# Using Inverse Resolution to Learn Relations from Experiments

*David Hume*
*Claude Sammut*[1]

School of Computer Science and Engineering
University of New South Wales
PO Box 1, Kensington NSW
Australia 2033

*ABSTRACT*

We are concerned with learning relations in a reactive environment. A learning agent observes sequences of actions that may change the properties or relationships of objects in the world. The observed sequence is then used to form a theory which can be generalised and tested by imitation. That is, the learner attempts to perform its own sequence of actions. If the test completes as expected then the generalisation is accepted. However, if it fails then a *regeneralisation* occurs. That is, further generalisations are found to explain the difference between expectation and reality.

Inverse Resolution is the primary generalisation mechanism. However, learning by experimentation in a reactive environment causes difficulties that inverse resolution alone cannot handle. We describe the additions required to the theory that enable us to deal with 'surprise' in an experiment, i.e. learning from unexpected results. New inverse resolution operations are used to allow us to generalise from partial matches between our theory and the world, thus enabling us to explain what went wrong with a theory.

[1] Address all correspondence to Claude Sammut (email: claude@spectrum.cs.unsw.oz.au)

## The Problem

We are concerned with the problem of an intelligent agent learning relations by interacting with its environment. We will claim that a number of problems arise in this kind of domain that do not arise in 'batch' learning where all the training data are available at once. Therefore, modifications to existing relational learning theories are required.

We describe a program called CAP that uses a weakly directed learning model to explore its environment. The representation language used is first order horn-clause logic. Being a general purpose representation language, we are able to insert CAP into a variety of different kinds of environments ranging from robot worlds to the worlds of numbers and lists. The induction mechanism used by CAP is based on inverse resolution (Muggleton, 1988; Muggleton and Buntine, 1988) and on an earlier program, MARVIN (Sammut, 1981; Sammut and Banerji, 1986).

If learning in a reactive environment were totally undirected then the intelligent agent could perform an almost infinite variety of experiments to try to deduce the structure of the world from their outcomes. This could be time consuming, so to provide some constraints on exploration, learning is only initiated when another agent, presumed to be knowledgeable, performs some sequence of actions.

One way of characterising the motivation for CAP's learning algorithm is to think of it as trying to produce a theory that will enable it to recognise each sequence of actions it sees. A naive way of accomplishing this is to simply store all observed sequences. Obviously this is wasteful of space and the likelihood of recognising a new action sequence is low since only a sequence identical to one stored could be recognised. So CAP tries to generalise from its observations. Thus, having created an initial theory we proceed to test it and then generalise it.

Testing a theory is relatively straightforward. A theory contains actions and the world states expected to be derived from those actions. Thus, if the program attempts to perform those actions and the resulting state is *not* consistent with the final state described in the theory then the theory has been disproved. When such a failure occurs, CAP attempts to generalise the theory to account for the world as it *is* rather than as it was originally *expected* to be. Thus, it can learn from its mistakes. We will refer to this process as *theory adjustment*. Theory adjustment gives rise to unplanned generalisations, so called because they

are done in order to correct a theory that failed in an unforeseen way. *Planned* generalisations occur after a theory has been tested and the experiment concludes successfully. Although not conclusive, this encourages CAP to accept the theory and generalise it further.

The description of one state of the world is extremely complex in anything other than a toy example. The larger the description, the more ways there are to generalise it. A radical generalisation would attempt to change many terms in the theory. Thus, if a failure occurred, it would be very difficult to isolate the cause of the failure so that it could be rectified by an adjustment to the theory. Therefore, CAP uses a very conservative form of generalisation. The program can be said to escalate by stages to attempt progressively more complex generalisations. CAP terminates when it can perform no generalisations on any of its theories such that the theories remain consistent with the world.

## Constructing Theories

Suppose another agent in the world has constructed an arch. CAP would try to construct a theory that describes the preconditions and post-conditions of each action in the construction sequence. Sequences are represented by expressions in first order logic that indicate the states of objects in discrete time instants and the actions that transform one state into another. We represent the sequence as follows:

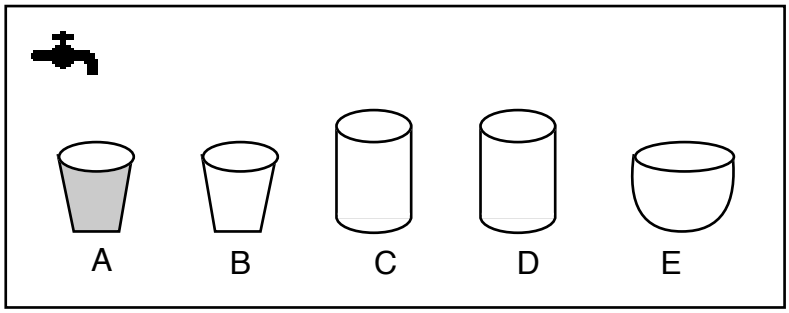
$$S_0 \xrightarrow{A_{0\to 1}} S_1 \ldots \xrightarrow{A_{n-1\to n}} S_n$$

Literals in the representation language are annotated with time stamps. Thus we could state that $P$ is true at time $t$ with the expression: *P at t*. We can also state that an action has certain duration from a start time to a finish time: *A during Start/Finish*.

As an example, suppose a world contains a tap from which water may be obtained and a collection of cups and bowls which can contain water and cylinders which cannot. Two types of actions are possible: pouring water from the tap into any of the objects and pouring from one object into another. A state in this world is illustrated in Figure 1.

The description of the sequence for pouring water from cup $A$ into cup $B$ is:

| | | |
|---|---|---|
| cup(a) at 0 | | cup(a) at 1 |
| cup(b) at 0 | pour(a, b) during 0/1 | cup(b) at 1 |
| contains_liquid(a) at 0 | ⟶ | ~contains_liquid(a) at 1 |
| ~contains_liquid(b) at 0 | | contains_liquid(b) at 1 |




| | |
|---|---|
| cup(A) | contains-liquid(A) |
| cup(B) | ~contains-liquid(B) |
| cylinder(C) | ~contains-liquid(C) |
| cylinder(D) | ~contains-liquid(D) |
| bowl(E) | ~contains-liquid(E) |

Figure 1: Water world

Notice that for simplicity we have omitted descriptions of objects *C*, *D* and *E*, but they too are part of the complete sequence description.

Given a sequence of actions applied to states in the world, what sort of theory can we devise? CAP behaves rather like a frog trying to catch a fly. The frog's attention is attracted by movement. CAP's attention is attracted by change. Let us take arch building as an example. During the construction process, our attention will be focussed at different times on different objects. While constructing the columns, we will move one block at a time onto a column. While moving one block certain actions will be performed repeatedly. For example, to lift a block to a certain height requires us to repeat the lifting action for several time units before changing to a lateral movement to locate the block over the column. These changes in operation lead us to the following heuristics:

- Divide the original sequence into sub-sequences when the objects being affected change. E.g. transfer one block to a column and then find another block.
- Divide the sub-sequences into sub-sub-sequences when actions on the same objects change. E.g. perform a sequence of lifting actions on one block and then switch to lateral movement of the same block.

In our water world example, CAP transforms the observation into the following initial theory:

```
transfer(Source, Destination) during Initial/Final :-
        cup(Source) at Initial,
        cup(Destination) at Initial,
        contains_liquid(Source) at Initial,
        ~contains_liquid(Destination) at Initial,
        pour(Source, Destination) during Initial/Final,
        cup(Source) at Final,
        cup(Destination) at Final,
        ~contains_liquid(Source) at Final,
        contains_liquid(Destination) at Final.
```

where 'transfer' would actually be an arbitrary symbol invented by the program to uniquely identify this concept and all of the object names have been turned into variables. There are other objects in the world that are not relevant to this example but which are still present. Predicates involving the irrelevant objects are trimmed out. A more detailed description of the procedure for transforming observations into theories is outside the scope of this paper but is described elsewhere (Hume, 1991; Hume and Sammut, 1991).

As we mentioned previously, CAP uses a very conservative method of generalising its theories. Since the representation language is horn-clause logic, we use inverse resolution to generalise clauses. The main generalisation operators are *absorption* and *intra-construction* with some modifications necessary for operation in the reactive environment. Absorption generalises expressions using only background knowledge while intra-construction is able to invent new terms in the description language, thus expanding its ability to describe novel situations. Details of these operations are given by Hume (1991) and the original theory behind them is described by Muggleton and Buntine (1988). In this paper we will describe how these operations must be extended in order to be useful in reactive learning.

Each theory is assigned a generalisation level. Beginning at the lowest level, CAP performs experiments to test the theory in its current state and then proceeds to more and more general theories as long as they are consistent with the world. The generalisation levels are described below.

*Level 1* At this level we simply try to repeat the example sequence. In fact this is testing a generalisation since, when the observed sequence became the seed for the initial theory, constants were replaced by variables, irrelevant terms were removed and recursive expressions were introduced to replace sequences where possible.

*Level 2* Here we try to generalise the primitive action predicates (e.g. move,

pickup, etc.). There are two internal levels of generalisation. First we attempt absorption, i.e. trying to use concept we already know about and then we try intra-construction, i.e. inventing new concepts.

*Level 3* At the next level we attempt to generalise sub-sub-sequences, i.e. sequences of the same action on the same objects. Again there are two internal levels of generalisation: absorption and intra-construction.

*Level 4* At the highest level we attempt to generalise sub-sequences, i.e. sequences of different action on the same objects. The same two internal levels of generalisation exists here.

Unplanned generalisations also go through a number of levels similar to those described above.

## Inverse Resolution and Experiments

The method of inverse resolution searches for a theory, in horn-clause logic, from which we could derive the examples in the training data. Several different operations are available for rewriting the example descriptions as more general expressions, thus providing candidate theories. Let us examine how this would work in the case of CAP.

We have already briefly seen how an initial theory is constructed from an observation. In a batch learning system, this theory would be checked against an unseen test set. However, in interactive learning, no such test data exists, so the program must create its own tests. We will demonstrate this procedure with several examples which will also indicate how generalisation is affected by experimentation.

**Example 1**

Suppose we initially wish to merely repeat an experiment as in the first level of generalisation. The theory is $M \cup W$ where $M$ is the set of stored concepts and $W$ is the set of predicates defining the current state of the world. Suppose $M$ is:

```
fill_from_tap(Container) during Start/Finish :–
        cup(Container) at Start,
        ~contains_liquid(Container) at Start,
        tap_on(Container) during Start/Finish,
        cup(Container) at Finish,
        contains_liquid(Container) at Finish.
```

and *W* is:

cup(a) at 5,
~contains_Liquid(a) at 5.

where 5 is an arbitrary time stamp. To test the theory we must perform an experiment, starting at the current time, with the current state of the world. Thus we wish to test fill_from_tap(X) during 5/End. Initially the test can proceed as an ordinary resolution proof as illustrated in Figure 2. Note however, that an action does not exist in the world state, so part of the proof must be the execution of an action what one is encountered, e.g. tap_on. Execution of an action results in new state information that allows the proof to proceed. In this case, the proof was successful, so the theory is given some support and further generalisations can be attempted. However, the most interesting cases arise when the proof cannot be completed. This suggests that the current theory is incorrect since it could not predict the results of the experiment.

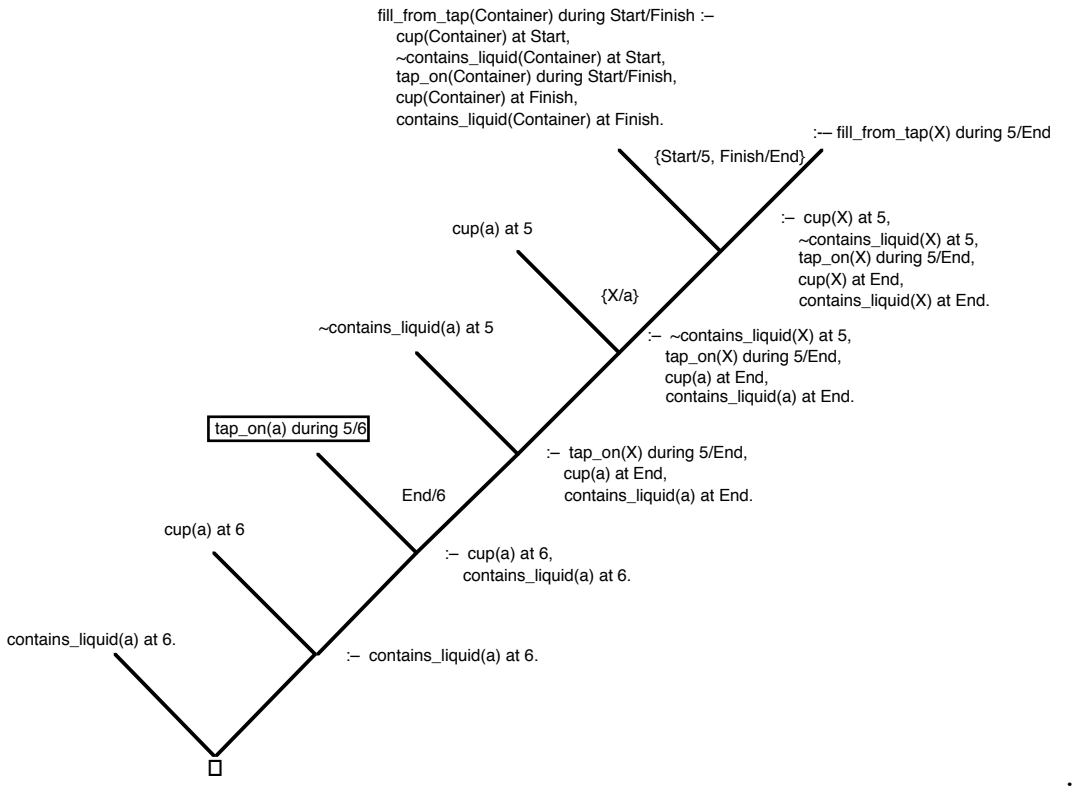



Figure 2: Proof tree for experiment

**Example 2**

Let us look at a further example to see how a failure in prediction can lead to the creation of new concepts. This time we wish to pour water from cup *A* in to cup *B* but *B* already contains water. Recall that the concept for transferring water from source to destination is:

```
transfer(Source, Destination) during Initial/Final :-
        cup(Source) at Initial,
        cup(Destination) at Initial,
        contains_liquid(Source) at Initial,
        ~contains_liquid(Destination) at Initial,
        pour(Source, Destination) during Initial/Final,
        cup(Source) at Final,
        cup(Destination) at Final,
        ~contains_liquid(Source) at Final,
        contains_liquid(Destination) at Final
```

If the state of the world contains the following:

```
cup(a) at 5
cup(b) at 5
contains_liquid(a) at 5
contains_liquid(b) at 5
```

An attempt a testing 'transfer' will fail because the destination does not satisfy the condition that it should be empty. We could abandon the test or else try a generalisation to force it to proceed. Let us compare the world as we would like it to be with what it is:

```
cup(a) at 5                    cup(a) at 5
cup(b) at 5                    cup(b) at 5
contains_liquid(a) at 5        contains_liquid(a) at 5
~contains_liquid(b) at 5       contains_liquid(b) at 5
```

Taking the difference, we construct a new concept:

```
may_contain_liquid(X) :– contains_liquid(X) at T
may_contain_liquid(X) :– ~contains_liquid(X) at T
```

We may now proceed with the experiment. Since CAP's goal is to ensure that the final state of the plan is achieved, pouring into an already full container is perfectly acceptable since it will contain water at the end. This is an example of the use of inverse resolution's W-operator for constructive induction. Given the two clauses at the base of the W, we attempt to construct clauses that could derive the initial clauses. This is illustrated in Figure 3.

Figure 4 puts this W-construction in the context of the attempt to check the initial transfer concept. Recall that to test a concept, we begin a proof. At some point we may discover that the proof cannot proceed and it is then necessary to see if we can generalise the concept. This leads to a digression in the proof.

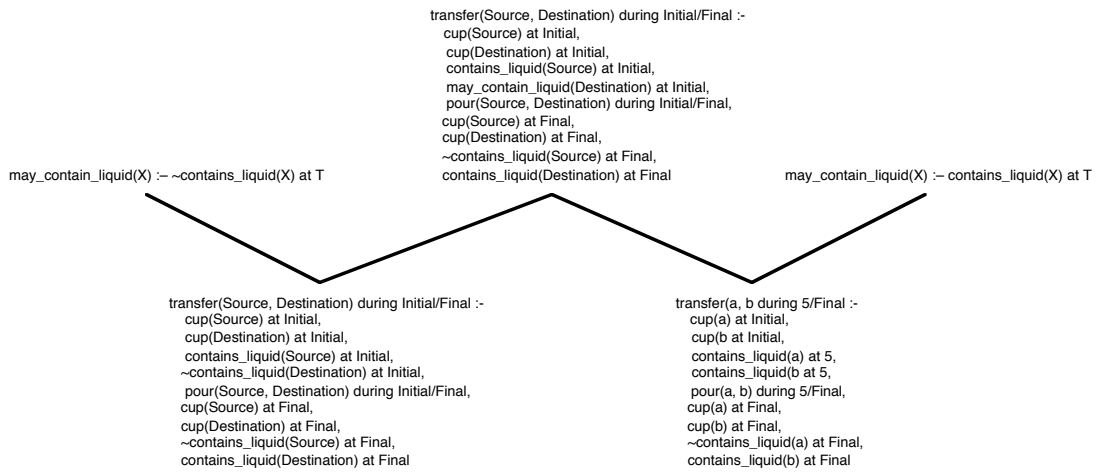



Figure 3: W-operator constructs new concepts

Figure 4 shows an attempt to use the concept, *C*, to complete a proof. In the previous example, this was 'transfer'. As in the example, we use the W operator to construct a generalisation that will allow us to produce a clause that can be used in the proof. This results in *C'* (e.g. a clause allowing the destination container to be full). A side effect of this process is to produce a more general clause, *G* and two auxiliary clauses, *A* and *B*.

Perhaps the strongest constraint that learning in a reactive environment imposes is that the generalisations and experiments possible are dictated by the environment. In this sense, CAP is opportunistic in its learning. It cannot necessarily plan an 'optimal' search for a theory. Instead, inverse resolution operations are triggered by the current state of the world, so CAP must make use of whatever information comes to hand. This is well illustrated in our final example.

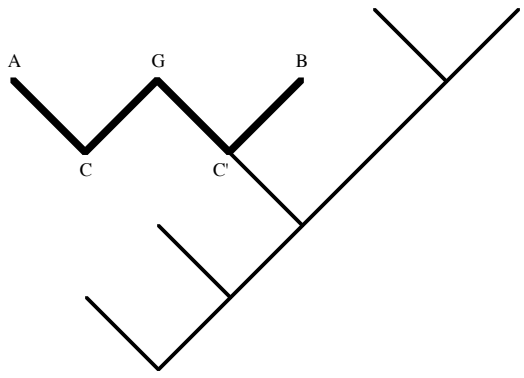



Figure 4: Unplanned generalisation during experimentation

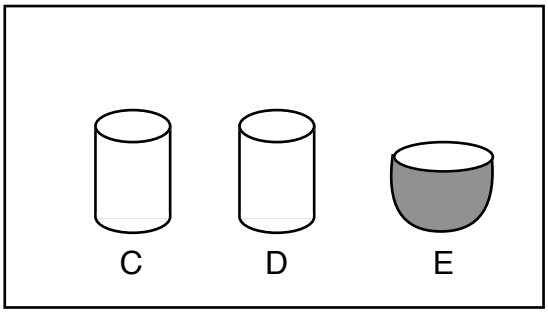



Figure 5: This state causes simultaneous generalisations.

**Example 3**

Figure 5 represents that state of the world after some experimentation has already taken place and the cups have been removed. For any further experimentation to take place, CAP must make two new assumptions: that the source of the transfer need not be a cup and the destination also need not be a cup. Thus two simultaneous generalisations are forced upon the system. While it is not good scientific method to test two hypotheses at once, the hope is that as a result of the experiment, the world will have changed in such a way as to enable further, conservative generalisation to continue.

There are many different ways in which we can use partial matches between the world and our hypotheses to drive generalisations. In this paper we do not have sufficient space to describe all the permutations, but the reader is referred to Hume (1991) for a complete account. Hume also describes a variety of experiments with CAP in different domains.

## Conclusion

The intent of this paper was to demonstrate that incremental learning by interaction with an external world poses problems not encountered in batch learning of relations. Because the world dictates what generalisations can be performed and tested, learning is necessarily opportunistic. Previous relational learning systems (Muggleton and Buntine, 1988) have used a complexity heuristic for guiding the selection of induction operations. For the problems we have described, such heuristics are not available. Instead, we must focus on how to use the current state of the world to maximise information gain during an experiment.

Another problem, not addressed here, is how the program should deal with large quantities of irrelevant information. Spatial domains often have many objects that are 'by-standers' to the main events. Despite their non-par-

ticipation in actions, relationships between them and the objects directly effected by actions do change. How should irrelevant relations be filtered out?

Despite these difficulties we have found that the horn-clause representation has been extremely useful for a number of reasons. Because concepts can be treated as logic program they can be executed as procedures, this making them easy to test. This representation also makes it possible to use inverse resolution as the generalisation method. This provides us with the most general means of discovering relations that we know of and can be extended relatively easily to deal with temporal relations and to the partial matching necessary for comparing a hypothesis with 'reality'.

## References


Hume, D. (1991). *Induction of Procedures in Simulated Worlds*. Ph.D. Thesis, University of New South Wales.

Hume, D. and Sammut, C. (1991). "Applying Inductive Logic Programming in Reactive Environments" in *Proceedings of the Inductive Logic Programming Workshop*, Porto, Portugal.

Muggleton, S. (1988). "A Strategy for Constructing New Predicates in First Order Logic" in *Proceedings of the Third European Working Session on Learning*, D. Sleeman (ed). Glasgow: Pitman.

Muggleton, S. and Buntine, W. (1988). "Machine Invention of First-order Predicates by Inverting Resolution" in *Proceedings of the Fifth International Machine Learning Workshop*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds). Ann Arbor: Morgan-Kaufmann.

Sammut, C. (1981). *Learning Concepts by Performing Experiments*, Ph.D. Thesis, University of New South Wales.

Sammut, C. and Banerji, R.B. (1986). "Learning Concepts by Asking Questions" in *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds). Morgan-Kaufmann.