On Repairing Reasoning Reversals via Representational Refinements *

Alan Bundy, Fiona McNeill and **Chris Walton**
A.Bundy@ed.ac.uk, F.J.McNeill@sms.ed.ac.uk, C.D.Walton@ed.ac.uk
School of Informatics, University of Edinburgh, EH8 9LE, Scotland

Abstract

Representation is a fluent. A mismatch between the real world and an agent's representation of it can be signalled by unexpected failures (or successes) of the agent's reasoning. The 'real world' may include the ontologies of other agents. Such mismatches can be repaired by refining or abstracting an agent's ontology. These refinements or abstractions may not be limited to changes of belief, but may also change the *signature* of the agent's ontology. We describe the implementation and successful evaluation of these ideas in the ORS system. ORS diagnoses failures in plan execution and then repairs the faulty ontologies. Our automated approach to dynamic ontology repair has been designed specifically to address real issues in multi-agent systems, for instance, as envisaged in the Semantic Web.

Introduction

The first author [AB] has a vivid memory of his introductory applied mathematics lecture during his first year at university. The lecturer delivered a sermon designed to rid the incoming students of a heresy. This heresy was to entertain a vision of a complete mathematical model of the world. The lecturer correctly prophesied that the students were dissatisfied with the patent inadequacies of the mathematical models they had learnt at school and impatient, now they had arrived in the adult university world, to learn about sophisticated models that were free of caveats such as *treating the weight of the string as negligible* or *ignoring the friction of the pulley*.

They were to be disappointed. Complete mathematical models of the real world were unattainable, because it was infinitely rich. Deciding which elements of the world were to be modelled and which could be safely ignored was the *very essence* of applied mathematics. It was a skill that students had to learn – not one that they could render redundant by modelling everything.

This all now seems obvious. AB is surprised at the naivety of his younger self — since, before the sermon, he certainly was guilty of this very heresy. But it seems this lesson needs to be constantly relearnt by the AI community. We too model the real world, for instance, with symbolic representations of common-sense knowledge. We too become impatient with the inadequacies of our models and strive to enrich them. We too dream of a complete model of common-sense knowledge — and even aim to implement such a model, *cf.* the Cyc Project[1]. But even Cycorp is learning to cure itself of this heresy, by tailoring particular knowledge bases to particular applications, underpinned by a common core.

If we accept the need to free ourselves of this heresy and accept that knowledge bases only need to be good enough for their application, then there is a corollary that we must also accept: the need for the knowledge in them to be fluent, i.e., to change during its use. And, of course, we *do* accept this corollary. We build adaptive systems that learn to tailor their behaviour to a user or improve their capabilities over time. We have belief-revision mechanisms, such as truth maintenance (Doyle 1979), that add and remove knowledge from the knowledge base.

However, it is the thesis of this paper that none of this goes far enough. In addition, we must consider the dynamic evolution of the *underlying formalism* in which the knowledge is represented. To be concrete, in a logic-based representation the predicates and functions, their arities and their types, may all need to change during the course of reasoning.

Once you start looking, human common-sense reasoning is full of examples of this requirement. Consider, for instance, Joseph Black's discovery of latent heat. Before Black, the concepts of heat and temperature were conflated. It was thus a paradox that a liquid could change heat content, but not temperature, as it converted to a solid or a gas. Before formulating his theory of latent heat, Black had to separate these two conflated concepts to remove the paradox (Wiser & Carey 1983). Representational repair can also move in the opposite direction: the conflation of "morning star" and "evening star" into "Venus", being one of the most famous examples.

But such representational refinement is not a rare event reserved to highly creative individuals; it's a commonplace occurrence for all of us. Everyday we form new models to describe current situations and solve new problems: from

*The research reported in this paper was supported by EPSRC grant GR/S01771, the EU Open Knowledge project and an EPSRC studentship to the second author. We are grateful to an anonymous FLAIRS-06 referee for helpful comments on an earlier draft.

[1] http://www.cyc.com/

making travel plans to understanding relationships with and between newly met people. These models undergo constant refinement as we learn more about the situations and get deeper into the problems.

Consider, for instance, the commonplace experience of buying something from a coin-in-the-slot machine. Suppose the item to be bought costs £2. Initially, we may believe that having £2 in cash is a sufficient precondition for the buying action. However, we soon learn to refine that precondition to having £2 *in coins* — the machine does not take notes. When we try to use the coins we have, we must refine further to exclude the new 50p coins — the machine is old and has not yet been updated to the new coin. But even some of the, apparently legitimate, coins we have are rejected. Perhaps they are too worn to be recognised by the machine. Later a friend shows us that this machine will also accept some foreign coins, which, apparently, it confuses with British ones. Refining our preconditions to adapt them to the real world of this machine does not just involve a change of belief. We have to represent new concepts: "coins excluding the new 50p", "coins that are not too worn to be accepted by this particular machine", "foreign coins that will fool this machine", etc.

As another example, consider the experiment conducted by Andreas diSessa on first-year MIT physics students (diSessa 1983). The students were asked to imagine a situation in which a ball is dropped from a height onto the floor. Initially, the ball has potential but not kinetic energy. Just before it hits the floor it has kinetic but not potential energy. As it hits the floor it has neither. Where did the energy go? The students had trouble answering this question because they had idealised the ball as a particle with mass but no extent. To solve the problem they had to refine their representation to give the ball extent, so that the 'missing' energy could be stored in the deformation of the ball. Note that this requires a change in the *representation* of the ball, not just a change of belief about it.

The investigation of representational refinement has become especially urgent due to the demand for autonomous, interacting software agents, such as is envisaged in the Semantic Web (Berners-Lee, Hendler, & Lassila 2001). To enable such interaction it is assumed that the agents will share a common ontology. However, any experienced programmer knows that perfect ontological agreement between very large numbers of independently developed agents is unattainable. Even if all the ontology developers download their ontologies from the same server, they will do so at different times and get slightly different versions of the ontology. They will then tweak the initial ontologies to make them better suited to their particular application. We might safely assume a $\sim$90% agreement between any two agents, but there will always be that $\sim$10% disagreement — and it will be a different 10% for each pair. The technology we discuss below provides a partial solution to just this problem.

Note that our proposal contrasts with previous approaches to ontology mapping, merging or aligning[2]. Our mechanism does not assume complete access to all the ontologies whose mismatches are to be resolved. Indeed, we argue that complete access will often be unattainable for commericial or technical reasons, e.g., because the ontologies are being generated dynamically. Moreover, our mechanism doesn't require an off-line alignment of these mismatching ontologies. Rather, it tries to resolve the mismatches in a piecemeal fashion, as they arise and with limited, run-time interaction between the ontology owning agents. It patches the ontologies only as much as is required to allow the agent interaction to continue successfully. Our mechanism is aimed at ontologies that are largely in agreement, e.g., different versions of the same ontology, rather than aligning ontologies with a completely different pedigree, which is the normal aim of conventional ontology mapping. It works completely automatically. This is essential to enable interacting agents to resolve their ontological discrepancies during run-time interactions. Again, this contrasts with conventional ontology mapping, which often requires human interaction.

Ontologies

Wikipedia defines[3] an ontology as:

> *"the product of an attempt to formulate an exhaustive and rigorous conceptual schema about a domain"*

We will divide each ontology into two parts: the signature and the theory. By *signature* we will mean: the type hierarchy, the names of the predicates and functions, and the types or arities of these predicates and functions. By *theory* we will mean the definitions, axioms, inference rules and theorems that can be written in the language defined by the signature. The fields of belief revision[4] and truth maintenance (Doyle 1979) describe mechanisms for repairing the theory. While theory repair plays a role in our work, our main focus is on *signature repair*.

Our ontologies are written in a restricted version of KIF: the Knowledge Interchange Format: "a syntax for first order logic that is easy for computers to process[5]", which also has facilities for meta-level reasoning. Our restrictions are to disallow complex class definitions and to quantify only over finite sets. A variety of KIF ontologies are available from the Ontolingua Server[6]. We chose KIF because it provides a rich language on which to test our ideas and is in widespread use with many example ontologies on which to test our ideas. Our future plans include the transfer of our techniques to OWL[7], which has been adopted by W3C as the standard for representing ontologies in the Semantic Web.

A fault in an ontology may be detected when an inference unexpectedly fails (or sometimes unexpectedly succeeds). It

[2] §4 of the "OWL Web Ontology Language Guide" http://www.w3.org/TR/owl-guide/

[3] http://en.wikipedia.org/wiki/Ontology_\%28computer_science\%29

[4] See §4.3 of the "Stanford Encyclopedia of Philosophy" http://plato.stanford.edu/entries/reasoning-defeasible/\#4.3.

[5] http://en.wikipedia.org/wiki/KIF

[6] http://www.ksl.stanford.edu/software/ontolingua/

[7] http://en.wikipedia.org/wiki/Web_Ontology_Language

is then necessary to diagnose the fault and to repair it. Repair can involve either refinement or abstraction, or a mixture of both. The repair may be carried out on either the theory or the signature, or both.

Abstraction *vs* Refinement

A common technique in automated reasoning is the use of abstraction (see (Giunchiglia & Walsh 1992) for a survey and Figure 1 for a diagrammatic representation). A problem to be solved is first abstracted to a simpler problem. This simpler problem is solved and its solution is then refined back and patched to be a solution to the original problem. This abstraction process can be nested recursively.

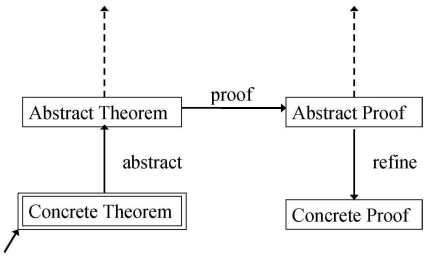


Figure 1: The use of abstraction in problem solving

We propose turning this mechanism on its head. The initial representation of a problem is simple and, thus, easily solved. As we come to understand the problem and the environment more deeply, and as we attempt to apply our simple solution, we discover flaws in our representation of the problem and the environment. Our simple solution fails and must be repaired. We refine the initial, over-simple representations to obtain a more sophisticated solution that addresses the previous failures (see Figure 2). Again, this process can be nested recursively.

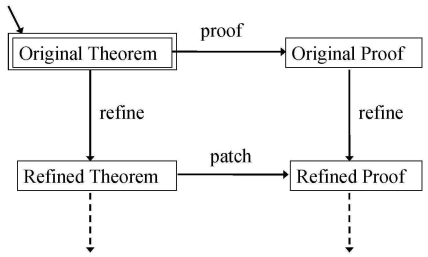


Figure 2: The use of refinement in problem solving

In (Giunchiglia & Walsh 1992), Giunchiglia and Walsh categorise the various kinds of abstraction operation they found in a wide-ranging survey. By inverting each abstraction operation, a corresponding refinement operation is suggested. We use operations from both these categorisations to repair ontologies.

Giunchiglia and Walsh's categories are as follows:

Predicate: Two or more predicates are merged, typically to the least general generalisation in the predicate type hierarchy, e.g.,

(Bottle ?X) + (Cup ?X) $\mapsto$ *(Container ?X)*.

Domain: Two or more terms are merged, typically by moving the functions to the least general generalisation in the domain type hierarchy, e.g.,

(Aunt Me) + (Cousin Me) $\mapsto$ *(Relation Me)*.

Propositional: One or more arguments are dropped, e.g.,

(Bottle A) + (Bottle B) $\mapsto$ *(Bottle)*.

Precondition: The precondition of a rule is dropped, e.g.,

[(Ticket ?X) $\rightarrow$ *(Travel ?X)]* $\mapsto$ *(Travel ?X)*.

Note that the first three abstractions modify the signature but the last one modifies the theory.

These four abstraction operations can be inverted to suggest the following four refinements:

Predicate: One predicate is separated into two or more, e.g.,

(Container ?X) $\mapsto$ *(Bottle ?X) + (Cup ?X)*.

Domain: One term is separated into two or more, e.g.,

(Relation Me) $\mapsto$ *(Aunt Me) + (Cousin Me)* .

Propositional: One or more arguments are added, e.g.,

(Bottle) $\mapsto$ *(Bottle A) + (Bottle B)* .

Precondition: A new precondition is added to a rule, e.g.,

(Travel ?X) $\mapsto$ *[(Ticket ?X)* $\rightarrow$ *(Travel ?X)]* .

Note that applying refinements is inherently more complex than applying abstractions. In abstraction, information is removed, but in refinement it is added. This additional information must be constructed, inferred or guessed.

Between abstraction and refinement lies **Argument Permutation**. Here the arguments of a predicate or function are reordered, e.g., *(Is_Route ?X ?Y)* $\mapsto$ *(Is_Route ?Y ?X)*.

The above abstractions and refinements are at the heart of our dynamic ontology repair technique. They constitute a classification of ontological mismatch. In particular, we intend this classification to be complete for signature mismatches, although we are always interested to learn of signature mismatches we have overlooked. This classification forms the basis both for the diagnosis of faulty ontologies and of tactics for patching them.

Agents

Our mechanism for dynamic ontology repair is intended to work in a domain of automated, peer-to-peer agents. For instance, below we give an implemented, worked example in which agents for paper publication, conference registration, hotel accomodation and paper-format conversion interact to assist the author of a paper to attend a conference.

The underlying motivation for our work is that the peer-to-peer agent domain raises requirements that are not addressed by conventional ontology mapping mechanisms. In particular, there is a requirement for run-time, totally automated, good-enough alignment between agents with limited access to each other's ontologies.

Our current prototype system, ORS, does not fully realise a peer-to-peer architecture, due to various simplifications we have made in the initial implementation. We plan to incrementally address these simplifications in subsequent implementations. These current simplifications include various assumptions about agents, including that:

- agents are all plan formation and execution softbots with a simple query/reply performative language;
- currently only one agent, the *planning agent*, forms plans; the other agents merely assist in the execution of these plans;
- agents are honest and knowledgeable;
- agents will perform tasks requested of them, if possible;
- agents have broadly similar ontologies;
- agents other than the planning agent, are always right;
- agents are prepared to answer yes/no and wh*[8] questions;
- but agents will not reveal much detail about their ontologies.

In particular, note that we assume that an agent will not transmit its whole ontology in order to enable the planning agent to align its ontology with it. We think it is unrealistic to expect that most agents will be designed with the functionality to download their ontologies. Moreover, ontological information will often be confidential, e.g., a business asset, that the agents' owners will not be prepared to make public. Additionally, some ontologies are virtual, i.e., they are generated dynamically in response to requests, e.g., RSS feeds. Thus conventional ontology mapping is not a solution to the problem posed above, since most such mapping mechanisms *do* assume complete access to all the mismatching ontologies. We do, however, assume that agents will be able to answer simple yes/no and wh* queries about propositions that can be derived from their ontologies, e.g., propositions that might be preconditions of action rules. We thought this was the weakest assumption we could make about external agent functionality, while supporting ORS in its fault diagnosis. Inter-agent question-answering exchanges take place during both plan execution and fault diagnosis.

The ORS System

Our system is called ORS, which stands for Ontology Refinement System. An overview of ORS is given in Figure 3.

Communication between agents consists of the exchange of query and reply performatives. These take the form:

$$query(Sender, Receiver, QueryType, Query)$$
$$reply(Sender, Receiver, QueryType, Answer)$$

where a $Query$ is a KIF formulae and an $Answer$ is either a truth value or variable instantiations, depending whether the $QueryType$ is a yes/no question or wh*-question, i.e., wh*-questions have $Query$s with uninstantiated variables and the $Answer$ instantiates these variables. For instance, the planning agent might ask a travel agent the cost of a plane flight from Edinburgh to Miami with the $Query$, *(Flight Edinburgh Miami ?Cost)* and the travel agent might reply with the $Answer$ $?Cost = 300$. If there is no flight from Edinburgh to Miami then the $Answer$ would be $False$. This inter-agent communication is currently provided by a simple blackboard architecture implemented in Prolog-Linda[9]. As we move towards a fully peer-to-peer implementation of ORS, this blackboard architecture will be replaced.

The planning agent has two ontologies: object-level and meta-level. The object-level ontology contains its model of the world including its model of other agents, with action rules corresponding to the services they provide. The meta-level ontology contains information about the predicates, functions, rules, constants and variables of the object-level ontology. In particular, it contains a mapping of services to the external agents that it believes provide these services. By "ontology" we will usually mean the object-level ontology, unless specified otherwise.

Plan Formation, Deconstruction and Analysis

The planning agent uses its ontology to form a plan to achieve its goals. This plan is a sequence of actions, some of which require other agents to perform services on its behalf. The planning agent's ontology contains *action rules*, of the form $premise \Rightarrow conclusion$. The $conclusion$ consists of a prediction of how the world will change when the action is executed and the $premise$ consists of the preconditions under which the action can be successfully applied.

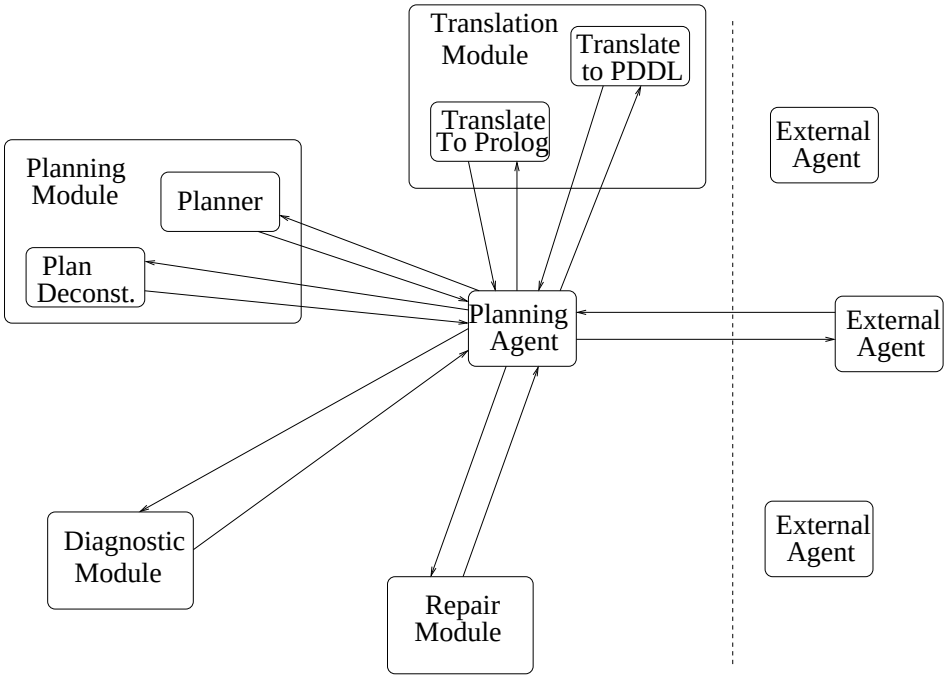
When the planning agent attempts to execute the plan, it may fail. This will be due to some mismatch between the planning agent's ontology and the environment in which the plan is being executed. For instance, the preconditions under which an external agent is prepared to perform a service may not be accurately represented in the $premise$ of the corresponding action rule. In the next section we will see how such ontological faults can be diagnosised and repaired.

To form its plans, the planning agent uses Metric-FF (Hoffmann & Nebel 2001), a state-of-the-art planner that uses PDDL to represent its world models and plans. PDDL is a classical logic with bounded quantification over finite sets, enabling PDDL formulae to be translated into propositional logic. The ORS translation module translates from KIF into PDDL (McNeill, Bundy, & Walton 2004).

The plan is executed by running each of its actions. Most of these actions consist of instructions to external agents to provide services of various kinds. Each external agent either performs the service or informs the planning agent that it is unable to do so. In deciding whether it is able to perform a service, an external agent must check each of the preconditions of *its* action rule for that service. Some of these preconditions can be checked by reference to the external agent's own ontology. We will call these *internal preconditions*. But some of them require communication with the planning agent or other external agents. We will call these

[8] e.g., why?, when?, what?, which?, how?, etc.

[9] `http://www.sics.se/sicstus/docs/latest/html/sicstus.html/Linda-Library.html\#Linda\%20Library`



A vertical dotted line separates the planning agent from the other external agents. The planning agent calls upon modules for planning, diagnosis, repair and translation. The planner forms plans for specified goals, which the planning agent then attempts to execute by passing requests to the external agents. When this fails, the plan deconstructor provides a justification for the original plan, which the diagnostic module uses to try to figure out what went wrong. Diagnosis may require further communication with the external agents. If a fault is detected then the repair module is asked to repair the ontology. A variety of representational formalisms are used by the different modules, so the translation module converts one formalism into another.

Figure 3: An Overview of the ORS System

external preconditions. Thus, the performance of each service may be preceded by a flurry of inter-agent dialogue.

For fault diagnosis, ORS requires a justification of the plan: essentially a proof recording how facts in the ontology derive the preconditions of the plan's actions. Unfortunately, Metric-FF does not provide such a justification. We, therefore, built a plan deconstructor that, given the plan and the ontology, provides a suitable justification (McNeill *et al.* 2003). Since this deconstructor is implemented in Prolog, it uses a Prolog-like representation of the justification. Again the translation module translates KIF into this Prolog representation (McNeill 2005).

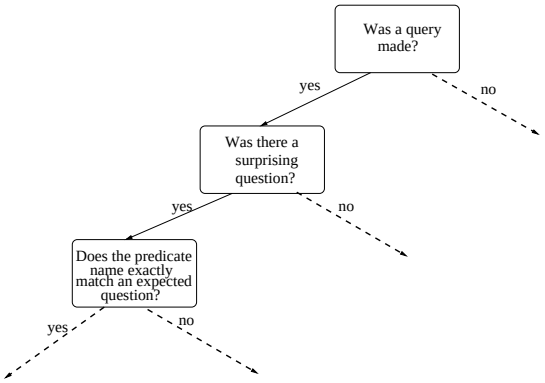
The plan justification is used by the *Shapiro Algorithm*. This algorithm is used in fault diagnosis and repair to discover why two agents have different truth values for the same formula. The Shapiro Algorithm is modelled on the algorithm in (Shapiro 1983), which was designed to debug logic programs. It steps through a justification pursuing the planning agent's false beliefs to their underlying cause. The justification is essentially a proof, in which each goal is supported by the set of subgoals from which it was derived. At each stage the Shapiro Algorithm has identified a false goal and needs to find out which of the supporting subgoals are also false. It poses each of the subgoals, in turn, to the external agent, which it treats as an oracle. The Shapiro Algorithm recurses on those subgoals found to be false. When it reaches a false leaf, this leaf is treated as an underlying cause of falsity. If it was a fact in the original ontology, it is removed, since it is no longer believed to be true; if it came to be believed as an expected effect of a previous action, further investigation into that action rule is necessary.

Fault Diagnosis

Fault diagnosis is at the heart of the ORS system. The various forms of abstraction and refinement define a set of potential repairs, e.g., a predicate needs to be separated into two, an argument must be added or dropped, etc. Fault diagnosis consists of the analysis of the unexpected failure of a service to be performed and tries to identify a repair.

Since the diagnostic module does not have access to a full description of the problem, i.e., since it does not have access to the ontology or inference processes of the external agent, fault diagnosis is inevitably a heuristic process. It is not intended to fix all misalignments between communicating agents. Nor is there any guarantee that the repair will totally remedy the fault. Rather, we hope that the repair will be good enough, perhaps in conjunction with subsequent repairs, to allow the service to be performed.

Fault diagnosis is implemented as a decision tree labelled by queries at the non-terminal nodes and repairs at the leaves. See Figure 4 for part of this decision tree.



The fault diagnosis system traverses the decision tree from the root. At each note it poses a query about the events surrounding the plan execution failure or to the agent who failed to provide a service. The answers to these queries determine the path taken from that node. When it reaches a leaf, some repairs are proposed.

Figure 4: Part of the Fault Diagnosis Decision Tree

When the planning agent asks an external agent to perform a service, both agents will have STRIPS-like action rules that specify the action to be performed, the preconditions under which the external agent will execute this action and the effects of the action. However, these two action rules may differ. Such differences can cause the planning agent's plan to fail. Discovering these differences is the purpose of fault diagnosis.

One of the key concepts in the fault-diagnosis decision tree is that of a *surprising question*. The planning agent *expects* to be queried about each of the external preconditions in its action rule. If the external agent represents one of these external preconditions in a different way to the planning agent, or if it has an external precondition that the planning agent does not have, then the query corresponding to this external precondition will be *surprising* to the planning agent.

The main part of the fault-diagnosing decision tree is summarised below.

1. **No query was made:** i.e., the external agent did not check any external preconditions with the planning agent. Three possible diagnoses are suggested:
   (a) *This external agent is unable to perform services of this kind.* ORS will ask the external agent whether it can perform this service. If not, then the ontological metadata that maps services to agents must be amended.
   If so, then ORS asks the external agent the truth of each of the preconditions of the planning agent's action rule for this service. Two possibilities arise:
   (b) *The external agent says that one of these preconditions is false.* The planning agent's belief is assumed to be wrong. Using the Shapiro Algorithm, the planning agent must discover the underlying cause of its false belief and correct it.
   (c) *The external agent agrees that all of these preconditions are true.* The planning agent's action rule must be missing a precondition, which the external agent believes to be false. In the absence of hints as to what this missing precondition is, ORS can classify the failure but cannot repair it.
2. **A surprising question was asked and its predicate matches that of a rule precondition, although the question and the precondition differ:** It appears that the agents disagree about the exact form of the external precondition. The nature of this mismatch can be inferred by comparing the surprising question with the planning agent's precondition. It will fall into one of the following classes:
   (a) *The arities of the two predicates are different.* Propositional abstraction or refinement is used to align the planning agent's predicate with the external agent's.
   (b) *The types of the predicate's arguments appear to be different, but reordering them enables a match to be made.* Argument permutation is used to reorder the arguments so that they match.
   (c) *The types of the predicate's arguments are different — even after argument permutation.* Domain abstraction or refinement is used to align the types of the planning agent's predicates or functions with the external agent's.
   (d) *There is no signature mismatch.* Just as in case 1b, the two agents disagree on the truth of this precondition. The Shapiro Algorithm is again used to detect the underlying cause of this disagreement and correct it.
3. **A surprising question was asked and its predicate does not match any rule precondition:** Either this question does nevertheless correspond to a planning agent's precondition or the external agent's action rule appears to have an external precondition that the planning agent's doesn't.
   (a) *One of predicates is a subtype of the other.* Predicate abstraction or refinement can be used to force the planning agent's precondition to match that of the external agent.
   (b) *There is no type relation between the predicates.* There is a missing precondition. The surprising question can provide a clue that enables ORS to construct the missing precondition to be added to the planning agent's action rule using precondition refinement.
4. **Only non-surprising questions were asked:** Assume the most recent one caused the action failure. There are two possibilities:
   (a) *The question was a wh*-question and the planning agent instantiated the variable to the wrong value.* ORS can classify this error, but cannot repair it.
   (b) *The question was not a wh*-question and the planning agent gave the wrong truth value.* So the two agents

differ over the truth of this precondition. ORS uses the Shapiro Algorithm to identify and fix the underlying cause of the planning agent's false belief.

Repairing Diagnosed Faults

Fault diagnosis not only identifies a fault in the planning agent's ontology, but when this fault can be repaired, it also identifies the abstraction, refinement or belief revision that implements this repair. For instance, leaf 1b in the decision tree identifies belief revision via the Shapiro Algorithm; leaf 2a identifies either propositional abstraction or refinement; while leaf 2c identifies either domain abstraction or refinement.

In the case of abstraction, implementing the repair is usually unproblematic. But, as mentioned above, refinement usually requires constructing, inferring or guessing missing information. Sometimes the context of the diagnosis suggests the form of the missing information, such as in leaf 3b. Sometimes it does not, such as in leaf 1c. Sometimes, in the absence of useful contextual information, ORS resorts to using default values to fill-in gaps. These mechanisms are illustrated in the next §.

A Worked Example

The following worked example is intended to illustrate the ORS system as it creates and executes plans, detects and diagnoses failure, repairs its ontologies, then replans recursively until it either constructs a plan that executes successfully or it fails. The domain concerns the submission of the camera-ready copy of an accepted paper to a conference, then making plans to attend the conference and claim expenses. It uses a peer-to-peer, multi-agent system. Various agents assist the paper writer to submit the paper and attend the conference. They interact together to achieve the various goals this entails.

In this example, the planning agent acts as the paper writer's personal assistant. The plan includes steps in which the other agents perform various services, e.g., accept the submitted paper. If the planning agent's ontology were an accurate model of the world, then it would be bound to succeed. So a failure during plan execution signals an ontology failure. For instance, the submission agent might refuse to accept the paper. The traveller must then diagnose the fault in its ontology and repair it. A new plan for the goal is then derived and executed. This new plan may also fail, triggering a further round of diagnosis and repair. This process recurses until either a plan is derived that successfully executes or the planning agent is unable to diagnose or to repair a fault.

The agents involved are as follows:

Planning Agent: which is responsible for forming and executing the overall plan.

Publication Agent: through which camera-ready copy can be sent for publication in the proceedings.

Registration Agent: which can register conference attendees.

Accommodation Agent: which can book accommodation for a conference delegate.

Paper Conversion Agent: which is able to convert papers into different formats.

The planning agent forms the following plan from its initial ontology:
(SendPaper Researcher My-Paper.ps Ai-Conf),
(Register Researcher Ai-Conf Registration-Cost),
(BookAccom Researcher Ai-Conf Accommodation-Cost).

The first action, *SendPaper*, fails to be executed. There was some questioning before the failure occurred, which included a query from the publication agent: *(Class My-Paper.ps Pdf-Paper)*, which was surprising. The planning agent had believed that papers could be sent in any format, and thus attempted to send a *ps* paper, but it seems that any paper submitted must be of class *Pdf-paper*. The planning agent consequently uses precondition refinement to add a precondition to the *SendPaper* action rule that the paper must be in *pdf* format. It now plans again. Because of this additional precondition, the new plan requires an additional action, *Convert-Paper* to convert the original *ps* paper into *pdf*. The new plan is:
(Convert-Paper Researcher My-Paper.ps My-Paper.pdf)
(SendPaper Researcher My-Paper.pdf Ai-Conf),
(Register Researcher Ai-Conf Registration-Cost),
(BookAccom Researcher Ai-Conf Accommodation-Cost).

The first and second action are now executed successfully. The third action, *Register*, fails, following a surprising question: *(Money PA Dollars ?Amount)*, where *PA* is the planning agent. The predicate *Money* matches a precondition that *PA* was expected to be asked about: *(Money PA 100)*; however, the surprising question has an extra argument concerning currency. The planning agent discovers what the class of this extra argument is (if it does not currently have this information, it can question the registration agent) and performs propositional refinement. Every instance of this predicate must also be changed. It is possible to use heuristics to guess what this value should be: for example, if the conference is held in the USA, then the default value for currency may be US dollars.

After this change has been made, the planning agent replans to form the following plan:
(Register Researcher Ai-Conf Dollars Registration-Cost),
(BookAccom Researcher Ai-Conf Dollars Accommodation-Cost).
These actions are performed successfully.

This worked example has been implemented in the ORS system and successfully executed.

Evaluation

We evaluated the ORS system by testing it on a number of ontologies. To be tested, an ontology had to meet two main requirements:

1. There had to be successive versions of the ontology, so that we could show that ORS could deal with the kinds of ontological revisions that arise in real-world examples. The task of ORS was to repair an earlier version of the ontology into a later version.

2. It had to be possible to pose multi-agent planning problems using the ontology. This enabled ORS to employ its machinery for fault diagnosis and repair.

These criteria were quite difficult to meet. To meet requirement 1 we had to talk to ontology developers, but not all developers stored historical versions of their ontologies. Ontologies that already meet requirement 2 are rare: the worlds of ontology development and plan formation are mostly disjoint. So, we had either to retrofit a planning scenario into the ontology, e.g., by providing action rules etc. as additional material, or by translating knowledge-bases developed for planning applications into KIF ontologies. Both of these adaptations were time consuming, so some of our evaluation was done by hand examination of the differences between successive ontologies to analyse whether these differences could, in principle, be handled by ORS.

We found suitable evaluation material in the following ontologies:

PSL: The Process Specification Language was developed by the Manufacturing Systems Integration Division of the National Institute of Standards and Technology. It defines a neutral representation for manufacturing processes.

AKT: An ontology of academic research in Computer Science developed by the Advanced Knowledge Technologies interdisciplinary research collaboration in the UK.

SUMO: The Suggested Upper Merged Ontology provides a generic structure and general concepts upon which more specific ontologies can be developed. It was developed within the Standard Upper Ontology Working Group.

Planet: Three planning knowledge bases from the PLANET plan formation network, converted to KIF ontologies.

For each of these ontologies we collected two or more versions of them and identified the differences between successive versions. Each such difference was regarded as a data-point for our evaluation. Let `before` and `after` be the differing parts of the earlier and later versions of the ontology, respectively. There might be several such differences between successive versions of an ontology: each one was considered separately. Altogether, we identified 325 such data-points. The question addressed in the evaluation was:

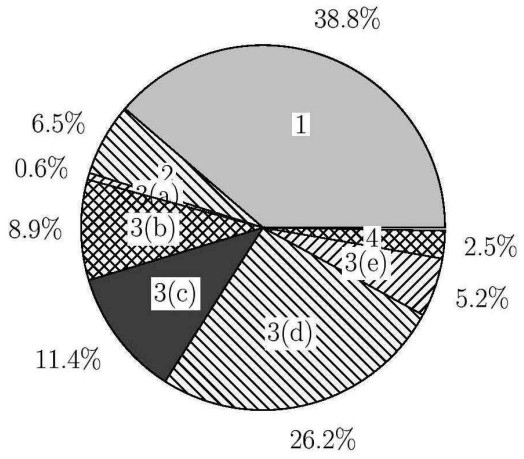
Could ORS *automatically correct* `before` *to* `after`*, given an appropriate goal for which to plan?*

The correction was usually effected by a single refinement or abstraction.

We classified the results of our analysis as follows.

1. ORS could refine the mismatch;
2. ORS could not currently refine the mismatch, but non-major changes to the system would allow ORS to refine it;
3. ORS could not refine the mismatch. This was because:
   (a) ORS did not have sufficient functionality;
   (b) This particular mismatch was outside the scope of the project;
   (c) This mismatch was irrelevant to an automated system — this was usually a change to commenting or formatting;
   (d) This mismatch could not occur in the restricted KIF that ORS was designed to use;
   (e) This mismatch could not be represented in a planning context.
4. The information we had about the mismatch was insufficient to diagnose which of the above categories it would fall into.

The proportions of results falling within each of these categories from our analysis of the above ontologies is represented as a pie-chart in Figure 5.



Each segment of the pie chart represents and is labelled by one of the mismatch categories defined in the above classification. Its size represents the proportion of examples of that kind of mismatch found in the ontologies we analysed. It will be seen that over a third (38.8%) of the examples can be handled by ORS *in its current early stage of development (category 1) and a further 6.5% are within its scope (category 2). The rest are outside its scope for a variety of reasons (category 3) or we have insufficient information to tell (category 4).*

Figure 5: Results of the Evaluation of ORS

Related Work

Most existing ontology mapping techniques have a large static element, require complete access to the ontologies whose mismatches are to be resolved and work only with classification hierarchies. So they cannot provide the functionality required to solve the problem identified in this paper. The ORS system implements an ontology repair mechanism, for a large fragment of first-order logic, that is dynamic, has only limited access to the mismatching ontologies and is entirely automatic. On the other hand, we have to assume that the mismatching ontologies are largely in agreement.

Belief revision is part of what is required in ontology repair, but changes to the underlying ontological representation are also required. In fact, most of ORS's repair operations and repairs are changes to the underlying representations, i.e., predicate, domain or propositional refinement or abstraction.

The work of Pease, Colton, Smaill and Lee (Pease *et al.* 2004) on implementing Lakatos's methods of proofs and refutations (Lakatos 1976) is also aimed at the problem of ontological repair. They use Colton's HR system to discover conjectures that are nearly always true and a multi-agent system interacts to modify the ontology to make these conjectures into theorems. This is done by analysing the counter-examples or failed proofs then adjusting the definitions of predicates, functions and data-structures to either repair the failed proofs, adjust the conjectures or remove the counter-examples. This work differs from ORS in two main respects. Firstly, the trigger for ontological repair is a nearly-true conjecture, rather than a failed plan. Secondly, the repair is to adapt some definitions in terms of fixed underlying primitives, rather than to change the syntactic structure of the ontology's signature. There is undoubtably scope for combining these complementary approaches.

The problem of resolving conflicts between initially identical representations that have diverged is also addressed by the field of optimistic replication (Saito & Shapiro 2003). Replicated data often diverges from the original source due to updating, and this problem is commonly addressed by blocking access to a replica until it is provably up to date; however, optimistic replication allows such changes to occur asynchronously, discovers conflicts after they happen and then attempts to resolve them. These conflicts are indicated by the violation of a conflict detecting precondition, which must be given, by the user or automatically, when the update is made. This differs from our approach, where we assume that updates will be made without thought as to how they may conflict with a different version and conflicts are detected when communication failure between agents occurs, indicating some underlying mismatch. Because the mismatch occurs between agents that can communicate and reason about potential differences, we then have a rich source of information that can be used in diagnosis and refinement which is not available in optimistic replication, where conflicts must usually be resolved by being passed back to the user.

Conclusion

We have argued that representation is a fluent in commonsense reasoning, and that repairs to and evolution of representation is an everyday event. If intelligent agents are to conduct commonsense reasoning, it will be necessary to build automated reasoning systems in which the representation can evolve dynamically in reaction to unexpected failures and successes of reasoning, and to other triggers yet to be explored. In particular, such functionality will be an essential ingredient in interacting, peer-to-peer multi-agent systems, such as are envisaged in the Semantic Web. Agents will need to be able to cope dynamically with minor variations between their ontologies.

We have initiated research into automatic dynamic ontology evolution. The ORS system operates in a planning domain over KIF ontologies, where failures in plan execution trigger repairs to the planning agent's ontology. Despite the simplifying assumptions and limited functionality of this prototype system, it can account for over a third of the ontological mismatches between different versions of several popular and widely used ontologies. More details about this work can be found in (McNeill 2005).

Further work is clearly required to lift the current limitations of this work: removing the simplifying assumptions about agents; extending it to other kinds of ontology such as those based on description logics; extending the kinds of mismatch and repair triggers it can deal with; applying it to non-planning domains; and implementing it within a fully peer-to-peer architecture, in which *all* agents are forming plans and repairing their ontologies. We are currently engaged on this further work within the EU Open Knowledge Project[10].

References

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American* 284(5):34–43.

diSessa, A. 1983. Phenomenology and the evolution of intuition. In Stevens, A., and Gentner, D., eds., *Mental Models*. Erlbaum. 15–33.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence* 12(3):251–272.

Giunchiglia, F., and Walsh, T. 1992. A theory of abstraction. *Artificial Intelligence* 56(2–3):323–390. Also available as DAI Research Paper No 516, Dept. of Artificial Intelligence, Edinburgh.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Lakatos, I. 1976. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

McNeill, F.; Bundy, A.; Walton, C.; and Schorlemmer, M. 2003. Plan execution failure analysis using plan deconstruction. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2003)*.

McNeill, F.; Bundy, A.; and Walton, C. 2004. An automatic translator from KIF to PDDL. In *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2004)*.

McNeill, F. 2005. *Dynamic Ontology Refinement*. Ph.D. Dissertation, School of Informatics, University of Edinburgh.

Pease, A.; Colton, S.; Smaill, A.; and Lee, J. 2004. Modelling lakatos's philosophy of mathematics. In *Proceedings of the Second European Computing and Philosophy Conference, E-CAP2004*.

Saito, Y., and Shapiro, M. 2003. Optimistic replication. Technical Report MSR-TR-2003-60, Microsoft Research.

[10] `http://www.openk.org/`

Shapiro, E. 1983. *Algorithmic Program Debugging*. MIT Press.

Wiser, M., and Carey, S. 1983. When heat and temperature were one. In Stevens, A., and Gentner, D., eds., *Mental Models*. Erlbaum. 267–297.