Updating Action Descriptions and Plans for Cognitive Agents

Peter Stringer  
University of Manchester  
Manchester, United Kingdom  
peter.stringer@manchester.ac.uk

Rafael C. Cardoso  
University of Aberdeen  
Aberdeen, United Kingdom  
rafael.cardoso@abdn.co.uk

Clare Dixon  
University of Manchester  
Manchester, United Kingdom  
louise.dennis@manchester.ac.uk

Michael Fisher  
University of Manchester  
Manchester, United Kingdom  
michael.fisher@manchester.ac.uk

Louise A. Dennis  
University of Manchester  
Manchester, United Kingdom  
louise.dennis@manchester.ac.uk

ABSTRACT

In this paper, we present an extension of Belief-Desire-Intention agents which can adapt their performance in response to changes in their environment. Our main contributions are the underlying theoretical mechanisms for data collection about action performance, the synthesis of new action descriptions from this data, the integration with plan reconfiguration, and a practical implementation to validate the semantics.

KEYWORDS

Beliefs-Desires-Intentions; Action Descriptions; AI Planning

1 INTRODUCTION AND BACKGROUND

Long-term autonomy requires autonomous systems to adapt once their capabilities no longer perform as expected. To achieve this, a system must first be capable of detecting such changes and then adapt its internal reasoning processes to accommodate these. For example, deploying an autonomous robot into a dynamic environment can result in actions becoming unreliable over time, as the environment changes, producing unexpected outcomes that were unforeseeable before runtime. The autonomous agent must be capable of observing these changes and adapting accordingly.

Our work focuses on cognitive agents \cite{3,17,21} programmed in a Belief-Desire-Intention (BDI) \cite{16,17} programming language providing high-level decision-making in an autonomous system, as outlined in \cite{8}. Programs written in these languages use plans created in advance by a programmer to select actions to execute in the environment. These plans make implicit assumptions about the behaviour of the actions they execute. Therefore, in this context, the challenge becomes to make these assumptions explicit, detect when they no longer hold, and then modify the plans accordingly.

2 METHODS

Some Belief-Desire-Intention (BDI) languages use action descriptions (sometimes referred to as capabilities in the literature), which consist of explicit pre- and post-conditions for all known actions. These have their roots in AI planning and STRIPS operators \cite{10}. Mechanisms and semantics used for such functionality are discussed in \cite{7,14,19}. A version of GWENDOLEN (which we have used for our implementation) also exists that contains an implementation of action descriptions \cite{19}.

GWENDOLEN tracks the performance of actions over time in an action log. An action log keeps a record of action outcomes in an array of fixed, application-specific size, where the oldest entry is removed before adding a new one, once the log reaches its size limit. The action log therefore enables GWENDOLEN to reason about the probability of action success and opens the possibility of implementing an action lifecycle \cite{20}, inspired by the concept of goal life-cycles for BDI languages \cite{13}.

The automated planning research community has invested considerable effort in the modelling of actions with stochastic outcomes, both theoretically \cite{15,22}, and practically (e.g. \cite{5,11}). This community deploys action descriptions to flexibly plan on-the-fly for each new goal, which avoids the problem faced in BDI languages that an action whose behaviour has changed may result in failing, and therefore useless, plans. Plan failure has been extensively researched in BDI programming languages (e.g., \cite{2,9,18}), however, it has not been linked with action descriptions perhaps because most languages do not use action descriptions as a mechanism to detect action failure. The closest work to our own is in \cite{13} with a proposal for BDI goal life-cycles.

A key component of our approach is synthesizing or learning a new action description when an action ceases to perform as expected. Using algorithms to discover the effects of actions has been explored in the AI Planning domain \cite{1}. We have based our approach on ideas from \cite{6} and \cite{12} where new action descriptions are learned from traces of action behaviour with a weighting

\[ \text{Pre} \rightarrow \text{Post} + \text{t} \]

\[ \text{Post} \rightarrow \text{Pre} - \text{t} \]

\[ \text{t} \text{ represents time.} \]

\[ \text{Pre} \rightarrow \text{Post} + \text{t} \]

\[ \text{Post} \rightarrow \text{Pre} - \text{t} \]

\[ \text{t} \text{ represents time.} \]
We synthesize a new action description by extracting, from the action log, all the failed instances of the deprecated action. We then have a list of new candidate post-conditions for the action in the form of the change in beliefs as the action executed. Each item in this list is assigned a weight score based on how recent the item is. The weights for identical items are then summed, and the highest-scoring item is selected as the new post-condition for the action. Pseudo-code for this process is shown in Algorithm 1. Line 2 instantiates the initial weight score (n) to 1, and in Line 3 it sets post_scores to an empty map. Lines 4–7 will loop through every entry in the action log to find entries that match with the deprecated action (same action) and where the outcome of the entry was reported as a failure. When this happens, the post-conditions of the action are added to the post_scores map along with the weight score, which is then incremented by one for the future iterations of the action log. In line 8 we initialise best with 0. Lines 9–11 iterate over the keys in the post_scores map to select the candidate post-condition with the highest weight score.

Algorithm 1: Algorithm for synthesizing post-conditions.

```
1 if action is deprecated then
2   n ← 1;
3   post_scores ← {} // map data-structure
4   for entry ∈ action log do
5     if entry[0] = action & entry[2] = Failure then
6       post_scores[entry[1]] ←
7         post_scores[entry[1]] + n;
8     n ← n + 1
9     best ← 0;
10    for post ∈ keys(post_scores) do
11      if post_scores[post] > best then
12        best ← post
```

Once we have a new action description, we use the plan patching mechanism from [4] to patch any plans containing the action.

3 EVALUATION AND CONCLUSION

We evaluated our approach with a navigation example. Our environment consisted of five waypoints and our agent had a plan for a patrol mission to visit each waypoint in turn. Each move action had a description of the form: \( \{at(X)\} move(X, Y) \{\neg at(X), at(Y)\} \). We allowed one move action to change its behaviour so that the agent arrived at a different waypoint to the one anticipated (e.g., because of obstacle avoidance behaviour). The system would then observe this changed behaviour and update the action description. It then attempts to patch its plans – typically by finding a different route to the desired waypoint that avoided whatever was blocking the altered move action.

We have presented here the over-arching template of a framework for adapting BDI agent plans in the face of changed action behaviour. While there is a great deal of scope for extending the framework we believe the basic architecture and concept provides a sound foundation for this further work.

1All code can be found at https://github.com/mcapl/mcapl/tree/reconfig_peter
ACKNOWLEDGMENTS

This work has been supported by The University of Manchester’s Department of Computer Science and the EPSRC “Robotics and AI for Nuclear” (EP/R026084/1) and “Future AI and Robotics for Space” (EP/R026092/1) Hubs.

REFERENCES


