From: AAAI-93 Proceedings. Copyright © 1993, AAAI (www.aaai.org). All rights reserved.

Learning Plan Transformations from Self-Questions: A Memory-Based Approach

R. Oehlmann D. Sleeman P. Edwards
University of Aberdeen, King's College Department of Computing Science
Aberdeen AB9 2UE, Scotland, UK
{oehlmann, sleeman, pedwards}@csd.abdn.ac.uk

Abstract

Recent work in planning has focussed on the re-use of previous plans. In order to re-use a plan in a novel situation the plan has to be transformed into an applicable plan. We describe an approach to plan transformation which utilises reasoning experience as well as planning experience. Some of the additional information is generated by a series of self generated questions and answers, as well as appropriate experiments. Furthermore, we show how transformation strategies can be learned.

Planning, Execution, and Transformation

Over the past few years a new view of planning has emerged. This view is based on the re-use of pre-stored plans rather than on building new plans from first principles.

Debugging of Interaction Problems uses pattern based retrieval of pre-stored programs and subroutines. Programs containing bugs are repaired by *patching* faulty steps, (Sussman 1974).

Adaptive Planning employs abstraction and specialisation of plans as transformation strategies, (Alterman 1988).

Case-Based Planning is an extension of Sussman's ideas of retrieval and repair and the application of these ideas to planning. Hammond's CHEF system modifies a retrieved plan to satisfy those goals which are not already achieved, executes the plan, repairs it when it fails, and stores the planning problems related to the failure in the plan. This approach is characterised as memory-based, because the organisation of the memory of previous plans is changed during the process (Hammond 1989).

These approaches stress the necessity to modify previous plans in order to obtain plans which can be used in a new situation. In particular, the memory-based approach to plan transformation attempts to integrate the phases of building a plan and learning from planning failures.

In addition to the standard features of plan modification, execution, and repair, a planning system should be able to transform plans based on knowledge about actions, with the objective being to improve the system's reasoning about actions as well as plan execution. Furthermore, it should be able to learn the transformation strategy as a higher level plan in order to apply similar methods of modification and repair to the strategy as to other plans. Oehlmann, Sleeman, & Edwards (1992) argue that learning a more appropriate plan can be supported by the generation of appropriate questions and answers which we refer to as self-questions and answers; this is done using case-based planning. If an answer can not be generated, a reasoning component plans and executes an experiment in an attempt to acquire the missing knowledge.

In particular, we address the following planning issues:

- reasoning about plans before plan execution,
- reasoning about all known plans,
- generating self-questions to build a plan transformation,
- learning transformation strategies.

We have implemented our approach to plan transformation in an exploratory discovery system, IULIAN[1] (Oehlmann, Sleeman, & Edwards 1992) and have tested the system in various domains.

In the remainder of this paper, we present a top level view of the IULIAN system followed by an example demonstrating the interaction between case-based question and experimentation planning. We then describe our approach to plan transformation. Finally, we evaluate the approach and indicate various options for future work.

[1] IULIAN is the acronym for Interogative Understanding and Learning In AberdeeN.

Reasoning about Actions from Self-Questions

IULIAN uses the planning of self-questions, answers and experiments to model reasoning about plans and actions. The main stages of IULIAN's top level algorithm are given in Figure 1.

1. **Question Stage.** Input a description of a new experimental setting (problem), execute a pre-stored question plan about the expected experimental result.
2. **Answer Stage.** Try to generate an Answer for the Question.
3. **Experimentation Stage.** If the answer can not be generated, conduct an experiment to obtain the answer.
   a) Generate a hypothesis about the experimental result.
   b) Retrieve and execute an appropriate pre-stored experimental plan (interacting with a simulator).
   c) Compare the experimental result with the hypothesis and determine the expectation failure.
   d) Store experimental setting and experimental result as a new case.
4. **Question and Answer Loop.** If an expectation failure is found, generate a *why* question about the expectation failure. If the *why* question can be answered, store the answer as a new explanation, otherwise generate a sequence of questions, answers, and experiments in order to answer the *why* question and to obtain an explanation. If during the generation of this sequence a case is retrieved which is not sufficiently similar to the input problem, transform the plan associated with the retrieved case into a new plan and generate a new case by executing that plan.
5. **Model Revision Stage.** Use the final explanation to modify the causal model.

Figure 1: IULIAN's Top Level Algorithm

The main task of the system is the discovery of new explanations to revise an initial theory. The basic data structures of the IULIAN system are cases, causal models, and plans (see Figure 2). A case comprises two components, an experimental setting (e.g. a description of an electric circuit with battery, lamp, and switch) and the result of an experiment such as the statement that *the lamp is on when the battery is switched on.*

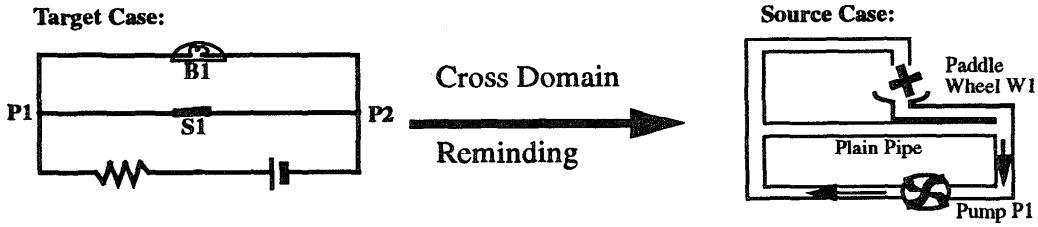
Cases are represented by objects and relations between objects. Causal models have a similar representation. However, they are stored in a separate library and their objects are viewed as abstract concepts, eg., the concept "lamp" rather than an actual lamp used in a given experiment (Oehlmann 1992). In addition, causal models use particular relations between concepts to represent causal links. A causal model is linked to the cases used to generate that model. Experimental plans describe the steps which have to be executed in order to produce an experimental setting and result, i.e. a case. Each case has a link to the experimental plan which generated it. In addition, each plan contains a set of descriptors (indexes), such as goals and planning failures, to enable plan retrieval. Plans are retrieved by matching their index values with the characteristic values of a given situation. The effects of plan execution are simulated by a rule-based simulator. The same basic plan structure is employed for question and answer plans, although the index vocabulary differs. Executing these plans allows the system to connect small parts of questions and answers in order to generate complete questions and answers. Question strategies are higher level plans[2] which organise the execution of single question and answer plans, when it is important to generate questions in a particular sequence.

An Example

Before we describe the details of the transformation process, we will discuss the core of an example focusing on the question and answer loop (stage 4 of the top level algorithm). The entire example and the questions used during the reasoning process are described in (Oehlmann, Sleeman, & Edwards 1992).

We assume that the IULIAN system receives as input the description of a electric circuit with lamp and closed switch in parallel (target domain). Associated with the description of the circuit (target case) is an experimental (target) plan to build the circuit by connecting the various components and to observe the status of the lamp. The lamp is reported as being off, as a result of this experiment. This result is inconsistent with the systems expectation based on a previous case of a serial circuit in which the lamp was on. The situation characterised by this expectation failure (partially) matches the index of a question plan for generating a *why* question which would identify an explanation of the expectation failure (Stage 4 of the top level algorithm). IULIAN is able to retrieve the question plan but not an appropriate answer plan, because the explanation has not been stored. This new situation determines an index for retrieving the question strategy CROSS-DOMAIN-REMINDING which supports analogical case retrieval between domains. Executing the first step of the question strategy initialises the retrieval and execution of a question plan to generate an additional top level question. The answer plan to this question can be executed and the generated answer comprises a (source) case in the domain of water pipes (source domain). The interplay between the execution of question and answer plans has actually lead to a case-retrieval (reminding) across domain boundaries.

[2]Note that all planning components in the IULIAN system are case-based planners, see (Oehlmann, Sleeman, & Edwards 1993).



Result: *Lamp B1 is off*

Target Plan:

```
(plan :name parallel#switch-bulb
 :domain electric-circuits
 :bindings ((object1 battery-2) (object2 bulb-3)
            (object3 switch-2) (object4 triple-connector-A-1)
            (object5 triple-connector-B-1) (object6 wire1-2)
            (object7 wire2-2) (object8 wire3-2)
            (object9 wire4-1) (object10 wire5-1)
            (object11 wire6-1) (object12 battery-on-1)
            (object13 battery-off-1) (object14 switch-on-1)
            (object15 switch-off-1))
 :goals (parallel%obj2-obj3 obj2-state)
 :steps
   connect-battery
     :action (connect3 object1 object6 object11)
   connect-triple-connector1
     :action (connect4 object4 object6 object7 object8)
   connect-switch :action (connect3 object3 object7 object10)
   connect-bulb :action (connect3 object2 object8 object9)
   connect-triple-connector2
     :action (connect4 object5 object9 object10 object11)
   set-switch-to-on-state
     :action (set-state object3 object14 object15)
   switch-battery-on
     :action (set-state object1 object13 object12)
   check-bulb-state :action (check-state object2))
```

EXPERIMENTATION PLAN and CASE
Parallel Circuit with Switch and Bulb

Result: *Paddle Wheel W1 is not turning*

Source Plan:

```
(plan :name parallel#plainpipe-paddlewheel
 :domain water-pipes
 :bindings ((object1 pump-1) (object2 paddle-wheel-1)
            (object3 plain-pipe-1)
            (object4 pipe-triple-connector-A-1)
            (object5 pipe-triple-connector-B-1)
            (object6 pipe1-1) (object7 pipe2-2)
            (object8 pipe3-2) (object9 pipe4-1)
            (object10 pipe5-1 (object11 pipe6-1)
            (object12 pump-on-1) (object13 pump-off-1))
 :goals (parallel%obj2-obj3 obj2-state)
 :steps
   connect-pump
     :action (connect3 object1 object6 object11)
   connect-pipe-triple-connector1
     :action (connect4 object4 object6 object7 object8)
   connect-plainpipe :action (connect3 object3 object7 object10)
   connect-paddlewheel :action (connect3 object2 object8 object9)
   connect-pipe-triple-connector2
     :action (connect4 object5 object9 object10 object11)
   switch-pump-on
     :action (set-state object1 object13 object12)
   check-paddlewheel-state :action (check-state object2))
```

EXPERIMENTATION PLAN and CASE:
Parallel Water Pipes with Plain Pipe and Paddle Wheel

Figure 2

In the source case a plain pipe and a paddle wheel are in parallel. Additionally the observation that the paddle wheel does not turn is stored in the source case, which is associated with both a causal model and the source plan. A summary of the causal model is given below:

In order for the paddle wheel to turn, water must flow over it. There is a plain pipe in parallel with the paddle wheel. The smaller the resistance in a given pipe, the greater is the water flow in this pipe. If one of two parallel pipes has a very low resistance and the other one has a very high resistance, most of the water flows through the pipe with low resistance. Since the paddle wheel offers resistance to the flow and the plain pipe does not, all of the water flow goes through the plain pipe. Since there is no water flow over the paddle wheel, the paddle wheel does not move.

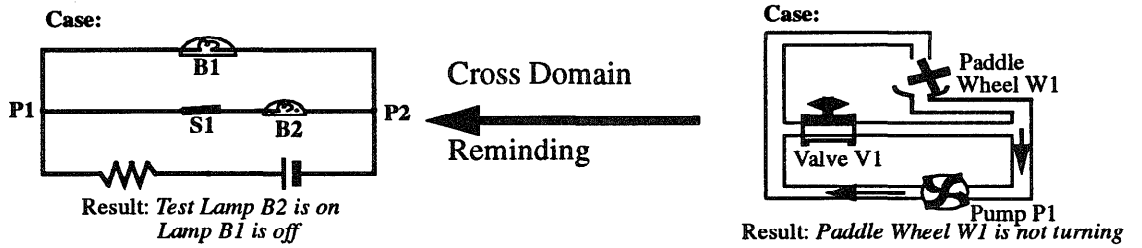
The source model can not be applied to the original electric circuit, because the switch and the plain pipe are not sufficiently similar.[3] Therefore, the source plan is transformed into a new source plan able to generate a new source case which is sufficiently similar to the target case. The transformation process is described in the following section.

During plan transformation, IULIAN replaces the step which refers to the insertion of a plain pipe by a step which refers to the insertion of a valve. The valve is more similar to the switch in the target domain, because both components are used to pursue the goal "select:flow-interruption/flow-support". In addition, the system inserts a step which opens the valve.[4]

Once the transformation process is finished, two additional reasoning stages are needed. First, the reasoner has to collect evidence that the causal model associated with the source case is valid for the transformed source case. Second, the reasoner has to modify the causal model to make it applicable to the target case, and it then has to ensure that the modified causal model is valid for this case.

[3]For a discussion of how similarity is measured see (Oehlmann, Sleeman, & Edwards 1992).

[4]The inserted steps are marked in Figure 3 in italics.



Modified Target Plan:

```
(plan :name parallel#switch-bulb-test
 :domain electric-cicuits
 :bindings ((object1 battery-1) (object2 lamp-1)
            (object3 switch-1) (object4 triple-connector-A-1)
            (object5 triple-connector-B-1) (object6 wire1-1)
            (object7 wire2-1) (object8 wire3-1)
            (object9 wire4-1) (object10 wire5-1)
            (object11 wire6-1) (object12 battery-on-1)
            (object13 battery-off-1) (object14 switch-off-1)
            (object15 switch-on-1)
            (object16 test-lamp-1)
            (object17 wire7-1))
 :goals (parallel%obj2-obj3 obj2-state)
 :steps
   connect-battery
      :action (connect3 object1 object6 object11)
   connect-triple-connector1
      :action (connect4 object4 object6 object7 object8)
   connect-switch :action (connect3 object3 object7 object17)
   connect-test-lamp :action (connect3 object16 object17 object10)
   connect-lamp:action (connect3 object2 object8 object9)
   connect-triple-connector2
      :action (connect4 object5 object9 object10 object11)
   set-switch-to-on-state
      :action (set-state object3 object14 object15)
   switch-battery-on
      :action (set-state object1 object13 object12)
   check-test-lamp-state :action (check-state object16))
```

EXPERIMENTATION PLAN generated by IULIAN:
Parallel Electric Circuit with Switch, Lamp and Test Lamp

Transformed and Modified Source Plan:

```
(plan :name parallel#valve-paddlewheel
 :domain water-pipes
 :bindings ((object1 pump-1) (object2 paddle-wheel-1)
            (object3 valve-1) (object4 pipe-triple-connector-A-1)
            (object5 pipe-triple-connector-B-1) (object6 pipe1-1)
            (object7 pipe2-1) (object8 pipe3-1)
            (object9 pipe4-1) (object10 pipe5-1)
            (object11 pipe6-1) (object12 pump-on-1)
            (object13 pump-off-1) (object14 valve-closed-1)
            (object15 valve-open-1))
 :goals (parallel%obj2-obj3 obj2-state)
 :steps
   connect-pump
      :action (connect3 object1 object6 object11)
   connect-pipe-triple-connector1
      :action (connect4 object4 object6 object7 object8)
   connect-valve :action (connect3 object3 object7 object10)
   connect-paddlewheel :action (connect3 object2 object8 object9)
   connect-pipe-triple-connector2
      :action (connect4 object5 object9 object10 object11)
   set-valve-to-open-state
      :action (set-state object3 object14 object15)
   switch-pump-on
      :action (set-state object1 object13 object12)
   check-paddlewheel-state :action (check-state object2))
```

EXPERIMENTATION PLAN generated by IULIAN:
Parallel Water Pipes with Valve and Paddle Wheel

Figure 3

The first stage can be achieved by modifying the transformed plan (Hammond 1989). This stage has the following effect: in executing the modified plan the water pipe circuit is built by connecting the various components and a small test paddle wheel is placed after the valve. The test paddle wheel allows the planner to test whether the water runs through the valve.

In the second stage, the table of similar objects (see the following section) enables the system to replace all objects from the water pipe domain which appear in the causal model with similar objects from the electric circuit domain. The validity of the modified causal model can be tested by modifying the target plan in the same way as the transformed source plan. A step is added to the plan, allowing the planner to insert a test lamp after the switch in the electric circuit in order to test whether the current flows through the path with the switch. The result of plan execution shows that the current flows through the switch rather than through the main lamp and confirms the new causal model.

Plan Transformation from Self-Questions

In this section, we describe our approach to plan transformation by extending the example in the previous section. We will focus in particular on the interaction between self-questions and transformation steps.

The system has to accept that the plan is appropriate for mapping the relevant parts of the knowledge from the water pipe domain to the domain of electrical circuits. Therefore it retrieves a question plan to generate an evaluation question.[5]

Question 1.2: *Is the retrieved experimental plan (source) appropriate?*

Answer 1.2: *No, there are steps in the target plan without equivalents in the source plan.*

[5] We assume that the appropriate question and answer plans are available rather than a higher level question strategy representing the transformation process. This strategy will be learned during the process of self-questioning and answering.

The answer reveals that the plan is not completely appropriate; however, it is the best plan IULIAN can obtain. Therefore, the strategy CROSS-DOMAIN-REMINDING is suspended and the plan is transformed into a more appropriate one.[6] It is important to note that the strategy goals are not abandoned, the planner still has the goal to perform a mapping between the water pipe domain and the electric circuit domain. IULIAN will achieve this goal only if it has a plan available to build the appropriate links between source and target domains. It is therefore desirable that the system is able to reason about all the plans it knows (Pollack 1992). The stages of plan transformation are summarised in Figure 4.

1. **Match objects** and relations in target and source plan with respect to the abstract descriptor values *goal, task, and belief.* If a match succeeds store the matching object pairs in the **object-transformation-table**. If one of the matches does not succeed perform the following stages.
2. Instantiate steps in target and source plan using the information from the **object-transformation-table** and **match the steps** in target and source plan according to identical *goals* (omitted in Figures 2 and 3) and *actions*. Build the **step-transformation-table** of matching steps.
3. Identify the non matching steps in the target plan and **search the plan library for matching steps** in plans from the source domain. Store these pairs in the **retrieved-step-table**.
4. **Build a new source plan** by inspecting every step in the target plan. If the **step-transformation-table** contains a matching source step, copy this step into the new plan. If the **retrieved-step-table** contains a matching retrieved step, copy this step into the new plan.
5. Collect the abstract indices used to retrieve the appropriate questions and answers during the plan transformation process and **build a new question strategy.**

Figure 4: The Transformation Steps

In order to reason about plans in different domains, a similarity measure has to be established, i.e. the planner has to know which steps in the plan are similar and which steps are different. In a second stage, the planner uses the table of matching objects and the two binding lists to build a table of equivalent steps.

The system is able to generate Answer 1.2, because there are steps which do not appear in the table of equivalent steps. In order to identify these differences, IULIAN retrieves a question plan whose execution results in Question 1.2.1.

Question 1.2.1: *Which steps in the target plan have no equivalent in the source plan and which steps in the source plan have no equivalent in the target plan?*
Answer 1.2.1: *The step CONNECT-SWITCH in the target plan has no equivalent in the source plan and the step CONNECT-PLAINPIPE in the source plan has no equivalent in the target plan.*

After establishing the exact differences between both plans, the system has a choice between modifying the source plan or the target plan. In our example, IULIAN searches in the water pipe domain for the equivalent to the step CONNECT-SWITCH and uses this equivalence to modify the source plan. The system retrieves and executes a question plan focusing on the step CONNECT-SWITCH.

Question 1.2.2: *What is the equivalent for the step CONNECT-SWITCH in the WATER-PIPE domain?*
Answer 1.2.2: *The step CONNECT-VALVE in the plan SERIAL#VALVE-PADDLEWHEEL.*

The generation of the answer involves the search for an equivalent step in the remaining experimentation plans. The system is looking for a step which supports the same goals it pursued in the circuit domain using the step CONNECT-SWITCH. However, the new step has to be found in the domain of water pipes, because the plan to be transformed describes an experiment in this domain. In addition, the new step and the step CONNECT-SWITCH should perform actions with the same name. IULIAN might have identified additional steps in the source plan without equivalents in the target plan. If this happened, the system would then attempt to find appropriate steps in the plan SERIAL#VALVE-PADDLEWHEEL. If this limited search is not successful, it would start a new general search through the entire plan library. However, it would use similar goal, domain and action constraints as described above. After retrieving the step CONNECT-VALVE, the system has the necessary information to build a new plan. The following question focuses on the plan transformation goal and attempts to combine all the single pieces of information obtained.

Question 1.2.3: *How can I make the source plan more similar to the target plan?*
Answer 1.2.3: *I can make the source plan more similar to the target plan by removing the step CONNECT-PLAINPIPE and replacing it with the step CONNECT-VALVE.*

The new experimental plan PARALLEL#VALVE-PADDLEWHEEL is then generated (see Figure 3). We take the memory-based view that organisation of memory should reflect its function (Hammond 1989) and so memory supporting learning should reflect this process by changing its organisation. The learning task described here involves the transformation of plans. Therefore, the overall transformation process should be reflected by changing the indices of plans. IULIAN

[6] If the transformation fails, the problem is presented to the user who can add additional knowledge and re-start the process.

improves plan indexing by storing information about equivalent objects or equivalent steps and information about the problem the system had with the source plan. After successful generation of the new plan, the question and answer goals used to retrieve the appropriate questions and answers are packaged to form a strategy which can be executed when a similar problem arises. In this way, the planner learns a new plan, and additionally a new and more efficient way to cope with the problem of an inappropriate plan in the context of cross domain reminding.

With the newly generated plan, IULIAN continues the execution of the suspended question strategy CROSS-DOMAIN-REMINDING which finally leads to the explanation given in the example.

Discussion

Evaluation

The three main goals of our transformation approach are: improving the execution of question strategies, improving plan execution, and learning new transformation strategies. In this section, we discuss the achievements and limitations of our approach in the light of these goals and compare the approach with previous systems discussed in Section 1.

- Execution of the question strategy is improved, because the transformation process generates new plans. These plans enable the system to execute the strategy. In previous approaches, (Alterman 1988; Hammond 1989; Hanks & Weld 1992), plans are transformed to facilitate plan execution. Similarly, approaches to cross domain analogy rely on existing knowledge structures rather than on knowledge newly generated by experimentation (Vosniadou & Ortony 1989). IULIAN additionally views plans as knowledge about actions which facilitates the reasoning process.
- Plan execution is improved in that the transformed plan addresses additional planning situations. In contrast to previous systems, the close integration of reasoning and planning enables IULIAN to transform plans based on its knowledge about the reasoning process, rather than only using its experience with plan execution.
- A new transformation strategy is learned by storing the components of the transformation process in a question strategy. Learning plan transformations in this way is a novel contribution. In previous approaches, e.g. the CHEF system (Hammond 1989), plan transformation is implemented as a set of fixed, pre-stored rules.
- The current scope of our plan transformation approach is limited to the cross domain reminding strategy. However, we expect that new strategies can be easily incorporated because the structure of our transformation approach is highly modular and each component can be replaced with a new component.

Future Work

Our system evaluation indicates that the current transformation mechanism is restricted to reasoning in the context of cross domain reminding. Although the reasoner is able to apply the approach to a large variety of different situations, we intend to extend our concept of plan transformation to additional reasoning strategies. An important advantage of our approach is the application of case-based planning to the transformation process itself. We will continue to address this question by investigating the modifications needed to adapt a transformation strategy learned in the context of a given reasoning strategy to a second reasoning strategy.

Acknowledgements

We are grateful to Jeff Berger, University of Chicago, for his helpful comments on a previous version of this paper. This research is partially founded by a University of Aberdeen studentship.

References

Alterman, R. 1988. Adaptive Planning. *Cognitive Science* 12:393-421.

Hammond, K. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* New York: Academic Press.

Hanks, S., and Weld, D. 1992. Systematic Adaptation for Case-Based Planning. In Proceedings of the First International Conference on Artificial Intelligence Planning Systems, 96–105.

Oehlmann, R. 1992. Learning Causal Models by Self-Questioning and Experimentation. In Workshop Notes of the AAAI-92 Workshop on Communicating Scientific and Technical Knowledge, 73–80.

Oehlmann, R., Sleeman, D., and Edwards, P. (1992). Self-Questioning and Experimentation in an Exploratory Discovery System. In Proceedings of the ML-92 Workshop on Machine Discovery, 41–50.

Oehlmann, R., Sleeman, D., and Edwards, P. (1993). Case-Based Planning in an Exploratory Discovery System. In Working Notes of the IEE/BCS Symposium on Case-Based Reasoning, 1/1-1/3.

Pollack, M. 1992. The Uses of Plans. *Artificial Intelligence* 57:43-68.

Sussman, G. 1974. The Virtuous Nature of Bugs. In Proceedings of the First Conference of the Society for the Study of AI and the Simulation of Behaviour.

Vosniadou, S. and Ortony, A. (1989). Similarity and Analogical Reasoning. Cambridge, UK: Cambridge University Press.