Northwestern University

The Institute for the Learning Sciences

CATMS: AN ATMS WHICH AVOIDS LABEL EXPLOSIONS

Technical Report # 13 • May, 1991

Dennis DeCoste
John W. Collins

Established in 1989 with the support of The Arthur Andersen Worldwide Organization

CATMS: AN ATMS WHICH AVOIDS LABEL EXPLOSION

Dennis DeCoste
John W. Collins

May, 1991

The Institute for the Learning Sciences
Northwestern University
Evanston, IL 60201

This research was supported by NASA Langley, under contract NASA-NAG-11023. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech, an Institute Partner, and from IBM.

CATMS: An ATMS Which Avoids Label Explosions

Dennis DeCoste

Qualitative Reasoning Group
Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue
Evanston, Illinois 60201
email: decoste@ils.nwu.edu

John W. Collins

Qualitative Reasoning Group
Beckman Institute
University of Illinois
605 North Mathews Street
Urbana, Illinois 61801
email: jcollins@cs.uiuc.edu

Abstract

Assumption-based truth maintenance systems have developed into powerful and popular means for considering multiple contexts simultaneously during problem solving. Unfortunately, standard ATMS node labels tend to grow combinatorically as problem complexity increases. In this paper, we present a new ATMS algorithm (CATMS) which avoids the problem of label explosions, while preserving most of the query-time efficiencies resulting from label compilations. CATMS generalizes the standard ATMS subsumption relation, allowing it to compress an entire label into a single assumption. These *compressions* of labels are balanced by *expansions* of environments to include any implied assumptions. The result is a new dimension of flexibility, allowing CATMS to trade-off the query-time efficiency of uncompressed labels against the costs of computing them. To demonstrate the significant computational gains of CATMS over de Kleer's ATMS, we compare the performance of the ATMS-based QPE [9] problem-solver using each.

A condensed version of this report appears in the Proceedings of the Tenth National Conference on Artificial Intelligence, July 1991.

1 Introduction

Assumption-based truth maintenance systems [3] have proven useful for reasoning about multiple contexts, especially in domains such as qualitative physics [9,7] and signal processing [15,12]. Each context indicates a different set of assumptions (i.e., choices) made during problem solving. An ATMS reasons with multiple contexts simultaneously, with larger contexts inheriting inferences made in smaller ones, so that the cost of reasoning is amortized over the set of contexts. The power of an ATMS comes from its compiling of inferences into *labels*, each representing the alternative contexts in which a particular proposition is true. Although such compilations can potentially lead to enormous efficiency gains at query time, those efficiencies may be unrealized due to an ATMS problem called *label explosion*. This problem arises during compilation when labels grow exponentially large. The potential for label explosion is a consequence of the cross-product nature of label computations.

In this paper, we present a new ATMS algorithm (CATMS) which can avoid the problem of label explosions, while preserving most of the query-time efficiencies resulting from label compilations. CATMS generalizes the standard ATMS notion of *subsumption*, allowing it to compress an entire label into a single assumption. This *compression* of labels is balanced by an *expansion* of environments to include implied assumptions. The result is an added flexibility to trade-off the query-time efficiency of uncompressed labels against the costs of computing them. In short, CATMS provides an efficient hybrid of the compiled approach taken by a traditional ATMS and the interpreted approach noted in [16].

Section 2 briefly provides the ATMS background relevant to this paper. Section 3 explains the key idea underlying the CATMS algorithm: our theory of label compression. Section 4 presents the basic CATMS algorithm and Section 5 describes how it can detect and avoid label explosions. Section 6 describes some refinements of the basic CATMS algorithm which further improve its efficiency. Section 7 demonstrates the significant computational gains allowed by CATMS over de Kleer's ATMS, by comparing their performance for the ATMS-based QPE [9] problem-solver. Finally, Section 8 discusses the implications of CATMS to future ATMS technology.

2 ATMS Background

Briefly, an ATMS can be characterized as follows. Each proposition considered by the problem solver is assigned an ATMS *node*. As the problem solver derives relations among propositions, it declares them to the ATMS as *clauses* among nodes. A particular ATMS implementation may support general CNF clauses or just Horn clauses. The problem solver declares a subset of the nodes to be *assumptions* — propositions which are assumed to be true. Each set of assumptions indicates a particular context and is represented as an ATMS *environment*. Throughout the paper, we let E, E_i, E_j refer to environments. We denote the assumptions comprising E by *Asns(E)*. A node is considered *true* in E if it can be propositionally deduced from *Asns(E)* together with the clauses C. A node is considered *false* in E if its *negation node* is true in E. The ATMS *contradiction node* denotes a logical contradiction. It is inferred whenever a node is both true and false in some E, in which case E is called *nogood* (i.e., inconsistent). The primary responsibility of an ATMS is to process an incremental stream of nodes ($\mathcal{N}$), assumptions ($\mathcal{A}$), and clauses (C) to efficiently answer queries about the status (i.e., true, false, or unknown) of some node in a given environment, or about the consistency of some environment.

An important concept in any ATMS is that of subsumption among environments:

Definition 2.1 (Environment Subsumption) E_i subsumes $E_j \equiv Asns(E_i) \subseteq Asns(E_j)$.

Because an ATMS operates monotonically, all nodes true in E_i are also true in the subsumed E_j, as long as E_j is consistent. Thus, one can avoid making redundant inferences by inheriting inferences from subsuming environments. However, the typical problem solver rarely requires an explicit list of all nodes true in some environment. So, instead of caching implied nodes with each environment, an ATMS caches for each node the minimal set of implying environments, called its *label*.

Definition 2.2 (Node Label) *Let Label(N) denote the set of environments for node N satisfying:*

1. **Consistency:** *No $E_i \in$ Label(N) is a nogood;*
2. **Soundness:** *N is true in each $E_i \in$ Label(N);*
3. **Completeness:** *If N is true in E_j, then some $E_i \in$ Label(N) subsumes E_j;*
4. **Minimality:** *For E_i, $E_j \in$ Label(N) and $E_i \neq E_j$, E_i does not subsume E_j.*

By compiling node labels, an ATMS can efficiently determine whether a node N is true in a given E by checking whether some E_i in *Label(N)* subsumes E.

As assumptions and clauses are given to an ATMS, labels are incrementally maintained through a technique called *boolean constraint propagation* (BCP) [5], which ensures that each clause is locally satisfied. This process involves computing *cross-products* of labels and unioning the results with the existing label. Due to the nature of cross products, ATMS labels have the potential to grow exponentially, resulting in label explosions.

Conceptually, an ATMS maintains a special label for the contradiction node, where consistency is not enforced. This label represents the set of minimal nogoods. Thus, an ATMS can determine whether a given E is nogood by checking whether E is subsumed by some E_i in the label of the contradiction node. To maintain label consistency, an ATMS must further provide a means for removing minimal nogoods, and any environments subsumed by them, from all other labels.

Let $\mathcal{E}$ denote the union of the labels of all nodes, plus any environments mentioned in queries posed by the problem solver. $\mathcal{E}$ indicates the set of all environments which must be explicitly represented by an ATMS in order to reason over all possible contexts. Typically, $\mathcal{E}$ will be much smaller than the worst-case (the power set of all assumptions), due to the consistency and minimality properties of labels.

3 Theory of Label Compression

CATMS generalizes the standard ATMS algorithm by generalizing the notion of environment subsumption. This impacts the minimality and completeness requirements for labels, as explained below.

3.1 Assumption Closures

In CATMS, an important property of each environment is its *assumption closure* — the set of assumptions which logically follow from its base assumptions:

Definition 3.1 (Assumption Closure) $Closure(E) \equiv \{A_i \in \mathcal{A} \mid Asns(E) \cup \mathcal{C} \models A_i\}$.

An obvious property of these closures is that they include the base assumptions:

Property 3.1 (Base Assumption Inclusion) $Asns(E) \subseteq Closure(E)$.

It is possible for two environments with different base assumptions to have identical closures:

Definition 3.2 (Equivalent Environments) E_i *is* equivalent *to* $E_j \equiv Closure(E_i) = Closure(E_j)$.

Recall that in a standard ATMS, two environments are compared for subsumption by comparing the (base) assumption sets comprising them. In CATMS, subsumption is based on assumption closures:

Definition 3.3 (Closure Subsumption) E_i c-subsumes $E_j \equiv Closure(E_i) \subseteq Closure(E_j)$.

Closure subsumption is a generalization of standard ATMS subsumption, due to transitivity and the nature of closures:

Property 3.2 (Subsumption Generalized) *If* E_i *subsumes* E_j *then* E_i *c-subsumes* E_j*; alternatively, if* $Asns(E_i) \subseteq Asns(E_j)$ *then* $Closure(E_i) \subseteq Closure(E_j)$.

Proof of Property 3.2:
Assume $Closure(E_i) \not\subseteq Closure(E_j)$ and $Asns(E_i) \subseteq Asns(E_j)$.
Then $\exists\, A \in Closure(E_i) \mid A \notin Closure(E_j)$. $Asns(E_i) \cup \mathcal{C} \models A$ (by Definition 3.1).
But $Asns(E_i) \subseteq Asns(E_j)$, so $Asns(E_j) \cup \mathcal{C} \models A$ (by monotonicity). So $A \in Closure(E_j)$ (by Definition 3.1), which is a contradiction. Q.E.D.
Properties 3.1 and 3.2 suggest a test for closure subsumption that requires a single closure computation:

Property 3.3 (Closure Subsumption Test) *E_i c-subsumes E_j iff $Asns(E_i) \subseteq Closure(E_j)$.*

Proof of Property 3.3:

$\Rightarrow$:

$Asns(E_i) \subseteq Closure(E_i)$ (by Property 3.1). $Closure(E_i) \subseteq Closure(E_j)$ (by Definition 3.3). $Asns(E_i) \subseteq Closure(E_j)$ (by transitivity of $\subseteq$).

$\Leftarrow$:

Assume $Closure(E_i) \not\subseteq Closure(E_j)$ and $Asns(E_i) \subseteq Closure(E_j)$.
Then $\exists\ A \in Closure(E_i) \mid A \notin Closure(E_j)$. $Asns(E_i) \cup C \models A$ (by Definition 3.1). But $Asns(E_i) \subseteq Closure(E_j)$, so $Closure(E_j) \cup C \models A$ (by monotonicity). So $A \in \{A_k \in \mathcal{A} \mid Closure(E_j) \cup C \models A_k\}$ (by Definition 3.1). However, $\{A_k \in \mathcal{A} \mid Asns(E_j) \cup C \models A_k\} = \{A_k \in \mathcal{A} \mid Closure(E_j) \cup C \models A_k\}$, since $Closure(E_j) \equiv \{A_k \in \mathcal{A} \mid Asns(E_j) \cup C \models A_k\}$ (by Definition 3.1). Therefore, $A \in \{A_k \in \mathcal{A} \mid Asns(E_j) \cup C \models A_k\}$. So $A \in Closure(E_j)$ (by Definition 3.1), which is a contradiction. Q.E.D.

3.2 Using Assumption Closures

Applying CATMS's notion of closure subsumption to the ATMS minimality property for labels can result in CATMS labels being smaller than the corresponding labels in a standard ATMS. For example, this new notion of minimality prevents a label from containing two equivalent environments. We use the term *compression* to refer to the act of removing from a standard ATMS label those environments which are c-subsumed by others in that label. A compressed label retains the soundness and consistency of the original label since no environments are added to the label by such compression.

Viewed in isolation, compressed labels violate the completeness requirement for labels; however, label completeness is, in effect, reestablished by using closure subsumption for queries as well. More formally, if E_i is in $Label(N)$ in a standard ATMS and E_i subsumes query environment E_q (i.e., N is true in E_q), then after compression there will still be some env in $Label(N)$ that c-subsumes E_q. *Proof:*

either E_i is still in *Label(N)* after compression or $\exists\ E_k \in$ *Label(N)* $\mid E_k$ c-subsumes E_i. If $E_i \in$ *Label(N)*, then E_i c-subsumes E_q (by Property 3.2). Otherwise, E_k c-subsumes E_q (by transitivity through E_i). Q.E.D.

Figure 1 shows an example justification structure and the corresponding node labels that CATMS and a standard ATMS would respectively compute from it. In this example, there are no equivalent environments since environments can only be equivalent when there are cycles in the justifications of assumptions. However, several environments c-subsume others that would not subsume them in the standard ATMS sense. For instance, environment $\{A_7A_8\}$ c-subsumes $\{A_7A_5A_6\}$, since $\{A_5A_6\}$ implies A_8. This example illustrates how CATMS's compressed labels can be much smaller than the standard ATMS labels.

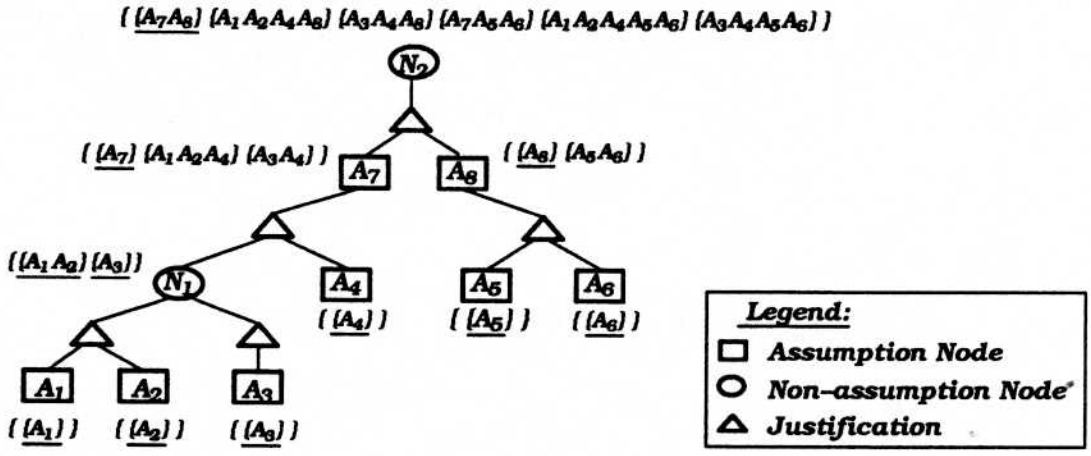

Figure 1: Example justification structure

Beside each node are the labels that a standard ATMS would compute for them. The (smaller) labels that CATMS would compute for them are indicated by the underlined environments.

A CATMS label is always a subset of the corresponding standard ATMS label. The *compression* of CATMS labels accounts for the "C" in CATMS. Label compression can occur only when there are implied assumptions for environments — that is, when some assumption node is the recipient of a justification.[1] de Kleer has argued

[1] We use the terms "justification" and "clause" somewhat interchangeably; CATMS supports both Horn and general CNF clauses.

[3] that allowing assumptions to be implied makes little intuitive sense.[2] However, in practice, assumption nodes are commonly implied. This occurs in part because it is often difficult for the problem solver to know in advance which nodes it will eventually assume (as noted in [9]). Furthermore, allowing implied assumptions enables our technique for avoiding label explosions (see Section 5).

The cost of CATMS's compressed labels comes at query time. To answer a query of whether a node N is true in a given E, CATMS must first *expand* E to the current *Closure*(E), and then check whether E is c-subsumed by any environment in *Label(N)*. Because labels in CATMS are potentially much smaller, the cost of a query in CATMS may actually be less than in a standard ATMS.

CATMS labels can be significantly smaller than standard ATMS labels, including the label of the contradiction node (which defines the minimal nogoods). Since maintaining consistency is a major expense in an ATMS, CATMS's reduced number of minimal nogoods is an especially significant advantage. As with node status queries, nogood queries require first expanding the query environment to compute its current assumption closure. That closure set is then used to check if any minimal nogood c-subsumes the query environment. This need to expand is the price that CATMS must pay for having smaller labels and fewer minimal nogoods to check against — it usually turns out to be quite a bargain.

4 The Basic CATMS Algorithm

This section describes the fundamental aspects of the CATMS implementation. For simplicity, we focus on how CATMS handles Horn clauses (as *justifications* of the form $N_1 \wedge \ldots \wedge N_n \rightarrow Nc$). However, all of CATMS's advantages are retained when general CNF clauses are used as well — although CATMS's handling of clauses suffers from the same incompleteness suffered by all boolean constraint propagation (BCP) approaches, as discussed in [5].

[2] In fact, our notion of equivalent environments is similar to de Kleer's notion of *characterizing environments* for a context [3]. Apparently he never made anything of such groupings because he felt assumptions should never be implied.

4.1 Label Propagation

CATMS computes compressed labels using the standard ATMS label propagation algorithm, but using CATMS's generalized notion of subsumption based on closures. The most dramatic difference occurs when an assumption node is reached during label updating. In CATMS, propagation terminates at that assumption. The CATMS label of an assumption node A always consists of a single environment E_A containing only A itself. Any other environment that a standard ATMS would include in A's label must necessarily contain A in its assumption closure, and thus be c-subsumed by E_A. All label propagations through non-assumption nodes proceed as in a standard ATMS, but using the CATMS version of subsumption based on closures.

The ability of assumptions to block label propagations can be understood as follows. In a standard ATMS, label environments are propagated "forward" from the assumed propositions. In CATMS, assumed propositions are, in effect, propagated "backward" to the minimal environments in which they are true and indirectly to the c-subsumed environments, via assumption closures. This avoids the need to propagate through assumptions, thereby reducing the potential for label explosion.

The ability in CATMS to propagate selected nodes (i.e., assumptions) backward to environments provides a continuum between the two extremes of caching nodes with environments and caching environments with nodes. In the extreme case where all propositions are assumed, CATMS performs no label updates and every label will contain exactly one environment. In that case, CATMS becomes a mechanism for caching the status of each node with particular environments of interest. In the case where no assumptions are justified, the CATMS labels are identical to those of a standard ATMS.

4.2 Computing Assumption Closures

As noted in Section 3, a CATMS environment must be expanded (i.e., have its current assumption closure computed) before testing whether it is c-subsumed by some other environment. Conceptually, an environment E could be expanded by simply running over all of the justifications. One would start with a set S of nodes, representing the union of $Asns(E)$ and the set of always-true propositions, and would

extend S by the consequent nodes of justifications whose antecedents are all members of S. Such expansion would terminate within time linear in the total size of clauses in C. *Closure*(E) would be the set of assumption nodes in S.

However, the justification structure is inefficient for computing assumption closures because it typically contains a large percentage of nodes which are not assumptions. In practice, we employ that straight-forward *expansion by justifications* only to verify CATMS's performance. A more efficient structure results from collapsing out the non-assumption nodes, leaving only assumptions and equivalent justifications. The actual implemented structure, which we call the *expansion lattice*, is conceptually analogous to this. The expansion lattice contains only those environments which minimally imply one or more assumptions.[3] For instance, in the example of Figure 1, the corresponding expansion lattice would contain environments $\{A_1A_2A_4\}$ and $\{A_3A_4\}$ (both implying A_7) and $\{A_5A_6\}$ (implying A_8). These environments are connected such that a path exists from each single-assumption environment up to every *superset* environment in the expansion lattice, where "superset" is based on standard ATMS subsumption — not c-subsumption. This is the only instance in CATMS where standard subsumption is used instead of c-subsumption.

Consider an assumption node A with incoming justifications. As label propagation reaches A, each environment E_i that a standard ATMS would add to A's label is marked as minimally implying A. E_i is then added to CATMS's *expansion lattice*, if it is not already there.

When a new environment E is added to $\mathcal{E}$, CATMS expands E by searching upward from the single-assumption environments corresponding to *Asns(E)*. The set S starts equal to *Asns(E)*. When an environment E_i satisfying *Asns*$(E_i) \subseteq S$ is encountered for the first time, CATMS adds the minimally implied assumptions cached with E_i to S. Each such assumption actually added to S indicates a new single-assumption environment from which to search. This search terminates, with *Closure*$(E) = S$, as soon as all active search paths are exhausted without picking up any new implied assumptions.

[3] However, Section 4.4.2 describes another use for this expansion lattice.

4.3 Maintaining Assumption Closures

In a standard ATMS, an environment is a static collection of assumptions. In CATMS, an environment's assumption closure can grow as new assumptions or clauses are added.

When label propagation reaches an assumed node A, any E that a standard ATMS would add to *Label(A)* must now include A in *Closure*(E). The same is true for all environments c-subsumed by E. The problem is how to index environments so that those c-subsumed environments can be found and updated.[4] One obvious, but inefficient, approach would be to search through $\mathcal{E}$ for all environments c-subsumed by E. A slight improvement results by indexing environments by *size* (i.e., size of their assumption closures), as environments smaller than E cannot be c-subsumed by it. This scheme is complicated by the need to re-index environments whenever their assumption closures grow.

Further improvement results from maintaining pointers from environments to c-subsumed environments. This involves maintaining a *dynamic lattice* of all relevant environments, having the following properties:

1. A path of pointers exists from E_i to E_j if and only if E_i c-subsumes E_j.

2. No two environments are both directly and indirectly connected.

This lattice is dynamic in that the required pointers up to E change as *Closure*(E) grows.

The dynamic lattice provides a mechanism for continuously maintaining assumption closures of all E_i in $\mathcal{E}$. Each E_i newly added to $\mathcal{E}$ is first expanded and then inserted into the lattice. Once inserted, *Closure*(E_i) is updated by propagating newly implied assumptions up through the lattice. When these propagations cause *Closure*(E_i) to grow, then E_i must be re-expanded and possibly repositioned higher in the lattice. Collisions occur as environments become equivalent. Only one representative of an equivalence class is kept in the lattice, with pointers to the rest of the class.

[4] This problem is identical to finding subsumed environments of a new minimal nogood.

The maintenance of the dynamic lattice is quite complex and costly. While the dynamic lattice was the basis for initial implementations of CATMS, it has been disabled in the current implementation, in favor of the lazy approach described in Section 4.5.

4.4 Maintaining Consistency

New minimal nogoods arise as environments are added to the label of the contradiction node. Consistency maintenance involves marking new minimal nogoods, finding and marking any new non-minimal nogoods in $\mathcal{E}$, removing (marked) nogoods from labels, and checking additions to $\mathcal{E}$ against the minimal nogoods. CATMS's use of closure subsumption complicates the standard methods for finding new non-minimal nogoods and for checking new environments, as shown below.

4.4.1 Finding Non-Minimal Nogoods

The traditional ATMS approach for finding subsumed environments of a new minimal nogood is to cache environments in a table indexed by their size, as the nogood can only subsume environments larger than it. The use of such a table is more complicated in CATMS because the size of an environment grows as its assumption closure grows. Thus, CATMS must move environments up in the table as their assumption closures grow.

If the dynamic lattice is being used to maintain assumption closures, it provides an efficient structure for propagating inconsistency. The new non-minimal nogoods in $\mathcal{E}$ which are c-subsumed by a new nogood are exactly those environments found above that nogood in the lattice. Non-minimal nogoods are removed from the dynamic lattice once they are found.

4.4.2 Checking Consistency

As new environments are added to $\mathcal{E}$, they must be initially checked for consistency. We have explored three different approaches for performing this check in CATMS. The trade-offs of these approaches are still being explored.

Dynamic Lattice By keeping minimal nogoods in the dynamic lattice, detecting whether a new environment is a non-minimal nogood is easy: a non-minimal nogood will always be placed directly above some minimal nogood in the lattice. This is clearly the best choice as long as the dynamic lattice is being maintained.

Nogood Table The standard ATMS approach is to maintain a table of minimal nogoods. An environment is detected as nogood if a c-subsuming nogood can be found in the table. As for the environment table mentioned above, nogoods in the nogood table are indexed by size, as nogoods bigger than the query environment cannot c-subsume it. Thus, when the assumption closure of a nogood grows, CATMS moves that nogood up in the table. Failure to do so will not lead to inconsistencies, as all smaller nogoods will be checked; however, it could cause CATMS to make unnecessary checks of those nogoods.

Bad Assumptions in the Expansion Lattice Another approach does not cache minimal nogoods directly; it instead maintains the set of *bad assumptions* for each environment. Upon finding a new minimal nogood E_n, a consistent subset environment E_s is found by removing (any) one assumption A from E_n. A is added to E_s's list of *minimally bad assumptions*, and E_s is added to the expansion lattice (unless already there). The bad assumptions are then picked up during expansion, in a manner analogous to computing assumption closures. An environment is recognized as nogood whenever some assumption is in both its assumption closure and in its set of bad assumptions. We have found this technique to be particularly useful when using the lazy update approach described next.

4.5 Lazy Updating of Closures

Because of the high cost of continuously maintaining assumption closures via the dynamic lattice, the current implementation of CATMS employs an alternative which updates assumption closures upon demand. This involves temporarily relaxing the minimality and consistency requirements for labels. Non-minimal or inconsistent environments are allowed to remain in a label until that label is examined during a

label propagation or a query.

When assumption closures are not continuously maintained, it is especially impractical to attempt to fully maintain consistency. That would require updating the assumption closures of all environments in $\mathcal{E}$ each time a new minimal nogood is found, to see if any are c-subsumed by that nogood and are thus new non-minimal nogoods. Therefore, CATMS takes a lazy approach for detecting nogoods, whereby non-minimal nogoods are not identified until their consistency is required during a label propagation or a query. Consistency checking is performed on such environments using either of the last two techniques of Section 4.4.2. This lazy approach results in slightly larger CATMS labels, though typically still much smaller than the corresponding labels of a standard ATMS.

CATMS also employs some time-stamp efficiencies which allow it to avoid a significant percentage of lazy updates and consistency checks. For example, it avoids updating *Closure*(E) when no new assumptions have been minimally implied since the last time E was updated. Similarly, it avoids consistency checks for E whenever no new minimal nogood has been found since the last consistency check for E.

5 Avoiding Label Explosions

CATMS can avoid the problem of label explosion by reducing the label of any node N to size-one, by assuming N. The resulting *Label(N)* contains just one environment representing the single assumption N. All of the environments previously in *Label(N)* are added to the expansion lattice as minimally implying the assumed N, as explained in Section 4.2. For example, if a large *Label(N)* is about to participate in a costly cross-product operation, then assuming N can make the operation tractable. We call this technique *auto-assuming* a node, and distinguish such assumptions from ones made by the problem solver. Even though minimality and consistency can make the final result of a cross-product significantly smaller than the worst-case, tractability may demand avoiding the cost of even performing the cross-product.

The complexity of CATMS's expansion lattice grows at worst quadratic in the total (precompressed) size of assumption node labels. Thus, auto-assuming to avoid ex-

ponential explosions in any label eliminates the potential for exponential explosions inherent in a standard ATMS. The trade-off, of course, is that fewer inferences are compiled into the labels, requiring some of them to be made repeatedly each time a new environment is expanded. CATMS reduces the problem of label explosion to one of deciding the best time for auto-assuming versus computing a standard ATMS cross-product.

The worst-case complexity of assuming all nodes in CATMS is linear in the total size of the clauses C, due to the limited complexity of expansion by justifications. Therefore, as label growth exceeds some linear function of problem complexity, the performance of CATMS using auto-assumptions becomes increasingly superior to that of a standard ATMS.

We are experimenting with an assortment of criteria for when to auto-assume, based on the particular N, A, C and E of a given problem. For now, we have found a simple constant threshold function that avoids performing cross-products of size greater than about 50 to be sufficient for performance superior to a standard ATMS for many problems. More sophisticated criteria could be based on some limited look-ahead into the justification structure defined by C, to avoid making overly-conservative auto-assumptions.

6 Efficiency Issues

The actual CATMS implementation allows many more trade-offs and efficiencies than have been discussed in the previous sections. For example, label propagation need not stop at all justified assumptions — such blockings can be limited to just auto-assumptions. Blocking label propagations only at auto-assumptions allows the maximal label compilation which still avoids label explosions. However, we have found that CATMS's efficient techniques for expanding environments often make blocking label propagation at all assumptions perform better.

Resolving that trade-off, as with most CATMS trade-offs, depends on the nature of the queries. When most queries involve a small set of environments and occur after most justifications have been asserted, the sum cost of environment expansion will typically be quite low, due to the efficient use of time-stamps discussed in Sec-

tion 4.5. However, as the number of unique query environments increases, the sum cost of all environment expansions increases. There will usually be some number of unique query environments for which standard ATMS label compilation would lead to overall performance superior to using some compressed labels. However, our experience indicates that for many problems that threshold is much higher than the number of unique query environments that are considered in practice.

One typical way in which a large number of unique query environments may be considered is during ATMS *interpretation construction.* Interpretation construction involves finding the maximal set of consistent *interpretation environments* (i.e., *interpretations*), each consisting of one assumption from each given choice set. For example, for choice sets $((A_1,A_2),(A_3,A_4,A_5,A_6),(A_7))$ and minimal nogoods $\{A_1\}$, $\{A_2A_5\}$, and $\{A_2A_4A_7\}$, the consistent interpretations are: $\{A_2A_3A_7\}$ and $\{A_2A_6A_7\}$. A standard technique for efficient interpretation construction is to extend an interpretation of the first $K-1$ choice sets by an assumption of the Kth choice set. Each resulting environment is itself extended unless it is inconsistent or it contains an assumption from each choice set. Thus, for N choice sets, all of size M, interpretation construction may need to consider as many as $O(M^N)$ unique query environments.

CATMS reduces the cost of expanding such query environments during interpretation construction by providing an efficient way of expanding a new environment E which is known to be a union of two already-expanded environments E_i and E_j. This involves starting with set $S = \mathit{Closure}(E_i) \cup \mathit{Closure}(E_j)$ for the expansion lattice search described in Section 4.2. Furthermore, that search starts at only the single-assumption environments corresponding to the assumptions of either S - $\mathit{Closure}(E_i)$ or S - $\mathit{Closure}(E_j)$, whichever is smaller. In the case of expanding a query environment generated during interpretation construction, that means that typically the only seed environment of the search will be the one corresponding to the assumption used to extend the previous partial interpretation.

To reduce the cost of expanding an environment E not known to be union of two expanded environments, CATMS uses a technique that we call *leg search.* For each environment in the expansion lattice, one of its links down to a subsuming environment is designated to be its *leg.* In this case, the set S of the expansion

lattice search described in Section 4.2 is considered to be the current *Closure*(E) and the seeds are the single-assumption environments corresponding to the assumptions of S. Initially, CATMS considers only those environments which can be reached via the leg links alone. If S grows during that first pass, then search continues in the normal way by considering the non-leg links as well.

To further reduce the cost of expansions, we are experimenting with several other trade-offs. For example, one can introduce *phantom environments* in the expansion lattice which do not minimally imply any assumptions but which do increase the hierarchy of the lattice. Using such phantoms can reduce the number of subsumption tests required to realize when many environments above the current one in the lattice are not subsumed by the current assumption closure. In the extreme, such phantoms can be used to maintain the lattice equivalent of a full discrimination tree.

7 Examples

Because our CATMS implementation provides all the functionality of a standard ATMS, problem solvers can use it in place of de Kleer's ATMS program. To evaluate the utility of CATMS's use of compressed labels, we have run a variety of benchmarks comparing CATMS versus ATMS on real, complex problems. In particular, we have extensively tested CATMS with the Qualitative Process Engine (QPE) [9], which was originally designed to use ATMS. The results vary with the specific QPE domain and scenario being modelled. For small examples the differences are minor, but as the size and complexity of the examples increases, the advantages of CATMS become clear.

Table 1 compares the results of running QPE on two different scenarios: the familiar two container liquid flow and a linkage example from the domain of mechanisms. In fact, this latter example provided the original inspiration for developing CATMS. Although CATMS's superior performance in these examples is sufficient to suggest the leverage it can provide, these results are somewhat misleading because CATMS does not yet employ many of the standard ATMS efficiency hacks that ATMS does. We expect that using de Kleer's extremely fast (but complex) bit-vector, hash table,

	QPE Example			
	Two Containers		Mechanical Linkage	
	CATMS	ATMS	CATMS	ATMS
Nodes	1049	1050	1585	1586
Assumptions	18	16	26	24
Environments	135	188	810	4270
in labels	58	50	239	1130
nogoods	52	85	198	1514
minimal	36	66	90	402
Avg. label size	1.15	1.73	1.46	9.91
QPE time (secs)	33.5	36.4	244.4	479.9

Table 1: Example QPE results using CATMS versus ATMS

and label weave operations could help significantly.

These examples demonstrate the following (typical) results: 1) average CATMS label sizes are much smaller, 2) CATMS considers fewer unique environments and nogoods, and 3) CATMS may take slightly longer at query time but the speedup at justification time leads to overall better performance.

8 Conclusions

CATMS provides an efficient hybrid of the compiled approach taken by a traditional ATMS and the interpreted approach noted in [16]. It does so by generalizing the standard ATMS notion of subsumption to allow label propagation to stop at justified assumptions. To use the resulting compressed labels for queries, CATMS provides a means for computing the closure of assumptions implied by a query environment. Furthermore, CATMS uses its ability to maintain compressed labels to prevent the label explosion problem inherent in a fully compiled ATMS approach. CATMS avoids label explosions by auto-assuming a node if its label is too large to allow a tractable cross-product during label propagation. Thus, CATMS transforms the problem of label explosion to one of deciding when a node should be auto-assumed.

The label compression of CATMS is similar to the partitioned approach of PATMS [2], in that both allow a single assumption to take the place of an entire label. However, we believe that CATMS provides a more general solution to the problem of label explosions.

The following issues seem especially worthy of future exploration: 1) better criteria for auto-assuming (perhaps based on limited look-ahead through the justification structure); 2) allowing assumptions to selectively block some environments while propagating others, during label propagation; 3) when using the dynamic lattice (based on c-subsumption) is worth its cost to maintain (using an explicit lattice based on standard ATMS subsumption is discussed in [4,3]); and 4) the impact of CATMS on standard ATMS techniques for focusing [10], backtracking [8], parallelizing [17,6], and others [14,11,13].

Finally, some problem solvers (such as GDE [7]) actually require, for their own use, the standard ATMS labels. To support such reasoning, CATMS would have to either *explode* a compressed label into its standard ATMS form, or provide a means of protecting particular labels from ever becoming compressed.[5] Of course, for such tasks, as well as when the number of queries is exponential or ATMS interpretation construction is performed, CATMS cannot prevent overall exponential complexity. What it can do, however, is perform enough label compilation so that any tractable reasoning over multiple contexts can be performed efficiently.

9 Acknowledgements

Thanks to Ken Forbus and Gordon Skorstad for useful comments. This research has been supported by NASA Langley, under contract NASA-NAG-11023.

[5] However, one of the authors is investigating the potential of CATMS to allow hierarchical reasoning within a GDE-like diagnostic system.

References

[1] John Collins and Dennis DeCoste. CATMS: an ATMS which avoids label explosions. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, July 1991. To appear.

[2] Bruce D'Ambrosio and James Edwards. A partitioned ATMS. In *Proceedings of the Seventh IEEE Conference on AI Applications*, pages 330–336, 1991.

[3] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, March 1986.

[4] Johan de Kleer. Choices without backtracking. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 79–85, August 1984.

[5] Johan de Kleer. Exploiting locality in a TMS. In *Proceedings of the Nineth National Conference on Artificial Intelligence*, pages 264–271, July 1990.

[6] Johan de Kleer. Massively parallel assumption-based truth maintenance. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 199–204, August 1988.

[7] Johan de Kleer and Brian Williams. Reasoning about multiple faults. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 132–139, August 1986.

[8] Johan de Kleer and Brian C. Williams. Back to backtracking: controlling the ATMS. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 910–917, August 1986.

[9] Kenneth D. Forbus. The qualitative process engine. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 220–235, Morgan Kaufmann, 1990. (Technical Report UIUCDCS-R-86-1288, University of Illinois at Urbana-Champaign, December 1986).

[10] Kenneth D. Forbus and Johan de Kleer. Focusing the ATMS. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 193–198, August 1988.

[11] Matthew L. Ginsberg. A circumscriptive theorem prover: preliminary report. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 470–474, August 1988.

[12] Rowland R. Johnson et al. Interpeting signals with an assumption-based truth maintenance system. In *SPIE Vol. 786 Applications of Artificial Intelligence V*, pages 332–337, 1987.

[13] Ulrich Junker. A correct non-monotonic ATMS. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1049–1054, August 1989.

[14] Gregory M. Provan. An analysis of ATMS-based techniques for computing Demster-Shafer belief functions. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1115–1120, August 1989.

[15] Gregory M. Provan. Efficiency analysis of multiple-context TMSs in scene representation. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 173–177, August 1987.

[16] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: preliminary report. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 183–188, July 1987.

[17] Edward Rothberg and Anoop Gupta. Experiences implementing a parallel ATMS on a shared-memory multiprocessor. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 199–205, August 1989.