From: AAAI-83 Proceedings. Copyright ©1983, AAAI (www.aaai.org). All rights reserved.

DATA DEPENDENCIES ON INEQUALITIES

Drew McDermott

Department of Computer Science
Yale University

Abstract: Numerical inequalities present new challenges to data-base systems that keep track of "dependencies," or reasons for beliefs. Care must be taken in interpreting an inequality as an assertion, since occasionally a "strong" interpretation is needed, that the inequality is best known bound on a quantity. Such inequalities often have many proofs, so that the proper response to their erasure is often to look for an alternative proof. Fortunately, abstraction techniques developed by data-dependency theorists are robust enough that they can be extended fairly easily to handle these problems. The key abstractions involved are the "ddnode," an abstract assertion as seen by the data-dependency system, and its associated "signal function," which performs indexing, re-deduction, and garbage-collection functions. Such signal functions must have priorities, so that they don't clobber each other when they run.

1. The Problem

Programs in the field of artificial intelligence often deal with sophisticated (if small) data bases. These data bases can contain predicate-calculus formulas of greater-than-usual complexity, and often perform deductions of new formulas. Such a data base is used to keep track of an evolving problem solution, during which tentative assumptions are made and withdrawn. Since the data base is responsible for routine deductions, it ought to be responsible for undoing them when assumptions change.

A useful mechanism for aiding this process is the *data-dependency note*, a record of the reasons for belief in a formula. In the simplest case, the dependency note records that a formula follows from a set of other formulas. The note is usually diagrammed as a circular node, with arcs from the *justifying* beliefs to the node, and an arrow from the node to the supported belief, or *justificand*. See Figure 1-1.

When the justificand is inferred, it is the responsibility of the "inference engine" to record this dependency. When a justifier is erased, the justificand must be erased as well, unless there is an independent justification for it. Detecting this is done by a *reason maintenance system* (RMS). [Doyle 80] Another subtlety is that a belief may depend on the *absence* of another belief. More on this below.

*This work was supported by the National Science Foundation under contract MCS-8203980

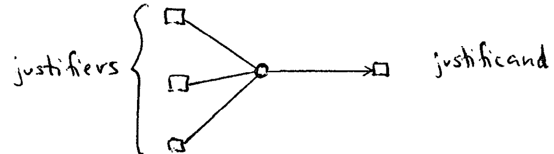

Figure 1-1: A Data-Dependency Note

Special problems are raised when you try to extend these methods to numerical inequalities. Such inequalities arise in the domains of spatial and temporal reasoning. Often a series of facts about objects and events can be captured as a set of *fuzzy maps*, in which the coordinates (and other parameters) of objects are known to within an interval. For instance, if an event E2 occurs between 5 and 6 hours after event E1, we can record that the T coordinate of E2 in the frame of E1 is [5,6]. (Assuming the scale of E1 is "1 hr per time unit.") I will write such "coordinate terms" as (T E2 E1).

The coordinate values interface to other facts in a natural way. For instance, (T E2 E1) = [5,6] may have been inferred from these facts: (1) E1 is the beginning of a painting episode EP. (2) E2 is the end of EP. (3) Paint takes between 5 and 6 hours to dry. And, in turn, from (T E2 E1)=[5,6], we may infer other facts, like this: If E1 occurs at 4 o'clock, then E2 will occur after sunset.

I call a statement like (T E2 E1) = [5,6] a *term value statement*. (I will focus on terms of the form (*parameter object frame*) in this paper, but many of the same considerations apply to terms computed from these, like (DISTANCE A B FRAME).)

The standard RMS algorithms are inadequate to update data bases involving term-value statements. The problem is that RMS algorithms simply decide whether to keep or delete a belief. In the case of an inequality, it is often worthwhile to invest some effort in recomputing the quantities involved, and see if the inequality is still true. If it is, everything it supports can stay around.

Another problem is the logical status of term-value statements. We can distinguish "weak" and "strong" interpretations of such a statement. The weak interpretation of P=[l,h] is that the true value of P lies in the interal [l, h]. (In most applications, it is irrelevant whether the endpoints are included in this interval, except for point intervals.) The strong interpretation is that no

narrower interval can be inferred. Weak interpretations are more common, but there are cases that require the strong interpretation.

Strong term-value statements are closely intertwined with *non-monotonic* inference. A strong statement depends upon the *absence* of any more informative weak value statement about the same term.

As I mentioned, such dependency nets have already been investigated. They may be diagrammed by adding "signs" to the arcs from justifiers, indicating the sense of the dependency. For instance, Figure indicates that C is believed ("IN") if A is IN and B is absent from the data base ("OUT").

But the relationship between weak and strong values cannot be handled like this, because there are potentially an infinite number of weak values that could impact on a

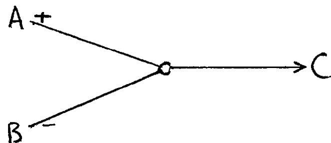

Figure 1-2: A Non-Monotonic Data-Dependency Note

strong value.

The final problem to describe is the origin of term-value statements. We can distinguish two sources: computational and non-computational. The former label applies to term values computed from other term values; the latter, to all other inferences. A computational inference involves a fact of the form

$$t_0 = f(t_1, \ldots, t_n)$$

where $t_1,\ldots,t_n$ are numerical-valued terms, and f is a computable function. I will use the word ***derivation*** for such an equality. Note that a typical equality gives rise to several derivations. E.g., A=B+C gives rise to three:

```
A=B+C
B=A-C
C=A-B
```

I will not worry about generating derivations from other equalities, but just assume derivations are generated in this form when required.

Derivations are like Steele and Sussman's "constraints" [Sussman 80] with one important difference: they are supposed to be applied using interval arithmetic. That is, if B=[3,4] and C=[-2,-1], then from the derivation A=B+C, we can infer A=[1,3].

In general, nothing prevents the creation of circular derivations, which may be difficult to make use of. Such a derivation cannot be used by substituting value of right-hand side quantities, but may require solving simultaneous equations. In this paper, I will assume that such derivations do not arise; that is, that the only circular derivation that can be obtained by substitution is of the form A=A. This is a reasonable assumption in our spatial-reasoning domain, where derivations come from coordinate transformations, which do have this property. If you translate the X coordinate of A from one frame to another and back, you wind up with the term you started with.

The spatial-reasoning domain is simpler in this respect, but more complex in others. One complexity is that there are in the abstract an infinite number of derivations for a term; any term can be derived from the value of that term in another frame of reference, plus the values of the relative coordinates of the two frames. Even in one dimension, we have

$$(\mathtt{X\ A\ F_1}) = (\mathtt{X\ A\ F_2}) \cdot (\mathtt{SCALE\ F_2\ F_1}) + (\mathtt{X\ F_2\ FF_1})$$

for all objects A and reference frames F_1 and F_2. But at any time, most such derivations will give useless answers (very wide intervals). We must provide heuristic means of finding good derivations of a quantity.

In summary, therefore, our problems are these:

1. Avoiding erasing inequalities after minor recomputation
2. Managing the non-monotonic relationship between weak and strong term-value statements.
3. Keeping track of useful derivations of quantities.

I should point out that none of these problems is found only in numerical inference. The methods described here may be adaptable to other applications. But the numerical situation makes the problems obvious and urgent.

2. Solution

The solution is more a matter of careful bookkeeping than blinding insight. Many of the techniques needed are already known from the work of Doyle and McAllester. The trick is to stack up the layers of abstraction just right.

One abstraction we need is the *memory term (memterm)*, a data structure to which is attached all the information about a numerical-value term, like (X A B), that has been computed in the past, and whose values are being kept track of. (There must be an indexing scheme to get you from the object A to this memterm, but I will neglect this issue.)

Data dependencies run between "propositional" objects (objects that have a truth value). Memterms are not propositional, but they have several propositional entities associated with them:

- Weak values (*wvals*): zero or more assertions about intervals this memterm lies in. The intersection of the intervals for all IN wvals must be non-empty, and represents the "current value" of the memterm. If there are no wvals, then the memterm has some (large) default fuzz as value. The default for a scale might be [0, 10000]; for a position, [-10000, 10000].
- Strong values (*svals*): zero or more assertions about the narrowest interval this term is known to lie in. At most one may be IN at a time, and

its interval must be the current value of the memterm (the intersection of the IN wvals' values).

- Others to be described below.

From the RMS's point of view, propositional objects are called *"ddnodes,"* and are thought of merely as things to attach data-dependency notes to, and to slap IN or OUT labels on. This abstraction, due to Jon Doyle, permits the RMS to handle a wide variety of assertion types, and not to be committed to any particular format for them. It raises the following problem, though: how can the RMS know how to do the bookkeeping (e.g., assertion indexing) associated with bringing a ddnode IN or OUT? Doyle's answer was to provide each ddnode with a "signal function," which is called by the RMS whenever a node changes label. The signal function knows how to do bookkeeping, so the RMS doesn't have to.

So wvals, svals, and other propositional objects are implemented as ddnodes, with signal functions to do their bookkeeping. These ddnodes can support other ddnodes, that also have signal functions. For instance, A∈[5,6] might support A>4. If the wval is erased, the signal function for A>4 might ask to recompute A, see if A>4 is still true, and if so, resupport it with the new wval.

Whenever the RMS adjusts the IN/OUT labels, signal functions will go off all over the data base, like popcorn (or like "demons," [Charniak 72]). For instance, suppose we have three ddnodes like those in Figure

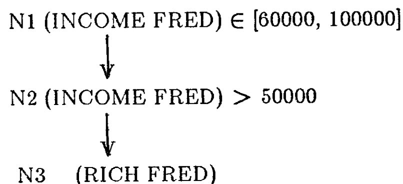

Figure 2-1: Nodes Being Updated

The signal function F1 for N1 might say: If I go OUT, free my storage. The function F2 for N2 might say, If I go OUT, recompute (INCOME FRED) and resupport me if necessary. The function F3 for N3 might say, If I go OUT, remove me from the data base.

Suppose N1 goes OUT. Then all three signal functions get called. The resulting chaos could cause inefficiency or even error. Suppose F3 goes first, and unindexes N3. Then F2 might decide N2 should be IN after all, which would require N3 to be re-indexed.

The solution is for most signal functions to do nothing immediately, but queue up functions to do the real work on a system of queues (or "agendas") with different priorities. In the example, F2 should get higher priority than F3. That way, by the time F3 runs, F2 may have already patched things up so that F3 need do nothing. One way to think about this (due to David McAllester [McAllester 82]) is that each queue maintains an "invariant," an assertion that is true when the queue is empty, as it normally is. Lower-priority queues can thus assume the invariants associated with higher-priority queues. F3 can assume that "all OUT inequalities are really unsupported," because signal functions like F2 have ensured this. Intuitively, the job of a queued function is to bring its queue's invariant "closer" to being true. When the queue is empty, the invariant is true.

A large part of applying a data-dependency system is choosing queue priorities. In particular, for the application at hand, we are going to need a queue level at which signal functions can call for memterms to be recomputed. Recomputation can mean two things: re-applying existing derivations, or finding new ones. I will say more about this shortly.

Figure 2-2 shows the ddnodes associated with a memterm.

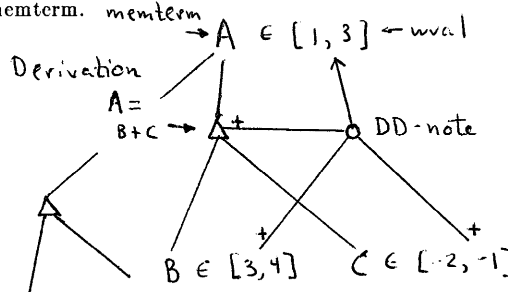

Figure 2-2: Derivation and Wvals

I use a triangular node for an active derivation, that is, one that has been generated and stored in a memterm. The inputs are connected to the base of the triangle, the output to its apex. A derivation is an assertion (i.e., it looks like a ddnode to the RMS), which connects memterms. Memterms have wvals, which are also assertions. A wval will be supported by a dd-note which includes the derivation used to compute it.

In addition to the ddnodes shown in the figure, there are some other useful propositional entities associated with memterms. First, we need a ddnode, the *derivedness-node*, which asserts that every useful derivation is IN. We need a second ddnode, the *completeness-node* for the memterm, which asserts that every IN derivation is actually contributing a wval. In addition, the completeness node depends on the the derivedness node.

Maintaining the derivedness ddnode is the responsibility of the modules using the memterm system. The first time the value of a memterm is asked for, all its derivations are found (usually heuristically), and the derivedness node is brought IN. It usually stays in from then on, but outside modules can force it OUT if they become aware of new derivations.

Maintaining the completeness ddnode is the responsibility of the memterm system itself. Whenever a new wval is added to a memterm, then all of the memterms that are derived from it are marked "incomplete." That is, their completeness node are forced OUT. Whenever the value of a memterm is sought, if it is incomplete, then its derivations are used to re-derive new wvals, and it is marked complete again.

Note that this ddnode is being used essentially as a Boolean flag. Why not just use a Boolean, and save some

storage? Because a ddnode can participate in data dependencies; that is, it can support other propositional entities. In the present context, the main propositional entity is the strong value ("sval"). Most memterms will not have a strong value most of the time, but if one is needed, we create a ddnode for it, and support that ddnode with the extant wvals, plus the completeness node for the memterm.

Normally, not much happens when a wval or sval goes OUT. But it is simple in this system to attach a signal function to a ddnode, such as an inequality, supported by such a value. Such a signal function can ask that an attempt to recompute the memterm be made. The result will be that all wvals will be up to date, the completeness node will be brought IN again, and, if necessary, a new sval will be created.

The user of a numerical inference system usually does not want to perceive it as numerical as such. He would rather treat an inequality as just another assertion. In the classic treatments of data dependencies [Doyle 79, deKleer 77], it is assumed that all likely proofs for an assertion are represented explicitly. If any is unsatisfied, then the assertion should be deleted. This is, of course, not true for beliefs about quantities, which often have an infinite number of equally reasonable potential proofs. $X > 4$ may be believed because $X = 5$, or $X \in [6,7]$, or .... We cannot generate all possible proofs, but would like to make it look to many inference programs as if that's exactly what happens.

This suggests organizing the queue hierarchy as follows:

High priority: recomputation of terms supporting important inequalities

Intermediate priority: User's queue levels

.....

Low priority: Storage allocation; in particular, reclaiming storage of OUT ddnodes

The reason for giving term recomputation a high priority is so that a term will be recomputed if possible before anyone examines it. This allows us to simulate keeping an infinite number of potential of proofs around, because the recomputer will look for another IN proof on demand. The user's code can then always assume the invariant "If inequality is OUT then there is no reason to believe it."

The reason to put storage reclamation at such a low priority level is to allow "experimenting" with the data base. The user may wish to try various hypotheses before adopting a final one. During this experimentation, ddnodes may temporarily become "dead," that is, OUT and not supporting anything. By "locking" the low-priority queue, we can postpone garbage-collecting these nodes until they are worthless for certain.

3. Results and Conclusions

The code for this system has been written, and is in the process of being debugged. It is written in NISP, a portable LISP macro package that runs in ZetaLisp, ILISP, Franz, and T. For now, the most likely host dialect is T, a Scheme-like Lisp that runs on Apollos and Vaxes.

Data-dependency notes have a mixed reputation. Most people appreciate the need for a flexible method of updating a data base. However, they have trouble seeing exactly how dependency notes meet this need. Some of the literature (e.g., [Doyle 79, deKleer 77]) seems to imply that data dependencies can, by making and retracting assumptions, take over most of the control structure of a practical program. This is unlikely for all but the simplest applications. In practice, data dependencies are simply one tool among many, which perform two tasks quite well: erasing beliefs that are no longer justified, and queueing "signal functions" to rethink their justifications. The conclusion of the present paper is that this ability extends naturally to beliefs involving numbers, specifically, beliefs about the intervals quantities lie in.

ACKNOWLEDGEMENTS: Thanks to Rod McGuire for a lot of ideas.

References

[Charniak 72] Charniak, E.
Towards a Model of Children's Story Comprehension.
Technical Report 266, MIT Artificial Intelligence Lab, 1972.

[deKleer 77] de Kleer, Johan, Doyle, Jon, Steele, Guy L., and Sussman, Gerald J.
Explicit control of reasoning.
Memo 427, MIT AI Laboratory, 1977.
Also in *Proc. Conf. on AI and Prog. Lang.* , Rochester, which appeared as *SIGART Newsletter* no. 64, pp. 116-125.

[Doyle 79] Doyle, J.
A truth maintenance system.
Artificial Intelligence 12:231-272, 1979.

[Doyle 80] Doyle, Jon.
A model for deliberation, action, and introspection.
TR 581, MIT AI Laboratory, 1980.

[McAllester 82] McAllester, David.
Reasoning Utility Package User's Manual, Version One.
Memo 667, MIT AI Laboratory, 1982.

[Sussman 80] Sussman, G.J. and Steele, G.L.
CONSTRAINTS -- A Language for Expressing Almost-Hierarchical Descriptions.
Artificial Intelligence 14(1):1-39, August, 1980.