Multiagent Truth Maintenance

Michael N. Huhns and David M. Bridgeland

Microelectronics and Computer Technology Corporation
Artificial Intelligence Laboratory
3500 West Balcones Center Drive
Austin, TX, U.S.A. 78759-6509
huhns@mcc.com

January 26, 2001

Abstract

In this paper we define the concept of logical consistency of belief among a group of computational agents that are able to reason nonmonotonically. We then provide an algorithm for truth maintenance that guarantees local consistency for each agent and global consistency for data shared by the agents. Furthermore, we show the algorithm to be complete, in the sense that if a consistent state exists, the algorithm will either find it or report failure. We discuss the implications and limitations of this for cooperating agents, and describe several extensions. The algorithm has been implemented in the *RAD* distributed expert system shell.

1 Introduction

Two trends have recently become apparent out of the widespread use of knowledge-based systems: 1) systems are being developed for larger and more complicated domains, and 2) there are attempts to use several small systems in concert when their application domains overlap. Both of these trends argue for knowledge-based systems to be developed in a distributed fashion, where modules are constructed to interact productively. The individual modules then are characteristic of intelligent agents. The interconnected agents can cooperate in solving problems, share expertise, work in parallel on common problems, be developed modularly, be fault tolerant through redundancy, represent multiple viewpoints and the knowledge of multiple experts, and be reusable. Additional motivations are presented in [Huhns 1987] and [Gasser and Huhns 1989]. But in order for these agents to coordinate their activities and cooperate in solving mutual problems, it is essential that they be able to communicate with each other. Further, in order for them to interact intelligently and efficiently, we believe that the agents must be able to assess and maintain the integrity of the communicated information, as well as of their own knowledge.

2 Knowledge Base Integrity

There are many desirable properties for the knowledge base of an expert system or agent, such as completeness, conciseness, accuracy, and efficiency. For an agent that can reason nonmonotonically, there are additional properties used to describe the *integrity* of the agent's knowledge base: stability, well-foundedness, and logical consistency. A *stable* state of a knowledge base is one in which 1) each knowledge base element that has a valid justification is believed, and 2) each knowledge base element that lacks a valid justification is disbelieved. A *well-founded* knowledge base permits no set of its beliefs to be mutually dependent. A *logically-consistent* knowledge base is one that is stable at the time that consistency is determined and in which no logical contradiction exists. Depending on how beliefs, justifications, and data are represented, a consistent knowledge base may be one in which no datum is both believed and disbelieved (or neither), or in which no datum and its negation are both believed. These concepts are often extended to include other types of contradictions.

In addition, any algorithm that attempts to maintain well-founded stable states of a knowledge base, such as one of the many algorithms for truth maintenance [Doyle 1979, de Kleer 1986, Martins and Shapiro 1988, McAllester 1980, Russinoff 1985], should be *complete*, in the sense that if a well-founded stable state exists, the algorithm will either find it or report failure. In general, we desire each agent in a multiagent environment to have a complete algorithm for maintaining the integrity of its own knowledge base.

However, the above definitions of properties for a single knowledge base are insufficient to characterize the multiple knowledge bases in such a multiagent environment. When agents that are nonmonotonic reasoners exchange beliefs and then make inferences based on the exchanged beliefs, then new concepts of knowledge-base integrity are needed. In addition, the relevant concept of global truth maintenance becomes especially problematic if agents must compute their beliefs locally, based on beliefs communicated and justified externally. The next sections extend the above definitions to the multiagent case.

2.1 The JTMS

We presume that each agent has a problem-solving component, separate from its knowledge base, that makes inferences and supplies the results to the knowledge base. Our discussion applies to the set of beliefs that are held and maintained in this knowledge base. In particular, we focus on the systems for maintaining beliefs known as truth-maintenance systems (TMS) [Doyle 1979].

TMSs are a common way to achieve knowledge base integrity in a single agent system, because they deal with the frame problem, they deal with atomicity, and they lead to efficient search. Furthermore, the justification networks they create can be used for nonmonotonic reasoning, problem-solving explanations to a user, explanation-based learning, and multiagent negotiation. Our research is based on a *justification-based* TMS, in which every datum has a set of justifications and an associated status of `IN` (believed) or `OUT` (disbelieved).

In the example considered below, an initial state of a distributed knowledge base is given and presumed consistent. Our goal is to construct a consistent extension of this state or determine that no such extension exists.

The distributed TMS (DTMS) algorithm presented for this task is most often invoked to restore consistency when a consistent state is disrupted by altering the justification for a datum.

2.2 Consistent Beliefs among Agents

Consider a network of many agents, each with a partially-independent system of beliefs. The agents interact by exchanging data, either unsolicited or in response to a query. Each agent has two kinds of data in its knowledge base, as illustrated in Figure 1:

Shared Data Beliefs that the agent has shared with another agent at some time in the past.

Private Data Beliefs that the agent has never shared with another agent.

A private datum might become a shared datum by being told to another agent, or by being the answer to some other agent's query. Once shared with other agents, a datum can only become private again if unresolvable inconsistencies ensue. Each shared datum is shared by a subset of the agents in the network—precisely those that have either sent or received assertions about the datum.

We extend the concept of knowledge-base consistency stated above by defining four degrees of consistency and well-foundedness that are possible in a multiagent system.

Inconsistency: one or more agents are individually inconsistent, i.e., at least one agent has a private datum without a valid justification and labeled `IN`, or a private datum with a valid justification and labeled `OUT`.

Local Consistency: each agent is individually consistent, i.e., no private `OUT` datum has a valid justification, and each private `IN` datum has a valid justification. However, there may be global inconsistency among agents: there may be a shared datum that one agent believes to be `IN` and another believes to be `OUT`.

Local-and-Shared Consistency: each agent is individually consistent and groups of agents are mutually consistent about any data they all share,

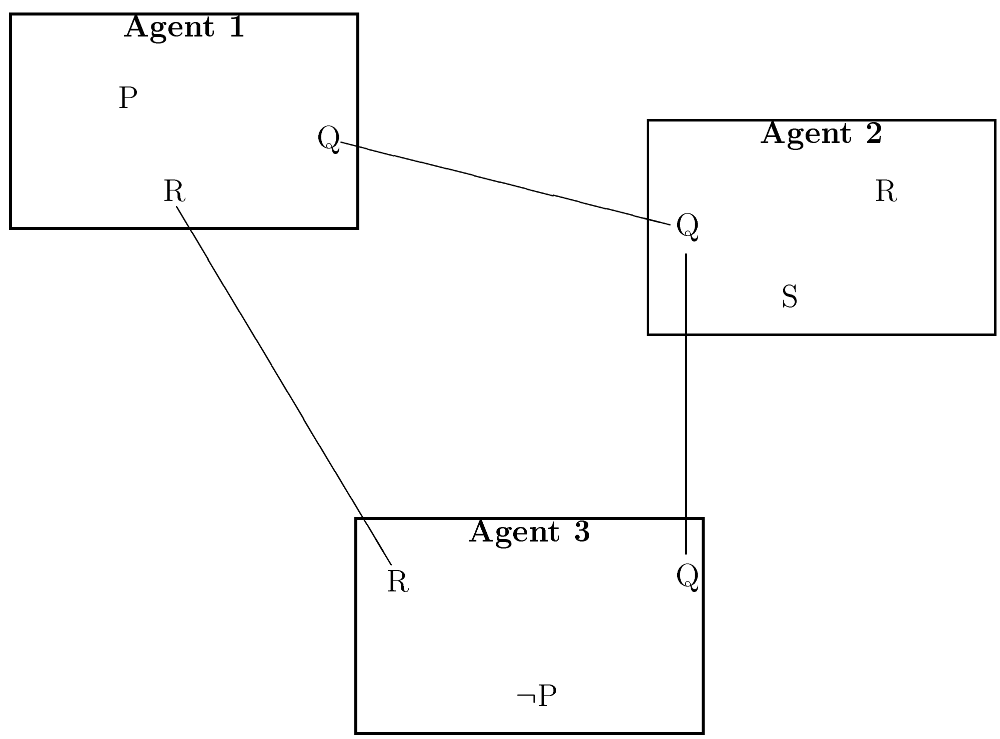

Figure 1: Shared and private data—belief R is shared between Agents 1 and 3, but is private to Agent 2

i.e., each shared datum is either `IN` in all the agents that share it or `OUT` in all those agents. There is, however, no global consistency.

Global Consistency: the agents are both individually and mutually consistent, i.e., their beliefs could be merged into one large knowledge base without the status of any datum necessarily changing.

In the absence of interagent communication, and presuming the local environment of each agent is consistent, then Local Consistency should hold. The introduction of interagent communication, however, tends to drive the system towards Inconsistency, because the agents might receive data that conflict with their current beliefs.

In most cases, we believe that Global Consistency is desirable. However, it may not be practical due to knowledge representation considerations, such as a desire to maintain different viewpoints, and due to computational considerations, as follows. Agents usually have two types of knowledge—explicit and implicit [Levesque 1984]. *Explicit knowledge* is that which an agent directly represents and maintains beliefs about. *Implicit knowledge* is that which an agent could infer or compute, given sufficient resources. Agents are typically unaware of all of their implicit knowledge and, therefore, do not maintain any beliefs about it. One result of performing inference is that some implicit knowledge may become explicit. When this occurs, an agent's TMS must compute a belief status for this new explicit knowledge. To guarantee Global Consistency would require interagent communication and global relabeling after any inference step that affected the explicit knowledge of an agent, i.e., almost every step. This is clearly impractical.

However, truth maintenance has proven beneficial and even essential in many applications, so clearly consistency is desirable. The trade-off is to find a compromise that is practical. We have chosen to maintain consistency over a subset of the knowledge: the shared data. Our presumption here is that the shared data are the most important, because they affect the problem solving of another agent, and so special effort should be made to maintain their consistency. If an agent has communicated a belief to another agent, it seems useful and reasonable for the agent to communicate a retraction of that belief when he infers it. And because the shared data are explicit, it is apparent when their belief status is affected and interagent relabeling is required. It thus becomes practical for each agent to strive for Local-and-Shared Consistency.

Although our goal is to maintain Local-and-Shared Consistency, we at times allow the system to fall short of this goal in order to permit agents to have different viewpoints. In this case, one agent may hold a belief that is known to be contrary to the belief of a second agent. The agents do not attempt to resolve this dispute if resolution would result in their being individually inconsistent: "they agree to disagree." A consequence of this is that these agents should then not believe any data originating from each other, unless the originating agent can prove that its belief for that data is independent of the disputed data. In our current implementation, the agents simply consider each other "unreliable" (see Section 3.2); they then disbelieve everything that originates from an agent they consider unreliable.

Ill-Foundedness: one or more agents have beliefs that are internally ill-founded.

Local Well-Foundedness: each agent has beliefs that are internally well-founded; however, one or more agents may have shared data that are `IN` but have no valid justifications in any agent.

Local-and-Shared Well-Foundedness: each agent has beliefs that are internally well-founded, and every `IN` shared datum has a valid justification in some agent; however, there may be ill-founded circularities of beliefs among groups of agents.

Global Well-Foundedness: every datum has a globally valid justification and no set of data, whether local to an agent or distributed among a group of agents, is mutually dependent.

Our algorithm for multiagent truth maintenance enables each agent to strive for Local-and-Shared Well-Foundedness, as well as Local-and-Shared Consistency. However, we require several characteristics of the agents involved in order for this to succeed, as follows:

1. The agents must be completely cooperative, i.e., they must be willing to negotiate about and revise any of their knowledge.

2. The agents must be able to provide explanations or justifications for their beliefs; this implies that their knowledge must be represented explicitly.

3. The agents must have identical representations of knowledge, i.e., the same ontology and epistemology. Their syntax, however, may be different.

We do not require that the agents exist in a consistent world, so it is possible for agents legitimately to have different viewpoints and to disagree. As discussed in Section 4, we also relax some of these requirements in order to allow less capable agents to benefit from our algorithm.

3 A Multiagent TMS

In the classical TMS, a datum can be either `IN` or `OUT`. For the DTMS, we refine the `IN` status to two additional statuses: `INTERNAL` and `EXTERNAL`. An `INTERNAL` datum is one that is believed to be true, and that has a valid justification. An `EXTERNAL` datum is believed to be true, but need not have a valid justification. Intuitively, the justification of an `EXTERNAL` datum is "so-and-so told me." Hence, only a shared datum can be `EXTERNAL`. For Local-and-Shared Well-Foundedness, a shared datum must be `INTERNAL` to at least one of the agents that shares it and either `INTERNAL` or `EXTERNAL` to the rest of the agents.

The only justification-based TMS labeling algorithm known to be complete [Russinoff 1985] takes a generate and test approach, first unlabeling a collection of data, then attempting to relabel that collection. On failure to relabel, a superset of the last unlabeled collection is unlabeled. We take a similar approach in the DTMS. Since new data in one agent can change not only the status of that agent's beliefs, but also that of other agents, our unlabeling and subsequent labeling will sometimes involve multiple agents.

The support status of a shared datum is jointly maintained by several agents. Hence, a single agent is generally not free to change the status of a shared datum on its own accord. It must coordinate with the other agents so that they are all consistent on the status of the datum. Central to the DTMS then is the single agent operation of **label-with-constraints**. **label-with-constraints** is a variation of classical TMS labeling in which the statuses of some data—though unlabeled—are fixed by external requirements.

More precisely, **label-with-constraints** is given a network of data. Some of the data have statuses of `IN`, `OUT`, `INTERNAL`, or `EXTERNAL`. Other data are unlabeled. For each shared datum, there is a *desired* label of either `OUT`, `INTERNAL`, or `EXTERNAL`. **label-with-constraints** either finds a consistent well-founded labeling of the network that satisfies the shared data requirements, or it reports failure. Our implementation of **label-with-constraints** is a variation of Russinoff's well-founded and complete labeling algorithm [Russinoff 1985].

3.1 Algorithm Schema

The DTMS performs a double generate and test. Relabeling is invoked by the addition or removal of a justification. When invoked, the DTMS does the following three things:

1. Unlabels some data, including the newly justified datum and, presumably, its consequences. This unlabeled data set might be confined to a single agent or it might span several agents. If a shared datum is unlabeled in some agent, it must be unlabeled in all the agents that share it.

2. Chooses labelings for all the unlabeled shared data, as defined above.

3. Labels each of the affected agents with respect to the requirements imposed by the shared data, invoking **label-with-constraints** as described above. If any of the affected agents fails to label successfully, it then backtracks. It either chooses different labelings for the shared data (step 2), or unlabels a different set of data (step 1).

This schema will be refined later, but some nice properties emerge at this abstract level:

- Any labeling found by the DTMS will have Local-and-Shared Consistency and Well-Foundedness.

- If the two generate steps are exhaustive, the DTMS is complete: it will find a labeling should one exist.

Note that these properties are true both of the DTMS algorithm described in this paper, and any other algorithm that conforms to this schema.

3.2 Unlabeling

When the DTMS algorithm is invoked, it starts by unlabeling a collection of data. This collection may be confined to a single agent or it may span many agents. However, it must meet the following constraints:

1. It must include the datum that originally acquired the new justification.

2. A shared datum that is unlabeled in one agent must be unlabeled in all the agents that share it.

3. On failure to label the collection, it must generate a new collection of unlabeled data. To guarantee completeness, the generation must be exhaustive: it must eventually generate a collection sufficiently large that failure to label it means the whole network cannot be labeled.

However, this specification of unlabeling is underconstrained, because many algorithms would satisfy these constraints. For example, on any status change one could unlabel all data in all agents. This global unlabeling satisfies all the constraints and is also quite simple, but is also too inefficient for practical use. The global unlabeling does reveal two DTMS principles that motivate the more complex algorithm presented later:

Principle 1 *Belief changes should be resolved with as few agents as possible.*

Principle 2 *Belief changes should be resolved by changing as few beliefs as possible.*

Most belief changes can be resolved by changing things only "downstream" of the new justification, i.e., those data, termed *consequents*, that directly or indirectly depend on the datum newly justified. It is sometimes necessary to move "upstream" as well, and relabel data, termed *foundations*, that directly or indirectly support the status of the newly justified datum. Consider the knowledge base for a single agent shown in Figure 2 [Russinoff 1985]. Here, datum Q acquires the new justification shown in dashed lines. If only P and Q are reassigned, the system is forced to report an unsatisfiable circularity. In order to restore stability, the status of the data upstream from P must be changed: if the system makes S `IN` and R `OUT`, both P and Q can be `OUT`.

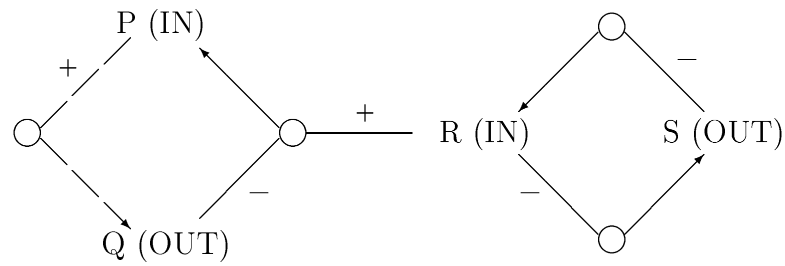

Figure 2: Relabeling upstream data to resolve an odd loop

However, relabeling the foundations of a datum causes much greater computational complexity than relabeling its consequences. We express this by the following principle:

Principle 3 *Belief changes should be resolved among consequences, if possible; foundation relabeling should be minimized.*

The above principles motivate the algorithm **unlabel**. It attempts to minimize both the involvement of other agents and the unlabeling of upstream data, but prefers the former to the latter. **unlabel** is invoked on a list containing either the newly justified datum, when **unlabel** is first invoked, or the unlabeled data that could not be labeled on a previous invocation. **unlabel** attempts to find yet to be unlabeled private data downstream of those already labeled. If there are none, it looks for shared data downstream, unlabeling those in all the agents that share them, and also unlabeling private data downstream of the shared data. Finally, if there are no shared data downstream that are yet to be unlabeled, it unlabels data just upstream of all the downstream data, and all private data downstream of that. If there is nothing yet to be unlabeled upstream, **unlabel** fails and, in fact, the data admit no Local-and-Shared Consistent and Well-Founded labeling.

Consider the DTMS network in Figure 3. There are two agents, Agent 1 and Agent 2, and they share the datum T. As in Figure 2, the initial labeling shown in the diagram is perturbed by the addition of the new dashed justification for Q. Agent 1 initially unlabels just the changed datum and private data downstream, P and Q, but there is no consistent relabeling. Hence, Agent 1 unlabels all shared data downstream of P and Q, and all private data downstream from there: P, Q, both Ts, and U. Again labeling

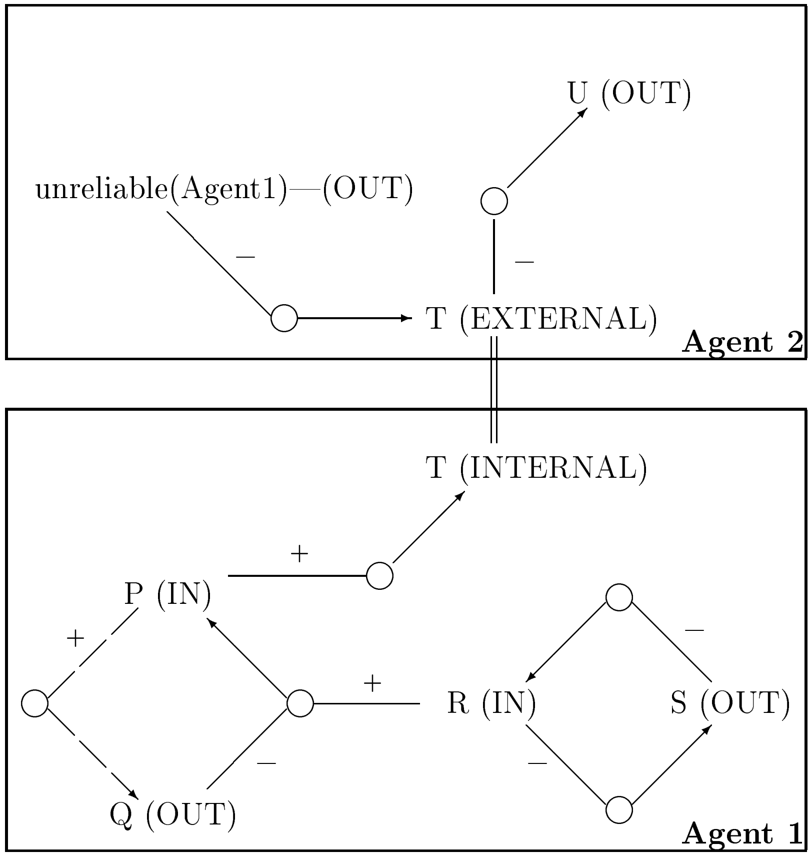

Figure 3: A DTMS network before a new justification for datum Q (shown dashed) is added; this invokes the DTMS algorithm and results in a relabeling of the network

fails. Since there is no further shared data downstream, Agent 1 and Agent 2 unlabel upstream and privately downstream from there: P, Q, Ts, U, R, and S. Now labeling succeeds, with S and U `IN` and everything else `OUT`, as shown in Figure 4. Had labeling failed, **unlabel** would not be able to unlabel more data, and would report that the network is inconsistent.

Since we do not wish to leave agents in inconsistent and unlabeled states, we would have the agents drop the constraint that T is a shared datum. They would each record a new belief that the other agent is `unreliable`, as illustrated in Figure 5, and then attempt to relabel their beliefs individually. If this again fails for any agent, that agent would continue problem solving, but with inconsistent beliefs.

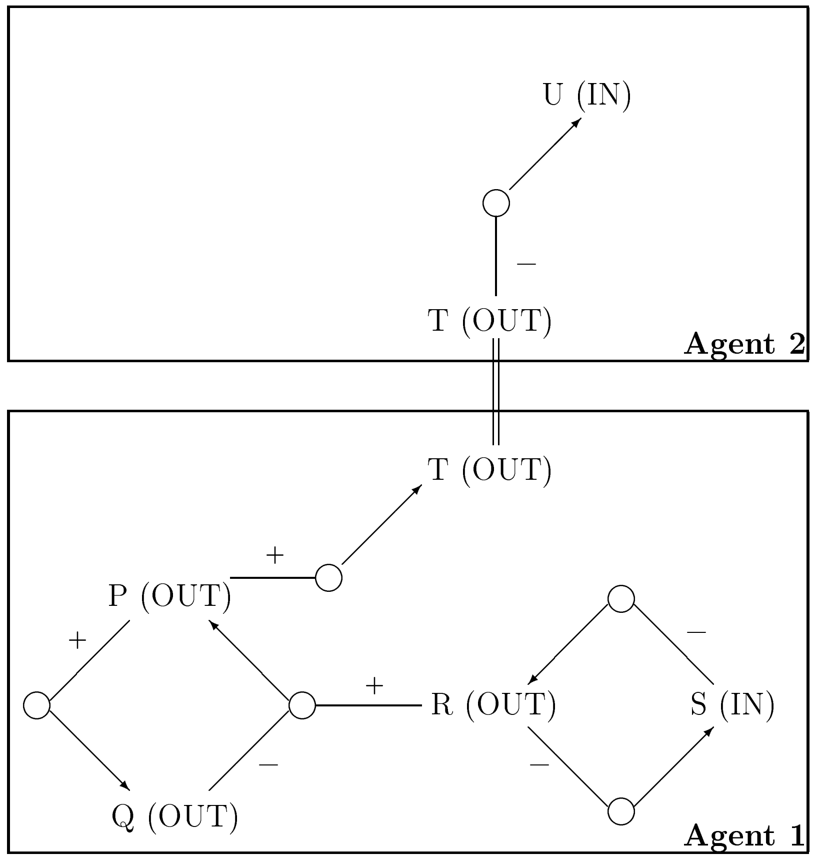

Figure 4: The resultant stable labeling of the justification network that is produced by the DTMS algorithm

Agent 1	**Agent 2**
T (IN)	T (OUT)
unreliable(Agent2)—IN	unreliable(Agent1)—IN

Figure 5: The beliefs of two agents that consider each other unreliable, because they are unable to agree on a labeling for previously shared datum T

Distributed System Issues

To be implemented, the algorithm **unlabel** needs to be distributed. This is straightforward if each agent keeps track of which data are currently unlabeled and reports to other agents only whether yet to be unlabeled data became unlabeled. Upstream and downstream messages mention only which shared datum is affected, and the corresponding acknowledgments report only whether new data were unlabeled.

When a group of agents is labeling, their beliefs are in an unsuitable state for reasoning. Hence, queries from other agents must be queued until labeling is complete. However, if two agents share a datum, and are involved in separate unlabeling tasks (i.e., initiated by different changes to beliefs), then deadlock could occur. Fortunately, the separate unlabeling tasks can be combined into a single one, with a corresponding combination of the subsequent labeling.

3.3 Labeling

Once an area of data is unlabeled, the DTMS must pick candidate labels for the shared data, such that each datum is either `OUT` in all the agents that share it, or `INTERNAL` in at least one agent and `INTERNAL` or `EXTERNAL` in the rest. Any exhaustive means of picking will guarantee completeness. The following Prolog code shows one means:

```
label-shared([], Shared-labels, Shared-labels).

label-shared([Agent|Agents], So-far, Final) <--
  label-one-shared(Agent, So-far, New),
  label-shared(Agents, New, Final).
```

label-shared relates its first argument—a list of agents—and its third argument—a list of shared data and their labels. The second argument is used to pass commitments about labels for shared data to recursive calls to **label-shared**. The relation calls **label-one-shared**, attempting to assign labels to a single agent that are consistent with those already assigned to others. If it finds such an assignment, it recursively attempts to find labels for the other agents. On failure, it backtracks and looks for alternatives to the previous single-agent labeling.

This algorithm could be implemented on a fully connected multiagent system by making each agent responsible for generating labels that are consistent with others already generated, as in **label-one-shared**, and implementing the recursive nature of **label-shared** with a message passed depth-first from agent to agent. This message needs to contain a list of agents already visited, so that none are revisited, and a list of the already labeled shared data. Also, before an agent passes a message to another, it needs to record its state for future backtracking. The shared-data labeling fits into the larger labeling process as follows:

```
label(Agents) <--
  label-shared(Agents, [], Shared),
  label-private(Agents, Shared).

label-private([],Shared).

label-private([Agent|Agents],Shared) <--
  find-local-shared(Agent, Shared, Local-Shared),
  label-with-constraints(Agent, Local-Shared),
  label-private(Agents, Shared).
```

The private labeling follows the shared data labeling. The private data are labeled one agent at a time. First the relation **find-local-shared** extracts the shared labels relevant to a single agent from the list of all the shared labels. Then **label-with-constraints** attempts to label the private data consistently with the shared data. Any failure to label causes backtracking. This algorithm will find a Local-and-Shared Consistent and Well-Founded labeling of an unlabeled set of data in a collection of agents.

Unfortunately, this algorithm has poor computational performance. If there is no consistent labeling of the agents, the DTMS will generate all shared data labelings and attempt to label each privately. The performance can be improved by interleaving the labeling of the shared data and the private data. Failure in a single agent to find a private labeling consistent with the labels of the shared data will then cause earlier backtracking:

```
label(Agents) <-- label-internal(Agents, []).

label-internal([], Shared).
```

```
label-internal([Agent|Agents], So-far) <--
  label-one-shared(Agent, So-far, New),
  find-local-shared(Agent, New, Local-shared),
  label-with-constraints(Agent, Local-shared),
  label-internal(Agents, New).
```

label-internal could be implemented by a message passed depth-first from agent to agent. This message needs to contain a list of the agents visited so far, and a record of the labels given so far to the shared data.

Consider again Figure 3. R, S, P, Q, U, and both Ts have now been unlabeled. Agent 1 chooses labels for T and attempts to label his private data in a consistent manner. If Agent 1 chooses `INTERNAL` as T's label, he finds there is no labeling of his private data to make T internally justified. A next attempt with `EXTERNAL` is consistent (with S `IN` and everything else `OUT`), and Agent 1 passes his label of T to Agent 2. Agent 2 must then label T `INTERNAL`, but finds there is no way to label U. Agent 2 then backtracks and Agent 1 tries a final attempt to label T, this time as `OUT`. This succeeds with S `IN` and everything else `OUT`, and Agent 2 can also label T `OUT` by labeling U `IN`.

3.4 Computational Efficiency

In Section 2 we defined several degrees of consistency and well-foundedness. It is interesting to compare the theoretical worst-case complexity for each of the degrees of consistency. The complexity of nonmonotonic justification-based truth maintenance has been shown to be NP-complete [Elkan 1990]. Furthermore, it is shown that even incremental updating of an already-labeled TMS justification network is NP-complete. Assume that there are m agents and, without loss of generality, that each has n clauses of arbitrary form. Each agent can label its knowledge base in time proportional to $\mathcal{O}(2^n)$. This is the time required for the agents to maintain Local Consistency.

Next, assume that each agent shares k clauses with the other $m-1$ agents. Each clause can have one of the three labels `INTERNAL`, `EXTERNAL`, and `OUT`. Our algorithm will explore each of these labelings until it finds a consistent one, resulting in 3^k possibilities in the worst case. For each of these possibilities, the agent may have to relabel the rest of its clauses. The time required is thus $\mathcal{O}(3^k 2^n)$.

If each agent shared all of its data with every other agent, which is equivalent to maintaining Global Consistency, then the time required in the worst case is $\mathcal{O}(3^{n(m-1)} 2^n)$, which is approximately $\mathcal{O}(2^{mn})$. This is the same time needed to maintain consistency if each of the clauses were distinct and they were all combined into one agent. Distributing the clauses over m agents and using the algorithm we present herein does not result in any savings in complexity. The savings we achieve is due to relaxing the requirement for Global Consistency. To summarize, Local-and-Shared Consistency increases worst-case complexity by a factor of $\mathcal{O}(3^k)$ over Local Consistency, but reduces it by $\mathcal{O}(3^{n(m-1)-k})$ compared to Global Consistency.

These are somewhat negative results, in that NP-complete problems are intractable in theory. However, nonmonotonic JTMSs have been used successfully for many significant-sized applications [Petrie 1989]. One reason for this is that the above results are for worst-case complexity, which does not seem to hold in practice, because typical justification networks are not fully connected and have many independent elements. Another reason can be based on a result reported by [Provan 1990], who showed that if the representation is restricted to Horn clauses, then the complexity is $\mathcal{O}(n)$, where n is the number of such clauses. Because much of a typical knowledge base is in the form of Horn clauses, this portion can be labeled in linear time, leaving a smaller portion that might require exponential time.

3.5 Optimizations

Our DTMS algorithm is triggered only when there is a change in belief status of a shared datum. It introduces no overhead for monotonic reasoning among agents or for nonmonotonic reasoning within an individual agent. It does, however, admit several local optimizations:

1. An agent can forego the labeling of its unlabeled shared data by **label-one-shared** and instead label everything that is unlabeled by **label-with-constraints**. This requires a more sophisticated **label-with-constraints** that can generate `INTERNAL` and `EXTERNAL` labels for the shared data, as well as `IN` and `OUT` labels for the private data.

2. An agent can keep a record of label attempts, caching for each combination of shared data labels whether it succeeded or failed to find a

private labeling. A call to **label-with-constraints** will first consult the cache, thus avoiding redundant work. Ordering the shared data and then indexing the combinations in a discrimination net would be a good implementation for this cache.

3. In the above DTMS algorithm, only one agent is active at a time. However, there is something productive that the other agents can do: fill in their label caches by attempting to find private labelings for shared data combinations not yet examined. In fact, this effort could be guided by other agents. If agent 1 shares data with agents 2, 3, and 4, when agent 1 passes a **label-internal** message to agent 2, it could advise agents 3 and 4 about its decisions on the labels of shared data. Then other agents could work only on that portion of their caches that are consistent with agent 1's decision.

Other algorithms are also possible. A parallel implementation of truth maintenance is presented in [Fulcomer and Ball 1989]. The implementation assumes one datum per agent, but could be generalized and adapted to handle more knowledge per agent and, possibly, multiagent truth maintenance as defined here.

Single-agent truth maintenance has been shown to be a type of constraint satisfaction problem [Bodington and Elleby 1988, de Kleer 1989]. It is shown in [Yokoo *et al.* 1990] that distributed truth maintenance can be described as a distributed constraint satisfaction problem, and an efficient algorithm is presented that yields Global Consistency and Well-Foundedness (if it exists). The algorithm is limited in its applicability, however, because it makes the strong assumptions that 1) agents either know all of the constraints on their variables or know the agents that know them, and 2) all variables are known in advance, i.e., that all knowledge is explicit (as defined in Section 2.2). The paper does, however, suggest utilizing its technique for asynchronous backtracking for our DTMS algorithm, where all shared data are known in advance by all affected agents. This would produce another optimization in addition to those listed above.

3.6 Example of Use

Figures 6–8 show an example of the use and operation of our DTMS algorithm as two agents interact. Figure 6 shows the initial state of the knowledge bases for the two agents, Agent 1, an investor, and Agent 2, a stockbroker. First, Agent 1 asks Agent 2 to recommend a stock. Agent 2 recommends `XCorp`, which causes Agent 1 to believe that he should buy that stock, as shown in Figure 7. However, Agent 2 then learns (not shown) that the basis for his recommendation, that `XCorp` is cash-rich, is no longer valid. He revises his beliefs and notifies Agent 1 that he has retracted his recommendation for `XCorp`. The final knowledge bases for the agents are shown in Figure 8.

It is interesting to note that these agents maintain different beliefs about whether or not they can afford to buy the stock of `XCorp`. Allowing this difference in belief—this global inconsistency—is useful in this case. It allows different viewpoints to be represented, it simplifies the representation of knowledge, in that the predicate `afford` really should have an additional argument indicating *which* agent can or cannot afford the stock, and it eliminates the interactions that would be needed to resolve the difference. Of course, the DTMS would detect, and subsequently correct, the difference if the agents ever share or discuss this predicate.

4 Discussion

There have been many other attempts to develop systems of cooperating agents or knowledge sources. Early attempts, based on the blackboard model, involved agents with independent knowledge bases. The independence was achieved by restricting agent interactions to modifications of a global data structure—a blackboard—and by minimizing overlap in the agents' knowledge. Later systems allowed richer agent interactions and overlapping knowledge, but the agents were required to have consistent knowledge and to reason monotonically. This led to representational problems, because different experts in the same domain often have different perspectives and *conflicting* knowledge, making it difficult to construct a coherent problem-solving system for that domain. Earlier solutions were to allow inconsistent knowledge bases; this enabled the conflicting knowledge to be represented, but it did not confront the problem of how to resolve the conflicts.

Other researchers have explored negotiation as a means to mediate among conflicting agents. These systems have involved either monotonic reasoners, such as [Sycara 1989], or nonmonotonic, but memoryless, reasoners, such as [Zlotkin and Rosenschein 1989], i.e., reasoners that simply discard old solu-

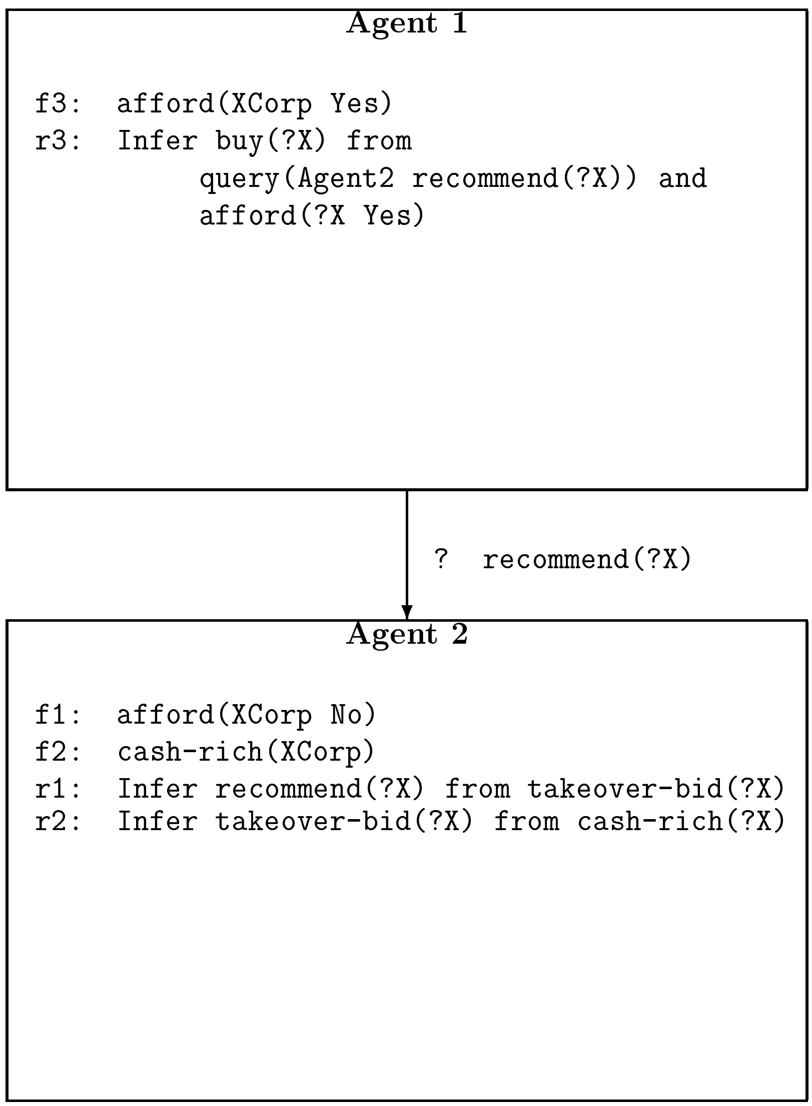

Figure 6: Initial knowledge bases of two interacting DTMS-based agents, before Agent 1 queries Agent 2 for a recommendation

Agent 1

```
f3:  afford(XCorp Yes)
r3:  Infer buy(?X) from
          query(Agent2 recommend(?X)) and
          afford(?X Yes)
f4:  recommend(XCorp)
      Status - EXTERNAL
      Shared with - Agent2
      Justification - ()
f5:  buy(XCorp)
      Status - IN
      Justification - (f3 f4 r3)
```

$\uparrow$ `recommend(XCorp)`

Agent 2

```
f1:  afford(XCorp No)
f2:  cash-rich(XCorp)
r1:  Infer recommend(?X) from takeover-bid(?X)
r2:  Infer takeover-bid(?X) from cash-rich(?X)
f3:  recommend(XCorp)
      Status - INTERNAL
      Shared with - Agent1
      Justification - (f1 r1 r2)
```

Figure 7: Knowledge bases of two interacting DTMS-based agents after Agent 2 has replied to Agent 1's query

Agent 1

```
f3:  afford(XCorp Yes)
r3:  Infer buy(?X) from
          query(Agent2 recommend(?X)) and
          afford(?X Yes)
f4:  recommend(XCorp)
      Status - OUT
      Shared with - Agent2
      Justification - ()
f5:  buy(XCorp)
      Status - OUT
      Justification - (f3 f4 r3)
```

↑ Relabel recommend(XCorp)

Agent 2

```
f1:  afford(XCorp No)
f2:  cash-rich(XCorp) ⟶ OUT
r1:  Infer recommend(?X) from takeover-bid(?X)
r2:  Infer takeover-bid(?X) from cash-rich(?X)
f3:  recommend(XCorp)
      Status - OUT
      Shared with - Agent1
      Justification - (f1 r1 r2)
```

Figure 8: Final knowledge bases of two interacting DTMS-based agents after Agent 2 has notified Agent 1 that a fact must be relabeled

tions and re-solve in the face of conflicts.

Shared, or common, knowledge is analyzed in [Halpern and Moses 1990]. This paper shows that common knowledge is an essential state for reaching agreements and coordinating action. However, common knowledge can never be attained. Instead, agents can act as if they had common knowledge if they are *internally knowledge consistent*. This is a different kind of consistency from ours, where an agent is internally knowledge consistent if it can never conclude that the knowledge principle ($K\varphi \models \varphi$, "only true things are known") has been violated. The DTMS might be useful for maintaining this kind of knowledge consistency also.

Another approach is to consider the research efforts in multiple-context truth-maintenance systems [de Kleer 1986, Martins and Shapiro 1988] from a distributed viewpoint. These systems manipulate belief spaces, or contexts, in order to remove inconsistent ones. One might imagine each belief space represented and maintained by a different agent. However, in this model, the belief spaces themselves do not interact and, in fact, the belief revision system treats each space separately.

A notable exception to this is the work of [Mason and Johnson 1989], who developed a distributed *assumption-based* TMS, called a DATMS. In this system, agents interact by exchanging data, with their associated assumption sets, and `NOGOODS`, i.e., bad assumption sets. `NOGOODS` are sent only to a user-specified list of agents, not necessarily to all of the agents that might be affected by the `NOGOODS`. Belief revision is thus problem dependent. As in the DTMS, justifications are not communicated; contrary to the DTMS, the justifications are limited to monotonic implications. By associating each fact with its originator, agents in the DATMS are able to maintain separate belief spaces. This enables the use of communicated `NOGOODS` to be restricted to the invalidation or revision of communicated data only. As a result, an agent cannot affect another agent's purely local beliefs. The restriction is imposed by requirements of the application domain for which the DATMS was designed. But it allows agents to disagree about an exchanged datum. The agents therefore have Local-and-Shared Well-Foundedness, but only Local Consistency. A distributed ATMS that results in Global Consistency is presented in [Yokoo and Ishida 1988].

The system we presented herein, although it satisfies our goal of Local-and-Shared Consistency, nevertheless suffers from several deficiencies:

- First, by not supporting some form of explicit negation or reasons for disbelief in a datum, the system allows an agent with less information to dominate an agent with more. For example, if two agents each have an identical justification for belief in a shared datum, and one agent learns a fact that invalidates its justification, the other agent's still-valid justification will be sufficient for both to continue believing in the datum.

- Second, our algorithm can involve significant computational overhead, if the agents have shared a large amount of data, if the data have many connections to the rest of the agents' belief networks, and if the statuses of their beliefs change frequently.

- Third, we believe unsatisfiable circularities are more likely in a distributed system.

- Fourth, our system deals with only one type of inconsistency: inconsistent belief status for a shared datum. There are many other types of inconsistencies that can arise among a group of agents, such as

  1. one agent could believe a datum, while another agent could believe its negation, and these beliefs could be used to justify a datum they share
  2. two agents could believe an object to be of two incompatible types, i.e., they could use different terms for the same object
  3. two agents could believe two different objects are of the same type, i.e., they could use the same term for two different objects
  4. the agents' beliefs may be inconsistent at a semantic level, e.g., one agent could believe that an object is made out of plastic, while another believes that it is made out of steel.

- Fifth, the DTMS is agnostic about what data should be shared among agents. The research of [Courand 1990] and [Galliers 1990] has produced principles governing the incorporation of data from other agents. In [Courand 1990], agents share goals and plans in order to achieve the mutual beliefs necessary to take cooperative action. They incorporate beliefs from other agents only when the resultant belief system will be more *coherent*, a measure (not defined) that includes consistency, completeness, plausibility, and suitability for action. [Galliers 1990] presents a set of heuristics for deciding which beliefs to incorporate. Her agents are skeptical, rather than cooperative, and prefer to adopt beliefs that reinforce existing beliefs without revising any. The Rational Distributed Reason Maintenance System [Doyle and Wellman 1990] similarly suggests a basis for deciding rationally which beliefs and plans to revise.

Our system allows a developer to signal some of the other types of inconsistencies mentioned above by use of the predicate **contradiction**. This causes an implementation of knowledge-guided, dependency-directed backtracking to be invoked individually by each agent [Petrie 1987]. We are investigating how to apply this type of backtracking across agents—we view it as a type of negotiation. As part of our research, we are developing a mechanism for interagent negotiation that uses the beliefs supplied by our DTMS. We are also investigating the likelihood and severity of the other listed deficiencies in real-world application domains.

The DTMS algorithm has been implemented in the *RAD* distributed expert system shell. *RAD* provides each agent with forward and backward inferencing, a frame system with multiple inheritance, simple built-in frames for modeling other agents, distributed truth maintenance on all data and slot values, a mechanism for contradiction resolution, integrated neural nets for hybrid reasoning, interagent communication via TCP/IP, and an implementation of the Contract Net [Smith 1980] for task distribution. *RAD* allows other types of agents, including OPS5 expert systems and human agents, to interact with DTMS-based *RAD* agents. But these types of agents do not have a DTMS as described here. How does the DTMS function with such agents?

- These agents (obviously) never initiate a relabeling.

- The presumed behavior for agents during relabeling is tacit agreement, i.e., they reply only if they disagree with a change in belief status. In general, when nonDTMS-based agents are informed of a change in belief status during a relabeling, they do not respond, and relabeling proceeds by the DTMS-based agents.

- Human agents have an opportunity to object to a proposed relabeling of a datum they share with a DTMS-based agent, thereby enabling them to influence the next stable belief state of an agent.

- For the special case of OPS5 agents, notification of a change in belief status from `INTERNAL` or `EXTERNAL` to `OUT` for a shared datum causes the datum to be removed from working memory. Notification of a change from `OUT` to `INTERNAL` causes the datum to be added to working memory.

RAD is a first step toward cooperative distributed problem solving among multiple agents. It provides the low-level communication and reasoning primitives necessary for beneficial agent interactions, but it does not yet guarantee successful and efficient cooperation. The next steps will require increased intelligence and capabilities for each agent, resulting in more sophisticated agent interactions occurring at a higher level.

References

[Bodington and Elleby 1988] R. Bodington and P. Elleby, "Justification and Assumption-Based Truth Maintenance Systems: When and How to Use Them for Constraint Satisfaction," in B. Smith and G. Kelleher, eds., *Reason Maintenance Systems and Their Applications*, Ellis Horwood, Ltd., Chichester, UK, 1988, pp. 114–133.

[Courand 1990] Gregory J. Courand, "Cooperation Via Consensus Formation," in *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence*, Chapter 10, Bandera, Texas, October 1990.

[de Kleer 1986] Johan de Kleer, "An Assumption-Based TMS, Extending the ATMS, and Problem Solving with the ATMS," *Artificial Intelligence*, Vol. 28, No. 2, March 1986, pp. 127–224.

[de Kleer 1989] Johan de Kleer, "A Comparison of ATMS and CSP Techniques," *Proceedings IJCAI-89*, Detroit, MI, August 1989, pp. 290–296.

[Doyle 1979] Jon Doyle, "A Truth Maintenance System," *Artificial Intelligence*, Vol. 12, No. 3, 1979, pp. 231–272.

[Doyle and Wellman 1990] Jon Doyle and Michael P. Wellman, "Rational Distributed Reason Maintenance for Planning and Replanning of Large-Scale Activities (Preliminary Report)," in *Proceedings DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, CA, Morgan Kaufmann, San Mateo, CA, November 1990, pp. 28–36.

[Elkan 1990] Charles Elkan, "A Rational Reconstruction of Nonmonotonic Truth Maintenance Systems," *Artificial Intelligence*, Vol. 43, No. 2, May 1990, pp. 219–234.

[Fulcomer and Ball 1989] Rosanne M. Fulcomer and William E. Ball, "Correct Parallel Status Assignment for the Reason Maintenance System," *Proceedings IJCAI-89*, Detroit, MI, August 1989, pp. 30–35.

[Galliers 1990] Julia Rose Galliers, "Cooperative interaction as strategic belief revision," in *Draft Proceedings of International Working Conference on Cooperating Knowledge-Based Systems*, Keele, England, October 1990, pp. 79–81.

[Gasser and Huhns 1989] Les Gasser and Michael N. Huhns, eds., *Distributed Artificial Intelligence, Volume II*, Pitman Publishing, London, 1989.

[Halpern and Moses 1990] Joseph Y. Halpern and Yoram Moses, "Knowledge and Common Knowledge in a Distributed Environment," *Journal of the ACM*, Vol. 37, No. 3, July 1990, pp. 549–587.

[Huhns 1987] Michael N. Huhns, ed., *Distributed Artificial Intelligence*, Pitman Publishing, London, 1987.

[Levesque 1984] Hector J. Levesque, "A Logic of Implicit and Explicit Belief," *Proceedings AAAI-84*, Austin, TX, August 1984, pp. 198–202.

[Martins and Shapiro 1988] Joao P. Martins and Stuart C. Shapiro, "A Model for Belief Revision," *Artificial Intelligence*, Vol. 35, No. 1, May 1988, pp. 25–79.

[Mason and Johnson 1989] Cindy L. Mason and R. R. Johnson, "DATMS: A Framework for Distributed Assumption Based Reasoning," in [Gasser and Huhns 1989], pp. 293–317.

[McAllester 1980] David A. McAllester, "An Outlook on Truth Maintenance," AI Memo No. 551, Artificial Intelligence Laboratory, MIT, Cambridge, MA, August 1980.

[Petrie 1987] Charles J. Petrie, "Revised Dependency-Directed Backtracking for Default Reasoning," *Proceedings of AAAI-87*, Seattle, WA, July 1987, pp. 167–172.

[Petrie 1989] Charles J. Petrie, "Reason Maintenance in Expert Systems," *Künstliche Intelligenz*, Vol. 2, 1989, pp. 54–60.

[Provan 1990] Gregory M. Provan, "The Computational Complexity of Truth Maintenance Systems," *Proceedings ECAI-90: Ninth European Conference on Artificial Intelligence*, Pitman Publishing Ltd., London, 1990, pp. 522–527.

[Reinfrank 1986] Michael Reinfrank, "Reason Maintenance Systems," *Workshop on Truth Maintenance Systems*, Berlin, October 1986

[Russinoff 1985] David M. Russinoff, "An Algorithm for Truth Maintenance," MCC Technical Report No. ACA-AI-062-85, Microelectronics and Computer Technology Corporation, Austin, TX, April 1985.

[Smith 1980] Reid G. Smith, "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers*, Vol. C-29, No. 12, December 1980, pp. 1104–1113.

[Sycara 1989] Katia Sycara, "Multiagent Compromise via Negotiation," in [Gasser and Huhns 1989], pp. 119–137.

[Yokoo and Ishida 1988] Makoto Yokoo and Toru Ishida, "The Use of the ATMS in DAI Problems," NTT Communications and Information Processing Laboratories Technical Report, 1988.

[Yokoo *et al.* 1990] Makoto Yokoo, Toru Ishida, and Kazuhiro Kuwabara, "Distributed Constraint Satisfaction for DAI Problems," in *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence*, Chapter 18, Bandera, Texas, October 1990.

[Zlotkin and Rosenschein 1989] Gilad Zlotkin and Jeffrey S. Rosenschein, "Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains," *Proceedings IJCAI-89*, Detroit, MI, August 1989, pp. 912–917.