LEARNING BY RE-EXPRESSING CONCEPTS FOR EFFICIENT RECOGNITION *

Richard M. Keller

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

Abstract

Much attention in the field of machine learning has been directed at the problem of inferring concept descriptions from examples. But in many learning situations, we are initially presented with a fully-formed concept description, and our goal is instead to re-express that description with some particular task in mind. In this paper, we specifically consider the task of recognizing concept instances efficiently. We describe how concepts that are accurate, though computationally inefficient for use in recognizing instances, can be re-expressed in an efficient form through a process we call **concept operationalization**. Various techniques for concept operationalization are illustrated in the context of the LEX learning system.

I Introduction and Motivation

Historically, machine learning research has focused on the task of inferring concept descriptions based on a set of examples, and it is only recently that other types of learning have begun to come under investigation [2]. In this paper, we focus on the process of concept reformulation rather than concept acquisition. We assume that a learning system acquires concepts in some unspecified manner, whether by inductive or deductive means or by being told. For a variety of reasons, it may be necessary to re-express an acquired concept in different terms. In particular, the concept may be expressed in a manner that is computationally inefficient for the purpose of recognizing examples. In such cases, the learning system is faced with the task of reformulating the concept so that recognition can occur more swiftly. We call this task **concept operationalization**, in the spirit of recent, related work by Mostow [8] and Hayes-Roth [3].

Consider the problem of using a concept description to efficiently recognize examples of arch structures. Winston's pioneering concept learning system [11] succeeded both in formulating an arch concept description and in subsequently using that description to recognize arch instances. An arch instance was given in terms of various visual and structural features of its component parts (e.g. shape, orientation, and relative placement). The program inductively inferred a structural description of the arch concept, similar to the one shown below, based on a set of training examples:

Structural Arch Concept

An **arch** is a structure which:
(i) is composed of 3 objects, 2 of which must be bricks
(ii) the bricks must be standing adjacent yet not touching
(iii)the other object must be lying horizontally, supported by the bricks.

Using this description, arch recognition was quite efficient. The program simply had to match the structural features of a prospective arch instance against the structural description.

Now imagine that instead of giving his system a set of training examples to be used in concept formation, Winston short-circuited the process and initially provided his system with a complete, although non-structural description of the arch concept. For example, he might have provided a *functional* description of an arch:

Functional Arch Concept

An **arch** is a structure which:
(i) spans an opening, and
(ii) supports weight from above, or to either side of the opening.

Although the functional description is just as valid as the structural one, recognition is no longer a simple matter. It is not clear how to match the structural features of an instance against a functional definition without some intermediary processing. In particular, either (i) the instance must be altered to include functional features as well as structural features *a priori*, or (ii) the functional features must be computationally derived each time a new structural instance is processed, or (iii) the functional definition must be re-expressed permanently in structural terms. Of these options, (iii) represents the most practical long-term solution to the recognition problem. In the context of the arch example, the structural re-expression of the functional definition involves the use of physics knowledge, as well as other domain-independent knowledge, to relate form and function**. Once the arch description has been re-expressed in a manner suitable for efficient recognition we will consider it to be an **operational** concept description.

In the balance of this paper, the task of concept operationalization is more precisely defined. Section II describes how the notion of concept operationalization initially arose in the context of our recent experiences with the LEX learning system [5]. Section III follows with a more formal specification of the concept operationalization task. Various techniques for dealing with this task are then introduced and illustrated in Section IV. Section V concludes with some comments about related research and some issues that must be addressed prior to a full-scale implementation of the proposed techniques.

II Concept Operationalization and Problem Solving

To explore further the notion of concept operationalization, we base our discussion on the LEX

*Work supported by NIH Grant #RR-64309, and a Rutgers University Graduate Fellowship.

**Coincidentally, since the initial writing of this paper it has come to my attention that Winston is pursuing the relationship between form and function using techniques related to those described here [12].

learning system. While the framework proposed in this paper has not been incorporated into LEX, it arises out of our recent experience with this system, and our attempt generalize upon its methods.

LEX is a program which learns to improve its problem solving performance in integral calculus. Problems are solved by the system using a least-cost-first forward state space search method. The starting state in the search contains a well-formed numeric expression with an integral sign. LEX's task is to solve the integral by applying a sequence of operators that will transform the starting state into one containing no integral sign. A set of approximately 50 calculus and arithmetic simplification operators is initially given to the system. Each operator specifies (i) an operation to be performed, and (ii) a set of numeric expressions to which the operator may legally be applied. Here are three sample operators:

OP1: $\int \sin(x)dx \longrightarrow \cos(x)$

OP2: $\int k \cdot f(x)dx \longrightarrow k\int f(x)dx$

OP3: Integration by Parts:

$\int f1(x) \cdot f2'(x)dx \longrightarrow f1(x) \cdot f2(x) - \int f2(x) \cdot f1'(x)dx$

where k:constant, f(x):function, and f'(x):derivative of f(x)

As LEX solves problems, it learns to prune its search tree by refusing to apply operators in those situations where experience has shown an application to be "nonproductive", although legal. For example, it is often legal to apply 'integration by parts' (whenever the integral contains a product), but it is much less frequent that the application will lead to a solution. The criterion to use in deciding whether an operator application is to be considered productive or nonproductive is given to the system initially as the Minimum Cost Criterion shown below:

Minimum Cost Criterion: Any operator application that extends the search tree in the direction of the minimum cost*** solution is considered a **productive operator application**.

LEX's learning task is to acquire the ability to efficiently distinguish between productive and nonproductive operator applications. By definition, this ability should improve the system's problem solving behavior. Note that with prior knowledge of the Minimum Cost Criterion, LEX begins by knowing *in principle* how to distinguish between productive and nonproductive operator applications: simply expand the search tree until a minimum cost solution is found and then classify the operator applications accordingly. However, this method is grossly self-defeating, since a process that simply deliberates about which operator to apply next cannot be granted the luxury of conducting an exhaustive search for the minimum cost solution!

Although direct use of the Minimum Cost Criterion to recognize productive operator applications is prohibitive, the Criterion is used by LEX in other significant ways. The most recent version of LEX**** [6] employs both a procedural and a declarative representation of the Criterion in the processing of instances. Initially, the procedural Criterion (called the CRITIC) classifies an individual training instance (i.e. an operator application) as positive or negative (productive or nonproductive). If the instance is positive, the declarative Criterion is then used to analyze which features of the instance were specifically relevant to the CRITIC's positive classificatory decision. Once the relevant features have been identified, LEX generalizes the positive instance by discarding irrelevant features. The generalized, rather than the original, positive instance is then fed to LEX's inductive learning mechanism*****. This mechanism then constructs an efficient method for recognizing positive instances based on syntactic patterns in the instances.

It is possible to view the process described above in terms of concept operationalization. LEX starts with a concept (in the guise of the Minimum Cost Criterion) which can be used to recognize productive operator applications. However, since the Criterion serves as an extremely inefficient recognizer, it is necessary for LEX to operationalize it in terms of an efficient pattern matching procedure. In the next section we draw out the relationship between LEX and concept operationalization in further detail.

III Problem Definition and Terminology

In order to proceed with our discussion of concept operationalization, it will be necessary to introduce some terminology. An **instance** is a data structure that efficiently encodes a set of **features**. A **positive instance** stores a set of feature values that constitute an example of the concept to be learned, while a **negative instance** stores a set incompatible with the concept. A **recognizer** for a concept is a predicate written in some **recognizer language (RL)**. A recognizer identifies only positive instances of a concept. A recognizer may be equated with what we have been calling a **concept description**.

Let us describe a subset of the RL that is of particular interest. The **efficient recognizer language (ERL)** contains only those RL terms that describe either (i) features encoded directly in instances, or (ii) features efficiently computable from encoded features. Any RL term that references other types of features is part of the **IRL (inefficient recognizer language)**. Finally, we will define a recognizer expressed solely in terms of the ERL to be an **operational recognizer******. Notice that for the arch example, the RL is a language containing both functional and structural terms, while the ERL is restricted to structural terms and the IRL is restricted to functional terms. Thus, the structural arch recognizer is operational, while the functional arch recognizer is not.

We are now in a position to define our task:

Concept Operationalization Task

Given:
1. Recognizer language RL,
2. Efficient recognizer language ERL where ERL $\subset$ RL,
3. Inefficient recognizer language IRL where IRL=RL-ERL,
4. Recognizer R, expressed in RL, containing IRL terms.

Find:

An operational recognizer, ER, expressed in the ERL, which recognizes the same instances as R.

***Cost is measured by cpu time expended to reach tne goal state

****All components of this version have been implemented and tested on several examples, but have not been completely interfaced.

*****Since the generalized instance represents a set of positive instances, the LEX analysis technique has the effect of squeezing a number of training instances out of a single one. LEX does not currently generalize negative instances in this manner, although the corresponding analysis is feasible in principle.

******In formulating this problem, we make the simplifying assumption that a recognizer is either operational or nonoperational. The operationality of a recognizer is more properly considered to be a matter of degree.

Informally, the task here is to move a recognizer into the ERL so that it can be used for efficient recognition. Since the set of positive instances recognized by R is never explicitly given, a major difficulty with this task lies in proving that a candidate ERL recognizer will identify the same set of positive instances as R. Our approach to the task addresses this problem by applying a series of transformations to R that have well-defined effects on the set of instances recognized by R. Another problem with this task concerns what to do when there exists no ERL recognizer that identifies all of the positive instances. There are primarily two options available in this case: (i) settle for an ERL recognizer that identifies a subset of the positive instances, or (ii) expand the ERL until an appropriate recognizer is included in the ERL. The second option involves finding an appropriate IRL term to incorporate into the ERL, and then developing an efficient method of computing the feature described by the IRL term. This approach has been pursued by Utgoff [10].

We are now in a position to illustrate how we view LEX's learning task in terms of operationalization. Figure III-1 presents LEX's non-operational Minimum Cost Criterion, phrased in terms of a recognizer in predicate calculus. The POS-INST-0 predicate recognizes only productive operator application instances. The instance data structures are pairs of the form (OP,STATE), where OP is to be applied to the integral calculus expression contained in STATE. In order for (OP,STATE) to be a positive instance, POS-INST-0 specifies that STATE must contain a non-solved integral expression, and that the application of OP to STATE must produce a new state that lies on the path to a solved expression. Furthermore, no other path should lead to a less costly solution.

Note that POS-INST-0 contains reference to features (e.g. solvability and cost) which are neither directly encoded nor easily derivable from the instance data structure. For example, to determine whether STATE is SOLVABLE, it is necessary to complete an exhaustive search of the subtree beneath STATE. Furthermore, to compute the cost of STATE requires a complete record of all operators applied to reach STATE from the root of the search tree. Due to these inefficiencies, POS-INST-0 is non-operational and terms such as SOLVABLE and MORE-COSTLY-STATE are part of LEX's IRL.

On the other hand, consider the following example recognizer which is expressed in LEX's ERL:

POS-INST-A(OP3,state)←
    MATCH($\int$trig(x)·poly(x)dx, Expression(state))

ERL terms have been restricted to apply solely to features of integral calculus expressions. If specific features are present in an expression, then the features are said to MATCH the expression. A feature MATCH is efficiently computable from the instance data structure with the aid of a grammar for calculus expressions [10]. For example, if we want to evaluate POS-INST-A(OP3, $\int \sin(x)\cdot 3x^2 dx$), we invoke MATCH to determine whether sin(x) is a trigonometric function and $3x^2$ is a polynomial function.

In the next section will illustrate how POS-INST-0 can be transformed into a more efficient ERL recognizer.

IV Operationalization Techniques

Each operationalization technique discussed in this section specifies a transformation that can be applied to a given recognizer in order to produce a new, and hopefully more efficient recognizer. The set of techniques described is intended to be general, although not comprehensive. Consult [8] for Mostow's thorough treatment of a broader spectrum of operationalization techniques.

A. General description of techniques

In Table IV-1 we describe and characterize a number of operationalization techniques: **definitional expansion**, **enumeration**, **redundancy elimination**, **constraint propagation**, **disjunct selection**, **instantiation**, **conjunct addition** and **conjunct deletion**. Each technique consists of a single replacement rule that can be used to transform a predicate calculus subexpression found in a recognizer. For example, we can use Definitional Expansion-1 to transform **Foo(x)∧Baz(x)** into **Bar(x)∧Baz(x)** if we know that Foo(x)↔Bar(x). Starting with a non-operational recognizer, the replacement rules are sequentially applied with the goal of transforming IRL terms into ERL terms. This process will be illustrated in the next subsection.

Figure III-1: IRL Recognizer for productive operator applications

```
POS-INST-0(op,state)↔ ~GOAL(state)
                 ∧ APPLICABLE(op,state)
                 ∧ SOLVABLE(Successor-state(op,state))
                 ∧ [(∀otherop∈operators | otherop≠op)
                        ~APPLICABLE(otherop,state)
                      ∨ ~SOLVABLE(Successor-state(otherop,state))
                      ∨  MORE-COSTLY-STATE(Goal-state-reachable(otherop,state),
                                          Goal-state-reachable(op,state))]
SOLVABLE(state)↔ GOAL(state)
                 ∨ ∃op [APPLICABLE(op,state)
                        ∧ SOLVABLE(Successor-state(op,state))]
```

<u>**Meaning of PREDICATES (uppercase) and Functions (capitalized):**</u>

- POS-INST-0(op,state): the application of op to state is productive
- SOLVABLE(state): there is a path leading from state to a goal state
- APPLICABLE(op,state): legal to apply op to state
- GOAL(state): state is a goal state
- MORE-COSTLY-STATE(state1,state2): cost to state1 exceeds cost to state2
- Successor-state(op,state): returns state resulting from application of op to state
- Goal-state-reachable(op,state): returns goal state at the end of the path starting with the application of op to state

TECHNIQUE	REPLACEMENT RULE
CONCEPT-PRESERVING TRANSFORMS	
Definitional Expansion-1	Replace A with B if A ⟷ B
Enumeration	Replace [∀x]f(x) with f(A)∧f(B)∧...
Redundancy Elimination	Replace (A∧(B∧A)) with (B∧A)
Constraint Propagation	Replace c(X,f(Y)) with c(Z,Y), where $Z=f^{-1}(X)$
CONCEPT-SPECIALIZING TRANSFORMS	
Definitional Expansion-2	Replace A with B if B ⟶ A
Conjunct Addition	Replace (A∧B) with (A∧B∧C)
Disjunct Selection	Replace (A∨B∨C) with B
Instantiation	Replace ∃x|f(x) with f(A)
CONCEPT-GENERALIZING TRANSFORMS	
Conjunct Deletion	Replace (A∧B∧C) with (A∧C)
Definitional Expansion-3	Replace A with B if A ⟶ B

Table IV-1: OPERATIONALIZATION TECHNIQUES

It is useful to view the operationalization process in terms of a heuristic search through a space of concepts, where states in the space are recognizers and operators are operationalizing transforms. The starting state represents the initial IRL recognizer. The goal states contain ERL recognizers that identify the same positive instances as the initial IRL recognizer. (Failing this, the goal states are those that identify the largest number of positive instances.) There are various pieces of knowledge that can be used to guide this search. Broadly, these include:

- Knowledge about the effect of a transform on the set of instances recognized as positive. Any transform which modifies this set should be avoided.

  Ex: If only **concept-preserving transforms** are used during concept operationalization, the set of instances recognized by the final ERL recognizer is guaranteed to be identical to the set recognized by the initial IRL recognizer. On the other hand, **concept-specializing transforms** reduce the set recognized as positive, while **concept-generalizing transforms** enlarge the set so that some negative instances will be falsely included as positive. Concept-generalizing transforms are therefore the most dangerous type to apply during concept operationalization.

- Definitional knowledge about which IRL terms have ERL expansions. This knowledge can be used in means-ends guidance.

  Ex: A definition that expresses GOAL in terms of MATCH translates between the IRL and the ERL. Re-expressing parts of the original IRL recognizer in terms of GOAL is therefore a useful subtask.

- Examples of transformation sequences that have resulted in useful ERL recognizers in the past. These can be used to focus the search.

  Ex: To provide guidance in the selection and expansion of clauses in an IRL recognizer, it may be possible to use a previously-formulated transformation sequence as a a kind of macro-operator. In this way, the construction of a new sequence could be guided by previous experience.

- Domain-specific knowledge that can be used to prevent the over-specialization of a recognizer.

  Ex: Any recognizer containing the following instantiated definitional expansion of SOLVABLE (defined in Figure III-1) is legal, but identifies no instances at all:

  GOAL(Successor-state(OP3,Successor-state(OP1,state)))

  We know that the use of this expansion causes over-specialization because our knowledge about integration operators informs us that OP3 cannot actually be applied to any state produced by OP1.

- Knowledge about the goals and constraints of the learning system. This type of knowledge can serve to justify the use of a concept-altering transform.

  Ex: If the predicate **Red(x)** is not in our ERL, it may be necessary to perform Conjunct Deletion to remove it from the expression **Red(x)∧Round(x)∧Heavy(x)**. To justify the use of this concept-generalizing transform, it may be helpful to know that i) the goal of the system in question involves (for example) learning how to make efficient use of mechanical equipment, and that ii) color does not generally effect mechanical properties.

B. Operationalization techniques applied to LEX

To more fully illustrate the techniques introduced in Section IV.A, we would like to illustrate operationalization in the context of a specific hand-generated example. In particular, consider the task of operationalizing LEX's IRL recognizer for productive operator applications (POS-INST-0 in Figure III-1). One possible transformation sequence is depicted in Figure IV-2.

Figure IV-2: Operationalization of POS-INST-0

```
POS-INST-0  (from Figure III-1)
 ↓ Via Disjunct Selection on ~APPLICABLE
POS-INST-1(op,state)←  ~GOAL(state)
  ∧  APPLICABLE(op,state)
  ∧  SOLVABLE(Successor-state(op,state))
  ∧  [(∀otherop∈operators|otherop≠op)
           ~APPLICABLE(otherop,state)]
 ↓ Via Definitional Expansion-1 on SOLVABLE
POS-INST-2(op,state)←  ~GOAL(state)
  ∧  APPLICABLE(op,state)
  ∧  [GOAL(Successor-state(op,state))
          ∨ ∃opx{APPLICABLE(opx, Successor-state(op,state))
                ∧ SOLVABLE(Successor-state(opx,
                               Successor-state(op,state)))}]
  ∧  [(∀otherop∈operators|otherop≠op)
           ~APPLICABLE(otherop,state)]
 ↓ Via Disjunct Selection on GOAL
POS-INST-3(op,state)←  ~GOAL(state)
  ∧  APPLICABLE(op,state)
  ∧  GOAL(Successor-state(op,state))
  ∧  [(∀otherop∈operators|otherop≠op)
           ~APPLICABLE(otherop,state)]
 ↓ Via Instantiation of op,Enumeration of ~APPLICABLE
POS-INST-4(OP1,state)←  ~GOAL(state)
  ∧   APPLICABLE(OP1,state)
  ∧   GOAL(Successor-state(OP1,state))
  ∧  ~APPLICABLE(OP2,state)
  ∧  ~APPLICABLE(OP3,state)
  •      •
  ∧  ~APPLICABLE(OPN,state)
 ↓ Via Def. Expansion-1 on GOAL and APPLICABLE
POS-INST-5(OP1,state)←  ~MATCH(expr-with-no-∫,state)
  ∧   MATCH(∫sin(x)dx,state)
  ∧   MATCH(expr-with-no-∫,Successor-state(OP1,state))
  ∧  ~MATCH(∫k·f(x)dx,state)
  ∧  ~MATCH(∫f1(x)·f2'(x)dx,state)
  •      •
  ∧  ~MATCH(Preconditions(OPN),state)
    Via Constraint Propagation
 |   MATCH(expr-with-no-∫,Successor-state(OP1,state))
 ↓          ⟹ MATCH(f(x)∫sin(x)dx,state)
    and Redundancy Elimination
POS-INST-6(OP1,state)←  MATCH(f(x)∫sin(x)dx,state)
```

To motivate this operationalization sequence, consider applying means-ends analysis to the task. In order to arrive at an ERL recognizer, it is necessary to apply a technique that translates IRL predicates into ERL predicates. The only such technique available is Definitional Expansion. Among those predicate definitions that translate between the IRL and the ERL are:

APPLICABLE(op,state) $\leftrightarrow$
  MATCH(Preconditions(op), Expression(state)), and
GOAL(state) $\leftrightarrow$
  MATCH(term-with-no-$\int$, Expression(state)).

The transformation sequence shown can be viewed as an attempt to re-express POS-INST-0 solely in terms of APPLICABLE and GOAL. This has been accomplished in POS-INST-4. From this point, it is easy to move into the ERL. The final step of the sequence produces POS-INST-6, which recognizes the application of OP1 to STATE as a productive operator application whenever **$\int$sin(x)dx** appears within the numeric expression contained in STATE.

While LEX does not actually represent the operationalizing transforms of Table IV-1 explicitly, there is a close relationship between the operationalization sequence in Figure IV-2 and its counterpart in LEX. In particular, the process (explained in Section II) that LEX goes through in analyzing a single positive instance can be viewed as the application of a sequence of operationalizing transforms. This sequence changes POS-INST-0 into a very specific ERL recognizer that identifies only the single instance. However, the transformation sequence can then be used a template for the construction of a new sequence leading to a more general recognizer. By using the equivalent of a template, the LEX implementation eliminates much of the search process inherent in operationalization. Selection and expansion of clauses is carried out in accordance with the template, and Constraint Propagation and Redundancy Elimination are automatically invoked after IRL predicates have been translated into the ERL. The use of concept-generalizing transforms is avoided altogether. These pre-compiled decisions have made the LEX operationalization task manageable, and have permitted us to avoid some difficult control issues that arise in the full-blown concept operationalization scenario described in Section IV.A.

V Conclusions

The operationalization techniques described in this paper are not particularly novel; similar methods have been applied to automated design and synthesis tasks (see, for example, work in automatic programming [4, 1]). What is different about our approach, along with Mostow's [9], is the explicit application of these techniques in the context of learning and problem solving [7, 6]. We are beginning to acknowledge that design activities are intimately related to learning abilities, and that the ability to use one's knowledge appropriately to achieve a particular goal (i.e. to design a solution to a problem) is a fundamental learning skill.

It is clear that much work remains to be done in the area of controlling the search for solutions to the concept operationalization task. In particular, we need to understand how various sources of knowledge can be used not only to guide, but also to *justify* the operationalization process. Justification involves the defense of operationalization decisions based on goals, constraints and preferences operating within the concept learning environment. Based on the nature of recent research in machine learning, more attention to such environmental factors is central to progress in this field.

VI Acknowledgments

I would like to thank Tom Mitchell for his significant contribution to the ideas expressed in this paper, and for his help and encouragement. Many of my ideas have also been clarified by discussions with Smadar Kedar-Cabelli. I thank Paul Utgoff for his assistance in implementation and for several careful readings of earlier drafts of this paper. John Bresina, Donna Nagel, Don Smith, and Peter Spool also supplied useful comments.

References

[1] Barstow, D. R., *Automatic Construction of Algorithms and Data Structures Using a Knowledge Base of Programming Rules*, Ph.D. dissertation, Stanford University, November 1977.

[2] Carbonell, J. G., Michalski, R. S. and Mitchell, T. M., "An Overview of Machine Learning," *Machine Learning*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga, 1983.

[3] Hayes-Roth, F., Klahr, P., Burge, J. and Mostow, D. J., "Machine Methods for Acquiring, learning, and Applying Knowledge", Technical Report R-6241, The RAND Corporation, 1978.

[4] Manna, Z. and Waldinger R., "Knowledge and Reasoning in Programming Synthesis," *Studies in Automatic Programming Logic*, Manna, Z. and Waldinger R. (Eds.), North-Holland, 1977.

[5] Mitchell, T. M., Utgoff, P. E. and Banerji, R. B., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," *Machine Learning*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga, 1983.

[6] Mitchell, T., "Learning and Problem Solving," *Proceedings of IJCAI-83*, Karlsrhue, Germany, August 1983.

[7] Mitchell, T. M. and Keller, R. M., "Goal Directed Learning," *Proceedings of the Second International Machine Learning Workshop*, Urbana, Illinois, June 1983.

[8] Mostow, D. J., *Mechanical Transformation of Task Heuristics into Operational Procedures*, Ph.D. dissertation, Carnegie-Mellon University, 1981.

[9] Mostow, D. J., "Machine Transformation of Advice into a Heuristic Search Procedure," *Machine Learning*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga Press, Palo Alto, 1983.

[10] Utgoff, P. E. and Mitchell, T. M., "Acquisition of Appropriate Bias for Inductive Concept Learning," *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, August 1982.

[11] Winston, P. H., "Learning Structural Descriptions from Examples," *The Psychology of Computer Vision*, Winston, P. H. (Ed.), McGraw Hill, New York, 1975, ch. 5.

[12] Winston P.H., Binford T.O., Katz B. and Lowry M., "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," *Third National Conference on Artificial Intelligence*. Washington, D.C., 1983.