example, given the correct input, the DSS can calculate the entire option space. The user often cannot, and need not, keep track of all the available options. Nevertheless, some of this knowledge should be shared with the user, as the user will have difficulty making the final decision without some idea of the available options.

Equipment mental model: The user and the DSS should have a mental model of any equipment that is relevant for their part of the task. Thus, here, mental models can be compared directly, and therefore, the sharedness for this mental model category is of Type 1. For this mental model, we also see that detailed knowledge often resides with one team member, but general knowledge is shared. That is, the user and DSS each need a detailed mental model of the equipment that they themselves use. By contrast, if a team member does not operate certain equipment, but its results are relevant for the task, (only) some general knowledge about the equipment should be shared.

Team member mental model: As there are two team members, we consider two different mental models, one concerning the DSS and one concerning the user. First, we consider the team member mental model concerning the DSS. We observe that the user should have such a mental model. However, as most DSSs are not able to explicitly reason about their own knowledge and abilities, the DSS usually does not have an explicit model of itself. Thus, Type 2 sharedness applies here: the mental model that the user has of the DSS is compared directly to the DSS (design). The user's mental model of the DSS need not contain every detail about the DSS. Rather, it should con-

tain sufficient information about DSS knowledge, skills/strengths, limitations, so that team performance can be successful. This is thus the desired SMM.

Second, we consider the team member mental model concerning the user. We assume that the user has a mental model of herself, in so far that she is explicitly aware of her own knowledge, skills, limitations and preferences. If the DSS does not have an explicit user model, there is Type 2 sharedness: the mental model of the user about herself can be compared to the user characteristics as integrated into the DSS design. That is, the DSS is typically built with certain intended users in mind. Certain characteristics of these intended users are integrated into DSS design. For example, a medical DSS intended for doctors will be built differently from a medical DSS that has laymen as intended users. We note that in this case the level of sharedness cannot be changed, as the intended user characteristics built into the DSS cannot be changed, nor can the user adapt her mental model about herself to fit these intended user characteristics.

If the DSS does have an explicit user model, Type 1 sharedness applies, where the DSS's mental model of the current user is compared to the mental model the user has of herself. The former can be used to create more sharedness between the DSS and the user regarding this team member mental model, as knowledge about current user characteristics allows the DSS to *adapt* the interaction to this user (see e.g., [8]). For the team member mental model concerning the user, the desired SMM contains the user characteristics (knowledge, skills, limitations, preferences) relevant for adaptation. For example, if the DSS should be able to adapt to the user's expertise level, it should have knowledge about this expertise level. Any other user characteristics remain distributed, residing with the user.

Team interaction mental model: The user has a mental model of the interaction, whereas the DSS usually does not have such an explicit mental model. Thus, also here, Type 2 sharedness applies. The user's mental model of team interaction is compared with the team interaction as integrated into DSS design. Again, the user needs sufficient team interaction information to achieve successful interaction. This is the desired SMM.

4 Achieving Sharedness

The SMM framework presented in the previous sections provides a basis for determining what knowledge should be shared between user and DSS. The next step is to address how such sharedness can be achieved and maintained. We define a *discrepancy* to be a lack of sharedness regarding a knowledge element. As discrepancies may also exist for elements that need not be shared, we will sometimes use the term 'problematic discrepancy' to emphasize that we are referring to a discrepancy regarding an element of the desired SMM.

To achieve the desired SMM, problematic discrepancies should be resolved. First the discrepancy must be detected, then it can be resolved by adapting (one of) the mental models. We argue that explanation capabilities on the part of the DSS are important for achieving sharedness in mental models. That is, a DSS should be able to explain to the human user its reasoning and the input upon which it bases its reasoning. Explanation has been studied in the context of expert systems [11] and is increasingly being

investigated in a wide variety of intelligent systems such as intelligent virtual agents [6] and recommender systems [17].

Explanation can serve various purposes, such as improving effectiveness, increasing the user's trust in the system and improving transparency of the system [17]. We are particularly interested in the latter, as this facilitates detecting and resolving discrepancies between mental models of DSS and user. Transparency means explaining how the system works, thus giving the user a better understanding of the DSS's reasoning process. This allows the user to detect any discrepancies between the mental models, and subsequently to resolve these by updating the mental models where necessary. Our framework can form a basis for these explanations, determining what should be shared, and thus what should be explained. That is, when adding an explanation facility to a DSS, our framework can be used to determine the requirements for the explanations. Such an explanation facility should provide the transparency necessary to detect and resolve problematic discrepancies. Hence, the explanations should be targeted at the content of the desired SMM.

5 Case Study: Negotiation Support System

In this section we apply our SMM framework to a concrete DSS: the Pocket Negotiator [7]. The Pocket Negotiator is a negotiation support system (NSS). Its aim is to provide focus and structured support for the negotiation process, increasing the user's capacity for structuring and exploring the negotiation space, and reducing the cognitive task load while doing so. The Pocket Negotiator currently exists as a web application prototype (demo video: <http://mmi.tudelft.nl/?q=node/40>). This prototype guides the user through the four major stages that can be discerned in integrative negotiation: private preparation, joint exploration, bidding, and closing. The actual decision making occurs during the bidding stage, when the user must make a decision about about what bid to place next, or whether to accept the opponent's bid.

Using our framework, we provide some examples that illustrate the extent to which this DSS supports the achievement of a SMM, and where there is room for improvement. These examples also illustrate how explanation can improve sharedness.

Task mental model: As a first example, we look at *user preferences*. In the Pocket Negotiator, user preferences are elicited and visible in the preparation and exploration phase. This part of the interface has been designed to help the user explicate her preferences, and to allow sharing this information with the NSS, thereby facilitating achievement of sharedness.

However, during the bidding phase, the user preferences are only implicitly visible in an overall utility rating per bid. The Pocket Negotiator uses the user preferences to determine a utility function, which assigns a utility value to each available option. The link between the user preferences and the assigned bid utility is not very clear, making it difficult for the user to monitor if the DSS has a correct mental model of user preferences and the corresponding utility function. This could be improved using explanation, to maintain sharedness regarding preferences and utility function (cf. [10]). Sharedness regarding user preferences is important, because these preferences form the basis for the choice strategy. If the DSS has incorrect information regarding the

user's preferences (a problematic discrepancy), it will not be able to provide appropriate advice.

An example of a knowledge element for which sharedness is clearly not facilitated in the Pocket Negotiator, is the *strategy*. The Pocket Negotiator does not make clear to the user what kind of strategy it is using. Also, the user has no option to choose a strategy, and thus cannot achieve sharedness in this manner. This lack of sharedness regarding strategy makes it difficult for the user to understand the Pocket Negotiator's advice, giving rise to a problematic discrepancy. If the Pocket Negotiator were to explain how it used the decision model and choice strategy to come to its advice, it would be easier for the user to evaluate the advice. Here of course, the Pocket Negotiator need not share all the details of the choice strategy it uses. The choice strategy involves a number of calculations, which the user does not need to understand fully. It should be possible to provide a reasonable explanation to the user, without going in to the details of the calculations.

Team member mental model concerning user: The Pocket Negotiator design has taken into account the knowledge, abilities and limitations of its intended users. For example, the preference elicitation interface takes into account that users often find it easier to think about their interests than their specific preferences per issue. However, the Pocket Negotiator does not store any knowledge about the current user that it could use for adaptivity purposes. If the Pocket Negotiator were able to have a mental model of the user's level of expertise, and adapt accordingly, this could greatly improve the team performance. For example, novice users may need more explanation to gain sufficient understanding of the decision problem, whereas expert users might get frustrated by too much explanation. Indeed, research has found that novices and experts have different ways of making use of explanations [13].

6 Conclusion

In this paper we have made precise what it means for a user and a DSS to have a SMM. We analyzed the task and team, to specify the relevant knowledge. Subsequently, we specified what we mean by 'shared', defining two types of sharedness that are relevant for user-DSS teams. We then proceeded to describe what knowledge should be shared between user and DSS, and what knowledge can remain distributed. A case study of an NSS illustrated our approach, where we identified explanation as suitable for improving sharedness. Our work can be used as the basis for analyzing to what extent a DSS supports the achievement of a SMM, or to design a DSS such that it does this.

In future work we plan to perform a more elaborate case study. This should include user studies to determine the effect of (different kinds of) explanation on sharedness, and to determine if improved sharedness indeed improves user-DSS team performance. In order to perform such user studies, we must first define ways in which to measure sharedness between user and DSS, to be developed on the basis of [9].

Acknowledgements This research is supported by the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Ministry of Economic Affairs. It is part of the Pocket Negotiator project with grant number VICI-project 08075.

References

1. Burgemeestre, C.B., Liu, J., Hulstijn, J., Tan, Y.: Early requirements engineering for e-customs decision support: Assessing overlap in mental models. In: *Proceedings of the CAiSE Forum*. pp. 31–36 (2009)
2. Cannon-Bowers, J.A., Salas, E.: Reflections on shared cognition. *Journal of Organizational Behavior* 22(2), 195–202 (2001)
3. Cannon-Bowers, J.A., Salas, E., Converse, S.A.: Shared mental models in expert decision making teams. *Current Issues in Individual and Group Decision Making* pp. 221–246 (1993)
4. Damasio, A.R.: *Descartes' Error: Emotion, Reason, and the Human Brain*. Putnam Publishing (1994)
5. Druzdzal, M.J., Flynn, R.R.: Decision support systems. In: Kent, A. (ed.) *Encyclopedia of Library and Information Science*. New York: Marcel Dekker, 2nd edn. (2003)
6. Harbers, M., van den Bosch, K., Meyer, J.J.: A methodology for developing self-explaining agents for virtual training. In: *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems: Volume 2*. pp. 1129–1130 (2009)
7. Hindriks, K.V., Jonker, C.M.: Creating human-machine synergy in negotiation support systems: Towards the pocket negotiator. In: *Proceedings of the 1st International Working Conference on Human Factors and Computational Models in Negotiation*. pp. 47–54 (2008)
8. Hudlicka, E., McNeese, M.D.: Assessment of user affective and belief states for interface adaptation: Application to an air force pilot task. *User Modeling and User-Adapted Interaction* 12(1), 1–47 (2002)
9. Jonker, C.M., van Riemsdijk, M.B., Vermeulen, B.: Shared mental models: A Conceptual Analysis. In: *Coordination, Organization, Institutions and Norms in Multi-Agent Systems at AAMAS2010* (2010)
10. van de Kieft, I., Jonker, C.M., van Riemsdijk, M.B.: Explaining negotiation: Obtaining a shared mental model of preferences. In: *IEA/AIE 2011, Part II, LNAI 6704*. pp. 120–129. Springer Verlag (2011)
11. Lacave, C., Diez, F.J.: A review of explanation methods for heuristic expert systems. *The Knowledge Engineering Review* 19(02), 133–146 (2005)
12. Langan-Fox, J.: Skill acquisition and the development of a team mental model: an integrative approach to analysing organizational teams, task, and context. In: West, M.A., Tjosvold, D., Smith, K.G. (eds.) *International Handbook of Organizational Teamwork and Cooperative Working*, chap. 16, pp. 321–360. John Wiley & Sons (2003)
13. Mao, J., Benbasat, I.: The use of explanations in knowledge-based systems: Cognitive perspectives and a process-tracing analysis. *Journal of Management Information Systems* 17(2), 153–179 (2000)
14. Mohammed, S., Dumville, B.C.: Team mental models in a team knowledge framework: Expanding theory and measurement across disciplinary boundaries. *Journal of Organizational Behavior* 22(2), 89–106 (2001)
15. Scerri, P., Pynadath, D.V., Tambe, M.: Why the elf acted autonomously: Towards a theory of adjustable autonomy. In: *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems: part 2*. pp. 857–864. ACM (2002)
16. Simon, H.A.: *The new science of management decision*. Prentice-Hall, Englewood Cliffs, N.J. (1960)
17. Tintarev, N., Masthoff, J.: A survey of explanations in recommender systems. In: *ICDE'07 Workshop on Recommender Systems and Intelligent User Interfaces*. pp. 801–810 (2007)

Consistent Explanations of Answers to Queries in Inconsistent Knowledge Bases

Hendrik Decker

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain

Abstract. Answers to queries in knowledge bases (a.k.a. deductive databases) can be explained by the facts and the rules used to derive the answers. Similarly, inconsistencies can be explained by facts and rules that violate integrity constraints. We define and show how to compute consistent explanations of answers that possibly lack integrity.

1 Introduction

In order to gain acceptability, decision support systems need to be able to explain their answers to queries posed by users. Often, the derivation that led to the answer may serve as an explanation. More precisely, a sufficient basis for explaining an answer is given by the structure of the query as posed to the underlying knowledge base and the sequence of inference steps by which the answer is derived from some subset of the knowledge base.

Each valid explanation must at least be consistent, i.e., it must neither contradict itself nor the knowledge base. The semantic consistency of the knowledge base usually is encoded by a set of integrity constraints. They are invariant conditions that must be satisfied in each state of the underlying database. If one of them is not satisfied in a given state, then that state is inconsistent.

In practice, however, the databases of large knowledge bases are rarely 100% consistent with the integrity constraints imposed on them. Nevertheless, many, if not the majority of answers derived from such knowledge bases typically turn out to be useful. From a theoretical point of view, that may seem surprising, since, according to the classical *ex falso quodlibet* principle, anything whatsoever can be derived from inconsistency, hence no answer can be trusted.

In [3, 2], we have introduced “answers that have integrity” (abbr., AHI), by devising criteria for deciding whether an answer to a query is trustworthy, even if the queried knowledge base is inconsistent, i.e., if its integrity constraints are violated. Essentially, an answer has integrity if one of its explanations does not overlap with any explanation of integrity violation. For more general classes of knowledge bases than those considered in [3, 2], we are going to define in this paper that an explanation of the answer to a query is consistent if it does not overlap with any explanation of the violation of any constraint. Apart from greater generality, this paper distinguishes itself from the aforementioned ones mainly by its emphasis on consistent explanations, by which any answer in a knowledge base can be explained, no matter if the answer has integrity or not.

In Section 2, we outline the formal setting and the concept of predicate completions as a basis for defining explanations. In Section 3, we define explanations. In Section 4, we characterize and show how to compute consistent explanations of answers that may or may not have integrity. In Section 5, we conclude.

2 Preliminaries

In 2.1, we outline the formal framework upon which this paper is built. In 2.2, we make explicit some implicit parts of the knowledge base, since they may contribute to explain negative answers.

2.1 The Formal Setting

The formal setting of this paper is the conventional database logic. A *knowledge base* is a tuple (D, IC) ; D is a *database*, i.e., a finite set of clauses of the form $A \leftarrow B$ where the *head* A is an atom and the *body* B is a conjunction of literals, and IC is an *integrity theory*, i.e., a finite set of integrity constraints. The variables in each clause are implicitly $\forall$ -quantified. W.l.o.g., each constraint and each query is assumed to be represented as a *denial* of the form $\leftarrow B$, where B is a conjunction of literals. For ease of notation, we only consider propositional top-rank queries; a generalization to quantified queries is straightforward.

Let $\mathcal{L}$ denote a universal database language and $\mathcal{L}_c$ the set of constant terms in $\mathcal{L}$, which we represent by natural numbers. Depending on the context, $=$ may stand for the identity predicate in $\mathcal{L}$, the assignment symbol in substitutions, or meta-level equality. We use ‘;’ as a delimiter between elements of sets of clauses, while ‘ $\wedge$ ’ symbolizes conjunction in the body of clauses.

For each database D and each denial $\leftarrow B$, we write $D(B) = \text{true}$ and say ‘ B is true in D ’ if B is a logical consequence of the theory associated to D (e.g., the completion of D , or some standard model theory such as the set of stable models of D). Otherwise, we write $D(B) = \text{false}$ and say ‘ B is false in D ’.

We do not flatten databases by materializing predicate extensions, since such an unfolding of clauses may lead to a loss of explanatory information.

Example 1. The two databases $D_1 = \{p \leftarrow r; r; s\}$ and $D_2 = \{p \leftarrow s; r; s\}$ have the same flattened version: $\{p; r; s\}$. That version, however, does not allow to identify that r is part of the explanation of p in D_1 and s is not, nor that s is part of the explanation of p in D_2 and r is not.

2.2 Only-if Halves

Explanations of negative answers need to recur on the negative information that is conveyed by the database. Negative consequences of databases usually are justified by some closure axioms. The well-known completion $\text{comp}(D)$ is a standard axiomatization for inferring negative consequences from a database D . For each predicate p in $\mathcal{L}$, the *iff-completion* (also called *predicate completion* or, in short, *completion*) of p in D be defined as in [1, 7].

Example 2. Let $D = \{p(x, 1) \leftarrow r(x); p(1, y) \leftarrow s(y, z); s(1, 2); s(2, 3)\}$. Then, $\text{comp}(D)$ contains the following iff-completions, each of which is a universally closed sentence with existentially quantified ‘local’ variables that do not occur in the head of any clause. We omit the universal quantifier prenex for all non-local variables in the completions below, and also the equality theory associated to $\text{comp}(D)$ that interprets $=$ as identity.

$$p(x, y) \leftrightarrow (y = 1 \wedge r(x) \vee x = 1 \wedge \exists z(s(y, z)))$$

$$r(x) \leftrightarrow \text{false}$$

$$s(x, y) \leftrightarrow (x = 1 \wedge y = 2 \vee x = 2 \wedge y = 3)$$

Definition 1. Let D be a database and p a predicate in $\mathcal{L}$.

- a) Let $\underline{p}$ denote the iff-completion of p in D .
- b) Each formula obtained by substituting each $\forall$ -quantified variable in $\underline{p}$ by some ground term is called a *basic instance* of $\underline{p}$.
- c) The sentence obtained by replacing $\leftrightarrow$ in $\underline{p}$ or in any basic instance $\underline{p}^*$ of $\underline{p}$ with $\rightarrow$ is called the *only-if half* of $\underline{p}$, or, resp., of $\underline{p}^*$. The atom to the left of $\rightarrow$ of an only-if half H is called the *head* of H and be denoted by $h(H)$.
- d) Let D^+ denote the set of ground instances of clauses in D , and D^- the set of basic instances of only-if halves of predicates in $\mathcal{L}$. Assume that D^- is factorized modulo renamings of variables.

For clarity, elements of D^- should be represented in a simplified form, if possible. It is obtained by replacing equations with their truth value and applying common equivalence-preserving laws for the composition of subformulas with *true* or *false*. Elements of D^- whose simplification is *true* are omitted.

Example 3. The only-if half of $\underline{p}$ in Example 2 is

$$p(x, y) \rightarrow (y = 1 \wedge r(x) \vee x = 1 \wedge \exists z(s(y, z))).$$

Its instance $p(1, 1) \rightarrow (1 = 1 \wedge r(1) \vee 1 = 1 \wedge \exists z(s(1, z)))$, obtained by the substitution $\{x = 1, y = 1\}$, is obviously equivalent to $p(1, 1) \rightarrow (r(1) \vee \exists z(s(1, z)))$. Similarly, the instance $p(2, 3) \rightarrow (3 = 1 \wedge r(2) \vee 2 = 1 \wedge \exists z(s(3, z)))$ of $\underline{p}$ simplifies to $p(2, 3) \rightarrow \text{false}$, which is equivalent to $\sim p(2, 3)$. Similarly, the instance $s(2, 3) \rightarrow (2 = 1 \wedge 3 = 2 \vee 2 = 2 \wedge 3 = 3)$ of the only-if half of $\underline{s}$ simplifies to $s(2, 3) \rightarrow \text{true}$, which is equivalent to *true* and thus can be omitted.

3 Excerpts, Explanation Bases, Explanations

In 3.1, we define an ‘excerpt’ of a database D to consist of a ‘positive’ and a ‘negative excerpt’, i.e., a subset of D^+ and a subset of D^- . In 3.2, we define, for each denial $\leftarrow B$ such that $D(B) = \text{true}$, an ‘explanation base’ of B in D to be an excerpt E of D such that B is a logical consequence of E . In 3.3, we define an ‘explanation’ of B in D to be a minimal explanation base of B in D .

3.1 Excerpts

Definition 2. Let D be a database. Each subset of D^+ is called a *positive excerpt* of D , and each subset of D^- is called a *negative excerpt* of D . The usual equality theory of $\text{comp}(D)$ be associated by default to each negative excerpt. An *excerpt* (P, N) of D consists of a positive excerpt P and a negative excerpt N of D . For each excerpt E , we denote its positive part by E^+ , its negative part by E^- , and the union of E^+ and E^- by $\hat{E}$. We say that two excerpts E, E' *overlap* if $\hat{E} \cap \hat{E}' \neq \emptyset$.

A partial order $\leq$ is defined in a natural way for excerpts, as follows.

Definition 3. Let D be a database and $E = (P, N), E' = (P', N')$ be two excerpts of D . We define that $E \leq E'$ if $\hat{E} \subseteq \hat{E}'$, i.e., if $P \subseteq P'$ and $N \subseteq N'$. We write $E < E'$ and say E is *smaller than* E' if $E \leq E'$ and not $E' \leq E$ holds.

The lemma below, which is easily deduced from Definition 2, states an invariance property whose importance will reveal itself in 3.2, Theorem 1.

Lemma 1. For each database D , each excerpt E of D and each query $\leftarrow B$ such that $\hat{E} \models B$, B is *true* in each database D' such that E is an excerpt of D' .

3.2 Explanation Bases

An ‘explanation base’ E of the truth of a positive answer to a query $\leftarrow B$ (shortly, an ‘explanation base’ of B) in a database D is going to be defined as an excerpt E of D such that B is a logical consequence of $\hat{E}$.

Example 4. Let $D = \{p \leftarrow q; p \leftarrow r; q; r; s\}$. Clearly, $P_1 = \{p \leftarrow q; q\}$ is a positive excerpt of D that serves to explain why p is *true* in D . Also $P_2 = \{p \leftarrow r; r\}$ is a positive excerpt of D that explains why p is *true* in D .

In general, no negative excerpts are needed for explaining positive consequences of any database without negation. However, it is not possible to infer any negative consequence, such as sentences represented by negative literals or constraints in denial form, from any positive excerpt.

Example 5. It is not possible to explain $\sim q$ in $D = \{p\}$ by any subset of D . In particular, none of the positive excerpts $\emptyset$ and $\{p\}$ of D may explain $\sim q$ in D , since each of them is also a positive excerpt of $D' = D \cup \{q\}$, while the truth value of $\sim q$ is not the same in D and D' .

In the preceding example, the absence of q in D must be captured explicitly for explaining why $\sim q$ is *true* in D . Thus, each explanation base is going to consist of a positive and a negative excerpt, as defined subsequently.

Definition 4. Let D be a database, E an excerpt of D , and $\leftarrow B$ a query such that $D(B) = \text{true}$. E is called an *explanation base* of B in D if $\hat{E} \models B$.

Example 6.

a) Let $D = \{p \leftarrow q, s; q; r\}$ and $B = \sim p$. Clearly, $E_0 = (\emptyset, \{p \rightarrow q, s; \sim s\})$ is an explanation base of B in D . However, neither $E_1 = (\{p \leftarrow q, s\}, \{\sim s\})$ nor $E_2 = (\{p \leftarrow q, s; q\}, \{\sim s\})$ is an explanation base of B in D , since B is not a logical consequence of $\hat{E}_i$ ($i=1,2$). Note that both excerpts E_i of D are also excerpts of $D' = D \cup \{p \leftarrow r\}$, but $D'(B) = false$.

b) Let $D = \{p(x) \leftarrow q(x), \sim r(x); q(1); q(2); r(3)\}$. Then, $D(p(1)) = true$, but $p(1)$ is not explainable by any positive excerpt of D alone, since each subset of D^+ is also a positive excerpt of $D' = \{p(x) \leftarrow q(x), \sim r(x); q(1); q(2); r(1)\}$, but $D'(p(1)) = false$. However, $p(1)$ can be explained by the excerpt $E = (\{p(1) \leftarrow q(1), \sim r(1); q(1)\}, \{\sim r(1)\})$ of D , since $\hat{E} \models p(1)$, and $p(1)$ is indeed *true* in each database D' of which E is an excerpt.

Theorem 1, below, follows from Definition 4 and Lemma 1. It states that for each explanation base E of a positive answer to any query $\leftarrow B$ in a database D , the truth of B in D does not depend on any clause that is not captured in E . In other words, the truth of B remains invariant when D is modified to D' by adding or deleting clauses that are not excerpted by E . It is essential that this invariance holds although truth in databases is non-monotonic. Hence, each explanation base of B contains the essence of the truth of B .

Theorem 1. For each database D , each query $\leftarrow B$, each explanation base E of B in D , and each database D' of which E is an excerpt, $D'(B) = true$.

3.3 Explanations

By Definition 4, each explanation base is sufficient for inferring the explained sentence. However, not each element in a given explanation base is necessary, in general.

Example 7. Clearly, $(\{p(1) \leftarrow r(1,2); r(1,2); s(3)\}, \emptyset)$ is an explanation base of $p(1)$ in $D = \{p(x) \leftarrow r(x,y); r(1,1); r(1,2); r(2,2); s(3)\}$, but $s(3)$ is unnecessary for explaining $p(1)$ in D .

Thus, an explanation is going to be defined as a minimal explanation base, without unnecessary elements, as follows.

Definition 5. Let D be a database and $\leftarrow B$ a denial.

a) An explanation base E of F in D is called an *explanation* of B in D if there is no explanation base of B in D such that $E' < E$. If $\leftarrow B$ is a query, we also say: E is an *explanation of the answer 'yes'* to $\leftarrow B$ in D . If $\leftarrow B$ is a constraint, we also say: E is an *explanation of the violation* of $\leftarrow B$ in D .

b) Let E be an explanation of the answer 'yes' to the query $\leftarrow \sim answer$ in $D \cup \{answer \leftarrow B\}$, where *answer* is a 0-ary predicate not occurring in D . If $\leftarrow B$ is a query, we say: E is an *explanation of the answer 'no'* to $\leftarrow B$ in D . If $\leftarrow B$ is a constraint, we say: E is an *explanation of the satisfaction* of $\leftarrow B$ in D .

Example 8.

a) Let $D = \{p \leftarrow q(1, 2); q(2, y) \leftarrow r(y); r(1)\}$. The only explanation of the answer *yes* to $\leftarrow p$ in D is $(\emptyset, \{p \rightarrow q(1, 2); \sim q(1, 2)\})$.

b) Let $D = \{p \leftarrow \sim q; q \leftarrow \sim r; q \leftarrow \sim s\}$. The two explanations of the answer *yes* to $\leftarrow \sim p$ in D are $(\{q \leftarrow \sim r\}, \{p \rightarrow \sim q; \sim r\})$ and $(\{q \leftarrow \sim s\}, \{p \rightarrow \sim q; \sim s\})$.

c) Let $D = \{p \leftarrow q; p \leftarrow \sim q\}$ and $I = \leftarrow p$. The two explanations of the violation of I in D are $(\{p \leftarrow \sim q\}, \{\sim q\})$ and $(D, \emptyset)$.

d) Let $D = \{p(x) \leftarrow r(x); r(1)\}$ and $I = \exists x(r(x) \wedge \sim p(x))$ be a constraint. Clearly, $D(I) = \text{false}$ (in fact, I is violated in each database that contains $p(x) \leftarrow r(x)$). A denial form of I is $\leftarrow \text{violated}$, where *violated* is defined by $\{violated \leftarrow \sim q; q \leftarrow r(x), \sim p(x)\}$ (q is a fresh 0-ary predicate). Thus, explanations of *violated* in $D' = D \cup \{violated \leftarrow \sim q; q \leftarrow r(x), \sim p(x)\}$ explain the violation of I in D . Hence, for each $\mathcal{K} \subseteq \mathcal{L}^c$ such that $1 \in \mathcal{K}$, an explanation of *violated* in D' is given by $(\{violated \leftarrow \sim q\} \cup \{p(i) \leftarrow r(i) \mid i \in \mathcal{K}\}, \{q \rightarrow \exists x(r(x) \wedge \sim p(x))\} \cup \{\sim r(i) \mid i \notin \mathcal{K}\})$. There is no other explanation of *violated* in D' .

e) Let $D = \{p \leftarrow q(1, x); q(2, y) \leftarrow r(y); r(1)\}$. The only explanation of the answer *no* to query $\leftarrow p$ in D is $(\emptyset, \{p \rightarrow q(1, i) \mid i \in \mathcal{L}^c\} \cup \{\sim q(1, i) \mid i \in \mathcal{L}^c\})$.

f) Let $D = \{p \leftarrow r(x), s(x); r(1); s(2)\}$. Each explanation of the answer *no* to $\leftarrow p$ in D contains the excerpt $E_0 = (\emptyset, \{p \rightarrow \exists x(r(x), s(x)); \sim s(1); \sim r(2)\})$ of D . Moreover, each such explanation contains, for each $i \in \mathcal{L}^c$, $i > 2$, either $\sim r(i)$ or $\sim s(i)$, and nothing else.

g) Let $D = \{p \leftarrow q; p \leftarrow \sim q; q\}$ and $I = \leftarrow p$. The two explanations of the violation of I (i.e., of $\sim p$) in D are $(\{p \leftarrow q; q\}, \emptyset)$ and $(\{p \leftarrow q; p \leftarrow \sim q\}, \emptyset)$.

h) Let $D = \{r(1), r(3), s(2), s(4)\}$ and $I = \leftarrow r(x), s(x)$ a constraint. Depending on the extent of $\mathcal{L}^c$, there may be many explanations of the satisfaction of I in D . The positive part of each such explanation is empty, and each contains the excerpt $(\emptyset, \{answer \rightarrow \exists x(r(x), s(x))\} \cup \{\sim s(1), \sim s(3), \sim r(2), \sim r(4)\})$ of $D \cup \{answer \leftarrow r(x), s(x)\}$. (Recall that the explanation of the satisfaction of I in D is the explanation of $\sim answer$ in $D \cup \{answer \leftarrow r(x), s(x)\}$.) Moreover, the negative part of each explanation of the satisfaction of I in D contains, for each $i > 4$ in $\mathcal{L}^c$, either $\sim r(i)$ or $\sim s(i)$, and no other element.

i) Let $D = \{p \leftarrow q, \sim q\}$, $I_1 = \leftarrow p$ and $I_2 = \leftarrow \sim p$. The only explanation of the satisfaction of I_1 as well as of the violation of I_2 in D is $(\emptyset, \{p \rightarrow q, \sim q\})$.

Note that, in examples 8c, g, i the constraints, together with the database rules, are either always satisfied (i.e., tautological) or always violated (i.e., contradictory), independent of the facts in the database. However, the explanations of tautological satisfaction and contradictory violation are not empty, as one could suspect. In particular, this means that our concept of explanations is applicable even in databases with unsatisfiable integrity theories.

4 Consistent Explanations

To begin with, let us briefly address the issue of self-consistency, i.e., the requirement that an explanation be free of contradictions in itself. Each explanation is a subset of $(D^+ D^-)$, which is essentially equivalent to $comp(D)$. Thus, if $comp(D)$ is consistent, then so is each explanation in D . It is well-known that the completions of significantly large classes of databases (e.g., stratified ones) are consistent. So, let us focus on databases with consistent completion, by which the self-consistency of all explanations is guaranteed. However, self-consistent explanations may be inconsistent with the integrity theory of the knowledge base, as shown by Example 9, which in turn motivates Definition 6.

Example 9. Let the knowledge base $(D, \{I\})$ be defined by $I = \leftarrow p(x, x)$ and $D = \{p(x, y) \leftarrow q(x), r(y, x); q(0); q(1); r(0, 1); r(1, 1)\}$. The explanation $E = (\{p(1, 1) \leftarrow q(1), r(1, 1); q(1); r(1, 1)\}, \emptyset)$ of $p(1, 1)$ in D is inconsistent with I , since it coincides with the explanation of the violation of I in D .

Definition 6. Let D be a database, IC an integrity theory and E an explanation of an answer to some query in D . We say that E is *consistent* in (D, IC) if it does not overlap with any explanation of the violation of any $I \in IC$ in D .

In 4.1 and 4.2, we show how to compute explanations.

4.1 Computing Explanations

As seen in [3], explanations in knowledge bases without negation in the body of clauses can be easily computed by the well-known query answering procedure SLD [5, 7]. Each explanation of each answer corresponds to a refutation R in any SLD tree that computes the answer: the input clauses of R , instantiated with the substitution computed by R , are the elements of the explanation. Similarly, each SLDNF refutation serves to explain its root, and each finitely failed SLDNF tree serves to explain the answer *no* to its root. To describe that in more detail, we first recall some basic SLDNF issues from [7] and introduce some denotations.

Each SLDNF computation involves one top-rank computation and possibly several subsidiary computations of lower rank. It is easy to see that no finitely failed tree of rank $n-1$ that is subsidiary to some finitely failed tree T of rank n could contribute anything to explain the answer *no* computed by T . Thus, such subsidiary trees are ignored in Definition 7. For an SLDNF computation S , it characterizes the set of computations involved in S that contribute to an explanation of the answer to the root of S .

Definition 7. Let S be an SLDNF computation of rank n ($n \geq 0$).

- a) The set S_r of *explanatory refutations* of S consists of each refutation R of rank k ($k \leq n$) involved in S such that R either is the top-rank refutation of S or R is subsidiary to a tree T in S_t of rank $k + 1$.
- b) The set S_t of *explanatory trees* of S consists of each finitely failed tree T of rank k ($k \leq n$) involved in S such that T either is the top-rank computation of S or T is subsidiary to a refutation in S_r of rank $k + 1$.

Note that the mutual recursion in parts *a* and *b* of Definition 7 poses no problem, due to the inductive structure of ranks.

For each refutation R , let θ_R denote the substitution computed by R . For each clause C , each only-if half H and each substitution σ , let $C\sigma$ and $H\sigma$ be obtained by applying σ to the $\forall$ -quantified variables in C or, resp., H .

Now, we are in the position to define computed explanation bases.

Definition 8. For each SLDNF computation S , the *computed explanation base* E_S of S consists of

$$E_S^+ = \{C\theta_R \mid C \in D, R \in S_r, C \text{ is input clause in } R\}$$

and

$$E_S^- = \{H\gamma \mid H\gamma \in D^-, \gamma \text{ is a ground substitution, } h(H) \text{ is selected in a node of a tree in } S_t\}.$$

Thus, E_S^+ is obtained by instantiating the positive input clauses of each refutation $R \in S_r$ with θ_R . E_S^- is obtained by collecting the only-if-halves of all ground instances of each positive literal selected in any node of any tree in S_t .

Example 10. Let $D = \{p \leftarrow q(x, x); q(1, 2); q(2, 3)\}$. The answer *no* to the query $\leftarrow p$ is computed by a finitely failed tree consisting of a single branch rooted at the goal $\leftarrow p$, which is reduced to the goal $\leftarrow q(x, x)$, which fails. The only-if halves of the two selected positive literals in the tree are $p \rightarrow q(x, x)$ and $q(x, x) \rightarrow (x=1 \wedge x=2 \vee x=2 \wedge x=3)$. The latter obviously is equivalent to $\sim q(x, x)$. Thus, $(\emptyset, \{p \rightarrow q(i, i) \mid i \in \mathcal{L}^c\} \cup \{\sim q(i, i) \mid i \in \mathcal{L}^c\})$ is the computed explanation base of the tree, which in fact explains the answer *no* to $\leftarrow p$ in D .

The following result follows straightforwardly from Definitions 7 and 8.

Theorem 2. If D is a database, $\leftarrow B$ a query, S an SLDNF computation of $D \cup \{\leftarrow B\}$ and α the computed answer of S , the computed explanation base of S is an explanation base of α . If $S_t = \emptyset$ and D does not contain any self-recursively defined predicate, the computed explanation base of S is an explanation of α .

The following example illustrates that, in general, explanation bases computed by SLDNF are not necessarily minimal.

Example 11. Let $D = \{p \leftarrow q, r; r \leftarrow s\}$. Depending on the selection function, there are three SLDNF trees of $D \cup \{\leftarrow p\}$. Each of these trees computes the answer *no*. In each, the goal $\leftarrow q, r$ is derived from the root. Then, if q is selected, the computation terminates with failure. If, instead, r is selected, the derived goal $\leftarrow q, s$ may fail in two ways, after selecting either q or s . Hence, depending on the selection, only one of the following three explanation bases $(\emptyset, \{p \rightarrow q \wedge r; \sim q\})$, $(\emptyset, \{p \rightarrow q \wedge r; r \rightarrow s; \sim q\})$, $(\emptyset, \{p \rightarrow q \wedge r; r \rightarrow s; \sim s\})$ can be drawn from the respective SLDNF tree. Only the first and the last of these explanation bases are explanations, while the middle one is not because it properly contains the first one and thus is not minimal.

Thus, each explanation base E computed by SLDNF must eventually be minimized, by checking if any proper subset of E satisfies Definition 5.

4.2 Computing Consistent Explanations

To check explanations for consistency is easy: once they are computed, it is straightforward to check them for overlappings. However, not each finitely failed SLDNF tree provides a consistent explanation of the answer *no* to its root, even though such an explanation and such a tree may exist, as shown by the following continuation of Example 11.

Example 12. Depending on whether q or r is selected in the goal $\leftarrow q, r$, either $E_1 = (\emptyset, \{p \rightarrow q \wedge r; \sim q\})$ or $E_2 = (\emptyset, \{p \rightarrow q \wedge r; r \rightarrow s; \sim s\})$ is the explanation obtained from a fair finitely failed SLDNF tree of $D \cup \{\leftarrow p\}$. Now, let $\leftarrow \sim q$ be a constraint. Clearly, the explanation $(\emptyset, \{\sim q\})$ of its violation in D overlaps with E_1 but not with E_2 . Hence, the finitely failed SLDNF tree that selects literals from left to right does not provide a consistent explanation of the answer *no* to $\leftarrow p$ in D , while the tree that selects from right to left does.

The incompleteness of finitely failed SLDNF trees for computing explanations had been identified already in [4]. The solution in [4] can be adapted to the present context. It essentially consists in selecting and resolving not just one, but possibly all literals in each goal of a finitely failed tree. Thus, more derivations than those that are needed for computing the answer have to be explored, so that more explanations can be drawn from them. In Example 12, for instance, both q and r have to be selected and resolved for obtaining a tree from which both explanations E_1 and E_2 can be drawn.

Definition 9 describes a class of SLDNF trees from which sufficiently many explanation bases and thus explanations can be drawn.

Definition 9. Let D be a database and $\leftarrow B$ a query.

- a) An *explanation-of-success refutation* R of $D \cup \{\leftarrow B\}$ is constructed like an SLDNF refutation of $D \cup \{\leftarrow B\}$, except that, for resolving each selected ground negative literal, an explanation-of-failure tree of $D \cup \{\leftarrow B\}$ is built.
- b) An *explanation-of-failure tree* T of $D \cup \{\leftarrow B\}$ is a tree of finitely failed SLDNF derivations rooted at $\leftarrow B$. For each goal node N in T , each literal in N is selected and resolved as usual against each matching input clause. Each subsidiary refutation of $D \cup \{\leftarrow A\}$ in T for terminating a derivation in T with failure at a selected ground literal $\sim A$ is an explanation-of-success refutation of $D \cup \{\leftarrow A\}$.
- c) An *explanatory computation* S of $D \cup \{\leftarrow B\}$ is either an explanation-of-success refutation of $D \cup \{\leftarrow B\}$ or an explanation-of-failure tree of $D \cup \{\leftarrow B\}$.

Note that the mutual recursion in Definition 9 does not pose any problem as long as all derivations are finite, i.e., the rank of each subsidiary computation decreases with each recursion step.

Let answers and explanations computed by explanatory computations be defined in analogy to answers and explanations computed by SLDNF computations.

Although not each explanation of the answer computed by some explanatory computation S can be drawn from S , the following result can be shown by induction on the rank of explanatory computations.

Theorem 3. Let D be a database, I an integrity constraint that is violated in D , S an explanatory computation of the violation of $I \in IC$, $\leftarrow B$ a query, α a computed answer of $D \cup \{\leftarrow B\}$, and E an explanation of α . E is consistent if and only if it does not overlap with any explanation drawn from S .

5 Conclusion

Traditionally, explanations of answers in knowledge bases are meant to inform the user why the answers have been given, or how they have been derived. That setting has been transgressed in two ways in this paper.

Firstly, human agents are not meant to be the main recipients of explanations that are provided by tracing *SLDNF* computations, since such traces may still be too machine-oriented for the typical end user, and thus may only be the basis for more human-oriented displays. Rather, programmed agents such as modules for deciding if given answers have integrity [2] are meant to be served by the explanations as defined in this paper.

Secondly, our explanations do not only detail why and how answers have been found. Rather, they also provide the essential information whether a given answer is consistent with regard to the integrity constraints that characterize the intended semantics of the knowledge base. That is especially important in knowledge bases in which extant inconsistencies have to be tolerated. Inconsistency tolerance on the side of users and programmed agents is made feasible by indicating that a given answer may conflict with any integrity constraint, i.e., by explaining if it is consistent or not. A related application of our concept of explanations is cause-based integrity checking [3].

Further work is necessary to reduce the amount of derivations that have to be constructed for deciding the consistency of answers. Earlier results in [4] can be exploited for that. Also, we intend to explore ways of deriving answers that are alternative to *SLDNF*, such as answer set computing [6].

References

1. K. Clark. Negation as Failure. In H. Gallaire, J. Minker (eds): *Logic and Data Bases*, 293-322. Plenum Press, 1978.
2. H. Decker. Answers that Have Integrity. Presented at the ICALP workshop SDKB 2010, to appear in post-workshop proc. of SDKB, Springer LNCS, 2011.
3. H. Decker. Toward a Uniform Cause-based Approach to Inconsistency-tolerant Database Semantics. In R. Meersman, T. Dillon and P. Herrero (Eds.), *On the Move to Meaningful Internet Systems: OTM 2010*, Part II, 983-998. Springer LNCS 6427, 2010.
4. H. Decker. On Explanations in Deductive Databases. *Proc. 3rd Workshop on Foundations of Models and Languages for Data and Objects*, Inst. f. Informatik, Tech. Univ. Clausthal, Informatik-Bericht 91/3, 173-185, 1991.
5. R. Kowalski. Predicate Logic as a Programming Language. *Proc. 6th IFIP*, 569-574. North-Holland, 1974.
6. V. Lifschitz. What is Answer Set Computing? *Proc. 23rd AAAI*, 1594-1597, 2008.
7. J. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer, 1987.

Explanation and Corrective Feedback in Grammar Checking Systems

Marie Garnier^{1,2}

¹ Cultures Anglo-Saxonnes, Université Toulouse 2, 5 allées Antonio Machado,
31058 Toulouse, France

² IRIT, Université Toulouse 3, 118 route de Narbonne,
31062 Toulouse, France
garnier@irit.fr

Abstract. This paper presents an analysis of a corpus of corrective feedback messages collected from the grammar checking component of MS Word 2007. We introduce the concept of corrective feedback in Second Language Acquisition and Computer-Assisted Language Learning research and focus on overlaps between explanation and corrective feedback. We analyze the rhetorical relations at play in the grammar checker's feedback messages in order to study the role and location of explanation in these messages. Drawing from previous SLA and CALL research, we sketch guidelines for the design of effective corrective feedback messages including explanations. This constitutes the first step in the implementation of tailored corrective feedback in a grammar checker targeted at French users and learners of English.

Keywords: corrective feedback, explanation, grammar checking, computer-assisted language learning, second language acquisition

1 Introduction

Explanation and its different facets are an essential part of language education. Specifically, explanation often comes into play in the correction of language learners' errors, and is part of what Second Language Acquisition (SLA) researchers call "feedback". In this paper, we introduce the concept of "corrective feedback", the type of feedback that is given to learners in the attempt to elicit a spontaneous correction of their erroneous production. Even though the role of corrective feedback (CF) is not strictly to explain, explanation is an element of CF, especially in the case of meta-linguistic feedback. CF and explanation are also used in the field of Computer-Assisted Language Learning (CALL) to guide learners in the correction of their errors, especially in CALL systems using Natural Language Processing (NLP), hence called Intelligent CALL (ICALL).

The research presented here is part of the *CorrecTools* project, which aims at designing detection and correction strategies for errors produced by French users of English, focusing on errors that are not detected by already existing commercial and

academic grammar checkers for English. We adopt a didactic approach and wish to design tailored CF to supplement error correction.

In addition to error detection and correction, most grammar checkers already provide users with feedback messages. This paper presents a study of feedback messages generated by the grammar checker component of Microsoft Office Word 2007, with the objectives of discovering how these messages are constructed, what rhetorical relations and elements they make use of, as well as to what extent they use explanation. From these observations, we propose guidelines for the generation of feedback messages in grammar checkers and sketch further directions for research.

We use SLA theory and CALL research on CF as a framework for our analysis of the feedback messages of grammar checkers. The rhetorical relations and elements used in our analyses are based on Rhetorical Structure Theory (RST) [1]. The concept of explanation is understood in its wider definition [2], according to which its main objective is to enable the reader to reach a goal. In this framework, explanation has fuzzy boundaries and involves a large number of different elements.

This paper is organized as follows: first, we introduce the notion of CF in SLA and CALL and present the main conclusions of previous research as to the effectiveness of different feedback types, and the ways in which they can be adapted for use in ICALL programs and grammar checkers; then we focus on the analysis of feedback messages and the role of explanation in the grammar checker component of Word 2007; finally we present a sketch of guidelines for the generation of explanatory feedback for learners and users of grammar checkers.

2 Corrective Feedback in SLA and CALL

2.1 Definition of Corrective Feedback

In [3], Ellis gives the following definition for "feedback": "In language acquisition, 'feedback' refers to information given to learners which they can use to revise their interlanguage [...]; negative feedback refers to information that indicates a hypothesis is incorrect". Negative feedback is increasingly referred to as "corrective feedback", putting the emphasis on the remediation that the feedback is supposed to enable. There are at least four identified dichotomies in CF: spoken vs. written, explicit vs. implicit (learner is provided with the correct form and is made aware that an error was committed / correct form is given without mention that an error was committed), direct vs. indirect (correct form is given / error is simply highlighted), focused vs. unfocused (only one type of error is the focus of feedback / all errors are corrected).

In [4], Ellis indicates that CF can consist of:

1. an indication that an error has been committed, or
2. provision of the correct target language form, or
3. meta-linguistic information as to the nature of the error, or
4. any combination of these.

In [5], Lyster and Ranta found that language teachers used six different types of CF:

1. explicit correction,
2. recast (teacher reformulates and corrects the erroneous segment),
3. clarification request,
4. meta-linguistic feedback,
5. elicitation,
6. repetition.

The only feedback type that includes explanations is meta-linguistic feedback, defined in [4] as "utterances that provide comments, information, or questions related to the well-formedness of the learner's utterance".

2.2 Corrective Feedback in SLA

There is no consensus about several of the facets of CF in SLA. Research on CF focuses in particular on the effectiveness and desirability of CF use [6, 7], the effectiveness of certain types of CF as opposed to others [8, 9, 10], and the appropriate contexts and timing for the use of different CF types [4, 11].

CF research reflects different theories of second language acquisition. It fits particularly into the framework of Schmidt's noticing hypothesis [12], which states that second language acquisition is a conscious process, and interactionist theory [13], which claims that negative evidence in the form of interactional feedback is necessary for the learner to notice the gaps between her/his interlanguage and the target forms. This theoretical framework is the one adopted in CALL studies of CF, and is the one we use in our analysis of feedback in grammar checkers.

2.3 Corrective Feedback in CALL

According to CALL researchers [14], there is little research on CF in CALL. It has mainly been used and researched in Intelligent CALL, i.e. CALL programs using NLP techniques to generate morpho-syntactic analyses of the productions of learners and provide tailored feedback. The use of CF in CALL is more limited than in the oral classroom, as the six feedback types presented in [5] correspond to only four feedback possibilities in CALL [15]:

1. correct answer given (explicit correction and recast),
2. "try again!" or "I cannot understand" (clarification request),
3. explanation of error type (meta-linguistic feedback),
4. highlighting of the error (elicitation and repetition).

Moreover, Heift and Schulze emphasize the fact that feedback type 1 (correct answer) cannot be used successfully in an interactionist framework, as no interaction or negotiation of meaning between learner and computer is possible once the correct answer has been given.

In [14], Heift studied the effect of different types of feedback on learner uptake (i.e. the learner's attempt to correct their errors after receiving feedback) and found that "feedback that provides an explanation of the error and also highlights the error in the student input (Meta-linguistic + Highlighting) is most effective at eliciting learner uptake." We use these findings as to the importance and role of explanation in learning to inform the creation of effective corrective feedback in grammar checkers.

3 A Survey of Feedback in a Grammar Checker for English

3.1 Grammar Checkers as Learning Tools

Grammar checkers and ICALL tutors are different tools with different goals and sets of users. Typically, grammar checkers are not targeted at second language learners, and their main goal is to correct errors rather than elicit learning. However, grammar checking technology enters into the creation of ICALL tutors to enable the automatic correction of learner productions and the generation of CF. Research has also been conducted on the potential and limitations of off-the-shelf or academic grammar checkers in conjunction with classroom teaching as an aid to writing instruction for both native speakers [16] and second language learners [17].

Moreover, most grammar checkers include at least some form of corrective feedback. We surveyed a set of ten grammar checkers of different types (off-the-shelf, freeware, online checkers, stand-alone applications, extensions for word processors, academic projects), and, using Heift and Schulze's [15] classification, we found that all of them use highlighting, 9 out of 10 use meta-linguistic feedback, 7 give the error type, and 4 use illustrations. Backed by SLA and CALL research, there is a potential for the development of grammar checkers with tailored CF including relevant explanations.

3.2 Corrective Feedback Messages in Microsoft Word 2007

According to our survey, the grammar checking component in Word 2007 is one of the two grammar checkers that use all of the relevant CF types for this type of tool. It is also one of the three grammar checkers to adopt a progressive approach, in which feedback is given to the user upon successive requests (see Screenshots 1 and 2) and presents the most developed feedback messages.

We ran the grammar checker on roughly 7200 words of learner English, taken from the Japanese and Chinese components of the *International Corpus of Learner English* [18]. These corpora were selected because their error rate was higher than that of the French component, which enabled us to collect feedback messages more efficiently. We then recorded the feedback messages issued by the grammar checker in Word 2007, filtering out doubles, without taking into account the accuracy of the error detection and correction process. We did not limit the analysis to grammatical feedback and included punctuation and style feedback in our exploratory corpus of feedback messages.

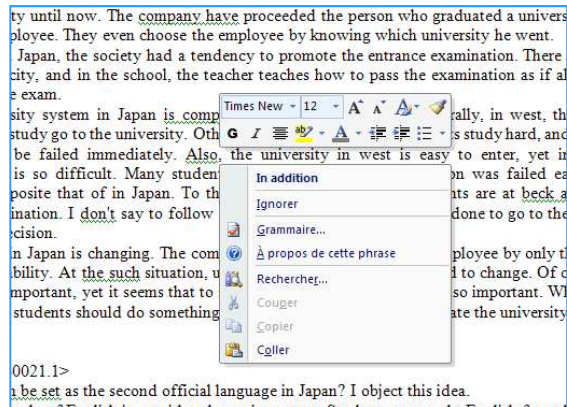


Figure 1. Feedback message given after right click

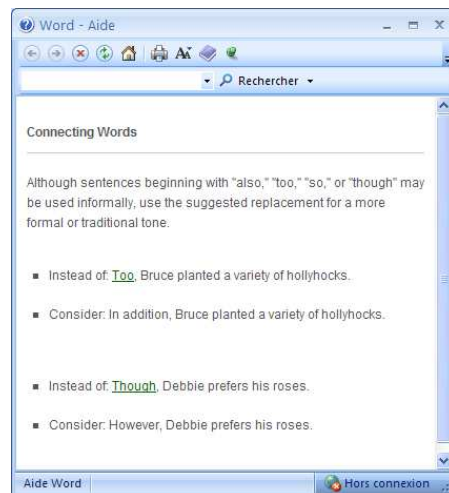


Figure 2. Feedback message given when user requests "Explanation" in previous window

We annotated a list of 36 feedback messages using tags from RST to annotate rhetorical relations, as well as recent research in explanation [2] to annotate the elements of explanation. Despite the wide variation in the messages we were able to collect, they all follow the same superficial pattern, visible in Figure 2:

1. color-coded highlighting of the error in the text
2. heading of the message
3. meta-linguistic feedback
4. illustration

We isolated three rhetorical patterns based on the nature of the nucleus (illustrations are not presented in their full form here, as they are very extensive in feedback messages; a sample can be seen on Figure 2):

1. nucleus has the form of an instruction:

[[summary Colloquialism] [concession Although "a lot of" or "lots of" may be used informally], substitute "many" or "much" [purpose for a more formal or traditional tone] [illustration ...]] (1)

2. nucleus has the form of a grammar/style rule:

[[summary Number Agreement] A noun and the words that modify that noun must agree in number. [elaboration [case 1 "Many" and "few" modify plural nouns.] [case 2 "Much" and "less" modify nouns that cannot be counted or divided [illustration such as "much oil," "less happiness.]]] [case 3 In addition, the phrase "one of" must modify a plural noun.]] [illustration ...]] (2)

3. nucleus has the form of an error diagnosis:

[[summary Extra Word] Only one of the marked words is necessary to signal that a noun follows.[illustration ...]] (3)

Table 1. Rhetorical elements and domain-specific functions

Rhetorical elements	Domain-specific functions	Examples
Nucleus	Instruction	Use the base form of the verb
	Grammar/style rule	A noun and the words that modify that noun must agree in number
	Error diagnosis	The marked word or phrase may be overused or unnecessary to the meaning of your sentence
Summary	Error domain	Number Agreement
	Error diagnosis	Colloquialism
	Error location	Beginning of sentence
Concession	Grammar/style rule	Although "a lot of" or "lots of" may be used informally,
Purpose	Grammar/style advice	To make your sentence easier to read or to signal a pause,
Elaboration	Grammar/style advice	"Kinds of" and "sorts of" are preferable to "kind of" and "sort of"
Circumstance	Grammar/style rule	(the subject performs the action)
	Meta-linguistic information	After certain auxiliary verbs such as "can" or "may,"
Enablement	Instruction	by adding a subject or a verb
Illustration	Linguistic information	such as "an LMN" or "a UVW"
Condition	Description	If the second half of your sentence begins with "then,"

Another possible analysis would be to consider the heading of the feedback as the nucleus, and the rest of the feedback as a structured elaboration of the heading.

Twenty-four of the 36 feedback messages are similar to type 1. Six messages demonstrate the same type of case structure as example (2), regardless of the nature of the nucleus. This presentation of several mutually exclusive cases may arise either from the existence of several possibilities for correction, or from the uncertainty of the system as to the accuracy of the detection and correction processes.

The different rhetorical elements found in feedback messages are listed in Table 1. In addition to their rhetorical function, the elements present in feedback messages very often carry meaning that is essential to their explanatory function, and specific to the domain of error correction. For example, a concession to an instruction can carry advice as to the use of a word in a specific language register (see example (1)). The second column of Table 1 presents the types of domain-specific functions that rhetorical elements can carry.

3.3 Discussion

The analysis of the feedback messages provided by a grammar checker led us to formulate several questions concerning explanation and automatically generated CF, to which we offer tentative answers in the next paragraphs:

- What is the request that explanation is expected to answer in feedback messages?
- What constitutes an explanation in feedback messages? Do all feedback messages provide explanations?
- What are the elements that should be included in feedback messages for them to be as effective as possible?
- What type of knowledge is required from the user of feedback messages?
- What are the constraints to take into account when designing feedback messages for language learners?

These questions may sound trivial, but they are actually at the root of CF generation; trying to generate feedback without having obtained clear answers to these questions is unlikely to lead to sound results.

In our analysis, we assume that explanation in feedback messages should at least provide information as to why the segment was flagged as erroneous in the text. This step is what we call "error diagnosis". It is performed in several steps in the grammar checking component of Word 2007. First, the erroneous segment is underlined using a color code that already issues information as to the type of the error. Secondly, a right click will call up a menu providing the user with a correction and more information concerning the error (a short string of words that will serve as the title of the feedback message). There are inconsistencies in the type of information given by Word's grammar checker at this point, as it may deal with error domain (i.e. the grammatical or stylistic issue at stake, e.g. "Subject-Verb Agreement" or "Comma Use"), error location (e.g. "Beginning of Sentence"), or error diagnosis (e.g. "Cliché").

Twenty-eight of the 36 feedback messages were found to provide explanations, but the wide structural variety in messages and their lack of clear explanatory protocol make it difficult to clearly isolate explanations from instructions, since meta-linguistic

feedback in the form of grammar/style rules or advice is often embedded in instructions, as visible in example (4):

[[[summary (+error domain) Passive Voice]
[purpose (+style advice) For a livelier and more persuasive sentence,] nucleus
(+instruction) consider rewriting your sentence using an active verb [elaboration
(+grammar rule) (the subject performs the action, [illustration (+linguistic information) as
in "The ball hit Catherine"])] rather than a passive verb [elaboration (+grammar rule)
(the subject receives the action, [illustration (+linguistic information) as in "Catherine
was hit by the ball"])]]. [[condition (+description) If you rewrite with an active
verb,] nucleus (+instruction) consider what the appropriate subject is – [elaboration
(+meta-linguistic information) "they," "we," or a more specific noun or pronoun.]
[illustration ...]]] (4)

The design of feedback messages should follow a clear explanation protocol, with well-defined steps, in order to avoid inconsistencies in the way errors are processed, and ensure that the necessary steps are present. The five steps in the protocol we identified are as follows:

1. **Error marking:** error is highlighted;
2. **Error diagnosis:** possible mention of error type, description of the erroneous segment, information as to the nature and/or causes of the error;
3. **Meta-linguistic feedback:** exposition of the relevant grammar/style rules;
4. **Remediation:** instructions enabling the user to successfully correct the segment;
5. **Illustrations:** examples of the correct use the grammar/style rules in question.

The explanatory elements of this protocol are steps 2 and 3, error diagnosis and meta-linguistic feedback. CF messages containing only steps 1, 4 and 5, as in some samples from our corpus, do not provide an explanation. See for example the following feedback message:

[summary (+error domain) Verb Form]
[[circumstance After certain auxiliary verbs such as "can" or "may,"] nucleus
(+instruction) use the base form of the verb.] [illustration ...] (5)

However, in some of the cases where no explanation was found in feedback messages, it is difficult to imagine what explanation could have been provided. These often deal with punctuation errors, a type of error linked to the knowledge of typographic conventions. Those non-explanatory messages rely on the assumption that the user of the grammar checker knows about punctuation conventions. From what we have observed in our corpus of feedback messages, users are required to possess a certain amount of meta-linguistic knowledge to benefit from the use of feedback messages. Following is a non-exhaustive list of those areas of knowledge:

- knowledge of the different grammatical categories and superficial understanding of the morpho-syntactic relations that can exist between them (e.g. subject-verb agreement, number agreement);
- superficial understanding of the links between syntax, pragmatics and semantics (e.g. use of passive voice, question formation);

- awareness of punctuation conventions and of the necessity to respect them;
- awareness of language registers and of the appropriateness of their use;
- awareness of writing requirements (e.g. concision, clarity, argumentative force).

In this section, we have discussed explanation and CF without distinguishing CF targeted at learners from CF targeted at native speakers. As we have seen in section 2, SLA and CALL research offer partial conclusions as to the most efficient feedback types and uses when addressing learners' needs. The following are some of the guidelines we have been able to sketch:

- feedback messages should use highlighting and meta-linguistic feedback [15], i.e. they should include steps 1, 2 and 3 of our protocol;
- feedback messages should not provide a correction of the error [15];
- feedback messages should not posit meta-linguistic knowledge that does not correspond to the learners' level, and/or be adjustable to their needs, preferences and level;
- meta-linguistic feedback should not be of excessive length, or it may not be read [19];
- metalinguistic feedback should be absolutely accurate and not resort to "quick fixes";
- the formulation of messages should not be detrimental to learning by triggering negative feelings about the learning tool or lowering self-confidence and motivation of the learner (for example with the use of messages like: "This is wrong").

4 Conclusion

The objective of the survey presented in this paper was to analyze the feedback messages provided by a widely used grammar checker for English, in order to study the form explanation can take in this context, and to what extent it is used. Even though the form of these feedback messages varies widely, we isolated three patterns of rhetorical structures, depending on the nature of the nucleus. Moreover, we identified a non-exhaustive set of 9 rhetorical elements, which often carry domain-specific information that support the explanatory function of these messages. We pointed out inconsistencies in the design of these feedback messages, in their heading and structure, as well as in the way they integrate explanation and meta-linguistic feedback. Drawing from SLA and CALL research, we proposed guidelines for the creation of effective feedback messages, following a five-step protocol including error marking, error diagnosis, meta-linguistic feedback, error remediation and illustrations. This protocol needs to be tested and enriched in order to guarantee its relevance and applicability. It will consequently be used for the creation of dynamic feedback in a grammar checker/learning tool that is adjustable to user profile (English user vs. learner) and the type of document produced.

References

1. Mann, W. C., Thompson, S. A.: Rhetorical Structure Theory: Towards a Functional Theory of Text Organization. *Text - Interdisciplinary Journal for the Study of Discourse*. vol. 8, issue 3, 243-281 (1988)
2. Bourse, S., Saint-Dizier, P.: The Language of Explanation Dedicated to Technical Documents (under submission)
3. Ellis, R.: *The Study of Second Language Acquisition*. 2nd edition. Oxford University Press, Oxford (2008)
4. Ellis, R.: Corrective Feedback in Theory, Research, and Practice. Presentation at the 5th International Conference on ELT in China, Beijing (2007)
5. Lyster, R., Ranta, L.: Corrective Feedback and Learner Uptake: Negotiation of Form in Communicative Classrooms. *Studies in Second Language Acquisition* 19, 37-66 (1997)
6. Truscott, J.: The Case Against Grammar Correction in L2 Writing Classes. *Language Learning*, 46, 327-369 (1996)
7. Ferris, D.: The Case for Grammar Correction in L2 Writing Classes: A response to Truscott (1996). *Journal of Second Language Writing*, vol. 8, issue 1, 1-11 (1999)
8. Bitchener, J.: Evidence in Support of Written Corrective Feedback. *Journal of Second Language Writing*, vol. 17, issue 2, 102-118 (2008)
9. Ellis, R., Sheen, Y., Murakami, M., Takashima, H.: The Effect of Focused and Unfocused Corrective Feedback in an English as a Foreign Language Context, *System* 36, 353-371 (2008)
10. Ferris, D., Roberts, B.: Error Feedback in L2 Writing Classes: How Explicit Does It Need To Be? *Journal of Second Language Writing* 10, 161-184 (2001)
11. Brandl, K.: Strong and Weak Students' Preferences for Error Feedback Options and Responses. *The Modern Language Journal*, vol. 79, issue 1, 194-211 (1995)
12. Schmidt, R.: The Role of Consciousness in Second Language Learning. *Applied Linguistics* 11, 129-158 (1990)
13. Gass, S.: *Input, Interaction and the Second Language Learner*. Lawrence Erlbaum, Mahwah, N.J. (1997)
14. Heift, T.: Corrective Feedback and Learner Uptake in CALL. *ReCALL*, vol. 16, issue 2, 416-431 (2004)
15. Heift, T., Schulze, M.: *Errors and Intelligence in Computer-Assisted Language Learning: Parsers and Pedagogues*. Routledge, New York (2007)
16. Potter, R., Fuller, D.: My New Teaching Partner? Using the Grammar Checker in Writing Instruction. *English Journal*, vol. 98, issue 1, 38-41 (2008).
17. Burston, J.: Exploiting the Potential of a Computer-Based Grammar Checker in Conjunction with Self-Monitoring Strategies with Advanced Level Students of French. *CALICO Journal*, vol. 18, issue 3, 499-515 (2001)
18. Granger, S., Dagneaux, E., Meunier, F., Paquot, M.: *International Corpus of Learner English*, v.2. Presses Universitaires de Louvain, Louvain-la-Neuve (2009)
19. Heift, T.: Error-Specific and Individualized Feedback in a Web-based Language Tutoring System: Do They Read It? *ReCALL*, vol. 13, issue 1, 99-109 (2001)

Author Index

A

Agne, Stefan	11
Aha, David	21

D

Decker, Hendrik	71
Dodson, Thomas	1

F

Forcher, Björn	11
----------------	----

G

Garnier, Marie	81
Goldsmith, Judy	1

J

Jonker, Catholijn	61
-------------------	----

K

Klenk, Matthew	31
Kuter, Ugur	21, 31

M

Mattei, Nicholas	1
Molineaux, Matthew	21, 31

O

Overton, James	41
----------------	----

R

Roth-Berghofer, Thomas	11
------------------------	----

S

Schulz, Christian	51
Sonntag, Daniel	51

V

Van De Kieft, Iris	61
Van Riemsdijk, M. Birna	61