GIT-CC-93-59

Reasoning About Function in Reflective Systems

Eleni Stroulia
Ashok K. Goel
eleni@pravda.gatech.edu
goel@pravda.gatech.edu

Abstract

Functional models have been extensively investigated in the context of several problem-solving tasks such as device diagnosis and design. In this paper, we view problem solvers themselves as devices, use functional models to represent how they work, and subsequently employ these models for performance-driven reflective reasoning and learning. We represent the functioning of a problem solver as a structure-behavior-function model that specifies how the knowledge and reasoning of the problem solver results in the achievement of its goals. We view performance-driven learning as the task of redesigning the knowledge and reasoning of the problem solver. We use the structure-behavior-function model of the problem solver to monitor its reasoning, reflectively assign blame when it fails, and redesign its knowledge and reasoning. This paper describes an architecture for reflective model-based reasoning that is capable of a broad range of learning tasks. It also illustrates reflective model-based learning using examples from the Autognostic system, a reflective path planner.

Acknowledgements

This work has been supported by the National Science Foundation (research grant IRI-92-10925), the Office of Naval Research (research contract N00014-92-J-1234), and the Advanced Projects Research Agency. In addition, Stroulia's work has been supported by an IBM graduate fellowship.

1 Reasoning About Function: A Retrospective

AI has devoted a lot of effort in constructing theories of comprehension of "how the world works". The concepts most widely used in this pursuit have been causality and function. The concept of causality has been essential for understanding the physical laws that govern the natural environment, and the concept of function has been central to reasoning about designed artifacts. Designed artifacts, such as physical devices, are teleological in nature: they have been created in order to accomplish the purposes of their designers. They are subject to the causality of physical laws just like the natural environment. Their creators designed them in such a way that the causal interactions among their components result in the accomplishment of their intended functions. Functional models and reasoning about functions exploit this relation between function and causality to reason more economically about physical devices. A functional model of a device explicitly represents the device function, and uses the device function as an encapsulation mechanism for abstracting the complex internal causal processes of the device. These models have been widely used for several tasks, such as diagnosis, design, and prediction, and in several domains such as mechanical and electrical devices [Sticklen & Chandrasekaran 1989, Umeda *et al* 1990, Abu-Hanna *et al.* 1991a, Franke 1991, Gero, Lee and Tham 1991, Price & Hunt 1992, Kumar & Upadhyaya 1992, Forbus 1993].

But AI problem solvers are designed artifacts too! Although not physical, they are devices with intended functions, structural elements, and complex internal processes. And, just like physical devices, problem solvers can fail in delivering their intended functions which may necessitate device diagnosis and redesign. This view of problem solvers as designed abstract devices offers two benefits. First, it suggests that functional models may provide a scheme for representing "how a problem solver works". That is, they might provide a scheme for representing and organizing knowledge of how the internal processes (methods, algorithms) of the problem solver result in its intended functions (tasks, goals). Second, it suggests that performance-driven learning may be viewed as a redesign task. When the performance of the problem solver is suboptimal, then the learning goal is to modify the problem solver's knowledge and reasoning in such a way that it does not fail under similar situations in the future. Thus, if a problem solver has a functional model of its own reasoning process, then, when it fails, it can use it as a guide for reflecting upon its reasoning and for redesigning itself appropriately.

In this paper, we discuss how reflection on its own reasoning and performance enables a problem solver to learn from its failures. We describe how the knowledge and reasoning can be represented in terms of a functional model, and how failure-driven reflective learning can be viewed as a model-based redesign task. We outline a taxonomy of learning tasks and sketch an architecture for reflective learning. We illustrate the process of learning using examples from the Autognostic system, a reflective path planner. In particular, we show how Autognostic can reorganize its world knowledge and modify its task structure by reflecting on its own suboptimal performance.

2 Reflection and Performance Improvement

There is always room for improvement of problem solving performance. All intelligent agents, natural or artificial, are bound to produce a suboptimal solution to some problem, or even fail in solving a problem. These failures are opportunities for learning. One way that an intelligent agent can improve its problem-solving performance is by reflecting upon these instances of failed problem solving. The important affects of reflection to learning and performance improvement have been established by a long line of psychological research [Flavell 1971, Piaget 1971, Kluwe 1982, Baker & Brown 1984]. The main result of this research is that reflection upon instances of failed problem solving enables the problem solver to reformulate the course of its own "thinking" so that, under similar circumstances in the future, it does not fail in a similar way.

The goal of our work is to develop a computational model of reflection as a self-adaptive mechanism that enables improvement of problem-solving performance. To achieve this goal, we need to address two interleaving issues:

1. a **language** for describing problem solving such that it can enable reflection, and
2. a **process** for reflection that uses the description of the problem solver to reason about its performance and redesign it to achieve better performance.

2.1 A Language for Modeling Problem Solving

Previous AI research has suggested functional analysis as the appropriate level of describing intelligent behavior. Newell [Newell 1982], for example, made a distinction between the level at which goals and knowledge are specified and lower levels at which goals and knowledge are represented in symbolic structures. He advocated that knowledge should be characterized functionally, in terms of what goals it helps to accomplish, and not structurally, in terms of particular objects and their relations. Marr [Marr 1982] too emphasized the need to separate the computational theory embodied in an intelligent system from its implementation. He advocated an analysis of information processing in terms of tasks and subtasks, and the mechanisms for accomplishing the tasks. Chandrasekaran [Chandrasekaran 1987] proposed that information-processing tasks fall into major classes called *generic tasks*. All tasks that are instances of the same generic task can be solved by the same methods. This led him to the use of task structures - task-method-subtask decomposition - as a way of building AI systems and modeling cognition [Chandrasekaran 1989]. Some of Clancey's [Clancey 1985], McDermott's [McDermott 1988], Steels [Steels 1990], and Wielinga and Breuker's [Wielinga & Breuker 1986] work also shares this functional perspective.

In our work, we have adopted Chandrasekaran's task structures as the framework for describing problem solving. A problem-solving task in this framework is specified by the information it takes as input and the information it produces as output. A task is accomplished by a method which decomposes it into a set of simpler subtasks. For each task, there may be several methods which can be potentially used to accomplish it. A method is specified

by the subtasks it sets up, the control it exercises over the processing of these subtasks, and the knowledge it uses. The subtasks into which a method decomposes a task can, in turn, be accomplished by other methods, or, if the appropriate knowledge is available, they may be solved directly. The reasoning process of a problem solver is thus described in terms of a recursive decomposition of its overall task into methods and subtasks.

We use the language of structure-behavior-function (SBF) models to describe the task structure of a problem solver. SBF models [Goel 1989] are a generalization of the Functional Representation language, originally developed for modeling physical devices [Sembugamoorthy & Chandrasekaran 1986]. Adapating this language for modeling problem solvers, we express tasks as transitions between information states. The annotations on the state transitions act as indices to the methods that are applicable to them. Methods are expressed as partially ordered sequences of state transitions which specify in greater detail how they accomplish the task for which they are applicable. Tasks which are not decomposable by any methods index the program modules that accomplish them directly. The semantics of the SBF models for reasoning task structures are shown in detail in Figure 1.

2.2 A Process Model for Reflection

The question then becomes how the comprehension of its problem solving in terms of a SBF model enables a problem solver to improve its performance.

1. By specifying a "road map" for problem solving which allows the problem solver to **monitor** the progress of its reasoning on a specific problem.

2. By specifying "correctness" criteria for the results of each of the problem solver's subtasks, so that, when it fails, the problem solver can **assign blame** for its failure to these subtasks whose results are not consistent with their corresponding criteria.

3. By guiding the problem solver to consistently **redesign** its own problem solving and thus improve its performance.

The problem solver can use the model of its reasoning to *monitor* its process on any specific instance of solving a problem. As it solves a given problem, the problem solver records which method it uses for its task, in which specific order it performs the resulting subtasks, which are the methods invoked for their respective accomplishment and what are their corresponding results. The model provides the problem solver with expectations regarding the information states it goes through as it solves the problem. For example, each information state should be related to the preceding one according to the semantics of the task carrying out the transformation between them. As it monitors its reasoning on a particular problem, the problem solver may realize that some of these expectations fail, and use that as an opportunity for learning. Alternatively, the problem solver may complete its reasoning, produce a solution, and learn from the world that another solution would have been preferable. This is yet another opportunity for learning.

$Tsk(task) :=$

$(info\_state_{input}, info\_state_{output}, \{instance\_of\}, by\_methods || structural\_elem, under\_conditions, semantic\_rels)$
where
$info\_state_{input} := \{Info\_Type\}^*$, the input information of the task.
$info\_state_{output} := \{Info\_Type\}^*$, the output information of the task.
$instance\_of := Tsk$, a task that accomplishes a transformation equivalent to or more general than the transformation of the current task.
$by\_methods := \{M\}^+$, a list of methods potentially applicable to the task.
structural_elem the name of the program module which accomplishes the task i.e. whose functional abstraction the task is. Only tasks which are not further decomposed by methods are associated with structural elements; we call these tasks "leaf tasks".
$under\_conditions := \{p(info\_state_{input})\}^*$, a set of predicates on the input information of the task, under which it is meaningful to accomplish the task.
$semantic\_rels := \{p(info\_state_{input}, info\_state_{output})\}^*$, a set of predicates that hold true between the input and the output of the task, i.e. rules that define the transformation that the task imposes on its inputs to produce its outputs.

$M(method) :=$
$(applied\_to, under\_conditions, subtasks, control)$
where
$applied\_to := Tsk$, the task to which the method is applicable.
$under\_conditions := \{p(Info\_Type)\}^*$ a set of predicates that need to be true in order for the method to be applicable to the task. The types of information on which these predicates are aplied, are all information types the values of which have been produced before the method selection.
$subtasks := \{Tsk\}^+$ a set of subtasks into which the method decomposes the task it is applied to.
$control := \{ctrl\_op(subatsks(M))\}^*$ a set of control operators applied to the subtasks of the method. Control operators define a partial order among these subtasks. They define precedence among tasks, potentiall parallelism, and repetitions of tasks until a condition is met.

$WO(worldobject) :=$
$(domain, attrs, id\_test)$
where
domain := the data structure with the legal values for the object; only the enumerated objects have domains, and usually this is the case for the objects that directly refer to objects in the world, as opposed to conceptual objects.
$attrs := \{(name, def, type)\}^*$, the set of attributes characteristic of the world object. Each one of them is specified in terms of a name, a definition of how evaluating its value given an instance of the world object type, and its type which can be either another world object or a number.
$rels := \{(name, table)\}^*$, the set of domain relations applicable to the world object. Each one of them is specified in terms of a name, and an association table where the knowledge of the problem solver regarding this relation resides.
id_test := the definition of a function to evaluate identity in the domain of the world object.

$Info\_Type :=$
$(is\_a, input\_to, produced\_by, syntactic\_type)$
$is\_a := WO$ a world object, of which this type of information is an instance.
$input\_to := \{Tsk : Info\_Type \in info\_state_{input}(Tsk)\}^*$ a list of tasks consuming the information.
$produced\_by := \{Tsk : Info\_Type \in info\_state_{output}(Tsk)\}^*$ a set of tasks that can potentially produce the information as output.
$syntactic\_type := \{simple, multiple\}$ specifying whether this information type consists of one or a set of world objects. Eventually, this should be expanded to handle more complex information types, in addition to elements and lists.

Figure 1: The language of SBF models

Once a failure has occurred, the problem solver can use the record of its failed reasoning process, and the model of its problem solving to *assign blame* for its failure to some element(s) of its task structure and propose modifications which can potentially remedy the problem. The task-structure view of problem solving gives rise to a taxonomy of learning tasks each one corresponding to a different type of potential cause of failure that the problem solver can identify. We will discuss this taxonomy in greater detail later.

After having decided on a learning task, the problem solver can *redesign* its task structure according to the modifications that this task suggests. The modifications may be as simple as integrating a new fact in the body of its knowledge about the world, or as complex as introducing a new task in its task structure. In any case, the semantics of the SBF models for problem-solving task structures can guide the problem solver in its modification process, so that the result will be a valid task structure.

There is no guarantee that the modification will result in an improved problem solver. However, it can be evaluated through subsequent problem solving. If the problem that triggered the modification can now be solved and the appropriate solution produced, this is strong evidence that indeed the modification was appropriate. If not, the problem solver may try other modifications or it may try to evaluate why the modification did not bring the expected results, assuming that it has a model of its reflection process. Reflection, after all, is a reasoning task, and as such it can be modeled in terms of SBF models, just like any other problem-solving task.

2.3 An Architecture for Reflective Learning

Figure 2 diagrammatically depicts the architecture we have developed for reflective learning. In this architecture, the problem solver has both reasoning and meta-reasoning capabilities. Figure 2 shows the different types of knowledge that the problem solver possesses, their organization, and their use. Tasks are depicted as solid-line, tilted boxes, knowledge is depicted as dashed-line boxes, control flow is shown by double arrows, input and output information flow is depicted by simple arrows, and access and use of knowledge by tasks is depicted by double-headed arrows.

In the reasoning space, the problem solver has domain knowledge. It may have a world model which contains knowledge about specific objects in the world and the relations between them. It may also have a case memory which consists of experiences of solving particular problems in the world. The problem solver uses these types of knowledge to reason about the world and solve new problems in it. At the reasoning level the problem solver does not have explicit knowledge about its problem-solving knowledge or reasoning, and thus it cannot reason about it.

At the meta-reasoning level, however, the problem solver understands its own reasoning in terms of a SBF model. The problem solver understands that it knows several methods that can be used to achieve its task. Each one of these methods decomposes the overall problem-solving task in different sets of subtasks, and uses different types of knowledge. One method may use the world model and the other may use the case memory. The problem solver has also explicit knowledge about the ontology on which its world model and case memory are

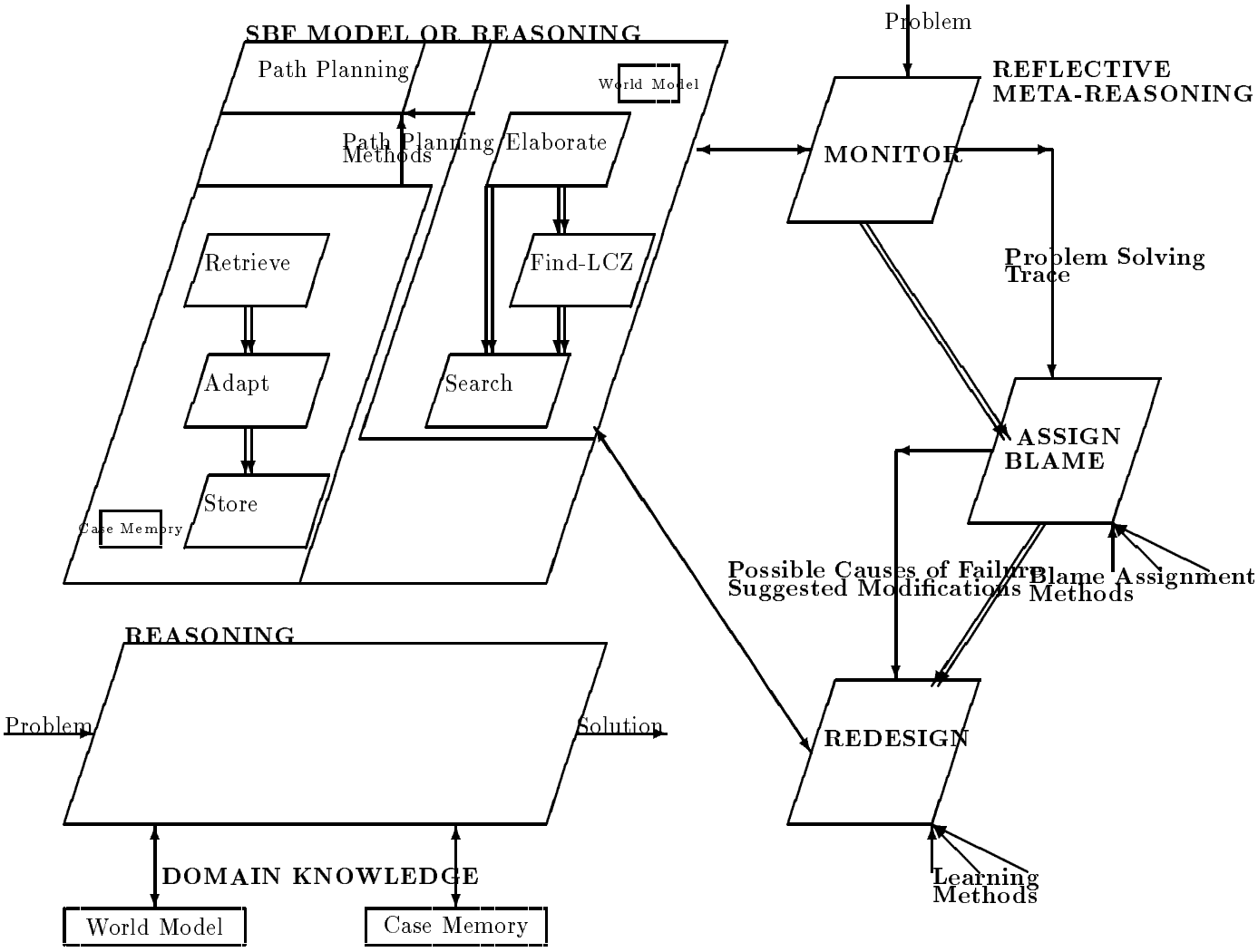

Figure 2: The Reasoner's Functional Architecture

based and their respective organizations. The problem solver uses the knowledge it has at the meta-reasoning level to monitor its reasoning, assign blame to some element of its reasoning process it when it fails, redesign it, and thus learn and improve its performance.

3 Router: A Case-Study Problem Solver

In our work, we use Router, [Goel *et al.* 1991], a path planning system, as a case-study problem solver. Router's task is to find a path from an initial location to a goal location in a physical space. It knows two methods that can achieve this task: a model-based method, and a case-based method. When Router is presented with a problem, it chooses one of them based on a set of heuristic rules. These rules evaluate the applicability and utility of each one of these methods on the particular problem at hand.

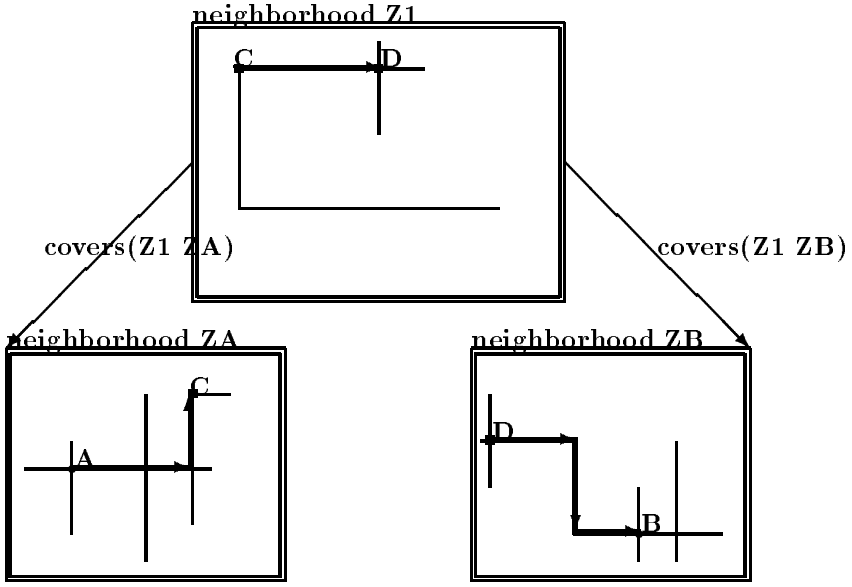

Figure 3: The Hierarchical Organization of Router's World Model

Router's world knowledge consists of a set of streets in the navigation space, the valid directions along which they can be traversed, and the intersections between them. The streets are grouped into neighborhoods and the neighborhoods are organized in a space-subspace hierarchy. The higher-level neighborhoods contain knowledge of a few major streets, and their intersections, and describe large spaces. The neighborhoods at the next level decompose a high-level neighborhood into several subspaces. The lower-level neighborhoods cover spaces smaller than the high-level neighborhood but contain knowledge of major and minor pathways that fall in their spaces. Figure 3 depicts the hierarchical organization of Router's world model. Each neighborhood is shown within a rectangular, dashed-line box. The high-level neighborhood *Z1*, depicted on the top of the figure, is decomposed in a set of sub-neighborhoods, two of which, *ZA* and *ZB*, are shown in the figure. The same intersection

can belong in more than one neighborhoods, because the same physical space is represented at several levels of detail, thus intersection *C* belongs both in *Z1* and *ZA*, and intersection *D* belongs in *Z1* and *ZB*. When Router uses the model-based method to find a path, it first finds the neighborhoods of the initial and goal intersections. If the two intersections belong in the same neighborhood, Router does a local search to find a path between them. Otherwise, it connects the initial intersection to a major pathway that belongs in the neighborhood immediately above the initial neighborhood. Then, it traverses the high level neighborhood upto a point that belongs both in the high level and in the goal neighborhood, and finally it connects this point to the goal intersection. For example, to connect the locations *A* and *B* in neighborhoods *ZA* and *ZB* correspondingly, Router would find a path in *ZA* connecting *A* to *C*, and then a path in *Z1* connecting *C* to *D* and finally a path in *ZB* connecting *D* to *B*.

The case-based method requires a memory of cases of path-planning problems. Each case in the memory contains knowledge about an initial and a goal intersection, the neighborhoods they belong to, and a path between them. When Router uses the case-based method to solve a problem, it first finds a case in the memory that connects two intersections in the same neighborhoods as the current initial and goal intersections. Then, it finds a path between the current initial intersection and the retrieved path's initial intersection. Similarly, it finds a path between the retrieved path's goal intersection and the current goal intersection, and connects the resulting paths. Finally, it stores the new problem as a case in its memory for future use.

Router was not developed specifically for the purposes of this work; thus, originally it did not have a model of its own problem solving. On top of Router, we have developed Autognostic which has a SBF model of Router's reasoning and which is also capable of the reflection process described above. Router and Autognostic together constitute a reflective reasoner and learner.

3.1 The SBF Model of Router's Path Planning

Figure 4 depicts a part of the SBF model of Router's problem solving. The problem-solving process is viewed as a sequence of transformations between information states. The SBF model specifies the information transformations that occur as Router's problem solving progresses. Each information state is depicted as a rectangular box, and contains the information available at the state. Each state transformation is depicted by a double arrow, and is annotated by the description of the task which accomplishes the transformation. The task description is shown below the double arrow of the corresponding transformation.

Router's task is `route planning`, i.e. to produce a path from an initial location to a goal location. Thus its overall information transformation takes as input information state, `info-state-1`, which contains the initial and goal locations, and the output information state, `info-state-5`, contains the path between them. The SBF model describes the conditions under which this task can be accomplished, i.e., when the initial and destination locations are different. Moreover, the model specifies the semantic relations that this task imposes on its output information state with respect to its input information state, i.e., that the initial node of the produced path should be the same point as the initial location, and the final node

of the path should be the same point as the goal location. Finally, the model specifies that the route-planning task can be accomplished by the `route-planning method`.

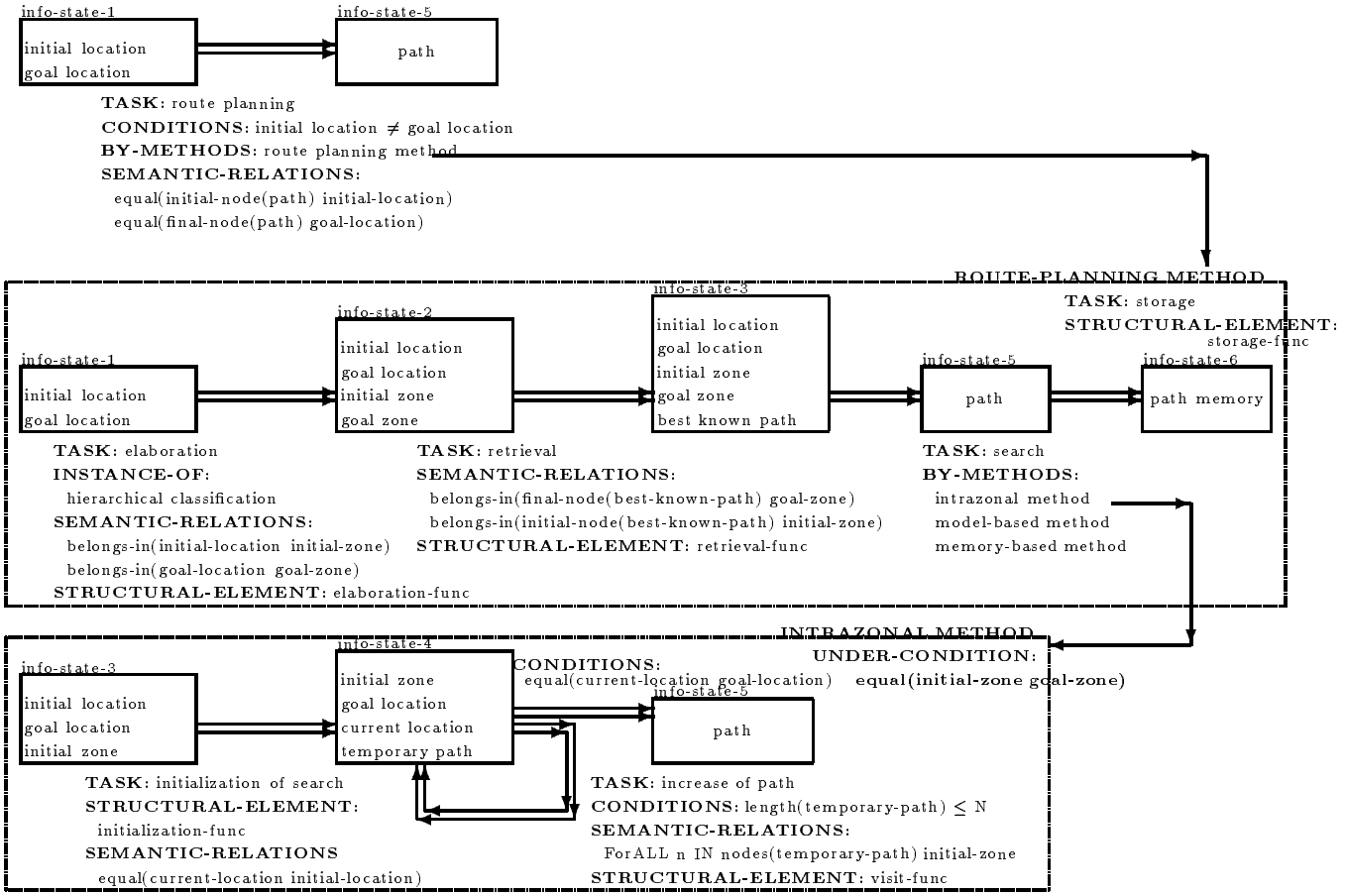

Figure 4: Fragment of Router's SBF model, instantiated in the context of a specific problem, explaining how the failed solution was produced

The route-planning method, shown in the first dashed-line box in Figure 4, decomposes the task into a set of simpler subtasks. These tasks, i.e. `elaboration`, `retrieval`, `search` and `storage` correspond to the four state transformations in this box. The elaboration subtask transforms `info-state-1` to `info-state-2`; at `info-state-2` Router knows, in addition to its input information, the `initial-zone`, and the `goal-zone`, i.e., the two neighborhoods in which the initial and goal location belong. The SBF model makes explicit the relations that the elaboration subtask imposes between its input and output information states: (i) the initial location belongs in the initial zone, and (ii) the goal location belongs in the goal zone. The model also specifies that the elaboration task is an instance of the generic task hierarchical classification, which makes possible the transfer of problem-solving methods from this generic task to its instance. The elaboration subtask is not decomposed by any method, but is directly accomplished by the elaboration function.

The next information transformation is from `info-state-2` to `info-state-3` and it is accomplished by the `retrieval` task. The retrieval task recalls from the planner's memory of previous planning experiences a path which connects locations spatially close to the initial and goal locations of the current problem. The `search` subtask actually produces the desired path, in transformation from `info-state-3` to `info-state-5`, and subsequently the `storage` subtask stores it in the memory for possible reuse in the future. The search task can be solved by three different methods, the `intrazonal`, the `model-based`, and the `memory-based` methods. However, Figure 4 depicts only the representation of the intrazonal search method. The SBF model explicitly specifies that this method is applicable under the condition that the initial and goal zones are identical.

The sequence of information transformations that explains how the intrazonal method accomplishes the `search` task is shown in the dashed-line box in the bottom of the Figure 4. In this sequence, `info-state-3` is transformed into `info-state-4` by the task `initialization of search`; at this state, Router has set up its current location to the initial intersection, and initialized its temporary path to contain only this intersection. The SBF model shows how `info-state-4` is repeatedly transformed by the `increase-of-path` subtask. This subtask incrementally adds additional intersections to the temporary path. As defined by the semantic relations of the `increase-of-path` subtask, all the nodes in the temporary path belong in the initial zone. The task is repeated under the condition that the length of the temporary path does not exceed N. If, at some point, the current location equals the goal location, then `info-state-4` becomes the final information state `info-state-5` and Router outputs the temporary path as the solution desired of it.

The SBF model of the problem solver's reasoning captures the interdependencies between the different tasks it can perform, its knowledge and its problem-solving methods. The model represents the internal information states of the system and the conditions under which the system transitions from one to another, in terms of method applicability and task conditions. The model is non-deterministic; there may be several methods for any given task, the order of the subtasks in a method is only partial and some tasks may sometimes be unnecessary. The model also specifies the logical relations between the different information states, in terms of task semantic relations.

4 A Taxonomy of Learning Tasks

The task-structure view of problem solving gives rise to a taxonomy of causes of failures, and to a corresponding taxonomy of learning tasks. In this section, we discuss each one of these causes of failure and the learning tasks they lead to, using specific examples from Router's reasoning. Figure 5 shows a part of Router's world-model necessary for following the examples. The small circles in the model are either initial or goal intersections for the example problems we discuss.

4.1 Knowledge Errors

The failure of the problem solver to solve a problem may be due to errors in its internal knowledge about the world. Its knowledge may be incorrect or incomplete. Or, alternatively, its knowledge may be inappropriately organized so that although it has all the facts necessary to produce the correct solution for a problem, it cannot access them appropriately to produce the desired solution. Finally, the representation language it uses for describing the objects in the world may be inappropriate, and thus it may not be able to even acquire all the relevant information regarding a specific world object.

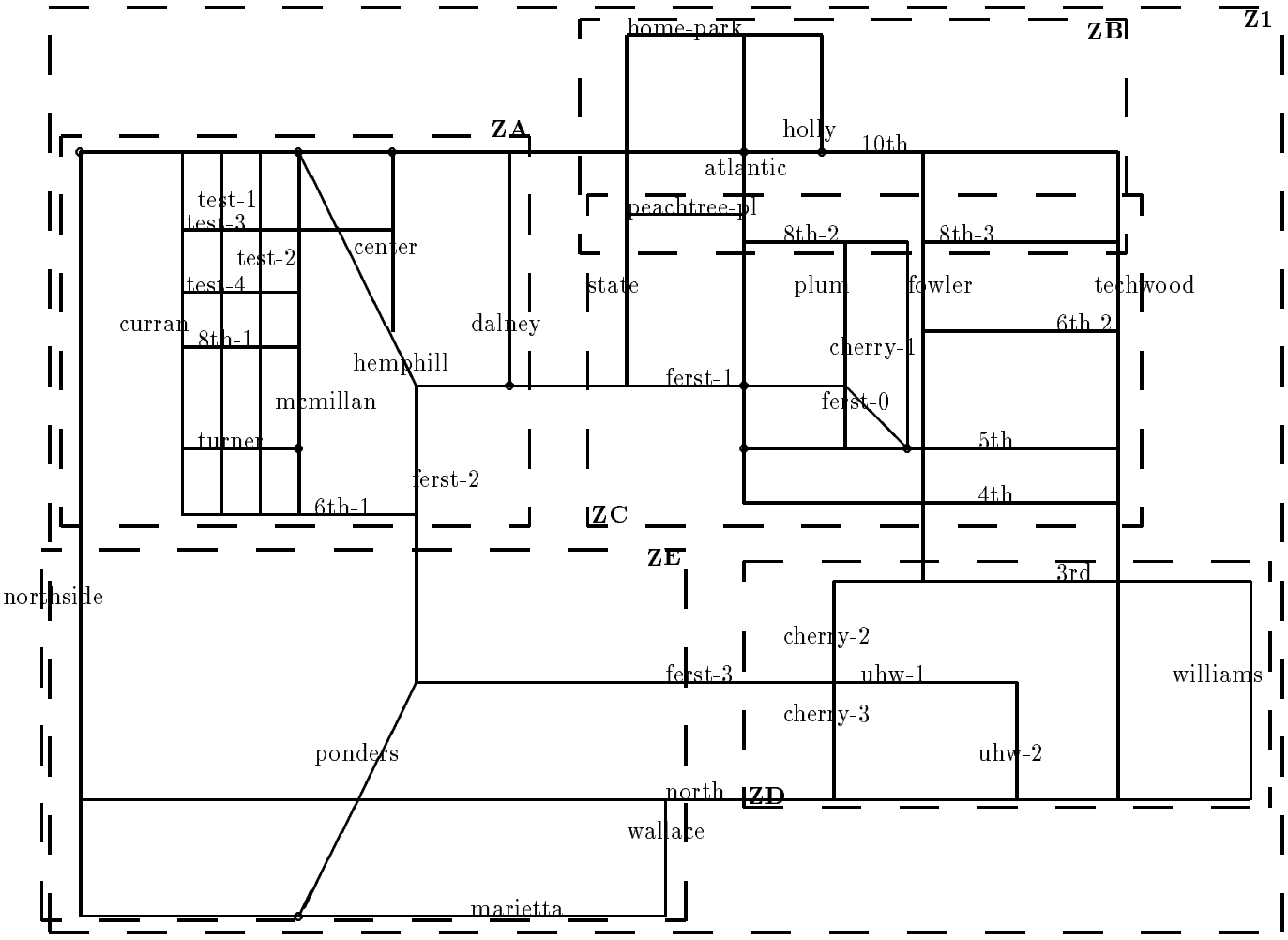

Figure 5: Router's world

Incomplete and Incorrect World Knowledge Router is presented with the problem of going from *10th & hemphill* to *10th & atlantic*. It elaborates the problem and finds that both intersections belong in neighborhood *z1*. The problem then is solved by a local search in this

neighborhood. Router produces the path *(10th hemphill SouthEast ferst-1) (hemphill ferst-1 East atlantic) (ferst-1 atlantic North 10th)* .[1] Although the above path is correct, it is suboptimal. A better path is provided as feedback to Autognostic: *(hemphill 10th East hemphill)* .

Router represents the street *10th* in its model as having only one valid direction, *West*. Because Router believes that *10th* is a one-way street, it produces the above suboptimal path. The feedback provides the information that *East* is also a valid direction for *10th*. The error lies in the incorrect representation of a world object in Router's model. In this case Autognostic has to update and correct Router's world model. The updated world-model will represent *10th* as a two way street.

Incorrect Knowledge Organization Router organizes its knowledge of neighborhoods hierarchically. Each neighborhood at some level, is decomposed into several neighborhoods at the lower, more detailed, level. Neighborhoods at the same level may overlap. Thus, two intersections may both belong to more than one neighborhood and the problem of finding a path between them may be solvable in more than one neighborhood. For example, in the case of going from *10th & center* to *dalney & ferst-1*, the problem can be solved both in *za* and *z1*. If solved in *z1* the resulting path is *(10th center East atlantic) (atlantic 10th South ferst-1) (ferst-1 atlantic West dalney)* ; if solved in *za* the path is *(10th center East dalney) (10th dalney South ferst-1)* . The second path is preferable, since it is shorter and less expensive to produce.[2]

Autognostic has to reorganize Router's neighborhood hierarchy in such a way that (i) not many conflicts of the above type occur, where a pair of intersections belong in more than one neighborhood at the same time, and (ii) each pair of intersections belongs to that neighborhood where the solution of the problem of finding a path between them is easiest.

In this example, Autognostic should decrease the detail of information that the neighborhood *z1* contains, in order to make the problem solvable only in *za*. This could be achieved by removing the intersection *10th & center* from neighborhood *z1*.

Incorrect Knowledge Representation Router describes each intersection with two attributes the names of the two streets that meet at the intersection, (*street_name_1 street_name_2*). It describes each street with four attributes the neighborhood in which the street belongs, its name, the directions along which traversal of the street is allowed, and the set of streets with which it intersects, ordered along the first of its valid directions (*neighborhood street_name direction* {*street_name*}*). Router is pre-

[1] A path is a sequence of path-segments, where the ending intersection of a path-segment is the beginning intersection of the next. A path-segment is a quadruple where the first, second and fourth elements are names of streets, and the third element is the relative direction between the beginning and ending intersection of the segment. The path-segment begins at the intersection of the first and second streets and ends at the intersection of the second and fourth streets. The segment lies on the second street. For instance, the segment *(tech-parkway ponders South marietta)* lies on *ponders*, begins at *tech-parkway & ponders*, and ends at *ponders & marietta*. *ponders & marietta* is *South of tech-parkway & ponders*.

[2] Shorter here refers to the length of the path, where cost of production refers to the amount of steps that the local search took to produce the path. The two need not correlate.

sented with the problem of going from *10th & northside* to *mcmillan & turner*. Router returns the path *(northside 10th East curran) (10th curran South 6th-1) (curran 6th-1 East mcmillan) (6th-1 mcmillan North turner)* . Although the above path is correct, it is suboptimal. A better path is provided as feedback to Autognostic, i.e., *(northside 10th East hemphill) (10th mcmillan South turner)*.

Router's concept of intersection is that an intersection is a location where two streets meet, and thus, at each intersection it has only two choices on which direction to go. Thus, Router misses the opportunity of going *10th mcmillan South turner* when at *10th & hemphill*. Because in each path, the ending intersection of one path-segment is the same as the beginning intersection of the next path-segment, the feedback provides the information that *10th & hemphill* and *10th & mcmillan* are the same intersection, and thus *mcmillan* is a street that passes through *10th & hemphill*. In this case the representation of intersection in Router's world-model is incorrect.

Autognostic has to modify Router's representation of intersections from (*street_name_1 street_name_2*) to a representation that does not limit the number of streets that meet at the intersection to two. Essentially Autognostic has to generalize its concept of intersection. This change in Router's representation of intersections, implies changes to its world-model to reflect the fact that *10th*, *mcmillan*, and *hemphill* meet at the same point.

4.2 Method Errors

A task can be potentially achieved by several different methods. For each problem, the problem solver has to select an applicable method. Each method decomposes the task into a set of subtasks. Each method contains knowledge of the subtasks that need to be achieved for the overall task to be achieved, and a partial ordering among these subtasks. Often, the failure of the problem solver may be due to the choice of the particular method for the problem at hand, or potentially to its incorrect understanding of what subtasks the method decomposes the overall task into, and in what order these subtasks should be accomplished.

Erroneous Choice of Method Router is presented with the problem of going from *10th & atlantic* to *ponders & marietta*. The best case Router knows for this problem is a case that connects *atlantic & ferst-1* and *ponders & north* through the path *(atlantic ferst-1 West hemphill) (hemphill ferst-1 West ferst-2) (ferst-1 ferst-2 West ponders) (ferst-2 ponders South north)* . This case matches the current problem on the neighborhoods of the initial and goal intersections. Router chooses the memory-based method over the model-based method only when the best available case matches the current problem on at least one intersection. Thus, for the current problem, Router chooses the model-based method. It produces the path *(10th atlantic South ferst-1) (atlantic ferst-1 West hemphill) (hemphill ferst-1 West ferst-2) (ferst-1 ferst-2 South ponders) (ferst-2 ponders South north) (north ponders South marietta)* .

This path produced by Router contains the path that the case-memory presented as a ball park solution to the problem. Using the model-based method, Router produced the same path that it would have produced using the case-based method. Feedback informs Autognostic that

the cost of solving the two problems of going from *10th & atlantic* to *atlantic & ferst-1* and from *north & ponders* to *ponders & marietta* is less than the cost of solving the problem of going from *10th & atlantic* to *ponders & marietta*.

In this example, Autognostic has to change the heuristic rules Router uses for choosing among methods. These rules take into account the efficiency of each method in each one of the different classes of problems it is applicable to, the quality of the output they produce, the knowledge they need to be applicable, and the specific problem instance that is to be solved. As more and more problems get solved, the heuristic rules may be refined, as the problem solver's understanding of the applicability and efficiency of the methods improves.

Incorrect or Incomplete set of subtasks Router is presented with the problem of going from *atlantic & ferst-1* to *ferst-2 & ponders*. The best-case it has available in its memory for this problem is a case connecting *atlantic & ferst-1* with *ponders & north* through the path *(atlantic ferst-1 West hemphill) (hemphill ferst-1 West ferst-2) (ferst-1 ferst-2 West ponders) (ferst-2 ponders South north)* . Because the old case matches the new case in the initial intersection, Router chooses the case-based method to solve the problem. For this problem, since the initial intersections match, Router has simply to find a path between the old and the new goal intersections, and compose it with the old path. The resulting path is *(atlantic ferst-1 West hemphill) (hemphill ferst-1 West ferst-2) (ferst-1 ferst-2 West ponders) (ferst-2 ponders South north) (north ponders North ferst-2)* . The path is suboptimal because it traverses the same street *ponders* twice in two different directions.

This error occurs because the overall path is a composition of independently derived paths. Although neither the paths in the case-memory nor the paths that the model-based method produces are redundant, their composition may be redundant. This error can be found only after the overall path is produced and before it is stored in memory. Essentially, Autognostic needs to learn that another subtask, i.e. *evaluation in terms of redundancy*, should be added to the set of subtasks into which Router's case-based method decomposes the task of finding a path.

Incorrect ordering of the subtasks The model-based method decomposes the search task into (i) finding a neighborhood which covers both the neighborhoods of the initial and goal intersections (ii) finding the common intersections between the initial neighborhood and the common neighborhood, `common-initial-ints`, (iii) finding the common intersections between the goal neighborhood and the common neighborhood, `common-dest-ints`, (iv) finding a path between the initial intersection and `int1`, an intersection in the `common-initial-ints`, (v) finding a path between `int1`, and an intersection in the `common-dest-ints`, `int2`, (vi) finding a path between `int2` and the goal intersection, and (vii) compose the paths that tasks (iv) (v) and (vi) produced. If the initial selection of `int1` and `int2` leads to the failure of any of the tasks (iv) (v) and (vi) to produce a path, new intersections `int1` and `int2` must be chosen.

The higher the level of a neighborhood, the less detailed it is. The less detailed a neighborhood, the fewer pathways it knows of and the more difficult it is to find a path between

two intersections. As a result, the local search in a neighborhood is more likely to fail the higher the level of the neighborhood. Thus task (v) is more likely to fail than the other two local search tasks (iv) and (vi). The ordering (v), (iv) and (vi) is preferable then, because when task (v) fails Router can search for another pair of intersections `int1` and `int2` to connect the two lower-level neighborhoods, without investing resources in connecting the initial intersection with `int1`. Autognostic needs to notice the bad performance of Router, identify the cause of it to be a problem in the interactions among the subtasks that the method decomposes the `search` task into, and then reconfigure the task structure appropriately.

4.3 Task Errors

Another kind of potential cause of failure is the incorrect problem solver's understanding of the information transformation accomplished by a task. For example, the problem solver may not know all the input information that is necessary for the accomplishment of a task, or it may not know that an existing task can easily produce, in addition to its current output, some other information, of finally it may not have a correct understanding of the semantic relations between the input and the output of a task.

Incorrect Task Input When Router uses the model-based method for the search task, one of the subtasks it has to accomplish, is to find the common intersections between the high-level neighborhood and the two lower-level ones. The task of `identification-of-common-intersections`, a subtask of the `model-based` method for the `search` task, takes as input the two neighborhoods and produces as output a set of common intersections which can be used as connecting points between the two. Depending on the specific problem at hand a different intersection from the set may be more appropriate to be used as a connecting point.

For example, to connect *5th & fowler* with *holly & 10th* Router has to select an intersection common between *zc* and *z1* and an intersection common between *z1* and *zb*. If it chooses as `int1` *atlantic & ferst-1* then it will produce the path *(fowler 5th West atlantic) (5th atlantic North ferst-1) (ferst-1 atlantic North 10th) (atlantic 10th East holly)* . However there is a better path *(5th fowler North 8th-3) (8th-3 fowler North 10th) (fowler 10th West holly)* which could have been produced if Router had chosen as `int1` *fowler & 8th-3*.

The problem is that Router ignores the actual initial and destination intersections in deciding on common intersections between the neighborhoods. In this case, Router has to learn that the subtask of `identification-of-common-intersections` should take as input the initial and goal intersections, in addition to the neighborhoods, and it should prefer these common intersections which lie on the same street with one or both these two locations.

Incorrect Task Output Router's task is path planning, i.e. it produces a path between two given locations. Let us consider a situation where Router is required to interact with a scheduler, which needs to assign delivery tasks to different agents. The scheduler needs to provide each agent with a route for its delivery task, and it, also, needs an estimate of the

time to traverse the route in question, in order to predict when this agent will be available again for a new assignment. In this situation, in order to accommodate the scheduler needs, Router could "redefine" its path planning task in order to include in its output another type of information, i.e. the time to traverse the produced path. Such a modification also implies modification of the problem-solving methods that accomplish the path planning task, in order to include a task which will produce the additional output information.

Incorrect Task Semantics Router is presented with the problem of going from *5th & ferst-0* to *atlantic & 5th*. It produces the path *(ferst-0 5th West plum) (plum 5th West atlantic)* . Although the above path is correct, it is suboptimal because it traverses the path segment *(5th ferst-0 West atlantic)* in two steps, where it can traverse it in one, thereby reducing the cost of producing the requested path.

The task of searching for a path locally in a neighborhood gets decomposed into a repetitive execution of the task of finding a path segment from Router's current location to a new location, `increase-of-path`. This task takes as input the `current-location`, and produces as output the path segment that is traversed, and the new current-intersection. The current-intersection of the output needs to be immediately next to the current-intersection of the input. Thus when presented with the problem of going from *5th & ferst-0* to *atlantic & 5th*, Router has to perform the task twice to connect the two intersections. This is an error in the semantics of a task.

In this case Autognostic has to modify the semantics of the `increase-of-path` subtask. In our example, the task should produce as output current-intersection any intersection that lies on the same street as the input current-intersection, not just the one immediately next to it. The change in the task semantics may imply changes to the task decompositions by the different methods applicable to it, or, if the task is directly achieved by a programming module - as in this case -, reprogramming of that module.

5 Using the SBF Model of Problem-Solving for Learning: An Example

In this section we will describe in detail Router's reasoning for a specific problem. In this problem, feedback from the world informs Router that its solution to the problem is not optimal. Thus Autognostic reflects on Router's reasoning using the SBF model of its problem solving as a guide, identifies the cause of the failure and proposes two alternative adaptations to Router's task structure. We discuss both adaptations and show how they improve Router's planning performance.

5.1 Monitoring the Problem Solving Process

Router is presented with the problem of connecting *10th & center* with *dalney & ferst-1*. Autognostic monitors Router's planning process and generates its trace. The trace is a partial instantiation of the SBF model of Router's problem solving. The model is non-deterministic,

in that several methods are possibly applicable to the `search` task, and the subtasks that each of these methods sets up are only partially ordered. However, when solving a specific problem, Router decides on one specific method (each time it has a choice), and executes the subtasks in some order. Thus only a part of the model will be instantiated, the part which explains the subtasks actually executed during the process of solving the specific problem. In this case, the trace is the instantiation of the part of the SBF model depicted in Figure 4, where each one of the different types of information is instantiated with the specific values produced during the particular planning session.

Router uses its `route-planning` method to solve the task, and sets up its corresponding subtasks. The `elaboration` subtask produces as output *z1* as the value of the `initial-zone` and the *z1* as the value of the `goal-zone`. The next task is `retrieval`, which returns no path similar enough to the current problem from Router's memory. When the `search` task is performed, the `intrazonal` method is chosen, because its applicability test, i.e., equality of the initial and goal zones is true, for the current values of these types of information, *z1* = *z1*. Thus the `intrazonal` method is applied to the `search` task which is decomposed into the `initialization-of-search` and the `increase-of-path` subtasks. The repeated execution of the latter produces the path *(center 10th East atlantic) (10th atlantic South ferst-1) (atlantic ferst-1 east dalney)* which is returned as the output of the overall route-planning task.

5.2 Assigning Blame for Producing a Suboptimal Solution

The path that Router produced is correct. However it is suboptimal, because it is too long compared to the path *(center 10th East dalney) (10th dalney South ferst-1)* which is presented to Router as feedback from the world.

Autognostic uses the feedback, the trace of Router's reasoning on the specific problem, and the SBF model of Router's problem solving, to identify the cause of its failure. This model-based blame-assignment process is a heuristic search through the task structure of Router's problem solving, guided by the feedback and the problem-solving trace. In this example, the feedback contains the information that, during its planning process, Router produced a sub-optimal value for its output information, `path`. Moreover, the feedback specifies the desired value for this output information. The blame-assignment process regresses this optimal value and its properties back through the SBF model and identifies modifications to the problem solver's knowledge, or task-structure that could result in the production of desired value.

The specifics of the blame-assignment process is shown in Figure 6. From the SBF model of Router's path planning, Autognostic identifies the subtask `increase-of-path` which should have produced the correct path (feedback-path). Based on the semantic relations that this subtask imposes on its input and output, as specified by the model, it infers the sequence of intermediate information states that could have led to the production of that path. This process, in our example, concludes that the desired value of the `path` could potentially be produced if the elaboration task had produced as values for the `initial-zone` and `goal-zone` *za* and *za* correspondingly. The process also evaluates that these values could are consistent with the semantic relations of the `elaboration` subtask, as are the values actually produced, i.e. *z1* and *z1*. At this point, Autognostic knows that the `elaboration` task can potentially

ASSIGN-BLAME-SUBOPTIMAL-VALUE(path,
actual value *(center 10th East atlantic)*
(10th atlantic South ferst-1)
(atlantic ferst-1 east dalney)
feedback value *(center 10th East dalney)*
(10th dalney South ferst-1))
PRODUCED(path) = { increase of path }
SEMANTIC-RELATIONS(increase of path):
ForAll n IN nodes(path) belongs-in(n initial-zone)
Semantic-Relations hold TRUE for actual values of *path* and *initial-zone*
Semantic-Relations DO NOT hold TRUE for feedbcak value of *path* and actual value of *initial-zone*
INFERRING ALTERNATIVE VALUE FOR *initial-zone*
ForAll n IN nodes(feedback path) belongs-in(n za) $\Rightarrow$ feedback-value(initial-zone) = za

ASSIGN-BLAME-SUBOPTIMAL-VALUE(initial-zone,
actual value *z1*
feedback value *za*)
PRODUCED(initial-zone) = {elaboration }
SEMANTIC-RELATIONS(elaboration):
belongs-in(initial-intersection initial-zone)
Semantic-Relations hold TRUE for actual values of *initial-zone* and *initial-intersection*
Semantic-Relations hold TRUE for feedbcak value of *initial-zone* and actual value of *initial-intersection*

$\Longrightarrow$ `elaboration` can produce either *za* or *z1* for value of `initial-zone`

Figure 6: Blame Assignment using the SBF model of Router's Path-Planning process

produce either *za* or *z1* as values for the initial and goal zones, because the domain relation `belongs-in(intersection zone)` is not one-to-one.

In situations where "a leaf task can produce alternative values" Autognostic has two possible modifications actions to choose from: (i) modification to domain relation, in order to refine the domain relation which allows multiple mappings so that it allows only the preferred mappings, and (ii) insertion of selection task, in order to enable the problem solver, when multiple mappings are possible, to select the preferred one. We will illustrate how each of these modifications affect Router's problem solving.

5.3 Redesigning the Problem Solver

Modification to Domain Relation The motivation behind modifying the domain relation *belongs-in* is that it allows mappings, and in the problem currently being solved one of these mappings was chosen and an inappropriate path was produced, where there was an alternative mapping which could have led to the preferred path. Thus, if the relation is modified to allow only the mapping that is consistent with the preferred path, the preferred path might have been produced.

In the specific example, the modification is instantiated as exclusion of the intersection *10th & center* from the set of *belongs-in(?X z1)*.

When a domain relation is modified, the consequences of this modification are also propagated to other domain relations which are interrelated with it. As we mentioned above, in Router streets are described as $(neighborhood\ \ street\_name\ \ direction\ \ \{street-name\}^*)$ tuples. Thus the exclusion of *10th & center* from *belongs-in(?X z1)* also means that *center* street is removed from the set of streets intersecting with *10th* street in zone *z1*.

Insertion of a Selection Task The motivation behind inserting a selection task after the elaboration task in Router's task structure is to enable Router to reason about the two possible values for `initial-zone` and select the most appropriate one.

The SBF model of the problem solver's reasoning, as shown in Figure 1, explicitly specifies the ontology of the problem solver's domain. For each type of information that its tasks consume and produce, the SBF model specifies what type of world object it is. Moreover, for each type of world object, among other things, the model specifies the domain relations which are applicable to it. Autognostic uses this knowledge, along with the specific values (actual and preferred) of the information type to be selected, to discover a relation which can be used to differentiate between these values. If there is such a relation, then Autognostic can use it as a semantic relation for the new task to be inserted in the task structure.

In our example, Autognostic knows that one domain relation applicable to zones is the `covers` relation. Given the actual and the alternative values for the `initial-zone`, *z1* and *za* correspondingly, Autognostic notices that *covers(z1 za)*. It then hypothesizes that this can be used as a differentiating criterion between possible alternative values for the `initial-zone`. Thus it inserts in the set of subtasks of the `route-planning` method, after `elaboration`, the `selection-after-elaboration` subtask, with input `initial-zone`, output `selected-initial-zone`, and semantic relation

`covers(initial-zone selected-initial-zone)`. The new task has as a goal, given a specific path-planning problem, to reason about the possible values of the `initial-zone` in the context of this problem and select the one which is covered by the rest of them, that is the most specific one.

In more general terms, a newly inserted task in the problem solver's task structure has as a goal to reason about the possible values of some type of information in the context of a specific problem and select the most appropriate one for the given problem. Thus, the selection-task insertion implies the discovery of a characteristic property of the information type to be selected which will enable the problem solver to discriminate among the possible values of this information, and select the most appropriate one for the given problem.

In order for the problem solving task structure to be consistent after this modification, Autognostic needs to perform some more modifications in addition to the insertion of the `selection-after-elaboration` task: (i) change the syntactic type of the `initial-zone` from a simple zone element to a list of elements, (ii) change (reprogram) the function `elaboration-func` to actually return appropriately a list of values instead of a single one, and (iii) modify the elements of the task structure which originally were after the `elaboration` task to use instead of the information type `initial-zone` the `selected-initial-zone`. Autognostic can autonomously perform modifications (1) and (iii) but not (ii), which is currently performed by a human programmer, at the suggestion of Autognostic.

5.4 Evaluating the modified Problem Solver

After Router's process is modified, Autognostic evaluates the appropriateness of the revision by presenting Router with the problem that led to failure before. As Router solves the same problem once again Autognostic goes back again to its monitoring task. In our example, for both the modifications, Router produces the desired path this time, so any one of these modification can be evaluated as successful. Had Router failed, once again, to deliver the desired path, Autognostic would have another learning opportunity, and it would repeat its blame-assignment-and-learning task.

6 Discussion

Reflective reasoning has received much attention in AI [Davis 1977, Davis 1980, Mitchell *et al.* 1989, Kuokka 1990, Freed *et al.* 1992, Ram & Cox 1992]. The focus of this paper, however, is the use of functional models in reflection. Hence, here we only compare our work to other research which has investigated the uses of functional models of abstract devices [Allemang 1990, Weintraub 1991, Johnson 1993].

Allemang has used the Functional Representation framework to model computer programs, and has shown how such models can be used for program verification. Weintraub has used the Functional Representation framework to model the operation of a diagnostic system, and has shown how such a model can be used for assigning blame when the knowledge-based system produces an incorrect diagnosis. Autognostic too uses the SBF model for blame

assignment but its process for assigning blame is different from Weintraub's. In his work, the information-state transitions associated with each task of the knowledge-based system are annotated by associative rules which indicate the likely sources of error. Moreover, the knowledge-based system being diagnosed has a deterministic task structure, i.e. each task is accomplished by a single method. In contrast, Autognostic admits the applicability of multiple methods for accomplishing the same task. It uses the derivational trace of problem solving in conjunction with the SBF model to identify the sources of error.

In Johnson's work, the functional model of a student's potential problem-solving behavior is used by a tutoring system to provide help. She monitors the student's actions, uses the functional model to form hypotheses about the method being used by the student, and thus provides context sensitive help. Autognostic's monitoring of Router's behavior is similar to the tutor's monitoring of the student.

Autognostic takes the idea of modeling problem solvers a step further: it is able to redesign the problem solver after identifying the causes of its failure. Its SBF model provides the vocabulary for indexing repair plans that correspond to different types of failure causes. In addition, the semantics of the model enable the modification of the planner in a manner that maintains the consistency of problem solving.

In this paper, we showed how Autognostic reorganizes Router's model of the world, and how it modifies its task structure. We have already tested the model reorganization capabilities of Autognostic. We completely removed the original hierarchical organization of Router's world model, and let Autognostic induce a neighborhood hierarchy. The efficiency of the planner and the quality of its solutions with the new model were comparable to the original one. In future work, we plan to investigate other kinds of modifications, such as reorganization of existing tasks in the problem solver's task structure, modification of method selection criteria, and knowledge acquisition.

7 Conclusions

In this paper, we described how a problem solver's knowledge and reasoning can be modeled in the language of structure-behavior-function models within the framework of task-method-subtask structures. The language of SBF models describes the problem solving process as a non-deterministic sequence of transformations to which the input of the problem solver's task is subjected in order to produce the output of the task.

We also described a process model for reflective reasoning. We showed how the reflective process uses the SBF model of the problem solver's reasoning for three tasks:

- In **monitoring**, the SBF model of the problem solver provides a language for interpreting the problem solver's reasoning steps, and makes explicit their logical relations.
- In **blame assignment**, the SBF model of the problem solver with the trace of the reasoning on a specific problem, enables Autognostic to localize the cause of the failure to some element of the problem solver's task structure. Moreover, the language of SBF models provides an vocabulary for indexing redesign methods.

- In **redesign**, the task-structure framework for problem solving gives rise to a taxonomy of learning tasks, i.e. modifications to the problem solver's reasoning that can help improve its performance. Moreover, the semantics of the language of SBF models enable the modification of the problem solver in a manner that maintains the consistency of problem solving.

We showed how Autognostic's functional models of Router's knowledge and reasoning enable it to reorganize Router's world knowledge and modify its task structure by inserting new tasks. Although we described the functional models and the reflective processes in the context of a particular problem solver, a planner called Router, neither the models nor the processes are particular to Router or to path planning.

References

[Abu-Hanna *et al.* 1991a] Abu-Hanna, A., Benjamins, V.R., and Jansweijer, W.N.H. (1991a) Device Understanding and modeling for Diagnosis. In *IEEE EXPERT*, 6(2):26-32.

[Allemang 1990] Allemang, D., (1990), *Understanding Programs as Devices*, PhD Thesis, The Ohio State University.

[Baker & Brown 1984] Baker, L., and Brown, A.L., (1984) Metacognitive skills of reading. In D. Pearson (ed.), *A Handbook of reading research*, New York: Longman.

[Chandrasekaran 1987] Chandrasekaran, B., (1987) Towards a functional architecture for intelligence based on generic information processing tasks, In *Proc. of the Tenth IJCAI*, 1183-1192, Milan.

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

[Clancey 1985] Clancey, W.J., (1985) Heuristic Classification, *Artificial Intelligence*, 27(3):289:350

[Davis 1977] Davis, R., (1977) Interactive transfer of expertise: Acquisition of new inference rules, *Artificial Intelligence*, 12:121-157.

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

[Flavell 1971] Flavell, J.H., (1971) First discussant's comments: What is memory development the development of? *Human Development* 14:272-278.

[Franke 1991] Franke, D.W (1991) Deriving and Using Descriptions of Purpose, In *IEEE Expert* April.

[Freed *et al.* 1992] Freed, M., Krulwich, B., Birnbaum, L., and Collins, G. (1992) Reasoning about performance intentions. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, 7-12.

[Forbus 1993] Forbus, K. (1993) Qualitative Reasoning about Function: A Progress Report. In *Working Notes of Reasoning About Function Workshop, AAAI 93*, pp 31-36.

[Gero, Lee and Tham 1991] Gero, J.S., Lee, H.S., and Tham, K.W., (1991), Behaviour: A Link between Function and Structure in Design, *Proceedings IFIP WG5.2 Working Conference on Intelligent CAD*, pp. 201-230.

[Goel 1989] Goel, A., (1989) Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving, PhD Thesis, The Ohio State University.

[Goel *et al.* 1991] Goel, A., Callantine, T., Shankar, M., and Chandrasekaran, B., (1991) Representation, Organization, and Use of Topographic Models of Physical Spaces for Route Planning. In *Proc. Seventh IEEE Conference on AI Applications*, 308-314, Miami Beach FL, IEEE Computer Society Press.

[Johnson 1993] Johnson, K., (1993), *Exploiting a Functional Model of problem Solving for Error Detection in Tutoring*, PhD Thesis, The Ohio State University.

[Kluwe 1982] Kluwe, R.H., (1982). Cognitive Knowledge and Executive Control: Metacognition. In D. R. Griffin (ed.), *Animal Mind - Human Mind* Springer-Verlag, Berlin.

[Kumar & Upadhyaya 1992] Kumar, A. and Upadhyaya, S., (1992) Framework for function-based Diagnosis. technical Report, State University of New York at Buffalo, Buffalo NY 14260.

[Kuokka 1990] Kuokka, D.R., (1990) The Deliberative Integration of Planning, Execution, and Learning, Carnegie Mellon, Computer Science, Technical Report CMU-CS-90-135.

[Marr 1982] Marr, D., (1982) *Vision: A Computational investigation into the Human Representation and Processing of Visual Information*, W.H. Freeman, San Francisco.

[McDermott 1988] McDermott J., (1988) Preliminary steps toward a taxonomy of problem-solving methods, In Automating Knowledge Acquisition for Expert Systems, Sandra Marcus (ed.), Kluwer Academic Publishers.

[Mitchell *et al.* 1989] Mitchell, T., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., Schlimmer, J., (1989) Theo: A Framework for Self-Improving Systems. In *Architecture for Intelligence*, Lawrence Erlbaum.

[Newell 1982] Newell, A., (1982) The Knowledge level *AI Journal* 19(2):87-127.

[Piaget 1971] Piaget, J., (1971) *Biology and Knowledge* Chicago: University of Chicago Press.

[Ram & Cox 1992] Ram, A. and Cox, M. T., (1992) Introspective Reasoning Using Meta-Explanations for Multistrategy Learning. To appear in *Machine Learning: A Multistrategy Approach IV*, R. S. Michalski and G. Tecuci (eds.), Morgan Kaufmann, San Mateo, CA.

[Price & Hunt 1992] Price, C.J. and Hunt, J.E., (1992) Automating FMEA through multiple models, In *Research and Development in Expert Systems VIII*, I. Graham (editor), pp 25-39.

[Sembugamoorthy & Chandrasekaran 1986] Sembugamoorthy, V. and Chandrasekaran, B., (1986) Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems, *Experience, Memory and Problem solving*, J. Kolodner and C. Riesbeck(editors), Lawrence Erlbaum, Hillsdale, NJ, pp. 47-73.

[Steels 1990] Steels, L., (1990) Components of Expertise AI Magazine 11(2):30-49.

[Sticklen & Chandrasekaran 1989] Sticklen, J., and Chandrasekaran, B, (1989) Integrating classification-based compiled level reasoning with function-based deep level reasoning. In *Causal AI Models: Steps towards Applications*, W. Horn (editor), Hemisphere Publishing Corporation, pp 191-220.

[Umeda *et al* 1990] Umeda, Y., Takeda, H., Tomiyama, T., Yoshikawa, H.: 1990, Function, Behaviour, and Structure, *Applications of Artificial Intelligence in Engineering, vol 1, Design, Proceedings of the Fifth International Conference*, Gero (editor), Springer-Verlag, Berlin, pp. 177-193.

[Weintraub 1991] Weintraub, M., (1991), *An Explanation-Based Approach to Assigning Credit*, PhD Thesis, The Ohio State University.

[Wielinga & Breuker 1986] Wielinga, B.J., and Breuker, J.A., (1986) Models of expertise, In *proc. Seventh European Conference on Artificial Intelligence*, 306-318, Brighton.