An Efficient First-Order Horn-Clause Abduction System Based on the ATMS*

Hwee Tou Ng
Raymond J. Mooney
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
htng@cs.utexas.edu, mooney@cs.utexas.edu

Abstract

This paper presents an algorithm for first-order Horn-clause abduction that uses an ATMS to avoid redundant computation. This algorithm is either more efficient or more general than any other previous abduction algorithm. Since computing all minimal abductive explanations is intractable, we also present a heuristic version of the algorithm that uses beam search to compute a subset of the simplest explanations. We present empirical results on a broad range of abduction problems from text understanding, plan recognition, and device diagnosis which demonstrate that our algorithm is at least an order of magnitude faster than an alternative abduction algorithm that does not use an ATMS.

1 Introduction

Abduction is an important reasoning process underlying many tasks such as diagnosis, plan recognition, text understanding, and theory revision. The standard logical definition of abduction is: Given a set of axioms \( T \) (the domain theory), and a conjunction of atoms \( O \) (the observations), find a minimal set of atoms \( A \) (the assumptions) such that \( A \cup T \models O \) (where \( A \cup T \) is consistent). A set of assumptions together with its corresponding proof of the observations is frequently referred to as an explanation of the observations. Although recent research has seen the development of several special purpose abduction systems (e.g., [de Kleer and Williams, 1987]) and some theoretical analysis of the problem ([Levesque, 1989, Selman and Levesque, 1990]), there has not been much emphasis on developing practical algorithms for the general problem. Existing algorithms tend to be too restrictive, too inefficient, or both.

The first problem is generality. [Levesque, 1989] has shown that the ATMS [de Kleer, 1986a] is a general abduction algorithm for propositional Horn-clause theories. However, many interesting abduction tasks require the expressibility of first-order predicate logic. In first-order logic, the important operation of unifying assumptions (factoring) becomes relevant. Frequently, simple and coherent explanations can only be constructed by unifying initially distinct assumptions so that the resulting combined assumption explains several observations [Pople, 1973, Stickel, 1988, Ng and Mooney, 1990]. This important problem does not arise in the propositional case.

The second problem is efficiency. The general purpose abduction algorithm proposed in [Stickel, 1988] can perform a great deal of redundant work in that partial explanations are not cached and shared among multiple explanations. The ATMS algorithm, though it caches and reuses partial explanations in order to avoid redundant work, has not been extended to perform general first-order abduction. Also, the ATMS algorithm of [de Kleer, 1986a] exhaustively computes all possible explanations which is computationally very expensive for large problems. Even in the propositional case, computing all minimal explanations is a provably exponential problem [Selman and Levesque, 1990]. This indicates that resorting to heuristic search is the most reasonable approach to building a practical abduction system.

This paper presents an implemented algorithm for first-order Horn-clause abduction that uses an ATMS to cache intermediate results and thereby avoid redundant work. The most important additions to the ATMS involve handling the unification of assumptions. The system also incorporates a form of heuristic beam search which can be used to focus the ATMS on promising explanations and avoid the intractable problem of computing all possible explanations. We have evaluated our algorithm on a range of abduction problems from such diverse tasks as text understanding, plan recognition, and device diagnosis. The empirical results illustrate that the resulting system is significantly faster than a non-ATMS alternative, specifically, the abduction algorithm proposed in [Stickel, 1988]. In particular, we show that even when.

*This research is supported by the NASA Ames Research Center under grant NCC-2-429. The first author is also supported by an IBM Graduate Fellowship. Equipment used was donated by Texas Instruments.
using heuristic search to avoid computing all possible explanations, the caching performed by the ATMS increases efficiency by at least an order of magnitude.

2 Problem Definition

The abduction problem that we are addressing can be defined as follows.

Given:

- A set of first-order Horn-clause axioms $T$ (the domain theory), where an axiom is either of the form $\forall v_1, \ldots, v_k \, C \leftarrow P_1 \land \ldots \land P_r$ (a rule), or $\forall v_1, \ldots, v_k \, F$ (a fact)
- $k \geq 0, r > 0, C, P_i, F$ are atoms containing the variables $v_1, \ldots, v_k$
- An existentially quantified conjunction $O$ of atoms (the input atoms) of the form $\exists v_1, \ldots, v_k \, O_1 \land \ldots \land O_m$
  - $k > 0, m > 0, O_i$ are atoms containing the variables $v_1, \ldots, v_k$

Find:

All explanations with minimal (w.r.t. variant-subset) sets of assumptions.

We define "explanations", "minimal", and "variant-subset" as follows. Let $A$ (the assumptions) be an existentially quantified conjunction of atoms of the form $\exists v_1, \ldots, v_k \, A_1 \land \ldots \land A_n$ where $k \geq 0, n \geq 0, A_i$ are atoms containing the variables $v_1, \ldots, v_k$ such that $A \cup T \models O$ and $A \cup T$ is consistent. An assumption set $A$ together with its corresponding proof is referred to as an explanation (or an abductive proof) of the input atoms. We will write $A$ as the set $\{A_1, \ldots, A_n\}$ with the understanding that all variables in the set are existentially quantified and that the set denotes a conjunction. We define an assumption set $A$ to be a variant-subset of another assumption set $B$ if there is a renaming substitution $\sigma$ such that $A \sigma \subseteq B$. The abduction task is to find the set $S$ of all minimal explanations such that there is no explanation in $S$ whose assumption set is a variant-subset of the assumption set of another explanation in $S$.

Since the definition of abduction requires consistency of the assumed atoms with the domain theory, the abduction problem is in general undecidable. We assume in this paper that consistency is checked in the following way: find the logical consequences of $A \cup T$ via forward-chaining of some Horn-clause axioms (some of them are of the form $P_1 \land \ldots \land P_r \rightarrow \text{FALSITY}$) up to some preset depth limit. If FALSITY is not derived, we assume that $A \cup T$ is consistent.

3 The SAA Algorithm

[Stickel, 1988] has proposed an algorithm for computing the set of all first-order Horn-clause abductive proofs. His algorithm, which we will call SAA, operates by applying inference rules to generate goal clauses. The initial goal clause is the input atoms $O_1, \ldots, O_m$. Each atom in a goal clause can be marked with one of proved, assumed, or unsolved. All atoms in the initial goal clause are marked as unsolved. A final goal clause must consist entirely of proved or assumed atoms.

Let $G$ be a goal clause $Q_1, \ldots, Q_n$, where the leftmost unsolved atom is $Q_i$. The algorithm SAA repeatedly applies the following inference rules to goal clauses $G$ with unsolved atoms:

- Resolution with a fact. If $Q_i$ and a fact $F$ are unifiable with a most general unifier (mgu) $\sigma$, the goal clause $Q_1 \sigma, \ldots, Q_n \sigma$ can be derived, where $Q_i \sigma$ is marked as proved.
- Resolution with a rule. Let $C \leftarrow P_1 \land \ldots \land P_r$ be a rule where $Q_i$ and $C$ are unifiable with a mgu $\sigma$. Then the goal clause $Q_1 \sigma, \ldots, Q_{i-1} \sigma, P_i \sigma, Q_i \sigma, \ldots, Q_n \sigma$ can be derived, where $Q_i \sigma$ is marked as proved and each $P_k \sigma$ is marked as unsolved.
- Making an assumption. If $Q_i$ is assumable, then $Q_1, \ldots, Q_n$ can be derived with $Q_i$ marked as assumed.
- Factoring with an assumed atom. If $Q_j$ is marked as assumed, $j < i$, $Q_i$ and $Q_j$ are unifiable with a mgu $\sigma$, the goal clause $Q_1 \sigma, \ldots, Q_{i-1} \sigma, Q_{i+1} \sigma, \ldots, Q_n \sigma$ can be derived.

[Stickel, 1988] also proposed the use of a cost metric to rank and heuristically search the more promising explanations first. All facts and rules in the knowledge base as well as assumed atoms are assigned costs. The best explanation is one with the least cumulative cost.

Before we proceed, we note that the explanations generated by the SAA algorithm may include some that are variant-subsets of another. For instance, given the following axioms

\begin{align*}
\text{(inst ?g going)} & \leftarrow \text{(inst ?s shopping)} \land \text{(go-step ?s ?g)} \\
\text{(goer ?g ?p)} & \leftarrow \text{(inst ?s shopping)} \land \text{(go-step ?s ?g)} \land \text{(shopper ?s ?p)}
\end{align*}

and the input atoms (inst go1 going) and (goer go1 john1), we can derive the explanation $F$ with assumptions $AF = \{\text{(inst ?x shopping)}, \text{(go-step ?x go1)}, \text{(inst ?y shopping)}, \text{(go-step ?y go1)}, \text{(shopper ?y john1)}\}$ by backward-chaining on the two axioms. Applying the factoring operation, we can obtain another explanation $E$ with assumptions $AE = \{\text{(inst ?x shopping)}, \text{(go-step ?x go1)}\}$. Note that although $AE \not\subseteq AF$, $AE \sigma \subseteq AF$ with the renaming substitution $\sigma = \{?x/?)$. Since explanations that are variant-supersets of other explanations are essentially redundant, they need to be eliminated. Unfortunately, it can be readily shown that determining variant-subset relation is an NP-complete problem by reduction from directed

\[ ^1 \text{Actually, the algorithm as presented in [Stickel, 1988] allows for unifying \(Q_i\) with a proved atom as well. However, it appears that in practice, omitting factoring with proved atoms does not cause any loss of good explanations while saving some redundant inferences.} \]

\[ ^2 \text{Variables are denoted by preceding them with a "?".} \]
subgraph isomorphism. This introduces yet another source of computational complexity when finding minimal explanations in a first-order Horn-clause theory.

4 The Basics of the ATMS

The Assumption-based Truth Maintenance System (ATMS) [de Kleer, 1986a] is a general facility for managing logical relationships among propositional formulas. An ATMS maintains multiple contexts at once and is particularly suited for problem solving that involves constructing and comparing multiple explanations.

Each problem solving datum is associated with a node in the ATMS. A node in the ATMS can be further designated as an assumption. Nodes are related via justifications. A justification is a propositional Horn-clause of the form \( a_1 \land \ldots \land a_n \rightarrow c \), where each \( a_i \) is an antecedent node and \( c \) is the consequent node. A restriction is that assumptions cannot be further justified. An environment is a set of assumptions. Associated with each node is a set of environments called its label. The ATMS supports several operations, including adding a node, making an assumption, and adding a justification.

[de Kleer, 1986b] also proposed the use of a problem-solver-ATMS interface called the consumer architecture. A consumer is essentially a rule-like mechanism that is invoked by a set of nodes. It checks whether some precondition governing this set of nodes is satisfied, and if so, the consumer fires and performs some problem solving work. For example, first-order Horn-clause forward-chaining can be implemented using consumers as follows. Define a class to be an ATMS construct representing a set of nodes with some common characteristics. Let Class(C) denote the class representing all nodes whose data have the same predicate symbol as the atom \( C \). For instance, \( \text{Class}(P(x)) = \{P(a), P(f(b)), \ldots\} \). Then a single consumer can implement a forward-chaining Horn-clause axiom such as \( P(x) \land Q(x) \rightarrow R(x) \) by actively looking for nodes with matching arguments in the classes \( \text{Class}(P(x)) \) and \( \text{Class}(Q(x)) \). If the consumer finds two such nodes, say \( P(a) \) and \( Q(a) \), then its precondition is satisfied. It then performs the forward-chaining work by inferring the node \( R(a) \), and adding the justification \( P(a) \land Q(a) \rightarrow R(a) \) to the ATMS.

5 The AAA Algorithm

We now present our ATMS-based, first-order, Horn-clause abduction algorithm AAA. Basically, the algorithm is much like the implementation of a first-order Horn-clause forward-chaining system in the consumer architecture discussed above, except that the inference direction is now reversed. We also have the additional operation of unifying assumptions (factoring). Our goal is to construct an algorithm similar to SAA, but one that relies on caching justifications and sharing them among different explanations.

First of all, note that in SAA, an unsolved atom in a goal clause can either be assumed or backward-chained on by some rule. However, in an ATMS, a node, once assumed, can never be further justified by some Horn-clause rule. To get around this restriction, when we want to assume an atom \( D \) with node \( n \), we create a similar assumption node \( n' \) and add the justification \( n' \rightarrow n \) (Table 1).

For every Horn clause axiom \( A_1 \land \ldots \land A_n \rightarrow C \), we create a consumer that is attached to every node in Class\( \{C\} \). The body of this consumer is shown in Table 1. We assume there is some preset backward-chain depth bound so as to prevent infinite backward-chaining on recursive rules. We also create a fact consumer for each fact.

### Algorithm AAA

**Assume an atom \( D \)**

If \( D \) is assumable then
- Let \( n \) be the node with datum \( D \)
- Let the predicate symbol of \( D \) be \( P \)
- Create another node \( n' \) whose datum \( D' \) is the same as \( D \) except that the predicate symbol of \( D' \) is \( A \)
- Make node \( n' \) an ATMS assumption node
- Add the justification \( n' \rightarrow n \)

**Backward-chain consumer**

(encode a backward-chaining axiom \( A_1 \land \ldots \land A_n \rightarrow C \))

For every node \( n \) newly added to Class\( \{C\} \)

If \( n \) is not an assumption node and
- backward-chain depth bound is not exceeded and \( n \) unifies with \( C \) with a mgu \( \sigma \) and
- \( \sigma \) does not instantiate any variable in \( n \) then
- \( A_1' := A_1 \sigma, \ldots, A_n' := A_n \sigma, C' := C \sigma \)
- Add the justification \( A_1' \land \ldots \land A_n' \rightarrow C' \)
- Assume the atoms \( A_1', \ldots, A_n' \)

**Fact consumer**

(encode resolution with a fact \( F \))

For every node \( n \) newly added to Class\( \{F\} \)

If \( n \) unifies with \( F \) with a mgu \( \sigma \) and
- \( \sigma \) does not instantiate any variable in \( n \) then
- Make \( n \) a fact (i.e., let its label be \( \{\} \))

### Algorithm AAA

Add the justification \( O_1 \land \ldots \land O_m \rightarrow \text{GOAL} \)

Assume the atoms \( O_1, \ldots, O_m \)

Run the fact consumers, backward-chain consumers, and forward-chain consumers

For every environment \( e \) in the label of \( \text{GOAL} \)

If factoring of assumptions has not been performed on \( e \) then

- For all pairs of assumptions \( a_i \) and \( a_j \) in \( e \)
  - If \( a_i \) and \( a_j \) are unifiable with a mgu \( \sigma \)
    - \( e' := e \sigma \)
    - Run forward-chain consumers but restricted to forward-chaining on the assumptions in \( e' \) (to check its consistency)

Eliminate environments that are variant-supersets of other environments

Table 1: The AAA Algorithm

---

**Table 1: The AAA Algorithm**
For simplicity, we assume in the construction of AAA that any inconsistent assumption set can be detected by forward-chaining on a single axiom of the form \( A_1 \land \ldots \land A_n \rightarrow \text{FALSITY} \). Forward-chain consumers as described in the last section are used to encode such forward-chaining axioms.

The AAA algorithm first computes all explanations that are obtained by resolution with facts, resolution with backward-chaining rules, and making assumptions. The last step in the algorithm performs factoring of assumptions in explanations. The resulting environments in the label of the \( \text{GOAL} \) node are all the minimal explanations of the input atoms \( O_1 \land \ldots \land O_m \).

The AAA algorithm is incomplete in the sense that some explanations that will be computed by SAA will not be computed by AAA. Specifically, such missed explanations are those that are obtained when, during resolution of an atom in a goal clause with a fact or the consequent of a rule, the most general unifier is such that variables in the goal clause get instantiated. However, for all the problems that we have tested AAA on, this incompleteness does not pose a problem. This is because the explanations we seek are constructed by chaining together general rules to explain specific ground facts. Aside from this incompleteness, we believe that our AAA algorithm computes all other explanations that the SAA algorithm computes. We are currently working on a formal proof of this equivalence. We also plan to explore the construction of a version of the AAA algorithm that is complete.

We have implemented and tested both algorithms. The empirical results section presents some data comparing their performance. The actual performance of both algorithms indicate very clearly that computing all minimal explanations is simply too explosive to be practically useful.

6 The Heuristic Algorithms

In this section, we present our heuristic algorithm AAA/H that uses beam search to cut down on the search space. To facilitate comparison, we also constructed a heuristic version of SAA called SAA/H. The differences between SAA/H and SAA are:

1. SAA/H is an incremental algorithm that constructs explanations one input atom at a time.
2. Instead of searching the entire search space, we employ a form of beam search. We restrict the number of goal clauses to \( \beta_{\text{intra}} \) during the incremental processing of an input atom. The search terminates when there are \( \beta_{\text{intra}} \) number of final goal clauses. The number of final goal clauses carried over to the processing of the next input atom is \( \beta_{\text{inter}} \), where \( \beta_{\text{inter}} \leq \beta_{\text{intra}} \).
3. Each goal clause is assigned a simplicity metric defined as \( E/A \), where \( E \) is the number of input atoms explained (i.e., atoms marked as proved) and \( A \) is the number of assumptions in a goal clause. Goal

For each input atom \( O_i, i = 1, \ldots, m \)
Add the justification \( \text{GOAL}_{i-1} \land O_i \rightarrow \text{GOAL}_i \)
(or \( O_i \rightarrow \text{GOAL}_i \) if \( i = 1 \))
Loop

Run forward-chaining consumers
Let \( L(\text{GOAL}_i) \) be the label of \( \text{GOAL}_i \)
Order the environments in \( L(\text{GOAL}_i) \)
Retain the best (simplest) \( \beta_{\text{intra}} \) number of environments in \( L(\text{GOAL}_i) \)
Unify the assumptions of the environments in \( L(\text{GOAL}_i) \)
If the number of environments in \( L(\text{GOAL}_i) \geq \beta_{\text{intra}} \) then
exit the loop
Execute backward-chaining and resolution with facts
If no backward-chaining occurs then
exit the loop
End of loop
Reduce the number of environments in \( L(\text{GOAL}_i) \) to the best \( \beta_{\text{inter}} \) environments

Table 2: The AAA/H Algorithm

Clauses with the highest simplicity metric values are processed first.

Analogous changes are made to the AAA algorithm such that AAA/H is an incremental algorithm that uses beam search to restrict the search space explored. To achieve incrementality, the AAA/H algorithm adds the justification \( O_i \rightarrow \text{GOAL}_1 \) when processing the first input atom \( O_i \). Subsequently, adding the input atom \( O_i \) results in adding the justification \( \text{GOAL}_{i-1} \land O_i \rightarrow \text{GOAL}_i \). By doing so, explanations of the input atoms \( O_1 \land \ldots \land O_i \) are exactly the environments in the label of the node \( \text{GOAL}_i \).

To implement beam search, the algorithm AAA/H uses the idea of focusing [Forbus and de Kleer, 1988, Dressler and Farquhar, 1990] to restrict the amount of work done by the ATMS. There are two main uses of focusing: to discard unwanted environments and to only search for those interesting (simplest) environments. When the algorithm decides to keep only \( \beta_{\text{inter}} \) number of environments, all other environments with higher simplicity metric values are removed from the labels of all nodes. To ensure that only the simplest environments are searched, each time a node is assumed, it is added to a list of focus assumptions and label propagation is such that only the focused environments get propagated.

Since we intend AAA/H to be an efficient and practical algorithm, we no longer assume in the construction of AAA/H that any inconsistency can be detected by forward-chaining on one axiom. That is, an assumption set may imply FALSITY via forward-chaining on several axioms with the last axiom having FALSITY as its consequent. Once we do away with this assumption, we are faced with the problem that any axiom of the form \( A_1 \land \ldots \land A_n \rightarrow C \) in the knowledge base can potentially be
used in two ways, backward-chaining during abduction and forward-chaining during consistency checking. To minimize duplicating work, we now implement backward-chaining by having the consequent node "suggest" the assertion of the antecedent nodes, and letting the forward-chaining consumers add the actual justification linking the antecedent nodes to the consequent node. The algorithm AAA/H is given in Table 2.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time (min)</th>
<th>#Justifications</th>
<th>#E</th>
</tr>
</thead>
<tbody>
<tr>
<td>hare</td>
<td>0.20</td>
<td>5.00</td>
<td>5</td>
</tr>
<tr>
<td>snake</td>
<td>3.57</td>
<td>10.94</td>
<td>3.28</td>
</tr>
<tr>
<td>fly</td>
<td>180+</td>
<td>62.78</td>
<td>2.8+</td>
</tr>
<tr>
<td>shop1</td>
<td>5.68</td>
<td>2.26</td>
<td>2.51</td>
</tr>
<tr>
<td>mean</td>
<td>3.41</td>
<td>12.62</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Empirical results comparing SAA and AAA

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time (min)</th>
<th>#Justifications</th>
<th>#E</th>
</tr>
</thead>
<tbody>
<tr>
<td>hare</td>
<td>1.38</td>
<td>0.13</td>
<td>10.62</td>
</tr>
<tr>
<td>snake</td>
<td>9.40</td>
<td>1.59</td>
<td>5.91</td>
</tr>
<tr>
<td>fly</td>
<td>12.68</td>
<td>1.66</td>
<td>7.64</td>
</tr>
<tr>
<td>work</td>
<td>8.70</td>
<td>0.33</td>
<td>26.36</td>
</tr>
<tr>
<td>court</td>
<td>10.85</td>
<td>0.57</td>
<td>19.04</td>
</tr>
<tr>
<td>move</td>
<td>11.53</td>
<td>0.76</td>
<td>15.17</td>
</tr>
<tr>
<td>copy</td>
<td>12.78</td>
<td>0.83</td>
<td>15.40</td>
</tr>
<tr>
<td>replace</td>
<td>12.28</td>
<td>0.93</td>
<td>13.20</td>
</tr>
<tr>
<td>paper</td>
<td>14.21</td>
<td>1.06</td>
<td>13.41</td>
</tr>
<tr>
<td>graph</td>
<td>14.58</td>
<td>1.17</td>
<td>12.46</td>
</tr>
<tr>
<td>ckt1</td>
<td>1.74</td>
<td>0.13</td>
<td>13.38</td>
</tr>
<tr>
<td>ckt2</td>
<td>3.04</td>
<td>0.09</td>
<td>33.78</td>
</tr>
<tr>
<td>ckt3</td>
<td>1.12</td>
<td>0.07</td>
<td>16.00</td>
</tr>
<tr>
<td>mean</td>
<td>6.66+</td>
<td>0.17</td>
<td>39.18+</td>
</tr>
</tbody>
</table>

Table 4: Empirical results comparing SAA/H and AAA/H

7 Empirical Results

In this section, we present our experimental results on running the various algorithms SAA, AAA, SAA/H, and AAA/H on a range of abduction problems from such diverse tasks as text understanding, plan recognition, and device diagnosis. We measure the performance and the amount of computation expended by the algorithms using two metrics: the runtime and the number of justifications (i.e., number of rule invocations) made. The results are presented in Table 3 and 4.3 The full-search data in Table 3 also gives the total number of minimal explanations (#E) for each problem. All runtimes are actual execution times on a Texas Instruments Explorer/2 Lisp machine. Note that the beam widths for SAA/H and AAA/H are set to the minimum values such that the best explanation is found for every problem in the set. For the current set of problems, $\beta_{\text{inter}} = 4$, $\beta_{\text{intra}} = 20$ for SAA/H, and $\beta_{\text{inter}} = 4$, $\beta_{\text{intra}} = 16$ for AAA/H.4

The text understanding problems include expository text examples about explaining the coloration of various animals (hare, snake, fly, bird) and narrative text examples about understanding the intention of someone entering a supermarket (shop1, shop2, work, court). The plan recognition problems (move, copy, replace, backup, paper, graph) involve recognizing UNIX users' plans from a sequence of primitive file commands. The device diagnosis problems include several simple logic circuit examples (ckt1, ckt2, ckt3) and a full adder example (adder). In the diagnosis problems, we restricted the assumable atoms to be those with one of the two predicates norm (the component is normal) or ab (the component is abnormal). (i.e., we use predicate-specific abduction [Stickel, 1988]). In the rest of the problems, all atoms are assumable.

To give a sense of the size of our problems and the knowledge base used, there is a total of 46 KB facts, 163 KB rules, and 110 taxonomy-sort symbols. The average number and maximum number of antecedents per rule are 2.8 and 6 respectively, and the average number of input atoms per problem is 5.6. (See [Ng and Mooney, 1991] for a complete listing of the knowledge base and the examples used.)

The empirical results indicate that AAA outperforms SAA and AAA/H outperforms SAA/H on the set of problems we ran. AAA runs about 3 times as fast as SAA on the few problems we were able to test. Due to the intractability of computing all minimal explanations, we were unable to run the full-search algorithms on the other problems, which clearly require heuristic search. Even for the simple problems for which we have comparative data, the heuristic versions are about 10 times faster than the full-search versions while still finding the simplest explanation (SAA/H is 5 times faster than SAA, AAA/H is 16 times faster than AAA). Comparing AAA/H with SAA/H, we see that there is on average at least an order of magnitude speedup on the problems that we have tested.

We believe our results are particularly significant because, to the best of our knowledge, this is the first empirical validation that an ATMS-based first-order

---

3 The "+" signs in Table 3 and 4 indicate that the corresponding examples take much longer to run and we only recorded the time taken and the number of justifications made at the time the program was aborted.

4 Due to the different ordering in which explanations are generated in both algorithms, the minimum beam widths for which the best explanations are found in both algorithms need not be the same.

5 Every taxonomy-sort symbol p will add an axiom (in addition to the 163 KB rules) of the form (inst ?x p) → (inst ?x supersort-of-p)
abduction algorithm employing caching of explanations performs better than a non-ATMS alternative even when pruning heuristics are used to find a relatively small number of good explanations. We also want to stress that although the AAA algorithm is incomplete compared to the SAA algorithm, this does not affect our comparison since on all the problems that we tested, the situation in which a most general unifier actually instantiates the datum of a node does not arise.

8 Related Work
As mentioned in the introduction, previous algorithms for automated abduction have been either too restrictive or too inefficient. Previous research on the ATMS [de Kleer, 1986a, Levesque, 1989] and its use in device diagnosis [de Kleer and Williams, 1987] has been propositional in nature. In particular, it has not dealt with the problem of unifying assumptions which occurs in general first-order abduction. There has been some previous work on focusing the ATMS on a subset of interesting environments [Forbus and de Kleer, 1988, Dressler and Farquhar, 1990]; however, this work was not in the context of general first-order abduction and did not specifically involve using focusing to perform beam search. Also, although [Dressler and Farquhar, 1990] has empirical results comparing focused ATMS performance with that of non-focused ATMS (thus demonstrating that limited search is better than complete search), there has been no previous work comparing limited search ATMS implementation with a limited search non-ATMS alternative. Finally, other systems have not been systematically tested on such a wide range of abduction problems from text understanding, plan recognition, and device diagnosis.

9 Conclusion
In this paper, we have presented a new algorithm for first-order Horn-clause abduction called AAA. The AAA algorithm uses an ATMS to avoid redundant computation by caching and reusing partial explanations. By comparison, previous abduction algorithms are either less general or less efficient. Since computing all minimal explanations is intractable, we also developed a heuristic beam-search version of AAA, which computes a subset of the simplest explanations. In order to evaluate AAA and AAA/H, we performed a comprehensive set of experiments using a broad range of abduction problems from text understanding, plan recognition, and device diagnosis. The results conclusively demonstrate that our algorithm is at least an order of magnitude faster than an abduction algorithm which does not employ an ATMS.

Acknowledgements
Thanks to Adam Farquhar for allowing us to use his ATMS code and for discussing technical details of the ATMS in the early stages of this research. Thanks to Siddarth Subramanian for writing some of the axioms for the diagnosis examples.

References