A Model-Based Approach to Blame Assignment: Revising the Reasoning Steps of Problem Solvers

Eleni Stroulia*
Center for Applied Knowledge Processing
Helmholtzstr. 16
89081 Ulm, Germany
stroulia@faw.uni-ulm.de

Ashok K. Goel
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
goel@cc.gatech.edu

Abstract

Blame assignment is a classical problem in learning and adaptation. Given a problem solver that fails to deliver the behaviors desired of it, the blame-assignment task has the goal of identifying the cause(s) of the failure. Broadly categorized, these causes can be knowledge faults (errors in the organization, content, and representation of the problem-solver's domain knowledge) or processing faults (errors in the content, and control of the problem-solving process). Much of AI research on blame assignment has focused on identifying knowledge and control–of–processing faults based on the trace of the failed problem-solving episode. In this paper, we describe a blame-assignment method for identifying content–of–processing faults, i.e., faults in the specification of the problem-solving operators. This method uses a structure–behavior–function (SBF) model of the problem-solving process, which captures the functional semantics of the overall task and the operators of the problem solver, the compositional semantics of its problem-solving methods that combine the operators' inferences into the outputs of the overall task, and the "causal" inter-dependencies between its tasks, methods and domain knowledge. We illustrate this model-based blame-assignment method with examples from AUTOGNOSTIC.

Introduction

Blame assignment is a classical problem in learning and adaptation (Samuel 1959). Given a problem solver that fails to deliver the behaviors desired of it, the general blame-assignment task has the goal of identifying the cause(s) of the failure. The types of the identified cause(s) can then be used as indices to appropriate learning strategies which can eliminate the causes of the failure and thus improve the problem solver.

A problem solver may fail due to a wide variety of causes that may be broadly categorized into knowledge faults and processing faults. The former (Davis 1980; Weintraub 1991) pertain to errors in the organization, content, and representation of the problem-solver's domain knowledge, while the latter refer to errors in the content of, and control over the steps of its processing. Much of AI research on blame assignment has focused on the identification of knowledge faults. In this paper, we focus on the identification of processing faults.

AI work on identification of processing faults itself has focused on faults in the control of processing. For example, both Lex (Mitchell et. al 1981) and Prodigy (Carbonell et. al 1989) assume that the causes of the failures of their problem solvers lie in their incorrect operator-selection heuristics. Their blame-assignment methods assume that the set of available operators is both complete and correctly specified. This assumes that the exact same operators used in the failed problem-solving episode can be in some way combined to produce a correct solution. In contrast, in this paper, we are interested in the issue of identifying faults in the specification of the operators. While we too assume that the set of available problem-solving operators is complete relative to the problem-solver's task, we admit the possibility that they may be incorrectly specified. That is, the information transformations the operators are designed to perform may not be sufficient for delivering a solution to (all of) the problems presented to the problem solver. Thus, the research issue becomes, given a problem-solver that fails to deliver the overall behavior desired of it, to specify a combination of knowledge and processing that enables the failing problem solver to identify faults in the specification of its operators.

Traditional blame-assignment methods for identifying faults in the control of processing are based on problem-solving traces. For example, both Lex and Prodigy require the trace of the processing in the failed problem-solving episode as well as a trace of the processing that would have led to problem-solving success. Under the assumption that the cause of the failure is that the problem solver does not know the exact conditions under which each of the operators should be used, both these systems compare the failed trace against the successful one to identify situations where operators were incorrectly used. In contrast, we describe a method which, in addition to the trace of the failed problem solving, uses a model of the problem-solver's processing and knowledge to identify faults in the specification of the problem-solving operators. We posit that the identification of faults in operator specification is facilitated by knowledge of (i) the functional semantics of the overall task of the problem solver, (ii) the functional semantics of its operators, (iii) the compositional semantics of the problem-

*In the Proceedings of the Thirteenth Annual Conference on Artificial Intelligence, Portland, Oregon, August 4-8, AAAI'96, pp. 959-965, AAAI Press.

solving methods that recursively synthesize the inferences carried out by the available operators into the outputs of its overall task, and (iv) the "causal" inter-dependencies between (sub)tasks, methods and domain knowledge. We use structure–behavior–function (SBF) models to capture this semantics of problem solving.

This model-based method for blame assignment is implemented in AUTOGNOSTIC, a "shell" which provides (i) a language for representing SBF models of problem solvers, and (ii) mechanisms for monitoring the problem solving, receiving and assimilating feedback on the result, assigning blame in case of failure, and repairing the problem solver. In this paper, we illustrate AUTOGNOSTIC's blame-assignment method for processing faults with examples from AUTOGNOSTIC's integration with ROUTER (Goel et. al 1994), a path-planning system.

SBF Models of Problem Solving

SBF models of problem solving analyze the problem-solver's task structure, its domain knowledge and their inter-dependencies. In this model, the problem-solver's tasks constitute the building blocks of its problem-solving mechanism. The problem-solving methods that it employs decompose its complex overall tasks into simpler subtasks. These, in turn, get recursively decomposed into even simpler subtasks until they become elementary reasoning steps, i.e., *"leaf" tasks*. These leaf tasks are directly accomplished by the problem-solver's domain operators.

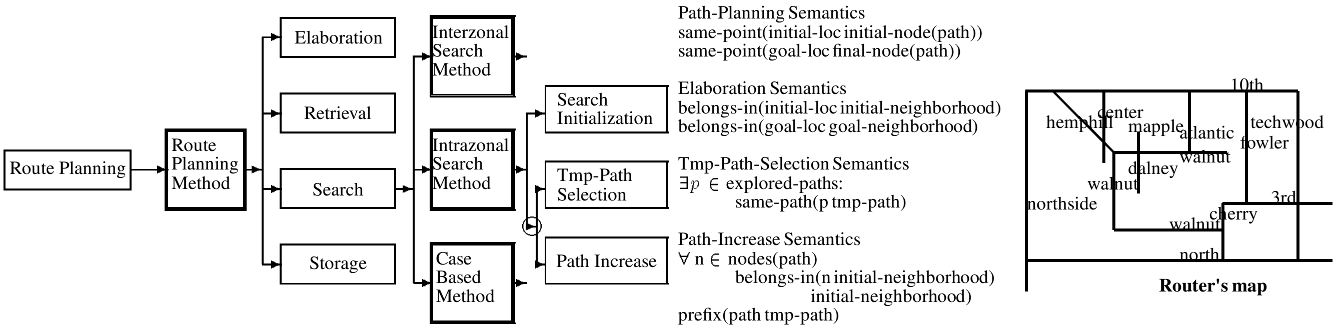
A task is specified as a transformation from an input to an output information state. It is characterized by the type(s) of information it consumes as input and produces as output, and the nature of the transformation it performs between the two. The functional semantics of a task describes the nature of the information transformation this task is intended to perform, and thus, constitutes a partial description of its expected, correct behavior. It is expressed in terms of specific domain relations that hold true among the task's inputs and outputs. For a non-leaf task, the functional semantics of the subtasks into which the task is recursively decomposed, and the ordering relations that the decomposing methods impose over them, constitute a partial description of a correct reasoning strategy for this task. Henceforth, we will use the term *strategy* to refer to the task tree that results from a task's decomposition by a particular method.

Methods can be thought of as general plans for how the solutions to different low-level subtasks get combined to deliver the output desired of higher-level tasks. They specify compositions of the problem-solver's domain operators into its higher-level tasks. Each method captures the semantics of the composition of a set of lower-level subtasks into a higher-level task in terms of control inter-dependencies (that is, a set of ordering relations), and information inter-dependencies, (that is, a set of information producer-consumer relations), among these subtasks.

In addition to the task structure, the SBF model of a problem solver specifies its domain knowledge in terms of the types of domain objects that the problem solver knows about, and the relations applicable to them. This specification of object types and relations captures the problem-solver's ontology of its domain. In addition, the information types flowing through the task structure are specified as instances of domain-object types, and thus each particular task input or output is related to the ontological commitments associated with the object type of which it is an instance. Finally, the specification of the tasks' functional semantics in terms of domain relations, that must hold between their inputs and outputs, captures the "causal" inter-dependencies of the inferences drawn to accomplish the tasks with the problem-solver's domain knowledge, where each inference is based on some specific domain knowledge. The assumption here is that if the semantics of a task is specified in terms of a particular domain relation, then in order to meet its semantics, the set of inferences drawn in service of this task will use the knowledge of the problem solver about this domain relation.

The Case Study: ROUTER is a multistrategy navigational planner which will be used in this paper to illustrate AUTOGNOSTIC's model-based method for blame assignment. ROUTER's task, `path-planning`, is to find a path from an initial location to a goal location in a physical space. Its spatial model of the navigation world is organized in a neighborhood-subneighborhood hierarchy. High-level neighborhoods describe large spaces in terms of major streets and their intersections. They get refined into lower-level neighborhoods which describe both major and minor streets and their intersections but over smaller spaces. Figure 1 diagrammatically depicts ROUTER's task structure and gives part of the SBF specification of some of its tasks and types of domain knowledge. In addition to the spatial model, Router contains a memory of past path-planning cases; the case memory is organized around the neighborhood-subneighborhood hierarchy.

AUTOGNOSTIC's SBF model of ROUTER's problem solving specifies that its overall task, `path-planning`, is decomposed into the subtasks `elaboration`, `retrieval`, `search` and `storage`. The elaboration subtask classifies the initial and goal locations into the neighborhood-subneighborhood hierarchy; it identifies the neighborhoods to which the two locations belong. This is a leaf task, i.e., it is directly solved by the domain operator `elaboration-op`. The `retrieval` subtask recalls from ROUTER's memory a similar path which connects locations spatially close to the current initial and goal locations. Next, the `search` subtask produces the desired path, and, subsequently, the `storage` subtask stores it in memory for future reuse. The search task can be accomplished by three different methods, the `intrazonal`, the `interzonal`, and the `case-based` method. The first two methods are model-based, that is, the semantics of the subtasks resulting from the use of these methods refer to model relations. In analogy, the semantics of the subtasks resulting from the use of



TYPES OF DOMAIN OBJECTS

	PATH		INTERSECTION
id_test	path-equivalence		intersection-equivalence
domain	–		intersection-domain
attrs	nodes	length	streets
	self	length(self)	self
	(list-of intersection)	number	(list-of 2 street)
rels	begins-in-neighborhood		connected-to
	ends-in-neighborhood		belongs-in

TYPES OF DOMAIN RELATIONS

	PREFIX	CONNECTED-TO	BELONGS-IN
input_args	path	intersection	intersection
output_args	(list-of path)	(list-of intersection)	neighborhood
truth_table	–	–	neighborhoods-of-ints-table
predicate	is-prefix	adjacent-ints	–
inv_predicate	prefix-of	adjacent-ints	–

Figure 1: Fragment of ROUTER's planning task structure and part of the SBF specification of some of ROUTER's domain objects and relations.

the case-based method refer to case-memory relations.

The first method is applicable only under the condition that the initial and the goal problem locations belong in the same neighborhood. It decomposes the `search` task into the subtasks `search-initialization`, `temp-path-selection` and `path-increase`. The first of these subtasks initializes the set of paths already explored by ROUTER to contain only a path consisting of a single location, i.e., the initial location. The `temp-path-selection` subtask takes as input this set of explored paths and selects from it a particular temporary path which feeds as input to the `path-increase` subtask. The latter task extends the temporary path to reach its neighboring points, i.e., all the intersections that belong in the common initial- and goal- neighborhood and are immediately adjacent to its last node. These extended paths are all added to the set of explored paths. The last two subtasks are repeatedly performed (as denoted by the small circle in the illustration of ROUTER's task structure in Figure 1), until one of the explored paths reaches the goal location, in which case, it is returned as the desired output of the overall task.

As shown in the bottom of Figure 1, ROUTER's world is described in terms of several different object types, such as `intersections`, `neighborhoods`, `streets` and `paths`. For each type of these domain objects, the SBF model specifies the set of values that specific instances of objects may take, the attributes of the object type, the predicate that evaluates whether two instances of this object type are identical, and the domain relations that relate it to other domain objects. The objects in ROUTER's world are related through `relations`, such as `belongs-in` which relates intersections to neighborhoods. For each type of domain relation the SBF model specifies the types of domain objects it applies to, and its truth table or the predicate that evaluates whether a tuple belongs in this relation or not.

Model-based Blame Assignment

In this paper, we focus on the blame-assignment task that arises when the problem solver is given as feedback information that a value desired for one of its outputs is different from the value actually produced. More specifically, the symptom of the failure is a divergence between the actual and the desired problem-solving behavior, although the actual behavior may be consistent with the range of behaviors intended of the problem solver. We will illustrate the model-based method for addressing this task with an example from ROUTER, which given the problem of going from *(10th center)* to *(walnut dalney)*, produces the path *((center 10th) (10th atlantic) (atlantic walnut) (walnut dalney))*, for which AUTOGNOSTIC receives the shorter path *((center 10th) (center mapple) (mapple dalney) (dalney walnut))* as feedback (see Figure 1 right for a map of ROUTER's navigational domain).

Before localizing the cause of the failure into a specific operator or piece of domain knowledge, the blame-assignment method evaluates whether the feedback is within the class of values the overall problem-solver's task was intended to produce; otherwise it would be meaningless to examine why it was not actually produced. To this end, it evaluates whether the feedback conforms with the overall task's expected correct behavior as specified by the task's functional semantics. As mentioned above, the SBF specification of a task's semantics consists of task's input and output information types and a domain relation. For each domain relation, the SBF model specifies a predicate (or, a truth table), which makes it possible to evaluate whether or not the specific values of input and output information in the episode belong to the domain relation. The tuple formed by the specific values of the input and output information in a particular problem-solving episode should belong in the domain relation.

In our example, AUTOGNOSTIC's first step is to establish, based on the semantics of the overall `path-planning` task (see Figure 1), whether the feedback path belongs in the class of paths that ROUTER was intended to produce given its actual input initial and goal locations. The feedback path begins at the initial and ends at the goal location, therefore, AUTOGNOSTIC infers that the feedback is indeed a valid output for ROUTER's current problem.

If the feedback belongs indeed in the class of intended correct outputs for the current problem input (see Figure 2[1]), then the strategy employed to accomplish this task should have produced it. Thus, the blame-assignment method postulates that the cause of the failure must lie within this strategy, that is, within some of its subtasks. From the trace of the failed problem-solving episode, it identifies the method which was used for the task in question, and focuses the search for the cause of the failure to the last subtask of this method producing the erroneous output (see Figure 2[1.2]).

Having established that the feedback belongs in the class of paths that `path-planning` could have produced for this problem, AUTOGNOSTIC postulates that the cause of the failure must lie within the strategy used to accomplish this task. Thus, it successively refines the focus of its investigation to the subtasks involved in the production of the path, i.e., `search` and next `path-increase`.

If at some point, the semantics of some task is not validated by its actual input and the feedback (see Figure 2[2]), then the blame-assignment method attempts to infer alternative inputs which would satisfy it. This is meaningful only when the task's input is not part of the overall problem specification, otherwise it would be an attempt to redefine the problem in order to fit the desired solution. If, however, the input of the task in question is produced by some earlier subtask, and, if alternative values can be found for it such that the current task's semantics is satisfied (see Figure 2[2.1]), then the blame-assignment method infers that the fault must lie within this earlier subtask which produced the "wrong" input for the task currently under examination. Therefore, it identifies the highest earlier subtask producing the information in question, and shifts its focus to assigning blame for its failure to produce the alternative desired value. To identify the producing subtask, the method uses the compositional semantics of the methods that gave rise to the subtask under examination. Had this knowledge not been available in the SBF model, the method would have to examine all the subtasks performed before the current subtask with failing semantics.

The ability to infer possible alternative values for types of information produced by the problem-solver's intermediate subtasks is based on the SBF specification first, of the functional semantics of the task under inspection, and second, of the domain relations on which this semantics is based. As we have already described, based on these types of knowledge, the blame-assignment method is able to evaluate whether or not a particular value tuple validates a task's semantics. In addition, given a partially specified tuple, the blame-assignment method is potentially able to identify possible values for the unspecified tuple members such that the tuple belongs in the relation and satisfies the semantics. If the domain relation is exhaustively described in a truth table, then the possible values are inferred through a search of this table for all the tuples that match the partially specified one. If it is evaluated by a predicate, then there are two possibilities. Either an inverse predicate is also specified in the SBF model, such that it maps the task's output to its possible inputs, or the task's input is an instance of an object type with an exhaustively described domain. In the former case, the inverse predicate is applied to the task's desired output to produce the possible values for the alternative input which could lead to its production. In the latter case, the input domain is searched for these values which, together with the desired output, would satisfy the semantics of the task. Thus, given the output desired of a task, and based on the SBF specification of its semantics the blame-assignment method is potentially able to infer the input which could possibly lead the task to produce it. Clearly, if none of the above conditions is true, then no alternative inputs can be inferred.

AB-undesired-value(*task info feedback*)
IF *feedback* belongs in the class of values *task* produces for *info*
THEN **[1]**
  IF *task* is accomplished by an operator
  THEN **under-specified-task-semantics** **[1.1]**
   IF the task semantics is an enumerated domain relation
   THEN **incorrect-domain-relation** **[1.1.a]**
  IF *task* is accomplished by a method *M*
  THEN AB-undesired-value(*task-i info feedback*) **[1.2]**
   where *task-i* $\in$ subtasks(*task M*) and *info* $\in$ output(task-i)
ELSE **[2]**
  IF there is alternative value, *v*, for info *i*, where *i* $\in$ input(task)
   for which *task* could have produced *feedback* for *info*
  THEN AB-undesired-value(task-i i v) **[2.1]**
   where *i* $\in$ output(task-i) $\cap$
   $\nexists$ *task-j*: *i* $\in$ output(task-j) $\cap$ *task-i* $\in$ subtasks(task-j)
  ELSE **over-constrained-task-semantics** **[2.2]**
   IF the violated semantics is an enumerated domain relation
   THEN **incomplete-domain-relation** **[2.2.a]**

Figure 2: The blame-assignment algorithm. The different diagnostic hypotheses that can be postulated are shown in boldface.

The semantics of `path-increase`, which specify that the produced path must be a extension of the path selected by `tmp-path-selection` (*prefix(tmp-path path)*), fails for the feedback path. The path selected in the last repetition of the loop was *((center 10th) (10th atlantic) (atlantic walnut))* and it is not a prefix of the feedback, therefore the desired path could not possibly have been produced by the `path-increase` task given the temporary path it received as input. Thus, AUTOGNOSTIC attempts to infer alternative values for the input temporary path which could enable `path-increase` to produce the desired path. The relation `prefix` is evaluated by a predicate, and the SBF model also specifies an inverse predicate for it which, given a path, produces all its possible prefixes. Given the possible prefixes of the desired path, AUTOGNOSTIC infers that if *((center 10th) (center mapple) (mapple dalney))* had been selected, the `path-increase` subtask could, in principle, have produced the feedback. Thus, the cause of the failure must lie within the subtask which selected the "wrong" path, `tmp-path-selection`. Therefore, it focuses the investigation towards identifying why this earlier subtask did not select the right path.

The blame-assignment method may reach a leaf task whose semantics is validated by both the feedback and the value actually produced for its output (see Figure 2[1.1]). This situation implies that the problem-solver's task structure is not sufficiently tailored to producing the right kind of solutions. In such situations the blame-assignment method postulates the following two types of errors as possible causes for the failure. First, the *task's semantics may be under-specified* and they allow both the actual and feedback values to be produced, when only the latter conforms with the requirements of the problem-solver's environment. In such cases, the function of this subtask should be refined (i.e., more domain relations should be added to its functional semantics), so that the overall problem-solving process becomes more selective. Second, if the task semantics refer to domain relations exhaustively described in truth tables, the blame-assignment method hypothesizes as an additional cause of the failure the *incorrect domain knowledge* of the problem solver regarding this relation which allows the mapping from its actual input to its actual output (see Figure 2[1.1.a]). This mapping could potentially be incorrect, in which case the task should have never produced the undesired actual value, and it should have preferred the feedback. Among these two hypotheses, the subsequent repair step will first attempt to address the former one, which is the more grave one, by identifying new semantics for the under-specified operator. If this is not possible, it will attempt to address the latter one.

AUTOGNOSTIC evaluates the functional semantics of `tmp-path-selection` and notices that it is satisfied by the desired temporary path. Indeed, this path belongs in the set of paths that ROUTER has already explored. Thus, AUTOGNOSTIC infers that this desired value could have been produced by `tmp-path-selection`. This task is a leaf task, and therefore, the error must lie within the operator that accomplishes it (notice, the task's semantics does not depend on any truth-table defined domain relations). That is, the specification of the `tmp-path-selection` operator is incomplete with respect to quality of solutions that is desired of the overall `path-planning` task. Indeed, the `intrazonal-search-method` performs a breadth-first search in the space of possible paths with no particular quality criterion to guide it. If there is a quality desired of ROUTER's paths, then a "best"-first search method would be more appropriate. By postulating the under-specification of the `tmp-path-selection` operator as cause for this failure of ROUTER, AUTOGNOSTIC's subsequent repair step is able to search for additional semantics with which to characterize the information transformation of this operator. As a result, it modifies this operator so as to select the shortest of the available explored paths, thus transforming the `intrazonal-search method` into a greedy, shortest-path first search.

Alternatively, the blame-assignment method may reach a task whose semantics is violated by the feedback, and no alternative values can be found for its input which can satisfy it (see Figure 2[2.2]). Under these circumstances, it postulates that the cause of the failure may be the *over-constrained task semantics* which does not allow it to produce the feedback as its output, although this value is acceptable given the information transformation intended of the overall task. In such cases the function of this subtask should be respecified in terms of other domain relations, in order to extend the class of its output values to include in it the feedback value. In addition, if the domain relations defining its semantics are exhaustively described by truth tables, the blame-assignment method postulates that the cause of the failure may be the *incomplete domain knowledge* of the problem solver regarding this relation which does not include the a tuple relating the task's actual input with its desired output, although it belongs in this relation (see Figure 2[2.2.a]).

This could have been the case in our example, if the `tmp-path-selection` semantics specified that the selected temporary path had to be the most scenic (or, for that matter, any other property that the desired temporary path does not satisfy) of the already explored paths. In this case, the blame-assignment method would have postulated that this operator was over-constrained.

Evaluation

Assigning blame is pointless unless it results in repairing the error and improving the problem solving. Indeed, whether or not repair of the fault identified by blame assignment results in improvement in problem-solving performance provides a good measure of the efficacy of the blame-assignment method. The repair of an incorrectly specified operator in AUTOGNOSTIC involves the discovery of relations that characterize the examples of behavior desired of the operator, and that differentiate them from the examples of its actual undesired behavior. As described in (Stroulia 1995), these discovered relations are used to respecify the operator's functional semantics.

In addition to ROUTER, AUTOGNOSTIC to date has been integrated with KRITIK2 (Goel 1989; Goel 1991), a design system, and REFLECS a reactive robot. In one set of experiments, AUTOGNOSTIC was tested with 8, 4, and 1 individual problems with ROUTER, KRITIK2, and REFLECS respectively. Each experiment in this set addressed a different learning task in that blame assignment identified a different kind of fault. We found that after repair the problem-solver's performance improved in each experiment. The differences among the three problem solvers (paradigm: deliberative vs. reactive; task: planning, design, navigation) provide some evidence for the generality of the SBF language and the model-based blame-assignment method.

Also, to evaluate long-term learning, in another set of experiments, AUTOGNOSTIC's integration with ROUTER was tested twice with 3 sequences of 40 randomly generated problems. For each problem, a different kind of "better" path was given as feedback. In this set of experiments, we found that AUTOGNOSTIC converged to a modified task structure of ROUTER after modifying the same three or four operators. The modified task structure was significantly superior to the original one in terms of problem-solving performance (Stroulia 1995). Collectively these experiments appear to indicate that the SBF language and the model-based method for blame assignment are appropriate for problem solvers whose behaviors can be described in terms of the interactions among a set of identifiable design elements with well-defined functionalities.

Related Research

The analysis of problem solving in terms of task structures builds on Chandrasekaran (1989). The representation language of SBF models is based on another type of SBF models that describe how physical devices work (Goel 1989). KRITIK2 uses SBF models of physical devices for diagnosis (Stroulia et. al 1992; Goel & Stroulia 1996) and for design adaptation (Goel 1991). In formulating AUTOGNOSTIC's SBF language for modeling problem solvers, we needed to make many changes to KRITIK2's SBF models. For example, we had to significantly enhance the SBF language for capturing the functional semantics of tasks.

Teiresias (Davis 1980), Gordius (Simmons 1988), Cream (Weintraub 1991), and Meta-Aqua (Ram & Cox 1994) identify knowledge faults. In addition to Lex (Mitchell et. al 1981) and Prodigy (Carbonell et. al 1989), which we have already discussed, Castle (Freed at. al 1992) too identifies processing faults. It uses a model of the problem solver that specifies the behavior expected of it, the interacting components of the problem solver, and the assumptions underlying their correct behavior. This is similar to the SBF specification of functional semantics of the tasks and subtasks of the problem solver. Like AUTOGNOSTIC, Castle's model provides a justification structure for the expected problem-solving behavior. But Castle's models lack the hierarchical organization and compositional semantics of SBF models, thus, they do not provide any guidance in searching through the inter-dependencies of the problem-solver's components. The blame assignment task in Castle is also different: given the failure of an explicitly stated assumption about the problem-solver's behavior, it identifies the component whose design assumptions support the failed expectation and postulates errors in its functioning. In contrast, given a behavior desired of the problem solver, AUTOGNOSTIC uses the functional semantics of tasks and subtasks to postulate alternative behaviors desired of them.

Conclusions

In this paper, we have described a blame-assignment method, able to identify faults in the specification of a problem-solver's operators, based on the problem-solver's SBF model. The SBF model of a problem solver captures (i) the functional semantics of the problem-solver's tasks, (ii) the compositional semantics of the methods that recursively synthesize the inferences drawn by its operators into the outputs of its overall task, and (iii) the "causal" inter-dependencies between its tasks and domain knowledge.

The SBF specification of the tasks' functional semantics plays a variety of roles in this blame-assignment method. First, the functional semantics of the problem-solver's overall task establishes the range of behaviors that the problem solver is intended to deliver, irrespective of whether or not it is explicitly designed to do so. Second, based on the functional semantics of the problem-solver's intermediate subtasks and the overall behavior desired of it, the blame-assignment method infers the behaviors desired of these subtasks. Third, by comparing the functional semantics of the problem-solver's operators with the behaviors desired of them, it identifies when the functions originally designed in these operators are incorrect (under-specified or over-constrained) with respect to the behaviors desired of the problem-solver.

The SBF specification of the compositional semantics of the methods that the problem solver uses to accomplish its overall task enables the blame-assignment method to investigate only the tasks involved in the production of the output for which an undesired value was produced. The blame-assignment method focuses the investigation from higher- to lower- level tasks and from one type of information to another, and thus, limits the number of information inter-dependencies that it examines.

Finally, based on the "causal" inter-dependencies between the tasks' functional semantics with the problem-solver's domain knowledge, the blame-assignment method is able to identify incorrect uses of this knowledge in the specification of the problem-solver's operators, and errors in this domain knowledge.

References

Carbonell, J.G.; Knoblock, C.A.; and Minton, S. 1989. Prodigy: An Integrated Architecture for Planning and Learning. *Architectures for Intelligence*, Hillsdale, NJ: LEA.

Chandrasekaran, B. 1989. Task Structures, Knowledge Acquisition and Machine Learning. *Machine Learning* 4: 341-347.

Davis, R. 1980. Meta-Rules: Reasoning about Control. *Artificial Intelligence* 15: 179-222.

Freed, M.; Krulwich, B.; Birnbaum, L.; and Collins, G. 1992. Reasoning about performance intentions. In *Proc. of the Fourteenth Annual Conference of Cognitive Science Society*, 7-12.

Goel, A. 1989. Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving, Ph.D. diss, The Ohio State University.

Goel, A. 1991. A Model-Based Approach to Case Adaptation, In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 143-148.

Goel, A.; Ali, K.; Donnellan, M.; Gomez, A.; and Callantine, T. 1994. Multistrategy Adaptive Navigational Path Planning. *IEEE Expert*, 9(6):57-65.

Goel, A. and Stroulia, E. 1996. Functional Representation and Functional Device Models and Model-Based Diagnosis in Adaptive Design. In *Artificial Intelligence in Design, Engineering and Manufacturing* (to appear).

Mitchell, T.M.; Utgoff, P.E.; Nudel, B.; and Banerji, R.B. 1981. Learning problem-solving heuristics through practice. In *Proc. of the Seventh International Joint Conference on Artificial Intelligence*, 127-134.

Ram, A.; and Cox M.T. 1994. Introspective Reasoning Using Meta-Explanations for Multistrategy Learning. Machine Learning: A Multistrategy Approach IV. (eds.) R.S. Michalski and G. Tecuci, 349-377. San Mateo, CA: Morgan Kaufmann,

Samuel, A. 1959. Some studies in machine learning using the game of checkers. IBM Journal of R&D. Reprinted in Feigenbaum and Feldman (eds): *Computers and Thought*, McGraw-Hill, New York.

Simmons, R.G. 1988. Combining Associational Causal Reasoning to Solve Interpretation and Planning Problems, Ph.D. diss, MIT.

Stroulia, E.; Shankar, M.; Goel, A.; and Penberthy, L. 1992. A Model-Based Approach to Blame Assignment in Design. In J.S. Gero (ed.) *Proc. of the Second International Conference on AI in Design*, 519-537. Kluwer Academic Publishers.

Stroulia, E. 1995. Failure-Driven Learning as Model-Based Self-Redesign, Ph.D. diss, Georgia Inst. of Technology.

Weintraub, M. 1991. An Explanation-Based Approach to Assigning Credit, Ph.D. diss, The Ohio State University.