USING EXAMPLES TO GENERATE INSTANTIATIONS OF SET VARIABLES

W. W. Bledsoe
Department of Mathematics and Computer Science
The University of Texas at Austin
Austin, Texas 78712, USA

ABSTRACT

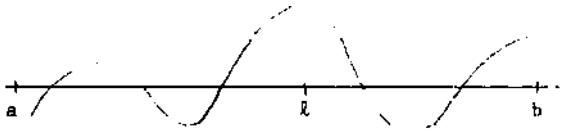
Examples play a crucial role in automated theorem proving, not only as counterexamples to help prune unproductive subgoals, but also to help guide proof discovery. In this paper we show how examples (interpretations) might be used to help determine Instantiations of set variables. We also discuss the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

I 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 = l.



To find a subset A of [a,b] for which ℓ = sup A. Describe A in general terms, in terms of f, a, b. E.g.,

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

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

using the LUB axiom to produce a set A for which l - sup A, then (1) can be written

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

(a higher order theorem)

which is equivalent to

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

E.g.,

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

$\rightarrow \exists \ell \in [a,b] \ \underbrace{\forall x \in [a,b] \ (f(x) \le (\ell))}_{P(f,\ell)}]$.

e.g.,

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

$\wedge f(a) \le 0 \wedge f(b) \ge 0$

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

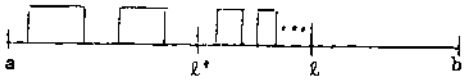
If we further assume that for each f satisfying H(f), there is a largest l in [a,b] for which (1) holds, then (3) is equivalent to

(4) $\forall f[H(f) \rightarrow \exists Q(\ell = \sup\{x \in [a,b]: Q(x)\}$

$\wedge \forall \ell' \in [a,\ell) \exists x \in (\ell',\ell) Q(x)$

$\wedge \forall x \in (\ell,b] \sim Q(x)$

$\wedge P(f,\ell))]$



So our objective is to prove

(4) $\forall f[H(f) \rightarrow \exists Q(\ell = \sup\{x \in [a,b]: Q(x)\}$

$\wedge \forall \ell' \in [a,\ell) \exists x \in (\ell',\ell) Q(x)$

$\wedge \sim \exists x \in (\ell,b] Q(x)$

$\wedge P(f,\ell))]$

This has the general form

(5) $\forall f(H(f) \rightarrow \exists Q: \psi(Q,f)]$.

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) 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.

II GENERATING PREDICATES

Suppose that P is a list of uninterpreted function symbols and predicate symbols (which does not contain the symbol Q). Let $L = \{\wedge, \vee, \sim, \rightarrow, \forall, \exists, \le, <, =\}$, $LE = \{\le, <\}$, and let H and ψ be functions over the symbols of $P \cup L \cup \{Q\}$. By an interpretation of P we mean a list of interpretations of its members. For example, if P is $\{a,b,\ell,f\}$, then $\hat{P}$ 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) $\forall P[H(P) \rightarrow \exists Q \psi(P,Q)]$.

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 $\hat{P}$ of P which satisfied $H(\hat{P})$

(ii) Generate a predicate Q, in terms of P, which satisfies

$$\psi(\hat{P},Q)$$

(iii) Test Q on other $\hat{P}$'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 this: if a Q can be found which satisfies $\psi(P,Q)$ for $\hat{P}$'s (which satisfy $H(\hat{P})$), then hopefully that Q will also satisfy $\psi(P,Q)$. 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 P, until Section 3 (a subroutine, INSTANTIATIONS, does this), and concentrate now on (ii), the generation of Q for which $\psi(\hat{P},Q)$ holds. Step (iii) is rather straightforward, and Step (iv) is not within the scope of this paper*.

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

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

It is not obvious how to build such an algorithm GENERATE, but somehow it should key on the structure of iii. The following, GENERATE, is a first attempt at such an algorithm. It is built on several additional assumptions, given below,

It is often the case, as in the examples of Section 4, that once an instantiation Q of Q has been given, the resulting theorem VP[H(P)-ij)(P,Q)] is first order. And while it still may not be easy to prove it, nevertheless, lends itself to standard procedures

about P and ψ. (The symbol "Q": in the following algorithms will always represent the predicate we are looking for - i.e., trying to generate.)

We will assume that complete typing* information is available on ψ, Q, and members of P. Thus, for example, Q might be a predicate over the reals, so that each $x \in R$, $Q(x)$ is either true or 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 R^n.)

We will start with a call to

GENERATE(n, $\mathcal{B}$, ψ, 'x),

with n=1, and where $\mathcal{B} = P \times \hat{P}$ and 'x is the argument of Q. B is treated as a set of bindings, tying members of P to those of P. (I.e., 8 is a substitution). B will be expanded as new variables and their instantiations are added. If A is a formula then AB is used to denote the result of applying the substitution 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. (Note: The parameter n counts both bound and free variables.)

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.

(Before presenting the algorithms GENERATE, TALLY and ALL-SOME, let us try to help the reader by pointing out that we are looking for statements that can be deduced from a given example, function f, that might hold for the whole interval [a,b] or for a part of it; that the method works by looking at the critical points of the example function; that GENERATE, TALLY, and ALL-SOME, return families of conditions that are true on the subintervals; that these families of conditions are combined in appropriate ways to give candidates for the Q that we are looking for; and finally that this combining action is governed by the structure of the formula containing Q.)

GENERATE(n, $\mathcal{B}$, ψ, x)

The objective is to find a Q for which $\psi(P,Q)$ *PIP* 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 Q1 = GENERATE(n, $\mathcal{B}$, ψ_1, x)

Q_2 = GENERATE(n, $\mathcal{B}$, ψ_2, x)

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 in ψ_2 (or generate Q_2 and verify it in ψ_1).

2. $\psi_1 \vee \psi_2$

Put Q_1 = GENERATE(n, $\mathcal{B}$, ψ_1, x)

Q_2 = GENERATE(n, $\mathcal{B}$, ψ_2, x)

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 $\mathcal{B}_1 = \mathcal{B} \cup \{(x', x_1)\}$

Q_i = GENERATE(n, $\mathcal{B}$, $P(x_i,Q)$, x),

i = 1, 2, ..., k

Return $Q_1 \cap Q_2 \cap \ldots \cap Q_k$

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

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

Put $\mathcal{B}_1 = \mathcal{B} \cup (x', x_1)$

Q_i = GENERATE(n, $\mathcal{B}$, $P(x_i,Q)$, x),

i = 1, 2, ..., k

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

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

*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 x1 when f is a piecewise linear continuous function.

5. $\sim\psi$

Put Q' = GENERATE (n, $\mathcal{B}$, ψ, x)

Return ~Q'*

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

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

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

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

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

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

P is a set of predicate and function symbols,
$\hat{P}$ is a set of interpretations for members of P,
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,
$\mathcal{B}$ is a set of bindings: $\mathcal{B} = (P \times \hat{P}) \cup$ (x-list x $\hat{x}$-list),
x' is a variable, the last element of x-list.

This routine is supposed to determine "what is true" about $\hat{P} \cup \hat{x}$-list (for x' only) and record that information in terms of $P \cup$ 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 $P \cup$ x-list (i.e., constants, variables, and one-level application of function symbols to these (part of the Herbrand Universe)). (In later versions we might try to use second-level applications, etc.)

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

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

Return S. (Treated as a conjunction.) (Note: S might be NIL).

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

The following is an example of the use of TALLY:

P = {f,a,b,0}, $\mathcal{L}$ = {≤ <}
$\hat{P}$ = {λx(2x-1),0,1,0}
x-list = {x} (i.e., x' = x).
$\hat{x}$-list = {1/4} (i.e., $\hat{x}$ = 1/4).

$\mathcal{B}$ = {λx(2x-1)/f, 0/a, 1/b, 1/4 / x}
H_1 = {a, b, 0, x, f(a), f(b), f(0), f(x)}.
A = {f(x) < f(x), f(x) ≤ f(x), f(x) < 0, f(x) ≤ 0, 0 < f(x), 0 ≤ f(x)}. (See Remark below).

Putting $\hat{P}$ for P and simplifying we get

$A\hat{P}/P$ = {2x-1 < 2x-1, 2x-1 ≤ 2x-1, 2x-1 < 0, 2x-1 ≤ 0, 0 < 2x-1, 0 ≤ 2x-1}, (tautologies)
$A\mathcal{B}$ = {-1/2 < -1/2, -1/2 ≤ -1/2, -1/2 < 0, -1/2 ≤ 0, 0 < -1/2, 0 ≤ -1/2},
S = {f(x) < 0, f(x) ≤ 0}.

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:

continuous f ∧ a < b ∧ f(a) ≤ 0 ≤ f(b) → ∃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"

{f(z), f(b), f(x), 0} "y-axis reals"

So in TALLY we should not build atoms of the form x ≤ 0, a ≤ 0, but only those of the form f(a) ≤ 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 will not be used unless n ≥ 0. If n = 2 it will 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 P, x-list), and finally deduce 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 $(\hat{x}_0,\hat{b}]$*.

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

**Recall that $\hat{x}$ is a real number in $[\hat{a},\hat{b}]$. By the time that step 6 is applied, all variables will have values.

*We show here only the case n = 2. For n ≥ 3, the procedure is appropriately generalized to handle intervals of the form $(\hat{x}_0,y_1)$, $(y_1,\hat{x}_0)$, etc., as well as $[\hat{a},\hat{x}_0)$ and $(\hat{x}_0,\hat{b}]$. Also See comment below.

2. For each i, put

$$B_i = B \cup \{(y, y_i)\},\ B'_i = B \cup \{(y, y'_i)\}$$

(NOTE: y is a new variable symbol).

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

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

3. If n = 2, put

$$Q_i = \text{TALLY}(B_i,\ x_i\text{-list},\ \hat{x}_i\text{-list},\ y)$$

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

Else put

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

$$Q'_i = \text{ALL-SOME}(n-1,\ B'_i,\ x_i\text{-list},\ \hat{x}'_i\text{-list},\ y'_i)$$

4. Put

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

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

5. Return QQ ∪ QQ'

Comment. 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 ≥ 4.

COMBINE-ALL-S(S, y, B)

This is used by ALL-SOME.

Here S is a set

$$S = \{S_1, S_2, \ldots, S_p\}$$

where each S_i is a result from TALLY. The S_i are treated as sets rather than conjunctions,

$$S_i = \{\delta_{i1}, \ldots, \delta_{in_i}\}$$

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

Let $\sigma S = \cup_i S_i$,

and for each $\delta \in \sigma S$, let

$$q(\ ) = \begin{cases} \forall y \in B\ \delta & \text{if } \delta \in S_i \text{ for all } i,\ i = 1, p \\ \exists y \in B\ \delta & \text{otherwise} \end{cases}$$

and return the conjunction of the members of the set

$$\{q(\delta):\ \delta \in \sigma S\}.$$

(This might be NIL).

The Algorithm COMBINE-ALL-S as defined here produces the simplest answer from a set S, as depicted by the following examples.

EX 1. S = {{f(x) < f(y)} {f(y) < f(x)}}

y = 'y

B = (x,b]

COMBINE-ALL-S returns

{ ∃y ∈ (x,b]f(x) < fy, ∃y ∈ (x,b]f(y) < f(x)}

EX 2. S = {{A B} {A C} {A D}}

where A, B, C, D are some formulas

y = 'z

B = (y b]

COMBINE-ALL-S returns

{ ∀z ∈ (y,b]A, ∃z ∈ (y,b]B, ∃z ∈ (y,b]C, ∃z ∈ (y,b]D}

EX 3. S = {{A B C} {B C D}}

y = 'z, B, A, B, C, D unspecified

COMBINE-ALL-S returns

{ ∃z ∈ B A, ∃z ∈ B B, ∃z ∈ B C, ∃z ∈ B D}.

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

{ ∃z ∈ B(A ∧ B ∧ C), ∃z ∈ B(B ∧ C ∧ D)}

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

III Obtaining Interpretations P

The fabrication of an interpretation P of P which satisfies H(P) for a given H, is itself a challenging problem. In general it cannot be handled automatically.

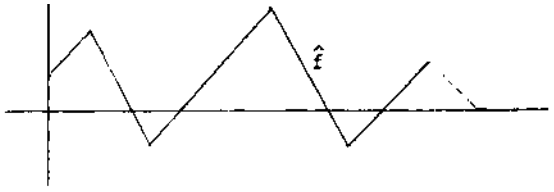
It is not the main purpose of this paper to discuss the problem of fabricating these P's but in using them. However, we do have some suggestions.

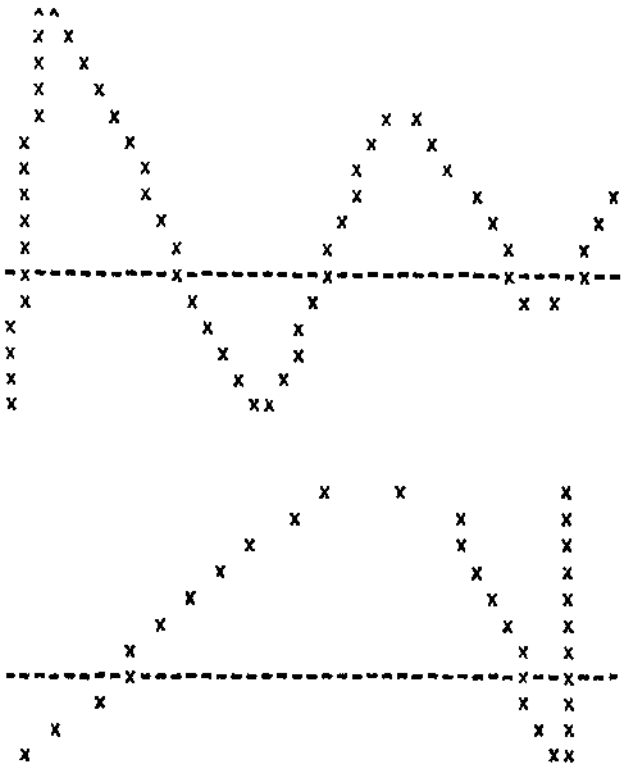
A. Human Supplied Interpretations

One possibility is that the user supplies the P'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.

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

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

or

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

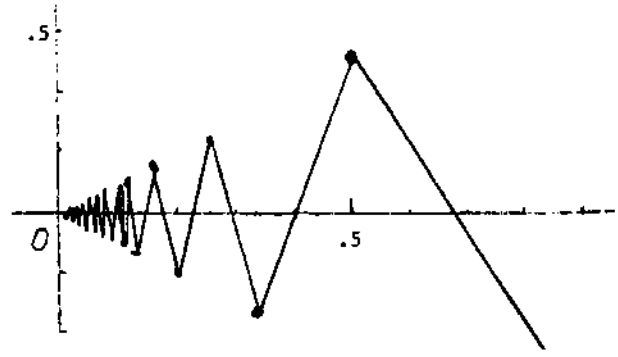
It might also be useful to build up a special set of routines for handling pfcf's, for evaluating f at specific x's (numbers), and for computing their maximum, minimum, zeros, etc.

C. 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 and y instead of the list of number pairs. For example,

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

represents the plcf

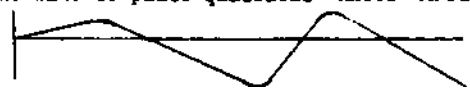


In working with such a plcf, the computer would deal with this formula instead of with a list of pairs of numbers.

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

D. Piecewise-linear Functions With "knees"

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

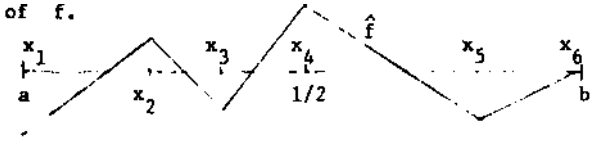


or "cubic" knees if the derivative of f is required to be continuous, etc. Again one would need to develop a set of routines for evaluating f at particular x's, and for finding maxima, minima, zero's , etc., for such f's.

We do not recommend that polynomials be used as 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.

E. Using the "Corners" of f

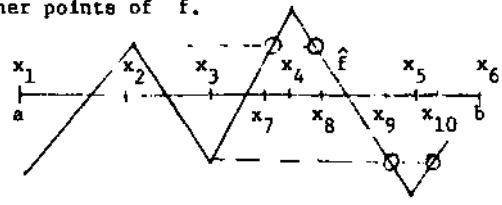
plcf's have another advantage besides being easy to compute. For example, if we are trying to tally the formula (f(x) < f(.499)) for values of x within the interval I1, , the program could just as well use the corner points of f, that is only x's for which (x,y) z f. Thus in the algorithm GENERATE, Steps 3 and 4, where random points, x1, ...Xn, are selected, we might instead have used for the x1's, the x's corresponding to the corners of f.



Corner points of $\hat{f}$

I.e., we need only check $\hat{f}(x) \le f(.499)$ for x equal to the values $x_1, x_2, \ldots, x_6$.

However, if we are using two variables x and y in the description of Q (see algorithm ALL-SOME, Section 3), and are trying to tally the formula (f(y) < f(x)) for x in interval I_1 and y in Interval I_2, it is also necessary* to consider additional x's and y's corresponding** to the corner points of $\hat{f}$.



Corresponding corner points

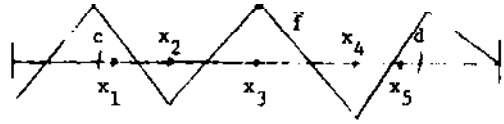
Thus x is allowed to take the values x1,x2,..., x10» which lie within I1 and once x is given such a value, x, y is also allowed to take these values x1,...,xn (within I2) as well as other values "close to" X (see Section 3.6) and points corresponding to them. Of course, this lacks generality but can be shown to be adequate in certain theorems about inequalities. For instance, it was successfully used in the examples of Section 4.

*Necessary because these give all the points a which simple inequalities might change truth values.

**We say a point x' corresponds to a point x" (with respect to $\hat{f}$) if $\hat{f}(x') = \hat{f}(x'')$.

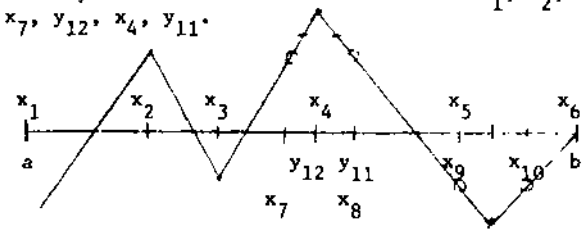
F. Choosing Points From Open Intervals

In algorithms GENERATE-1 and ALL-SOME we often are required to select points x1....x_n (or y1's or y!'s) from open and half open intervals as well as closed intervals, so we cannot always automatically include the endpoints. In these cases we have used a parameter e (small) to pick points "close" to the endpoints. For example, from the interval (a,b] we would select (among others) the point a + e. The parameter z is allowed to diminish in value by multiplying it by 1/10 each time it is used. For an open interval (c,d) we have



In this example the points x_2, x_3, x_4, are selected at the corners of $\hat{f}$, while $x_1 = c + \varepsilon$, $x_5 = d - \varepsilon$, for the open interval (c,d).

Using the example of the previous section (3.5), if I_1 is (1/2,b], and I_2 is [a,x), then x would be allowed to take the values $x_4 + \varepsilon$, x_8, x_9, x_5, x_{10}, x_6. And once x is given the value x_8, then y is allowed to take the values x_1, x_2, x_3, x_7, y_{12}, x_4, y_{11}.



IV Experimental Results .

We consider two theorems IMV and AM1, and show (partially) how GENERATE works on them. See [9] for further details and three other examples. (These examples were all done by a computer program; however, the program is not considered optimal or final at this time.)

IMV (The intermediate value theorem): If f is a continuous function on the non-empty, closed, interval [a,b], f(a) ≤ 0, f(b) ≥ 0, then for some x in [a,b], f(x) = 0,

AM1 Each continuous function f on a closed interval, attains in maximum (minimum) on that interval,

using the least upper bound axiom

LUB Every non-empty, bounded from above set A has a least upper bound, sup A.

In symbols we have

IMV LUB ∧ f is continuous on [a,b] ∧ a ≤ b ∧f(a) ≤ 0, ∧ 0 ≤ f(b)

$\rightarrow \exists \ell \in [a,b]\ (f(\ell) \le 0 \wedge 0 \le f(\ell))$*,

AM1 $\quad LUB \wedge f$ is continuous on $[a,b] \wedge a \le b$
$\rightarrow \exists \ell \in [a,b]\ \forall x \in [a,b](f(x) \le f(\ell))$,

where,

LUB $\quad \forall A \subseteq R(A \neq 0 \wedge \exists \lambda \forall x \in A(x \le \lambda)$
$\rightarrow \exists \ell(\forall x \in A(x \le \ell) \wedge \forall y[\forall z \in A(z \le y)$
$\rightarrow \ell < y]))$.

Our objective in each of IMV and AM1, is to instantiate the set variable A of the hypothesis LUB, and thereby reduce both IMV and AM1 to theorems in first order logic.

Notice that they both have the form (1), p. 4, where for IMV, H(f) is

$$LUB \wedge \text{continuous } f[a,b] \wedge f(a) \le 0 \wedge 0 \le f(b)$$

and for AM1, H(f) is

$$LUB \wedge \text{continuous } f[a,b],$$

and following the steps described on pages 4-6 we obtain formula (4), p. 6, with

$$P(f,\ell) \equiv (f(\ell) \le 0 \wedge 0 \le f(\ell))$$

for IMV, and

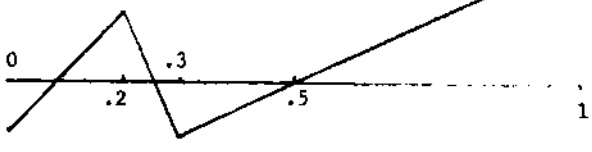
$$P(f,\ell) \equiv \forall x \in [a,b](f(x) \le f(\ell))$$

for AM1.

Example 1 (IMV)

$\forall f[f$ is continuous on $[a,b] \wedge a \le b$
$\wedge f(a) \le 0 \wedge 0 \le f(b)$
$\rightarrow \exists Q(\ell = \sup\{x \in [a,b]:\ Q(x)\}$
(7) $\quad \wedge \forall \ell' \in [a,\ell)\ \exists x \in (\ell',\ell)\ Q(x)$
$\wedge \sim \exists x(\ell,b]\ Q(x) \wedge f(\ell) = 0)]$

a call to INTERPRETATIONS yields $\hat{a} = 0$, $\hat{b} = 1$, $\hat{f} = \{(0 - .2)(.2.2)(.3-.4)(1\ 1)\}$.



A call to CALCULATE-L yields l - 1/2, (In the spirit of the "calculate vs. prove" remarks In Section 1, we of course call on the program to calculate a value of l satisfying the theorem (for a particular f)).

*We have chosen to represent f(x) - 0 as f(x) <; 0 AO < f(x), which is required by our present program. It is not at all clear whether we can easily change the program to handle this example otherwise.

For these values of a, b, f and ℓ, the formula (7) reduces to

$$(7') \quad \overbrace{\exists Q[\underbrace{\forall \ell' \in [0,1/2)\ \exists x \in (\ell',1/2)\ Q(x)}_{\psi_1}}^{\psi} \wedge \underbrace{\sim \exists x \in (1/2,1]\ Q(x)}_{\psi_2}]$$

A call is made to GENERATE-1(ψ). (We suppress the arguments n, β, and 'x) which (by Rule 1 of GENERATE) recalls itself on ψ_1 and ψ_2. GENERATE

$$\underbrace{(\forall \ell' \in [0,1/2)\ \exists x \in (\ell',1/2)\ Q(x))}_{\psi_1}$$

By Rule 3, the points $\ell'_1,\ldots,\ell'_4$ are selected as $\ell'_1 = 0$, $\ell'_2 = .2$, $\ell'_3 = .3$, $\ell'_4 = .499$*, and calls are made to

(8) GENERATE($\exists x \in (0,1/2)\ Q(x)$)
(9) GENERATE($\exists x \in (.2,1/2)\ Q(x)$)
(10) GENERATE($\exists x \in (.3,1/2)\ Q(x)$)
(11) GENERATE($\exists x \in (.499,1/2)\ Q(x)$)

and each of these, by Rule 4, select points $x_1,\ldots,x_n$ from $(\ell'_1,1/2)$ and calls GENERATE($Q(x_1)$) which in turn calls TALLY.

For example, when ℓ' takes the various values shown below, the corresponding values of $x_1,\ldots,x_n$ are chosen and calls to GENERATE($Q(x_1)$ and TALLY yield the results shown.

ℓ'	x_i's	Results of Calls to TALLY
0	.001,.2,.3,.4999	f(x)<0, 0<f(x), f(x)<0, f(x)<0
.2	.2001,.3,.4999	0<f(x), f(x)<0, f(x)<0
.3	.3001,.4999	f(x)<0, f(x)<0
.499	.4991,.4999	f(x)<0, f(x)<0

Then from Rule 4 of GENERATE, we obtain the union of these various subgroups for (8), (9), (10), and (11). I.e., {f(x) < 0, 0 < f(x)} is returned from (8); {f(x) < 0, 0 < f(x)} is returned from (9); {f(x) < 0} is returned from (10); {f(x) < 0) is returned from (11).

These are intersected (by Rule 3 of GENERATE) to obtain {f(x) < 0}.

Recapitulating from (7'): a call was made to GENERATE(ψ_1) which yielded {f(x) < 0}.

Next a call is made to GENERATE(ψ_2) = GENERATE(~ $\exists x \in (1/2,1]\ Q(x)$). By Rule 5, it calls first GENERATE($\exists x \in (1/2,1]\ Q(x)$), which yields {0 < f(x)} and this is negated to obtain {f(x) ≤ 0}. Then by Rule 1, {f(x) < 0} and {f(x) ≤ 0} are "intersected" to obtain {f(x) < 0} as the final answer. (In intersecting these, {f(x) ◁ 0} is treated as {f(x) < 0, f(x) = 0}).

*We are using the "corners" method described in Section 3.5.

This answer, $\{f(x) \leq 0\}$, is checked against other values of $\hat{f}$, and, finally the set A in Theorem IMV is given the value

(14) $\{x: \ a \leq x \leq b \wedge f(x) < 0\}$

which is a correct instantiation.

Incidently, when A is instantiated with this value (14), the theorem IMV is reduced to the first order theorem:

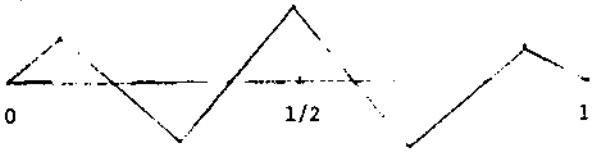
$$\forall x(a \leq x \leq b \wedge f(x) < 0 \rightarrow x \leq \ell)$$
$$\wedge \forall y(\forall z(a \leq z \leq b \wedge f(z) < 0 \rightarrow z \leq y) \rightarrow \ell \leq y)$$
$$\wedge f \text{ is continuous on } [a,b] \wedge f(a) \leq 0 \wedge 0 \leq f(b)$$
$$\rightarrow f(\ell) \leq 0 \wedge 0 \leq f(\ell).$$

Of course we would need to add the definition of continuity and, unless one uses a general inequality prover like [4], also add the axioms for the densely ordered reals.

Example 2. (AM1).

$$\forall f[f \text{ is continuous on } [a,b] \wedge a < b$$
$$\rightarrow \exists Q(\ell = \sup\{x \in [a,b]: \ Q(x)\}$$
$$\wedge \forall \ell' \in [a,\ell) \ \exists x \in (\ell',\ell) \ Q(x)$$
$$\wedge \sim \exists x \in (\ell,b] \ Q(x)$$
$$\wedge \forall y \in [a,b] \ (f(y) \leq f(\ell)))]$$

A call to INTERPRETATIONS yields $\hat{a} = 0$, $\hat{b} = 1$, $\hat{f} = \{(0\ 0)(.1\ .2)(.3\ -.2)(.5\ .5)(.7\ -.3)(.9\ .3)(1\ 0)\}$



a call to CALCULATE-L yields $\hat{\ell} = 1/2$.

As in Example 1 we obtain (exactly) the formula (7'), and calls are then made successively to GENERATE(ψ), GENERATE(ψ_1), GENERATE(ψ_2), which return the lists, $\{\forall y \in [a,x)(f(y) < f(x))$, $\exists y \in (x,b](f(x) < f(y))$, $\exists y \in (x,b](f(y) < f(x))\}$, and $\{\forall y \in [a,x)(f(y) \leq f(x))\}$, which are intersected to obtain $\{\forall y \in [a,x)(f(y) < f(x))\}$ which is returned by the call GENERATE(ψ).

Let us now examine in more detail the call, GENERATE(ψ_1), i.e.,

(15) GENERATE($\forall \ell \in [a,\ell) \ \exists x \in (\ell',\ell) \ Q(x)$).

Rules 3 and 4 are used to select $(\ell'_1,\ldots,\ell'_4)$ equal to (0, .1, .3, .499), and for each ℓ', $x_1,\ldots,x_n$ as shown below. We show here some of the corresponding output from GENERATE. (See [9] for further details.)

In the following table

ALL_1 is $\forall y \in [a,x) \ f(y) < f(x)$
ALL_2 is $\forall y \in [a,x) \ f(x) < f(y)$
ALL_3 is $\forall y \in (x,b] \ f(y) < f(x)$
ALL_4 is $\forall y \in (x,b] \ f(x) < f(y)$
$SOME_1$ is $\exists y \in [a,x) \ f(y) < f(x)$, etc.

If one of these is primed, then $\leq$ is used instead of $<$.

ℓ'	x_i	Result From GENERATE($Q(x_i)$)	Result From GENERATE($\exists x (\ell',\ell) \ Q(x)$)
0	.001	$\{ALL_1 SOME_3 SOME_4\}$	
	.1	$\{ALL_1 SOME_3 SOME_4\}$	
	.3	$\{ALL_2 SOME_3 SOME_4\}$	$\{ALL_1 ALL_2 SOME_3 SOME_4\}$
	.4999	$\{ALL_1 SOME_3 SOME_4\}$	
.1			$\{ALL_1 ALL_2 SOME_1 SOME_2 SOME_3 SOME_4\}$
.3			$\{ALL_1 \quad SOME_1 SOME_2 SOME_3 SOME_4\}$
.499			$\{ALL_1 \quad SOME_3 SOME_4\}$

These are now intersected to yield

(16) $\{ALL_1 SOME_3 SOME_4\}$

from (15), GENERATE(ψ_1).

The call GENERATE(ψ_2),

(17) GENERATE($\sim \exists x \in (1/2,1) \ Q(x)$),

causes (by Rule 5) a call first made to GENERATE-($\exists x \in (1/2,1) \ Q(x)$), whose result is negated and returned for (17).

Rule 3 is used to select (.5001 .7 .9 1.) for (x_1 x_2 x_3 x_4), and GENERATE)$Q(x_i)$) is called for each to obtain the results shown below.

x_i	Result from GENERATE($Q(x_i)$)
.5001	$\{SOME_1 \ SOME_2 \ ALL_3\}$
.7	$\{ALL_2 \ ALL_4\}$
.9	$\{SOME_1 \ SOME_2 \ ALL_3\}$
1.0	$\{SOME_2 \ SOME_1\}$

The union of these $\{SOME_1\ SOME_2\ ALL_2\ ALL_3\ ALL_4\}$ is then returned, and this is negated to yield

$$\{ALL'_2\ ALL'_1\ SOME'_1\ SOME'_4\ SOME'_3\}$$

from call (17). Finally, (16) and (18) are intersected to yield $\{ALL_1\} = \{\forall y \in [a,x)\ f(y) < f(x)\}$ from the call GENERATE(ψ).

This is checked against other values of $\hat{f}$, and finally, the set A in Theorem AM1 is given the value

$$\{x:\ a \le x \le b\ \wedge \forall y \in [a,x)\ f(y) < f(x)\}.$$

See [9] for further details and three other examples.

V Comments

A. Higher Order Logic

Since instantiating set variables is a part of higher order logic, one could also use procedures like those of Andrews [5], Huet [6], Darlington [7], or possibly Bledsoe [8]. In general we would expect these to be less efficient than the technique discussed here, but further experience is needed.

B. Conjecturing

The central component of this work is the routine GENERATE which attempts to general (describe) a predicate Q(x) satisfying a particular form iKQ(x)h This is much in the spirit of Lenat's work [2], where various conjectures are derived from examples.

In Lenat's work as well as ours, there is given a set of examples and the program is asked to determine "what is true" about them. There is a difference however: whereas Lenat asks all that is true (about his examples), we ask what is specifically true about certain objects in P, such as f.

Such conjecturing seems to play an important role in all of human endeavor, and we would expect a prominant place for it in future automatic reasoning systems.

C. Calculate vs. Prove

We cannot over emphasize the importance of being able to calculate properties about a particular $\hat{f}$ rather than prove the same properties about the uninstantiated variable f.

For example, for the continuous function, $\hat{f} = \lambda x\,(4x - 4x^2)$, it is rather easy to automatically calculate that $\hat{x} = 1/2$ is the maximum of $\hat{f}$ on the interval $[a,b] = [0,1]$. However, it is indeed difficult to prove automatically that any continuous function of $[a,b]$, with $a \le b$, attains its maximum on that interval.

D. Other Example Theorems

We hope to extend these results to other example theorems such as: Heine-Borel Theorem, Nestled Interval Theorem, Baire Category Theorem, Balzano-Weierstrass Theorem, etc.

In many of these one will work with families of sets (intervals) instead of functions. There is a natural analogy between piecewise-wise linear continuous functions (plcf's) and finite families of intervals. Accordingly we would expect instantiations which consist of a finite family of intervals, to suffice for many applications. But infinite families will also be needed. See [9].

REFERENCES

[1] Ballantyne, A. Michael, and Bledsoe, W. W. On Generating and Using Examples in Proof Discovery. Machine Intelligence 10. Ellis Harwood Limited, Chichester, 1982, pp. 3-39.

[2] Davis, R., and Lenat, D. Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill, 1982.

[3] Rissland, E., and Soloway, E. Generating Examples in LIPS: Data and Programs. COINS Technical Report 80-07, Amherst: Department of Computer and Information Science: University of Massachusetts at Amherst, 1980.

[4] Bledsoe, W. W., and Hines, Larry$_D$ Variable Elimination and Chaining in a Resolution-Based Prover for Inequalities„ Fifth Conference on Automated Deduction, Les Arcs, France, July 12, 1980, Springer Lecture Notes in Computer Science,

[5] Andrews, P. Bo Resolution in Type Theory. Jour, of Symbolic Logic, 36, 1971, pp. 414-432.

[6] Huet, G. P. Constrained Resolution: A Complete Method for Higher Order Logic. Ph.Do Thesis, Jennings Computer Center Report 1117, Cleveland, Case Western Reserve University, 1972.

[7] Darlington, J. L. Deduction Plan Formation in Higher Order Logic, Machine Intelligence 7, 1972, pp. 129-137. (eds. Meltzer, B. and Michie, D0), Edinburgh University Press.

Darlington, J. L. Talk at Oberwolfach Conference on Automatic Theorem Proving, 1976, Oberwolfach, Germany.

[8] Bledsoe, W. W. A Maximal Method for Set Variables in Automatic Theorem Proving, Machine Intelligence 9, Ellis Harwood Ltd., Chichester, 1979, pp. 53-100.

[9] Bledsoe, W. W. Using Examples to Generate Instantiations for Set Variables, The University of Texas Math. Dept. Memo, ATP-67, July 1982.