Solution-Relevant Abstractions Constrain Retrieval and Adaptation

Erica Melis*

Universität des Saarlandes, FB Informatik
D-66041 Saarbrücken, Germany. melis@cs.uni-sb.de

Abstract. Two major problems in case-based reasoning are the efficient and justified retrieval of source cases and the adaptation of retrieved solutions to the conditions of the target. For analogical theorem proving by induction, we describe how a solution-relevant abstraction can restrict the retrieval of source cases and the mapping from the source problem to the target problem and how it can determine reformulations that further adapt the source solution.

1 Introduction

Case-based reasoning retrieves one or more source problems and their solutions in order to transfer and adapt the source solution to the needs of a given target problem. The efficient retrieval of similar source problems from a large case-base has been a major issue in the case-based reasoning research which has employed efficient inductive techniques for learning indices, with similarity measures, with the retrieval from a case-base of abstracted problems, and with analytically extracting relevant features of problems for a specific solution to solve the retrieval problem. Problem indices, learned by inductive techniques, reliably characterize an appropriate source solution to the extend of the correctness of the learning result only. In particular, for complex solutions, indexing by object-level problem features may not suffice to produce correct results by analogical replay.

However, most techniques lack strong justifications for the retrieval, i.e., they do not guarantee the retrieved source problem to provide a solution useful for the target problem. This is due to the fact, that in order to retrieve a source problem that has a solution which is potentially useful for the solution of the target problem, a criterion for the **a priori** similarity of problems has to predict an **a posteriori** similarity of the solutions, as discussed e.g. in [15]. This problem was also addressed by Hesse who stated the problem to justify analogical reasoning [6] and Russell [4] who introduced formulae, called determinations, into domain theories in order to represent how certain predicates of the problem determine certain predicates of the solution. Russell's determinations, however, require to know logical implication at the object-level that logically express the dependency of a feature X on a feature Y in all objects which is a very strong requirement.

* The author was supported by the SFB 3782

A second problem in analogical reasoning, addressed for instance by Holyoak [7] and Gentner [5], is to choose from a variety of possible analogical mappings. As we shall show, solution-relevant abstractions can be exploited for this task as well.

A third non-trivial job in the analogy procedure is to determine the reformulations (adaptations) of problems and solutions. To some extent the proof-relevant abstractions can also be employed to trigger reformulations.

In this paper, we shall present semantic, solution-relevant abstractions in inductive theorem proving that can be found automatically and the preservation of which guarantees a successful transfer of large parts of the proofs. We shall not go into details of the implemented analogy, but the paper focuses on the use of the abstractions for restricting the retrieval and the problem mapping. As we shall see, these abstractions can also serve to determine certain reformulations of the source solution that go beyond an adaptation by symbol mapping.

2 Proof Planning for Proofs by Induction

A proof plan is an abstract representation of a proof that consists of trees of method nodes which are connected by sequents, called goals. The proof planner *CLAM* [3] has been applied successfully to inductive theorem proving because it knows patterns of inductive proofs consisting of base-cases and step-case and it employs a strong search heuristic, called rippling.

The major aim of the step-case proof is to rewrite the induction conclusion until the induction hypothesis is applicable, i.e., to reduce the differences between the induction conclusion and the induction hypothesis. To that end, rippling involves annotating the induction conclusion with *wave fronts* and *wave holes*: *Wave fronts* mark the differences between induction hypothesis and conclusion. *Waves* annotate the smallest terms containing wave-fronts. *Wave holes* represent the parts of waves that also appear in the induction hypothesis. For example, in planning the theorem `lenapp`:

$$\forall a, b.\ length(app(a, b)) = length(app(b, a)) \tag{1}$$

the induction hypothesis is

$$length(app(a, b)) = length(app(b, a)) \tag{2}$$

and the annotated induction conclusion is

$$length(app(\boxed{cons(h, \underline{a})}, b)) = length(app(b, \boxed{cons(h, \underline{a})})) \tag{3}$$

where $cons, app$ are the usual functions on lists. The boxes denote the waves. Wave holes are underlined and wave fronts are the non-underlined parts within the boxes. The *skeleton* of an annotated term is constructed from wave holes and the parts of the term that do not belong to a wave.

Rippling also annotates[2] rewrite rules such that the skeleton of the LHS equals the skeleton of the RHS. The resulting skeleton preserving rules are called *wave-rules*. E.g., a wave-rule for the function *app*, computed from defining rules of *app*, is

$$app(\boxed{(cons(X, \underline{Y}))}, Z) \Rightarrow \boxed{cons(X, (\underline{app(Y, Z)}))}. \tag{4}$$

The skeleton on each side of the implication is $app(Y, Z)$. Rippling works by systematically applying wave-rules to the induction conclusion and to subgoals in the plan. By applying wave-rules to successive goals in the planning process, wave fronts can be moved or removed. In this way, rippling works towards its goal of moving or removing those parts of the induction conclusion which are marked by the wave fronts, so that the induction hypothesis, represented by the skeleton, can be used. You can imagine that the wave fronts move through the term tree of the theorem when a wave-rule is applied. For instance, if (4) and another rule are applied to the goal (3) the following subgoal is obtained:

$$\boxed{s(\underline{length(app(a, b))})} = length(app(b, \boxed{cons(h, \underline{a})})) \tag{5}$$

For a visualization of rippling see Figure 1. Note how the wave front on the LHS has been moved outwards from the term tree of (3) to the term tree of (5).

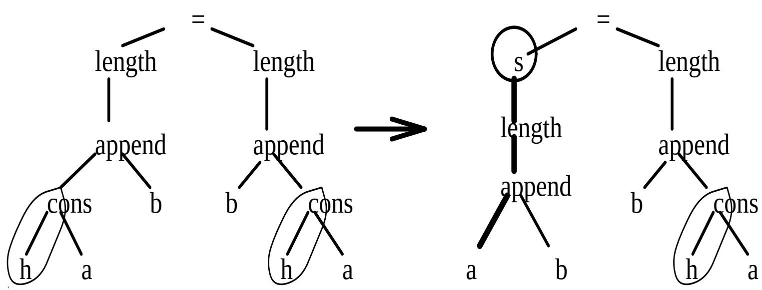

Fig. 1. Term tree of (3) changes to term tree of (5), rippling path in bold

3 Abstractions that Guide the Transfer of Proofs

Remember that one of the main problems for analogical reasoning is to recognize features of *problems* which ensure the similarity of their respective solutions in order to be able to retrieve useful sources and which allow for the choice of appropriate mappings from a possibly large set of (second-order[3])-mappings between source and target problems. We present a solution for this problem in analogy-driven proof plan construction for inductive theorem proving that employs proof-relevant abstractions.

[2] The annotations being the wave fronts and wave holes

[3] second-order mapping means that function symbols are mapped to function terms.

3.1 Analogy-driven Proof Plan Construction

The analogy-driven proof plan construction introduced in [11] is a control strategy of a proof planner that suggests methods and instantiations rather than searching for them. This analogy comprises retrieval, mapping, reformulation, and replay. The reformulations that yield changes of the theorem and proof assumptions that are accompanied by certain changes of the proof plan. The analogy-driven proof plan construction has been implemented on top of the proof planner *CLAM*. The resulting analogical transfer for the example step-case of the `lenapp` proof to a proof of `halfplus` is shown in Figure 2, where *length* is abbreviated by *len*, *app* by infix <>, *cons* by ::.

The retrieval based on proof-relevant abstractions is explained in section 4.

SOURCE PLAN

$$len(\boxed{::(h,\underline{a})} <> b) = len(b <> \boxed{::(h,\underline{a})})$$

$$\downarrow \ \text{WAVE}(\texttt{app2})$$

$$\downarrow$$

$$len(\boxed{::(h,\underline{(a <> b)})}) = len(b <> \boxed{::(h,\underline{a})})$$

$$\downarrow \ \text{WAVE}(\texttt{len2})$$

$$\boxed{s(\underline{len(a <> b)})} = len(b <> \boxed{::(h,\underline{a})})$$

$$\downarrow \ \text{WAVE}(\texttt{lenapp2})$$

$$\boxed{s(\underline{len(a <> b)})} = \boxed{s(\underline{len(b <> a)})}$$

$$\downarrow \ \text{FERTILIZE}$$

$$s(len(b <> a) = s(len(b <> a)$$

$$\downarrow \ \text{ELEMENTARY}$$

true

TARGET PLAN

$$half(\boxed{s(s(\underline{a}))} + b) = half(b + \boxed{s(s(\underline{a}))})$$

$$\downarrow \ \text{WAVE}(\texttt{plus2})$$

$$half(\boxed{s(\underline{\underline{\boxed{s(\underline{a})} + b}})}) = half(b + \boxed{s(s(\underline{a}))})$$

$$\downarrow \ \text{WAVE}(\texttt{plus2})$$

$$half(\boxed{s(\underline{\underline{\boxed{s(\underline{a+b})}}})}) = half(b + \boxed{s(s(\underline{a}))})$$

$$\downarrow \ \text{WAVE}(\texttt{half3})$$

$$\boxed{s(\underline{half(a+b)})} = half(b + \boxed{s(s(\underline{a}))})$$

$$\downarrow \ \text{WAVE}(\texttt{lemma})$$

$$\boxed{s(\underline{half(a+b)})} = \boxed{s(\underline{half(b+a)})}$$

$$\downarrow \ \text{FERTILIZE}$$

$$s(half(b+a)) = s(half(b+a))$$

$$\downarrow \ \text{ELEMENTARY}$$

true

Fig. 2. Step-case replay

Before the actual analogical replay, second-order mappings from the source theorem and wave-rules to the target theorem and wave-rules are found guided by proof-relevant abstractions as explained in section 5. An analysis of these mappings and of their result w.r.t. changing proof-relevant abstractions decides which reformulations to be applied. This is described in section 6.

Note that the abstraction is used at a *meta-level*. This means that rather than taking the proof/plan of an abstracted problem as the source solution to be replayed and adapted, we use abstract features of the source and target problems to restrict the retrieval, mapping, and to trigger reformulations.

3.2 The Labelled Fragment Abstraction

Labelled fragments(LFs), introduced in [8], are an abstraction of wave-rules obtained by removing the structure of the wave-fronts and those parts of the skeleton not affected by wave-fronts. For each function/relation symbol occurring in the source and target theorem we automatically compute labelled fragments corresponding to wave-rules that belong to the source and target problem, respectively. Take, for instance, the wave-rule `app2`

$$app(\boxed{cons(X,\underline{Y})}, Z) \Rightarrow \boxed{cons(X,(\underline{app(Y,Z)}))}.$$

Note that in the lhs of `app2` the wave-front is situated at the left(first) argument of *app* and how it moves to the top of *app* in the rhs of `app2`. This situation is reflected in the most right labelled fragment of *app* as shown in Figure 3. The wave-rule `len2` $length(\boxed{cons(X,\underline{Y})}) \Rightarrow \boxed{s(\underline{length(Y)})}$
is abstractly encoded into the labelled fragment of *length*.

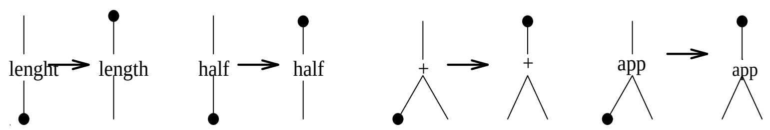

Fig. 3. Labelled fragments

As we shall see, the labelled fragment abstractions are semantically motivated by the course of inductive theorem proving that employs rippling, rather than purely syntactically motivated by symbol matching. In the following, we substantiate why labelled fragments are proof-relevant abstractions and explain how they are used by our analogy procedure.

4 How the Retrieval Employs Labelled Fragments

The term tree of a theorem is called the *theorem tree*. The *rippling paths*, as shown in Figure 4, are the paths in the theorem tree on which the wave-fronts are moved successfully by rippling.

The LFs of functions/relations in a theorem tree determine the rippling paths through the theorem tree. This can be seen in Figure 4, where the bold rippling paths start at the induction variables and are combined from LFs of Figure 3. Note how the rippling paths of `lenapp` and `halfplus` agree because the LFs of *app* and + and of *length* and *half* agree.

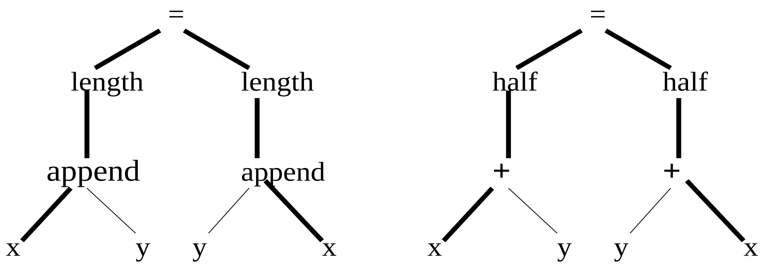

Fig. 4. Term trees of `lenapp` and `halfplus` with rippling paths in bold

The rippling paths in a theorem tree abstractly encode the step-case proof, in particular, the consecutive application of the WAVE method in proof plans. You can recognize how the WAVE methods in Figure 2 apply wave-rules, such as app2, that move the wave-fronts as abstractly shown in Figure 3. E.g., in Figure 2, first WAVE(app2) is applied and then WAVE(len2) which corresponds to the (abstract) rippling path of `lenapp` in Figure 4 which, in turn, is determined by the labelled fragments in Figure 3. Therefore, labelled fragments provide a proof-relevant abstraction of problems to be proved by induction. Since the step-case subplan of the source plan can likely be transferred successfully to the target if the rippling paths in the target tree are identical or very similar to the successful rippling paths in the source tree, only sources should be retrieved that have rippling paths identical or very similar to those in the target. If the corresponding functions in source and target theorem have identical labelled fragments, then the source and target proof plan will, consequently, largely resemble each other (at least for the step-case subplan). For a given target problem, the retrieval of a source should, thus, be restricted to those problems whose theorems have the same abstractions in terms of LFs.[4]

With the help of labelled fragments, the case base can be pre-structured into classes of cases with with the same LF-abstracted theorems. The elements of a class can be further distinguished by their rewrite rules (definitions, axioms). This makes the retrieval a **two-stage** process with a first cheap step that retrieves the source class and a second step more expensive that tries to second-order match the rewrites of each element of that class.

A class of source cases contains all stored cases with identical rippling paths. A class is represented by a theorem tree whose nodes on rippling paths are annotated with the LFs common to all its cases. The nodes occurring outside of rippling paths in any of the cases are abstracted to meta-variables in the class representation because they are irrelevant for the abstraction. The first stage tries to match the target theorem with one of the class representatives

[4] or differ in a controlled way - see below.

and checks the LFs in this process as well. It chooses the class with the fewest predefined deviations (that can be handled by reformulations later). Classes that have equal representatives except for these deviations can be linked in the memory in order to reduce the in the first stage. Only one of a set of linked representatives has to be matched with the target theorem tree.

The second stage chooses the best case from the retrieved class by trying to second-order match the target-rules (lemmata) with the rules of the cases in the retrieved class. Second-order match is decidable but can be expensive, in particular with many rules involved, and it has to be supported by heuristics to choose reasonable matches. The heuristics are not explained here, but see [16].

5 How Labelled Fragments Restrict the Mapping

In parallel to the retrieval the analogical mappings are computed. First, a basic second-order mapping m_b from source theorem to target theorem and then an extended mapping m_e from the source rewrites to the target rewrites (wave-rules). m_b maps the source theorem tree to the target theorem tree and thereby it maps the rippling-paths of the source theorem tree to rippling-paths of the target theorem tree. Similar to the previous section, we require the mapping m_b from source theorem to target theorem to **preserve the labelled fragments** or to change LFs in a controlled way such that the source rippling paths are mapped to similar successful target rippling paths.

According to [8], labelled fragments are rather insensitive to missing lemmas. This means that if a function f has a labelled fragment, constructed from f's definition or other wave-rules, that contributes to an (abstract) rippling path in the theorem tree, and if a particular lemma is missing for the *concrete* rippling at a node N, then very likely a lemma can be proven that can be used for rippling at N of the theorem tree. This situation is highly desirable for analogies because analogy should work despite missing wave-rules. We benefit from this situation in that – as a (re)action to a failed justification – our analogy procedure suggests target wave-rule based on m_b and some other information if no appropriate image of a source rewrite is given in the target problem. Space precludes us from decribing how m_e is computed. This mapping does not depend on LFs.

6 How Labelled Fragments Trigger Reformulations

Now we define reformulations that are devised for certain deviations from LF-preservation. Each reformulation is triggered by frequently occurring peculiarities in the mapping from source to target theorem or rules. The examples are presented for illustrating the situations in which reformulations are triggered.

shrink If m_b maps $f_i \mapsto \lambda w.w$, then the WAVE method for f_i becomes redundant and can be removed from the proof plan.
Example:
source: $sum(x) + sum(y) = sum(app(x, y))$

target: $+(x, s(y)) = s(+(x, y))$
For the first occurrence of sum[5] maps to the identity function. If the induction in the source is on x, a rule is needed to rewrite $sum(cons(v0, x))$ but this is redundant in the target.

blow If m_b maps a source function symbol f to a function $\lambda\overline{x}.g_1(g_2(\overline{x})$ that has the same (combined) labelled fragment as f, then m_e suggests a corresponding target rewrite with $\mathrm{m}_b(f)$ in its skeleton.
An alternative not considered yet, is to introduce two WAVE methods (for g_1 and g_2) instead of one source WAVE method for f.
Example:
source: $+(x, s(y)) = s(+(x, y))$
target: $double(+(x, s(y))) = s(double(+(x, y)))$

condt If the source theorem is Th_s and the target theorem is $C_t \to Th_t$, and if the basic mapping m_b preserves labelled fragments and $m_b(Th_s) = Th_t$, then a *condt* reformulation of the proof plan replaces (weak) FERTILIZE by VERY (WEAK) FERTILIZE.[6] *condt* introduces into the target planning the instantiated antecedents C_{tj} of the target base-cases j as additional subgoals. *condt* transfers the source base-case if C_{tj} can be proven, and it terminates a target base-case if C_{tj} is disproved. An example is the reformulation used for
source: $half(double(x)) = n$
target: $even(x) \to double(half(x)) = x$.

case [7] Suppose a casesplit produces n subgoals in the source plan. Each of these subgoals will be associated with a conditional wave-rule Suppose the subgoals of the casesplit in the source are $cs_1, ..., cs_n$ and in the target the subgoals are $ct_1, ..., ct_n$. Then cs_i is mapped to ct_j if the labelled fragment of the wave-rule associated with cs_i is identical to that associated with ct_j. In this way, casesplit branches are permuted in a way such that an analogy is made between branches with similar rippling paths.

7 Conclusion and Related Work

We described a particular type of proof-relevant abstraction for proofs by induction. The abstraction of problems is based on labelled fragments. This abstraction provides a basis for theorem proving by analogy as implemented in the system ABALONE that works on top of the proof planner *CLAM*. The preservation of the abstraction between source and target problems guarantees the step-cases of the corresponding source and target proof plans to be identical. Certain slight deviations from the strict preservation lead to forseeable changes of the proof plan that can be realized by reformulations.

[5] sum is a function on lists that returns the sum of the list members.
[6] VERY (WEAK) FERTILIZE produces the additional lemma $C_t' \to C_t$ to be proved, where C_t' is the antecedents of the induction conclusion and C_t is the antecedents of the induction hypothesis.
[7] This reformulation is due to Jon Whittle.

Evaluation The main goal of using analogy on top of CL^AM was to automatically provide proof plans for theorems that could not be proved automatically without analogy. That is, the primary goal was not to reduce the problem solving time because CL^AM plans very efficiently. Space precludes us from giving a list of theorems that have been proved with analogy and could not be proved without analogy, see [10, 16]. Even without automated retrieval of a source case the analogy would be useful because more theorems can be proved.

The results rely on the use of the LF-abstraction that guarantees success of the target step-case. Without these abstractions, full second-order problem matches had to be considered, as done in [9]. This is quite restricting. Our results do not depend of the size of the proof because only the size of the theorem tree and number of rules effect the retrieval, mapping, and reformulation. (The replay takes longer with complex plans to replay, of course.) That is, complicated theorems can be replayed.

For mathematical theorems, the retrieval (and mapping) is a very difficult task. Thus, any contribution is valuable. Since our technique applies to the whole class of theorems proved by induction, it provides progress. The presented abstraction is specific for proofs by induction. The general idea to look for proof-relevant abstractions is not specific, though.

Related Work Most attempts that make use of abstraction in the context of analogy, retrieve, transfer, and adapt solutions of an abstracted source problem, for instance the approaches in [9, 1, 14]. Instead, our approach retrieves, transfers, and adapts the object-level source proof plan and utilizes abstract features of the problem as a filter for the retrieval and the mapping. Börner [2] constructs prototypes by abduction. She abstracts a class of problems considered similar to each other to a prototype of the class by anti-unification. This abstract prototype is refined for the target. This suggests that a target problem that unifes with the prototype will have the same abstract solution but this largely depends on the sample set used for computing the prototypes.

Somewhat similar to our use of abstraction in analogical theorem proving is Veloso's footprint mechanism [13] that identifies solution-relevant features at the *object-level* and that works as a filter in the retrieval of source cases. The object-level features computed by the footprint contrast, however, with our abstract labelled fragment features and cannot be used to contrain the mapping or even trigger reformulations.

The idea of adaptation-guided retrieval in [12] is similar to our possibility of predefined deviations from the LF-preservation in the retrieval that triggers reformulatons. The mapping problem has been tackled syntactically, for instance, the Analogical Constraint Mapping Engine (ACME) [7]. Gentner's SME [5] is based on the systematicity principle that emphasizes syntactic structural constraints. Holyoak and Thagard's ACME [7] focuses on syntactic similarity constraints and manually predefined pragmatic constraint. Also purely syntactically, Kolbe and Walther [9] perform a second-order matching from a generalized source.

References

1. R. Bergmann and W. Wilke. On the role of abstraction in case-based reasoning. In B. Faltings and I. Smith, editors, *Fourth European Workshop on Case-Based Reasoning (EWCBR-96)*, Lausanne, 1996.
2. K. Börner. Structural similarity as guidance in case-based design. In *First European Workshop on Case-Based Reasoning 1993*, volume LNAI, Berlin, 1994. Springer Verlag.
3. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.
4. T.R. Davis and S.J. Russell. A logical approach to reasoning by analogy. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 264–270, Milan Italy, 1987. Morgan Kaufmann.
5. D. Gentner. Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.
6. M. Hesse. *Models and Analogies in Science*. Indiana: University of Notre Dame Press, Notre Dame, 1966.
7. K.J. Holyoak and P. Thagard. Analogical mapping by constraint satisfaction. *Cognitive Science*, 13:295–355, 1989.
8. D. Hutter. Synthesis of induction orderings for existence proofs. In A. Bundy, editor, *Proceedings of 12th International Conference on Automated Deduction (CADE-12)*, Lecture Notes in Artificial Intelligence 814, pages 29–41. Springer, 1994.
9. Th. Kolbe and Ch. Walther. Reusing proofs. In *Proceedings of 11th European Conference on Artificial Intelligence (ECAI-94)*, Amsterdam, 1994. http://kirmes.inferenzsysteme.informatik.th-darmstadt.de/ kolbe/.
10. E. Melis. Analogy in CLAM. Technical Report DAI Research Paper No 766, University of Edinburgh, AI Dept, Dept. of Artificial Intelligence, Edinburgh, 1995. available from http://jswww.cs.uni-sb.de/~melis/.
11. E. Melis. A model of analogy-driven proof-plan construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 182–189, Montreal, 1995.
12. B. Smyth and M.T. Keane. Retrieving adaptable cases. In *Topics in Case-Based Reasoning: The Role of Adaptation Knowledge in in Case Retrieval*, pages 209–220, Berlin, 1994. Springer.
13. M.M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer, Berlin, New York, 1994.
14. A. Villafiorita and F. Giunchiglia. Inductive theorem proving via abstraction. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics, AI/MATH96*, pages 150–153, Fort Lauderdale, Florida, 1996.
15. S. Wess. *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik*. PhD thesis, FB Informatik, Univ.Kaiserslautern, January 1995.
16. J. Whittle. Analogy in $CL^{A}M$. Technical Report MSc.thesis, University of Edinburgh, Dept. of AI, Edinburgh, 1995.