USING EXAMPLES TO GENERATE INSTANTIATIONS FOR SET VARIABLES

W. W. Bledsoe

July 1982 ATP-67

PRELIMINARY VERSION

Abstract. Examples play a crucial role in automated theorem proving, not only as counterexamples to help prune unproductive subgoals, but they serve to help guide proof discovery. In this paper we show how examples (interpretations) might be used to help determine instantiations of set variables. We also discuss the role of piecewise-linear continuous functions, and give some results of computer runs using these methods.

Table of Contents

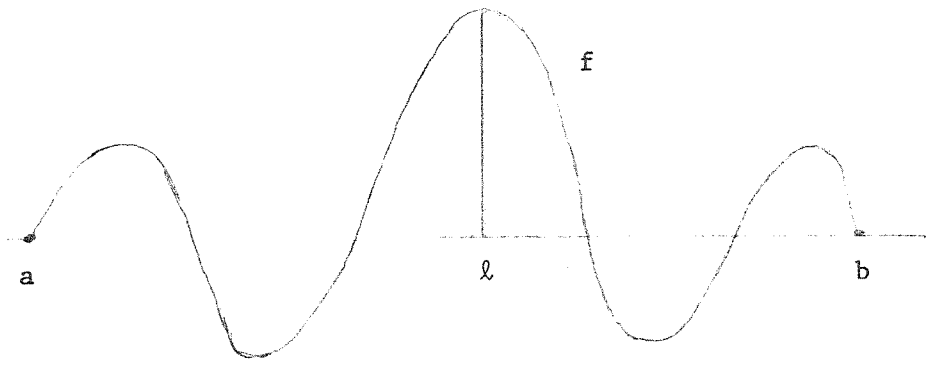
1. Introduction

This is part of an effort to use examples (interpretations) to help in automatic proof discovery [1]. We believe that examples are not only useful as "counterexample sieves", to prune the proof-search tree, but also can be used to help guide the search. One way this is done, is in finding instantiations of variables. See [1, Section 2]. Here we show how examples might be used to help instantiate set variables.

One reason that examples are so valuable in theorem proving is that they allow us to calculate, as opposed to prove. And since calculating is usually easier than proving, this gives an advantage. But of course the results from examples are less general, so the attack has to be balanced - with some calculation and some proving.

Let us start with the following problem:

> Given a continuous function f on $[a,b]$, with maximum value at $x = \ell$.



To find a subset A of $[a,b]$ for which $\ell = \sup A$.

Describe A in general terms, in terms of f, a, b.

e.g.,

$$A = \{x \in [a,b]: \forall y < x \; f(y) \leq f(x)\}$$

We desire to generate such an A automatically. How? Why?

Motivation for this Example.

If we wish to prove the theorem that

AM1 "Each continuous function on a closed interval, attains its maximum on that interval,"

using the least upper bound axiom

LUB "Every non-empty, bounded, set A has a least upper bound, $\sup A$",

then we must have a value of A (i.e., a description of A in terms of f, a, b), before we can utilize LUB in the proof.

Once A has been given, the theorem will have been reduced to a theorem in first order logic about general inequalities.

More generally, when we are trying to prove a theorem of the form

(1) $$\forall f[H(f) \longrightarrow \exists \ell \in [a,b] \; P(f,\ell)]$$

using the LUB axiom to produce a set A for which $\ell = \sup A$, then (1) can be written

(2) $\forall f[H(f) \longrightarrow \exists A \subseteq [a,b]\ P(f, \sup A)]$,

(a higher order theorem)

which is equivalent to

(3) $\forall f[H(f) \longrightarrow \exists Q\, P(f, \sup\{x \in [a,b]\ Q(x)\})]$

E.g.,

AM1 $\forall f[\overbrace{f \text{ is continuous on } [a,b] \wedge a \leq b}^{H(f)}$

$\longrightarrow \exists \ell \in [a,b]\ \ \forall x \in [a,b]\ (f(x) \leq f(\ell))]$

$\underbrace{\qquad\qquad}_{P(f,\ell)}$

e.g.,

IMV $\forall f[\overbrace{f \text{ is continuous on } [a,b] \wedge a \leq b}^{H(f)}$

$\wedge\ f(a) \leq 0 \wedge f(b) \geq 0$

$\longrightarrow \exists \ell \in [a,b]\ \ \underbrace{f(\ell) = 0}_{P(f,\ell)}$

If we further assume that for each f satisfying $H(f)$, there is a <u>largest</u> ℓ in $[a,b]$ for which (1) holds, then (3) is equivalent to

(4) $$\forall f[H(f) \longrightarrow \exists Q(\ell = \sup\{x \in [a,b]: Q(x)\}$$
$$\wedge \forall \ell' \in [a,\ell)\ \exists x \in (\ell',\ell) Q(x)$$
$$\wedge \forall x \in (\ell,b] \sim Q(x)$$
$$\wedge\ P(f,\ell))]$$

So our objective is to prove

(4) $$\forall f[H(f) \longrightarrow \exists Q(\ell = \sup\{x \in [a,b]: Q(x)\}$$
$$\wedge\ \forall \ell' \in [a,\ell)\ \exists x \in (\ell',\ell)\ Q(x)$$
$$\wedge \sim \exists x \in (\ell,b]\ Q(x)$$
$$\wedge\ P(f,\ell))]$$

This has the general form

(5) $\forall f[H(f) \longrightarrow Q: \psi(Q,f)]$.

So for this form of theorems our problem is:

Given H and ψ.

Generate Q for which (5) holds. How? We propose to do this (automatically) by using interpretations (examples) $\hat{f}$ of f (and interpretations of the other function and predicate symbols appearing in H and ψ).

We will first state our problem in a more general setting and then (in Section 4) return to the forms (5) and (4) as special cases.

2. Generating Predicates

Suppose that $\mathcal{F}$ is a list of uninterpreted function symbols and predicate symbols (which does not contain the symbol Q). Let $\mathcal{L} = \{\wedge, \vee, \sim, \rightarrow, \forall, \exists, \leq, <, =\}$, $\mathcal{E} = \{\leq <\}$, and let H and ψ be functions over the symbols of $\mathcal{F} \cup \mathcal{L} \cup \{Q\}$. By an interpretation of $\mathcal{F}$ we mean a list of interpretations of its members. For example, if $\mathcal{F}$ is {a,b, ℓ, f}, then $\hat{\mathcal{F}}$ is $\{\hat{a},\hat{b},\hat{\ell}, \hat{f}\}$, where $\hat{a}$, $\hat{b}$, $\hat{\ell}$, $\hat{f}$, could be $\hat{a} = 0$, $\hat{b} = 1$, $\hat{\ell} = 1/2$, $\hat{f} = \lambda x(2x - 1)$, for example.

Now suppose that we are to prove a theorem of the form

(6) $$\forall \mathcal{F}\,[H(\mathcal{F}) \rightarrow \exists Q\ \psi(\mathcal{F}, Q)].$$

The object then is to find (generate) a Q which satisfies (6).

We propose the following general method for finding Q.

(i) Obtain an interpretation $\hat{\mathcal{F}}$ of $\mathcal{F}$ which satisfied $H(\hat{\mathcal{F}})$.

(ii) Generate a predicate Q, in terms of $\mathcal{F}$, which satisfies

$$\psi(\hat{\mathcal{F}}, Q)$$

(iii) Test Q on other $\hat{\mathcal{F}}$'s.

(iv) Prove (6) using the Q so obtained.

Of course, finding Q is equivalent to proving a theorem in higher order logic, which requires the higher order variable Q to be instantiated. The central idea is that: if a Q can be found which satisfies $\psi(\hat{\mathcal{F}}, Q)$ for a $\hat{\mathcal{F}}$'s (which satisfy $H(\hat{\mathcal{F}})$), then hopefully that Q will also satisfy $\psi(\mathcal{F}, Q)$. We will indeed see that that is the case for some special instances given in Section 4 below.

We will defer discussion of Step (i), the fabrication of $\mathcal{P}$, until Section 3 (a subroutine, INSTANTIATIONS, does this), and concentrate now on (ii), the generation of Q for which $\psi(\hat{\mathcal{P}},Q)$ holds. Step (iii) is rather straightforward, and Step (iv) is not within the scope of this paper*.

So let there be given an interpretation $\hat{\mathcal{P}}$ satisfying $H(\hat{\mathcal{P}})$. We desire an algorithm GENERATE which will generate a Q satisfying $\psi(\hat{\mathcal{P}},Q)$. Of course such a Q may not be unique (even for a fixed $\hat{\mathcal{P}}$).

If the algorithm GENERATE is indeed to generate such a Q, then it must use an association between the members of $\mathcal{P}$ and those of $\hat{\mathcal{P}}$. Because, Q is to be given in terms of the symbols in $\mathcal{P}$, not $\hat{\mathcal{P}}$, and it must satisfy the condition $\psi(\hat{\mathcal{P}},Q)$ where every P in $\mathcal{P}$ has been replaced by an interpretation $\hat{P}$.

It is not obvious how to build such an algorithm GENERATE, but somehow it should key on the structure of ψ. The following, GENERATE, is a first attempt at such an algorithm. It is built on several additonal assumptions, given below, about $\mathcal{P}$ and ψ.

---

*It is often the case, as in the examples of Section 4, that once an instantiation $\bar{Q}$ of Q has been given, the resulting theorem $\forall\mathcal{P}[H(\mathcal{P}) \longrightarrow \psi(\mathcal{P},\bar{Q})]$ is first order. And while it still may not be easy to prove it, nevertheless, lends itself to standard procedures.

We will assume that complete typing* information is available on ψ, Q, and members of $\mathcal{J}$. Thus, for example, Q might be a predicate over the reals, so that for each $x \in \mathcal{R}$, Q(x) is either true of false. (E.g., $Q(x) \equiv f(x) \leq 0$).

For our first version we will assume that Q is a function of one real variable x. (In general we might restrict Q to a function on $\mathcal{R}^n$.)

We will start with a call to

GENERATE(n, $\mathcal{B}$, ψ,'x),

with n = 1, and where $\mathcal{L}$ = 'x $\mathcal{J}$ and 'x is the argument of Q. $\mathcal{B}$ is treated as a set of bindings, tying members of $\mathcal{J}$ to those of $\mathcal{J}$. (I.e., $\mathcal{B}$ is a substitution). $\mathcal{B}$ will be expanded as new variables and their instantiations are added. If A is a formula then A$\mathcal{B}$ is used to denote the result of applying the substitution $\mathcal{B}$ to A.

The integer n is the number of variables used in the description of Q. If n = 1, then Q is expressed only in terms of x, e.g., $0 \leq f(x)$; if n = 2 then Q is expressed in terms of x and y, e.g., $\forall y \in [a,x)\ f(y) \leq f(x)$; etc.

The first call to GENERATE is made with n = 1. If this fails (returns NIL), then n is increased by 1 and a new call made to GENERATE, etc., up to a maximum allowable value for n.

---

*See, for example, Peter B. Andrews, "Resolution in Type Theory", J. Sym. Logic 36 (1971), 414-432, [5].

GENERATE(n, $\mathcal{M}$, ψ, x)

The objective is to find a Q for which $\psi(\mathcal{M},Q)$ $\hat{}$ / $\mathcal{M}$ is true. x is a variable, it starts as 'x, the argument of Q

Form of ψ	ACTION
1. $\psi_1 \wedge \psi_2$	Put Q_1 = GENERATE(n, $\mathcal{M}$, ψ_1, x) Q_2 = " (" " ψ_2, ") Return $(Q_1 \cap Q_2)$ (Each of Q_1 and Q_2 is a set of formulas).

Note: alternatively we might generate Q_1 and verify it it in ψ_2 (or generate Q_2 and verify it in ψ_1).

2. $\psi_1 \vee \psi_2$	Put Q_1 = GENERATE(n, $\mathcal{M}$, ψ_1, x) Q_2 = " (" " ψ_2, ") Return $(Q_1 \cup Q_2)$

3. $\forall x' \in [a,b]^{*} P(x',Q)$

Select randomly† points $x_1,\ldots x_k$ from [a,b], including the endpoints.

Put $\mathcal{B}_i = \mathcal{B} \cup \{(x',x_i)\}$

$Q_i = \text{GENERATE}(N,\mathcal{B}_i,P(x_i,Q),x)$, $i = 1,2,\ldots,k$

Return $Q_1 \cup Q_2 \cup \ldots \cup Q_k$

4. $\exists x' \in [a,b]^{*} P(x',Q)$

Select randomly points $x_1,\ldots,x_k$ from [a,b], including the endpoints.

Put $\mathcal{B}_i = \mathcal{B} \cup (x',x_i)$

$Q_i = \text{GENERATE}(n,\mathcal{B}_i,P(x_i,Q),x)$, $i = 1,2,\ldots,k$

Return $Q_1 \cup Q_2 \cup \ldots \cup Q_k$

---

*Similarly for open and half open intervals (a,b), [a,b), (a,b]. In these cases the open endpoints are not selected for x, but "nearby" points are selected (see Section 3.5).

†The number k used here is a parameter supplied by the user. See Section 3.5 below for an alternate way of selecting the x_i when $\hat{f}$ is a piecewise linear continuous function.

5. ψ

Put $Q' = \text{GENERATE}(N, \mathcal{L}, \psi, x)$

Return $\sim Q'$*

6. $Q(\hat{x})$**

Put $\mathcal{L} = \mathcal{L} \cup \{(x,\hat{x})\}$

x-list = $\{x\}$, $\hat{x}$-list = $\{\hat{x}\}$

If n = 1, return TALLY($\mathcal{L}$, x-list, $\hat{x}$-list, x)

Else return ALL-SOME(n, $\mathcal{L}$, x-list, $\hat{x}$-list, $\hat{x}$)

---

* Since Q' is a list, the list of negations of its members is returned.

** Recall that $\hat{x}$ is a real number in $[\hat{a},\hat{b}]$.

TALLY ($\mathcal{F}$, x-list, $\hat{x}$-list, x')

$\mathcal{F}$ is a set of Predicate and function symbols,

$\hat{\mathcal{F}}$ " " " " interpretations for members of $\mathcal{F}$,

x-list is a set of variables (e.g., {x,y,z})

$\hat{x}$-list is a set of real numbers, instantiations for x-list,

$\mathcal{C}$ is a set of bindings:

$$\mathcal{C} = (\mathcal{F} \times \hat{\mathcal{F}}) \cup (\text{x-list} \times \hat{x}\text{-list})$$

x' is a variable, the last element of x-list.

This routine is supposed to determine "what is true" about $\hat{\mathcal{F}} \cup \hat{x}$-list (for x' only) and record that information in terms of $\mathcal{F} \cup$ x-list.

(NOTE: This is similar to the conjecturing of Lenat [2]. See Section 4 below.)

Let H_1 be the first and second level terms of $\mathcal{F} \cup$ x-list (i.e., constants, variables, and one-level application of function symbols to these (part of the Herbrand Universe)).

Let $\mathcal{A}$ be the atoms associated with $\mathcal{F}$, $\mathcal{C}$, and H_1, but only those containing x'.

Let S be the set of all atoms A of $\mathcal{A}$ for which $A\mathcal{C}$ is true, and for which A is not a tautology.

Return the conjunction of S.

(note: S might be NIL)

$A\mathcal{C}$ is defined to be the result of replacing any member of $\mathcal{P} \cup$ x-list in A by the corresponding value $\hat{\mathcal{P}} \cup \hat{x}$-list.

The following is an example of the use of Tally:

$\mathcal{F} = \{f, a, b, \ell, 0\}$, $\mathcal{R} = \{\leq <\}$,

$\hat{\mathcal{F}} = \{\lambda x\,(2x - 1), 0, 1, 1/2, 0\}$

$Q(\hat{x})$, $\hat{x} = 1/4$

$H = \{a, b, x, f(a), f(b), f(f(a)), \text{etc.}\}$

$\mathcal{A} = f(x) < f(x), f(x) \leq f(x), f(x) < 0,$

$f(x) \leq 0, 0 < f(x), 0 \leq f(x), \text{etc.}\}$

Putting $\hat{\mathcal{F}}$ for $\mathcal{F}$ and simplifying we get

$\mathcal{A}\hat{\mathcal{F}}$ $= \{2x - 1 < 2x < 1, \overbrace{2x - 1 \leq 2x - 1}^{\text{tautology}}, x < 1/2,$

$x \leq 1/2, 1/2 < x, 1/2 \leq x, \text{etc.}\},$

$\mathcal{A}\hat{\mathcal{F}}\hat{x}/x = \{0 < 0, 0 \leq 0, 1/4 < 1/2,$

$1/4 \leq 1/2, 1/2 < 1/4, 1/2 \leq 1/4, \text{etc.}\},$

$S = \{f(x) < 0, f(x) \leq 0, \text{etc.}\}$.

Remark. The list S might be rather large unless additional restrictions are placed on H. So in TALLY, it is convenient (for efficiency purposes) to have further "typing" information which, hopefully can be derived automatically from

the theorem being proved. For example, in proving the theorem:

$$\text{continuous } f \wedge a < b \wedge f(a) \leq 0 \leq f(b) \longrightarrow \exists x\,(f(x) = 0)$$

we note that: a, b, 0, x, f(z), f(b), f(x) all have type Real. But in the context of this theorem this set can be partitioned into two subsets

$\{a, b, x\}$ "x-axis reals"

and

$\{f(a), f(b), f(x), 0\}$ "y-axis reals"

So in TALLY we should not build atoms of the form $x \leq 0$, $a \leq 0$, but only those of the form $f(a) \leq 0$, $f(x) < 0$, etc. Such additional knowledge was used in the above example and in those of Section 4. This concept requires much further study.

ALL-SOME(n, $\mathcal{B}$, x-list, $\hat{x}$-list, $\hat{x}_0$)

This routine (if n = 2) is supposed to introduce a new variable 'y, and give it some values $y_1,\ldots,y_k$ in $[\hat{a}, \hat{x}_0)$, and $y_1',\ldots,y_k'$ in $(\hat{x}_0, \hat{b}]$, and tally "what is true" about each of these y_i's and y_i''s, (in terms of $\mathcal{A}$,x-list), and finally deduct statements of the form $\forall y \in [a,x)\ P(y)$, $\exists y \in (x,b]\ P(y)$, etc., and return these. If n > 2, then yet another variable 'z is introduced (for each y_i, y_i'), n is decreased by 1, etc.).

1. Select randomly $y_1,\ldots,y_k$, from $[\hat{a},\hat{x}_0)$, $y_1',\ldots,y_k'$ from $(\hat{x}_0,\hat{b}]$.*

2. For each i, put $\mathcal{A}_i = \mathcal{A} \cup \{('y,y_i)\}$, $\mathcal{B}_i' = \mathcal{B} \cup \{('y, y_i')\}$

(NOTE: 'y is a new variable symbol).

x_i-list = x-list $\cup \{y\}$, $\hat{x}_i$-list = $\hat{x}$-list $\cup \{y_i\}$

$\hat{x}_i'$-list = $\hat{x}$-list $\cup \{y_i\}$

3. If n = 2, Put Q_i = TALLY($\mathcal{B}_i$,x_i-list, $\hat{x}_i$-list, 'y)

Q_i' = TALLY($\mathcal{B}_i'$,x_i-list, $\hat{x}_i'$-list, 'y)

Else put Q_i = ALL-SOME(n - 1, $\mathcal{A}_i$, x_i-list, $\hat{x}_i$-list, y_i)

Q_i' = ALL-SOME(n - 1, $\mathcal{A}_i'$, x_i-list, $\hat{x}_i'$-list, y_i')

4. Put QQ = COMBINE-ALL-S (Q_i, i = 1, k, $[\hat{a}, \hat{x}_0)$)

QQ'= COMBINE-ALL-S (Q_i', i = 1, k, $(\hat{x}_0,\hat{b}]$)

5. Return QQ $\cup$ QQ'

---

* We show here only the case n = 2 for $n \geq 3$, the procedure is appropriately generalized to handle intervals of the form $(\hat{x}_0,y_i)$, $(y_i,\hat{x}_0)$, etc., as well as $[\hat{a},\hat{x}_0)$ and $(\hat{x}_0,\hat{b}]$.

Notice that we have arbitrarily restricted the new variable 'y (see 1. above) to the intervals $[\hat{a}, \hat{x}_0)$ and $(\hat{x}_0, \hat{b}]$. This might be too restrictive. Perhaps we should have also considered y's in the intervals, $(\hat{x}_j, \hat{x}_0)$ for each $\hat{x}_j$ in $\hat{x}$-list, or other possibilities.

In the case n = 3, the predicate Q will be described in terms of three variables x, y, z. In this case, after the y_i have been selected as indicated in Step 1, another call to ALL-SOME will cause (by Step 1) points $z_1,\ldots,z_k$ to be selected from each of the intervals $[\hat{a},\hat{x}_0)$, $(\hat{x}_0,\hat{b}]$, $(\hat{x}_0,y_i)$, (y_i,x_0), i = 1,k. Similarly when $n \geq 4$.

COMBINE-ALL-S($\mathcal{A}$, y, β)

This is used by ALL-SOME.

Here $\mathcal{A}$ is a set

$$\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p\}$$

where each $\mathcal{A}_i$ is a result from TALLY. The $\mathcal{A}_i$ are treated as sets rather than conjunctions,

$$\mathcal{A}_i = \{\ell_{i1}, \ldots, \ell_{in_i}\}$$

and β is an interval $[\hat{a}_0, \hat{x}_0)$, $(\hat{x}_0, \hat{b}]$, $(\hat{x}_0, \hat{y}_i)$, etc.

Let $\sigma\mathcal{A} = \bigcup_i \mathcal{A}_i$,

and for each $\mathcal{L} \in \sigma\mathcal{A}$, let

$$q(\mathcal{L}) = \begin{cases} \forall y \in \beta \ \mathcal{L} & \text{if } \mathcal{L} \in \mathcal{A}_i \text{ for all } i,\ i = 1, p \\ 0 & \text{" } \mathcal{L} \notin \mathcal{A}_i \text{ "} \\ \exists y \in \beta \ \mathcal{L} & \text{otherwise} \end{cases}$$

and return the conjunction of the members of the set

$$\{q(\mathcal{L}): \mathcal{L} \in \sigma\mathcal{A} \wedge q(\mathcal{L}) \neq 0\}$$

(This might be NIL).

The Algorithm COMBINE-ALL-S as defined here produces the simplest answer from a set $\mathcal{A}$, as depicted by the following examples.

EX 1.

$\mathcal{A}$ = {{f(x) < f(y)} {f(y) < f(x)}}

y = 'y

β = (x,b]

COMBINE-ALL-S returns

{$\exists$ y ε(x,b]f(x) < fy, $\exists$ y ε(x,b]f(y) < f(x)}

EX 2.

$\mathcal{A}$ = {{A B} {A C} {A D}}

where A, B, C, D are some formulas

y = 'z

β = (y b]

COMBINE-ALL-S returns

{$\forall$ z ε (y,b]A, $\exists$ z ε (y,b] B, $\exists$ z ε (y,b]C, $\exists$ z ε (y,b]D}

EX 3.

$\mathcal{A}$ = {{A B C} {B C D}}

y = 'z, β, A, B, C, D unspecified

COMBINE-ALL-S returns

{ ∃ z ε β A, ∃ z ε β B, ∃ z ε β C, ∃ z ε β D}.

But in EX 3 (and similarly in EX 2) it could have returned the correct but more complicated answer

$$\{\exists\ z\ \varepsilon\ \beta(A \wedge B \wedge C),\ \exists\ z\ \varepsilon\ \beta(B \wedge C \wedge D)\}$$

Our implementation of COMBINE-ALL-S allows both options; the simpler version produces more tractable answers for instantiation of set varialbes (see IMV and AM1 in Section 4), but the more complicated version was needed for EX 5 of Section 4.

3. Obtaining Interpretations $\hat{\mathcal{I}}$.

The fabrication of an interpretation $\hat{\mathcal{I}}$ of $\mathcal{I}$ which satisfies $H(\hat{\mathcal{I}})$ for a given H, is itself a challenging problem. In general it cannot be handled automatically.

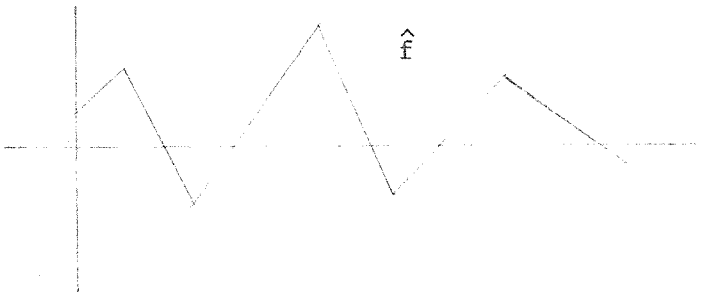
It is not the main purpose of this paper to discuss the problem of fabricating these $\hat{\mathcal{I}}$'s but in using them. However, we do have some suggestions.

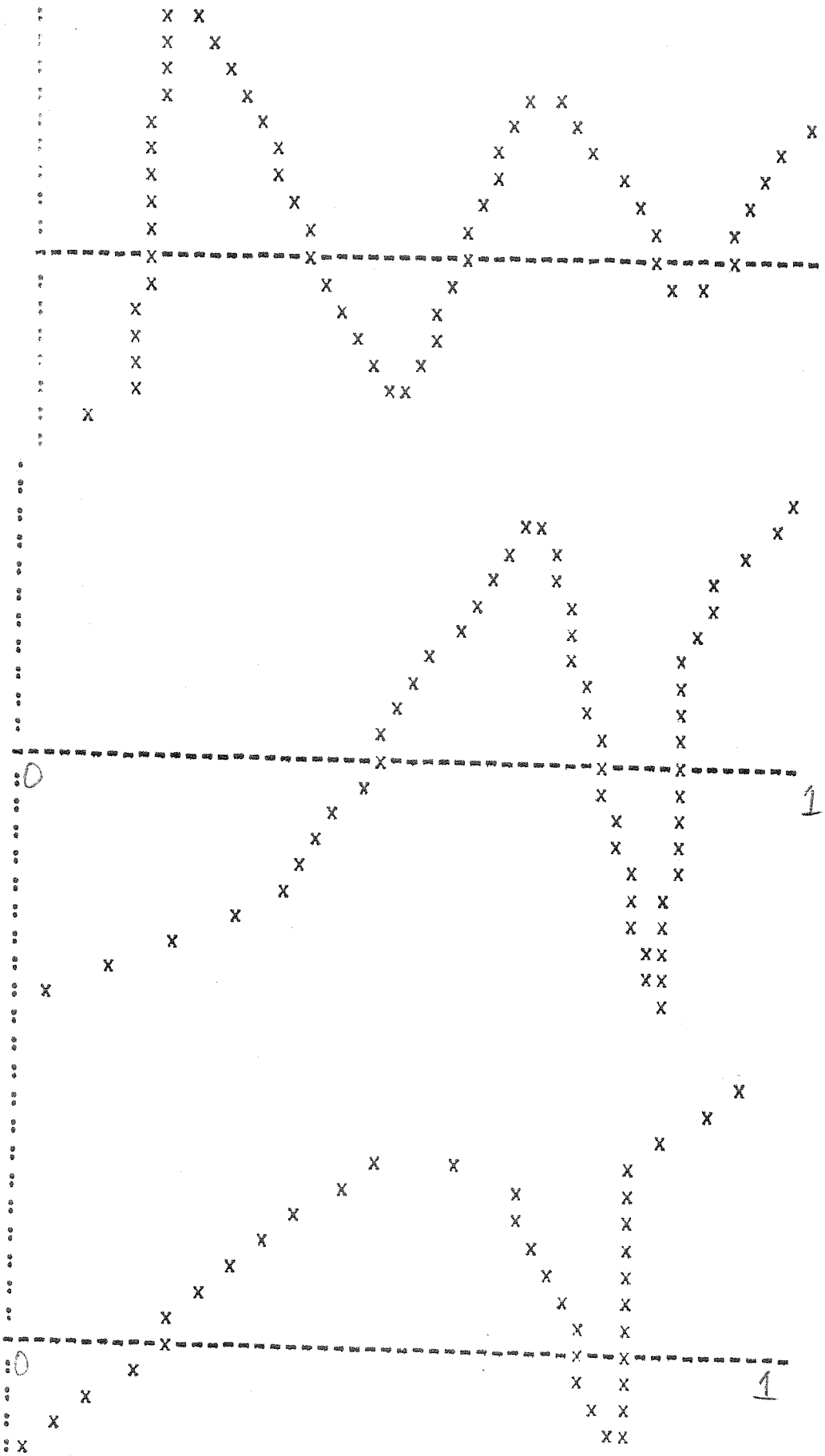
3.1. Human supplied interpretations.

One possibility is that the user supplies the $\hat{\mathcal{I}}$'s, either at run time, or in a convenient knowledge base which the program can effectively access. See, for example, [3]. Such a collection of examples could accumulate over a period of time and be used for a number of applications. In the examples of Section 4, we show the obtaining of the needed interpretations by a call to a subroutine INTERPRETATIONS.

3.2. Piecewise-linear continuous functions.

For the special case when the examples needed are interpretations of continuous functions on a closed interval of the real line, we might employ piecewise-linear continuous functions (pclf's) in a number of applications.





Computer Generated Piecewise-linear Continuous Functions
(Satisfying the hypothesis $f(0) < 0 \wedge f(1) > 0$)

These have the advantage that are easy to generate and use. Such an $\hat{f}$ with n corners, can be generated for the interval [a,b], by generating n random numbers $x_1,x_2,\ldots,x_n$ in the interval [a,b], (sorted), and n + 2 random numbers $y_a,y_1,\ldots,y_n,y_b$, and putting

$$\hat{f} = \{(a,y_a)(x_1,y_1) \ \ldots \ (x_n,y_n)(b,y_b)\} \ .$$

Such plcf's were used in the examples of Section 4.

If one needs a plcf $\hat{f}$ which satisfies an additional constraint $H(\hat{f})$, then one can either: generate $\hat{f}$'s and test them until one satisfying $H(\hat{f})$ is found; or try to build into the generating routine the ability to restrict such $\hat{f}$'s. Again this second approach appears to be difficult in general, though we were able to realize it for special cases such as:

$$H(f): \ f(a) \leq 0 \ \wedge \ f(b) \geq 0$$

or

$$H(f): \ \forall x \in [a,b] \ (f(a) \leq f(x)).$$

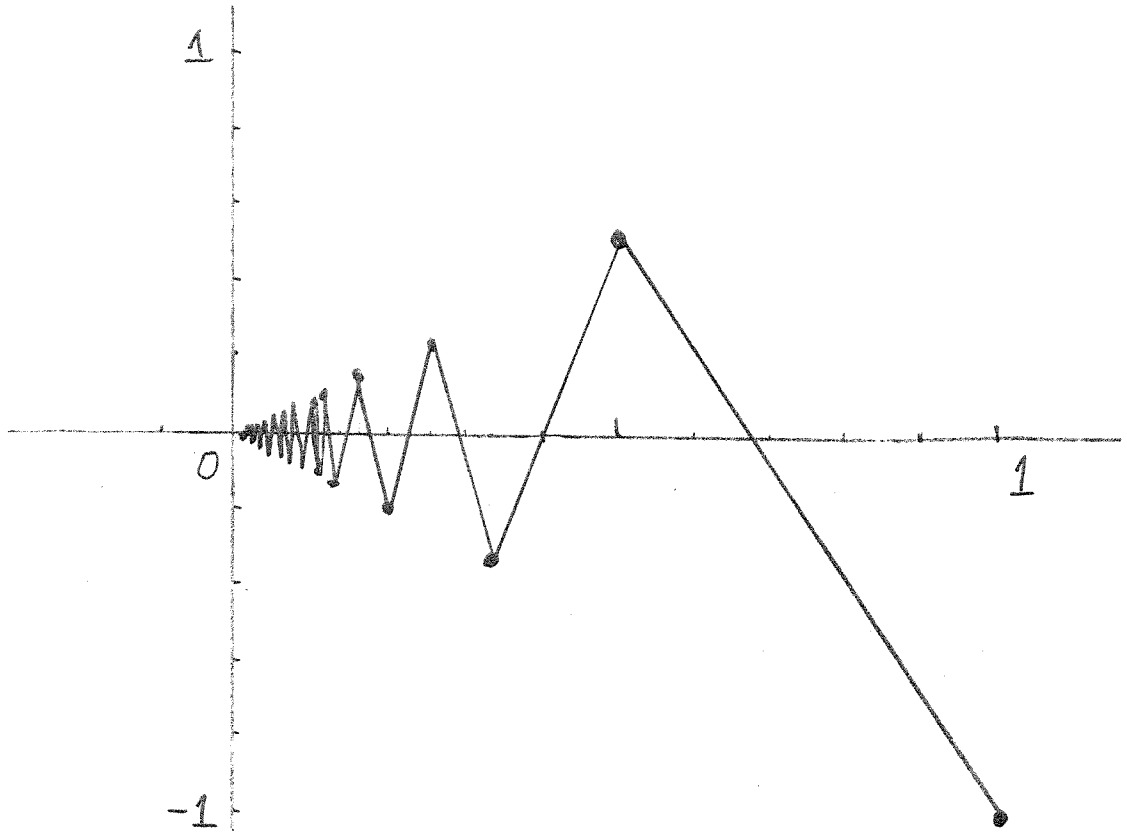
It might also be useful to build up a special set of routines for handling pfcf's, for evaluating $\hat{f}$ at specific x's (numbers), and for computing their maximum, minimum, zeros, etc.

3.3 Infinite Number of Corners

If a plcf with an infinite number of corners is needed, then one might use a formula for computing the x_n and y_n instead of the list of number pairs. For example,

$$(1/n,\ (-1)^n/n),\ n = 1,2,\ldots$$

represents the plcf

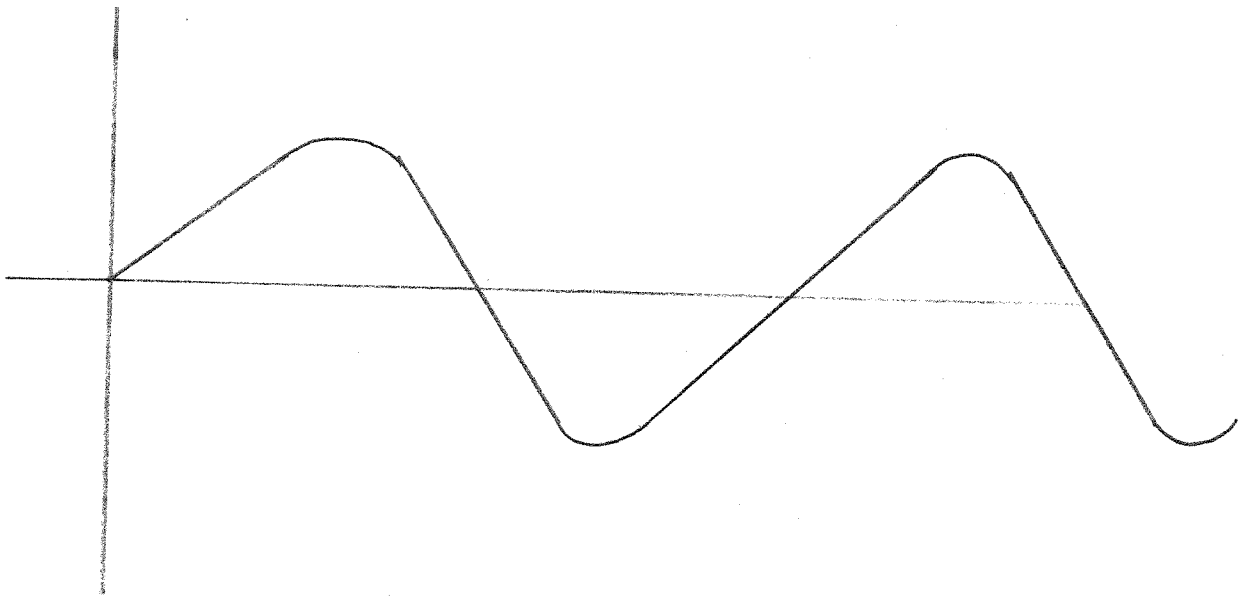


In working with such a plcf, the computer would deal with this formula instead of with a list of pairs of numbers.

In addition to such a description, one might want to add a finite number of fixed points (e.g., (0,0)).

3.4 Piecewise-linear functions with "knees".

If the interpretation $\hat{f}$ of f is required to be <u>differentiable</u> as well as <u>continuous</u>, we might want to place quadratic "knees" on our plcf's,



or "cubic" knees if the derivative of f is required to be continuous, etc. Again one would need to develop a set of routines for evaluating $\hat{f}$ at particular x's, and for finding maxima, minima, zero's etc., for such $\hat{f}$'s.

We do <u>not</u> recommend that polynomials be used as $\hat{f}$'s because in order to obtain an example with a few undulations, it is necessary to use a polynomial of order four or higher, and these are very difficult to compute.

3.5 Using the "Corners" of $\hat{f}$.

plcf's have another advantage besides being easy to compute. For example, if we are trying to tally the formula

$$\hat{f}(x) < f(.499)$$

for values of x within the interval I_1,