Using Strategies and AND/OR Decomposition for Back of the Envelope Reasoning

Praveen K. Paritosh (paritosh@cs.northwestern.edu)
Kenneth D. Forbus (forbus@northwestern.edu)
Qualitative Reasoning Group, Department of Computer Science,
Northwestern University, 1890 Maple Ave,
Evanston, IL 60201 USA

Abstract

Back of the envelope reasoning involves generating quantitative answers in situations where exact data and models are unavailable and where available data is often incomplete and/or inconsistent. A rough estimate generated quickly is more valuable and useful than a detailed analysis, which might be unnecessary, impractical, or impossible because the situation does not provide enough time, information, or other resources to perform one. Such reasoning is a key component of commonsense reasoning about everyday physical situations. In this paper we present an approach that uses strategies and creates an AND/OR decomposition to solve such questions. We present SOLVE, a general-purpose problem solving framework that uses strategies represented by *suggestions*, and keeps track of problem solving progress in an AND/OR tree. SOLVE can currently solve some fairly interesting back of the envelope estimation questions from different domains. We are building a library of strategies, which currently contains 23 strategies.

1 Introduction

One goal of qualitative reasoning (QR) is to understand and model common sense. Forbus and Gentner (1997) proposed a hybrid model of QR where analogical reasoning and qualitative reasoning are tightly interwoven. In this paper, we look at quantitative estimation (also called rough estimation, back of the envelope analysis, etc). Back of the envelope (BotE) analysis involves the estimation of rough but quantitative answers to questions where the models and the data might be incomplete. In domains like engineering, design, or experimental science, one often comes across situations where a rough answer generated quickly is more valuable than waiting for more information or resources. Some domains like environmental science [Harte, 1988] and biophysics [O'Connor and Spotila, 1992] are so complex that BotE analysis is the best that can be done with the available knowledge and data. BotE reasoning is ubiquitous in daily life as well. Common sense reasoning often hinges upon the ability to rapidly make approximate estimates that are fine-grained enough for the task at hand. We live in a world of quantitative dimensions, and reasonably accurate estimation of quantitative values is necessary for understanding and interacting with the world. Our life is full of evaluations and rough estimates of all sorts. How long will it take to get there? Do I have enough money with me? How much of the load can I carry at once? These everyday, common sense estimates utilize our ability to draw a quantitative sense of world from our experiences. We believe that the same processes underlie both these common sense estimates and expert's BotE reasoning to generate ballpark estimates.

The two critical parts of such reasoning are using heuristics and strategies to simplify complex problems, and using one's *feel for numbers* to make suitable numeric estimates. This paper presents the results of our work on the former. We have implemented SOLVE, a problem solver that uses a library of strategies and a large knowledge base to solve BotE problems. SOLVE can currently answer questions like "How many hotdogs are sold in a baseball season at Wrigley Field?"

The paper is organized as follows: next section argues that BotE reasoning has the same kind of constraints that we believe common sense QR to have. Section 3 is devoted to the design and implementation of SOLVE. Section 4 discusses the results of running SOLVE on various examples and talks about the limitations of an approach that just uses first principles reasoning, and briefly discusses our ongoing work that addresses those limitations. Section 5 concludes with future work.

2 Common Sense QR and BotE

Some of the central assumptions of QR in practice must be rethought when considering common sense knowledge, as opposed to narrow domain expertise. It is commonplace in QR to assume that a domain theory is complete. This assumption is implausible for common sense reasoning, whether or not one views QR purely in terms of a component in a performance system or as a psychological model. The closer one looks at human knowledge, the more it appears that it is fragmentary, and more concrete than abstract. It may be that such an organization is a necessity for human-level performance, whether or not one is making psychological claims. Let us call this approach *Common Sense QR* (CSQR) for concreteness. There is a striking resemblance between the key constraints guiding CSQR and BotE reasoning. This is one of the strong motivations for building a BotE problem solver. Below we mention the five important constraints that we believe underlie CSQR:

1. *Incompleteness*. Domain theories are incomplete in terms of their coverage, and even what they do cover is incompletely covered.
2. *Concreteness*. Domain knowledge includes knowledge of many concrete, specific situations. These concrete descriptions are used directly in analogical reasoning, in addition to first-principles reasoning.

3. *Highly experiential*. Domain expertise improves through the accumulation of information, both concrete and abstract. Experience improves our abilities to reason through similar situations, and helps us develop intuitions for what is reasonable.
4. *Focused reasoning*. Instead of maintaining uncertainty and ambiguity for completeness, assumptions are made aggressively to tightly constrict the number of possibilities considered. Common sense reasoning is required for action in the world, and there are opportunities for interaction and further reflection, reducing the amount of stress on any particular computation. Thus it is better to answer rapidly and sometimes be wrong than to answer slowly and vaguely.
5. *Pervasively quantitative*. Our interaction with the real world requires concrete choices for quantities. For example, the amount of salt one adds while cooking a certain dish cannot be safely specified as "+". While there are certainly tolerances, and we believe that estimation requires drawing upon lots of examples, our actions in the end require that estimates manifest as exact values. Quite possibly this is true for every step along the way, as per the focused reasoning constraint.

3 SOLVE: A Model of Problem solving

Problem solving is the process that takes us from a problem to its solution. A computational model of problem solving has to understand the problem representation, has to have access to domain knowledge and the ability to retrieve knowledge that might be relevant. It also needs to have strategies, which it can try when the problem is complex and the answer is not directly found. It needs to maintain the workspace, where it keeps track of the work done and progress made on the problem. We have implemented SOLVE, a problem solver that uses –

- A large knowledge base (a subset of Cycorp's Cyc KB plus knowledge represented and developed in our research group) for domain knowledge and the FIRE reasoning engine for retrieving and accessing the knowledge base.
- *Suggestions* as representation for strategies.
- AND/OR tree as a model for maintaining the workspace.

In this section we explain the above ideas, and then present the core algorithm of Solve. Using AND/OR decomposition for problem solving is not a new idea [Nilsson, 1994], but there are quite subtle issues and interesting design choices we made in Solve which we have not seen mentioned in the literature. Let's have a look at an example that SOLVE can find answer for, and as we describe the system we'll refer to the example to ground the discussion.

Example: What is total annual amount of gasoline consumed by cars in the US?

Total consumption = Total miles driven/ miles per gallon
Total miles driven = Number of cars in the US * Miles driven per car per year
Miles driven per car per year = Miles driven per day * 365
If we say that every household owns a car, since some don't and some might have more than one, then
Number of cars in the US = number of households = population / average size of American household.
Now we have a model, and using the following numbers,
Population ~ 300 million, Average size of household ~ 3,
Daily miles driven ~ 20, Miles per gallon for a car ~ 20.
We get an estimate of 36.5 billion gallons.

3.1 Domain Knowledge

The Knowledge Base (KB) and the FIRE reasoning engine are part of background infrastructure that the work reported on this paper builds on, and is provided to contextualize and make the current work more understandable. The contents of our knowledge base are a 1.2 million fact subset of Cycorp's Cyc knowledge base, which provides formal representations about a wide variety of everyday objects, people, events and relationships. Problems, solutions, strategies are all represented uniformly and stored in this KB. Our group (Northwestern in collaboration with Xerox PARC) has built the FIRE reasoning engine. FIRE uses a special purpose database for storing the knowledge base. It can do analogical reasoning using structure mapping [Forbus *et al*, 2002], and has facility for adding various kinds of reasoning source that allow it to do specialized reasoning, such as spatial reasoning [Forbus *et al*, 2003]. It provides the conventional ASK and TELL interface to the knowledge base, and QUERY which uses backward chaining to see if it can find answers.

3.2 Strategies

We represent strategies using suggestions. A suggestion provides a decomposition for the problem. In the above example of annual gas consumption, we use the idea that the number of cars can be estimated by finding the number of households. The suggestion `HouseholdStrategyFor-CountingUnits` in Figure 1 captures the idea. It says that if we know that something is owned by households (refered to as `FamilyCohabitationUnit` in the KB), then we can find how many units of it are owned by estimating the number of households and number of units per household and multiplying them.

```
(defSuggestion HouseholdStrategyForCountingUnits
   :trigger (unitsTotal ?obj ?place ?time ?total-units)
   :test (ownedBy ?obj FamilyCohabitationUnit)
   :subgoals ((numberOfHouseholds ?place ?time ?num-households)
              (unitsPerHousehold ?obj ?units-per-household))
   :result-step (evaluate ?total-units
                  (TimesFn ?num-households ?units-per-household)))
```

Figure 1. An example suggestion.

There are four parts to a suggestion –

1. *Trigger*: The form which is query for which the suggestion might be applicable.
2. *Test*: Additional test conditions which must be true in order for the suggestion to work.
3. *Subgoals*: A list of forms that this suggestion decomposes the current problem into. These are AND-subgoals, meaning if any one of them fails, this suggestion fails to solve the original problem. These subgoals are fully ordered.
4. *Result-step*: The final step of the suggestion, which combines the answers to the subgoals.

Inside the suggestion, the variables are order scoped such that any variable introduced can be used in the subsequent parts of the suggestion. The `defSuggestion` macro mentioned above is a facility for the suggestion author – it expands the suggestion into assertions in predicate calculus that are stored in the KB. Suggestions can be as concrete or abstract as intended. For example, the above suggestion applies not only to cars, but to televisions, family insurance policies, etc. Our strategy library currently consists of 23 such suggestions. One of the goals of the current work is to show that as we build a larger corpus of examples, there is re-use of these strategies and novel compositions, resulting in being able to solve newer problems with very little or no new problem specific knowledge added.

3.3 Tracking problem solving progress

SOLVE uses an AND/OR tree[1] to track the progress as it is working on a problem. The mapping between the AND/OR tree and our representations is very direct. For a problem, there could be many applicable strategies, any one of which succeeding lead to a solution to the problem. This results in an OR node in the tree. A suggestion, on the other hand, introduces one or more subgoals all of which have to solved in order to solve the original goal. This results in an AND node in the tree. Figure 2 shows the AND/OR decomposition for our annual gas consumption example. An AND/OR decomposition lets us keep track of dependencies between the original problem and new subgoals introduced. During the course of problem solving, a node can be –

- SOLVED: An OR-node is solved when any one of its children gets solved, and an AND-node is solved when all of its children are solved.
- FAILED: An OR-node fails when all of its children fail, and an AND-node when any one of its children fail.
- MOOT: A node is moot when it is not solved or failed, but when there is no point on working on it at the current point. So, if any one of the siblings of OR-nodes has succeeded, the other siblings are MOOT-VIA-SUCCESS, as at any point we are interested in finding one solution, so if one strategy has succeeded, we don't want to pursue others right now. However, we might come back and un-moot the other strategies if at some point we want more solutions, or after propagating this solution upwards in the tree we find that the original goal is still not solved. A strategy can generate many solutions, and we try the next sibling strategy only if we have exhausted all the solutions that this strategy has to offer. On the other hand, if an AND-node fails, its siblings are MOOT-VIA-FAILURE as there is no point in working on them, as the parent suggestion has failed as a result of one of its children failing. However, if later we find that we can solve the subgoal that failed by working more, we can un-moot the siblings.

These inferences are made by maintaining flags at each node, which are updated/propagated after every unit of problem solving.

3.4 The SOLVE Algorithm

As SOLVE works on a problem, it maintains its progress in an AND/OR tree as mentioned above. It also maintains an agenda, which is a list of things that it can do next. The agenda consists of suggestions that have been found that it can try, and subgoals that have been suggested. The agenda is ordered by difficulty estimates[2] so that the first thing on the agenda is the easiest one. SOLVE starts with enqueuing the original goal on to the agenda and running the main loop. Since SOLVE is an incremental algorithm, the rest of

[1] Because the solutions are obtained and cached in a TMS, we get the functionality of an AND/OR graph, i.e., we don't re-solve an already solved node, although the underlying representation is a tree. The advantage of having a tree is that the propagation algorithms are much simpler.

[2] Currently we use a very crude estimate of difficulty – the length+depth of the s-expression corresponding to the agenda item! Over many problem solving episodes, one might maintain statistics about which strategies work best, and how tough a particular subgoal is, based on experience, and use that.

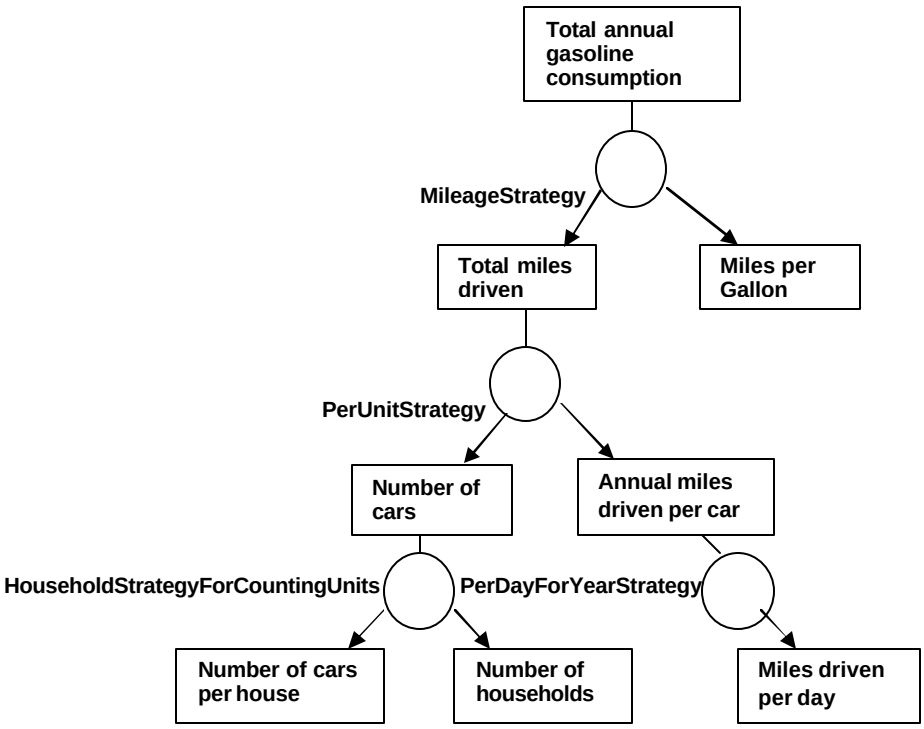

Figure 2. AND/OR tree for the gasoline consumption problem. The circles represent the suggestion nodes (AND-node) and the rectangles subgoal-nodes (OR-nodes).

the discussion will explain what happens at some point in midst of problem solving when we have done some work and have an already expanded AND/OR tree. There are two different ways in which solutions are generated in solve –

- AGENDA processing: The original goal hasn't been solved yet, and we are either trying to find suggestions that will solve it, or working on the subgoals that were suggested. It picks the easiest thing off the agenda. If it is a goal node, then sees if it can be solved by a primitive operation, ASK. If that fails, it gathers suggestions. Found suggestions are added the children of the original goal and enqueued on the agenda. If it is a suggestion node, then it instantiates the first subgoal of the suggestion node as a child node of the suggestion in the graph, and enqueues it on the agenda.
- IN-PLAY processing: This happens when the original goal has been expanded into a graph all of whose leaf nodes are solved. Now, no more problem solving needs to be done, and we can keep generating new solutions until we have exhausted all possible bindings found at the leaf nodes. We call a node that is solved and can possibly generate more solutions as an IN-PLAY node. Every subgoal maintains a pointer to the current IN-PLAY suggestion. IN-PLAY processing is implemented by the get-next-solution loop in SOLVE, whose main concern is to properly update what bindings have been already used.

All the bindings that are found as a result of a successful solution are maintained at the nodes locally and only those that are of interest to the parent from the first successful combination of the bindings are propagated upwards. Each node maintains marker to the bindings that it has already used, and these are updated to make sure we exhaustively go through the space of combination of bindings from the subgoals. Since the combinatorial possibilities of bindings from subgoals can be large, for example, consider a suggestion whose three subgoals are solved by a primitive operation (these will correspond to leaf nodes in the graph). For these leaf nodes if we found 2, 5 and 50 successful bindings, we have 500 combinations of bindings that could possibly lead to as many solutions for the parent. SOLVE tries each of these combinations one by one until it finds one solution for the parent and that is all it propagates upward in the graph. At the same point the binding markers at the nodes are updated appropriately so that solve could come back and try the next combination if that solution was not good, or we wanted more answers.

The main SOLVE loop drives both these kinds of processing. It checks if there are in-play solutions, if not it picks the next thing off agenda and processes it. The pseudocode for the main loop, get-next-solution, propagate-bindings and process-agenda steps are in the appendix.

4 Results

Table 1 shows the problems that SOLVE can find solve. Most of the times SOLVE finds an answer in the ballpark.

The goal of SOLVE is to find an answer that is no more than an order of magnitude off on either side. Sometimes, the estimates being off can be an interesting thing. So in question 4 below, we see that if we bought everybody personal insurance, we would be spending half of the US healthcare expense. The healthcare system in US is complicated, but this estimate provokes us to think about why the system is incurring more costs. In cases like these, carrying out an estimate and comparing it to the expected value might trigger a model refinement and the fact that one needs to know more to understand the process. In the last column, we have the number of suggestion specific axioms that were added to the knowledge base for the particular problem. One hope of this work is to show that that number decreases with increasing the corpus of problems that SOLVE can handle. As the number of strategies increases, we think we might asymptotically reach to a stage where very little or none problem specific knowledge is added for a new problem. The current work shows that this approach is promising, though we don't have enough data to make that claim here. We do find that some reuse of strategies in these examples which are from quite a broad range of domains, which we find encouraging.

Table 1. A summary of the problems that SOLVE can successfully do.

Problem Number	**Problem, and its predicate calculus representation**	**Answer found by SOLVE, and comparison to a known answer if available**	**Number of specific axioms added for this problem.**
1	How many popcorns would fill in the 1890 Maple big classroom? `(CountContained CS381ClassRoom Popcorn ?number)`	(?number . 1.343444e+7) SOLVE: 13 million Correct answer[3]: not available!	30
2	How much money is spent on Newspapers in the US? `(annualSales NewspaperCopy UnitedStatesOfAmerica (YearFn 2003) ?money)`	(?money . 2.1884363e+10) SOLVE: 21 billion Correct answer: 26 billion	30
3	How Many K-8 Teachers are there in the US? `(cardinality K-8SchoolTeacher ?numteachers)`	(?numteachers . 1056454) SOLVE: 1.05 million Correct Answer: 1.9 million	20
4	What is the annual cost of healthcare in the US? `(annualSales HealthCare UnitedStatesOfAmerica (YearFn 2003) ?money))`	(?money . 799428834000) SOLVE: 0.8 trillion Correct Answer: 1.6 trillion	12
5	How many cars are bought per year in the US? `(unitsBoughtPerYear Automobile UnitedStatesOfAmerica (YearFn 2003) ?num)`	(?num . 8920000) SOLVE: 8.9 million Correct Answer: 8 million	30
6	What is the weight of garbage thrown away by American families each year? `(annualProduction Garbage-Generic UnitedStatesOfAmerica (YearFn 2003) (Pound-UnitOfMass ?garbage-mass))`	(?garbage-mass . 446000000) SOLVE: 446 million pounds Correct answer: not available!	10
7	How many hotdogs are sold in a baseball season in Wrigley Field? `(unitsSold HotDogSandwich WrigleyField BaseballSeason ?num-dogs)`	(?num-dogs . 1600000) SOLVE: 1.6 million Correct answer: not available!	18
8	What is the total amount of gasoline consumption by cars in the US? `(annualAutomobileGasConsumption UnitedStatesOfAmerica (YearFn 2003) (Gallon-US ?oil-consumption)))`	(?oil-consumption . 32558000000) SOLVE: 32.6 billion gallons Correct Answer: 35 billion gallons	20

[3] The correct numbers are all from Statistical Abstracts of the United States, 2003.

4.1 Feel for Numbers

As mentioned earlier in the introduction, another key part of doing back of the envelope estimates is the feel for numbers. Once we have the model that relates the parameter to other quantities that might be known, the reasoning bottoms out with making good guesses for numeric parameters. Sometimes the exact numeric parameter might be known. But many times, we have observed in informal protocols of people doing this kind of reasoning, people are able to use an example (or multiple examples) from their experience to guess for a number that might not be directly known. Many times we know the range of values a parameter might fall in, but we need to guess a typical value rather than do interval math. For example, in the Cyc KB, it is known that –

```
(age Automobile (YearsDuration 5 50))
```

In question 5 above, this knowledge is used to infer how many cars are bought per year, by dividing the total number of cars by the age of a typical car. A typical value to assume might be 10 years as a life of a car. In many estimation problems, we are looking for typical, high, or low values for a parameter. For example, consider the question – What does a good gaming PC cost? Now, we know that a good gaming PC has a high RAM, expensive videocard, and a fast processor, and everything else might be the usual fare. Being able to represent and reason with notions like high, low and typical values for quantity is a key aspect of BotE reasoning. Using the theoretical framework laid out in Paritosh (2003), we are building CARVE [Paritosh 2004], a system that automatically builds symbolic representations of quantity. These symbolic representations will capture the feel for numbers part of BotE reasoning, thus making it much more flexible and powerful.

4.2 Related Work

The most similar project in this spirit was FERMI [Larkin *et al*, 1988]. FERMI used two general principles, *decomposition* and *invariance*, with domain specific knowledge to solve textbook problems in fluid statics, DC-circuits and centroid location. Our approach is simpler, more general, builds upon existing large knowledge bases, and is more concerned with the kind of breadth of common sense reasoning as opposed to natural science. Such reasoning has relevance to education, especially engineering. More than 90% of mechanical engineering seniors (100 at MIT, and 250 from five other universities) came up with wrong order of magnitude estimates of value of energy stored in a 9-volt "transistor" battery [Linder, 1999]. The responses varied by nine orders of magnitude excluding outliers! Having a clearer understanding of BotE reasoning at the knowledge level and computationally, might be helpful in fixing that kind of innumeracy.

5 Conclusions and Future Work

We presented SOLVE, a system that uses an AND/OR tree and suggestions to solve problems. SOLVE can already solve fairly interesting BotE estimation problems. We plan to represent a much larger corpus of BotE problems, of about fifty different problems. This will enable us to test the hypothesis that there is a stable collection of strategies that solve most BotE problems, by measuring the amount of additional background knowledge and strategies needed to solve each new problem. Tucked away in these strategies are interesting modeling decisions, approximations and simplifications. With a robust library of BotE strategies, we would like to get a better understanding of the similarities and differences between estimation-model building in BotE and model composition in compositional modeling [Falkenhainer and Forbus, 1991; Nayak, 1994]. We plan to combine SOLVE with representations built by CARVE, to give it the feel for numbers, which we expect will make it much more powerful and flexible.

Acknowledgments

This research is supported by the Computer Science Division of the Office of Naval Research.

References

Falkenhainer, B. and Forbus, K. "Compositional Modeling: Finding the Right Model for the Job", Artificial Intelligence, **51** (1-3), October, 1991.

Forbus, K., Gentner, D. 1997. Qualitative mental models: Simulations or memories? *Proceedings of Eleventh International Workshop on Qualitative Reasoning*, Cortona, Italy.

Forbus, K., Mostek, T., and Ferguson, R. 2002. An analogy ontology for integrating analogical processing and first principles reasoning. In *Proceedings of IAAI-02*, July 2002.

Forbus, K., Tomai, E., and Usher, J. 2003. Qualitative spatial reasoning for visual grouping in sketches. In *Proceedings of the 16th International Workshop on Qualitative Reasoning*, Brasilia.

Harte, J. 1988. Consider a spherical cow: A course in environmental problem solving, University Science Books, Sausalito, CA.

Larkin, J. H., Frederick R., Carbonell, J. and Gugliotta, A. 1988. FERMI: A Flexible Expert Reasoner with Multi-Domain Inferencing, *Cognitive Science*, **12**(1), 101-138.

Linder, B.M. 1999. Understanding estimation and its relation to engineering education, Ph.D. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology.

Nayak, P.P. 1994. Causal Approximations, Artificial Intelligence, **70**, 1-58.

Nilsson, N. 1994. Principles of Artificial Intelligence, Morgan Kaufman.

O'Connor, M.P., and Spotila, J.M. (1992). Consider a spherical lizard: Animals, models and approximations, American Zoologist, **32**, pp 179-193.

Paritosh, P. K. 2003. A Sketch of a Theory of Quantity, In *Proceedings of the 16th International Workshop on Qualitative Reasoning*, Brasilia.

Paritosh, P.K. 2004. Symbolizing Quantity, To appear in *Proceedings of the 26th Annual Meeting of Cognitive Science Society*, Chicago.

6 Appendix: Pseudocode for key SOLVE algorithms

```
get-solution(original-goal)
  if in-play?(original-goal)
    get-next-solution(original-goal)
    else if the agenda is empty then quit
  process-agenda

get-next-solution(ao-node)
  if there are cached solutions at this node
    ;; cur-bmarker points to the current solution
    increment cur-bmarker(ao-node)
    return cached solution
  if goal-node?(ao-node)
    if ao-node has a child suggestion that is in-play
       get-next-solution(in-play-suggestion(ao-node))
    ;; No in-play suggestion, if this goal has a younger
    ;; sibling that can give us more new bindings, re-instantiate
    ;; this node with those bindings.
    if younger-siblings(ao-node)
       bindings = get-next-solution(younger-sibling(ao-node))
       if bindings found
          enqueue-on-agenda(re-instantiate-node(ao-node,bindings))
  if suggestion-node?(ao-node)
    ;; seek downward
    solution = get-next-solution(eldest-child,ao-node)
    add-bindings(solution,ao-node)
    increment cur-bmarker(ao-node)
    return solution

propagate-bindings(goal-node)
  if elder-sibling?(goal-node)
    increment cur-bmarker(goal-node)
    next-node = instantiate-node(elder-sibling(goal-node),current-bindings(goal-
node))
    add-to-tree(next-node)
    enqueue-on-agenda(next-node)
    return
  ;; No elder sibling, so we can now see if the parent node got solved
  if parent(goal-node) exists
    bindings = get-next-solution(goal-node)
    while we have bindings for goal-node
      result-bindings = do-result-step(parent(goal-node))
      if result-bindings found
        add-bindings(result-bindings,parent(goal-node)
        propagate-bindings(parent(parent(goal-node)))
      bindings = get-next-solution(goal-node)
```

```
process-agenda(ao-node)
  if goal-node?(ao-node)
    solutions = ask(goal-node) ;; primitive problem solving step
    if solutions found
       propagate-bindings(goal-node)
    else suggestions = gather-suggestions(goal-node)
       if suggestions found
          enqueue-on-agenda(suggestions)
       else ;; no suggestions found, see if we can re-instiate this node
           if younger-siblings(goal-node)
              bindings = get-next-solution(younger-sibling(goal-node)
              if bindings found
                 enqueue-on-agenda(re-instantiate-node(ao-node,bindings))
              else update-failed (goal-node)
  if suggestion-node?(ao-node)
     ;; since the subgoals are ordered, add the first one to agenda
     enqueue-on-agenda(youngest-child(ao-node))
```