REPRESENTATION AND INDUCTION OF INFINITE CONCEPTS AND RECURSIVE ACTION SEQUENCES

Fritz Wysotzki

Dept. of Artificial Intelligence
Central Institute of Cybernetics and
Information Processes of the Academy of Sciences
1086 Berlin, German Democratic Republic

ABSTRACT

A general method for the inference of recursive functions (programs) from examples of computations is described. It is based on the so called algebraic semantics of recursive program schemes *and* in this paper mainly applied to the representation and induction of infinite concepts and recursive action sequences (which are of importance for problem solving and hierarchical planning). Additionally, the use of recursive program schemes leads to a new principle of generalization in the sense that families of structures or classes of programs could be treated simultaneously and that already existing solutions of old problems could be transferred to new problems which have to be solved.

1. Introduction

Program synthesis by induction as opposed to program synthesis by deduction from input-output specifications becomes a matter of increasing interest in AI and Computer Science. In this field much work has dealt with the synthesis of LISP programs from examples of computations /3/, /V, /5/.
We are interested on the one hand in the development of a general formalism not depending on a specific programming language and on the other hand in the induction of certain infinite concepts and recursive action sequences extending in this way our earlier work on concept learning /6/, /7/. By infinite concepts we mean classes of strings and other stuctures having some intrinsic periodicity (In Pattern Recognition only feature vectors of fixed dimensionality have been considered so far). Our method is based on the so called algebraic semantics and recursive program schemes */1/,* /2/ and it is intended to provide a general framework for the abovementioned representation and induction problems* The induction principle we introduce in this paper is a more general version of SUMMERS' main synthesis theorem /5/. Our main aim is to apply it to the representation and induction of recursive action sequencee from examples of observed behaviour. This is also of importance for problem solving and hierarchical planning.
Section 2 starts with a brief survey on the basic definitions together with examples taken from the recursive operation of clearing a block and preparing in this way the more detailed discussion of the CLEAR-operator in section 5. In section 3 the induction principle is stated and then applied to two basic examples in section 4 and 5 demonstrating the general method. The consideration of synthesized programs as recursive program schemes (RPS) allows us to introduce a new principle of generalization not investigated in AI until now.

2. Basic definitions

Term algebras, trees, rewriting rules, program schemes

In the following we use the notations and definitions of /1/. Let $V_k = \{v_1,\ldots,v_k\}$ and $V = \bigcup_{k\geq 1} V_k$ be sets of variables and F a set of primitive function symbols with arity or rank ($f \in F$ has arity $\varrho(f)$). F contains a distinguished symbol Ω representing the "undefined". Then let M(F,V) be the set of finite well-formed terms on $F \cup V$. Terms are inductively defined by

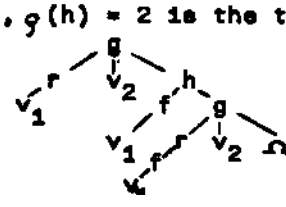
Definition 2.1.

1. F_0 is a term (F_0 = constants)
2. If $t_1,\ldots,t_n \in M(F,V)$ are terms then $f(t_1,\ldots,t_n)$ with $f \in F$, $\varrho(f) = n$ is a term.

Terms can be represented as trees.

Example 2.1.

The term $t = r(v_1, v_2, h(f(v_1), g(r(f(v_1)), v_2, \Omega)))$ with $f, g, r, h \in F$, $\varrho(f) = 1$, $\varrho(g) = 3$, $\varrho(h) = 2$ is the tree



(For the interpretation as a partially defined program for clearing a block see example 2.4.)

Definition 2.2.
On M(F,V) a partial order $\prec$ is defined, intuitively: $t \prec t'$ if t' can be generated from t by substitution of Ω by terms $\neq \Omega$ at some positions in t. Ω is the least element of $\prec$.
Every directed subset $A \subseteq M(F,V)$ has an upper limit $Sup(A)$ which is a finite or infinite tree. This allows one to extend M(F,V) to the $M^{\infty}(F,V)$ of infinite trees (which we will designate by T as opposed to finite trees t).
In addition to the primitive function symbols $f \in F$ we consider a set $\vec{\Phi} = \{G_1,\ldots,G_n\}$ of function variables with arity $\varsigma(G_i) = k_i$ which will later on play the role of "subprograms".
Let $M^{\infty}(F \cup \vec{\Phi},V)$ be the term algebra extended by $\vec{\Phi}$.

Definition 2.3.
A recursive program scheme (RPS) is a pair $\langle \Sigma, t \rangle$ consisting of a system of equations
$\Sigma = \langle G_i(v_1,\ldots,v_{k_i}) = t_i \mid 1 \leq i \leq n \rangle$
and a term $t \in M(F \cup \vec{\Phi}, V_k)$, $G_i \in \vec{\Phi}$, $t_i \in M(F \cup \vec{\Phi}, V_{k_i})$.
t plays the role of the "main program", which is composed of the "subprograms" G_i mutually "calling each other" by means of Σ.

Example 2.2.
Σ: $G(v_1,v_2) = g(r(v_1), v_2, h(f(v_1), G(f(v_1), v_2))) = t$
(For the interpretation as a program of clearing a block see ex. 2.4)
A system Σ can be "solved" using the following rewriting rule by which function variables in the t_i's can be iteratively eliminated.

Definition 2.4. (rewriting rule)
By this definition we allow in a tree t subtrees beginning with function variables $G \in \vec{\Phi}$ to be substituted by other trees $\Theta(G)$ (which appropriate correspondencee in the variables).
Formally: A tree t is transformed into another tree $\hat{\Theta}(t)$ inductively defined by
1) $\hat{\Theta}(x) = x$ if $x \in F_0 \cup V$
2) $\hat{\Theta}(f(t_1,\ldots,t_k)) = f(\hat{\Theta}(t_1),\ldots,\hat{\Theta}(t_k))$
3) $\hat{\Theta}(G(t_1,\ldots,t_k)) = \Theta(G)\,[\hat{\Theta}(t_1)/v_1,\ldots,\hat{\Theta}(t_k)/v_k]$

In 3) the r.h.s. means that in the tree $\Theta(G)$ the variable v_i is to be replaced by $\hat{\Theta}(t_i)$, $1 \leq i \leq k$. (Note that $\hat{\Theta}(t_i) = t_i$ if t_i contains no function variables.)
Instead of $\hat{\Theta}(t)$ we will also write

$t\,\{t_1/G_{i_1},\ldots,t_n/G_{i_n}\}$

if in t G_{i_j} is to be replaced by t_j.
Example: see example 2.3

Definition 2.5
Let Σ be a RPS as in def. 2.3. For $1 \leq i \leq n$ and $l = 0, 1, 2, \ldots$ we define a sequence $G_i^{(l)}$ by

$G_i^{(0)} = \Omega$, $G_i^{(l+1)} =$
$= t_i\,\{G_1^{(l)}/G_1,\ldots,G_n^{(l)}/G_n\} =$
$=_{Df} t_i\,\{\vec{G}^{(l)}/\vec{G}\}$

using the substitution rule of def. 2.4.
This sequence is called KLEENE-sequence.
It can be shown that $G_i^{(l)} \prec G_i^{(l+1)}$, $l = 0,1,\ldots$ and that $\mathrm{Sup}_l\, G_i^{(l)} =_{Df} T(\Sigma, G_i)$, $1 \leq i \leq n$ is the fixpoint solution (least solution) of Σ.

Example 2.3.
For the RPS of ex. 2.2. we define
$G^{(0)}(v_1,v_2) = \Omega$, $G^{(1)}(v_1,v_2) = g(r(v_1), v_2, h(f(v_1), \Omega))$
$G^{(2)}(v_1,v_2) = g(r(v_1), v_2, h(f(v_1), G^{(1)}(f(v_1), v_2)))$
$= g(r(v_1), v_2, h(f(v_1), g(r(f(v_1)), v_2, h(f^2(v_1), \Omega))))$, ...
$G^{(l+1)}(v_1,v_2) = g(r(v_1), v_2, h(f(v_1), G^{(l)}(f(v_1), v_2))),\ldots$
using the rewriting rule of def. 2.4. with $\hat{\Theta}(t) = G^{(l+1)}(v_1,v_2)$, $\Theta(G) = G^{(l)}(v_1,v_2)$.
Note that this KLEENE-sequence is generated by the following procedure which we will use for induction (section 3): There is a tree $tr = G^{(1)} = g(r(v_1), v_2, h(f(v_1), \Omega))$ and a fixed node m in tr (labelled by Ω) and $G^{(l+1)}$ is constructed by substituting $G^{(l)}\,[\vec{t}/\vec{v}] = G^{(l)}(f(v_1), v_2)$ at node m $(l = 2,3,\ldots)$. $G^{(l)}\,[\vec{t}/\vec{v}]$ means that the vector of variables $\vec{v} = (v_1,v_2)$ of $G^{(l)}$ has to be replaced by the vector of trees $\vec{t} = (f(v_1), v_2)$.

Interpretation and valuation
Until now we have considered syntactical structures only. By an interpretation I we assign to every function symbol f with arity i a function on some domain

$(D_{v_1} \times D_{v_2} \times \ldots \times D_{v_i})_I$. (We have to take into account sorted functions and variables $v_1,\ldots,v_i$).

$\preceq$ is interpreted by $\leq_I$, Ω by $\perp$ (the "undefined"). An interpretation assigns to each RPS a program. By a valuation we assign to each variable v_j a value from domain $(D_j)_I$. An interpretation together with a valuation assigns to a RPS a value (the result of the computation by Σ).

Example 2.4. (see fig. 1 and example 2.2.) Let $T, F \in F_0$ be the truth values "true" and "false" respectively and in example 2.2. g be the McCARTHY-conditional

$$g(x,y,z) \Leftarrow \text{if } x \text{ then } y \text{ else } z$$

$$g(T,y,z) = y,\ g(F,y,z) = z$$

$$g(\perp,y,z) = g(T,\perp,z) = g(F,y,\perp) = \perp$$

($\Leftarrow$ denotes interpretation)
Let v_1 be a variable denoting definite objects x (blocks), r(x) the predicate CT(x) = cleartop (x), f(x) the logical function $\iota u(ON(u,x))$ ("that block u lying on block x"), v_2 a state variable s and h(x,s) $\Leftarrow$ put (x,s) ("put x in state s on the table" getting the new state s' = put (x,s)). Then

$$G(v_1,v_2) \Leftarrow \text{CLEAR } (x,s)$$

is the (recursive) CLEAR-operator. In the case of fig. 1 we have

$$\text{CLEAR } (x,s) = \text{put } (f(x), \text{put } (f^2(x), g(CT(f^2(x)),s,\ldots) \ldots)$$
$$= \text{put } (f(x), \text{put } (f^2(x),s))$$

since g $(CT(f^2(x)), s, \ldots) = s$

s (initial state) s' = CLEAR (x,s) (final state)

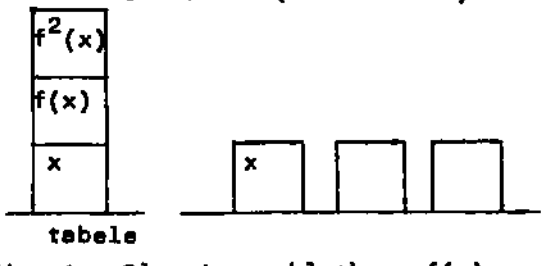


Fig. 1 Clearing a block x; $f(x) =_{Df} \iota u(ON(u,x))$
(ι : jota operator)

3. Principle of induction

Our main aim is to construct (recursive) programs from examples of computations. The induction principle is a generalization of the synthesis theorem of SUMMERS /6/. (The convergence properties are already established by our use of the KLEENE-sequence, def. 2.5., for induction.) In the first stage we build from a finite set of examples a finite tree $\hat{t}$ either by a learning algorithm or by a combination of alternatives similar to /5/. Then we try to interpret $\hat{t}$ as an element of a KLEENE-sequence of some RPS which has to be induced. Formally: If there exists a tree $tr \in M(F \cup \vec{\Phi}, V)$, a fixed node m in tr, a vector $\vec{t} = (t_1,\ldots,t_n)$ of trees $t_i \in M(F \cup \vec{\Phi}, V)$ $(i = 1,\ldots,n)$ and a sequence

$$t^{(0)} = \Omega,\ t^{(1)} = tr(\Omega/m),$$
$$t^{(2)} = tr(t^{(1)}\ [\vec{t}/\vec{v}]/m),\ldots,\hat{t} =$$
$$tr(t^{(1^*-1)}\ [\vec{t}/\vec{v}]/m)$$

then we consider $\hat{t}$ as the 1^*th element of a KLEENE-sequence. $t[\vec{t}/\vec{v}]$ means that the vector $\vec{v}$ of variables of t has to be replaced by the vector $\vec{t}$ and tr(t/m) means that m is the root node of subtree t in tr. (In general the $t^{(1)}$'s are defined only for some specific values of the v_i's and the domain has to be extended by the induction process.) The method can be generalized to the case where several nodes m_k in tr have to be replaced by "macro-functions" (see section 5). Having established $\hat{t}$ as $t^{(1^*)}$ and 1^* being sufficiently high we extrapolate the sequence for all l and get the simplest hypothesis i. e. RPS) which has $\hat{t}$ as the 1^*th element of its KLEENE-sequence. 1^* can be regarded as a measure of the reliability of the hypothesis. In sections 4 and 5 we demonstrate the method by two basic examples.

4. Induction of recursive Boolean expressions

We consider the following classification problem on the set $\{a_1a_2\ldots a_k \mid k = 1,2\ldots\}$ of binary strings ($a_j \in \{0,1\}$ for $1 \leq j \leq k$): all strings with $a_i = 1$ for $i = 1,\ldots,k$ belong to the class T ("true") and all other strings (i. e. containing at least one $a_j = 0$) belong to the class F ("false"). From a finite training sample of strings the algorithm has to infer the classification function for strings of arbitrary length. Our set of primitive functions g, x, r, $s \in F$ (see section 2) is defined as followe

Definition 4.1.
g is the McCARTHY conditional as in ex. 2.4.

$$x(i) =_{Df} \begin{cases} T \text{ if } a_i = 1 \\ F \text{ if } a_i = 0 \\ \text{undefined} \end{cases} \quad \begin{matrix} 1 \leq i \leq k \\ \text{(i. e. if } a_i \text{ defined)} \\ i > k \text{ (i. e. if } a_i \text{ not def.)} \end{matrix}$$

$$r(j) =_{Df} \begin{cases} T \text{ if } x(j) \text{ defined} \\ F \text{ if } x(j) \text{ undefined} \end{cases} \quad j = 1,2,\ldots$$

$s(k) = k + 1 \quad k = 0, 1, 2, \ldots$

x(i), r(j) are tests to be performed on strings, s is the successor function, T, $F \in F_0$ are the truth values for tests as in ex. 2.4. Suppose we have a training sample of all strings up to length k = 2, strings consisting only of symbols 1 being classified as $\overline{T}$, strings containing at least one 0 as $\overline{F}$. Then by a slight modification of an algorithm for automatically constructing decision trees from training samples of feature vectors /6/, /7/ the following initial tree $\hat{t}$ can be built

$$\hat{t} = g(x(1), g(r(2), g(x(2), g(r(3), \Omega, \overline{T}), \overline{F}), \overline{T}), \overline{F})$$

$$= g(x(1), g(r(s(1)), g(x(s(1)), g(r(s^2(1)), \Omega, \overline{T}), \overline{F}), \overline{T}), \overline{F})$$

This tree classifies (by definitions 4.1.) the strings 1 and 11 as $\overline{T}$, all other strings with length ≤ 2 as $\overline{F}$ and strings with length > 2 as Ω.
According to section 3 we have to search for a tree tr which appears as a subtree of itself (at a fixed node m) recursively with appropriate substitutions $\overline{t}/\overline{v}$ of variables. (That is we use a matching procedure with appropriate replacement of variables.)
Observing the "periodicity" of the appearance of the symbols r and x respectively in $\hat{t}$ at the same position in subtrees beginning with g we define

$t^{(0)} =_{Df} G^{(0)} = \Omega$, $t^{(1)} =_{Df} G^{(1)}(1) = g(x(1), g(r(s(1)), \Omega, \overline{T}), \overline{F}) = tr$

$\hat{t} =_{Df} G^{(2)}(1) = g(x(1), g(r(s(1)), \underline{G^{(1)}(s(1))}, \overline{T}), \overline{F})$

with

$$\underline{G^{(1)}(s(1))} = G^{(1)}(2) = g(x(s(1)), g(r(s^2(1)), \Omega, \overline{T}), \overline{F})$$

That is we have for $0 \leq l \leq l^*$, $l^* = 2$ a KLEENE-sequence

$$G^{(l+1)}(n) = g(x(n), g(r(s(n)), G^{(l)}(s(n)), \overline{T}), \overline{F})$$

which is partially defined for $1 \leq n \leq l^* - (l - 1)$ $(l = 1,2; l^* = 2)$ and undefined for other values of n. In the sense of section 3 we have (for n = 1,2)

$$tr = g(x(n), g(r(s(n)), c, \overline{T}), \overline{F})$$

and $\hat{t} = t_1 = s(n)$. c is an arbitrary tree labelling the node m mentioned in section 3.
Now we want to get the simplest hypothesis for continuation of the KLEENE-sequence. The simplest hypothesis is to extrapolate $G^{(l)}$ for all l and fpr all n (i. e. extending the domain) using the successor function s(n) on the natural numbers. This way we get the recursive program

$$\Sigma: G(n) = g(x(n), g(r(s(n)), G(s(n)), \overline{T}), \overline{F})$$

G(1) accepts (classifies) all strings containing only 1's as $\overline{T}$ all others as $\overline{F}$ and our starting tree $\hat{t}$ is the second element of its KLEENE-sequence. (With other words: Σ accepts for n = 1 a logical conjunction of arbitrary length. For $n > 1$ the first n - 1 elements of a string are ignored.) Note that the reliability of the hypothesis is low because we have used examples up to $l^* = 2$ only.

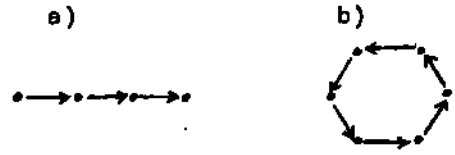


Fig 2 Chain and circuit

Now we introduce a new principle of generalization. We abstract from the interpretation of the function symbols G, g, x, r, $\overline{T}$, $\overline{F}$ used so far (see def. 4.1.) and consider Σ as a pure syntactical structure (i. e. RPS). If we interpret g, x, r as in def. 4.1. but $\overline{T}$ by $\overline{F}$ and $\overline{F}$ by $\overline{T}$ then Σ accepts all strings consisting only of 1's as $\overline{F}$ and all others as $\overline{T}$. If we additionally interpret x by $\sim x$ (negation of x) then Σ accepts all strings containing at least one 1, i. e. a logical disjunction of arbitrary length. With another interpretation of x, r, we could make Σ accept all graphs consisting of a directed chain (fig. 2a) and with still another interpretation all directed circuits (fig 2b). This means that Σ read as a RPS accepts families of structures which can be considered as being equivalent in this sense. This could be used to avoid the full induction process for a new problem for which one has a starting tree $\hat{t}$. Before performing the induction one could try to find a structural isomorphism of $\hat{t}$ with an element of a KLEENE-sequence of an already existing RPS (solution of an old problem) being at the same time a syntactical solution of the new problem too. Note that this procedure may also be regarded as some kind of analogical reasoning in which functions (or relations) would be replaced by other functions (relations).

5. Learning action sequences

Now we consider a more complicated example in which several nodes m_k in the tree tr (see section 3) have to be replaced by "macrofunctions". We return to the problem

of clearing a block x (fig. 1) which we have dealt with in examples 2.2. - 2.4. The variables and primitive functions are as in ex. 2.4. Suppose we have the following three examples of observed behaviour (training instances):

CT(x) → s ("if clartop (x) then nothing is to be done")

CT(f(x)) → h(f(x), s) ("if the block f(x) lying on x is clear then put f(x) on the table in state s")

$CT(f^2(x))$ → h(f(x), $h(f^2(x)$, s)) (two blocks on x, repeated clearing)

Combining these alternatives by means of the function g to a initial tree $\hat{t}$ leads to

$$\hat{t} = g(CT(x), s, g(CT(f(x)), h(f(x), s), g(CT(f^2(x)), h(f(x), h(f^2(x), s)), \Omega))) \quad (5.1)$$

From the first position in each subtree beginning with g we get the sequence

$CT(x), CT(f(x)), CT(f^2(x))$

and define

$$F(i,x) =_{Df} f^i(x) \quad i = 0,1,2;\ F(0,x) = x \quad (5.2)$$

With this definition we get from the second position in the subtrees beginning with g the sequence

s, h(F(1,x), s), h(F(1,x), h(F(2,x), s)).

The simplest hypothesis is to continue by

h(F(1,x), h(F(2,x), h(F(3,x), s))), ...

i. e. by using the successor function s(n) for extrapolation. With a function

$$H'(i,n,x,s) =_{Df} g(i> n,s,h(F(i,x), H'(s(i),n,x,s))) \quad (5.3)$$

the above sequence can be written as

$$H(n,x,s) =_{Df} H'(1,n,x,s);\ H(0,x,s) =_{Df} s \quad (5.4)$$

and the tree $\hat{t}$ (5.1) as

$$\hat{t} = g(CT(F(0,x)), H(0,x,s), g(CT(F(1,x)), H(1,x,s), g(CT(F(2,x)), H(2,x,s), \Omega))) \quad (5.5)$$

The simplest hypothesis is again to extrapolate by means of the successor function. Using the induction methods introduced in section 3 and 4 we get this way a KLEENE-sequence $G'^{(1)}$ and the fixpoint solution

$$\sup_{l \geq 0} G'^{(l)} = G'(n,x,s) = g(CT(F(n,x)), H(n,x,s), G'(s(n),x,s)) \quad (5.6)$$

$n = 0,1,2,...$

Proposition 5.1.

$G'^{(3)}(0,x,s)$ reproduces the tree $\hat{t}$ (5.1)

Proof: By def. 2.5. of the KLEENE-sequence and use of (5.2) - (5.4). G'(0,x,s) clears a block x from an arbitrary number of blocks (fig. 1).

Finally we compare the representation G' (5.6) of the CLEAR-operator with the representation G introduced in examples 2.2. and 2.4. which is the minimal one (but not easy to learn). The equvalence of both representations is established by the following

Proposition 5.2.

There exists a syntactic transformation which transforms G' into G.

Proof: By proving that each element $G'^{(1)}$ of the KLEENE-sequence of G' can be transformed into the corresponding element $G^{(1)}$ of G using the reduction rules

$g(r,h(x,s), \Omega) \rightarrow h(x,g(r,s,\Omega))$

$g(r,h(x,s), h(x,s')) \rightarrow h(x,g(r,s,s'))$

$h(x,\Omega) \rightarrow \Omega$ (5.7)

repeatedly. These rules could be made clear from the semantics in ex. 2.4. (→ means that the l.h.s. of the rule can be replaced by the r.h.s.)

6* Conclusion

A framework for representation and induction of infinite concepts and action sequences has been represented. It was applied to examples taken from typical AI domains, demonstrating the general construction algorithm* The induction principle was formulated with the help of the KLEENE-8equence# Equivalently the concept of a tree grammar could have been applied, i* e. our induction problem is equivalent to the problem of identification of a tree grammar* In our examples we have dealt with the construction of single loops only, multiple (nested) loops can be treated by a more step procedure in a similar manner* The use of program schemes (RPS) instead of programs allows one to treat classes of functions simultaneously* This can be used to avoid the full solution of new problems if there exists already program schemes isomorphic to then*

In future work we will investigate broader classes of functions playing an important rols in AI research especially in picture recognition and planning.

REFERENCES

/1/ Courcelle, B* "Infinite trees in normal form and recursive equations having a unique solution*" Math* System Theory 13 (1979), 131-180.

/2/ Courcelle, B.; Nivat, M. "The algebraic semantics of recursive program schemes*" In: Math. Found. of Comp. Science 1978 (Winkowski eds.). Lecture Notes in Comp* Sciences,Vol 64, Springer-Verlag, New York 1978, 16-30.

/3/ Jouannaud, J P., Guiho, G. "Inference of functions with an interactive system*" Machine Intelligence 9 (1979) 227-250.

/4/ Jouannaud, J. P., Kodratoff, Y. "Characterization of a class of functions synthesized from examples by a SUMMERS-like method using the "B.M.W." matching technique." In Proc. 6, IDCAI, Tokyo 1979.

/5/ Summers, Ph. D. "A methodology for LISP program construction from examples." 0. ACM 24/1 (1977), 162-175.

/6/ Unger, S.; Wysotzki, F. "Lernfahige Klassifizierungssysteme." Akademie-Verlag, Berlin 1981.

/7/ Wysotzki, F.; Kolbe, W.; Selbig, J. "Concept learning by structured examples - an algebraic approach." In: Proc. 7. IJCAI, Vancouver 1981.