# Knowledge Representation in Automated Scientific Discovery

**Susan L. Epstein**

Computer Science Department, Hunter College and The Graduate Center of The City University of New York
695 Park Avenue, New York, NY 10065, USA
susan.epstein@hunter.cuny.edu

**Abstract**

Scientific discovery is the most ambitious form of problem solving, one that attends to sets of examples rather than individual ones. Its goal is to identify novel, useful information that further develops a body of scientific knowledge. This paper considers the building blocks of automated scientific discovery and ways to control their interactions.

Given the right *representation* (way to describe knowledge), discovery becomes nearly effortless. For example, two classic papers demonstrate that new representations are the key to understanding an entire class of problems (Amarel, 1968; Anzai and Simon, 1979). More recently, a subject, intrigued by a challenge posed in a lecture, formulated 7 representations of the game tree for a simple game in 24 hours. He discovered, with the final one, a simple way to extract a perfect strategy for an entire class of games (Epstein, 2004). In all three cases, discovery was driven by new representations. Further evidence comes from great mathematicians who retrospectively chronicle their own discoveries (e.g., (Hardy, 1972; Pascal, 1964; Poincaré, 1970)). They often describe visual imagery that re-represented knowledge they had already been pondering. Chess masters have reported similar phenomena. These new representations are often replete with flashing lights or bright colors that focus attention in just the right place, much like the "aha!" experiences of cartoon characters. This paper considers what must be represented for automated scientific discovery, and how those representations impact a program's ability to reason and explore.

## The Building Blocks

Scientists think about *concepts,* sets of elements with some commonality, such as "acyclic graph" or "dog." The *general concept* is a superset of all the concepts under consideration. In biology, the general concept is "living organism;" in graph theory the general concept is "graph." The power set (i.e., the set of all subsets) of the general concept is the *domain* of interest.

An *example* is a member of the general concept, and is defined by the details that make it a member. A dog is a particular creature defined by its physical manifestation; a graph is an ordered pair of sets, where the second (the *edges*) is pairs of elements from the first (the *vertices*). A representation of an example captures those details in a description. The same example can be represented in more than one way, however. A graph can be represented either by lists of its vertices and edges, or by a *sketch*, which depicts each variable as a unique dot and each edge between two variables as a line that joins the corresponding pair of dots. A *property* of an example is a statement that the example is a member of some concept more specific than the domain's general one. For example, a graph is connected (has the property of connectedness) if it is a member of the set of all graphs with a single connected component.

Of course, a body of scientific knowledge is more than a collection of concepts. Its richness lies in the relations that link its concepts to one another. Examples of relations include "all dogs have tails" and "all trees are acyclic graphs." There are two kinds of relations: theorems and conjectures. A *theorem* is a relation with a *proof*, an argument that reasons correctly from true premises to a true conclusion. A *conjecture* is a relation for which there is extensive evidence, but no proof.

Because discovery concepts are sets, the most obvious relations among them are set relations, particularly "subset." Subsets are the basis of ontological hierarchies (e.g., "dogs are mammals" and "trees are acyclic") that link known concepts together. Subsets can also demonstrate concept equivalence, such as "trees are the same as acyclic connected graphs." Other relations are more likely to be domain-dependent, such as "share a recent evolutionary ancestor" or "contain some representative as a subgraph."

Scientific discovery mines new examples for previously undetected concepts and relations. A new example may be interesting because it is so different from prior experience, that is, it is a member of few concepts in the knowledge structure other than the general one. A new example also provides the opportunity to confirm existing conjectures and to postulate new ones. Each newly encountered dog strengthens many subset relations, such as the one between "dog" and "animals with tails." A newly encountered example is particularly interesting when its very existence disrupts some carefully organized hierarchy or forces one to reconsider existing relations. This is why a dog with wings would be an extraordinary find.

Science assembles examples and concepts and relations into a *knowledge structure.* The challenge in scientific discovery is to find new concepts and valid relations that enrich it. Essentially, scientists devote themselves to the embellishment of a knowledge structure.

## Representation and Inference

Although inspired by examples, discovery ultimately addresses concepts. Most interesting discovery domains are unwieldy because their examples are numerous (e.g., number theory) or ancient (e.g., astronomy) or rapidly evolving (e.g., living flu viruses). Efficient automated discovery therefore requires compact concept representations that can be readily manipulated. When a concept summarizes only the discoverer's limited experience in a domain, however, relations that employ it can only be conjectures. Rather than enumerating its examples, there are two more useful ways to represent a concept: as a tester or as a generator.

A *tester* is a predicate that, when presented with an example, returns true ("this is an example of the concept") or false ("this is not"). Because testers are induced from a set of examples, they often fall prey to overgeneralization. Consider, for example, a young child, whose discovery domain includes everyday physical objects. She is likely at some point to call every four-legged creature "dog" until told that some creature she has thus labeled is actually a horse. ("See the hooves!") At that point, the tester must be corrected. A thoroughly corrected tester, however, may be awkward to manage because it comes with a long list of exceptions or a lengthy checklist at a level of granularity that related concepts do not require. Penguins, for example, have provoked entire logics, ones that an ornithologist would find cumbersome.

In contrast, a *generator* is an engine that, when executed, returns an example of the concept. Because discovery is driven by examples, concept generators that readily create new examples are essential. Generators provide the discoverer with experience, particularly when examples are manipulated (e.g., graph coloring). Automated mathematical discovery programs often rely on concept definitions that are generators (Epstein, 1988b; Lenat, 1976). An example of a generator appears in Figure 1.

The importance of generators in a knowledge structure is well illustrated in biology. For centuries, biologists relied on comparative morphology and behavioral criteria to form concepts like "fungus," that is, to group together living things that are similar. These elaborate testers were based on careful scrutiny of many examples. They also provoked arguments that went on for decades. Today, however, the genome for a biological species is acknowledged to be its generator. The re-definition of all biological species through strong DNA similarity has finally resolved long-standing issues in phylogeny, including evolutionary relations among songbirds, and whether pandas are more closely related to bears or raccoons. (See Figure 2.)

*Concept generator:*
$(A_{xy}A_y + A_{zz})^*(K_1)$ where $x \in V$, $y \notin V$, $z \in V$

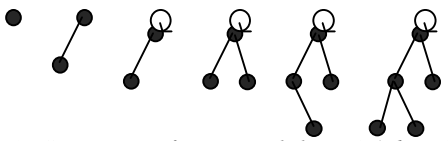
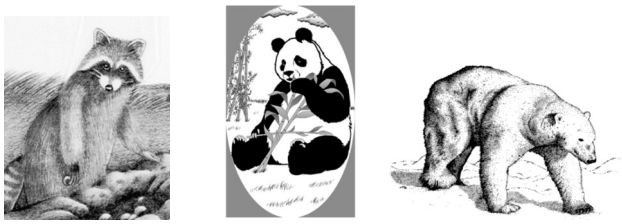


*Figure 1: Generation of "trees with loops" (edges from a vertex to itself). Begin with $K_1 = <V = \{v\}, E = \varnothing>$, the complete graph on one vertex, and then iterate. Each iteration either adds a loop to a vertex already in the graph ($A_{zz}$) or introduces a new vertex ($A_y$) with an edge ($A_{xy}$) to some vertex currently in the graph.*



(a) (b) (c)

*Figure 2: Tester definitions could not determine whether (b) a panda was more closely related to (a) a raccoon or to (c) a bear. Generators (DNA) have resolved this debate.*

Our understanding of the physical world is for the most part insufficient to produce unambiguous, sufficiently detailed definitions of examples. Mathematics, in contrast, begins with clear definitions of examples, and often allows us to formulate generators for concepts. As a result, most automated discovery has focused on mathematics, or has been restricted to mathematical representations for physical objects. For example, AM worked in set theory (Lenat, 1976) and GT in graph theory (Epstein, 1988a).

Counterexamples of testers (e.g., a dog-labeled horse) drive their revision; counterexamples of relations (e.g., winged dogs) invalidate them. Proof of a relation is more complex. When the structure of a generator or a tester is sufficiently transparent, discovery can use it to prove relations among concepts. With testers only of the form $a \leq b$, for example, one can use arithmetic principles to deduce subset relations. Figure 3 shows how to deduce that the sum of two even numbers is even, using a generator for "even." Given procedures to manipulate algebraic representations such as the one in Figure 1, a program can prove theorems about the sets it represents (Epstein, 1988a). More often, however, the generator is so opaque that the scientist can only examine additional examples with a tester, and hope to detect relations inductively.

## Heuristics in Discovery

Testers for interesting concepts are often computationally expensive, even intractable. In that case, discovery must find some way to enhance those testers. A tester is, after all, an executable procedure. When, as is often the case, the tester is of the form "there exists some $x$ such that $x$ has

*Concept generator*: To produce an even number, multiply any integer by 2.
*Theorem*: The sum of two even numbers is even.
*Proof*:
Generate even numbers $2x$ and $2y$ from numbers $x$ and $y$.
Since $2x + 2y = 2(x + y)$ and $x + y$ is a number
$2x + 2y$ can be generated from $x + y$
and $2x + 2y$ is also even.

*Figure 3: A proof that employs a concept generator.*

property $q$ with respect to this example," it requires a binding for $x$ (e.g., a 3-coloring of a graph). Execution of the tester searches for that binding. If $P \neq NP$, then some concepts are forever burdened with exponential testers. The best we can hope for is that most members of some concepts will be receptive to heuristics for those testers.

A potential pitfall in the construction of a discovery program is to commingle a *metric* (a numerically-valued function) or a representation with a heuristic (e.g., (Ritchie and Hanna, 1984)). When such a heuristic fails to support efficient testing, there are two possibilities: the metric or representation is not sufficiently relevant, or the heuristic uses it incorrectly. In graph coloring, for example, one can describe a graph by the degrees of its vertices (a metric that supplements the vertex-edge description), and then color first those vertices of maximum degree (the heuristic). It is possible, however, to construct graph-coloring problems where the number of permissible colors varies in a way that the vertices of *minimum* degree should be colored first — the metric is still appropriate but the heuristic is different. Moreover, there are often many heuristics that might enhance a tester.

An effective alternative is to provide multiple metrics and representations, but to decouple them from the heuristics. Decoupling makes all metrics and representations available to all heuristics; each heuristic uses any subset of them it chooses. Of course, representing the same information many ways is computationally expensive. Once a discovery program learns which heuristics best support its testers, it can pare down its portfolio, and retain only those metrics and representations that support the best heuristics.

This linkage of metric/representation quality to tester heuristics' quality has proved successful in game playing (Epstein, 2001), path finding (Epstein, 1998), and constraint solving (Epstein, Freuder and Wallace, 2005) All three programs are based on *FORR* (FOr the Right Reasons), an architecture for learning and problem solving (Epstein, 1994). Given examples of a concept and a set of tester heuristics for a candidate superset, FORR infers *weights* that represent the relative usefulness of those heuristics on the superset's tester. For example, the concept could be a set of coloring problems generated to have particular properties, the superset "has at least one solution," and the tester a general search engine. For a given tester, a heuristic may prove more effective on one concept than on another. Some heuristics may even rely on metrics that are inapplicable to a given concept (e.g., number of wings on a dog or cycles in a tree). The usefulness of a metric or a representation is ultimately the learned weights of the heuristics that reference it. Thus weight learning for tester heuristics both selects effective metrics and representations and improves the tester, which in turn supports discovery.

A new concept is simply a newly assembled set of examples. A new concept can be created from the intersection of other concepts (e.g., connected and acyclic graphs). Another way to generate a new concept is to restrict the values that an existing property's metric takes on examples. Consider, for example, the metric "number of integer divisors" applied to positive integers. Examples with the value 2 produce the concept "prime number." Once its metric-based heuristics have reliable weights, a discovery program can formulate a new concept from a highly weighted, metric-based heuristic. Under this approach, if a heuristic that maximizes vertex degree receives a high weight, complete graphs could be discovered.

Weight learning can also support the discovery of new heuristics. A learned, weighted combination of heuristics for a tester is itself a new heuristic. One can also specify a language whose expressions are interpretable as metric-based heuristics and then have a program learn weights for them. FORR provides such a facility. The language is domain-dependent: for game playing, its expressions are generalizations of patterns that can appear on the game board; for constraint satisfaction, they are arithmetic combinations of metrics that underlie input heuristics. Initially, as FORR applies its tester to examples, these expressions do not participate in decision making, but they earn weights as if they did. Expressions that earn and maintain high weights serve together as the metric for a single heuristic that does participate. Eventually, an outstanding individual expression can be identified to stand on its own as a new heuristic. Because it monitors every expression, this approach requires a language with limited expressive power. Nonetheless, it discovered a new heuristic which, exported to another program, improved search performance by as much as 96% on a new set of larger, more challenging problems (Epstein et al., 2002). The same approach also discovered forks for a non-trivial board game (Epstein, 2001). Thus, learning to identify effective heuristics for a tester both identifies good metrics and representations and gives rise to new heuristics and new metrics.

Because much of problem solving can be recast as concept membership testing, the discovery of new heuristics is important. "3-color this graph" can be construed as "is this graph an example of 3-colorable?" and "play checkers well" as "is checkers an example of a draw game?" From this perspective, discovery is indeed a more general form of problem solving, as observed earlier, one that focuses on sets of examples rather than individual ones.

## Challenges in Discovery

To establish relations among concepts in a domain, discovery focuses on subsets, is burdened by intractable testers, and requires heuristics to ease its way. Once metrics and representations are decoupled from heuristics, the process is clarified, but substantial challenges still remain. We consider several here: representational bias, focus of attention, interestingness, and the discovery of new representations.

A scientific discovery program manipulates, and therefore must represent, examples, concepts, relations, and heuristics. How they are represented *biases* a discovery program, that is, the program can only contemplate what it can represent. In example definition, representational bias delineates the domain. In concept description, representational bias is the only alternative to examining the power set of the general concept. A concept representation that is

also a generator, such as GT's, supports theorem proving. A sufficiently rich tester, such as graffiti's $a \leq b$, can produce many interesting conjectures (Fajtlowicz, 1988). Representational bias for relations delimits the space of conjectures and theorems; representational bias for heuristics restricts the ways that testers can be enhanced.

A representation with overly limited expressive power unacceptably constricts discovery. LEX, for example, learned to select integration formulae appropriate to examples of "integrand" (Mitchell, Utgoff and Banerji, 1983). The inadequacy of LEX's representation for "integrand" only became apparent when it began to work with rules that distinguished between odd and even exponents of trigonometric functions. "Even" and "odd" were not part of the representation. Extending a representation during discovery is extremely difficult.

At the other extreme, a concept representation with broad expressive power (e.g., first order predicate calculus) delineates an enormous search space where *focus of attention* (how to single out particular examples, concepts, and conjectures for investigation) becomes crucial. People focus their attention remarkably well, partly with semantics and partly from their familiarity with the knowledge structure. It is so difficult to construct a human-like theorem prover, for example, because a set of axioms and theorems offers few semantic clues as to which clauses ought to be combined (e.g., resolved) with one another — focus of attention is lacking. Purely syntactic approaches, such as attention to unit clauses, cannot consider the proof method (other than resolution) that led to them, the examples that drove the creation of the concepts involved, or alternative representations for them.

People focus on interesting concepts, and not all concepts are inherently interesting. What should make a new concept interesting to a discovery program still remains elusive. AM had an elaborate set of rules that quantified the interestingness of each concept (Lenat, 1976). For example, a concept was interesting if its examples all held the same, extreme (very large or very small relative to some metric) value of some descriptive feature. Primes were therefore interesting because they have one more than the minimum number of divisors. Nonetheless, interestingness ultimately proved to have little impact on the path of AM's discovery. One might further require that there be ample connections from such a set of extreme examples, but that proves inadequate too. GT once created many examples of a "new" graph concept, and proved many theorems about it, but it was the set of all edgeless graphs. Graphs with only one or two edges would be no more interesting. As observed earlier, a metric underlying a heuristic that earns a high weight while it guides a tester would be a better guide to interestingness. This area merits further study.

Finally, as chronicled by Pascal, Poincaré, and Hardy, human discovery has a strong visual component. The visual representations they described are a source of power, but an ill-understood one. Many graph heuristics originate from sketches, even though the definition of "graph" is purely mathematical. Incorporation of visual cognition in discovery programs is the next research frontier.

## Acknowledgements

This work was supported in part by the National Science Foundation under 9222720, IRI-9703475, IIS-0328743, IIS-0739122, and IIS-0811437. Thanks go to Peter Lipke for his patient tutelage on matters biological.

## References

Amarel, S. 1968. On Representations of Problems of Reasoning about Actions. Machine Intelligence 3. Michie, D. Edinburgh, Edinburgh University Press: 131-171.

Anzai, Y. and H. Simon 1979. The Theory of Learning by Doing. *Psychological Review* 36(2): 124-140.

Epstein, S. L. 1988a. Learning and Discovery: One System's Search for Mathematical Knowledge. *Computational Intelligence* 4(1): 42-53.

Epstein, S. L. 1988b. On the Discovery of Mathematical Concepts. *International Journal of Intelligent Systems* 3(2): 167-178.

Epstein, S. L. 1994. For the Right Reasons: The FORR Architecture for Learning in a Skill Domain. *Cognitive Science* 18(3): 479-511.

Epstein, S. L. 1998. Pragmatic Navigation: Reactivity, Heuristics, and Search. *Artificial Intelligence* 100(1-2): 275-322.

Epstein, S. L. 2001. Learning to Play Expertly: A Tutorial on Hoyle. *Machines That Learn to Play Games*. Fürnkranz, J. and M. Kubat. Huntington, NY, Nova Science**:** 153-178.

Epstein, S. L. 2004. Thinking through Diagrams: Discovery in Game Playing. In *Proceedings of Spatial Cognition IV*, 260-283. Springer-Verlag.

Epstein, S. L., E. C. Freuder, R. Wallace, A. Morozov and B. Samuels 2002. The Adaptive Constraint Engine. In *Proceedings of CP2002*, 525-540. Ithaca, Springer Verlag.

Epstein, S. L., E. C. Freuder and R. J. Wallace 2005. Learning to Support Constraint Programmers. *Computational Intelligence* 21(4): 337-371.

Fajtlowicz, S. 1988. On conjectures of graffiti. *Discrete Mathematics* 72(1-3): 113 - 118.

Hardy, G. H. 1972. *A Mathematician's Apology*, Cambridge University Press.

Lenat, D. B. 1976. AM: An Artificial Intelligence Approach to Discovery in Mathematics, Department of Computer Science, Stanford University.

Mitchell, T. M., P. E. Utgoff and R. Banerji 1983. Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. *Machine Learning: An Artificial Intelligence Approach*. Michalski, R. S., J. G. Carbonell and T. M. Mitchell. Palo Alto, Tioga Publishing**:** 163-190.

Pascal, B. 1964. *Pensées de Pascal*. Paris, Éditions Garnier Frères.

Poincaré, H. 1970. *La Valeur de la Sciènce*. France, Flammarion.

Ritchie, G. D. and F. K. Hanna 1984. AM: A Case Study in AI Methodology. *Artificial Intelligence* 23(3): 269-294.