Introducing Data Reduction Techniques into Reason Maintenance

Rüdiger Lunde
University of Applied Sciences Ulm
Prittwitzstrasse 10, 89075 Ulm (Germany)
email: r.lunde@hs-ulm.de

Abstract

Every problem which can be solved with a reason maintenance system can – in theory at least – be solved as well without it, using a simple generate and test algorithm. Therefore, its main purpose is to increase the efficiency of a reasoning system. In this paper, we analyze the impact of problem characteristics on the performance. For problems which include reasoning about physical models on quantitative level, the complete dependency network maintained by reason maintenance systems is identified as a major resource consumer. Based on that analysis a new component called 'value manager' is presented, which applies data reduction techniques to limit dependency management costs and is especially designed to support iterative solvers. As two examples of practical applications, experimental results from an automated FMEA generation run for an automotive system and a diagnosis of a fly-by-wire system are discussed.

1 Introduction

A reason maintenance system (RMS) is a book-keeping tool supporting a problem solver. It tracks dependencies between given and derived data during the inference process and contributes to the search for solutions in two ways. Firstly, it analyzes the causes of failures and thus helps to avoid useless search in subspaces without solutions. Secondly, it caches inferences and prevents the problem solver from redrawing the same inferences again and again.

An RMS views data as propositional symbols and relationships between the data as propositional clauses. This view is independent of the actual meaning of the data for the problem solver, which can be first or higher order, for example. Given sets of assumptions, data, and clauses, the RMS derives the belief status of every datum based on the current belief status of the assumptions. An RMS works incrementally. After the belief in some of the assumptions changes or a new clause is added, the belief status of all affected data is updated without starting from scratch. A contradiction is discovered if the belief status of a special datum denoting falsity changes from disbelieved to believed. Contradictions are immediately signaled to the problem solver, and information about the responsible assumptions is attached to the signal.

The use of reason maintenance systems has a long tradition in AI. Starting with the introduction of the non-monotonic JTMS [Doyle, 1979], a wide range of different reason maintenance systems has been developed (e.g. monotonic JTMS, ATMS, LTMS). They differ in the accepted types of clauses, their support w.r.t. contradiction resolution, and the way the belief status is represented in labels associated with data. The assumption-based truth maintenance system (ATMS) [de Kleer, 1986a][de Kleer, 1986b][de Kleer, 1986c] as part of the general diagnostic engine (GDE) [de Kleer and Williams, 1987] has attracted much attention in the field of model-based reasoning. It accepts Horn clauses and uses a special indexing scheme to store the sets of assumptions a datum ultimately depends on. It supports parallel search in different contexts and is especially suited for problems with many solutions, if all or several of them are of interest. This is usually the case in explanatory problems, where we want to know all or at least the most plausible causes for observed or assumed effects.

It is well known that the worst-case complexity of an ATMS is exponential in the number of assumptions. Reasoning systems utilizing an ATMS spend a considerable amount of time on label updates. Several extensions have been developed to improve the efficiency of the reasoning system as a whole. Forbus and de Kleer [Forbus and de Kleer, 1988] proposed a consumer control mechanism to focus inference selection on interesting contexts. Additionally, Dressler and Farquhar [Dressler and Farquhar, 1991] showed how the label update can be restricted to interesting contexts. Although label completeness is lost, completeness with respect to the current focus can still be guaranteed. Lazy label evaluation [Kelleher and van der Gaag, 1993] delays the label update until the problem solver shows interest in it. By combining focusing and lazy label evaluation, some synergy effects can be obtained [Tatar, 1994].

Reason maintenance systems have proven to be efficient tools to support reasoning about qualitative abstractions of real systems. However, when choosing lower levels of abstraction, two negative impacts on the efficiency of the reasoning system can be observed:

- The dependency management costs grow dramatically due to the increasing absolute number of assumptions and inferences needed for behavior prediction.

- The benefit achieved by the services of a reason maintenance system decreases because of the decreasing degree of similarity between different contexts and the gap between the true meaning of the derived data and the reason maintenance system's propositional view of it.

Both observations give reason to look for more efficient alternatives.

2 Reason Maintenance Systems seen from the Perspective of Machine Learning

From the point of view of machine learning, an RMS plays the role of a learning module. It tries to find out as much as possible about the currently solved problem by collecting and analyzing the inferences drawn so far.

In all reason maintenance systems memorizing plays a major role. All inferences drawn so far are stored together with their results in a dependency network. No attempt is made to generalize the given information or to remove unimportant nodes. The main problem with this learning method is that memory space grows monotonously and the management overhead with it. So there is a tradeoff between the gains resulting from learned information and the overhead to maintain the knowledge base. In large search problems, the efficiency of the overall performance of an RMS based reasoning system typically increases first, then reaches a maximum, and decreases steadily afterwards.

It is obvious that maintaining a cache for all inferences ever performed is inefficient for iterative solvers. To reduce memory consumption for intermediate results, the learning module should abstract from dependencies on inference layer and focus on the relationship between the assumptions defining the interesting contexts and the final results obtained from them. The ATMS already incorporates an efficient method to compile dependency information given on inference layer into an explicit representation of the relationship between assumptions and derived data. For each datum, it maintains a label which represents the set of all currently known consistent combinations of assumptions supporting the derivation of the datum. In contrast to the memorizing method used for low level dependency tracking, the learning method used here incorporates a generalization step. The combinations of supporting assumptions are not enumerated directly but characterized by a lower bound, which is updated after each inference step for all affected data. The label representation can be viewed as a conceptual description of all consistent contexts to which the corresponding datum belongs, and the incremental label update as a special kind of inductive concept learning.

Since the label update algorithms do not necessarily require the availability of a complete dependency network, a new class of dependency trackers can be defined which is based only on the second learning method. In the following, we discuss one special instance of this class.

3 The Value Manager

The value manager is a dependency tracking tool which is especially designed to support efficient reasoning about systems on quantitative physical level. It features fast inference selection and label update during behavior prediction and negligible management overhead for intermediate results. The value manager is based on Horn logic and also shares the label computation algorithms with an ATMS, but does not maintain a dependency network. Instead, it incorporates some new functionality:

- It forgets. By applying data reduction methods, resource consumption with respect to space and computation time is significantly reduced.
- It focuses on one context at a time.
- It distinguishes between short-term and long-term memory management. Separate data buffers give fast access to context specific data and use a common knowledge base to exchange learned inference results.

The main functional difference w.r.t. a focused ATMS is that the value manager actively controls resource consumption by forgetting unimportant data. The task of selecting data for removal from storage is seen here as a technical resource optimization task like garbage collection, rather than a mission critical strategic task. Therefore, the control about data reduction is assigned to the value manager and not to the problem solver. This design decision has significant consequences.

3.1 Consequences on the View of Data

A propositional view of data, which is common to all reason maintenance systems, is not sufficient for the value manager. If the value manager shall select data for removal, it must have the necessary knowledge to estimate the importance of data for further reasoning. This includes knowledge about the true meaning of data as well as methods to detect and remove redundancy. Since this knowledge cannot be formalized without assumptions about the inner structure of a datum, confinement to a certain kind of data is the price we have to pay for the extended functionality.

In our application focus, *state equations* are the atomic pieces of knowledge the problem solver reasons about. State equations are pairs $\langle v, d \rangle$ containing a model variable and a (possibly infinite) set of values which is a subset of the corresponding variable domain. They represent propositions about the possible values for a quantity of the physical system under analysis in a certain context. Every state equation with an empty value set denotes falsity. For effective data reduction, the value manager needs at least to be able to compare value sets for the same variable with respect to set inclusion. To reduce the communication overhead, the value manager should also be able to intersect those value sets.

While the rigidity of information hiding between problem solver and value manager is not as strict as in the classical approach, there is still some abstraction in the value manager's view of the inference process. For instance, the true meaning of assumptions, which are used to define the contexts of interest for the problem solver, is completely hidden from the value manager. Within a diagnostic problem, different contexts may represent system states of different candidates as well as state snapshots for dynamic systems at different points in time, or a combination of both.

3.2 Consequences on Label Completeness

Whenever a dependency tracking unit decides to remove a datum, it is possible that later on another justification may be found by the problem solver for the same datum. Together with the removed datum, the knowledge about its successors in the dependency network is lost. Consequently, when a previously removed datum gets a new justification, label completeness of its successors is lost, at least as long as the consequences have not been redrawn. Different strategies can be imagined to handle gaps in a dependency network. Their efficiency will strongly depend on the average size and absolute number of gaps. Our data reduction strategies aim at keeping memory consumption constant during iterative pruning loops and will therefore in general lead to rather large gaps. In that case, the usefulness of the dependency network itself can be doubted. Consequently, the value manager does not maintain such a data structure at all. Instead, it merely maintains compiled dependency information with respect to context-defining assumptions in the labels associated with the data. This reduces the overhead to zero when removing redundant or unimportant data. The resources additionally required for redrawing inferences depend on the desired degree of completeness. Completeness with respect to the original problem is not a realistic goal when dealing with numeric constraint networks, as is illustrated in Example 1 below. If we are satisfied with at least one (possibly not minimal) justification per datum, no redrawing is necessary at all during the investigation of a certain context. A slightly more liberal strategy suppresses the redrawing of consequences only for data which is classified as unimportant by the value manager. Independent of the chosen strategy, completeness of labels cannot be guaranteed any more at any time, not even with respect to the current focus.

Example 1 *Let CN* $= \langle \{x, y\}, \mathbb{R}^2, \{c_1, c_2, c_3, c_4, c_5\}\rangle$ *be a constraint network which contains the following constraints:*

$$\begin{array}{llll} c_1: & y = x + 1 & c_4: & y < 10 \\ c_2: & y = x * 2 & c_5: & y < 10000 \\ c_3: & y > 3 & & \end{array}$$

Let further for all $i \in \{1, \ldots, 5\}$, A_i *denote the assumption that* c_i *holds. The first two constraints obviously have only one solution:* $x = 1$ *and* $y = 2$. *Therefore, the first three constraints cannot hold at the same time, and consequently,* $\{A_1, A_2, A_3\}$ *is a conflict. But this conflict cannot be found by local propagation because the variable domains are not bounded. If we perform domain reduction based on the first three constraints, the lower bounds of the variable domains are pushed up steadily. After more than 2000 constraint evaluations, starting with* c_3 *and than iterating over* c_1 *and* c_2, *it becomes clear that no solution exists within the range of double precision machine numbers. But even this result does not guarantee that there is no solution at all. A fix point is never reached. Also domain splitting does not help to identify the conflict.*

In spite of the high number of potential propagation steps, which can be performed to compute reduced domains using different parts of the constraint network, the datum $\langle y, \{\}\rangle$ *denoting falsity can be obtained in less then 10 reduction steps by simply focusing the propagation on the smallest value sets known for each variable. Dependent on the order in which the constraints are evaluated the corresponding label will be* $\{A_1, A_2, A_3, A_4\}$ *or* $\{A_1, A_2, A_3, A_4, A_5\}$ *respectively. Obviously, both are not the minimal conflicts with respect to the original constraint problem.*

3.3 Focusing and Data Reduction Strategies

For efficient data management, focusing and data reduction techniques have to cooperate in a productive way. While the first technique tries to avoid the generation of unnecessary or unimportant data, the second aims at getting rid of ballast accumulating during the reasoning process. Both strategies must be based on the same criteria of importance with respect to the task at hand. To apply data reduction without focusing does not make sense. In general, it is more efficient to avoid the production of useless data in the first place than to remove it afterwards. So data reduction produces a real benefit only if applied to data which could not be avoided by focusing.

As described in Section 1, the current state of the art in ATMS technologies includes focusing techniques which restrict inference drawing to consequences of interesting combinations of assumptions. Especially when reasoning about numeric values, focusing on contexts of interest is necessary for efficiency, but not sufficient. In Example 1 we have seen that focusing inferences within a single context is required as well.

Publications in the area of model-based reasoning which address the subject of focusing within a single context are rare. However, an equivalent to the idea to focus inferences on most restricted values can be found in [Goldstone, 1992]. In this paper, a diagnostic system called Skordos is described. The concepts discussed address the difficulty of managing the tremendous number of possible predictions in quantitative value set propagation algorithms. Intervals of continuous domains are represented by inequalities like $x \leq 7$. A process called hibernation delays propagation of some inequalities until they become important for diagnostic reasoning. The importance of data is measured by its usefulness for finding new conflicts within the focused contexts[1]. The proposed strategy starts with the computation of consequences only for those inequalities which are the mathematically strongest for a certain variable in at least one focused context. All other consequences are delayed until conflicts have been found.

The more carefully the inference process is controlled within the focused contexts, the higher is the overhead for inference selection. For instance, to decide whether to compute consequences for a newly derived datum, in [Goldstone, 1992] an algorithm is used which determines for each focused context the corresponding mathematically strongest inequalities. The worst case complexity of that algorithm is quadratic with respect to the number of focused contexts. To focus inference drawing on those steps, whose results are valid in at least one focused context, additional checks on the set of antecedent nodes are necessary. While determining whether a

[1]In fact, [Goldstone, 1992] defines hibernation directly on diagnostic candidates, but replacing the set of candidates by an arbitrary set of focused contexts is a straightforward generalization.

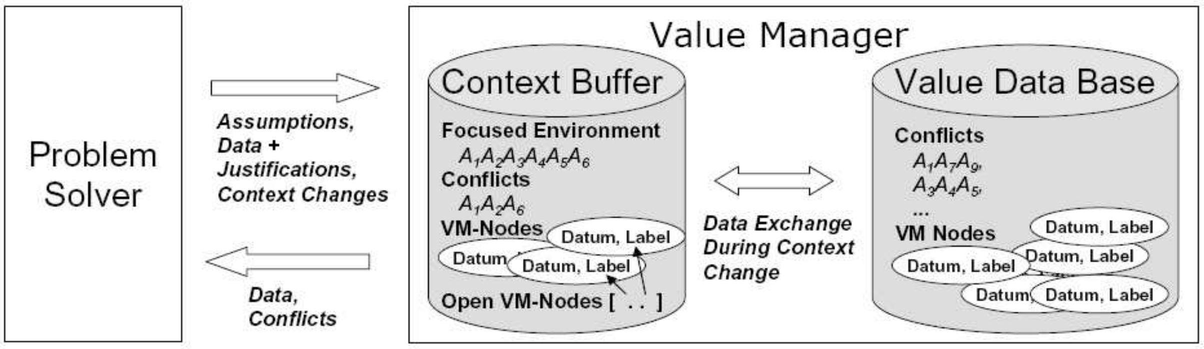

Figure 1: The value manager

set of data holds in at least one focused context can be organized very efficiently[2], the costs of testing mathematical strongness depend on the kind of data used. If state equations with interval sets as values are used, the costs are in the same order of magnitude as the costs of the inference step itself.

To avoid these extra costs, the value manager restricts reasoning to a single focused context at a time. Of course, this increases the number of necessary focus adjustments. Instead of investigating a set of candidates simultaneously, a diagnostic problem solver using a value manager is forced to investigate one candidate after another, state by state. Between every two investigations, the focused context has to be changed. The main reason why this strategy is more efficient for the value manager is the fact that the value manager forgets. As can be seen in Section 4, in relevant applications the amount of data produced during the investigation of a context is by orders of magnitude larger than the amount of data the value manager remembers after the investigation. When changing the focusing environment, still a search for the mathematically strongest data for the new focus is necessary. However, the amount of data to be checked is strongly reduced since no comparisons are necessary between intermediate results. A second reason is the explosion of focused contexts, which is caused by special assumptions needed for disjunction encoding (see Section 3.5). Without further focusing within the set of all interesting contexts, the selection of useful inference steps becomes extremely expensive.

The strategy of focusing on the mathematically strongest data defines the importance of data with respect to the currently investigated contexts. For the development of efficient data reduction strategies, this criterion is helpful, but we have to take into account another aspect of importance: The relevance of a datum for further context investigations.

The value manager provides a framework in which memory management is divided into short-term and long-term management. While short-term memory management is based on the former aspect of importance, the long-term memory management is based on the latter. Since the value manager does not maintain a dependency network, it can effectively separate the knowledge about the currently investigated context from the knowledge about previously investigated ones. We call the memory for context specific knowledge *context buffer* and the memory for learned knowledge about other contexts *value database*. Each memory maintains a set of conflicts, a set of value manager nodes – each comprising a datum, an ATMS-like label and possibly some other administrative information – and also means to look up nodes for a given variable efficiently (see Figure 1).

Short-term memory management is performed during the investigation of a context. Whenever a new datum together with at least one justification is added to the context buffer, the data reduction strategy decides whether to store it, to forget it, to combine it with one of the currently maintained nodes for the same variable, or to replace some of these nodes by it. This step includes value set manipulations as well as ATMS-like label computations. Several strategies have been tested during the development of our reference implementation. For models with a high proportion of quantitative relations, the far most efficient one turned out to be a strategy which computes intersections whenever possible and for each variable only keeps the node with the most restricted value set[3]. The main advantages of this strategy are the fast convergence and the limited memory consumption, which remains constant during propagation. While the value manager is open for strategies which also keep other than the most restricted value sets in memory, it expects all strategies to compute the most restricted value set and provides a special interface method to access the corresponding value manager node.

Long-term memory management is performed during context changes. Changing the focused environment of a context buffer includes knowledge transfer between the buffer and the value database in both directions. First, nodes which have been added to the context buffer after the last context change are selected for saving in the long-term memory. Adding clones of those nodes to the value database can include some reorganization, for example removing other nodes which are not necessarily needed any more and are unlikely to be useful in future. Then, the environments in the labels which are

[2]In [Tatar, 1994] a 2vATMS is described which checks whether a set of data holds within one of the given focused contexts in linear time with respect to set size.

[3]For debugging purposes, it is useful to maintain more than one node in case of conflicting data.

maintained by the context buffer are restricted to subsets of the new focused environment. Nodes with empty label are removed from the buffer. Finally, nodes from the database which are valid in the new focused context are cloned and added to the buffer. This addition adjusts the labels of the nodes to subsets of the new focused environment and makes use of short-term memory management, which can include intersection computations. The selection strategy of data to be stored in the value database, as well as the memory reorganization strategy, are not fixed within the value manager. For the experiments in Section 4, a very simple strategy was used. It adds all final context investigation results to the value database without any reduction on data level.

3.4 Supporting Inference Selection

Besides the strategic conflict information to direct the problem solver's search, the value manager also provides tactical support on inference selection level. After adding a newly derived state equation together with the corresponding Horn justification to a context buffer, the need for computing its consequences depends on whether the addition changed the available knowledge about that variable or not. Since the data reduction strategies of the value manager compare the newly added data with existing data for the same variable, the value manager can provide information about relevant value set reductions without extra costs. For that purpose, each context buffer maintains a list called *open nodes*. When adding new data to a context buffer, all modified and all newly created value manager nodes are added to that list. The problem solver can access that list whenever convenient to select new tasks for the agenda. After each access, the list is automatically cleared. The list of open nodes is also modified by the context buffer when changing the focused context. By tracking relevant changes during the update of the set of all maintained value manager nodes, all nodes which were affected by the context change can be identified. Local propagation benefits from that information, because the number of tasks initially put on the agenda can be reduced.

It should be emphasized that the open node list mechanism strongly differs from the consumer mechanism used in classical ATMS approaches (see [de Kleer, 1986c]). Both mechanisms have the same goal, namely to avoid unnecessary inferences, but the means are quite different. The consumer mechanism allows the problem solver to attach markers (so called consumers) to RMS nodes which indicate the consequences that should be computed for the node (and also contain the code to perform the necessary inferences). Propagation is performed by selecting one of those markers, removing it from the corresponding RMS node and performing the corresponding inferences. Advantages of this mechanism are that no inference needs to be drawn twice, even after context changes, and that the responsibility for completeness of the inference control is completely assigned to one component (the RMS). On the other hand, considerable management overhead is generated for inferences with more than one antecedent node. The more the inference process is focused within a focused context, the more useless consumers are created but never removed. For dependency trackers which apply data reduction, the consumer mechanism is even less suited since removing a node with an attached consumer may affect completeness.

3.5 Disjunction Handling

To solve cyclic dependencies in physical systems, inference methods which go beyond local propagation are needed. Current approaches combine different techniques such as local propagation, domain splitting and network decomposition. Since results of domain splitting steps do not have Horn justifications, the value manager has to be extended to support a branch&prune solver as described in [Lunde, 2005].

We first focus on the logical problem of computing sound labels. Let $L(eq)$ denote the label of the state equation eq. Labels are sets of sets of assumptions and have the same meaning as within an ATMS. The logical relationship between two split equations $\langle x, d_1\rangle$ and $\langle x, d_2\rangle$ resulting from splitting the value set of a third equation $\langle x, d\rangle$ can be expressed by means of two split assumptions A_1 and A_2. For both $\langle x, d_i\rangle$, we define $L(\langle x, d_i\rangle) = \{e \cup \{A_i\} | e \in L(\langle x, d\rangle)\}$. Since both split assumptions are related by disjunction, we can now define split assumption elimination based on hyperresolution. Let $\langle y, d_1\rangle$ be a consequence depending only on the first split assumption A_1 and $\langle y, d_2\rangle$ a consequence depending only on A_2. A sound, minimal, and consistent label for $\langle y, d_1 \cup d_2\rangle$ is obtained by removing supersets of contained environments and known conflicts from the following label:

$$\begin{aligned} L = \{e \mid \quad & e \in L(\langle y, d_1\rangle) \wedge A_1 \notin e \\ \vee \quad & e \in L(\langle y, d_2\rangle) \wedge A_2 \notin e \\ \vee \quad & \exists e_1 \in L(\langle y, d_1\rangle)\ \exists e_2 \in L(\langle y, d_2\rangle) : \\ & \quad e = e_1 \setminus \{A_1\} \cup e_2 \setminus \{A_2\}\} \end{aligned}$$

Since falsity can be expressed by arbitrary state equations which contain an empty value set, this specification also covers conflict handling.

The next question is how to control hyperresolution within the value manager. The branch&prune algorithm investigates in each recursion level the consequences of the split equations $\langle x, d_1\rangle$ and $\langle x, d_2\rangle$ in a sequence. Therefore, the problem solver could easily navigate the context buffer through both corresponding extended contexts (each defined by extending the original focused environment by one of the split assumptions). Following this idea, hyperresolution inferences could be realized as an extension of long-term memory management. Unfortunately, this usage of the value manager dramatically increases the number of context changes, and reduces the effectiveness of long-term memory management, since intermediate results now find their way into the value database. The resulting system will spend quite a large amount of time with context changes.

Therefore, the context buffer is extended instead. This extension supports domain splitting within the context buffer and eliminates the need to communicate with the value database until the original context is completely investigated or the problem solver loses the interest in it. The chosen solution exploits the depth first control strategy used by the branch&prune algorithm. In spite of maintaining just one set of value manager nodes and one set of nogoods, the extended context buffer maintains a tree called *context tree* which is composed of context tree nodes. This tree reflects the hierarchical structure of context extensions generated by domain splitting operations. Each context tree node comprises a focused environment, a set of value manager nodes, a set of nogoods, and optionally a split assumption and a split equation. The root node is initialized and marked as current node when changing the context. Child nodes are added to the current tree node whenever split operations are performed. The problem solver gets means to navigate to certain tree nodes and to evaluate their children. Evaluation is based on hyperresolution as described above and includes modification of the content of the current tree node and removal of the evaluated children.

Compared to an extension for general disjunction handling as suggested in [de Kleer, 1986b], the expressive power of the sketched functionality is rather limited. Nevertheless, it is very efficient because no search is necessary to apply hyperresolution, and because it supports removal of assumptions and data which are not needed anymore, without additional costs.

4 Experimental Results

A Java implementation of the presented concepts has been integrated into the commercial model-based engineering tool RODON (see [Lunde *et al.*, 2006]) and tested in various experiments. The results of three of them are summarized in the following. All measurements have been performed on a standard PC with 2.2 GHz and 512 MB RAM.

4.1 Automated FMEA Generation for an Automotive System

The analyzed system of the first experiment comprises the electrical equipment of the right door of a current car series. The most important components are the electronic control unit, the exterior mirror assembly, the door lock assembly, the window pane control motor, the switch assembly and some bulbs. The corresponding Rodelica[4] model is currently used in a commercial project by a major German car manufacturer to generate decision trees for workshop diagnosis. It comprises 167 subsystems and 580 atomic components, and covers more than 60 fault codes within the electronic control unit. The constraint network is composed of 8863 variables and 7338 constraints.

In this experiment, we focus on fault effect prediction for the operational state ‘window pane manually up’. This task includes 288 state investigations; in 265 states, domain splitting is activated, which leads to 1290 investigations of extended contexts. Table 1 summarizes the obtained reduction with respect to needed inference steps and computation time when using the value manager. The efficiency gains are significant even though no use is made of conflicts to direct a search. The comparatively small context navigation time confirms the decision to focus on one context at a time.

	computed data	computation time	
	total	ctx-nav	total
	#	[sec]	[min:sec]
Without value manager	45181934	0	16:55
With value manager	1051622	6.0	1:22

Table 1: Impact of the value manager on simulation performance

In spite of the large number of intermediate results, the size of the value database is quite limited at the end of the analysis. Only 23799 value manager nodes are maintained. Justifications are not maintained at all, and the storage consumption of the environments, which directly depends on the maximal size of the assumption database, is also limited. All in all 1611 assumptions are introduced during the analysis, but thanks to the removal of split assumptions when evaluating extended contexts, the size of the assumption database never exceeds 609. As a consequence, the memory space for label management is reduced by more than 50 percent and the performance of label computations is improved.

4.2 Model-based Diagnosis of a Fly-by-Wire System

The pitch elevator control system [Lunde, 2003] of the next experiments is a typical fly-by-wire system. It consists of an electronic control unit called primary flight control unit (PFCU) which controls the angle of a pitch elevator surface by means of an electro-hydraulic servo valve and a hydraulic cylinder. The top-level layout of the system (see Figure 3) also includes a power supply unit (PSU), three redundant position sensors, and some electrical wires. Figure 2 shows the actuator part of the system in more detail. Here, electrical signals are converted into hydraulic flows and finally into mechanical movements. Besides the three main components, some redundant components have been added to the design, to keep the system in a safe state in case of faults. To control

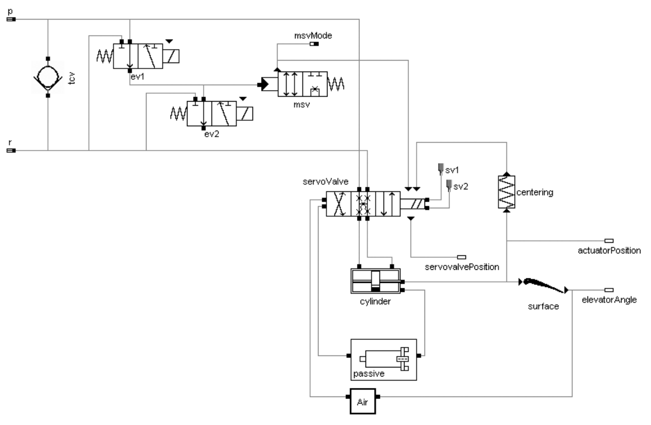

Figure 2: The actuator part of the pitch elevator control system

[4]Rodelica is a dialect of Modelica, which is a standardized object-oriented language to describe physical systems in a component-oriented and declarative way (see www.modelica.org). Rodelica differs from Modelica in some details since it uses constraints instead of differential algebraic equations to describe component behavior.

the surface angle, the PFCU compares the actual surface position with the required angle and uses the deviation to adjust the position of the electro-hydraulic servo valve. These adjustments determine the movement of the piston in the cylinder, which finally changes the actual angle of the surface.

As an example of an interesting diagnostic case we analyze the observed response of the system to a control command from the cockpit which requires the surface angle to change by 5 degrees. Starting with an initial angle of 0 degrees a movement into the right direction is observed but, due to a defect, the movement does not stop at the angle of -5 degrees.[5] We want to know which faults can explain the observed behavior.

Our Rodelica model of the pitch elevator control system is again component-oriented and exactly matches the structure shown in Figures 2 and 3. The system behavior is defined by 621 variables and 501 constraints. The main difference with respect to the automotive system of the first experiment is that this model is dynamic to a great extent. To provide the PFCU with a realistic feedback from the controlled components, difference equations are used in several parts of the system, e.g. in the cylinder. They compute Euler steps for the corresponding differential equations, which describe the behavior on physical level. A second difference is that reliable predictions are achievable even without domain splitting. Therefore, the experiments were performed with local propagation only. Due to the fact that most dynamic state variables are continuous (e.g. the cylinder position), we cannot expect too much efficiency gains by reusing results from previous state investigations. But here, the conflict computation contributes to our application, since it is basically a search problem.

In the second experiment, we simulate the response of the system to the cockpit command in nominal mode. To this purpose a sequence of 30 states is computed in which the initial value ranges of the dynamic state variables of each state are determined by the corresponding predecessor states and the difference equations. After 25 states, the surface angle converges at -5 degrees. Table 2 shows, that the value manager still improves the simulation performance, though the gains are not as impressive as in the automotive example. The high absolute number of inferences highlights the importance of data reduction.

	computed data	computation time	
	total	ctx-nav	total
	#	[msec]	[msec]
Without value manager	45318	0	1200
With value manager	30022	100	892

Table 2: Impact of the value manager on simulation performance

In the last experiment, we use the GDE based diagnostic engine of our reference implementation to diagnose the described symptom. For this purpose, we restrict the range of the actual surface position variable in the thirtieth state to the interval [$-\infty$ -6] and start diagnosis on that data. The search space is defined by the 26 component fault mode variables and their possible values. The model contains 50 single faults. The number of double faults is approximately 2400.

[5]In reality, this wrong behavior is detected by some monitors, and fault compensation functions are activated. But this mechanism is out of scope here.

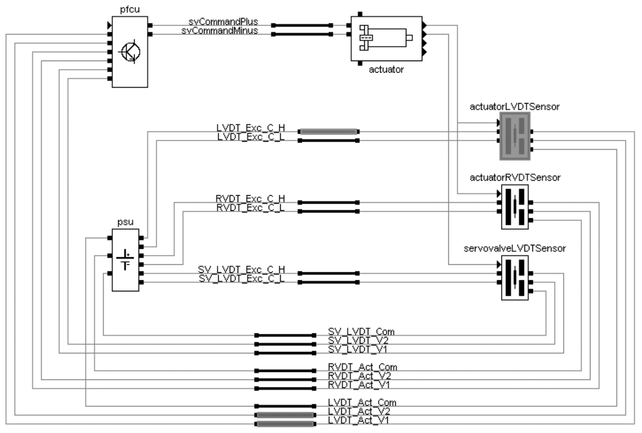

Figure 3: The pitch elevator control system

During the initial check of the candidate ‘system ok’ a conflict occurs in state 30 because the actual surface position is predicted to be around -5 degrees, which is out of the specified range. The corresponding conflict is mapped back (see e.g. [Tatar, 1996]) to the initial state. It comes out that nominal mode assumptions of 11 components are involved. This information reduces the search space, because it proves that 15 of the 26 suspicious components cannot explain the symptom, at least not by single fault. During the diagnostic process, the consistency of 61 candidates is checked with respect to the specified symptom. Conflict back-mapping leads to 11 additional conflicts between the initial fault state assumptions. At the end, the following three minimal candidates remain, which are guaranteed to be the only explanations within the scope of single and double faults.

- LVDT_Exc_C_H disconnected
- actuatorLVDTSensor disconnected
- LVDT_Act_V1 disconnected & LVDT_Act_V2 disconnected

Figure 3 shows the top-level view of the pitch elevator control system with the corresponding components highlighted. Our reference implementation needs 26.3 seconds for the computation.

In this diagnosis, the conflict sets computed by the value manager lead to a reduction of the search space size from more than 2000 to just 61 candidates. This result emphasizes that dependency tracking is very useful to solve explanatory problems, even if the underlying model is characterized by a low abstraction level and includes continuous dynamic behavior. The level of label completeness which is provided by the value manager has shown to be adequate for this application.

5 Conclusion

Quantitative reasoning about real physical systems usually leads to a huge amount of intermediate results, which may cause severe complications if the reasoning process is supported by a classical reason maintenance system. As shown in this paper, effective data reduction is crucial for efficient dependency tracking. The presented value manager is designed as a light-weight alternative to an RMS. It completely avoids inference caching and concentrates on ATMS-style label computation.

Although developed for a special model-based analysis tool, the concept of the value manager is rather general. It can be utilized to support any problem solver which reasons about values of variables and provides Horn justifications for all inferred results. The suggested solution for disjunction handling requires the problem solver to evaluate disjunctions in a special depth-first order. It is especially efficient in combination with a solver which is based on domain splitting.

Two applications have been discussed, which confirm the importance of data reduction and demonstrate the efficiency of the value manager. The significant difference in their characteristics also indicates that scalability is necessary for a widespread applicability in reliability analysis and diagnosis. The value manager is flexible regarding the actually used data reduction strategies, and thus well-prepared for task specific adaptations. Dependency tracking costs and the benefits obtained for the analysis task at hand can be balanced effectively.

The presentation in this paper focuses on performance with respect to single processor computers, but special care has been taken to support parallel computing as well. The number of context buffers within a value manager is not limited. A problem solver can benefit from this feature by delegating candidate checking to different threads. Each thread can open its own context buffer to access data. Since the data within the buffers are physically separated from the data of the commonly used value database, synchronization is only needed when changing the context of one of the buffers.

References

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[de Kleer, 1986a] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[de Kleer, 1986b] J. de Kleer. Extending the ATMS. *Artificial Intelligence*, 28(2), 1986.

[de Kleer, 1986c] J. de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, 28(2):197–224, 1986.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[Dressler and Farquhar, 1991] Oskar Dressler and Adam Farquhar. Putting the problem solver back in the driver's seat: Contextual control of the AMTS. In João P. Martins and Michael Reinfrank, editors, *Truth Maintenance Systems (ECAI-90 Workshop)*, volume 515 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1991.

[Forbus and de Kleer, 1988] K. Forbus and J. de Kleer. Focusing the ATMS. In *Proceedings of AAAI'88*, pages 193–198. MIT Press, 1988.

[Goldstone, 1992] David Jerald Goldstone. Controlling inequality reasoning in a TMS-based analog diagnosis system. In *Readings in model-based diagnosis*, pages 206–211. Morgan Kaufmann Publishers Inc., 1992.

[Kelleher and van der Gaag, 1993] Gerry Kelleher and Linda van der Gaag. The lazy RMS: Avoiding work in the ATMS. *Computational Intelligence: An International Journal*, 9(3):239–253, 1993.

[Lunde *et al.*, 2006] K. Lunde, R. Lunde, and B. Münker. Model-based failure analysis with rodon. In *Proceedings of ECAI'06, Italy*, 2006. (to appear).

[Lunde, 2003] K. Lunde. Ensuring system safety is more efficient. *Aircraft Engineering and Aerospace Technology: An international Journal*, 75(5):477–484, 2003. ISSN 0002-2667.

[Lunde, 2005] R. Lunde. Combining domain splitting with network decomposition for application in model-based engineering. In Armin Wolf, Thom Frühwirth, and Marc Meister, editors, *19th Workshop on (Constraint) Logic Programming W(C)LP 2005*, number 2005-01 in Ulmer Informatik-Berichte, pages 29–40. University of Ulm, Germany, 2005.

[Tatar, 1994] M. Tatar. Combining the lazy label evaluation with focusing techniques in an ATMS. In *Proceedings of ECAI'94, Amsterdam, the Netherlands*, 1994.

[Tatar, 1996] M. Tatar. Diagnosis with cascading defects. In *Proceedings of ECAI'96, Budapest, Hungary*, pages 511–518, 1996.