SISP/I AN INTERACTIVE SYSTEM ABLE TO SYNTHESIZE FUNCTIONS FROM EXAMPLES

Jean-Pierre JOUANNAUD FRANCE
Maitre-Assistant a 1'Institut de Programination
Universite Paris VI
A, Place Jussieu
75005 PARIS

Gerard GU1HO FRANCE
Maitre de Conference - Laboratoire de Recherche en Informatique
Universite Paris Sud
91405 ORSAY

Jean-Pierre TREUIL FRANCE
Chercheur - Laboratoire de Recherche en Informatique
Universite Paris Sud
91405 ORSAY

The research presented in this paper is supported by IklA-CESORI under contract number 76.

ABSTRACT

SISP/i is an interactive system whose goal is the automatic inference of LISP functions from a finite set of examples {(x., f(x.))) where x. is a list belonging to the domain of the function f we want to infer. SISP/I is able to infer the recursive form of many linear recursive functions and its stop-condition. SISP/I tries to work with one example only. When it fails, it asks for new ones: using then a method of generating new partial subproblems, SISP/I is able to perfect its generated recursive function until it gets a correct one.

I. INTRODUCTION

In this paper we describe the system SISP/I whose goal is the automatic inference of LISP functions from a finite set of examples {(x., l(x.))}, where x. is a list belonging to the domain of the function f we want to infer.

The problem originates from a more general one: how to build a "Learning-Question-Answering-System" (L.Q.A.S.) using a functional method to provide an answer to any given question. The method we propose in SISP/I is naturally well adapted to the L.Q.A.S. we are. developping (6.1, 17 1.

In the field of "Automatic Programming from Examples", an important piece of recent work is THESYS by SUMMERS L5J. The major result of this work, is the following: using a small number of well chosen examples ((NIL, f(NIL)), ((A), f((A)))...} THESYS is able to infer a recursive expression $ equivalent to f for every x belonging to the domain of f.

Only a small class of functions can however be obtained by Summers's method, which works by looking for a recurence relation between representative predicates p. of the given input structure and for a recurence relation between the map functions m. providing the given outputs from the given inputs. Then, using a fixed point theorem, V is constructed .

Although Summers's method is very powerful it has four important drawbacks:

1.- The constructed expression <p is necessarily recursive: for instance the identity function will be infered by V (x) » if X * NIL then NIL
else CONS (CAR(x), ^(CDR(x)))

2.- THESYS needs well chosen examples, which in particular must contain the stop condition of the recursive function V. For instance, the construction of the function REVERSE requires the following set of examples:
{(NIL + NIL), ((A) - (A)), ((A B) - (B A)), ((A B C) + (C B A))}

3.- The function to be constructed has to present only one "iterative level[11]. For instance, THESIS fails to construct a correct function corresponding to the example: (P Q R S) -> (P P Q P Q R P Q R S).

4.- When THESYS has to solve a difficult problem, it does not try to generate a partial, simpler problem for which it could either find a correct solution or perhaps use a knowledge previously stored in a data base by the system itself. Thus, THESYS cannot be efficiently used in a L.Q.A.S. without important modifications.

The method we propose in this paper is very differentin particular, it has the built capacity to use a Professor in interactive mode. It does not lie yet on any theorical groundwork, but allows us to overcome some of the previous drawbacks, although new ones appear:

- recursion is not automatically infered by the synthesis algorithm; for instance, using the example ((A B C) -* (A B C)), STSP/1 infers the function <p v*p (x) = x for any x.
- for some "simple" functions, S'ISP/1 needs only one example (x, f(x)).

In the case where a recursive expression is infered, the stop condition is then found by SISP/I itself. However the. list x must be long enough to be representative of the function f. For instance, REVERSE is obtained using the only example ((A B C D) + (U C B A)), but is not obtained with ((A B C) > (C B A)).

- when the function f is "more complicated" SISP/I fails to construct a correct function with only one example and it then tries to work with two examples.
- when the function f is "much more complicated", SISP/I generates a new partial simpler problem (y> &(y)) where y is defined in terms of x and g(y) is defined in terms of f(x). To solve this new problem, SISP/I sometimes needs a new example (x[1], f(x')) which is used to deduce an example (y'» g(y'))« The interaction is only used in the sense of asking for new examples, when necessary. SISP/I is thus extensible and has the potentiality to use a self constructed knowledge data base.

Some objections can be raised to our interactive method:

- when a function f needs several examples to be infered, the professor sometimes has to give an appropriate sequence of examples.
- we do not exactly know the class of functions which SISP/I is able to infer. However, it seems to be much larger than THESYS one. For instance ((PQRS)-»(PPQPQRPQRS)) is infered by SISP/I using only one example whereas the HALF function ((P Q R S T U) + (P Q R)), which is infered by THESYS, requires two examples by SISP/I. In fact,

we hope that SISP will be able to infer a larger class of linear recursive functions.

II. GENERAL DESCRIPTION OF THE METHOD

1.- L§ngu§ge

SISP/I infers functions defined on character strings "ABCD..." which will be represented by the list (A B C D...).

SISP/1 synthetizes LISP-functions built with the following basic functions, described here by examp1es:

LCAR: (A B C D) -* (A) CDR: (A B C D) -> (B C D)
LRAC: (A B C D) -+ (D) RDC: (ABC D) -> (A B C)
CONC: (A B), (C D) ■> (ABCD)
CONCT: (A B), (C D), (E F) + (A B C D E F)
PREF: (BC), (ABC D) •> (A) L Prefix of (B C) in (ABC D)]
SUFF: (B C), (A B C D) ■+ (D) l Suffix of (B C) in (A B C D)j

and a control structure using COND and NULL.

2*~ Notion_of_tYp_e

A type is a set of lists which can be defined by rules which are summarised as follows [6 1:

a) the set of known inputs "x" and the set of outputs "f(x)" of the function f to be synthetized are types.

b) if X is a type and f a LISP function, then the set of outputs of f restricted to X as input is a type.

c) if Y is a type and g a LISP function then the set X of x such as g(x) C- Y is a type.

3«~ Segment^].i_on_pattern

Let f be a function to be synthetized and (x, f(x)) an example of "input-output" of this function.

SISP/1 uses a general heuristic to create an expression of the function:

a) segmentation of strings x and y = f(x) into three consecutive segments such that:

CONCT (px, c, sx) ■+ X

CONCT (py, c, sy) -► y

where c denotes the larger string common to x and y, px and py denote the prefixs of c in x and y, sx and sy denote the suffixs of c in x and y.

b) building of relations between these segments.

A "Segmentation Pattern" of (x,y), for all x and y, is defined as the network shown in figure I.

We can see on this network:

- seven nodes representing types respectively associated to the strings x,y,c,px,py,sx,sy.
- twelve relations between nodes. Each relation consists of a function and a scheme (I,» I«, •••> I -> J) which indicates the input nodes I , I ,..., I_n in this order and the output node J. This order is represented on the network by a double arrow.

Note that functions FX, FY, GPX, GPY, GSX, GSY are built by SISP/1 using the basic functions LCAR, CDR, LRAC, RDC, and the composition rule. They are choosen of the less possible complexity (the smallest number of basic functions).

In some cases, the segmentation pattern is simpler:

- when one or several strings are empty (NIL), the associated nodes are suppressed from the pattern.
- when two strings are equal, the associated nodes are joined together. For instance, if x and y are the same, the pattern is reduced to one node; if x and y have no common part, the pattern is reduced to only two nodes.

4*"" §Y.£lt}£sis fron}_on£_exam£le

The synthesis consists of three steps:

a) SISP/1 generates a network (called a "Segmentation Structure") by the following process:

(1) Generate the segmentation pattern of (x,y).The generation gives the two sets of pairs: {(px» py), (c,py), (sx, py)} Kpx,sy),(c,sy),(sx,sy)}

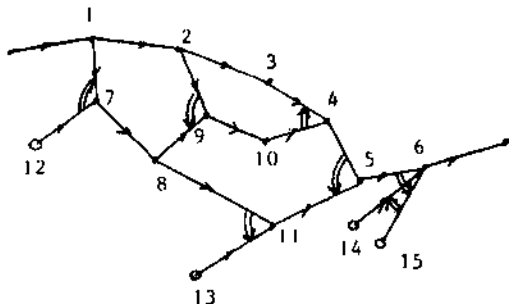
(2) As long as py and sy are not empty, choose one pair in each set by a heuristic way; for each of these pairs, rename it as (x,y) and go to step 1.

b) SISP/1 looks at the segmentation structure for a lattice in which the minimal and final nodes are respectively X and Y (that is x and y types). This lattice is stepwise constructed using Algorithm 1, defined as follows:

Definitions:

- LAT is the constructed part of the lattice at any step (except in the final step, LAT is not a lattice).
- an incomplete node of LAT is a node such that the relation ending at this node (in LAT) owns some entries which are not connected to X. These nodes are called unsatisfied entries.
- BEG (Z) is the set of nodes in LAT which are less than Z and which are not unsatisfied entries.
- P is a "path" from BEG (U) to V, where U and V are nodes of LAT, if P is an oriented path starting from one node belonging to BEG (U) and ending at V. This path may contain incomplete nodes together with their unsatisfied entries*

Example of LAT:



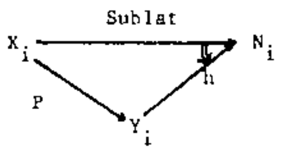
Nodes 6, 7, 11 are incomplete nodes
Nodes 12, 13, 14, 15 are unsatisfied entries
All others nodes are complete nodes.
BEG (9) - {X, 1, 2, 7, 8}

Algorithm 1:

1. LAT «- X
2. Look for a path P between X and Y.
3. Add path P to LAT.
4. Lf there is no incomplete node in LAT then stop
   else select the minimal one and call it N. (It can be demonstrated that Algorithm 1 generates a set of incomplete nodes which is totaly ordered on LAT).

5. Let Y_i be one unsatisfied entry of node N_i.
6. Look for a path P between BEG(N_i) and Y_i. Let X_i be the origin of P on BEG(N_i) and try to detect a recursivity between X_i and Y_i using Algorithm 2.
7. Go to step 3.

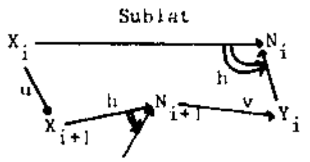
It follows from algorithm 1 that when Algorithm 2 is called, a part of LAT has the following structure:



where sublat is restricted to be a lattice, and Y_i is the previous unsatisfied node.

Algorithm 2:

1. If no path from X_i to N_i (in sublat) matches a subpath of P (in the sense of an identical sequence of operators) then stop algorithm 2, else let $X_{i+1} \to N_{i+1}$ be the subpath of P which has been matched. The above structure is changed to:



2. If the segmentation structure does not contain a lattice starting in X_{i+1}, ending in N_{i+1} and analogous to sublat then stop algorithm 2 else assume the following recursion:

$$X_i \xrightarrow{\varphi} N_i$$

$$\varphi(x) = h\ [\text{sublat } (x),\ v(\varphi(u(x)))]$$

or $$X_i \xrightarrow{\varphi} N_i$$

$$\varphi\ (x)=h[\ v(\varphi(u(x))),\ \text{sublat}(x)\ ]$$

depending on the order of arguments of h.

3. Find a primitive stop condition of the recursive function φ as follows: match the operators of path P from X_{i+1} to N_{i+1} in the segmentation structure, then from X_{i+2} to N_{i+2} and so on, until it fails. Assume it fails from X_j to N_j. Find a path w from X_j to N_j. The primitive stop condition is assumed to be:

   if $x \in X_j$, then w (x)

4. Reduce the primitive stop condition as follows:
   a) remove Sublat from LAT.
   b) if a relation from N_k to N_{k+1} for every k, $i \le k < j$, cannot be found in the segmentation structure, then set $P \leftarrow (X_i \xrightarrow{\varphi} N_i)$ and stop algorithm 2 else let r be the found relation.
   c) find k, the smaller non negative integer such that $w_k = r^k\ (w(x))$ is not a fixed point of equation:

   $$h\ [\text{sublat } (u^k(x)),\ v(r(w_k(x)))]\ = w_k(x)$$

   for every $x \in X_j$.
   d) Set $P \leftarrow (X_i \xrightarrow{\varphi} N_i)$

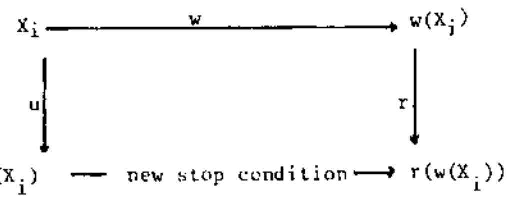
   with $\varphi(x) = \begin{cases} \text{If } x \in u^k\ (X_j) \text{ then } z_k(x) \\ \text{else } h\ [\text{sublat } (x),\ v(\varphi(u(x)))] \end{cases}$

   where z_k is solution of the following equation:

   $$z_k\ (u^k\ (x)) = r^k(w(x)) \text{ for every } x \in X_j$$

Remark : To reduce the primitive stop condition, the following process is iterated:
Suppose the last stop condition is
if $x \in X_j$ then w(x)

Using the functions u and r, it is possible to calculate φ (x), $x \in X_j$, as shown on the following figure:



that is: $\varphi\ (x) = h\ [\text{sublat } (x),\ v(\varphi(u(x)))]$
$= h\ [\text{sublat } (x),\ v(r(w(x)))] = w(x)$

which means, if the correct answer w(x) is obtained that w(x) is a fixed point of the last equation.

It thus follows that there exists a function z such that:

$$z(u(x)) = r(w(x)) \text{ for every } x \in X_j$$

Thus now the stop condition is:

if $x \in u\ (X_j)$ then z(x)

5.- Synthesis from two examples

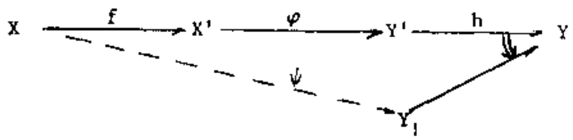
Let us suppose that after a first example, SISP/1 generates a function which fails on a second example. Let the two examples be (x,y) and (x',y').

The principle is always to build a structure from the generation of segmentation-patterns; SISP/1 here generates the segmentation patterns associated with the initial pairs (x,x'),(y,y'),(x,y),(x',y') and goes on in the same way as in the first method.

SISP/1 then tries to find a three parts splitted path from X to Y:
- a path from X to X'.
- the function φ itself from X' to Y'.
- a path from Y' to Y.

Remarks: - using this technique, SISP/1 looks explicitely for a recursive form of the function φ.
- when unsatisfied nodes are remaining in the path, SISP/1 generates sub-problems which are to be solved either by algorithm 1 or again by using one more example when algorithm 1 fails [3 bis].

For instance, let Y_1 be a remaining unsatisfied node in the path:



SISP provides the following expression of φ:

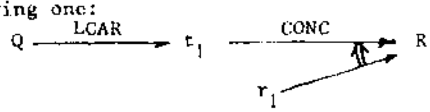
$$\varphi(x) = \begin{cases} \underline{\text{if}}\ x \in X'\ \underline{\text{then}}\ w(x) \\ \underline{\text{else}}\ h\ [\varphi(f(x)),\ \psi(x)] \end{cases}$$

where ψ is a sub-problem to be solved by SISP/I and where w(x) is the function which has been found by algorithm I working on only the example (x',y'). This stop condition can then be reduced as explained in algorithm 2.

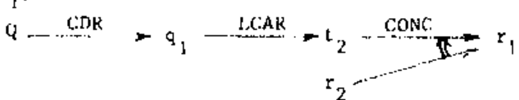
III. PRACTICAL EXAMPLES

1. Let us use our method to find the REVERSE function. The input (A B C D E) is given to SISP/I: it does not know the answer and asks the Professor who returns: (E D C B A). SISP analyses input and output and generates the segmentation structure indicated in figure 3.

SISP looks then for a path from the Question Q containing the list (A B C D E) to the answer R containing (E D C B A) and finds the following one:



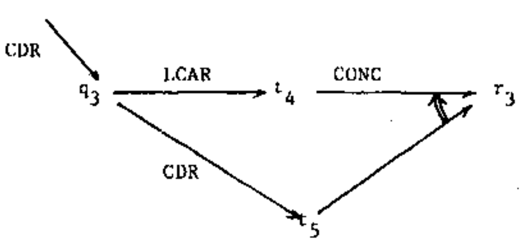
Looking for the unsatisfied entries, SISP finds r_1. It looks again for a path from BEG(R) to r_1, and finds the following one:



SISP now examines both paths. The mapping (LCAR-CONC) of the first one matches into the mapping (CDR-LCAR-CONC) of the second one. This statement is sufficient to infer a recursive expression φ:

$$\varphi(x) = \text{CONC}\ [\varphi(\text{CDR}(x)),\ \text{LCAR}\ (x)]$$

SISP has to still find the stop condition. Matching the three operators CDR, LCAR, CONC with the structure, it remarks that it can apply CDR on type q_3 giving t_5 but cannot apply LCAR on type t_5. SISP/I thus knows that the stop condition is to be found in this part of the structure:



SISP/I tries now to find a new binding of the path from t_5 to the unsatisfied entry of r_3 which happen here to be identical to t_5. It finds the trivial one and generates the primitive stop condition:

$\underline{\text{if}}\ x \in t_5\ \underline{\text{then}}\ x$

SISP now has to reduce the stop condition, using the following mappings:

$$q_i \xrightarrow{\text{CDR}} q_{i+1} \qquad r_i \xrightarrow{\text{RDC}} r_{i+1}$$

Assuming that CDR ((E)) = NIL $\xrightarrow{\varphi}$ RDC((E)) = NIL is the first reduced stop condition, SISP calculates now $\varphi(x)$ for every x belonging to t_5:

$$\varphi(E) = \text{CONC}\ [\varphi(\text{CDR}((E))),\ \text{LCAR}((E))]$$
$$= \text{CONC}\ [\varphi(\text{NIL}),\ (E)]$$

using the new stop condition: $\varphi(E)$ = CONC [NIL, (E)] = (E) which here gives the correct answer.

The process cannot be performed further, because CDR(CDR((E))) does not exist. The function generated by SISP is thus:

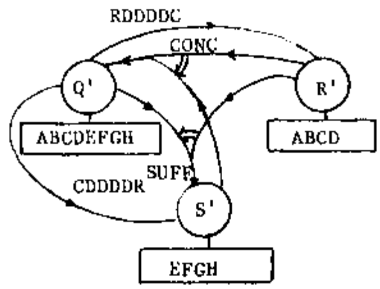
$$\boxed{\varphi(x) = \begin{cases} \underline{\text{if}}\ x = \text{NIL}\ \underline{\text{then}}\ \text{NIL} \\ \underline{\text{else}}\ \text{CONC}\ (\varphi(\text{CDR}(x)),\ \text{LCAR}\ (x)) \end{cases}}$$

that is the usual REVERSE.

2. The second example we display now needs more material than the first one since two couples (input - output) are necessary. Let HALF be the function to synthetize:

- <u>first couple</u>: (A B C D E F G H) → (A B C D)

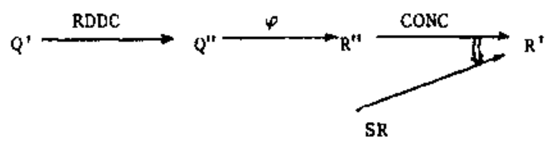
From this first example, SISP/1 constructs the following structure:



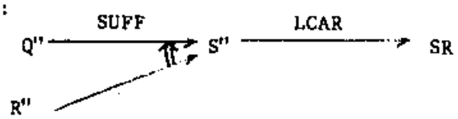
The relation φ:Q' → R' found here is thus $\varphi(x)$ = RDDDDC(x).

- The professor gives now the following input: (A B C D E F). SISP uses φ to answer (A B), which is false. The professor then gives the correct answer: (A B C D E F) → (A B C).

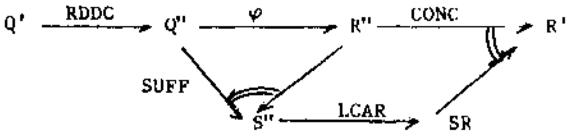
SISP then generates the structure displayed on figure 3. Assuming that R" can be obtained from Q" using the correct function φ to be synthetized, SISP, as explained before, uses the following path from Q' to R':



The problem is now to find a path from BEG (R') to SR. The simplest one which is found here is:



φ is thus represented by the lattice:



it follows that $\varphi(x)$ = CONC [φ(RDDC(x)), LCAR(SUFF [φ(RDDC(x)), RDDC(x)])] with the trivial stop condition:

if x ∈ Q' *then* RDDDC(x)

SISP now has to reduce this stop condition using the following mappings:

Q' —RDDC→ Q'' and R' —RDC→ R''

Assuming that RDDC ((A B C D E F)) = (A B C D)

—φ→ RDC ((A B C D)) = (A B)

SISP computes now φ(x) for every x belonging to Q'':

φ((A B C D E F)) = CONC [φ(RDDC((A B C D E F)), LCAR(SUFF [φ(RDDC((A B C D E F)), RDDC((A B C D E F))])]

= CONC [φ((A B C D)), LCAR(SUFF [φ((A B C D)),(A B C D)])]

= CONC [(A B), LCAR(SUFF [(A B), A B C D)])]

= CONC [(A B), (C)] = (A B C)

which is the correct answer.

The primitive stop condition can thus be reduced to:

if x ∈ RDDC(Q'') *then* RDDC(x)

where RDDC is the solution of the following equation on z

z(RDDC(x)) = RDC (RDDDC(x)) for every x ∈ Q''.

This process is recursively applied and stops when RDDC ((A B)) = NIL. At this step, we obtain the function HALF defined as follows:

φ(x) = { *if* RDDC(x) = NIL *then* RDC(x)
{ *else* CONC [φ(RDDC (x)),LCAR(SUFF [φ(RDDC(x)), RDDC(x)])]

IV. LIMITS AND PROSPECTS OF THE METHOD

1.- Prospects

SISP/1 is already able to synthetize most of the functions given in SUMMERS [5] and HEDRICKS [2] in particular it synthetizes the following ones by using algorithm 1:

(A B C D E) → (E D C B A)
(A B C D E) → (A X B X C X D X E X)
(A B C D E) → (A A B B C C D D E E)
(A B C D E) → (A A B A B C A B C D A B C D E)

By using two or more examples it synthetizes:

(A B C D E F G) → (A A G G B B F F C C E E D D)
(A B C D E F G H) → (A B C D)
(A B C D E) → (E D C B A E D C B E D C E D E)
(A B C D) → (D C B A C B A B A A D C B C B B D C C D)
(A B C D E) → (A B B C C C D D D D E E E E E)
(A B C D E F G H) → (D C B A H G F E)
(A B) → (A A A A A A A A)(cube of the entry length)
(A B C D) → (A A A A A A A A) (half square): such a way is not always easy to use, as we shall see now:

- assume SISP has to synthetize the HALF function using the previous example (A B C D E F G H) → (A B C D). Algorithm 1 fails and the professor gives as second example:

(B C D E F G) → (B C D)

SISP here generates the following HALF function:

φ(x) = { *if* x = NIL *then* NIL
{ *else* CONC [LCAR(x), φ(CDR(RDC(x)))]

which is much simpler than previous HALF function. This simplicity was however found by the professor who gave better examples.

- assume now that SISP has to synthetize a function using the example (A B C D E) → (A B B C C C D D D D E E E E E). Algorithm 1 fails and the professor gives as second example : (A B C D) → (A B B C C C D D D D). SISP generates the following functions:

φ(x) = { *if* x ∈ Q'' *then* R''
{ *else* CONC [φ(RDC(x)), ψ(x)]

where ψ(x) is bound to the following subproblem:

(A B C D E) → (E E E E E)

Algorithm 1 fails then to provide a correct function ψ and the professor now has to give the two particularly well choosen examples:

(B C D E) → (B C C D D D E E E E)
(B C D) → (B C C D D D)

they allow SISP to generate a new appropriate example in order to synthetize a correct ψ:

ψ(x) = { *if* x = NIL *then* NIL
{ *else* CONC [ψ(CDR(x)), LRAC(x)]

the stop condition of φ is then found:

if x = NIL *then* NIL

the generated function φ will thus be given by the linear recursive system Φ

Φ: { φ(x) = *if* x = NIL *then* NIL *else* CONC [φ(RDC(x)), ψ(x)]
{ ψ(x) = *if* x = NIL *then* NIL *else* CONC [ψ(CDR(x)), LRAC(x)]

These two last examples show the main importance of good examples. We hope however that it would be possible to use "bad examples" joined together with a unification process, in order to improve the given "bad examples".

2.- Limits

- with the exception of stop-condition, the functions generated by SISP do not use predicates in their definition . Thus the function:

if "length of x is even" then *reverse* (x)
else x

cannot be synthetized by SISP. This problem is attacked in [6].

- SISP/1 requires a good sequence of example in order to use the second technique. They have to be of decreasing length and consecutive.
- SISP/1 only works on atomic lists.

V. CONCLUSION

In summary, the described method consists of constructing a structure from an adequate set of examples $\{(x_i, f(x_i))\}$ and in extracting from the structure a lattice which represents an expression of the function f.

SISP is a LISP program working on PDP 10 using VLISP 10 [1].

Future developments will tend to make SISP able to:

- define and store self contained problems in its memory,
- recognize that a partial problem has already been encountered and solved,
- improve the professor's bad examples in order to be able to solve partial problems which have never before been encountered,
- synthetize n-any functions (the two presented techniques can easily be generalized).

VI. BIBLIOGRAPHY

[1] GREUSSAY P., at all: "VLISP 10. Reference manuel". Rapport interne Univ. Vincennes.

[2] HEDRICKS C.L.: "Learning Production System from examples", Artificial Intelligence Journal n° 7, January 1976.

[3] JOUANNAUD J.P., GUIHO G., TREUIL J.P., COALLAND P.: "SISP, système interactif de synthèse de programmes à partir d'exemples". Publication de l'Institut de Programmation, Mars 1977.

[3bis] JOUANNAUD J.P., GUIHO G.: "Inference of functions with an interactive system" to be published in Machine Intelligence n° 9 edited by D. MITCHIE.

[4] POHL Ir.: "Bi-Directional and Heuristic Search in Path Problems", Thesis, Computer Science Dept. Stanford University, 1969.

[5] SUMMERS P.D.: "Program Construction from examples". Phd. Thesis, December 1975.

[6] TREUIL J.P., JOUANNAUD J.P., GUIHO G.: "Une méthode d'apprentissage de concepts, Colloque AFCET: Panaroma de la Nouveauté informatique en France, Gif s/Yvette, Novembre 1976.

[7] TREUIL, J.P., JOUANNAUD J.P., GUIHO,G.: "L.Q.A.S., un système question-réponses basé sur l'apprentissage et la synthèse de programmes à partir d'exemples. Publication de l'Institut de Programmation, Mars 1977.

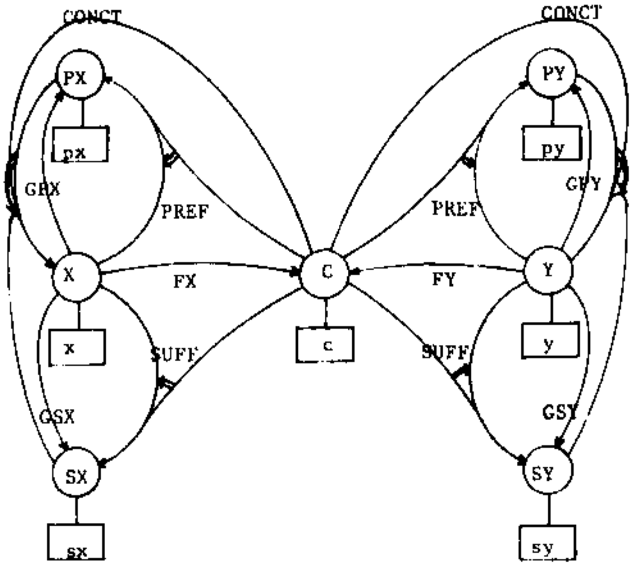


Figure 1: Segmentation pattern associated to (x,y).

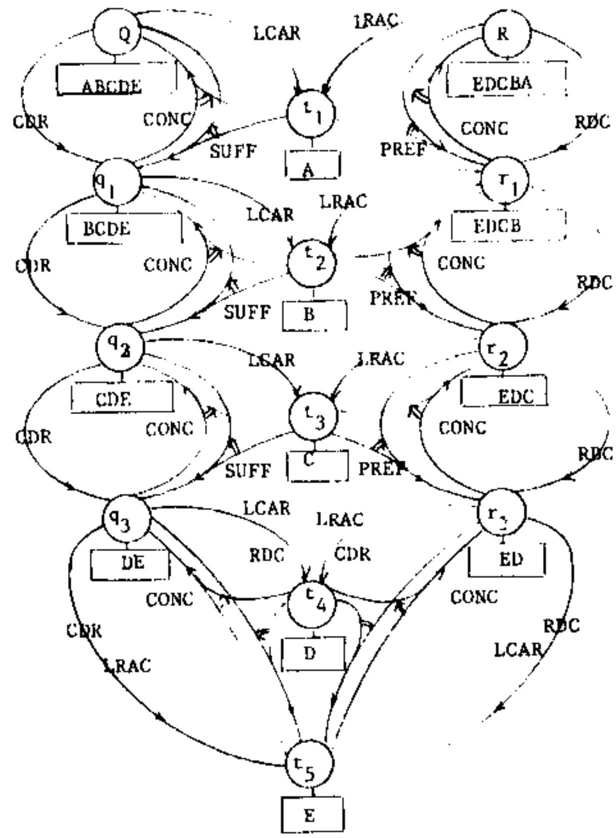


Figure 2: Segmentation structure associated to the REVERSE function.

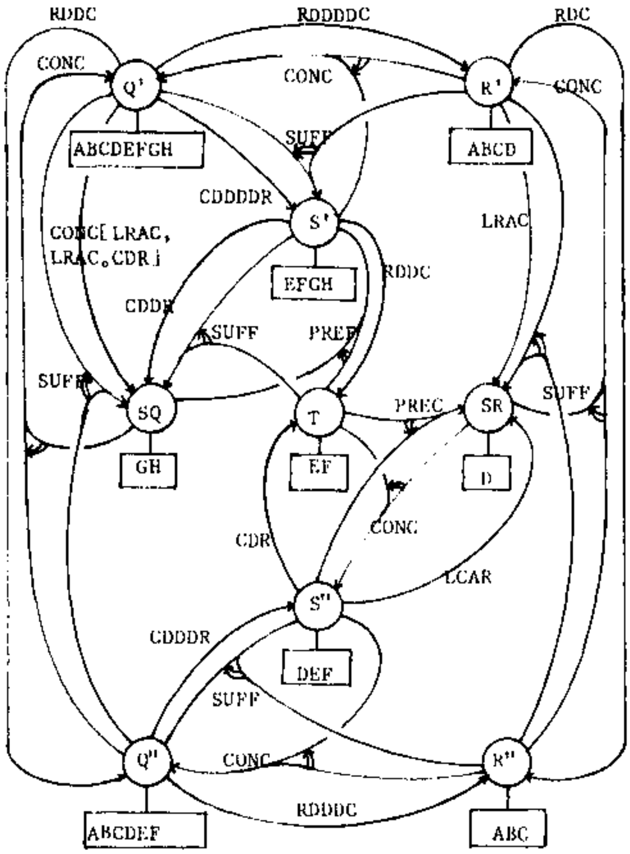


Figure 3: Structure associated to the HALF function.