Final Report on Rational Distributed Reason Maintenance for Planning and Replanning of Large-Scale Activities

USAF Rome Laboratories Contract F30602-91-C-0018
January 31, 1991—March 31, 1994

Jon Doyle, Principal Investigator
Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technology Square
Cambridge, Massachusetts 02139
Tel: (617) 253-3512, Fax: (617) 258-8682
doyle@lcs.mit.edu

Abstract

Efficiency dictates that plans for large-scale distributed activities be revised incrementally, with parts of plans being revised only if the expected utility of identifying and revising the subplans improve on the expected utility of using the original plan. The problems of identifying and reconsidering the subplans affected by changed circumstances or goals are closely related to the problems of revising beliefs as new or changed information is gained. But traditional techniques of reason maintenance—the standard method for belief revision—choose revisions arbitrarily and enforce global notions of consistency and groundedness which may mean reconsidering all beliefs or plan elements at each step. We develop revision methods aiming to revise only those beliefs and plans worth revising, and to tolerate incoherence and ungroundedness when these are judged less detrimental than a costly revision effort. We use an artificial market economy in planning and revision tasks to arrive at overall judgments of worth, and present a representation for qualitative preferences that permits capture of common forms of dominance information.

Contents

1 Planning and replanning

Planning is necessary for the organization of large-scale activities because decisions about actions to be taken in the future have direct impact on what should be done in the shorter term. But even if well-constructed, the value of a plan decays as changing circumstances, resources, information, or objectives render the original course of action inappropriate. When changes occur before or during execution of the plan, it may be necessary to construct a new plan by starting from scratch or by revising a previous plan. In fact, replanning may be worthwhile even when the new situation does not deviate significantly from prior expectations. The original plan may have been constructed to perform acceptably over a wide range of possible circumstances, and knowing more about the particular situation encountered may enable construction of strategies which are better suited to the case at hand. Thus rather than viewing activities as the result of a sequential process in which one first plans and then executes, we view the process more as depicted in Figure 1, which indicates planning and replanning interleaved with each other and with ongoing execution and observation actions. In this model of the plan construction process, the planner and replanner

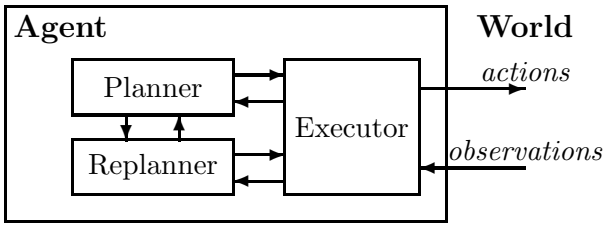

Figure 1: An integrated planning, replanning, and execution system.

continually evaluate and revise the existing plan in light of what happens in the world. The distinction between planning and replanning is that the latter uses the existing plan to focus attention on a restricted set of decisions about actions to be performed. The tight coupling of the planning and replanning modules is indicative of the strong interactions between their designs. Knowledge about the capability of the replanner dictates where planning effort should be spent anticipating particular contingencies, and the replanner requires reciprocal access to the planner's reasons for adopting the current strategy in order to intelligently adapt it for changing situations.

There are two central decisions surrounding the replanning process. First, given the information accrued during plan execution, which remaining parts of the original plan should be salvaged and in what ways should other parts be changed? Incremental modification is more efficient than wholesale replanning, but a restriction to local changes can compromise the value of the revised plan. Second, to what extent should the planner attempt to avoid the need for replanning by anticipating contingencies and providing for them in the original plan? Contingency planning improves the capacity for response when replanning time is limited, but the return on investment rapidly diminishes as the likelihood of particular contingencies decreases.

To replan effectively in crisis situations, replanning must be *incremental*, so that it modifies only the portions of the plan actually affected by the changes. Incremental replanning first involves *localizing* the potential changes or conflicts by identifying the subset of the extant beliefs and plans in which they occur. It then involves *choosing* which of the identified beliefs and plans to keep and which to change. For greatest efficiency, the choices of what portion of the plan to revise and how to revise it should be based on coherent expectations about and preferences among the consequences of different alternatives so as to be *rational* in the sense of decision theory.

2 Rational planning and replanning

According to decision theory, a choice is rational if it is of maximal expected utility among all alternatives. But planning and replanning involve at least two different sorts of decisions, and applying the standard of rationality to each yields different notions. The fundamental distinction is that between *result* rationality and *process* rationality. Rationality of result measures how efficiently the plan achieves specified objectives. Complementing this, rationality of process measures how efficiently the planner expends its efforts in constructing the plan, that is, the efficiency of the operation of the combined planning/replanning system. While most investigations of planning have focused on one or the other, both elements are essential to the overall rationality of the planning system.

Making any process rational is not easy, for straightforward mechanizations of decision-theoretic definitions can require more information than is available and more computation than is feasible to use that information. Sophisticated mechanizations are more tractable, but the main tool for achieving rationality in reasoning is to distinguish between *explicit* and *implicit* rationality in processes. Computational mechanisms may calculate and compare expected utilities in order to make explicitly rational choices. Explicit rational choice promises to be most useful in guiding some of the larger meta-level decisions about whether to replan globally or incrementally, and in choosing which contingencies call for planned responses. For the more numerous small decisions that arise, however, explicitly representing and calculating expected utilities may not be worth the cost. Instead, the more useful approach is to apply non-decision-theoretic reasoning mechanisms whose results may be justified as rational by separate decision-theoretic analyses. Such mechanisms may be viewed as "compiling" the results of explicit rational analysis into directly applicable forms. Each of these ways of implementing rationality is best in some circumstances, since compilation is not always possible or worthwhile.

Examples of implicitly rational procedures abound in AI under the name of heuristics. For instance, the "status quo optimality" heuristic [29, Section 6.4.1] constrains the set of possible revisions under the assumption that the current plan is optimal. In particular, the replanner need only respond to the specific changes. A related example is application of the basic theorem of optimization that says that if the only change is a tightening of constraints, the currently optimal plan remains optimal if it remains feasible. Another example, of somewhat different character, is provided by the assumptions made by nonmonotonic default rules. These rules for coming to assumptions are important forms of heuristics. Though algorithms for determining sets of conclusions from default rules do not involve any explicit rationality calculations, the conclusions drawn can be shown to be Pareto optimal sets, that is, rational choices of conclusions when the default rules are interpreted as preferences over states of belief [8, 11]. Viewed this way, default rules encode compiled preferences, and mechanisms for deriving sets of conclusions from default rules constitute an example of an implicitly rational choice mechanism.

Process rationality enters the task of planning in numerous ways. For example, in the development of a plan, contingency plans should be included only when the expected utility of preparing them is sufficiently great: if the contingency is likely to occur and if the costs of developing it in advance are less than the costs of constructing it under the tighter constraints existing while executing the enclosing plan. Similarly, a portion of a large plan should be revised only if, given the new information, the expected costs and benefits of identifying which plan elements need revising outweigh those expected for either using the original portion or replanning from scratch.

Making these judgments requires information about the likelihoods, costs, and benefits of different sorts of contingencies and planning responses. This includes the likelihood of specific contingencies arising, their importance if they do arise, and the costs of planning for them; similarly,

the likelihood of one part of the plan being affected by changes in another, the importance of those changes, and the costs of determining and effecting them.

While many of the likelihoods involved in planning derive from the specifications of the task, the costs and benefits of reasoning steps involved in planning are functions of the underlying representational and reasoning architecture. The theory of computation supplies some abstract notions of computational costs, such as worst-case time and space taken by Turing machines. However, significant differences in reasoning time and space can be lost in the translation to Turing machines, and the worst case is not the only one of interest. Use of the theory of rational decisions effectively in making judgments about plan revision requires realistic measures of computational costs and benefits appropriate to the particular architecture of the planner, as well as expectations appropriate to the domain of planning. Our development of the planning architecture attempts to make formalization and estimation of these measures more direct.

Most treatments of decision theory focus on single decisions. Those that treat decisions over time usually assume that the preferences of the decision-maker remain constant (or change little) over the interval in question. But such assumptions are inappropriate to many situations of replanning within large-scale activities. Due to the scope of the activities, it is unrealistic to expect human planners and authorities guiding the activities to be able to articulate all their preferences in advance. In addition, wars and other extended activities significantly increase the likelihood that some of the authorities in question will be replaced, either through death, discharge, coups or elections. In such settings, a replanning system must be prepared to accept changes to its preferences as well as changes in more "objective" information. Accordingly, we seek to make replanning truly flexible, accommodating changes in any element of information, including the problem specification, background knowledge, and preferences.

3 Reason maintenance for incremental replanning

Replanning in an incremental and local manner requires that the planning procedures routinely identify the assumptions made during planning and connect plan elements with these assumptions, so that replanning may seek to change only those portions of a plan dependent upon assumptions brought into question by new information. Consequently, the problem of revising plans to account for changed conditions has much in common with the problem of revising beliefs in light of new information. In both cases, one must determine which existing beliefs or plans conflict with the new information, on what these existing beliefs or plans depend, and what gaps in plans or beliefs appear as the revisions or updates are made. That is, one must localize the potential changes or conflicts by identifying the subset of the extant beliefs and plans in which they occur. Similarly, both belief revision and plan revision involve choosing which of the identified beliefs and plans to keep and which to change. In addition, the problem of providing for contingencies has much in common with the problem of choosing rules for reasoning by default, for both involve setting up primary plans or beliefs and the secondary plans or beliefs to use when the primary ones are not applicable.

The standard approach to belief revision, backtracking, and default reasoning is to use a reason maintenance system (RMS) to connect original information with derived conclusions and assumptions. Reason maintenance may be used in a similar way to revise plans as well as beliefs by indicating the dependence of plans on beliefs and on other plans, thus indicating the relevant portions for revision and the conflicts between prior plans and new circumstances. This possibility was, in fact, one of the original motivations for reason maintenance systems (see [4, 5]). However, to employ reason maintenance techniques as integral parts of the replanning process requires re-

assessing and rethinking most of the architectures for reason maintenance developed previously, for these architectures do not make rational choices, they do not distribute effort, and they do not fit cleanly into existing methods and architectures for planning.

3.1 The traditional conception of reason maintenance

In the traditional conception, a reason maintenance system acts as a subsystem of a more general system for reasoning. It revises the database states of the overall system by using records of inferences or computations to trace the consequences of initial changes. By keeping track of what information has been computed from what, such it reconstructs the information "derivable" from given information. Although we find it convenient to think of these bits of information and derivations as beliefs and arguments, one may apply reason maintenance more generally to all sorts of computational or mental structures, including intentions, preferences, and other plan elements.

For concreteness, we describe the structure of the particular reason maintenance system RMS developed by the author [5]. We may formalize its essential features as follows, simplifying its complex actual structure in ways that do not matter for the present discussion. (See [6, 7, 11] for more detailed discussions.) States of RMS contain two types of elements: *nodes* and *reasons*. We write $\mathcal{N}$ to denote the set of possible nodes, and $\mathcal{R}$ to denote the set of possible reasons. RMS uses nodes to represent information (beliefs, desires, rules, procedures, database elements, etc.) of significance to the overall reasoning system, but those "external" meanings have no bearing on the "internal" operation of RMS. RMS uses reasons to represent inferences, or more precisely, specific inference rules. Because nodes need not represent beliefs, RMS imposes no logic on nodes. Instead, the only relations among nodes are those indicated explicitly by reasons. These may, if desired, encode logical relations directly. In the simplest setting, no nodes are reasons, so that each state consists of a set $N \subseteq \mathcal{N}$ of nodes and a set $R \subseteq \mathcal{R}$ of reasons.[1] We say that each node in N is *in* (the set of beliefs), and that each node in $\mathcal{N} \setminus N$ is *out* (of the set of beliefs).

The original RMS provided two types of reasons: *support-list* reasons and *conditional-proof* reasons. For simplicity, we will ignore conditional-proof reasons and assume that all reasons are support-list reasons. Each support-list reason takes the form (I, O, c) where $I, O \subseteq \mathcal{N}$ and $c \in \mathcal{N}$. The components I, O and c are called the *inlist*, the *outlist*, and the *consequent*, respectively. We call the reason *monotonic* if O is empty, and *nonmonotonic* otherwise. Each reason is interpreted as a rule stipulating inferences the RMS must make, according to which the consequent holds if all of the nodes in the inlist are held and none of the nodes in the outlist are held. Adding a reason of the form $(\{\}, \{\}, c)$ makes c a premise or basic belief; adding a monotonic reason with a nonempty inlist corresponds to adding an ordinary argument step, and adding a nonmonotonic reason lets RMS adopt a belief as an assumption.

A state (N, R) is a *legal* state of RMS just in case N consists exactly of the *grounded* consequences of the reasons R, that is, N *satisfies* every reason in R and if every node in N is supported by a noncircular argument from the *valid* reasons in R. Formally, (N, R) is a legal state just in case

1. If $(I, O, c) \in R$, $I \subseteq N$, and $N \cap O = \emptyset$, then $c \in N$; and
2. If $n \in N$, then there is a finite sequence $\langle n_0, \ldots, n_m \rangle$ of elements of $N \cup R$ such that $n = n_m$ and for each $i \leq m$, either $n_i \in R$, or there is some $j < i$ such that

[1] It is easy to construct theories in which reasons are nodes themselves, and so may support other reasons. In such theories, one may express all reasons as defeasible reasons and express all changes of belief through addition of reasons. See [7] for details.

(a) $n_j = (I, O, n_i)$,

(b) for each $x \in I$, $x = n_k$ for some $k < j$, and

(c) $x \notin N$ for each $x \in O$.

Each set of reasons supports 0 or more legal states. For example, $\{(\{x\}, \{x\}, x)\}$ supports none, $\{(\{\}, \{\}, x)\}$ supports one, and $\{(\{\}, \{x\}, y), (\{\}, \{y\}, x)\}$ supports two if $x \neq y$.

Whenever the reasoner adds or deletes reasons, RMS updates the set of nodes to produce a new legal state. Because a single set of reasons may support any of several sets of nodes, these updates involve choices. The original RMS update algorithm was designed to attempt to update states conservatively. That is, if one takes the current state (N, R) and modifies R to obtain R', RMS should choose a new legal state (N', R') with a set of nodes N' as close as possible to the current set N. More precisely, RMS should choose N' so that for any other legal state (N'', R'), neither $N \triangle N'' \subseteq N \triangle N'$ nor $N \triangle N' \subseteq N \triangle N''$ holds, where $\triangle$ denotes symmetric difference ($X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$). Due to the difficulty of quickly computing this form of conservative updates, however, RMS only approximated conservative updates. Reason maintenance systems developed later use simpler notions of conservatism which may be computed more rapidly (see, for example, [19, 23].

The RMS also supports the operation of the *dependency-directed backtracking* (DDB) system, which attempts to defeat assumptions underlying those nodes identified as *contradictions*. Now consistency does not matter to RMS in adding or deleting reasons; instead, the basic operations of RMS only maintain coherence of reasons and nodes. (Many discussions of reason maintenance misrepresent the truth by claiming that RMS maintains consistency of beliefs. This misrepresentation may stem from the somewhat misleading role of logical consistency in nonmonotonic logic [20].) Since RMS has no knowledge of the meanings of nodes, the reasoner must tell it which nodes represent contradictions. RMS, in turn, informs DDB whenever a contradiction node becomes believed, at which point the backtracker attempts to defeat the arguments supporting the contradiction node by defeating assumptions underlying those arguments. DDB never attempts to remove reasons or premises, only to defeat nonmonotonic assumptions. If the argument for the contradiction node does not depend on any of these (i.e., it consists entirely of monotonic reasons), DDB leaves the contradiction node in place as a continuing belief.

Just as RMS seeks to minimize changes through its incremental update algorithm, DDB seeks to effect minimal revisions in choosing which assumption to defeat. Specifically, the backtracker defeats only "maximal" assumptions that do not depend on other assumptions. This means, in effect, that DDB topologically sorts the beliefs supporting the contradiction node by viewing the reasons comprising the supporting argument as a directed hypergraph. The maximally-positioned assumptions in this ordering of beliefs then constitute the candidates for defeat. The backtracker then chooses one of these assumptions and defeats the reason supporting the chosen assumption by providing a new reason for one of the nodes in its outlist. The new defeating reason is entirely monotonic, and expresses the inconsistency of the chosen assumption with the other maximal assumptions and with other maximally-ordered beliefs in the contradiction node's support. The actual procedure is fairly complex, and we do not attempt to formalize it here.

3.2 Rational reason maintenance

Essentially all the choices made by traditional RMSs are irrational since they are made without reference to any preferential information about what choices are better than others. The most obvious decisions concern backtracking: whether observed conflicts warrant resolution and if so, which assumption (maximal or otherwise) to retract in order to resolve them. Approaches to each

of these decisions play prominent roles in the design of different reason maintenance systems. But if we are to achieve the efficiency required for revising large plans, reason maintenance must be redesigned to make these choices rationally whenever possible. Accordingly, we developed formal foundations for the theory of rational belief revision [9, 10]. But to really achieve efficiency, the RMS must do more than choose rationally among assumptions in backtracking. It must in addition be much more incremental than the original architectures, which were based on making unbounded (potentially global) optimizing computations that in some cases may reconsider the status of *every* item in the plan and knowledge base, even though very few of these statuses may change as the result of the revision. Put another way, the original systems maintain global coherence (propositions are believed if and only if there is a valid argument for them) and global groundedness (all believed propositions have a well-founded argument from premises). While these unbounded computations are manageable in relatively small knowledge bases, they are infeasible for use in systems manipulating very large plans. Instead of global computations, the RMS must control how much effort is spent on revision and trade off coherence and groundedness for time or other resources. Specifically, it must be able to decide whether the benefits of updating some arguments or consequences justify the costs of updating them.

3.3 Distributed reason maintenance

To make the RMS amenable to rational control, we divide the knowledge base into parts, called *locales*, each of which may be revised or preserved separately. Each locale of this *distributed* RMS contains its own set of beliefs and plans (as well as other information) corresponding to different elements and purposes of the overall plan or to different dimensions of structure (hierarchical abstraction, overlapping views, spatial separation, temporal separation, flow of material and information, etc.). Decomposition of knowledge in this way is a familiar element of many representational schemes (e.g., those based on Minsky's [21] original frame-systems idea). The use of locality in planning is illustrated most explicitly by the encapsulation mechanisms of Lansky's [18] GEMPLAN system.

Along with the general desire for incrementality, there are several additional reasons for distributing reason maintenance across different processors. In the first place, the information and effort required may be too great to store or perform on a single machine. In managing very large activities, for example, the most effective representations may spread information across machines or storage media of different speeds and access times (e.g., disk storage, large spatial separations). Even when the information resides on a single processor, the most convenient representation may be a modular, distributed organization as described above. But more generally, the information and actions involved in some task may be naturally distributed. For example, the necessary information may come from geographically separated sensors or databases. If communication is either unreliable or costly, effective action may require on-site processing. Similarly, there may be numerous people or devices carrying out parts of the task. For example, in the task of operating a large manufacturing complex, plans are executed by line or cell managers acting independently except as coordinated by the plan. When changes occur, at least some of the changes in plan must be determined by the line or cell managers, since the complex manager will not be able to keep track of all of the activities or to respond quickly enough. Because authority is delegated and distributed, reactions to deviations may be completely decentralized and uncoordinated.

In addition, distributed revisions may be valuable because different beliefs and plans may serve different purposes. These purposes may dictate careful maintenance of some beliefs and more casual maintenance of others. A common case of this arises when reasoning is accomplished by different subsystems operating at different rates. Even if they share a common database, it is often natural

to view each subsystem as having distinct inputs, outputs, and local state. In this setting, different rates of inference or action in the subsystems call for differing treatment of the information in computing updates and checking support in the locales used by the subsystems. For example, outputs which change rapidly compared with how often they are used as inputs need not demand reconsideration of consequences each time they change. Instead, it may be much more efficient to leave the consequences untouched and to have the consuming subsystem recheck the support immediately prior to use—and then only if the risks of unjustified action are great enough. In many cases, we may expect that the success of the overall plan will not be adversely affected if the beliefs in the locale of one subsystem about plans involving some distant subsystem are mistaken.

For example, suppose one part of a manufacturing plan calls for receiving parts from San Diego at Los Angeles and then flying them to Detroit. If local difficulties promise to delay the parts from San Diego, the origination portions of the plan might be revised to reroute similar parts in San Francisco to Los Angeles. As long as this plan patch attaches appropriate shipping orders for the Los Angeles authorities, there is no need to notify them in advance about the change in plans. Indeed, if the origination plans change several times (say from San Diego to San Francisco, back to San Diego, etc.), notifying Los Angeles in advance just leads to wasted effort in revising the latter portion of the plan.

3.4 The new conception of reason maintenance

Making reason maintenance rational and distributed in these ways has major implications for the relationship of the RMS to systems using it. The original RMS architectures make reason maintenance the base-level stratum upon which all other reasoning procedures are erected. To enable belief revision, one must encode every bit of information that might change in reasons and tell these reasons to the RMS (cf. [25, 28]). This can present an excessive burden, as manifest by the observation that the RMSs supplied in expert system shells all too often go unused. If one must apply it to every step of reasoning, at every level down to the smallest inference, reason maintenance becomes a demanding duty rather than a flexible service to use or ignore as appropriate. To integrate existing application tools and systems that do not use reason maintenance into AI systems that do, the RMS must be able to use other databases and processes to effect its revisions. In particular, the RMS must be able to treat external databases as the authorities about certain beliefs, and it must be able to operate even though other processes may be changing these databases independently of the RMS. This makes the RMS just one of a set of distributed databases.

In such a setting, the purpose of the RMS is to maintain a description of the overall system's state of belief that is as good as possible given the reasoner's purposes and resources. This description may be approximate, partial, or imperfect, and it may be improved by performing further computation as the resources supplied to the RMS increase. As with the original architectures, the RMS still provides explanations, a way of answering hypothetical questions, and a way of maintaining coherence, groundedness, and consistency (given enough resources and information), but its primary purpose is to enable the reuse of past computations in whole or in part without having to repeat the possibly lengthy searches that went into constructing their results. That is, we view reasons as information about past computations or conditions which may be used to reconstruct results in changed circumstances, either exactly or in modified form (as in derivational analogy [2] or case-based reasoning). Treating reasons as aids to recomputation is in marked contrast with the traditional use of reasons as rigid requirements that belief states must satisfy instead of as information which may be used or ignored as suits the reasoner's purposes. Naturally, in this setting the RMS is not expected to determine completely and accurately what the system believes. Instead, it only offers a theory of what the overall system believes—an "autoepistemic" theory, in the sense

of Moore [22], but not necessarily a complete or correct one.

Given this purpose, the basic operation of the RMS is to record reasons and other information, and, when so instructed, to revise beliefs in accordance with the expectations and preferences supplied by the reasoner. Put another way, the default operation of the RMS is to ignore the information it records until it is told to revise beliefs, and then to revise them only as far as can be justified by purposes of the reasoner. In the absence of more specific instructions, the default revision is trivial, simply adding the new reasons and their immediate conclusions to the belief set. Thus without explicit instructions, the RMS does not propagate changes, does not ensure beliefs are grounded, and does not automatically backtrack to remove inconsistencies. We do not require that all inference be rationally controlled. Some amount of automatic inference is acceptable if it represents strictly bounded amounts of processing.

To give some structure to RMS operations, we define revision instructions relative to the locales of the knowledge base. These instructions may indicate that changes should propagate within the locale containing the belief, or to its neighbors, or globally; or that all beliefs in the locale should be grounded with respect to the locale, with respect to its neighbors, or globally; or that backtracking should be confined to the locale, or should look further afield for assumptions to change.

Reasons ordinarily supply only partial information in that the reasoner need not register all inferences with the RMS. In the extreme case, the external reasoners may command the RMS to simply believe some proposition, independent of reasons. This corresponds to the "revision" operation in philosophical treatments of belief revision [14]. Because of this partiality, the RMS will sometimes be unable to track all the consequences of all beliefs. Although knowledge is usually preferable to ignorance, this incompleteness of the beliefs of the RMS need not be detrimental since the underlying knowledge and inferences of the reasoner are incomplete anyway. Moreover, these consequences may not influence the reasoner's actions, in which case all effort expended in recording them would be wasted. The only discipline required of the reasoner is that any inferences that will not be performed by some other agency and that cannot be determined after the fact during backtracking should be described to the RMS.

Correspondingly, reasons may be incorrect in the RMS. That is, the reasoner may use a reason to describe the result of a computation, but may leave out some underlying assumptions. The result is a reason that is valid when those unstated assumptions hold, but which may be invalid otherwise. Incorrect reasons can be very troublesome in the traditional architectures, since they would be enforced as requirements on the state of belief, but they need not cause special problems in the new conception. Since the RMS may obey or ignore reasons depending on its instructions and experience, all reasons are implicitly defeasible. Thus incorrect reasons pose no problems not already present in explicitly defeasible nonmonotonic reasons.

Just as reasons may be incomplete, so may be the theories of mental states constructed from them, since if reasons are ignored, their consequences will not be believed. More generally, the RMS makes it possible to vary how many conclusions are drawn from reasons. For example, the system will ordinarily use reasons to construct a single global set of beliefs, as in the original conception. But for some specific sets of reasons, say those corresponding to a circumscribed problem, the RMS may determine all consistent sets of beliefs as in the ATMS [3]. Alternatively, only some consistent interpretations may be constructed, such as those maximal in some order (as in preferential nonmonotonic logics [27]). In general, the aim is to use the recorded reasons to draw as many conclusions as the reasoner needs.

One consequence of the incompleteness and incorrectness of reasons is that beliefs of the system may be inconsistent in routine operation. The overall set of beliefs may exhibit inconsistencies by including conflicting beliefs from different locales. Ordinarily the specialized beliefs corresponding to specific problems or subjects will be represented in locales that are internally consistent, but the

RMS need not be forced to keep all these locales consistent with each other. But inconsistency can arise even within a locale if too little inference is specified.

Another consequence is that the beliefs of the system may not be fully grounded. In the first place, the set of beliefs may be so large as to make global groundedness too costly. More fundamentally, large sets of beliefs always contain interderivable sets of propositions—alternative definitions provide the most common example—and which of these sets to choose as axioms can depend on the specific reasoning task being addressed. For example, the standard definition of nonplanar graphs is best for some purposes (e.g., teaching the concept), but Kuratowski's characterization is best for other purposes (e.g., recognition algorithms). Thus lack of global groundedness need not be cause for alarm. Ordinarily, however, specialized locales corresponding to specific problems will be kept grounded in the axioms formulating these problems. The system of beliefs can thus be thought of as "islands" of groundedness floating in a sea of ungrounded beliefs.

Since reasons merely record some of the inferential history of the reasoner, they do not by themselves determine whether consequences are updated or supports are checked. Instead, to make these decisions the RMS uses annotations supplied by the reasoner which give instructions, expectations, and preferences about alternative courses of action. These include specification of the conditions under which the RMS should pursue consequences and check support. For example, local propagation may be expressed as processing changes within the locale containing the changed belief, but not externally. Alternatively, changes might be communicated to neighboring locales (with or without local propagation). Other regimes are possible too, including the extreme of propagating the change globally. Similarly, the annotations may indicate to persist in believing the proposition without reevaluating the supporting reason, to check that the reason is not invalidated by beliefs within the containing locale, or to check validity with respect to external beliefs. We have developed in [7] and [11] a formalization of this more general framework of reason maintenance that permits local variability of consequential import, groundedness, and other properties of RMS states.

It is this limited scope, variety, and fine grain of RMS operations, that makes RMS choices (which reasons to use in reconstructing results, whether to propagate changes, whether to ground a conclusion, and whether to backtrack) amenable to rational control. For decisions about updating consequences and checking support, it is important that the individual operations be well-characterized computationally. Domain knowledge of probabilities and preferences should also be reflected in the revision policies. Because such information is not always available, the architecture provides default choices for each of these classes of decisions. Each domain may override these with other defaults that are more appropriate in its specific area. These default choices are then used whenever there is no evidence that a decision requires special treatment.

4 Implementation concepts

We implemented a prototype of the new RMS in CLOS, the Common Lisp Object System. In this section, we describe the primary structures and operations of the implementation.

4.1 RMS

The RMS consists of a set of locales, a set of procedures for operating on locales, and a set of mechanisms for communicating with the external world. This organization is displayed in Figure 2. Some of the locales contained in the RMS are *persistent*, that is, constitute permanent (or at least long-term) representations evolving over time. But other locales are *ephemeral*, constructed

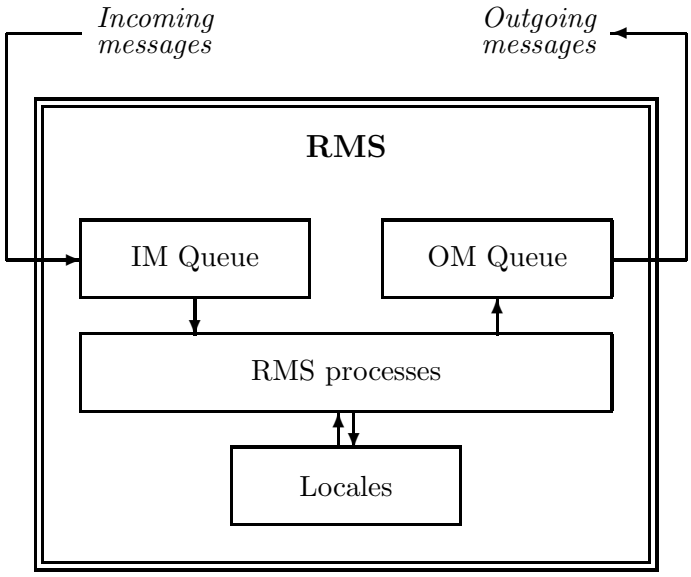

Figure 2: The organization of a reason maintenance server.

by the RMS during revision processes that expect to take some time to finish and seek to make no changes in the persistent structures until their computations are complete.

4.2 Locales

Locales, in turn, consist of sets of *nodes* and information about nodes, principally *reasons* and *stipulations*. Locales represent contexts of reasoning, so in principle they should be organized hierarchically, with some locales having sublocales, etc. In the present architecture, however, this hierarchy is flattened, as the communication protocols among locales is the same whether they are related to each other or not, so locales have no explicit sublocales. Figure 3 displays the organization of a locale. Currently the incoming message queues of locales are trivial, as since the actions are all fairly trivial (as explained below, they presently include only reason and stipulation creation and status change signals), there is no compelling reason not to just execute them directly. More general versions should reinstate the organization used in our original implementation, in which these queues are nontrivial agendas. With such an organization, the locales and RMS processing sites bear some resemblance to the LMC architecture developed to good effect by Pollack *et al.* [24].

4.3 Nodes

Nodes consist of both information about the relations between the node and external objects, locales, and other nodes, and information about this relational information itself.

The first purpose of a node is to denote some object, in most cases external to the the locale and the RMS. This object is called the *original* or *referent* of the node.

The node also records three lists of relations holding between the node and other nodes. The

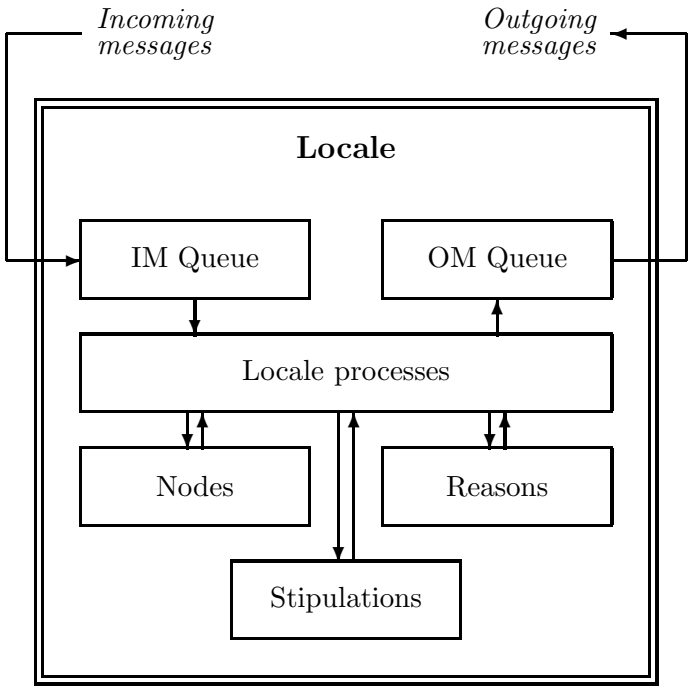

Figure 3: The operational structure of RMS locales.

list of the node's *dependencies* indicates a set (possibly empty) of dependencies that depend on the node in some way; the list of the node's *determiners* indicates relations that potentially determine the status of the node; and the node's *determiner*, if the node is *in*, indicates the element of the list of determiners that the RMS identifies as determining the status of the node. If the node is *out*, there is no identified determiner.

The node has two binary flags which indicate both whether its referent is included in the current view provided by the locale, and whether this inclusion is correct or possibly incorrect. The first flag is the *support label*, which is either *in* or *out*, meaning that the referent is either in or out of the current view provided by the node's locale. The second flag is the *label status*, which is either *certain* or *uncertain*, meaning that the support label correctly reflects the support provided by the determiners or may not correctly reflect the support provided by the determiners. In the original RMS, of course, the support label was *in* if and only if the node had a valid determiner, but in the new RMS, the possibility of incomplete updates means that a node can be *in* even if no determiner is valid, and can be be *out* even if some determiner is valid. (It is worth noting that even the original RMS employed the label status flag, but only during revision computations; control was passed back to the external system using the RMS only after all support labels had been determined with certainty.)

One other binary flag is used to mark those nodes that should not be included in current views. Nodes so marked are called *contradictions* after the familiar logical notion of a statement that should not be held. This nomenclature is somewhat misleading in the RMS, however, as nodes so marked need not even represent beliefs; instead, they are simply things that should not be made part of the view represented by a locale.

The RMS also provides for one important type of node, namely *propositions*. These may be used to stand for logical propositions, and differ from other nodes only in that they have an associated *negation*, which is another node (in fact, also a proposition) representing the negation

of the proposition node.

4.4 Reasons and Stipulations

We use the term *dependency* to mean any records stored by the RMS of relationships holding among nodes. These include the records stored with a node as the node's dependencies and determiners. In their most general form, dependencies are nondirectional and nonspecific; they serve mainly as an indication of relevance rather than as specific directives to action. The implementation of the RMS provides a hierarchy of subtypes of dependencies, as detailed below, but makes substantial use of only two subtypes at present: *support-list* reasons, which indicate nonmonotonic argument or derivational steps, and *stipulations*, which indicate observations or communications from outside the node's locale.

There are two main subtypes of dependencies: *determiners* and *influences*. A determiner is a dependency that determines status labels of one or more nodes, while an influence is somewhat more general and indicates that a set of *antecedent* nodes influence a set of *consequence* nodes in some way. Neither of these main subtypes is used directly; instead, the RMS really exploits more special subtypes of each.

The subtypes of determiners provided by the RMS are as follows:

- A *node-determiner* is a dependency that determines a single node's status.
- A *reason* is a node determiner that provides a reason for the node. There are several types of reasons. Each reason has a defeating node, which if *in* makes the reason invalid.
- A *SL-reason* (or support-list reason) is a reason that expresses support in terms of two lists of nodes, an *inlist* and an *outlist*, and is valid just in case each node of the inlist is *in* and each node of the outlist is *out*.
- A *CP-reason* (or conditional-proof reason) is a reason that expresses support in terms of the derivability or arguability of one node from others, as in a conditional proof or hypothetical. The current RMS implementation permits these to be defined, but does not process them at present, as they significantly increase the complexity of the algorithms.
- A *stipulation* is a node-determiner that simply stipulates the label of a node, providing no reason.
- A *clause* is a determiner indicating that at least one member of a list of proposition nodes must be *in*. This is intended to be used roughly in the ways one uses disjunctive clauses in logical reasoning.

As noted above, the prototype implementation concentrates on SL-reasons and stipulations, and provides only incomplete handling for the other types of determiners.

The main form of influences used in the RMS at present are *signals*, which effect communications among locales within an RMS, and communication between an RMS and external systems. The RMS and systems using the RMS associate with each type of signal a procedures that performs the appropriate signalling action (possibly conditional, possibly null) when invoked by the RMS. The subtypes of signals provided by the RMS are as follows:

- *Tell* signals are invoked by the RMS whenever new information is provided to the RMS. At present, two specific sorts of tell signals are defined. These indicate the addition of new determiner information, namely the addition of a new stipulation or a new reason.

- *Ask* signals are invoked by the RMS whenever information is asked of the RMS. The present implementation provides no specific uses of these signals.

- The signals most exploited in the current implementation are *relabeling* signals, which are used, as in the original RMS, to signal the changes in node information following the execution of relabeling procedures. The principal specific types signal changes in a node's support label: *recalling* signals indicate a change from *out* to *in*, while *forgetting* signals indicate a change from *in* to *out*. The RMS also provides a more general signal of *conditions*. Each condition signal includes a condition (expressed as a Lisp predicate) that determines the need to execute the signal.

As noted above, at present the main signals exploited in the RMS are the recalling and forgetting signals.

We define a number of relationships among nodes in terms of their support labels and their dependencies. The basic relations are those of *antecedents*, *supporting nodes*, *consequences*, and *affected consequences*, which are defined appropriately for each specific type of dependency. The antecedents, as one might expect, are all the nodes directly entering into the evaluation of the dependency, and the consequences are the nodes directly affected by a determiner or influence. The supporting nodes are the antecedents that actually determine the validity of the node, given the current support labelings, and the affected consequences are the consequences for which the node in question is a supporting node (i.e., enters into its determiner). The other relationships are defined in mainly in terms of these basic ones. The *believed consequences* (admittedly a poor name) consist of the affected consequences that are *in*; the *repercussions* are found by taking the transitive closure of the believed consequences; the *foundations* are found by taking the transitive closure of the antecedents; and the *ancestors* are found as the transitive closure of the supporting nodes.

We also distinguish two types of nodes based on their determiners. We say that *premises* are nodes that are *in* with no antecedents, and that *assumptions* are nodes that are *in* and have supporting nodes that are *out*. Ordinarily, we ignore the reason-specific defeater nodes included in reasons in making these distinctions. From these two distinctions, we may further identify the premises and assumptions supporting a node, in the sense of appearing in the node's ancestors.

Finally, as an aid to determining the currency of information, each dependency is marked with the time it was told to the RMS. Further, each node is marked both with the time its label was last checked and with the time of its latest determiner, so that the label status of a node updated prior to its most recent determiner is *prima facie* uncertain. Similarly, locales are marked with the time of their last update, and with the time of the most recent determiners provided to any of their nodes.

4.5 RMS procedures

The basic locale procedure is one which relabels a set of nodes; the wider variety of locale procedures consist primarily in different ways of selecting different sets of nodes for relabeling.

The main steps of the relabeling procedure are as follows:

- First, identify the nodes to be relabeled and create an ephemeral locale mirroring them.

- Second, find all support labels determined by stipulations.

- Third, find all support labels determined by reasons.

- Fourth, install all these support labels and the corresponding determiners.

- Fifth, execute all relabeling signals for these nodes.

We describe each of these steps in turn, starting with the second, and describe the first step last.

The method by which labels are determined from stipulations may vary from node to node. In general, one should use the most recent stipulations, but these may conflict. Each node may indicate a different function for reconciling its stipulations in cases in which they conflict, but so that nodes need not always indicate such a function, each locale may indicate a function for reconciling stipulations for its nodes. If neither the node nor its locale indicate such functions, the RMS default method is used, which is just to use the first stipulation recorded among the stipulations with the most recent dates. (We allow that several different pieces of information may arrive at the same time, and do not assume that the RMS has a general means to serialize these incoming messages.) An alternative method provided is more conservative, and simply retains the current support label when the latest stipulations conflict in the labels they specify.

The method by which labels are determined from reasons is akin to traditional methods for reason maintenance; a search for admissible labelings using the reasons as constraints on the labels assigned to nodes. The current implementation uses a very simple search derived from the algorithm used by the original RMS.

The prototype RMS installs the results of its revisions by simply setting the support label and determiners of the nodes it has examined to the newly determined values. One can contemplate more refined methods for the distributed asynchronous environment in which the RMS first checks to see if any of these nodes have received new labels or information in the intervening time, and then treats these nodes differently, but such more refined methods remain to be studied.

The RMS executes the relabeling signals by evaluating the conditions appropriate to all signals attached to nodes under revision and then executing the associated signalling procedures for all signals with valid conditions.

As stated previously, one of the main aims for the new RMS is to avoid the approach taken in the original RMS and to make more informed decisions about whether new information calls for revisions, and the extent of these revisions if so. The new RMS makes first steps towards addressing this aim by providing an array of revision procedures as alternatives to the revision-extent decision. These alternative revision procedures are as follows:

- The simplest procedure revises a single node. This amounts to simply reconsidering its current set of determiners and resetting it support label to the labeling justified by these determiners. The signals attached to the node, if any, are then executed. This simplest procedure takes little time, and often is all that is needed, especially for providing an initial labeling for newly created nodes receiving their first reason or stipulation. However, if used to the exclusion of more comprehensive methods, it produces locales with somewhat incoherent labelings.

- A similar procedure revises a given set of nodes. The only difference is that in this case the ephemeral locale contains the given set rather than just one node.

- The next simplest procedure revises an entire locale. This is done by giving all the nodes of the locale to the node-set relabling procedure just described.

- The original RMS relabled only the affected consequences of single nodes, and the new RMS provides corresponding capabilities in procedures that revise either the immediate, the local, or the global consequences of sets of nodes. In the immediate procedure, the RMS collects up only the immediate consequences of the initial nodes; in the local procedures, only the affected consequences that exist within the same locales as the initial nodes; and in the global procedures, the RMS collects up all affected consequences, chaining through all locales if necessary.

One of the more interesting avenues for extension of the RMS involves developing incremental versions of such procedures that make informed node-by-node decisions about whether to pursue consequences further. This would provide a large spectrum of possible consequential revision operations instead of the three provided in the prototype implementation.

- Since the original RMS always presented well-founded arguments for all labelings, it only needed to revise the antecedents of nodes when contradictions were discovered that required removing some of the grounds for holding a contradiction node. But in the new approach, in which labelings need not be completely coherent, it is necessary to provide means by which the support for a node can be verified prior to using the node to take significant actions. Accordingly, the final class of relabeling procedures provided by the new RMS revise either the immediate, local, or global antecedents of an initial set of nodes, with these variations involving distinctions exactly like the immediate, local, and global consequences just discussed.

In addition to relabeling or update procedures, the RMS provides a variety of other procedures as follows:

- The *tell* procedures provide means for providing information to the RMS. These procedures may be used to create new locales, nodes, reasons, or stipulations, and to mark nodes as contradictions.
- The *ask* procedures provide means for obtaining information from the RMS. These procedures may be used to retrieve the support label, label status, determiner, and the nodes standing in any of the derived relationships (antecedents, consequences, etc.) to a given node.
- The *backtracking* procedures provide the RMS the ability to attempt to make specific contradiction nodes *out*.

5 Market-guided reason maintenance

The new RMS described above provides the means for making partial revisions to information and views distributed throughout interconnected locales, but does not itself provide guidance about which revisions to effect. In this section, we discuss the approach of using artificial markets to allocate effort for distributed revisions and replanning, and extend the basic RMS to a market-guided RMS.

5.1 Market allocation of resources

Efficient allocation of resources constitutes an important special case of both result and process rationality. In transportation planning, for example, efficient allocation of transportation resources to transportation tasks constitutes result rationality, while efficient allocation of planning resources to solving competing transportation problems constitutes process rationality. Such competition between planning problems occurs frequently. Large-scale activities usually involve several organizations or authorities, each placing demands on its own and other resources, and these competing organizations often generate internal competitions as well, as they distribute information and authority, both geographically (different theaters of operation or manufacturing or clinic locations) and functionally (special-purpose databases and departments), in order to satisfy legal, regulatory, privacy, reliability/redundancy, or efficiency (e.g., communication or computational) considerations.

The overall success of large-scale planning activities often requires managing these competing demands effectively to coordinate the component efforts, and recognizing that computational and communication resources must be allocated along with fuel, diagnostic equipment, repair personnel, and other more visible goods. Computation and communication may seem "free" once one makes these resources available, but their use always involves opportunity costs. For example, computing more to improve one partial plan may save a flight and its associated fuel and pilot costs, while computing more to improve another partial plan may save more lives. But one cannot simply just decide what computations to make by comparing the non-computational goods at issue because some computations (e.g., creating a fast local database) can speed or improve other computations, thus representing non-computational goods only indirectly.

The usual approach taken to managing these competitions is bureaucratic, with some central or higher authorities attempting to make broad allocations of resources and requiring the different organizations and their components to operate within these allocations and to explicitly request reallocations if such operation becomes impossible. Unfortunately, bureaucratic management of the resources for complex activities is too inflexible to meet the ordinary flow of events, which continually undermines the assumptions underlying such allocations by changing the demands on component organizations in ways difficult to predict by central planners. In particular, the most important competition for resources is not a static competition among relatively permanent organizations and their components: instead, the most important competition for resources is among an ever-changing set of tasks and portions of the overall activity. This competition arises as changing circumstances undermines portions of the plans of the organizations involved in the activity. To deal with these changing circumstances, the organizations and their components must continually repair the plans directing their current activities, and the result is that the most important competition for resources is the competition among portions of the overall plan for the attentions of the authorities engaged in the activity. This competition is dynamic, cutting across organizational lines and changing much more rapidly than the structures of the organizations themselves (even when the organizations dynamically form teams to deal with problems). Bureaucratic management of resources does not appear adequate to deal with these dynamic resource competitions.

To manage large-scale planning activities and their dynamic resource competitions effectively, we must seek to address the competitions among tasks directly rather than indirectly through a fixed matrix of organizations. Instead of focussing on a fixed array of organizations and their capabilities, we must focus on the changing set of tasks, the importance of these tasks to each other (as accomplishing one task may facilitate or impede another), and the relevance of the organizational capabilities to each task. The question then becomes how to allocate resources to tasks, with the allocation of resources to organizations derived from the primary task-centered allocation.

The field of economics has a well-known answer to the problem of allocating resources to tasks: use markets. It suggests viewing the competition for resources in terms of resource-endowed consumers that represent tasks or goals of the activity and resource-transforming producers that represent computational or reasoning procedures, informational resources, or noncomputational agents. Equilibrium ("market") prices correspond to allocations of resources to tasks that balance supply and demand, thereby ranking particular resource demands by their importance relative to other needs and the activity's ability to supply or produce the resources.

Economics bases its market prescription on practical, theoretical, and organizational grounds. In practice, markets for many goods provide extremely rapid response to changing circumstances (as observation of commodity markets shows) as well as flexible and measured responses to changing demands (which is not to say that overreactions are not possible). For a sizable range of important goods, markets empirically provide the best known method of allocation, even to the point of springing up and supplanting more bureaucratic mechanisms when not effectively suppressed. In

addition, economic theory proves markets to be the most efficient allocation mechanisms possible, in the sense that the allocations determined by markets best satisfy the preferences of the agents involved. And markets do this while making smaller demands on the organizational structure of the activities using the resources, as they depend mainly on the calculations of the individual agents or organizational components competing for resources rather than on any calculation by central planning authorities.

Markets thus seem highly suited to complex resource allocation problems, as they do not require synoptic computational abilities and still achieve the best possible results. But their use in organizing activities has been limited for several reasons, especially the difficulty of recognizing competitions in a timely fashion and of tracking the wide variety of material and non-material goods involved in some activities. That is, markets in diesel fuel persist and succeed because diesel fuel is a commodity used by many agents over long periods of time, has very slowly changing or highly predictable properties, and is arbitrarily divisible, with different batches distinguished only by their size, not by other qualities. In contrast, it is difficult, at least within traditional organizational structures, to recognize the need for a market in strategies to repair a transportation plan upon the closure of an airfield, much less to identify the organizational agents that might appropriately enter into such a market. In consequence, traditional approaches to organizing large scale activities mix market and other mechanisms. The activities employ markets for resources when those markets already exist and are conveniently accessed by the component organizations, but use bureaucratic or other non-market mechanisms for the structuring the larger part of the activity. Traditional approaches thus are denied the benefits of market allocation for many parts of their activities.

Although virtually all traditional architectures for computational and intelligent systems have been based on the bureaucratic model, limited forms of computational markets go back a long way in computer science; one can even view some of the early job-scheduling procedures for batch processing as implementing a "market" in computation for which users "bid" by means of commands on their job-control cards (punched cards, that is). But the true flowering of computational markets has occurred only recently, mainly with the WALRAS computational economy developed by Wellman [30]. Designed to make use of the main ideas of theoretical economics about markets, WALRAS provides a general mechanism for implementing markets in arbitrary sets of goods, traded by arbitrary consuming and producing agents. WALRAS provides the basic market notions of consumer and producer agents, goods, and auctions for these goods. Each consumer and producer enters into a subset of the auctions. Consumers are endowed (possess) bundles of various goods, and enter into the auctions for each of the goods in their endowment. Producers do not possess goods, but instead transform a set of input goods into an output good, and enter into the auctions for their input and output goods. Consumers and producers place bids in each of the auctions in which they participate. These bids may be as simple as indicating the desire to trade a specific quantity of the good at the current price; or as complex as a full schedule of amounts to trade (buying or selling) at each possible price. Consumers and producers determine their bids in different ways. Consumers have a utility function or preference order over possible bundles of goods, and bid so as to trade a less preferred bundle for a more preferred bundle. Producers have a production function that describes how much output derives from each bundle of inputs, and use this production function to determine bids for input goods from prices for output goods or vice versa. WALRAS determines equilibrium prices by an iterative procedure that at each iteration determines the total supply and demand for each good at current prices and adjusts the prices accordingly if supply and demand do not match. In observations, WALRAS computes equilibrium or near-equilibrium prices in only a few iterations. WALRAS makes a number of special economic assumptions about agents, the most important one being that agents are "price taking", that is, no

agent is large enough in the market to influence prices simply by its own actions. This assumption is actually not true in many of the small reasoning markets of interest, but its failure has not yet been shown to matter in practice due to the iterative nature of equilibriation. WALRAS thus provides a mechanism for implementing markets in goods as they appear and disappear, as long as one can identify the need for such markets and the participants in them.

5.2 An experimental market-guided RMS

In order to provide rational guidance to distributed reason maintenance activities, we constructed a market-guided RMS, called MRMS, by combining the new RMS with an extension of WALRAS. MRMS represents each significant revision task by means of a market good and a consumer of that good, and each revision method by a producer of a revision good. Effort is then allocated among these tasks on the basis of relative prices of these goods.

The "glue" used in this combination is a system called RECON, the Reasoning ECONomy. The reasoning economy extends WALRAS and determines rational allocations of the full range of resources, computational and otherwise, by determining prices or trading ratios among resources. RECON builds on WALRAS by augmenting WALRAS's generic notions of goods, consumers, or producers with notions specific to reasoning tasks, and by introducing computation goods to allocate to different tasks. Its main additions to WALRAS are a good representing computation, the notions of computation consumers and producers, a taxonomy of action types, and an execution mechanism that takes actions in an order determined by bids for computation.

The taxonomy of action types introduces the distinction between actions and states, and differentiates a standard variety of action types reflecting mainly planning and reason maintenance actions (labeling, signalling, etc.). Base level actions have associated procedures.

We have explored two approaches to allocating computational resources. Our initial approach was based on an auction of computation opportunities, while a later approach developed by our student Nathaniel Bogan [1] conducts auctions to rent computational resources to the highest bidders.

The initial RECON implementation scheduled tasks on a single processor by auctioning off opportunities to compute. In this implementation, computation consumers and producers bid for amounts of the computation good as though they were bidding for different amounts of time. Computation consumers have a computation function in addition to their utility function, where their computation function indicates their use of computation in different circumstances. Computation producers take computation as input and produce outputs, with different subtypes of computation producers characterized by their computation production functions that indicate how much computation they take to produce their results (i.e., their running time). WALRAS is then run several steps towards equilibrium prices, with the resulting non-equilibrium prices used both because equilibria need not exist with the discrete bidding schedules used in MRMS and because finding them may take too long even when they do exist. The RECON execution mechanism schedules the execution of primitive procedures by using the bids each procedure's producer bids for computation. However, though the bids reflect the amounts of computation desired by consumers and producers, the execution mechanism in fact provides uninterrupted opportunities to compute as long as desired (possibly forever) rather than the opportunity to compute only as much as the bid specified. The selection of the producer to execute is made by one of several methods, the simplest being to simply pick the producer bidding for the maximum amount of computation among all producers bidding for computation. This corresponds intuitively to taking actions to make as much progress as possible at each step. Following each action, producers and the consumers they serve are charged for the computation consumed once a producer's procedure is executed. Due to

the use of non-equilibrium prices, these charges may bankrupt consumers. On the theory that most reasoning consumers correspond to goals that need not be pursued further once achieved, RECON seeks to speed future decisions by "garbage collecting" bankrupt or otherwise useless consumers and producers from the economy when these charges remove all the endowment of consumers.

Nathaniel Bogan developed an improved version of RECON by incorporating several standard approaches from economic theory and practice [1]. In his scheme, consumers and producers bid rental prices for computational resources (e.g., processors), and RECON allocates time to those producers offering the highest rental rates (in the non-equilibrium approximation). The winner is then charged at this rate according to how much time it actually uses.

We developed MRMS on the basis of the initial RECON version described above. Fairly simple estimates of costs were used instead of more accurate (and possibly statistical) analyses in describing production functions. We then tested MRMS in a simple problem-solving context using only simple all-or-nothing preferences for the tasks. Even this simple setting revealed interesting behavior. Varying the amount of computation required by the basic reason maintenance procedures relative to each other yielded series of reasoned updates either being performed sequentially as individual revisions (which take little time but may leave the network of justifications somewhat incoherent), or being performed indirectly by avoiding any revision during the incoming series but revising the entire set at once in a larger update operation at the end (which can take more time but leaves the network of justifications mutually coherent). This illustrates one of the strenths of the market-oriented approach: tasks which have implications for many other tasks are automatically accorded great importance, in some cases even more than tasks that are very important on their own. These determinations derive purely from the information local to each task and method and from the pattern of interaction of the tasks and methods as reflected in the market topology. There is no need to apply specific knowledge to recognize the importance; the market mechanisms provide that indication automatically.

6 Representing planning and reasoning preferences

Preferences constitute one of the basic elements of information about economic agents, and a key problem in both our investigation of market-guided reason maintenance and in rational planning in general is finding a good representation of preference information. Decision-theoretic treatments of preferences represent the objectives of a decision maker by an ordering over the possible outcomes of available plans. We view this ordering relation as an ideal, but cannot hope to completely and directly encode it in a planning system, as the domain of outcomes is combinatorially large or infinite, and the relevant preference criteria vary across problem instances. Therefore, in designing preference languages, we seek constructs for describing general *patterns* of preference that hold over *classes* of outcomes and situations. Toward this end, we have developed a qualitative logic of preference *ceteris paribus* or preference "other things equal" [12, 13, 31] in collaboration with Michael Wellman. This logic affords flexible specification of objectives, underpinned by a decision-theoretic semantics.

The most common representation used or assumed for preference information is that of numerical utility functions. These have the advantage of offering complete comparisions of the relative desirability of all alternatives and of applying numerical optimization procedures in decision making. But they also suffer from severe problems of convenience and generality, especially when attempting to integrate them into the automatic reasoning and planning system developed in the artificial intelligence literature. In the first place, numeric utility functions are too specific: the foundations of decision theory and economics start with qualitative preference orders, and many

different utility functions can represent the same preference order [26]. Thus a utility function may convey more information than is really there. In the second place, while utility functions make some optimization procedures convenient, they tend to make specification and elucidation of preferences difficult. Most people exhibit a well-known aversion to expressing their judgments in numerical terms, yet are often willing to provide qualitative expressions of preference with confidence. Working directly with utility functions thus makes people hesitant to supply the necessary information, and also provides no way to make use of the qualitative comparisions that informants might provide immediately. In the third place, utility functions provide no connection with the notion of *goal* upon which most reasoning and planning systems in use in artificial intelligence are based. Most of these systems solve problems and construct plans in response to stipulations of one or more propositions representing the goals to be achieved. Decision theory has no notion of "things to be achieved", only rankings of the relative desirability of outcomes, while goal-based systems frequently have only the notion of "things to be achieved" and no way of comparing the relative desirability of potential solutions or plans.

In response to these problems with utility functions, we worked toward finding ways of representing information about preferences that permits expression of both standard decision-theoretic information as well as standard goal-based specifications, so as to be able to exploit the strengths of both decision theory and artificial intelligence. More generally, we aimed to find preference representations capable of encoding and using whatever preference information is available, without having to wait until enough is specified to determine a utility function.

Our qualitative logic of preference *ceteris paribus* provides a uniform language in which one can express both ordinary decision-theoretic preferences as well as the standard notion of goal, which we interpret to mean conditions preferred to their opposites other things equal. It is easy to show that one cannot reasonably interpret goals as conditions preferred to their opposites without qualification, as that interpretation trivializes planning with multiple independent goals. In particular, that interpretation forces all states that satisfy some goals but not others to be indifferent; so that one cannot add any information to the goals to state that, for example, outcomes satisfying 5 goals are preferable to outcomes satisfying only 4. To avoid such trivialization, we relativize the comparisons to outcomes that keep everything the same except the conditions being compared. We write $p \trianglerighteq q$ to mean that p is weakly preferred to q *ceteris paribus*, and $p \rhd q$ to mean that p is strictly preferred to q *ceteris paribus*, that is, that $p \trianglerighteq q$ but not $q \trianglerighteq p$.

With the notion of preference relativized in this way, we can define goals simply as propositions preferred to their complements *ceteris paribus*. We write $\trianglerighteq(p)$, therefore, to mean that p is a weak goal, that is, that $p \trianglerighteq \bar{p}$, and write $\rhd(p)$ to mean that p is a strict goal, that is, that $p \rhd \bar{p}$.

This formal semantics for goals permits us to investigate principles for reasoning with goals. For example, some forms of reasoning decompose goals into sets of logically related propositions (for example, sets of propositions yielding the original one by conjunction or disjunction, as in subgoaling) and treat these new propositions as new goals. We may ask whether such operations are sound with respect to our semantics. In fact, the semantics confirms that these operations are not always valid. Decomposing goals into conjunctions and disjunctions of putative goals need not always produce *bona fide* goals because the decomposition propositions may have undesirable properties ("side-effects") in addition to their relation to the compound goal. In general, complex goals tell us little about the desirability of their constituent parts. We do not mean to suggest that reasoners stop using useful manipulations like conjunctive and disjunctive decompositions of goals just because these operations can be unsound. We only mean to point out that if a reasoner uses unsound operations, then either it risks exhibiting judgements through its actions that conflict with its represented preferences, or its reasoning introduces new assumptions that change the underlying preference order. These possibilities deserve explicit recognition, and explicit treatment in some

cases.

More importantly, the semantics entails various important principles for reasoning about goals and preferences. We identify some important cases in which preference *ceteris paribus* satisfies a dominance principle, in that if p and q concern aspects of outcomes "orthogonal" to r (a concept made precise by the semantics), then $p \trianglerighteq q$ whenever $pr \trianglerighteq qr$ and $p\bar{r} \trianglerighteq q\bar{r}$. We also relativize the notion of logical independence of propositions to the notion of *independence ceteris paribus*, and show that for propositions that are orthogonal and independent *ceteris paribus*, conjunctions and disjunctions of goals are themselves goals. For such propositions, we also derive a variety of transitivity results; for example, propositions preferred to goals are also goals. It appears that many important decisions can be made simply on the basis of dominance arguments expressed completely qualitatively in this form, so this approach should permit important decision-making and planning to proceed even without numeric utilities. Wellman [29] has developed the probabilistic version of this idea into a dominance-guided planning procedure that plans "up to tradeoffs" in his terminology, meaning it constructs plans that are optimal as far as the qualitative probabilistic information is concerned, and we expect an even more robust planning procedure could be constructed in a similar fashion using only qualitative preference information.

We are continuing to develop the theoretical structure and inferential capabilities of this logic and some close variants, but the basic language already provides a useful tool for encoding qualitative preferential information. Of course, a rich language for encoding preference information would include quantitative representations as well, and we are working toward a preference language that spans the spectrum from completely qualitative representations like our language of comparative preference to ordinary numeric utility functions, including intermediate representations of multiattribute utility functions such as subutility composition trees [32], the standard forms of multiattribute utility functions [17], and their application to expressing different types of planning goals [15, 16].

7 Conclusion

Reason maintenance offers important abilities for use in planning and replanning, but to prove useful for large-scale activities, the techniques must be capable of incremental application that does not incur the costs of global reconsideration. We extended traditional reason maintenance techniques to make use of instructions, expectations, preferences, and market mechanisms in deciding how to establish and revise beliefs and plan elements. In our conception, the *rational distributed reason maintenance service* maintains only as much coherence and grounded support as is called for by the planner's purposes. In essence, the fundamental operations of finding supporting arguments and pursuing consequences become flexible rather than routine, with different sorts of reasons indicating different sorts of processing during revisions in addition to their more overt indications of likelihood and preference information.

Acknowledgments

Much of this work, and especially the work on representing preferences, was conducted in collaboration with Michael Wellman, who coauthored some of the papers on which this report draws, and who originated the WALRAS system. This research was supported by ARPA and Rome Laboratory under contract F30602-91-C-0018.

References

[1] N. R. Bogan. Economic allocation of computation time with computation markets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1993. Thesis Proposal.

[2] J. G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 371–392. Morgan Kaufmann, 1986.

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

[4] J. de Kleer, J. Doyle, G. L. Steele Jr., and G. J. Sussman. AMORD: Explicit control of reasoning. In *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages*, pages 116–125, 1977.

[5] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(2):231–272, 1979.

[6] J. Doyle. The ins and outs of reason maintenance. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 349–351, 1983.

[7] J. Doyle. Some theories of reasoned assumptions: An essay in rational psychology. Technical Report 83-125, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1983.

[8] J. Doyle. Reasoned assumptions and Pareto optimality. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 87–90, 1985.

[9] J. Doyle. Artificial intelligence and rational self-government. Technical Report CS-88-124, Carnegie-Mellon University Computer Science Department, 1988.

[10] J. Doyle. Rational belief revision (preliminary report). In R. E. Fikes and E. Sandewall, editors, *Proceedings of the Second Conference on Principles of Knowledge Representation and Reasoning*, pages 163–174, San Mateo, CA, 1991. Morgan Kaufmann.

[11] J. Doyle. Reasoned assumptions and rational psychology. *Fundamenta Informaticae*, 20(1-3):35–73, 1994.

[12] J. Doyle, Y. Shoham, and M. P. Wellman. A logic of relative desire (preliminary report). In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, 6*, volume 542 of *Lecture Notes in Artificial Intelligence*, pages 16–31, Berlin, Oct. 1991. Springer-Verlag.

[13] J. Doyle and M. P. Wellman. Representing preferences as *ceteris paribus* comparatives. In S. Hanks, S. Russell, and M. P. Wellman, editors, *Proceedings of the AAAI Spring Symposium on Decision-Theoretic Planning*, 1994.

[14] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, MA, 1988.

[15] P. Haddawy and S. Hanks. Issues in decision-theoretic planning: Symbolic goals and numeric utilities. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 48–58, 1990.

[16] P. Haddawy and S. Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 71–82, San Mateo, CA, 1992. Morgan Kaufmann.

[17] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley and Sons, New York, 1976.

[18] A. L. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4:319–340, 1988.

[19] D. McAllester. Truth maintenance. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 2, pages 1109–1116, Menlo Park, CA, 1990. AAAI Press.

[20] D. McDermott and J. Doyle. Non-monotonic logic—I. *Artificial Intelligence*, 13:41–72, 1980.

[21] M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, chapter 6, pages 211–277. McGraw-Hill, 1975.

[22] R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[23] B. Nebel. *Representation and Reasoning in Hybrid Representation Systems*. Number 422 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 1990.

[24] M. E. Pollack, T. Znati, E. Ephrati, D. Joslin, S. Lauzac, A. Nunes, N. Onder, Y. Ronen, and S. Ur. The DIPART project: A status report. In *Technical Papers of the ARPA Planning Initiative Workshop*, 1994.

[25] C. Rich. The layered architecture of a system for reasoning about programs. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[26] L. J. Savage. *The Foundations of Statistics*. Dover Publications, New York, second edition, 1972.

[27] Y. Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, Cambridge, MA, 1988.

[28] M. B. Vilain. The restricted language architecture of a hybrid representation system. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 547–551, 1985.

[29] M. P. Wellman. *Formulation of Tradeoffs in Planning Under Uncertainty*. Pitman and Morgan Kaufmann, 1990.

[30] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[31] M. P. Wellman and J. Doyle. Preferential semantics for goals. In *Proceedings of the National Conference on Artificial Intelligence*, pages 698–703, 1991.

[32] M. P. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proceedings of the First International Conference on AI Planning Systems*, 1992.

A List of personnel

The following people were supported by contract F30602-91-C-0018.

1. Jon Doyle, principal investigator
2. Michael Frank, graduate student
3. Tze-Yun Leong, graduate student
4. Vijay Balasubramanian, graduate student
5. Ronald J. Bodkin, graduate student
6. Nathaniel R. Bogan, graduate student
7. Russell Manning, graduate student
8. Whitney Winston, undergraduate student
9. Annette Ellis, secretary
10. Scott Reischmann, secretary

B List of contract publications

The following publications were produced either with support from contract F30602-91-C-0018 or in initiating the project.

1. Ronald J. Bodkin. Extending computational game theory: Simultaneity, multiple agents, chance and metareasoning. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1992.

2. Nathaniel R. Bogan. Economic allocation of computation time with computation markets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1994.

3. Jon Doyle. Rational belief revision (preliminary report). In R. E. Fikes and E. Sandewall, editors, Proceedings of the Second Conference on Principles of Knowledge Representation and Reasoning, pages 163–174, San Mateo, CA, 1991. Morgan Kaufmann.

4. Jon Doyle. Rationality and its roles in reasoning. Computational Intelligence, 8(2):376–409, 1992.

5. Jon Doyle. Reason maintenance and belief revision: Foundations vs. coherence theories. In P. Gardenfors, editor, Belief Revision, pages 29–51. Cambridge University Press, Cambridge, 1992.

6. Jon Doyle. Reasoned assumptions and rational psychology. Fundamenta Informaticae, 20, 1994.

7. Jon Doyle. Inference and acceptance: comments on Kyburg's "Believing on the basis of the evidence". Computational Intelligence, 10(1):46-48, 1994.

8. Jon Doyle, Yoav Shoham, and Michael P. Wellman. A logic of relative desire (preliminary report). In Z. W. Ras and M. Zemankova, editors, Methodologies for Intelligent Systems, 6, volume 542 of Lecture Notes in Artificial Intelligence, pages 16–31, Berlin, Oct. 1991. Springer-Verlag.

9. Jon Doyle and Michael P. Wellman. Rational distributed reason maintenance for planning and replanning of large-scale activities. In K. Sycara, editor, Proceedings of the DARPA Workshop on Planning and Scheduling, pages 28–36, San Mateo, CA, Nov. 1990. Morgan Kaufmann.

10. Jon Doyle and Michael P. Wellman. Impediments to universal preference-based default theories. Artificial Intelligence, 49(1-3):97–128, May 1991.

11. Michael P. Frank. Advances in decision-theoretic AI: Limited rationality and abstract search, Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1994.

12. T. Y. Leong. Toward A General Dynamic Decision Modeling Language: An Integrated Framework for Planning Under Uncertainty, AAAI Spring Symposium 1994 Working Notes for Decision-Theoretic Planning.

13. S. Yeh and T. Y. Leong. Automatic Generation of Transition Probabilities in Dynamic Decision Modeling: A Case Study, AAAI Spring Symposium 1994 Working Notes for Artificial Intelligence in Medicine.

14. T. Y. Leong. Dynamic Decision Modeling in Medicine: A Critique of Existing Formalisms, submitted to the Journal of American Informatics Association. (Extended version of paper presented at Symposium of Computer Applications in Medical Care, 1993.)

15. Michael P. Wellman and Jon Doyle. Preferential semantics for goals. In Proceedings of the National Conference on Artificial Intelligence, pages 698–703, 1991.

16. Michael P. Wellman and Jon Doyle. Modular utility representation for decision-theoretic planning. In Proceedings of the First International Conference on AI Planning Systems, 1992.

17. Jon Doyle and Michael P. Wellman. Representing preferences as ceteris paribus comparatives. In Steve Hanks, Stuart Russell, and Michael P. Wellman, editors, Working Notes of the AAAI Spring Symposium on Decision-Theoretic Planning, 1994.