Towards an Automated Empirical Assistant

Dawn E. Gregory and Paul R. Cohen
Experimental Knowledge Systems Laboratory
University of Massachusetts, Amherst

It is not suprising to realize that all sciences have much the same goal: given some kind of system, develop an understanding of its *behavior* in terms of its task, environment, and architecture[2]. Although techniques may vary widely, all of the sciences utilize the same *scientific process*. First, the scientist must develop one or more *hypotheses*, which provide questions to be asked about behavior. In order to answer these questions, an *experiment* is designed. The experiment defines a *protocol* for collecting supporting data, which is then carried out. Finally, the data is *analyzed* to determine if the questions have been adequately answered. In this paper, we look toward developing an intelligent agent which can support the scientist in each of these stages.

Although science itself can be decomposed into simple steps, the fact remains that individual sciences have different approaches to each one. Specific knowledge of a field is irreplacable in the hypothesis generation phase. The types of questions asked and the techniques used to test them are largely dependent on the field. These issues pose serious problems for an automated assistant: as often happens, it is primarily *knowledge* that is needed, and that knowledge can only be obtained from (human) experts.

Of all the sciences, computer science should be the easiest to automate. First of all, it is very predictable: a system will always behave the same way under the same conditions.[1] Second, it is very accessible: it is not difficult for a computer program to collect experiment data when the subject is another computer program. Finally, it is very malleable, in that we can easily substitute one component for another in order to obtain comparisons. For these and other reasons, we will first automate experiments in various areas of computer science, extending to other sciences in future work.

In this paper, we describe a hypothetical agent which assists the scientist throughout the empirical cycle. The agent is intended to provide four major capabilities: (1) a method for gathering task-specific knowledge which places minimum burden on the user; (2) automated support for tedious tasks, such as data collection and analysis; (3) an ongoing notebook which keeps track of a scientific project from beginning to end; and (4) a topic memory, which retains facts about subjects, tasks, measurements, and exceptions, in order to reduce the future burden on the scientist.

In section 1, we describe the agent's methodology for representing scientific problems and the corresponding interface to the scientist. Section 2 shows how the agent can use this information to design and carry out experiments. Section 3 describes an experiment in which the assistant works with a computer scientist to solve a problem in computation theory.

[1] The **exact** same conditions, of course. If some kind of randomness were introduced, we should not believe that the same conditions hold.

1 Problem Representation

Since the goal of the scientist is to develop a complete model of a system, it is sensible that the automated assistant use models as its primary form of representation. Given that the modelling schema is sufficiently rich, the assistant will continue to update its model as new information is derived. The possibility of using a single modelling paradigm for all phases of scientific study is among the most interesting issues in this project.

1.1 Modelling a System

The first step is to develop an approach to modelling the systems that scientists study. The model must be capable of representing the four aspects of a system: its behavior, its task, its constriants, and its architecture. If we observe that any system basically converts its task into its behavior, then it seems natural to model the system as a *function*. The four aspects of a function map directly to those of a system: a function uses its mechanisms (architecture) to convert its inputs (task) into its output (behavior), in the presense of constraints. Thus, we will use the technique of *function modelling*[????].

In order to fully understand a system, it must be observed at many levels of detail. Specifically, the scientist may wish to decompose the system into sub-systems, each with its own parameters, in order to study the components and their interactions. Thus, we will require the assistant to develop hierarchies of models that represent the system at various levels of abstraction.

We will use graphic diagrams to work with the function models. This will provide an intuitive user interface and simplify the task of decomposing the models. An example of such a diagram is shown in figure 1. In this diagram, we have the program's inputs coming in from the left, its results leaving from the right, time and space constraints at the top, and the architecture at the bottom.

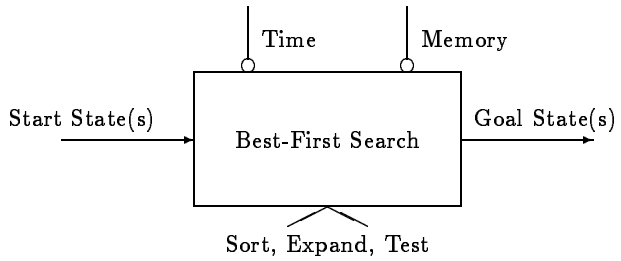

Figure 1: The top-level function model for a best-first search algorithm

When a block in a model is decomposed, its components must maintain the connections (i.e. inputs, outputs, constraints, mechanisms) of the original block. This means that: (a) all connections must be connected to some "child" block; and (b) any other connections must be created and consumed within the detailed model. For example, consider the **current state** and **successor state(s)** connections in the more detailed model shown in figure 2.

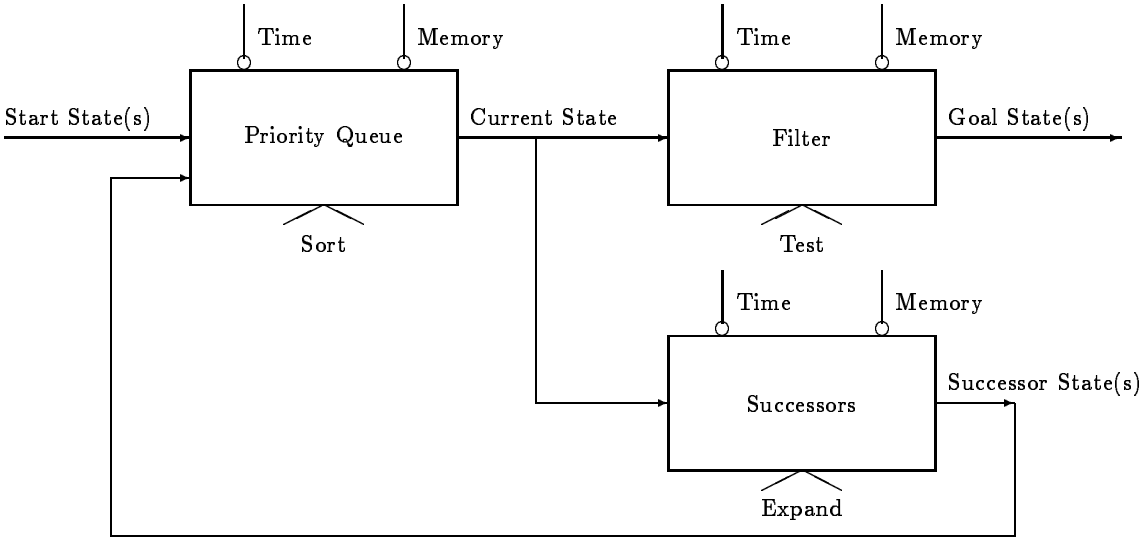

Figure 2: The decomposed function model for a best-first search algorithm

1.2 Specifying Measurements

If we wish to represent the complex behavior of a system, we will certainly need to do more than label connections and decompose the model; namely, we will need to define *measurements* and record the values they produce. In the search example, we would want to measure the **value** of the Start State(s) and Goal State(s), the **size** of Memory, the **length** of time, and the **type** of mechanism for sorting, expanding, and testing states.

The assistant will provide the ability to define one or more *attributes* for each connection. Each attribute defines a *variable* that can be used in an experiment. When a variable is dependent, we must know how to *measure* the variable. When a variable is independent, we must know how to *generate* the appropriate feature, if possible.[2] Once the assistant knows these things it is well on the way to producing experiments.

2 Using the Models

Although function modelling is a natural representation of systems, it is useless unless it can support the four phases of an experiment. Fortunately, the isomorphism between systems and functions is so strong they are easy to accomodate. In addition, the model provides a concise structure around which to build knowledge of generic scientific procedures as well as specific topics.

The key problem in experiment design is to decide whether each variable is dependent, fixed, or independent. The assistant can rely on cues from the scientist, recorded in the function model, to solve this problem. For example, the first series of experiments with a system will

[2]On computers, it is easy to accomplish this task. However, it is sometimes not possible to generate the desired feature or even to know which variables are independent.

normally use the inputs as independent variables, the outputs as dependent variables, and fix the constraints and mechanisms. These will be followed by experiments with dependent constraints and fixed outputs, or independent mechanisms and fixed inputs. Thus as the model evolves, we will use its "status" to guide the selection of experiment hypotheses.

There are several types of knowledge that must reside in the assistant's knowledge base. First, there are *project plans*, which dictate the proper sequence of experiments for a particular type of system. These plans use *experiment templates* to generate the experiment protocol. Experiment protocols are instantiated with values from the function model, as well as those stored in the *topic memory*, which is the repository for knowledge gained from previous experiments. Developing this knowledge base will be the major bottleneck in designing the empirical assistant.

Finally, the most critical feature of the assistant is its ability to generate intelligible output. This will take the form of an interactive notebook, which stores all of the information relevant to a project: model definitions, hypotheses and the rational behind them, collected data, analysis results, knowledge resources used and updated, notes, and so on. It will be necessary to implement the notebook feature early, so we can track the assistant's progress as it develops.

The infrastructure created by the function model, knowledge base, and interactive notebook provides the context in which to design an experiment. Given this context, the assistant can provide a high level of support for each of the four phases of an experiment. In the following sections, we describe in detail how each phase will be treated.

2.1 Generating Hypotheses

The first step in any experiment is to define one or more hypotheses to be confirmed or disproven. These hypotheses generally take two forms. First, an hypothesis may make a *qualitative* statement about relationships inherent in a system; these are often statements about causal dependencies between variables. Second, an hypothesis may make a *quantitative* statement about a qualitative relationship, by positing a functional characterization of the relationship. In general, quantitative hypotheses are generated as results of qualitative experiments; qualitative hypotheses can be derived from pilot experiments or from the function model.

Function models and causal models

Each block in a function model makes a statement about the dependencies between the parameters of a system, as shown in figure 3. In this context, the notion of a dependency is equivalent to that of a *causal relationship*, satisfying three conditions:

1. **Covariance:** the variables are highly correlated.
2. **Control:** no other variable causes the strong correlation between the variables.
3. **Precedence:** one variable explicitly precedes the other in some partial ordering.

It is relatively easy to see that a function model induces causal relationships between the connection variables. First, the fact that a pair of connections are attached to the same block satisfies the *covariance* condition. Second, all of the incoming connections are assumed to be at

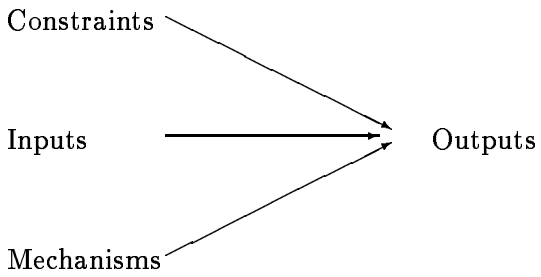

Figure 3: The default causal model for a function model.

the same "level", so we assume they cannot affect the other relationships, thereby establishing the *control* condition. Finally, all incoming connections are logically "upstream" of the output connection, conforming to the *precedence* condition.

Qualitative hypotheses

The causal model shown in figure 3 indicates six qualitative hypotheses. The first three are induced directly from the links in the model:

1. **Inputs** affect the **outputs**.
2. **Constraints** affect the **outputs**.
3. **Mechansims** affect the **outputs**.

The other hypotheses come from the notion of *conditional dependence* in the model; when two variables are causes of another variable, the two are said to be conditionally dependent given the third. Thus, we have three more hypotheses:

1. **Inputs** and **mechanisms** are conditionally dependent given the **Outputs**.
2. **Inputs** and **constraints** are conditionally dependent given the **Outputs**.
3. **Constraints** and **mechanisms** are conditionally dependent given the **Outputs**.

If any of these hypotheses are shown to be false, we know that the default causal model is not accurate. This inaccuracy indicates that there is more relevant detail to be accomodated in the function model, through the process of decomposition. For example, suppose we discover that the **constraints** do *not* affect the **outputs** (link hypothesis 2 fails). In order to resolve this finding, we need to remove the link between the constraints and the outputs; however, we need the constraints to remain attached to the causal model. We have two choices: make constraints the cause of some other variable in the model, or introduce a new variable to mediate the relationship. The latter approach is well suited to function modeling — introducing a new variable simply means that the model has well-defined functional components.

At this point, it is important to stress the relationship between qualitative hypotheses and the process of decomposing a function model. When a function model is decomposed, the

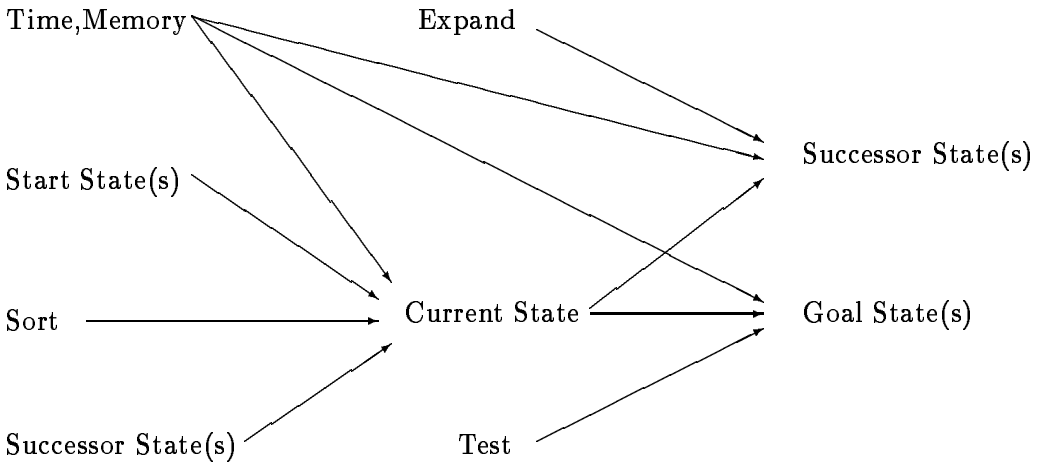

Figure 4: The causal model for the best-first search algorithm.

associated causal model becomes more complex. This is primarily due to the following: when a function model is decomposed, each connection must be passed on to every component in the new model, *unless we have a priori knowledge that it should not be.* Due to this restriction, the new model may have many links that are unnecessary. To see this, consider the causal model in figure 4, which was extracted directly from the function model in figure 2.

This causal model is complicated, and many of the connections are superfluous. Although we may believe that the inputs, outputs, and mechansims have been assigned correctly by the scientist, the constraint connections have been simply passed on to each component. We can attempt to remove some of these connections by introducing qualitative hypotheses about the existence of each connection.

Quantitative hypotheses

Once experiments have derived a good qualitative model of the system, the next step is to develop quantitative descriptions of each relationship. As qualitative hypotheses are derived from the structure of a model, quantitative hypotheses are derived from its attributes.

Different quantitative hypotheses are selected by specifying which connections will be dependent, fixed, or independent variables in the experiment. It may seem that we are "putting the cart before the horse", in making this decision before generating the hypothesis; however, the causal assumptions of the function model provide this information. Given that we know the status of each of the variables in the experiment, the problem of generating an hypothesis becomes a *sampling* problem: the issue is no longer *what* to test, but *how* to test it. This issue is discussed in the next section.

2.2 Designing Experiments

Once an hypothesis has been selected, the experiment design can be generated directly from an *experiment template*; these templates will reside in the assistant's knowledge base. Developing these experiment templates is not a major issue; however, instantiating the template poses some interesting problems and opportunities for the automated assistant.

There are several items that must be included in the experiment template. First, we will need space for the experiment hypothesis and its rationale. Next, we need to describe the management of the variables: which values are sampled for the independent variables, what value is assigned to the fixed variables, how the dependent variables are measured. Finally, we need to decide how to analyze the data and what results we expect. Once filled in, the experiment template becomes a set of instructions for completing the experiment.

Sampling the independent variables

The first issue we encounter when instantiating an experiment template is the *granularity* of the independent variables. In rare situations, a variable will have a small number of values which can be sampled exhaustively. Unfortunately, most variables will not provide this luxury and we must sample the values at larger granularity. In this case, we would like the assistant to do some extra work to select appropriate values for the experiment at hand.

There are several standard techniques for sampling an input space, but they all rely on the idea of *partitioning* the space into classes. In the simplest case, we create three classes: low-extreme, average, and high-extreme. If we use exactly one value from each class, we have reason to believe we are sampling the space fairly. Unfortunately, variables are not always ordered and spaces are not always nicely distributed. This provides an additional challenge for the automated assistant.

One possible solution to the sampling problem is for the assistant to generate sampling distributions for the independent variables, and then use them to select appropriate input samples. For example, when a variable can grow from 0 to infinity, the assistant will run a fully factorial experiment for several small values. The data are then analyzed to determine the relationships between values and the behavior as values change. This information can then be used to select values that are likely to be interesting.

Control conditions

As discussed in [2, Chapter 3], the most crucial feature of experiment design is to establish that appropriate *control conditions* are considered. Generally, it is quite difficult to select control conditions. However, the assistant only knows what is stored in its function model, and the function model does a reasonably good job selecting control conditions. This is largely due to its modular nature: the assistant need not consider variables that are "outside" (i.e. not connected to) each block in the model. Thus, the selection of control conditions should be a straightforward extension of the function modelling paradigm.

2.3 Running Experiments

In order to run an experiment from the function model, the scientist will need to provide the necessary "hooks" into the true system. As we mentioned above, computer science is simpler to automate because it is easily accessible. Consider all the bits of code we write to test our algorithms: input generators or pre-processors, analysis routines, data storage, looping over values, and so on. EKSL has developed a package called CLIP which will run a series of experiment trials when given the appropriate parameters. CLIP will provide the basic execution model for the automated assistant.

Consider again our example, the algorithm that performs a best-first search. The decomposed function model shows three components that need to be defined: Priority Queue, Filter, and Successors. In order to run experiments at this level of detail, the scientist must provide code for each of these functions; the types of the parameters and the result must coincide with the function model. In addition, the scientist must define functions to generate input values from the independent variables and to measure the dependent variables. CLIP then uses these *alligator clips* to run a series of experiments.

Unfortunately, the assistant cannot write the code for the clips. However, there are two ways in which the function models can support the scientist in defining experiment parameters. Since the definition is attached to the function model, it is available throughout the lifetime of the project. In addition, this information can be stored for future projects, since many of the components are re-usable. Thus, the agent will construct a knowledge base of program fragments and control strategies for those programs.

The knowledge base will be constructed out of function model components and their associated definitions. The best approach is to store the components that are well explained at many levels of granularity. In order to get to the lowest level, we will eventually want to provide a programming language, as in Scott Anderson's MESS simulator[1]. There are significant opportunities in this. First, if we simply use MESS and its language, the tool will be invaluable to a large population of researchers in Artificial Intelligence. This integrated system would allow a scientist to create an algorithm, design an execution environment, and then let the assistant take over and run comprehensive experiments.

The other opportunity in developing a programming language around the function model is of broader scope. Current research in model checking focuses on complete characterization of the dynamic behavior of computer programs.[3] Although these algorithms are necessarily exponential, we can improve their performance with abstraction. If the model language is logically sound, it becomes the core of a "theory" from which we can derive a theory of function models. This would enable the agent to truly assist in the development of programs.

2.4 Analyzing Results

The final phase of the experiment is to use the collected data to confirm or refute the hypothesis. Automated assistance for this analysis is an ongoing research project in our lab and many others. Rob StAmant is developing an agent called AIDE[4] which uses planning and visualization to assist in exploratory data analysis. The techniques and tools developed for AIDE will provide analysis capabilities for the empirical assistant.

The assistant can work closely with AIDE during analysis. First, the experiment design can be used to guide AIDE in selecting analysis plans. Then, if AIDE determines that more data is needed, the assistant can develop a new experiment to collect it. Finally, AIDE's findings can be recorded into the function model for presentation to the user.

3 An Example

In this example, we consider a problem from the field of Theoretical Computer Science. Specifically, the scientist is studying the behavior of a *ring network*. A ring network is a distributed processing environment, consisting of several processors $\{P_0 \ldots P_{P-1}\}$, and network connections between the processors. Since the processors are arranged in a ring, each has exactly two connections: one to its **left** neighbor P_{i-1} and one to its **right** neighbor P_{i+1}. The goal of the scientist is to understand the behavior of this network over a variety of task domains, and ultimately to prove an upper bound on the time required to complete the task.

Figure 5 shows the function model representation of a ring network. This diagram supplies the assistant with a problem description that is on par with the one provided in the preceding paragraph.

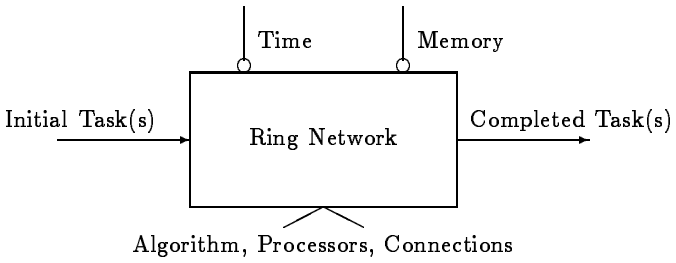

Figure 5: The top-level function model for a ring network.

At this point, the scientist has two options: assign attributes to the connections or decompose the model. Since attributes will be carried through to the lower-level model, we will assign attributes first. Table 1 defines the attributes for each of the top-level connections.

Connection	**Attribute**	**Values**
Initial Task(s)	number	$n \geq 1$
Completed Task(s)	correct	$x \in \{Yes, No\}$
Algorithm	code	λ = LISP Program
Processors	number	$P \geq 2$
Connections	number	$C = 2$
Time	length	$t \geq 0$
Memory	size	$m \geq 0$

Table 1: Attributes for the top-level ring network.

In this example, the scientist again wants to prove a limit on a distributed algorithm;

however, the constraining parameter is no longer the form of the data but the architecture of the network. In a ring network each processor is connected to exactly two others, its neighbors. Processors not only evaluate algorithms but also pass jobs to each other. The model for this problem is slightly different from the last, in order to reflect these changes.

This diagram tells the assistant that we are working with an allocation strategy. Since we want to evaluate behavior over an entire space of inputs, we may want to vary the **size** or **structure** of the data. Since a specific algorithm will be used for each step, it is a fixed point in the problem. If we believe the algorithm is correct, we may also take the result as a fixed point. Finally, we will want to vary the number of processors P and record the time t required to solve each instance.

There is still one important feature missing from the description of the problem: how are inputs allocated to the P processors? Thus, we will want to decompose the top level diagram to show the inner connections of the problem. The second level diagram is shown below:

Much of this diagram could be automatically hypothesized by the agent. First, if we define the attribute **number** to be *special*, the assistant could interpret **Processors (number=P)** as "there will be **P** copies of **Processors**, and they will be identical". If the agent also knew that **Processors** often used an **Algorithm** and took **time**, these connections could be automatically created. Finally, it is reasonable to believe that either the **Data** or **Result** are somehow transformed before connecting to the **Processors**; since the **Result** is a fixed point, the agent could assume a single unknown block which converts the data to proper input for the processors. Thus it is left to the scientist to label the new block and connect the time constraint to it.

The final phase of defining the experiment is to select the independent variables and define a means of emulating the desired system. In order to proceed quickly, the scientist decides to first sample the input space. The **generating function** will use a probability feature p; this value is an independent variable in the range $0 \leq p \leq 1$. The input **size** will be measured from the data that is generated. The **number** of processors will be independently varied over the trials. The fixed **algorithm** for the processors must be defined, as is the **allocation** algorithm. If the algorithm is correct, the **Result** can be ignored, leaving only **Time**, which is a special variable updated during the simulation process, as described above.

The analysis of the data is also directly suggested by the design. There are 4 significant variables: p, n, P and t. Since we know p and P are independent , t is dependent, and n and P are not directly related, there are only 3 possible models:

1. $P \rightarrow t, p \rightarrow t, n \rightarrow t$
2. $P \rightarrow t, p \rightarrow n \rightarrow t$
3. $P \rightarrow t, p \rightarrow n, p \rightarrow t$

Thus, we can ask two questions: Is p independent of t given n (2)? Is n independent of t given p (3)?

Now the assistant performs the following experiment: Fix P at a small number and vary p over its range. If $I(p, t \mid n)$ posit (2) else if $I(n, t \mid p)$ posit (3) else posit (1). These relationships suggest new experiments. When $p \rightarrow t$, there is reason to suspect the sampling model provided. One approach is to provide new sampling models that may eliminate this

dependency. Alternatively, the experimenter may wish to evaluate the sampling model against the complete input space. This technique is discussed in a later example.

Once we have shown $I(p, t|n)$, we can prove the overall hypothesis that $t = O(n/P)$. When $n \to t$, we need only sample a variety of values for n and P, measure corresponding t, and show that tP/n is a constant value. A statistically significant result will confirm the hypothesis.

References

[1] Scott D. Anderson. A simulation substrate for real-time planning research: Thesis proposal. Technical Report 94-84, University of Massachusetts, Amherst, MA, 1994.

[2] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[3] Yuri Gurevich. Evolving algebras: An attempt to discover semantics. In G. Rozenberg and A Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 266–292. World Scientific, 1993.

[4] Robert St.Amant. A mixed-initiative planning approach to exploratory data analysis. PhD Thesis Proposal. Computer Science Department, University of Massachusetts, Amherst, 1994.