OSCAR — A General-Purpose Defeasible Reasoner

John L. Pollock
Department of Philosophy
University of Arizona
Tucson, Arizona 85721
(e-mail: pollock@ccit.arizona.edu)

In its present incarnation, OSCAR is a fully implemented programmable architecture for a rational agent. This architecture is described in detail in a forthcoming book entitled *Cognitive Carpentry*. It is an important characteristic of OSCAR that most practical reasoning is reduced to epistemic reasoning, through a process I call "doxastification". In particular, most (but not quite all) of the details of planning, plan adoption, and plan execution are carried out by epistemic reasoning. If we focus just upon the epistemic reasoning in OSCAR, what we have is a powerful general-purpose defeasible reasoner. The purpose of this paper is to describe that reasoner.

OSCAR's defeasible reasoner is based upon seven fundamental ideas. These are (1) an argument-based account of defeasible reasoning, (2) an analysis of defeat-status given a set of interrelated arguments, (3) the proposal of a general adequacy criterion for automated defeasible reasoners, called "i.d.e.-adequacy", (4) the claim that a general-purpose defeasible reasoner must have the kind of structure I have called "flag-based", (5) an algorithm for computing defeat-status, (6) an interest-driven monotonic reasoner, and (7) an account of degrees of justification, degrees of interest, and their interactions.

1. Argument-Based Defeasible Reasoning

The basic conception of defeasible reasoning embodied in OSCAR can, in its most general form, be regarded as the received view in philosophical epistemology. It is the argument-based approach pioneered by Roderick Chisholm and myself in the late 60's and early 70's, and developed further by a number of people in the intervening years (e.g., Kyburg [1983], Loui [1987], and Nute [1988] and [1990]). The basic idea is that reasoning consists of the construction of arguments, where reasons are the atomic links in arguments. Defeasibility arises from the fact that some reasons are subject to defeat. I call these "prima facie reasons", and the considerations that defeat them are "defeaters". Although this is somewhat controversial, I have long argued and will here assume that there are only two kinds of defeaters. A "rebutting defeater" is a reason for denying the conclusion of the reason. An "undercutting defeater" attacks the connection between the premises and the conclusion, and can be regarded as a reason for denying that the premises wouldn't be true unless the conclusion were true. Reasons that are not defeasible are "conclusive".

A convenient way to encode arguments is as "inference-graphs". The nodes of an inference-graph represent premises and conclusions, and the links between the nodes represent dependency relations. The *node-basis* of a node is the set of nodes from which it is inferred. We can combine all of the reasoning of the reasoner into a single global inference-graph, and that will be the central data-structure used by the reasoner. Defeat relations can be encoded by adding a separate category of *defeat links* to the inference-graph. An important point about OSCAR's inference-graphs is that if the same conclusion is obtained by two different arguments, this is represented by two different nodes. In this way, we can take the nodes to have univocal defeat-statuses. In effect, each node encodes an argument for the conclusion represented at that node.

Because OSCAR engages in "suppositional reasoning", wherein conclusions are drawn relative to suppositions, the nodes of the inference-graph will be taken to encode sequents rather than formulas. A sequent is a pair $\langle \Gamma, P \rangle$ where P is a formula and Γ is a set of formulas (the supposition).

2. Justification and Defeat

Given a set of arguments some of which may support defeaters for inferences contained in others, we require an account of which of these arguments are defeated and which are undefeated. My analysis of defeat-status has evolved over the years, largely in an attempt to handle some intuitively complicated cases like the paradox of the preface and cases involving self-defeat (where an argument may defeat some of its own steps). I now believe that I have an adequate account of these phenomena, and my account is defended in a new paper (my [1994]). The analysis is as follows. We first define:

> A node of the inference-graph is *initial* iff its node-basis and list of node-defeaters is empty.
>
> σ is a *partial status assignment* iff σ is a function assigning "defeated" and "undefeated" to a subset of the nodes of an inference-graph in such a way that:
> 1. σ assigns "undefeated" to all initial nodes;
> 2. σ assigns "undefeated" to a node η iff σ assigns "undefeated" to all the members of the node-basis of η and σ assigns "defeated" to all node-defeaters of η; and
> 3. σ assigns "defeated" to a node η iff either some member of the node-basis of η is assigned "defeated", or some node-defeaters of η and is assigned "undefeated".
>
> σ is a *status assignment* iff σ is a partial status assignment and σ is not properly contained in any other partial status assignment
>
> A node is *undefeated* iff every status assignment assigns "undefeated" to it; otherwise it is *defeated*.

It is shown in my [1994] that this analysis is able to handle all of the problem cases that motivated it. I will not defend it further here.

This analysis has the consequence that a distinction can be made between two kinds of defeated nodes. Sometimes a node is defeated "outright" by another node that is undefeated. In that case, if the defeated node is a

defeater for a third node, the third node remains undefeated. But sometimes a node is defeated "collectively". This happens when two or more otherwise undefeated nodes are defeaters for each other. Then they are all defeated. But if one of them is a defeater for another node, that further node is also defeated, despite the fact that its defeater is defeated. Defeated nodes that retain the power to defeat other nodes are "provisionally" defeated.

A conclusion is "justified" at a particular stage of reasoning iff it is supported by some undefeated argument. However, further reasoning could produce further relevant arguments that will change the status of a conclusion from justified to unjustified, or vice versa. Given a set of premises and an array of reasons and rules of inference, we can say that a proposition is "warranted" iff the inference-graph produced by the set of all possible arguments built from those building blocks contains an undefeated node supporting the conclusion. Warranted propositions are the "ultimately reasonable" conclusions a defeasible reasoner is striving to identify.

3. I.D.E.-Adequacy

Perhaps the greatest problem facing the designers of automated defeasible reasoners is that the set of warranted conclusions resulting from a set of premises and reason schemas is not in general recursively enumerable. This was first observed informally by David Israel and Raymond Reiter in 1980. I gave a more formal proof of it in my [1992]. This has the consequence that a defeasible reasoner cannot look like a theorem prover, just systematically grinding out its conclusions in a mechanical way. Something more sophisticated is required. In my [1992], I urged that we take defeasibility seriously, and allow a defeasible reasoner to draw conclusions tentatively, sometimes retracting them later, and perhaps reinstating them still later, and so on. The set of conclusions drawn by such a reasoner cannot constitute a recursive enumeration of the set of warranted propositions, but I have proposed instead that it should systematically approximate the set of warranted conclusions in the following sense:

> The rules for reasoning should be such that if the reasoner is interested in a proposition *p* then:
> (1) if *p* is warranted, the reasoner will eventually reach a stage where *p* is justified and stays justified;
> (2) if *p* is unwarranted, the reasoner will eventually reach a stage where *p* is unjustified and stays unjustified.

If these conditions are satisfied, the sequence of sets of justified conclusions at different stages of reasoning constitutes a "defeasible enumeration" of the subset of warranted propositions in which the reasoner is interested, and the reasoner is said to be "i.d.e.-adequate".

Because the set of warranted propositions is not recursively enumerable, an i.d.e.-adequate reasoner may never stop reasoning. In simple cases it may stop, by running out of things to do, but in complicated cases it will go on reasoning forever. The significance of such a reasoner, embedded in a rational agent, is that it tells the agent what to believe "given the current state of its reasoning", and if the agent has to take action, it does so on the basis of what it believes at that time, even if there is no guarantee that it would not change its beliefs later if it had more time to reason. This involves the conception of defeasible conclusions as "innocent until proven guilty". In other words, they are pefectly reasonable beliefs, and it is reasonable to act upon them, despite the fact that the agent cannot "prove" that they will never have to be retracted.

4. Flag-Based Reasoners

In the attempt to build an i.d.e.-adequate defeasible reasoner, a natural first inclination is to suppose that when a conclusion is defeated, the reasoner should stop reasoning from it unless or until it is subsequently reinstated. It turns out, however, that such a proposal cannot work. Consider two long arguments, and suppose the final step of each defeats an early step of the other. It follows from my analysis of defeat-status that both arguments are defeated. They undergo "collective defeat". But a reasoner that stopped reasoning from a conclusion once it was defeated would never discover the collective defeat, because having completed one of the arguments, it would cease developing the other one. For this reason, an i.d.e.-adequate reasoner must continue reasoning from conclusions even when they are defeated. This suggests that such a reasoner should simply flag conclusions as defeated, and go on reasoning. Defeat-status may affect the priorities that are assigned to performing various inference steps, so that undefeated conclusions are given some precedence over defeated ones, but inferences from defeated conclusions must still go forth. I call such a reasoner a "flag-based" reasoner. We can think of a flag-based reasoner as consisting of two largely autonomous modules -- a "monotonic reasoner" that reasons and builds the inference-graph, without paying much attention (except in prioritizing inferences) to defeat-status, and a module that computes or recomputes defeat-statuses as new inferences are made. Such a reasoner has the form of a simple loop:

```
(loop
    (draw-a-conclusion)
    (recompute-defeat-statuses))
```

In my [1992], I was able to prove that subject to some reasonable assumptions about the structure of prima facie reasons provided to the reasoner and the adequacy of the monotonic reasoner, the resulting flag-based reasoner will be i.d.e.-adequate.

5. Computing Defeat-Status

A direct implementation of the defeat-status computation described in section two would have us look at all possible partial assignments of "defeated" and "undefeated" to the nodes of the inference-graph, determine which are status assignments by checking them for consistency and maximality (which will eliminate most of them), and then compute defeat-status on that basis.

That approach is combinatorially impossible for large inference-graphs. For an inference-graph with n nodes, we would have to generate and check 3^n assignments. Much of OSCAR's power results from a more efficient algorithm for generating the status assignments used for computing defeat-status. For this purpose, it is convenient to redefine (partial) status assignments to be three-valued functions, assigning "defeated", "undefeated", or "unassigned":

> σ is a *partial status assignment* iff σ is a function assigning "defeated", "undefeated", or "unassigned" to the nodes of an inference-graph in such a way that:
> 1. σ assigns "undefeated" to all initial nodes;
> 2. σ assigns "undefeated" to a node α iff σ assigns "undefeated" to all members of the node-basis of α and assigns "defeated" to all node-defeaters of α; and
> 3. σ assigns "defeated" to a node α iff either some member of the node-basis of α is assigned "defeated", or some node-defeaters of α and is assigned "undefeated";
> 4. σ assigns "unassigned" to a node α iff it does not assign either "defeated" or "undefeated", i.e., iff it does not assign "defeated" to any member of the node-basis of α or "undefeated" to any node-defeater of α, and it either assigns "unassigned" to some member of the node-basis of α or to some node-defeater of α.
>
> If σ and η are partial status assignments, σ is a *proper sub-assignment* of η iff σ and η have the same domain, $\sigma \neq \eta$, and for every node n, if $\sigma(n) \neq$ "unassigned" then $\eta(n) = \sigma(n)$.
>
> σ is a *status assignment* iff σ is a partial status assignment and σ is not a proper sub-assignment of any other partial status assignment.

Given a partial assignment to a subset of the nodes of the *inference-graph*, define the *assignment-closure* of that assignment to be the assignment that results from recursively applying the following rules until no further nodes receive assignments or some node receives inconsistent assignments:

- if all members of the node-basis of a node have been assigned "undefeated" and all defeaters for the node have been assigned "defeated", assign "undefeated" to the node;
- if some member of the node-basis of a node has been assigned "defeated" or some defeater for the node has been assigned "undefeated", assign "defeated" to the node;
- if all members of the node-basis of a node and the node-defeaters of a node have received assignments, no member of the node-basis has been assigned "defeated", no member of the node-defeaters has been assigned "undefeated", and some member of either the node-basis or node-defeaters has been assigned "unassigned", then assign "unassigned" to the node;
- if some node is assigned two different statuses, the assignment-closure is empty.

Before discussing the algorithm for generating all maximal partial-assignments, consider a related (but simpler) algorithm for producing all total assignments, where the latter are defined as follows:

> A *total assignment* is a partial assignment that does not assign "unassigned" to any node.

The following algorithm will generate all total assignments:

- Let σ_0 be the assignment-closure of the partial assignment that assigns "undefeated" to all initial nodes, and let *P-ass* = $\{\sigma_0\}$.
- Let *Ass* = $\emptyset$.
- Repeat the following until no new assignments are generated:
  - If *P-ass* is empty, exit the loop.
  - Let σ be the first member of *P-ass*:
    - Delete σ from *P-ass*.
    - Let n be a node which has not been assigned a status but for which all members of the node-basis have been assigned "undefeated".
    - If there is no such node as n, insert $\langle\sigma,A\rangle$ into *Ass*.
    - If there is such an n then:
      - Let S be the set of all assignments that result from extending σ by assigning "defeated" and "undefeated" to n.
      - Insert all non-empty assignment-closures of members of S into *P-ass*.
  - If *Ass* is unchanged, exit the loop.
- Return *Ass* as the set of assignments.

This algorithm generates all total assignments by trying to build them up recursively from below (ordering nodes in terms of the "inference-ancestor" relation). When this leaves the status of a node undetermined, the algorithm considers all possible ways of assigning statuses to that node, and later removes any combinations of such "aribtrary" assignments whose assignment-closures prove to be inconsistent. The general idea is that assignments are generated recursively insofar as possible, but when that is not possible a generate-and-test procedure is used.

To modify the above algorithm so that it will generate all maximal partial assignments, instead of just deleting inconsistent arbitrary assignments, we must look at proper sub-assignments of them. When such a proper sub-assignment has a consistent assignment-closure, and it is not a proper sub-assignment of any other consistent assignment, then it must be included among the maximal partial assignments. To manage this, the algorithm must keep track of which nodes have been assigned statuses arbitrarily in the course of constructing an assignment. Let σ_0 be the assignment-closure of the partial assignment that assigns "undefeated" to all initial nodes. Let us take an "annotated-assignment" to be a pair $\langle\sigma,A\rangle$ where A is an arbitrary assignment to some set of nodes, and σ is the assignment-closure of $\sigma_0 \cup A$. The algorithm constructs

annotated assignments:

COMPUTE-ASSIGNMENTS

- Let σ_0 be the assignment-closure of the partial assignment that assigns "undefeated" to all initial nodes, and let *P-ass* = $\{\langle\sigma_0,\varnothing\rangle\}$.
- Let *Ass* = $\varnothing$.
- Repeat the following until an exit instruction is encountered:
  - If *P-ass* is empty, exit the loop.
  - Let $\langle\sigma,A\rangle$ be the first member of *P-ass*:
    - Delete $\langle\sigma,A\rangle$ from *P-ass*:
    - Let n be a node which has not been assigned a status but for which all members of the node-basis have been assigned "undefeated".
    - If there is no such node as n, insert $\langle\sigma,A\rangle$ into *Ass*.
    - If there is such an n then:
      - Let *Ass** be the set of all $A \cup X^*$ such that X^* is an arbitrary assignment of "defeated" or "undefeated" to n.
      - Let S be the set of all maximal sub-assignments S^* of members of *Ass** such that the assignment-closure of $S^* \cup \sigma_0$ is non-empty.
      - For each member A of S^*:
        - If any member of *P-ass* is a sub-assignment of A, delete it from *P-ass*.
        - If any member of *Ass* is a sub-assignment of A, delete it from *Ass*.
        - If A is not a sub-assignment of any member of *P-ass* or *Ass*, insert A into *P-ass*.
- Return as the set of assignments the set of all σ such that for some A, $\langle\sigma,A\rangle$ is in *Ass*.

The correctness of this algorithm turns on the following observations:
(1) Every partial assignment can be generated as the assignment-closure of the assignment to the initial nodes and an arbitrary assignment to some otherwise undetermined nodes.
(2) If a partial assignment is inconsistent, so is every extension of it.
The algorithm makes use of (1) in the same way the previous algorithm did. In light of (2), in ruling out inconsistent assignments, we can shrink the search space by ruling out inconsistent sub-assignments and then only test for consistency the extensions of the remaining consistent sub-assignments.

Although this algorithm is much more efficient than a brute-force approach, it too encounters an efficiency problem. This algorithm constructs assignments for all nodes of the *inference-graph* simultaneously. Then when it finds an inconsistency in the assignment-closure, it considers all maximal consistent sub-assignments of that entire assignment. This has the result that if we present two copies of the same problem to the algorithm simultaneously, the number of sub-assignments it considers will increase exponentially rather than linearly. To illustrate, consider the four-way collective defeat of figure 1 (where defeasible inferences are represented by dashed arrows "⋯⋯▸", non-defeasible inferences by solid arrows "⟶", and defeaters by arrows of the form "⟿"):

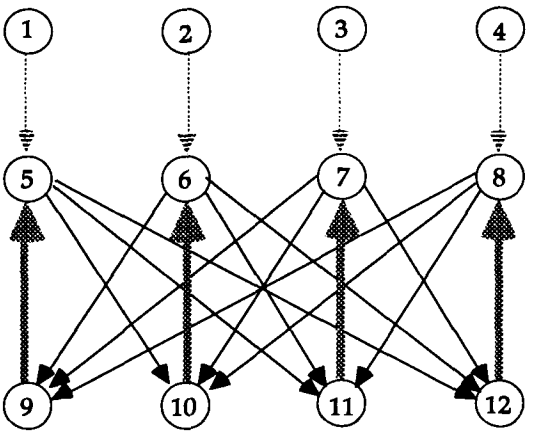

Figure 1. Four-way collective-defeat.

The preceding algorithm for computing assignments produces four assignments (one assigning "defeated" to each of nodes 5,6,7,8), in 0.07 seconds. But when we consider composite problems consisting of multiple instances of this single problem, the situation deteriorates rapidly:

Figure 1	0.07 sec 4 assignments	
2 *instances of #1*	1.20 sec 16 assignments	(increase by a factor of 17.14)
3 *instances of #1*	18.24 sec 64 assignments	(×260.57)
4 *instances of #1*	310.67 sec 256 assignments	(×4438.14)

There should be a better way of doing the computation. The different parts of the composite problem are really separate subproblems, and it should be possible to separate the computations, which would result in a linear increase in computation time rather than an exponential one. Note that in realistic cases, this will be important, because the *inference-graph* of a real agent will generate a large base-assignment consisting of the assignment-closure of an assignment of "undefeated" to the initial nodes, and then a possibly large number of small independent problems consisting of collective defeat and kindred phenomena, each generating multiple extensions to the base-assignment. It should be possible to handle each of these sets of multiple extensions independently, rather than lumping them all together.

Let us define the inference/defeat-ancestors of a node to be the set of nodes that can be reached by working

backwards via inference-links and defeat-links. Precisely:

> $Node_1$ is an *inference/defeat-ancestor* of $node_2$ iff $node_1$ is a member of the node-basis or node-defeaters of either $node_2$ or an inference/defeat ancestor of $node_2$.

In computing the defeat-status of a node, the only nodes that are relevant are its inference/defeat-ancestors. When we are presented with different subproblems embedded in a single inference-graph, we can divide the inference-graph into regions as in figure 2. Here *D* is the set of nodes assigned statuses by the base-assignment. The "triangle-sets" *A*, *B*, and *C* represent isolated subproblems. They are characterized by the fact that they are minimal sets *X* such that every inference/defeat-ancestor of a member of *X* is in either *X* or *D*. For the purpose of computing the defeat-statuses of the members of the triangle-sets, we can treat the *AUD*, *BUD*, and *CUD* as separate inference-graphs. This will yield the same value for the defeat-status as a computation based upon the entire inference-graph.

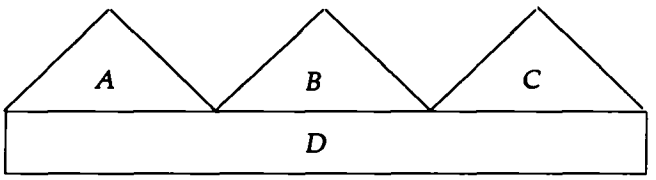

Figure 2. Triangle-sets.

The construction of triangle-sets can be made recursive, because there may be further triangle-sets that sprout above the existing ones. Notice, however, that higher-level triangle-sets must be relativized to assignments to the lower-level triangle-sets rather than to the lower-level triangle-sets themselves. This is because different assignments to a lower-level triangle-set may leave different sets of nodes unassigned and candidates for inclusion in a higher-level triangle-set.

A simple way to conceptualize this is in terms of an *assignment-tree*. The base of an assignment-tree is the base-assignment. That divides into a set of nodes representing triangle-sets. Each of those nodes gives rise to a set of assignments. Those assignments divide in turn into a new set of nodes representing triangle-sets, and so on, as diagramed in figure 3. This can be accommodated formally by taking an *assignment-tree* to be a data-structure consisting of an assignment and the list of resulting triangle-sets, and taking a *triangle-set* to be a data-structure consisting of a set of nodes (its domain) and the list of assignment-trees generated by the assignment-closures of maximal-consistent assignments to the nodes in the triangle-sets. For the purpose of evaluating the defeat-statuses of the nodes of the *inference-graph*, we can construct the assignment-tree *global-assignment-tree* generated by the base-assignment. Then a node is undefeated iff no assignment in *global-assignment-tree* assigns "defeated" or "unassigned" to it.

The complete assignments operative in a triangle-set will be the result of appending all of the assignments on the branch of the assignment-tree leading from the base-assignment to the triangle-set, and then appending to that the assignments to the triangle-set. The list of assignments on the branch leading to the triangle-set, beginning with the base-assignment, will comprise the *ancestor-assignments* of the triangle-set. These will consist of the assignment to the parent-tree, and the assignment to its parent-tree, and so on. These are computed recursively by working backwards through the tree. The assignment-closure of a new partial assignment must accordingly be understood as relative to an assignment-tree. A node *has a defeat-status* in a tree iff it is assigned a status either by the assignment of the tree or one of its ancestor assignments.

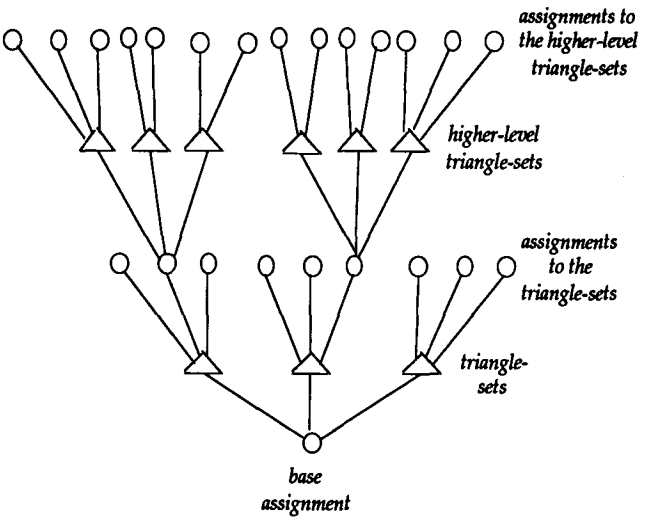

Figure 3. An assignment-tree.

The difference between the performance of the algorithms is illustrated dramatically by considering composites of figure 1 again. For a single instance of that problem, the original assignments-algorithm is the same speed as the assignment-tree-algorithm. But when we consider composite problems, the comparison changes dramatically:

	assignments-algorithm	**assignment-tree-algorithm**
Figure 1	0.07 sec 4 assignments	0.07 sec 4 branches
2 instances of #1	1.20 sec (×17.14) 16 assignments	0.15 sec (×2.14) 8 branches
3 instances of #1	8.24 sec (×260.57) 64 assignments	0.25 sec (×3.57) 12 branches
4 instances of #1	310.67 sec (×4438.14) 256 assignments	0.35 sec (×5.0) 16 branches

The difference is attributable to the fact that n copies of a simple problem that produces k assignments and an assignment-tree with k branches will produce n^k assignments but will produce an assignment-tree with only $n \cdot k$ branches. The times required to run the algorithms will be roughly proportion to the number of assignments and branches produced. The proportions are not exact because partial branches and assignments may be produced and later discovered to be inconsistent and so pared from the tree or dropped from the list of assignments.

We have an efficient algorithm for computing defeat-statuses for an inference-graph that has already been constructed. However, that is an artificial problem. A rational agent must construct its *inference-graph* one node at a time, and update the defeat-statuses with the addition of each node. The result of updating the assignment-tree with the addition of each node should be the same as the result of computing the assignment-tree as above for the completed inference-graph. One of the virtues of the assignment-tree algorithm is that it leads to an efficient procedure for updating the defeat-statuses of the nodes of the *inference-graph* in response to adding new nodes. When a node or set of nodes is added to the *inference-graph*, the only nodes that can be affected are the inference/defeat-descendants of the new nodes. The latter concept is defined as follows:

> Node$_1$ is an *inference/defeat-descendant* of node$_2$ iff node$_1$ is a member of the node-consequents or node-defeatees of either node$_2$ or an inference/defeat-descendant of node$_2$.

When we add new nodes, the changes to the inference-graph are confined to the new nodes and their inference/defeat-descendants. Call these the *affected-nodes*. If the new nodes have no defeatees, we can just compute their statuses and add them to the assignment-tree. Otherwise, we must update *global-assignment-tree*.

Updating a tree is performed only if updating the lower-order trees has not changed anything. In updating a tree, we check to see whether its assignment assigns anything to affected-nodes. If not, we update its triangle-sets. If these are unchanged, we go on to update the assignment-trees of the triangle-sets, and so on. Space limitations preclude further discussion of the updating algorithm, but this conveys the general idea.

6. Conclusions

In deductive reasoning, there is no point to producing more than one argument for a conclusion, but in nondeductive reasoning we cannot ignore multiple arguments, because some of them may get defeated. Having found one argument for a conclusion, we can lower the priority of searching for other arguments, but we cannot be totally oblivious to them. This creates an efficiency problem, however. In the interest of theoretical clarity, I defined the inference-graph in such a way that different arguments for the same conclusion are represented by different nodes. This made it clearer how the algorithm for computing defeat-status works. However, for the purpose of implementing defeasible reasoning, this is an inefficient representation of the reasoning, because it leads to needless duplication. If we have two arguments supporting a single conclusion, then any further reasoning from that conclusion will generate two different nodes. If we have two arguments for each of two conclusions, and another inference proceeds from those two conclusions, the latter will have to be represented by four different nodes in the inference-graph, and so on. This is illustrated in figure 4, where P and Q are each inferred in two separate ways, and then R is inferred from P and Q.

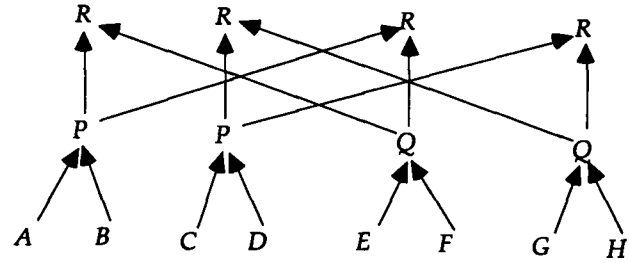

Figure 4. An inference-graph with multiple arguments for a single conclusion.

A more efficient representation of reasoning would take the inference-graph to be an and/or graph rather than a standard graph. In an and/or graph, nodes are linked to *sets* of nodes rather than individual nodes. This is represented diagramatically by connecting the links with arcs. In an and/or inference-graph, when we have multiple arguments for a conclusion, the single node representing that conclusion will be tied to different bases by separate groups of links. This is illustrated in figure 5 by an and/or inference-graph encoding the same reasoning as the standard inference-graph in figure 4.

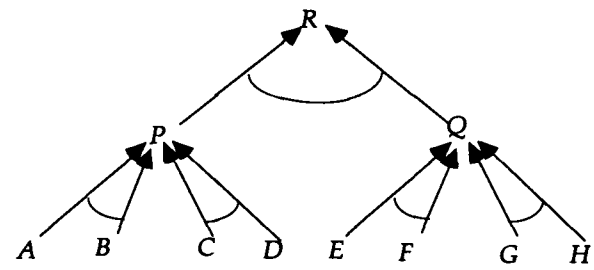

Figure 5. An and/or Inference-graph.

Although and/or graphs provide an efficient representation of reasoning, it turns out that they do not preserve enough information to enable us to compute defeat-status. Consider the two inference-graphs in figures 6 and 7. They could arise from different degrees of justification for the two nodes supporting R. In figure 6, only the rightmost node is strong enough to defeat S, whereas in figure 7, only the leftmost node is strong enough to defeat S. The difference has repurcusions, because applying the analysis of section two to figure 6 yields the result that S and T are both (provisionally) defeated, but

applying the analysis to figure 7 yields the result that S is defeated (outright) but T is undefeated. The difficulty is now that if we try to represent these as and/or graphs, we get the same graph, drawn in figure 8. This representation cannot distinguish between the situation diagramed in figure 6 and that diagramed in figure 7, and accordingly leaves the defeat-status of T indeterminate.

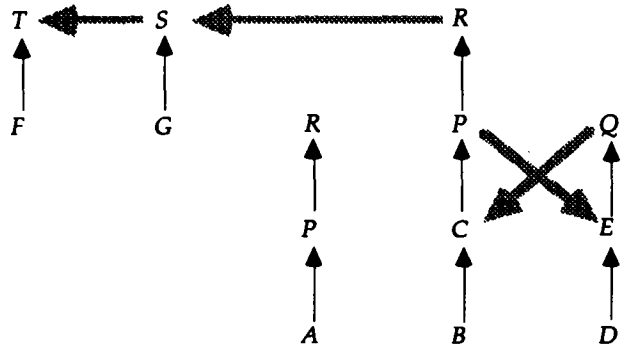

Figure 6. T is provisionally defeated.

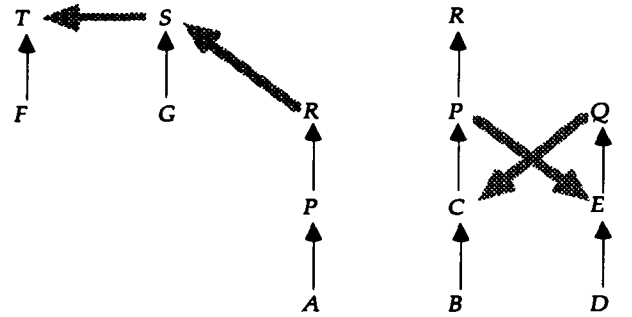

Figure 7. T is undefeated.

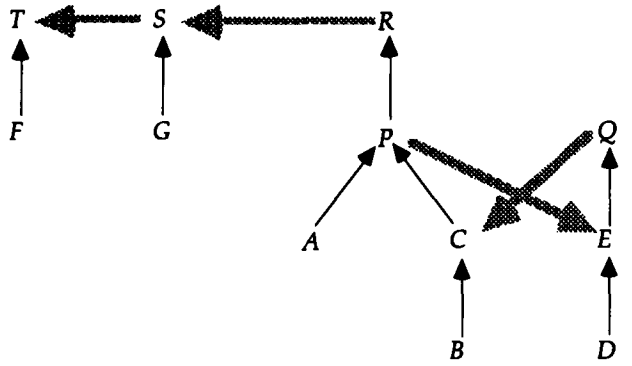

Figure 8. The status of T is indeterminate.

The difficulty we are encountering is that although reasoning is driven by what conclusions have been obtained, and is independent of the arguments for those conclusions, the computation of defeat-status depends essentially on those arguments. The and/or graph is adequate for the representation of reasoning, but it does not preserve enough information about the arguments to enable us to compute defeat-statuses. The problem has to do specifically with degrees of justification and the strengths of arguments. In the and/or graph, there is no way to represent that a particular conclusion is supported to different degrees by different arguments. The natural way to handle strengths in and/or graphs is to take the strength of a node to be the maximum of the strengths of the corresponding nodes in the standard inference-graph, but that leads to the difficulty illustrated in figure 8. We might instead associate each node of the and/or graph with a *set* of strengths, but that still does not enable us to associate the strengths with the arguments that have those strengths. The only obvious alternative is to associate each node with the set of complete arguments supporting it. But then the whole point of the and/or graph has been lost, because we are in effect just building the standard inference-graph into it.

To make both the reasoning and the defeat-status computation efficient, it appears that we must employ two separate data-structures. The defeat-status computation is performed on the *inference-graph*, and may require us to have multiple *inference-graph* nodes with the same node-sequent. Inferences, on the other hand, are made from sequents rather than from nodes. Rather than having to search the *inference-graph* for multiple nodes validating the same sequent, it is convenient to keep a record of nodes validating each sequent. OSCAR assumes the principle of "foreign adoptions", according to which in making an inference relative to a supposition, the reasoner can use any previously drawn conclusions that appeal to less inclusive suppositions. Let us say that a sequent $\langle \Gamma, P \rangle$ *subsumes* a sequent $\langle \Xi, Q \rangle$ iff $P = Q$ and $\Gamma \subseteq \Xi$. Then an *inference-graph* node *validates* a sequent iff the node-sequent subsumes the validated sequent. I will take a *conclusion* to be a data structure encoding the following information:

- the conclusion-sequent;
- the node-list — the list of *inference-graph* nodes validating the conclusion-sequent;
- the degree-of-support — the maximum of the degrees-of-support of the nodes in the node-list.
- the degree-of-justification — the maximum of the degrees-of-support of the undefeated nodes in the node-list, or 0 if they are all defeated.

The set of conclusions will be stored in the list of *conclusions*, and inference rules will access that list rather than looking directly at the *inference-graph*. To facilitate moving back and forth between *inference-graph* nodes and conclusions, *inference-graph* nodes contain a slot listing all conclusions in whose node-list the node is contained. I will say that a sequent is *supported* by a conclusion iff it is the conclusion-sequent of the conclusion, and a sequent is *validated* by a conclusion iff it is subsumed by the conclusion-sequent of the conclusion. Inferences are typically made from validated sequents rather than just concluded sequents.

7. Interest-Driven Reasoning

OSCAR's monotonic reasoner is based upon the deductive reasoner described in my [1990a]. This is an

"interest-driven suppositional reasoner" for first-order logic. The reference to suppositional reasoning just means that it can accommodate natural deduction rules like conditionalization, reasoning by cases, and *reductio-ad-absurdum*. For this purpose, the nodes of the *inference-graph* are taken to encode *sequents* rather than formulas. The sense in which the reasoner is interest-driven is that it reasons both backwards from the desired conclusions and forwards from the given premises. Reasoning backwards can be regarded as deriving interests from interests. This is related to backwards chaining and forwards chaining, but it is not quite the same thing. The difference lies in the fact that different rules of inference and different reason schemas are employed for backwards reasoning and for forwards reasoning. The motivation for this is that natural rules of inference are often very reasonable when applied in one direction but combinatorially explosive when applied in the other. For instance the rule of *addition* tells us to infer the disjunction $(P \vee Q)$ from the disjunct P. As a rule of backwards reasoning, this is eminently reasonable. It tells us that if we want to establish a disjunction, one way to do that is to try to get the first disjunct. But as a rule of forwards reasoning it would be catastrophic. It would have the reasoner infer every disjunction involving any conclusion it obtains.

As a deductive reasoner, this monotonic reasoner turns out to be surprisingly efficient, as is documented in my [1990a]. However, the original motivation for developing it was to incorporate it into a system of defeasible reasoning. This is because standard deductive reasoners based upon resolution refutation cannot serve that purpose. Such systems, in effect, always reason by reductio ad absurdum, but reductio ad absurdum is invalid for reasoning involving prima facie reasons. If a contradiction is obtained from some premises by reasoning via prima facie reasons, that does not support an inference to the negation of the premises. Instead, it defeats the defeasible inferences.

7. Degrees of Justification and Interest

In deductive reasoning, all reasons are equally good (i.e., perfect). But defeasible reasons can vary in strength. This has important consequences for the behavior of a defeasible reasoner. For instance, given an argument for a conclusion and a stronger argument against it, the stronger argument wins. A common view in epistemology has been that argument strengths should behave like probabilities, but I have argued against that in numerous places (e.g., my [1990c] and [1994a]). OSCAR instead computes argument-strengths in terms of the "weakest link" principle.

Interests also come in degrees. A rational agent has practical goals, and this leads to queries regarding how to satisfy those goals being sent to the epistemic reasoner. The reasoner thus becomes interested in answering those queries, and that initiates backwards reasoning. But some of the goals will typically be more important than others, and this is reflected by differing degrees of interest in the queries deriving from those goals. One effect of those degrees of interest is to prioritize the backwards reasoning -- preference is given to answering more important questions first.

OSCAR imposes an important connection between degrees of justification and degrees of interest. Basically, more important questions require better answers. In other words, when the agent reasoners backwards to a certain question and forwards to an answer, the answer is only deemed adequate if the degree of justification is at least as great as the degree of interest.

8. The Status of the OSCAR Project

The defeasible reasoner described here is currently running, and will shortly be made available to others through anonymous FTP. It constitutes the most important part of a general architecture for a rational agent. It will provide the basis for an attempt to implement my [1990] theory of probabilistic and inductive reasoning, and to implement the defeasible approach to planning and acting described in my [1992a] and in more detail in *Cognitive Carpentry*..

Bibliography

Israel, David

1980 What's wrong with non-monotonic logic? *Proceedings of the First Annual National Conference on Artificial Intelligence*. 99-101.

Kyburg, Henry, Jr.

1983 The reference class. *Philosophy of Science* **50**, 374-397.

Loui, Ron

1987 Defeat among arguments: a system of defeasible inference. *Computational Intelligence* **3**, 100-106.

Nute, Donald

1988 Defeasible reasoning: a philosophical analysis in PROLOG. *Aspects of AI*, ed. J. Fetzer. Reidel.

1990 Basic defeasible logic. In *Intensional Logics for Programming*, ed. L. Farinas-del-Cerro and M. Pentonnen.

Pollock, John L.

1990 *Nomic Probability and the Foundations of Induction*. Oxford University Press.

1990a Interest driven suppositional reasoning. *Journal of Automated Reasoning* **6**, 419-462.

1990b OSCAR: a general theory of rationality. *Journal of Experimental and Theoretical AI*..

1990c A theory of defeasible reasoning. *International Journal of Intelligent Systems* **6**, 33-54.

1991 Self-defeating arguments. *Minds and Machines* **1**, 367-392.

1992 How to reason defeasibly. *Artificial Intelligence* **57**, 1-42.

1992a New foundations for practical reasoning. *Minds and Machines* **2**, 113-144.

1994 Justification and defeat. *Artificial Intelligence*, forthcoming.

1994a The role of probability in epistemology. *Computational Intelligence*, forthcoming.

Reiter, Raymond

1980 A logic for default reasoning. *Artificial Intelligence* **13**, 81-132.