On the Discovery of Mathematical Concepts

S. L. Epstein
Department of Computer Science, Hunter College of The City University of New York, 695 Park Avenue, New York, New York 10021

The Graph Theorist (GT) is a system intended to perform mathematical research in graph theory. This paper focuses upon GT's ability to discover new mathematical concepts by varying the definitions in its input knowledge base. Each new definition is a correct and complete generator for a class of graphs. The new concepts arise from the specialization of an existing concept, the generalization of an existing concept, and the merger of two or more existing concepts. Discovery is driven both by examples (specific graphs) and by definitional form (algorithms). GT explores new concepts either to develop an area of knowledge or to link a newly-acquired concept into a pre-existing knowledge base. From an initial knowledge base containing only the definition of "graph," GT discovers such concepts as acyclic graphs, connected graphs and bipartite graphs. Given an input concept, such as "star," GT discovers "trees" while searching for the appropriate links to integrate star into its knowledge base. The discovery processes construct a semantic net linking frames for all of GT's concepts together.

I. INTRODUCTION

The Graph Theorist (GT) is a knowledge-intensive, domain-specific learning system[1] which uses algorithmic class descriptions to discover new mathematical concepts and relations among them. GT is based upon a set of powerful representation languages for object classes discussed and defined formally in Epstein.[2] A variety of well-defined operations can be coerced from these languages. In particular they can be used to:

- generate correct examples of any class
- test whether or not an object belongs in a particular class
- reason about relations among classes

The representation of a class in any of these languages is an algebraic definition. Each definition has a semantic interpretation as a stylized algorithm which defines the class by generating it correctly and completely. GT incorporates

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 3, 167–178 (1988)

CCC 0884-8173/88/020167-12$04.00

such definitions as slots in its frame representation for a graph property. The entire frame represents a concept.

GT operates either independently or under interactive guidance. It generates correct examples of any of its concepts, constructs new concepts, and conjectures and proves relations among concepts. The discovery and proof of relations among a set of input concepts is discussed in Epstein.[3] This article is devoted to the origin of new concepts and relations among them.

II. BACKGROUND

Much important work in machine learning (for example, Laird,[4] Langley,[5–6] Lee,[7] Michalski,[8–9] and Stepp[10]) constructs concepts to explain empirical data—data either input or encountered in the course of task execution. These concepts are rules which people consider coherent classifications and/or explanations. Such data-driven scientific research is based upon observation and inductive inference, in domains where partial information and real-world noise are the norm. There are, however, scientific research areas, particularly in mathematics, where classes of objects are already clearly delineated by definitions. In modelling such domains, the origin of these classes (what Michalski[1] calls *constructive induction*) and the relations among them should be the focus of attention. An outstanding example of concept learning for exploration, rather than explanation, was Lenat's AM.[11]

AM began with a set of 115 frames, each representing a mathematical concept. A frame consisted of fixed slots containing information on examples, hierarchical pointers, conjectures about such pointers and heuristics attached to the concept. Some of AM's initial (i.e., input) concepts were for classes of mathematical objects, such as sets or lists. Other initial concepts were for activities defined on those classes, such as set-insert or list-intersect. Examples for a class were generated by applying seemingly-appropriate operators. For example, if an operator f were defined on the classes A and B with $f: A \rightarrow B$, and if C were a subset of B, AM might have selected some examples $a_1, a_2, \ldots, a_n$ of A, constructed $f(a_1), f(a_2), \ldots, f(a_n)$ (all of which would have belonged to B), and then examined them to see if they were indeed examples of C, or merely in $B-C$. This uncertainty in the generation of examples was due to the nature of an AM definition for a class of mathematical objects. Each definition was a LISP predicate which tested whether or not an object was a member of the class.

AM explored these fundamental frames under the guidance of 243 heuristic rules. It generated examples of the concepts, conjectured about relations among them, and defined new frames (concepts) for subsequent exploration. Beginning only with set-theoretic concepts, it included among its discoveries natural numbers, primes and the fundamental theorem of arithmetic. AM, however, was never able to prove its conjectures, to generate new examples certain to be in a specified class, or to introduce radically new descriptions into class definitions.

III. CONCEPT DESCRIPTION IN GT

GT derives much of its power from a set of representation languages whose theoretical formulation is detailed rigorously in Epstein[2] and summarized in Epstein.[12] The treatment of the representation here is informal and describes only selected, implemented segments of the theory. For example, GT currently only supports undirected, unlabelled graphs, but coding provisions have been made for directed and labelled graphs, and the theoretical framework supports them. This section provides fundamental definitions and describes the algebraic representation and its semantic interpretation.

Let V be an arbitrary, finite set of elements (*vertices*) and let E be any subset (*edges*) of the Cartesian product $V \times V$. Then the ordered pair $G = \langle V,E \rangle$ is said to be a *finite graph*. Let U be the universe of all finite graphs. Then any subset P of U is said to designate a *graph property* p and, for G in P, G is said to *have property* p. (The distinction between the property p and the class P is syntactic, not semantic. The context will dictate which is used.) Any algorithmic definition of the graph property p must specify precisely the set P. In particular, if an algorithm claims to generate P, that algorithm must be both *correct* (i.e., every generated graph must be in P) and *complete* (i.e., for each graph G in P there must be a finite sequence of steps executed by the algorithm with final output G).

In GT, a *concept* is a frame representing a graph property and knowledge associated with it. An edited example of a GT frame for the concept CONNECTED appears on the left in Figure 1. The slots of the frame include a list of examples, a list of hierarchical relations with other concepts and a description of the origin of the property. (Entries of NIL for relations are statements of partial knowledge, to be read as "none discovered yet.") The frame also includes one or more definitions of the graph property in a specific, three-part formulation.

In GT, a *definition* of a graph property is an ordered triple $\langle f,S,\sigma \rangle$. S is the *seed set*, a set of one or more minimal graphs (*seeds*), each of which has the property in question. (Typically the seed set is finite and GT lists its elements.) The seed set in the example of Figure 1 for CONNECTED contains only a single graph, K_1, the complete graph on one vertex. The *operator* f in the definition describes the way(s) any graph with the given property may be transformed to construct another graph with the same property. An operator in GT is built from the set of primitives listed in Table 1. These primitives may be concatenated into terms (such as "$A_{yz}A_z$") to denote sequential operation from right to left. Terms may be summed (as in "$A_{wx} + A_{yz}A_z$") to represent alternative actions. Thus the operator $A_{wx} + A_{yz}A_z$ for CONNECTED is read "either add an edge from w to x or else add a vertex z and then an edge from y to z." The *selector* σ in the definition describes the restrictions for binding the variables appearing in the operator f to the vertices and edges in a graph. The currently-implemented selector descriptions of vertices and edges appear in Table 1. Selector descriptions may be empty, i.e., need not constrain binding at

Property-Name: CONNECTED
Number-of-Seeds: 1
Seed-Set-List: $\{K_1\}$
Function: $A_{wx} + A_{yz}A_z$
Sigma: $w,x,y \in V$, $Z \notin V$
Origin: specialization of property-13
Example-list: $\{K_1$, connected-2,connected-3,...,connected-13$\}$
Extremal-cases: $\{K_1\}$
Subsumes-list: TREE
Does-not-subsume-list: {ACYCLIC}
Subsumed-by-list: {IS-A-GRAPH, property-3, property-13}
Is-not-subsumed-by-list: {TREE, ACYCLIC}
Merger-created-with-list: {ACYCLIC}
Merger-explored-with-list: NIL
Primary-definition: T
Is-equivalent-to-list: NIL
Equivalence-explored-with-list: NIL
Delta-pairs: ((0 1)(1 1))

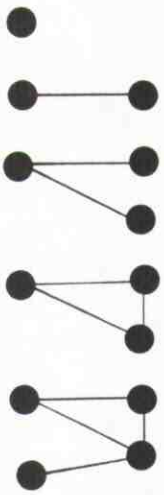

Figure 1. GT Representation of CONNECTED.

all. In the example of Figure 1, the selector for CONNECTED is read "where w,x and y are in the vertex set, and z is not in the vertex set."

The semantic interpretation of such a three-part definition for a graph property p is a single, uniform algorithm called a *p-generator*. A p-generator capitalizes on the underlying commonality of its class, the view of the set P as one or more prototypes (seeds) which can be methodically deformed (under f and σ) to produce exactly those graphs in the class. The p-generator may be thought of as an automaton which is started by the input of any graph in seed set S. CONNECTED, for example, would require K_1. The p-generator then iterates an undetermined number of times. On each iteration the selector σ

Table 1. Implemented GT Operators and Selectors.

Symbol	Application	Interpretation
A_x	Operator	Add vertex x to the graph: $V \leftarrow V \cup \{x\}$
A_{xy}	Operator	Add edge xy to the graph: $E \leftarrow E \cup \{xy\}$
D_x	Operator	Delete vertex x from the graph: $V \leftarrow V - \{x\}$
D_{xy}	Operator	Delete edge xy from the graph: $E \leftarrow E - \{xy\}$
$x \in V$	Selector	x is a vertex in the graph
$x \notin V$	Selector	x is a vertex not in the graph
$x \neq y$	Selector	x and y are distinct vertices
$d(x) = n$	Selector	x's degree is n, a non-negative integer
$d(x) = \max$	Selector	x's degree is the largest in the graph
$xy \in E$	Selector	xy is an edge in the graph
$xy \notin E$	Selector	xy is an edge not in the graph

chooses vertices and/or edges with respect to the current graph G, and then the operator modifies G, using those choices, to produce a new G. CONNECTED, on each iteration, either adds a new vertex x to the graph, or adds a new vertex z and an edge from an old vertex y to z.

Thus the algorithm for generating the class P of graphs is:

Accept $G \in S$
Output G
Until σ fails do
    $G \leftarrow f\sigma(G)$
    Output G
Halt

Under all possible initial choices from S and all possible iterations of f subject to σ, the output of this algorithm is precisely P, that is, if the superscript i denotes "iterate i times,"

$$P = \bigcup_i (f\sigma)^i(S)$$

The graphs on the right in Figure 1 illustrate several possible iterations of the definition of CONNECTED; each pictured graph is output by the algorithm and is connected. The definition generates the infinite class of connected graphs; it will never halt because bindings for the variables in σ can be found on each iteration.

The content of the following three general texts is taken as *graph theory*: a classical development in elegant mathematical fashion,[13] a broad overview of topics presented as definitions and theorems,[14] and an algorithmic approach.[15] What guarantee is there that p-generators exist for every P in U, or at least for every interesting P in graph theory? At this writing, more than 40 properties have been selected from the three benchmark texts and described correctly and completely as p-generators.[2] Among these are k-connected, k-chromatic, planar and Hamiltonian.

The use of p-generators as property definitions entails several kinds of nondeterminism. Any graph in the seed set is an acceptable input; any binding satisfying σ is valid; any summand in f suffices for an iteration. In addition, many different sequences of iterations will construct isomorphic graphs, and more than one definition may be written for certain properties. This ostensible indefiniteness and redundancy is tolerated because the property definitions preserve detail in a concise and flexible format.

IV. CONSTRUCTION OF NEW CONCEPTS IN GT

GT definitions transparently display both the changes they force upon objects (in the operator f) and the preconditions they require (in the selector σ). This separation encourages the development of completely and correctly defined new concepts whose relations to their parent concepts do not require proof. GT currently has three methods for constructing new concepts: specialization, generalization and merger.

A. New Concepts Discovered by Specialization

A property p_2 is said to be a *specialization* of a property p_1 if and only if P_2 is a subset of P_1, i.e., every graph with property p_2 also has property p_1. To specialize from a definition of the form $\langle f,S,\sigma\rangle$, GT performs one of the following actions:

- constrain the seed set
- constrain the operator
- constrain the selector

When GT discovers property p_2 as a specialization of property p_1, the facts that P_2 is a subset of P_1, and P_1 a superset of P_2, are recorded in GT's knowledge base.

A seed set is constrained by using a proper subset of it. Consider, for example, the property

$$p_1 = \langle A_x + A_{yz}A_z, \{K_1,K_4\}, [x \notin V, y \in V]\rangle$$

The definition p_1 begins either with K_1 or K_4, the complete graph on four vertices. On any iteration, p_1 either adds an isolated vertex x to the graph or adds a vertex z and an edge yz from a vertex y in the graph to z. One specialization of p_1 is

$$p_2 = \langle A_x + A_{yz}A_z, \{K_4\}, [x \notin V, y \in V]\rangle$$

created by eliminating K_1 from the seed set of p_1. Every graph generated by p_2 begins with a seed from p_1 (namely, K_4), iterates according to the definition of p_1, and, therefore, has property p_1. There are, however, graphs (for example, K_2 and K_3) which are in P_1 but not in P_2. Thus p_2 is a proper subset of p_1.

An operator may be constrained in two ways. First, a term may be eliminated from the operator (with irrelevant constraints removed from the selector). For example,

$$p_3 = \langle A_{yz}A_z, \{K_1,K_4\}, [y \in V]\rangle$$

constructs only the connected p_1 graphs. One may prove p_3 a specialization of p_1 by the argument used for p_2 above. Second, and less obviously, recall that a p-generator assumes iteration. Thus any forced repetition of terms from the operator forms a special case of the operator. (The selector requires readily-computable additions.) Consider:

$$p_4 = \langle A_xA_w + A_tA_{yz}A_z, \{K_1,K_4\}, [x,w,t \notin V, y \in V]\rangle$$

Property p_4 adds either a pair of isolated vertices or an isolated vertex, an edge, and a vertex z, on each iteration. Property p_4 begins with a seed from p_1, and each of its terms is equivalent to a finite number of iterations of p_1; therefore, p_4 is a specialization of p_1.

GT constrains the selector of a graph property by making the binding restrictions more detailed, either by the addition of a constraint or the identification of variables. As an example of the first, consider:

$$p_5 = \langle A_x + A_{yz}A_z, \{K_1, K_4\}, [x,z \notin V, y \in V]\rangle$$

Using the argument employed for p_2, p_5 is seen to be a specialization of p_1; P_5 is that subset of P_1 which is acyclic everywhere except possibly in a single K_4 subgraph. Additional constraints must always be consistent with the definition of a graph, and never obviously make binding impossible. (For example, $x \in V$ would not be added when the selector already specifies $x \notin V$). As an example of the second selector specialization, consider:

$$p_6 = \langle A_x + A_{yy}A_y, \{K_1, K_4\}, [x \notin V, y \in V]\rangle$$

Here GT has selected two variables, y and z in p_1, whose σ-descriptions do not contradict each other, and has made them identical.

Figure 2 shows how GT applies specialization to discover new concepts in graph theory. Initially, the knowledge base consists only of the p-generator for all finite graphs, which appears in line 1. The more interesting properties have been renamed for the figure. (The discovered version of CONNECTED is an alternative definition equivalent to that in Figure 1.)

Another of GT's discoveries is

$$\text{BIPARTITE} = \langle A_x + A_{yy}A_y + A_{wz}, \{K_1\}, [x,y \notin V, w,z \in V, ww \in E, zz \notin E]\rangle$$

For BIPARTITE, GT has partitioned the vertices of the graph into two sets, one with loops and one without; edges are drawn only between one vertex with a loop and one without. (A loop is an edge from a vertex to itself.)

B. New Concepts Discovered by Generalization

A property p_2 is said to be a *generalization* of a property p_1 if and only if P_1 is a subset of P_2, i.e., every graph with property p_1 also has property p_2. Because "p_2 is a generalization of p_1" is equivalent to "p_1 is a specialization of p_2," the construction of generalizations is fairly obvious from the preceding discussion. To generalize a concept, GT may:

- expand the seed set
- expand the operator
- relax the selector

Property	*Origin*
(1) IS-A-GRAPH: $\langle A_{xy} + A_z, \{K_1\}, [x,y \in V]\rangle$	Given
(2) PROPERTY-4: $\langle A_{xy}A_z, \{K_1\}, [x,y \in V]\rangle$	Forced repetition (1)
(3) PROPERTY-6: $\langle A_{xy}A_z + A_w, \{K_1\}, [x,y \in V]\rangle$	Forced repetition (1)
(4) PROPERTY-14: $\langle A_{xz}A_z, \{K_1\}, [x \in V]\rangle$	Identification of variables (2)
(5) TREE: $\langle A_{xz}A_z, \{K_1\}, [x \in V, z \notin V]\rangle$	Add binding restrictions (4)
(6) CONNECTED: $\langle A_{xz}A_z, \{K_1\}, [x \in V, x \neq z]\rangle$	Add binding restrictions (4)
(7) PROPERTY-30: $\langle A_{xz}A_z + A_w, \{K_1\}, [x \in V]\rangle$	Identification of variables (3)
(8) ACYCLIC: $\langle A_{xz}A_z + A_w, \{K_1\}, [x \in V, z \notin V]\rangle$	Add binding restrictions (7)

Figure 2. Examples of Discovery by Specialization.

To expand a seed set, GT adds another graph or set of graphs to it. To expand an operator, GT adds new terms or splits existing ones (the inverse of forced iteration). To relax the selector, GT removes details from the binding restrictions in σ. Each of the examples of specialization in Section 4.A may be read, in reverse, as an example of generalization. When GT discovers a new concept p_2 as a generalization of a concept p_1, the facts that P_2 is a superset of P_1, and P_1 a subset of P_2, are recorded in the knowledge base.

Why would GT need to know how to generalize at all? GT models a variety of research behaviors, one of which is the appropriate insertion of new information into a pre-existing knowledge base. A new property is generalized until it can be linked into GT's relational hierarchy. For example, when given

$$\text{STAR} = \langle A_{xy}A_y, \{K_{1,3}\}, [x \in V, y \notin V, d(x) = \max] \rangle$$

the concept is generalized until it is identified with one in the knowledge base. Directed to relax STAR's binding constraints, GT produces three new property definitions, one of which differs from TREE only in its seed. When directed to relax the constraints once again, GT produces two definitions, one of which differs from line 4 in Figure 2 only in its seed. GT eventually recognizes stars as a special case of connected graphs, discovering trees along the way. Another motivation for concept generalization is the conjecture and proof of relations among properties. When a property is highly-detailed, the entailed matching can be expensive. Reasoning about a more general case, which typically has a simpler form, may be much more efficient. For example, if A is a special case of B and B is disjoint from C, A will also be disjoint from C. Often GT's discovery and proof that B is disjoint from C is faster.

C. New Concepts Discovered by Merger

In mathematics, the intersection of classes of objects is frequently very useful and applied often. GT represents such an intersection by merger. The *merger* of a property p_1 for class P_1 with a property p_2 for a class P_2 results in a new property representing $P_1 \cap P_2$, the set of graphs with both properties. Let $p_1 = \langle f_1, S_1, \sigma_1 \rangle$ and $p_2 = \langle f_2, S_2, \sigma_2 \rangle$. GT currently has four algorithms to construct the merger of p_1 and p_2. The first three are fairly straightforward:

- If p_1 is a generalization of p_2, the merger is simply p_2. For example, the merger of CONNECTED and TREE is simply TREE.
- If f_2 is a constrained version of f_1 and every seed in S_2 lies in P_1, the merger is $\langle f_2, S_2, \sigma \rangle$, where σ is σ_1 and σ_2, but eliminates any references to variables not in f_2. For example, the merger of STAR and TREE is simply STAR.
- When f_1 is a generalization of f_2, σ_1 is a generalization of σ_2, and S is nonempty, the merger is $\langle f_2, S, \sigma_2 \rangle$, where S is $S_1 \cap S_2$ plus those seeds of S_2 in P_1 and those seeds of S_1 in P_2. For example, the merger of

$$p_1 = \langle A_x + A_{yz}A_z, \{K_1, K_4\}, [x \notin V, y \in V] \rangle$$

with

$$p_7 = \langle A_{rs}A_s, \{K_1,K_3\}, [r \in V, s \notin V]\rangle$$

is

$$p = \langle A_{uv}A_v, \{K_1,K_3\}, [u \in V, v \notin V]\rangle$$

based on matching y with r as u and z with s as v.

The fourth and most interesting of GT's merger algorithms deals with the cases which do not fit these categories. Let n be the number of vertices in a graph and m be the number of edges. Each iteration of a p-generator effects a change (Δn) in the number of vertices and a change (Δm) in the number of edges. GT calculates the Δn and Δm values for each term in the properties to be merged. A *delta pair* (Δn,Δm) is the ordered pair of the changes for a term in a property; it captures some aspects of the minimal effect of one iteration of a p-generator. For example, the only delta pair for

$$\text{TREE} = \langle A_{xy}A_y, \{K_1\}, [x \in V, y \notin V]\rangle$$

is (1,1), meaning that on each iteration one vertex and one edge are added to the graph and for

$$\text{ODD-VERTICES} = \langle A_rA_s + A_{tu}, \{K_1\}, [r,s \notin V, t,u \in V, r \neq s]\rangle$$

the delta pairs are (2 0) and (0 1). GT seeks a minimal positive integer solution to that system of equations which asserts that some number of repetitions of the delta pair for each term in one property is equivalent to some number of repetitions of the delta pairs in the second property. In the example, let α represent the number of applications of the single term in TREE, let β represent the number of applications of the first term in ODD-VERTICES and let μ represent the number of applications of the second. GT seeks the positive minimal integer solutions to:

$$1\alpha = 2\beta + 0\mu \quad (\Delta n)$$

$$1\alpha = 0\beta + 1\mu \quad (\Delta m)$$

The answer, $\alpha = \mu = 2$ and $\beta = 1$, indicates that in the merger both Δn and Δm will be 2. Each of the properties is specialized by forced iteration to meet these requirements:

$$\text{TREE}' = \langle A_{xy}A_yA_{uv}A_v, \{K_1\}, [x,u \in V, y,v \notin V]\rangle$$

$$\text{ODD-VERTICES}' = \langle A_rA_sA_{tz}A_{pq}, \{K_1\}, [r,s \notin V, t,z,p,q \in V, r \neq s]\rangle$$

When GT attempts a merger of TREE′ and ODD-VERTICES′ it discovers that the first is really a special case of the second, under the matching of r and z with y, s and q with v, t with x and p with u. (An extremely limited form of commutativity is used here to shift operators of the form A_x to the right when the vertex does not appear elsewhere in the term.) Thus the merger is

$$\text{ODD-TREE} = \langle A_{xy}A_yA_{uv}A_v, \{K_1\}, [x,u \in V, y,v \notin V, y \neq v]\rangle$$

Upon inspection, this property is clearly correct and complete, with $\Delta n = \Delta m = 2$. GT has discovered, among other merged properties, TREE as ACYCLIC merged with CONNECTED.

V. RESULTS AND PERSPECTIVE

GT discovers new mathematical concepts by syntactic changes whose semantics are well-understood and accessible to the program. The key in GT is a more transparent and flexible class definition, one which generates guaranteed examples, constructs efficient intersections, and creates from a broad descriptive vocabulary. These concepts form a rich knowledge base conducive to further mathematical discovery.

From the examples and definitions of Section 4, the following is evident:

> **Theorem:** The heuristics used by GT to constrain/relax any definition of a graph property P construct valid specializations/generalizations of P.

This theorem guarantees that the definitions GT constructs are, in fact, graph properties. It also justifies the hierarchical links GT inserts during the discovery process.

Since GT has several different starting points, a concept may be discovered in more than one way. Running on a Symbolics 3675 in Symbolics Common Lisp and beginning only with the definition of a graph and the heuristics described here, GT discovers, among other concepts:

- acyclic graphs
- connected graphs
- bipartite graphs
- trees
- stars

GT is able to incorporate all of these correctly into its hierarchical knowledge structure. A demonstration during which all of these discoveries takes place requires approximately three and a half minutes of elapsed time. Alternatively, GT can begin with a small initial knowledge base of concept definitions and generalize the concept "star" until it is able to link it into its knowledge base. During the (less than one minute of elapsed) time required to do this, GT also discovers "tree."

When GT "invents" a new property definition, it is subjected to careful scrutiny before a concept frame is created for it. Many generated definitions are trivial, i.e., they may iterate only once or twice, or even be limited entirely to their seed set. Other definitions, intended as a specialization of some parent

concept, may very quickly produce many more examples than were known for the parent. Still other definitions, intended as generalizations of some parent concept, may produce only graphs already known as examples of the parent. All of these definitions are deemed uninteresting and rejected as potential concepts.

Some of GT's discovery paths are a bit surprising. For example, although TREE is a special case of CONNECTED, definitions for both concepts appear during a single exploration cycle. In another unanticipated action, when STAR is being generalized, GT moves backward, first to TREE and then to a definition for "connected graphs with loops," skipping over ACYCLIC and CONNECTED completely. Even well-planned inductive leaps do not always arrive where expected.

VI. FUTURE WORK

Michalski's and Dietterich's work on generalization rules for concept acquisition provide some excellent suggestions for concept discovery in GT.[16–17] GT already embodies both selective and constructive generalization techniques, such as the "dropping condition" rule (as selector relaxation) and the "closing interval" rule (as a merger heuristic). Other rules currently under consideration and/or development include extending reference, counting arguments and internal disjunction. GT's descriptive ability lies in the number and nature of the primitive operators permitted in f and of the selector descriptions permitted in σ. As the set of such operators and descriptions is extended, a lattice of descriptive languages (detailed in Epstein[2]) can be constructed. Such an "extended" language offers additional alternatives, and ordinarily has greater expressive power (as measured by the number of graph properties it defines) than GT's current representation. In turn, operations with an extended language are likely to require more computer resources. Within the discovery framework described here, plans exist to extend the p-generator language for the representation of directed graphs and, eventually, for labelled graphs. These extensions will also provide a testbed for the study of performance under representational shifts.

The key to the most interesting specializations, those involving additional descriptions in σ, is the language in which those descriptions may be written. Utgoff[18] warns that, unless the [σ-]language is extensible, GT may not be able to access many interesting ideas. Ways to have GT extend the σ-language itself are currently being studied. Despite substantial empirical support, the existence of a definition of the form $\langle f,S,\sigma\rangle$ for every property p remains an open question.

Finally, GT's ability to spawn new concepts with such variety dooms it to a combinatoric explosion. At the moment GT accepts directives to generalize, specialize or merge; it does not originate these directives itself. The ability to differentiate an "interesting" new concept from an uninteresting one is a nontrivial control issue which is the focus of current work.

References

1. R.S. Michalski, "Understanding the Nature of Learning: Issues and Research Directions," in *Machine Learning: An Artificial Intelligence Approach,* **2,** R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds), Tioga Publishing, Palo Alto, 1986, 3–25.
2. S.L. Epstein, "Knowledge Representation in Mathematics: A Case Study in Graph Theory," (Ph.D. Dissertation), Rutgers University, 1983.
3. S.L. Epstein, "On the Discovery of Mathematical Theorems," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Milan, 1987, 194–197.
4. P.G. Laird, "Inductive Inference by Refinement," *Proceedings of the Fifth National Conference on Artificial Intelligence,* Philadelphia, PA, 1986, 472–476.
5. P. Langley, G.L. Bradshaw, and H.A. Simon, "Rediscovering Chemistry with the BACON System," in *Machine Learning: An Artificial Intelligence Approach,* R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds), Tioga Publishing, Palo Alto, 1983, 307–329.
6. P. Langley, J.M. Zytkow, H.A. Simon, and G.L. Bradshaw, "The Search for Regularity: Four Aspects of Scientific Discovery," in *Machine Learning: An Artificial Intelligence Approach,* **2,** R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds), Tioga Publishing, Palo Alto, 1986, 425–469.
7. W.D. Lee and S.R. Ray, "Rule Refinement Using the Probabilistic Generator," *Proceedings of the Fifth National Joint Conference on Artificial Intelligence,* Philadelphia, PA, 1986, 442–447.
8. R.S. Michalski and R.E. Stepp, "Concept-based Clustering versus Numerical Taxonomy," *Technical Report 1073,* Department of Computer Science, University of Illinois, 1981.
9. R.S. Michalski and R.E. Stepp, "Learning from Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach,* R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds), Tioga Publishing, Palo Alto, 1983, 331–363.
10. R.E. Stepp and R.S. Michalski, "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects," in *Machine Learning: An Artificial Intelligence Approach,* **2,** R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds), Tioga Publishing, Palo Alto, 1986, 471–498.
11. D.B. Lenat, "AM: An Artificial Intelligence Approach to Discovery in Mathematics," (Ph.D. Dissertation), Stanford University, 1976.
12. S.L. Epstein, "Languages for Problem Solving in Graph Theory," in *The Role of Language in Problem Solving,* **2,** North Holland, New York, 1987
13. O. Ore, American Mathematical Society Colloquium Publications, **38:** *Theory of Graphs,* Providence, RI, American Mathematical Society, 1962.
14. F. Harary, *Graph Theory,* Addison-Wesley, Reading, MA, 1972.
15. J. Bondy and U. Murty, *Graph Theory with Applications,* North-Holland, New York, 1976.
16. R.S. Michalski, "A Theory and Methodology of Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach,* R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds), Tioga Publishing, Palo Alto, 1983, 83–134.
17. T.G. Dietterich and R.S. Michalski, "A Comparative Review of Selected Methods for Learning from Examples," in *Machine Learning: An Artificial Intelligence Approach,* R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds), Tioga Publishing, Palo Alto, 1983, 41–81.
18. P.E. Utgoff, "Shift of Bias for Inductive Concept Learning," in *Machine Learning: An Artificial Intelligence Approach,* **2**, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds), Tioga Publishing, Palo Alto, 1986, 107–148.