Plan execution failure analysis using plan deconstruction

Fiona McNeill[*], Alan Bundy[*], Chris Walton[*], Marco Schorlemmer[†]

[*]CISA, School of Informatics, The University of Edinburgh, Appleton Tower, 11 Crichton Street, Edinburgh, EH8 9LE, United Kingdom.
f.j.mcneill@ed.ac.uk,{bundy,cdw}@inf.ed.ac.uk

[†]Escola de Tecnologies d'Informaci i Comunicaci, Universitat Internacional de Catalunya, Immaculada 22, 08017 Barcelona
marco@cir.unica.edu

November 14, 2003

Abstract

We consider the challenges that arise when plans are based on an incorrect representation of the domain in which they are executed. We describe how information about plan formation, and how the way in which each plan step is related to the domain representation, can help identify problems in the underlying ontology when plan failure is encountered. We introduce the notion of a *plan deconstructor*, used to extract this information from the domain representation so that is it available when plan failure occurs. Thus a more accurate ontology can be developed and more robust plans can be formed.

1 Tracking plan formation

In order to generate executable automated plans, it is necessary to have an accurate representation of the domain in which the plans are being executed. In practice this is often difficult. Planning may be taking place in a complex domain where full representation is impractical, or in a dynamic domain where information about the domain may become outdated. Plans formed using this inaccurate representation will often not be executable. Execution failure can give important information about how the domain representation is inaccurate, but only if we can relate the failed plan step back to the parts of the underlying ontology on which that step is based. Hence, in order to extract useful information from plan failure, we need to have a record of how the plan has been formed: what rules were used to justify each plan step, why we believed the preconditions of these rules to be true, and so on. We use *ontology* to refer to the whole of this domain knowledge, both the types of things that exist, the *signature*, and the axioms that constrain the possible interpretations, the *theory.*

Such information is hard to extract from modern planners. The methods employed by most efficient planners do not provide the kind of information necessary for relating each plan step back to rules and facts in the underlying ontology, and in addition the black box nature of most planners makes any access to this information difficult. On the other hand, planners which are good at providing such information, such as situation calculus planners, are much less efficient at planning, especially with plans of more than a few steps, as they face large search problems.

2 The plan deconstructor

We propose to address this problem by using a modern planner combined with a *plan deconstructor*, loosely inspired by the plan validator developed for the planning competition [2, 3]. The purpose of the plan deconstructor is to take the completed plan from the planner and *meta-interpret* it. We are not literally reconstructing how the plan was formed, but rather are building up a picture of how it would have been formed by a first-order planner. The plan deconstructor behaves in a similar way to a situation calculus planner, but avoids the huge search problems of such a planner by having a plan provided for it by a more efficient planner. The plan is stepped through; at each stage the relevant rule from the ontology is found, along with the justification for believing each of the preconditions to be true. This information is annotated to the particular plan step.

The plan deconstructor returns the annotated plan, which is executed as normal without reference to the annotation. However, if failure occurs, this annotation is referred to for information about what part of the ontology may be at fault. The annotation from the plan deconstructor will not identify a precise problem but instead will lead us to an area where this problem must lie. We have at first two options: the rule on which the plan step is based is incorrect, or one of the preconditions is falsely believed to be true. If one of the preconditions is discovered to be the cause of the problem, we then need to find why this precondition was considered to be true, which will involve chaining back through the deconstruction to the point where the value of this precondition was last altered. If this was as a result of an earlier plan step, then we must inspect the rule underlying that plan step and its preconditions before we can identify where the flaw in the ontology lies. If the problem is not caused by one of the preconditions, then the rule must be at fault.

We are considering the situation in which plans are executed by agent interaction; a plan step is executed by a plan implementation agent, who will attempt to communicate with other agents, such as a ticket selling agent, in order to achieve a goal. The additional information we need to narrow down the information given by the annotation to a specific flaw in our ontology will come from our questioning of these agents as to their understanding of the ontology. Thus agents that would initially have been unable to interact appropriately due to ontological mismatches may become able to communicate correctly. For further details of this procedure, see [1].

3 Conclusion

Tracking plan formation using a plan deconstructor can turn failure into an opportunity to learn more about the domain. We use the plan deconstructor to identify which area of the ontology is at fault. The deconstruction guides the agent interaction so that the correct questions are asked in order to pinpoint the exact problem in the ontology. By linking plan failure back to errors in the underlying understanding of the domain, we can refine our ontology so that future plans based on it are more likely to be executable.

References

[1] A. Bundy F. McNeill and M. Schorlemmer. Dynamic ontology refinement. In *Proceedings of ICAPS'03 Workshop on Plan Execution*, Trento, Italy, June 2003.

[2] R. Howey and D. Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proc. of ICAPS Workshop on the IPC*, 2003.

[3] D. Long, S. Cresswell, and R. Howey. Validator for PDDL2.1 plans. Available at `www.cis.strath.ac.uk/~rh/val.html`, 2003.