# Putting the Problem Solver Back in the Driver's Seat: Contextual Control of the ATMS


Oskar Dressler
SIEMENS AG
Otto-Hahn-Ring 6
D-8000 Munich 83
Germany
dressler@zfe.siemens.de

and

Adam Farquhar
Department of Computer Sciences
University of Texas
Austin, Texas 78712, USA
farquhar@cs.utexas.edu



**AREA:** Truth Maintenance



### Abstract

The ATMS is a powerful tool for automated problem solvers and has been used to support several model-based reasoning tasks such as prediction and diagnosis. It provides an efficient mechanism for maintaining consistent sets of beliefs and recording the assumptions underlying them. This enables the problem solver to switch rapidly between contexts and compare them. Such capabilities are central to diagnostic systems, and are also valuable to design and planning systems. Applications to larger problems have been hampered, however, by the problem solver's inability to maintain control over the ATMS.

We present a new approach, implemented in a system called COCO, which allows the problem solver to maintain tight control over the contexts explored by the ATMS. COCO provides means for expressing local and global control over both normal and nogood consumers. Local control is achieved by attaching guards to individual consumers. These guards express control, rather than logical, knowledge and consist of sets of environments. Global control is achieved by specifying a set of *interesting* environments. Consumers are fired only when its antecedents are true in some interesting environment. We also successfully apply the same technique to *limit label propagation* in the ATMS. This ensures that the ATMS respects the problem solver's wishes and only makes derivations in interesting contexts.

We demonstrate the both the dramatic increases in efficiency which are made possible by these techniques, as well as their tremendous expressive power, in four examples.

# 1 Introduction

The ATMS is a powerful tool for automated problem solvers. It provides an efficient mechanism for maintaining consistent sets of beliefs and recording the assumptions underlying them. This enables the problem solver to switch rapidly between contexts and compare them.

The ATMS, however, has two shortcomings: (1) the existing problem solver–ATMS interface is hard to control, and (2) the ATMS attempts to compute all solutions, even when they are irrelevant or unnecessary. Thus, in problems with large, perhaps infinite, search spaces new techniques must be used to control the ATMS.

We define an expressive, flexible, and efficient problem solver–ATMS interface, called COCO (COntext driven COntrol), to address these problems. Using COCO, the problem solver defines a focus — sets of environments which it finds interesting — which is used to control rule execution, and to restrict the contexts in which the ATMS looks for solutions. COCO ensures that the ATMS respects the problem solver's wishes. This means not only that no rule will be executed unless its antecedents are true in some interesting environment, but that only the interesting environments are propagated. In a sense, our approach makes a philosophical break with the previous ATMS research which has emphasized exhaustivity: COCO ensures completeness *only* with respect to the focus.

COCO has seen extensive use in the GDE$^+$ diagnostic system [15]. The basic ideas have also been adapted for use in the SHERLOCK diagnostic system [6]. In Section 5, we present four additional examples of its use along with empirical results which demonstrate both the dramatic increases in efficiency which are made possible by these techniques, as well as their tremendous expressive power.

# 2 ATMS Background

The ATMS [2] supports problem solvers that are able to explicitly mark out some of the data that they manipulate as *assumptions*. Belief in all other data is then characterized by the assumptions which support them. These sets of assumptions are called *environments*. Each problem solver datum has an ATMS *node* associated with it. Each node has a *label* which is a list of the environments supporting it. The problem solver interacts with the ATMS by making assumptions and by *justifying* one datum in terms of other data. The primary responsibility of the ATMS is to compute the correct label for each datum. A justification with antecedents $x$ and $y$ for the consequent $z$ is written [x y] $\rightarrow$ z.

Derivation is naturally defined using justifications. The consequent of a justification is derivable when the antecedents are either assumptions or derivable nodes. Inconsistencies arise when a specific node, $\perp$, is derived. The environments that derive $\perp$ are called *nogoods*. Since derivation is monotonic, the ATMS only needs to represent minimal environments. This applies both to the environments that derive a specific node (its label) and the nogoods. The minimal environment characterized by assumptions $a$ $b$ $c$ is written $\langle a\ b\ c\rangle$. The set of nodes derivable from a consistent (non-nogood) environment is called a *context*. *Context(E)* denotes the context characterized by the environment $E$.

Following de Kleer, we are considering problem solvers which are capable of expressing most of their knowledge in the form of rules. The consumer [2] is a device for linking rules to the ATMS nodes. A consumer consists of a set of antecedents which are ATMS nodes (or classes) and a body of code which computes some result given the problem solver data corresponding to the antecedents. Once the antecedents are established, the consumer may be selected and be *fired*; it asserts some consequents which are justified by the antecedents. If the ATMS is to support sound deductions, the consumers must obey certain conventions: they must not use any data not present in the antecedents and they must include all of the antecedents in a justification they add. The consequence of these restrictions is that a consumer need only be executed once for a given set of antecedents. Firing a consumer, therefore, may be viewed as compiling it into a set of justifications. A consumer for antecedents $x$ and $y$ is written $(x\ y \Rightarrow)$, or with a body, $(x\ y \Rightarrow [z := x + y])$. The latter indicates that the consumer compiles into a justification such as [x=2, y=3] $\rightarrow$ z=5.

# 3 The Problem of Control

Two major steps towards controlling the ATMS have been reported in [4] and [10]. This research attempts to control the ATMS by restricting the number of consumers which are executed. Unfortunately, in the ATMS, delaying the execution of a consumer is only half the job – it is like keeping the floodgate closed as long as possible. As soon as the consumer fires, it introduces a justification, and all of the environments in the labels of its antecedents flood through the justification, and flow throughout the justification network. Executing a consumer is a simple constant time operation, propagating environments, however, is worst-case exponential.

Therefore, COCO has two main techniques for expressing control: a consumer focus, which restricts the execution of consumers, and an environment focus, which restricts the propagation of environments.

# 4 Controlling Consumer Execution

## 4.1 Context and Data Driven Strategies

Standard rule–based problem solvers maintain a single database of facts which are referred to in the preconditions of rules. Each fact in the database is believed and the rules are fired when their antecedents are in the database. In TMS–based problem solvers, these facts are labeled as in or out depending on their support. Rules may fire because certain data are in or out, or because a datum has changed its label. The ATMS is more expressive. Data are not simply in or out, but belief in them is characterized by sets of environments. A datum may be thought of as being in any context that is characterized by one of these environments. Thus, rules used in conjunction with an ATMS must fire on a combination of *data and environments*. Data and environments can be viewed as two orthogonal axes.

All existing strategies for controlling rule execution in the ATMS can be located somewhere on these two axes. There have been a variety of strategies discussed in the literature. In [10], Forbus and de Kleer review :intern, :in, and :addb, and propose two new ones: :implied-by and :contradiction.

:Intern and :contradiction represent the two extremes of data and context driven strategies. :Intern is purely data driven. A consumer fires as soon as its antecedents are in the ATMS database, regardless of their labels. They need not even be members of the same context. :Contradiction is purely context driven. A :contradiction consumer is associated with a single environment. When that environment becomes a nogood, the consumer fires. This is a simple way for the ATMS to signal inconsistencies to the problem solver. There are no data associated with the contradiction consumer at all. The standard consumer execution strategy is :in, which requires that the antecedents are in at least one common context. Thus, it is mostly data driven, but also puts a weak constraint on when it is interesting to fire the rule.

Both :addb and :implied-by make use of a *control environment*. The control environment is specified by the problem solver, and is used to define the current problem solver task. It is a set of assumptions whose consequents the problem solver is currently interested in.

:Addb (assumption-based dependency directed backtracking) puts more constraints on the contextual part of a consumer's precondition. An :addb rule will be executed only when there is at least one environment $E$ where all of its antecedents hold such that (1) the set of control assumptions in $E$ is a subset of the actual control environment, and (2) the union of the actual control environment and $E$ is consistent. In :addb the problem solver delegates part of its responsibility for control to the ATMS. A fixed scheme is used to aggregate the control environment from so-called control disjunctions that are known in advance.

In the :implied-by strategy, the control environment is used as an upper bound. Whenever the antecedents hold in a subset of the control environment the consumer is fired. Using :implied-by, the problem solver is free to install and retract control environments in any order it wishes.

Another approach to controlling consumer execution is to provide control antecedent nodes. At some point the problem solver justifies the control node with a control assumption C, and the consumer may then fire, introducing a justification (see

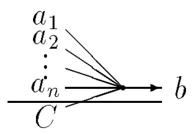

Figure 1: Using antecedents for controlling the ATMS is inadequate.

Figure 1). If the consumer ascribes to the convention of using all of its antecedents in the justifications it introduces, then the control assumptions will be included in the consequent's label. Thus, then we have achieved a limited amount of control at the price of doing more work after the control condition is satisfied. Labels and environments will become bloated with control assumptions. If control nodes are true in different contexts, their consequents will also become discriminated into many contexts which differ only in their control decisions. There is no way to control the propagation of environments after the consumer has fired and installed a justification, nor is there a way to retract a control decision. One final difficulty with this representation is that the control knowledge and decisions are mixed in with the domain knowledge.

## 4.2 Control of Consumer Execution in COCO

When using the ATMS, people carefully select the data that will be used as assumptions from the data their problem solvers are going to manipulate. In doing so they implicitly define a set of possible contexts. Therefore, a context is normally associated with an application dependent meaning. For instance, in GDE, a system for model-based diagnosis [5], a context is associated with a set of components that are working correctly. Another application views a context as characterizing a specific perceptual interpretation [13]. [12] uses contexts to represent situations for a planning task. In [14] the framework for assumption-based reasoning as provided by the ATMSis used to represent various structural and behavioral aspects of components in a device. During analysis or diagnosis of the device one wants to focus the problem solver on certain aspects of specific sets of components. These are represented as assumption sets, i.e. contexts. Struß in [14] explicitly expresses the need for a means to convey information about these interesting contexts to the ATMS. In Section 6.2 we discuss an example in which contexts are associated with hierarchical layers in a component model. The same technique can be used to encode aspects of components as needed in [14].

¿From the preceding discussion, it is clear that a consumer strategy consists of two parts: a specification of which *nodes* it should fire *on*, and a specification of which *environments* it should fire *in*. A third dimension, that is present in none of the strategies above, is spanned by an axis that specifies whether the control information is available locally or globally. Local control information is attached to individual consumers, whereas global control information applies to all consumers. In COCO, we express all of this information explicitly. As usual, the data are specified by a rule's antecedents. Control information is specified by a local *guard* and a global *focus*. Conceptually, both the guard and the focus are simply sets of environments. These sets of environments can most often be conveniently described by upper and lower bound in the lattice of all environments. For instance, $focus = \{E | \{a, b\} \subseteq E \subseteq \{a, b, c, d, e\}\}$ describes the set of environments above {a,b} and below {a,b,c,d,e}.

The *guarded consumer* may fire only if its antecedents hold in some common environment that is a member of the *guard set*. The guarded consumers allow the problem solver to maintain tight *local* control over consumer execution by restricting the contexts in which an individual consumer fires.

The *global focus* affects all consumers equally. A consumer may fire only if its antecedents hold in a common environment that is a member of the global focus. The :implied-by strategy presented in [10] is a restricted version of the global focus in which there is only a single control environment. It uses focus descriptions of the form:

$$focus = \{E | E \subseteq \textit{control-environment}\}$$

There are several examples of focusing using an upper bound in the literature, such as [4] and [10]; we provide one more in Section 6.2.

Although very useful, e.g. reasoning about hypothetical situations, using a lower bound to specify a focus environment leads to a minor complication. The difficulty is that the environments in labels are minimal ones. If a consumer's antecedents hold in the common minimal environment, ⟨a⟩, then they hold in ⟨a b⟩, ⟨abc⟩, and so on. Therefore, even if we specify a lower bound focus, say ⟨ab⟩, there is a non-null intersection between the focus and the minimal environments "below" it — ⟨a⟩ and ⟨b⟩ in this case.

By default, we restrict consumers to apply to the minimal environments where all of their antecedents hold. Thus, in our example a consumer sitting on the environment ⟨a⟩ is not fired on ⟨a b⟩.

## 4.3 Nogood consumers

By specifying the global focus and guards for the consumers, the problem solver can keep tight control of the problem solving activities as far as consistent environments are concerned. A major part of problem solving, however, is finding out about inconsistencies. When nogoods are discovered, the problem solver must be informed so that it can react appropriately. For this purpose we use *nogood consumers*, a special kind of guarded consumers that is executed on the minimal nogoods contained in the attached guard set. Nogood consumers are prioritized. They are scheduled before all other consumers because the problem solver must immediately be informed about changes of the search space as to avoid useless activities. Often the inconsistency of the focus (or parts of it) will be signaled this way. The "contradiction consumer" from [10] is a specialized nogood consumer with an attached guard set of cardinality one.

# 5 Focusing Label Propagation

Regardless of how sophisticated our mechanisms for controlling consumer execution are, control will be lost after a consumer fired. Label propagation inside the ATMS still occurs throughout the whole network and for all contexts. It is possible, however, to use the techniques described above to control the propagation of environments through the justifications. A guard set similar to the one for consumers allows a justification to propagate environments only when they fall within the specified range. Justifications that have an attached guard set are called *guarded justifications*. Besides guards for local control information we can also use the global focus to exercise control of label propagation globally.

## 5.1 Local Control of Label Propagation

If a consumer is controlled by a guard, it will under no circumstances be executed on environments outside of the guard set. However, if it is executed, it will, following the consumer architecture [2], compile into a justification. The advantages of a guarded consumer would be lost once the justification is given to the ATMS, as *all* of the environments in the label of its antecedents would be propagated. Label updating does not respect the guard set. For example, suppose there is a guarded consumer ($x$,$y$ $\Rightarrow$) with the guard $\{E|\{a, b, c\} \subseteq E\}$. Initially, let label($x$) be $\{\langle \mathrm{d} \rangle\ \langle \mathrm{e} \rangle \ldots\}$ and label($y$) be $\{\langle \mathrm{e} \rangle\ \langle \mathrm{f} \rangle \ldots\}$. The consumer does not fire because the label of its antecedents, label($x, y$), is $\{\langle \mathrm{e} \rangle\ \langle \mathrm{d\ f} \rangle \ldots\}$, which does not satisfy the guard. If the environment $\langle \mathrm{a\ b\ c\ d} \rangle$ is subsequently added to label($x$) and label($y$), the label($x, y$) becomes $\{\langle \mathrm{a\ b\ c\ d} \rangle\ \langle \mathrm{e} \rangle\ \langle \mathrm{d\ f} \rangle \ldots\}$, which satisfies the guard. The consumer fires, installing the justification $[x,\ y] \rightarrow z$. Unfortunately, the label($z$) now contains the complete label($x, y$), including the *un*interesting environments which did not fire the consumer to start with. Even worse, these uninteresting environments will be propagated through the label of every node which $z$ is connected to.

We would like the consumer's guard set to be respected by the ATMS and used to control label propagation. If the principles of the consumer architecture are strictly followed, justifications are nothing but instances of rules. Therefore, we can use the consumer's guard set to control label propagation: a justification will propagate when the guard set allows it to do so. Although the label of the conjunction of antecedents is correctly computed (and recorded in the justification's label), only the minimal environments that represent the lower bound of the intersection of the environments (implicitly) represented by the justification's label and the guard set are propagated to the justification's consequent. Thus, using our previous example, label($z$) becomes $\{\langle$a b c d$\rangle\}$, and only the interesting environments are propagated to $z$'s children.

It is crucial to note that the guard set must only be used for control. It must not be misused for specifying logical dependencies. The guard set must not be used to restrict the validity of a justification, just its interestingness. It should always be the case that if the guard were removed, a program would still be logically correct, though perhaps infinitely inefficient.

## 5.2 Global Control of Label Propagation

It is worth considering whether the technique of delaying propagation from justification labels can be applied to the focus, too. Can label propagation for *all* justifications be limited by the global focus? The answer is yes. First, we can limit label propagation by the upper bound of the focus. Any environment above the upper bound that appears in some antecedent will result in an environment for the consequent that is at least as large. The resulting environment will thus be out of the focus. If it becomes inconsistent the focus will not be affected. Second, we use the lower bound of the focus to decide which environments we propagate from a node. For example, if a node's label is $\{\langle$a$\rangle$ $\langle$b$\rangle\}$ and the lower bound is given by $\{\{$c$\}$ $\{$b d$\}\}$ then we propagate the environments $\langle$a c$\rangle$, $\langle$b c$\rangle$ and $\langle$b d$\rangle$. The lower bound has the same effect as adding a virtual node to the antecedents of a justification. Now, if e.g. $\langle$a$\rangle$ without restricting label propagation would become a nogood, then we will detect the nogood $\langle$a c$\rangle$ when applying the lower bound. This is exactly what we want: we detect those nogoods that effect the focus.

By limiting label propagation we loose label completeness as far as the whole context lattice is concerned. We retain soundness and minimality. We give up label completeness and consistency only for those contexts that are out of the focus, as is desired. But label completeness and consistency is guaranteed for the focus.

## 5.3 Changing the focus

It is natural that the problem solver will want to modify its focus as the current task changes. This is supported in a straight forward and efficient way. If the new focus is more restrictive than the old one, then no additional work need be done. E.g. suppose that *label*($x$) $=\{\langle$a b c$\rangle$, $\langle$d$\rangle\}$, and there is no focus — all environments are propagated without control. If the focus is now changed to be $\{E|E \supseteq \{$a d$\}\}$, we can leave the label of $x$ as it is. It is still correct, but some part is just uninteresting. Any new propagation from $x$ will be done using the interesting part of the label,

$\{\langle \text{a d} \rangle\}$.

If the focus is relaxed, then we need to propagate the environments which were outside of the old focus, but are within the new one. The overhead is quite small, linear in the number of environments which were not wholly within the old focus. In the worst case of focus change, every environment is initially outside of the focus and after the change is within the new focus. All of the environments must then be propagated, which is just what the ATMS would have done without the focus. Thus, relaxing the focus just forces us to do some of the work which we avoided by using the focus to start with.

The only complication involves deciding when the environments should be propagated, and how to order their propagation. Propagation starts from a node, but a node's label might contain several effected environments. Thus, it is clear that we want to check all of the environments first, so that we do not have to propagate from a node once for each of its effected environments. The second issue is choosing an order in which to propagate from the effected nodes. In order to do a minimum amount of work, we can compute an ordered set of strongly connected components starting with the assumptions and compute new node labels in this order. This is an adaptation of the technique from [11] used there for a justification-based truth maintenance system. A standard algorithm is $O(max(|nodes|, |justifications|))$, so this requires little additional work, and is well worth the effort.

# 6 Empirical Results

In this section we provide four examples to illustrate scope of application, conceptual clarity, and pragmatic efficiency gains which can be achieved with COCO. First, we show its effectiveness in *controlling* the multiple model prediction problem. Second, we show how it can be used to suppress uninteresting consumer execution that arises in constraint satisfaction. Third, we show how it can be used to implement flexible and tightly controlled hierarchical modeling. All three of these methods are in daily use as parts of the GDE$^+$ diagnostic system. The first and third examples make use of the global focus to control both consumer execution and label propagation. The second example highlights the use of guarded consumers and guarded justifications. Fourth, we show an implementation of GDE using a nogood consumer.

## 6.1 Model-based prediction

We look at a model-based predictive problem in which we have a device composed of several components. Each component is described by an OK model describing its behavior when it functions correctly, and several fault models. The task is to compute predictions for the various parameters of the device given these models and observed values for some of the parameters. A combinatoric problem results because combinations of the predictions from different models need to be considered. With the standard ATMS, all (minimal) combinations of all of the models will be computed. This will include what the problem solver might consider to be extremely unlikely combinations involving many fault models. If the problem solver is willing to focus on an interesting subset of the possible combinations, however, COCO provides a means of dramatically reducing the work done by the ATMS.

In the example, we will use an $N$*-fault* focus which states that a derivation requiring more than $N$ fault models is uninteresting. This is less restrictive than what most authors mean by an $N$*-fault assumption.* The $N$-fault assumption is a global statement about the whole device – no more than $N$ of the components are faulty. Our $N$-fault focus says that no more than $N$ fault models should be used to make any single derivation. Thus, independent failures, each of which require no more than $N$ fault models, will be predicted. This definition fits well with the use of local propagation techniques to determine predicted values.

Two key advantages of using this sort of focus are (1) that derivations made within the focus are sound — they will not be contradicted by a change in the focus; and (2) that the focus can be easily changed by the problem solver and any new interesting deductions will be efficiently computed by COCO. This allows the problem solver to start with a very tight focus, and then gradually relax it until a satisfactory solution is found.

We have used the standard example with 3 multipliers and 2 adders (Figure 2). Each component is described by seven models:

1. OK. The component functions as expected.
2. Unknown. The component may exhibit arbitrary behavior.
3. Zero. The output is 0.
4. One. The output is 1.
5. Left. The component outputs its "left" input.
6. Right. The component outputs its "right" input.
7. Left-Shift. The output is shifted left one bit.

The input values A=3, B=2, C=2, D=3, and E=3 are asserted in succession. ¿From this information, the OK models would predict a value of 12 for both F and G. We then assert the observations F=2 and G=8. These observations cannot be explained by a single fault, but two independent faults are adequate. A reasonable diagnosis might be {zero(M1), left(M2)} or {left(M2), right(A1)}.

Table 1 summarizes the results of using a single-fault focus. There is a dramatic 40-fold speedup in execution time when using the focus! There are several interrelated reasons for this tremendous reduction. Many rules are not fired because there antecedents do not hold in any *interesting* environments. If these rules had fired, they would have introduced justifications, sometimes for previously unpredicted values. These new values then enable additional rule firings, and the effect cascades.

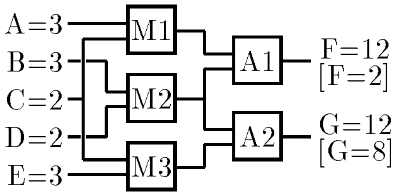


Figure 2: A standard diagnostic problem.

| | | Rules | | | Labels | | Nogoods | | |
|---|---|---|---|---|---|---|---|---|---|
| Experiment | Time | fired | Justs | Nodes | Max | Avg | Ct | Max | Avg |
| Focused | 40 | 359 | 1875 | 128 | 28 | 4 | 203 | 10 | 4 |
| Unfocused | 1600 | 1096 | 4233 | 214 | 186 | 32 | 409 | 10 | 5 |
| Improvement: | 40* | 3.05* | 2.26* | 1.7* | | | | | |

Table 1: A single-fault focus provides a 40-fold speedup when using seven models per component. The Labels column gives the maximum and average number of environments in a label; Nogoods gives the count of the minimal nogoods, as well as the maximum and average number of assumptions in a nogood.

| | | Rules | | | Labels | | Nogoods | | |
|---|---|---|---|---|---|---|---|---|---|
| | Time | fired | Justs | Nodes | Max | Avg | Cnt | Max | Avg |
| Focused | 1 | 424 | 666 | 217 | 9 | 1.2 | 36 | 4 | 1.7 |
| Unfocused | 43.5 | 4912 | 5298 | 2353 | 19 | 2.3 | 197 | 427 | 415.4 |
| Improvement: | 43.5* | 11.6* | 7.9* | 10.8* | | | | | |

Table 2: Using hierarchical models, two faults in a circuit of 845 gates provides a 43-fold speedup by limiting prediction to the relevant parts of the circuit.

Furthermore, the effect of adding a single justification may result in large amounts of work, as the new environments in the label of its consequent must be propagated throughout the justification network.
It is worth noting that using a consumer focus without the environment focus results in almost no improvement in this example. For a more complete discussion see [8].

### 6.2 Implementing Hierarchical Models

Another interesting application of global focusing is the use of hierarchies in model-based diagnosis. Consider again the device in Figure 1 (adder multiplier), but let each of the components be hierarchically defined in by subcomponents. Let the adder be a ripple-carry adder composed of half– and full-adders. The full-adders are composed of half-adders and gates; the half-adders are composed of gates. This model consists of four levels. Further, suppose that a diagnostic engine has concluded failure in the components M1 and A1 would account for the observations. In order to further localize the malfunction, the diagnostic engine needs to focus on the subcomponents of M1 and A1. Of course, it wants to do this refinement of the model without executing consumers which are associated with components other than M1 and A1 or which are associated with other levels of these two components.
We can easily implement this sort of control by associating an assumption with each component level, e.g. A1-gate-level, M1-adder-level. Let $L$ be the set of these component-level assumptions. Every consumer that implements a predictive piece of knowledge for this component-level has the associated assumption as an additional antecedent. When we want to focus on a specific set of component-levels, *focused-levels*, use the following focus $\{E \mid (E \cap L) \subset \textit{focused-levels}\}$. E.g. if we want to focus on the gate level M1 and the adder level of A1 we would use the focus: $\{E \mid (E \cap L) \subset \{\textit{a1-adder-level m1-gate-level}\}\}$.
This focus guarantees that consumers are only fired when their antecedents hold in an environment that depends only on the focused component-levels. Table 2 presents the empirical results. Using hierarchical models, two faults on the gate-level, a faulted AND and OR gate, are identified requiring the resources reported in the first line. Without focusing, the entire device must be simulated on the gate level which consumes the resources reported in the second line, which does not include the time for the diagnosis algorithm, but summarizes only prediction.

## 7 Related Work and Applications

In [7] Collins and DeCoste report a maximum speedup factor of 2 resulting from the use of their CATMS on the prediction task within QPE [9]. They achieve this effect by compressing labels of assumptions that are justified by other nodes. If assumptions are not justified, their CATMS behaves like a normal ATMS with the usual blowup of label sizes. In [3] de Kleer reports on the use of techniques that are quite similar to ours and actually build on a predecessor of the current system. While COCO guarantees completeness of labels within the focus, the de Kleer's HTMS gives up this property, and resembles a conventional single-context TMS. Coco has been successfully applied in an industrial application for fault localization in power transmission networks [1]. The network contains roughly 1000 components. Application of the diagnostic system using message bursts recorded during actual transmission network failures has resulted in correct correct diagnoses in under 60 seconds. As part of the model-based diagnosis system [15] COCO allows focusing on those parts of the network where reactions of the transmission network's protection system indicate potential malfunctions.

# 8 Conclusion

We have presented an approach that allows ATMS–based problem solvers to reduce the complexity of their tasks by exploiting additional control knowledge. As the notion of context often has an application dependent meaning, this control knowledge is naturally expressed in terms of contexts. Therefore, we have provided ways for the problem solver to express itself in terms of contexts in addition to the conventional ways in terms of nodes. Coco allows the problem solver to specify a global focus and local guards in order to:

- guide consumer execution,
- limit label propagation.

Furthermore, the problem solver is immediately informed about changes in the search space by *nogood consumers* and can take appropriate action.
This new architecture has proven to be both efficient and expressive. We have presented four applications: controlling feedback in a constraint system, providing hierarchical modeling, controlling multiple model prediction, and implementing GDE. Furthermore, we have presented empirical evidence supporting the claim that the gains are real and dramatic.

# References

[1] A. Beschta, O. Dressler, H. Freitag, M. Montag, and P. Struss. A model-based approach to fault localization in power transmission networks. *Intelligent Systems Engineering*, 1992.

[2] Johan de Kleer. An assumption-based truth maintenance system, Extending the ATMS, Problem solving with the ATMS. *Artificial Intelligence*, 28(2):127–224, 1986.

[3] Johan de Kleer. Focusing on probable diagnoses. In *AAAI-91 Proceedings of the Tenth National Conference on Artificial Intelligence*, 1991.

[4] Johan de Kleer and Brian Williams. Back to backtracking: Controlling the ATMS. In *AAAI-86 Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 910–917, August 1986.

[5] Johan de Kleer and Brian Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987.

[6] Johan de Kleer and Brian Williams. Diagnosis as identifying consistent modes of behavior. In *IJCAI-89 Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[7] Dennis DeCoste and John Collins. Catms: An atms which avoids label explosions. In *AAAI-91 Proceedings of the Tenth National Conference on Artificial Intelligence*, 1991.

[8] Adam Farquhar. Focusing ATMS-based diagnosis and prediction. In *Proceedings of the Model-Based Reasoning Workshop at IJCAI-89*, 1989.

[9] Kenneth Forbus. The qualitative process engine. In J. de Kleer D. Weld, editor, *Readings in Qualitative Reasoning*. Morgan Kaufmann, 1980.

[10] Kenneth D. Forbus and Johan de Kleer. Focusing the ATMS. In *AAAI-88 Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 193–198, August 1988.

[11] James W. Goodwin. An improved algorithm for non-monotonic dependency net update. Technical Report LITH-MAT-R-82-83, Linkoeping University, 1982.

[12] Paul Morris and Robert Nado. Representing actions with an assumption-based truth maintenance system. In *AAAI-86 Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 13–20, 1986.

[13] Gregory Provan. Efficiency analysis of multiple-context TMSs in scene representation. In *AAAI-87 Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 173–177, 1987.

[14] Peter Struß. Multiple representation of structure and function. In *J. Gero (Ed.) Expert Systems in Computer Aided Design*, Amsterdam, 1987.

[15] Peter Struß and Oskar Dressler. Physical negation — integrating fault models into the general diagnostic engine. In *IJCAI-89 Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.