TEST SETS GENERATION FROM ALGEBRAIC SPECIFICATIONS USING LOGIC PROGRAMMING

L. Bougé

N. Choquet

L. Fribourg

M-C. Gaudel

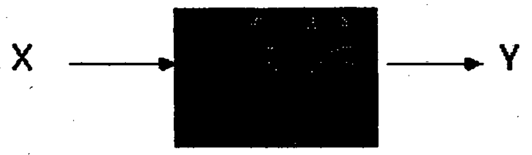
Laboratoires de Marcoussis

Centre de Recherches de la CGE

funded by CIT-Alcatel

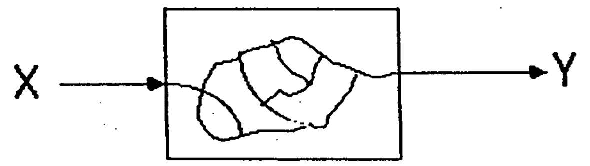
THE PROBLEM

FUNCTIONAL
OR
BLACK-BOX } TESTING



** TEST of ‹ X, Y › AGAINST SOME SPECIFICATION*

PROGRAM
WHITE-BOX } TESTING



**Implementation dependent*
**Test of each possible path*
** Search for specific kinds of errors*

NON-REGRESSION TESTING

CHECK THAT RELEASE$_{n+1}$

"INCLUDES" RELEASE$_n$

implementation independent

expensive

-SPECIFIC TEST DATA SETS

-SYSTEMATIC CONSTRUCTION AND MANAGEMENT

AIMS OF THE STUDY

*TO GIVE RIGOUROUS BASIS TO THE TESTING PROCESS:

TO STUDY THE COMPLEMENTARITY BETWEEN TESTING AND PROVING;

TO MAKE $\boxed{\textit{EXPLICIT}}$ ALL THE ASSUMPTIONS WHICH ARE GENERALLY $\boxed{\textit{IMPLICIT}}$ WHEN SOMEONE SAYS:

"TEST T IS SUCCESSFUL ⇒ PROGRAM P IS CORRECT"

*TO PROVIDE METHODS FOR TEST SETS CONSTRUCTION FROM FORMAL SPECIFICATIONS

PLAN

I)BASIC DEFINITIONS ON TESTING

II) APPLICATION TO ALGEBRAIC SPECIFICATIONS

III) WHY LOGIC PROGRAMMING IS AN APPROPRIATE TOOL

THE PROBLEM

Does a Program P satisfy a Specification S ?

i.e.

Does a Σ-algebra X satisfy a Σ-axiom A ?

BASIC IDEA

IF A IS AN EQUATION:

lhs(x) = rhs(x)

- *instantiate x by some constant Σ-term;*
- *compute both sides in X;*
- *decide if the results are equal;*
- *start again.*

TESTING PROCESS DIAGRAM

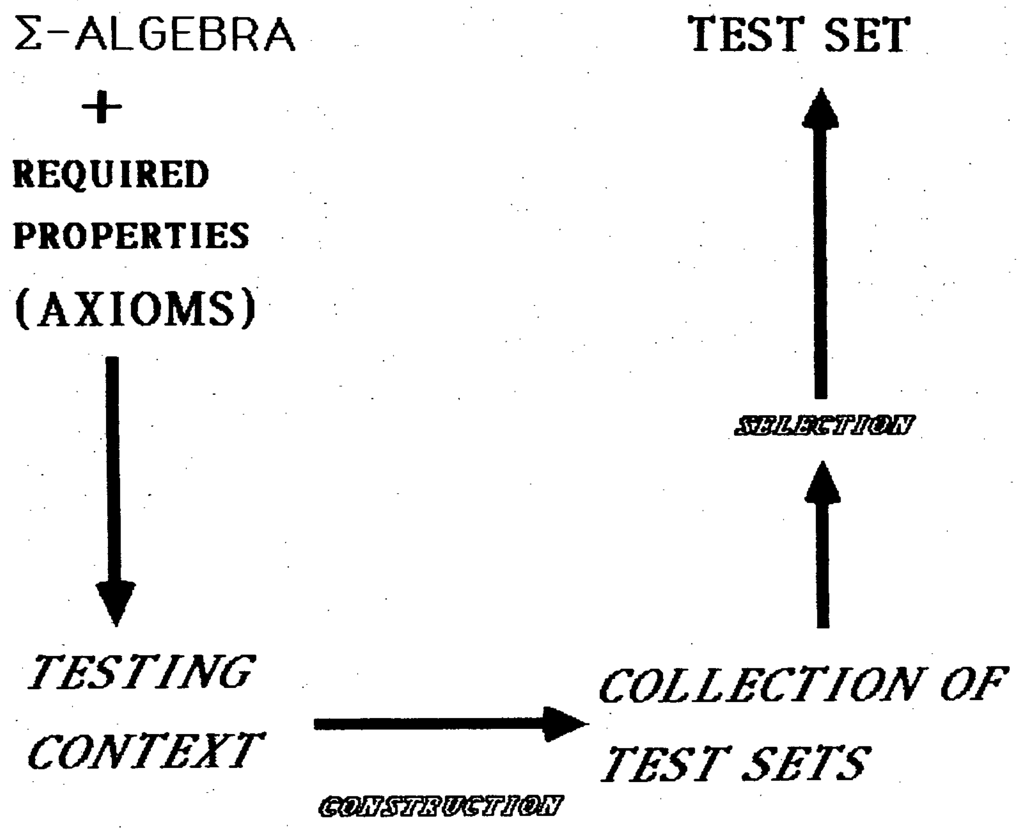
Σ-ALGEBRA

+

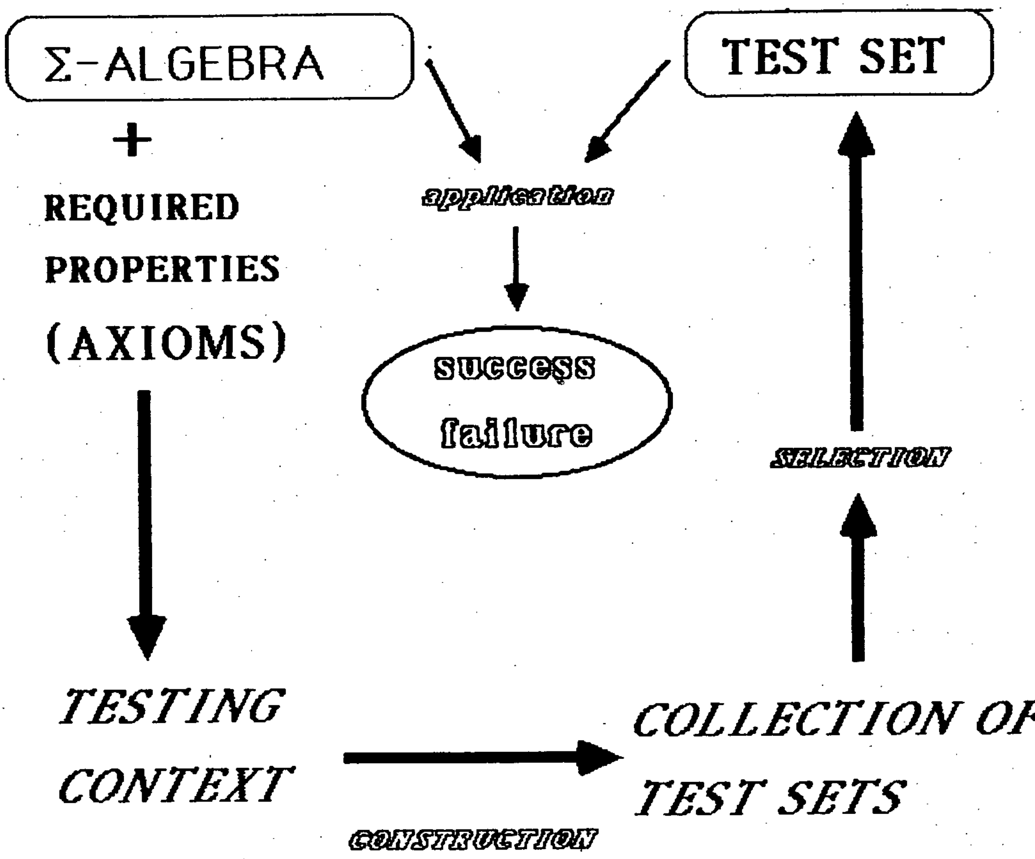
REQUIRED PROPERTIES

(AXIOMS)

TESTING PROCESS DIAGRAM



TESTING PROCESS DIAGRAM



TESTING CONTEXT

A	*Property to be tested (formula)*
(S)	*Class of Σ-algebras (P belongs to (S))*
H	*Testing Hypotheses*

COLLECTION OF TEST DATA SETS

$$(T_n)_{n \in N}$$

$T_p = (a_i \; : i=1, \dots, f(p))$

where a_i are closed instantiations of A

A NAIVE EXAMPLE
QUEUE OF INTEGERS

Σ = { emptyq, append, remove, first}

P = a piece of software implementing functions of these names

A = (q ≠emptyq => first(append(q,x)=first(q))

TESTING CONTEXT: < A, (S), H>

H : Properties of Integers

(S) : Class of all possible pieces of software implementing functions named emptyq append, remove, first

EXAMPLE OF A COLLECTION OF TEST DATA SETS

T_p = { first(append(tq.tx))=first(tq) with $|tq|<p$, $|tq|_{append}<|tq|_{remove}$, $tx<p$ }

SUMMARY

DOES THE Σ-ALGEBRA P SATISFY THE EQUATION lhs(x) = rhs(x) ?

BUILD $(T_n)_{n \; N}$

SELECT $T_p = \{ lhs(t_i)=rhs(t_i) \; / \; i=1,\dots,f(p)\}$

FOR EACH t_i COMPUTE $lhs(t_i)$ and $rhs(t_i)$

DECIDE IF THE RESULTS ARE EQUAL

QUESTION: IS THE CONCLUSION SENSIBLE?

OR

WHAT ARE THE REQUIREMENTS ON (T_n) TO GET A MEANINGFUL CONCLUSION?

RELIABILITY

A collection of test sets is said to be reliable if a test set of higher index is "better" than a test set of lower index whatever potential Σ-algebra is considered.

$$\forall n \in N, (H \cup T_{n+1}) \vdash T_n$$

Property slightly weaker than Goodenough and Gerhart's one:
=> **asymptotic reliability.**

VALIDITY

Any incorrect behavior will be revealed by some test data in some T_n, i.e.

$$(H \cup (\cup_n T_n)) \vdash A$$

If testing is successful using all test sets T_n then the algebra fulfills the required properties. A collection which satisfies this property is said to be

=> **asymptotically valid.**

LACK OF BIAS.

Any correct algebra should pass any test set T_n.
Converse of validity:

$$\forall n \in N, (H \cup A) \vdash T_n$$

If an experiment using the test set T_p selected from the collection $(T_n$ fails, then the algebra does not satisfy the axioms.

limits of this approach

- $S = \mathbb{N} \quad L = \{P\} \quad A = \{\exists x\, P(x)\}$

$(\mathcal{S})$: $P_{\mathcal{S}}$ is computable

let us suppose that (T_n) is a not biased test set $\Rightarrow$

$$\forall \mathcal{S},\ \mathcal{S} \models A \Rightarrow \mathcal{S} \models T_n \ \forall n \in \mathbb{N}$$

all non-logical axioms of T_n can be reduced

$$P(n_1) \vee \dots \vee P(n_p) \vee \neg P(m_1) \dots \vee \neg P(m_q)$$

let us consider $\mathcal{S}$ s.t.

$$P_{\mathcal{S}} = (x = m_1) \vee \dots \vee (x = m_q)$$

$$\mathcal{S} \models A \quad \text{and} \quad \mathcal{S} \not\models T_n$$

Thus the axioms are :

$$P(n_1) \vee \dots \vee P(n_p)$$

but with

$$P_{\mathcal{S}} : (x = m) \qquad m \neq n_1 \ \dots \ m \neq n_2$$

$$\mathcal{S} \models A \quad \text{and} \quad \mathcal{S} \not\models T_n$$

... (T_n) is empty

The only test set which is not biased is the empty one ! (when there is an existential quantifier in A)

EQUATIONAL CASE

THE SPECIFICATION IS HIERARCHICAL

EQUATIONAL

(i.e there is a sort of interest in Σ

there are predefined sorts

axioms are equations)

FOR EACH AXIOM A OF THE SPECIFICATION CONSIDER (T_n) obtained by:

*** IF x IS A VARIABLE OF THE SORT OF INTEREST, INSTANTIATE IT BY ALL THE TERMS OF THIS SORT, OF size LESS THAN n, WITHOUT VARIABLES OF THE SORT OF INTEREST;**

*** IN THE RESULTING SET OF EQUATIONS INSTANTIATE THE VARIABLES OF PREDEFINED SORTS BY RANDOM TERMS.**

WHAT ARE THE HYPOTHESES?

REGULARITY HYPOTHESIS

$$\forall x\ (\text{complexity}(x) \leq k \Rightarrow t(x)=t'(x)) \Rightarrow \forall x\ (t(x)=t'(x))$$

$\{x \mid complexity(x) \leq n\}$ finite for all integer n =>
$T_n = \{\ t(x)=t'(x) \mid \text{complexity}(x) \leq n\ \}$
is an acceptable collection of test sets.

COMPLEXITY <=>
length of a representative Σ-term denoting an object
(computation complexity)

UNIFORMITY HYPOTHESES

QUITE COMMON in TESTING METHODOLOGIES
<=> RANDOM SAMPLING TESTING STRATEGIES

$$\exists x\ (t(x)=t'(x)) \Rightarrow \forall x\ (t(x)=t'(x))$$

derived form of the above hypothesis:
Introduction of a META-CONSTANT.

The value of such a constant is intuitively a random value of the subdomain. The hypothesis becomes:

$$(t(c)=t'(c)) \Rightarrow \forall x\ (t(x)=t'(x))$$

An acceptable collection of test sets is given by

$$T_n = \{\ t(c)=t'(c)\ \}$$

for all integer n.

2nd (more realistic) case

Axioms are Conditional

Example:

le (x, y) = true & sorted (append (l, x)) = true =>
insert (append (l, x), y) =
append (append (l, x), y)

Pb: provide relevant values for
l, x, y.

Note: the uniformity hypothesis on
predefined sorts is no more valid!

1)- CASE OF EQUATIONAL AXIOMS

Regularity hypothesis for the sort of interest

Uniformity hypotheses for lower sorts

2)- CASE OF CONDITIONAL AXIOMS

Finite decomposition hypothesis

$$\forall x \in D, a(x)=b(x) \Rightarrow t(x)=t'(x)$$

$$\Leftrightarrow$$

$$\forall x \in D1, t(x)=t'(x)$$
with $D1 = \{x \mid a(x) = b(x)\}$

D1 may be an union, for instance:

$$\forall x \in N, or(le(x,2),le(5,x))=true \Rightarrow t(x)=t'(x)$$

$$\forall x \in D1_1, t(x)=t'(x)$$
$$\forall x \in D1_2, t(x)=t'(x)$$

where $D1_1 = \{n \mid n\leq 2\}$ and $D1_2 = \{n \mid n>5\}$

INFINITY OF SUCH SUBDOMAINS => REGULARITY-LIKE HYPOTHESES:

$$\forall i, 1\leq i\leq k, \forall x\in D1_i, t(x)=t'(x) \Rightarrow \forall x\in D1, t(x)=t'(x)$$

FINITE DECOMPOSITION HYPOTHESES

Method

Transform any conditional axiom
[f.i. $a(x) = true \Rightarrow f(x) = g(x)$]
into an equivalent set of equational axioms :

$$\{ f(t) = g(t') \mid a(t) = true \}$$

and apply the previous method.

G general, complete procedure for generating terms satisfying ~~boolean~~ equations

$$\begin{tikzcd} E(\Sigma\text{-Axioms}) \arrow[r] & G \arrow[r] & \sigma = \{\sigma_1 \dots \sigma_n \dots\} \\ P(\text{Premises}) \arrow[ru] & & \end{tikzcd}$$

$$E \models \sigma_i(\overset{A}{\cancel{P}})$$

The axiom becomes :

$$\{ f(\sigma_i(x)) = g(\sigma_i(x)) \mid \sigma_i \in \sigma \}$$

EXAMPLE

G is PROLOG

A is $le(2,x) = true \Rightarrow f(x) = g(x)$

E is
```
le(0, x, true).
le(s(x), 0, false).
le(s(x), s(y), B) :- le(x, y, B).
```

P is `?- le(s(s(0)), x, true).`

Prolog's answer is

$s(s(\_))$

A becomes

$f(s(s(y))) = g(s(s(y)))$

```
C1: isempty(emptyq,true).
C2: isempty(append(Q,I),false).
C3: remove(emptyq,emptyq).
C4: remove(append(Q,I),emptyq):-isempty(Q,true).
C5: remove(append(Q,I),append(Q',I)):- isempty(Q,false),remove(Q,Q').
C6: first(append(Q,I),I):- isempty(append(Q,I),false),isempty(Q,true).
C7: first(append(Q,I),J):- isempty(append(Q,I),false),isempty(Q,false),first(Q,J).
```

fig.3 Translation of the queue specification into PROLOG

A1: no constraint, no variable.

A2: no constraint, **regularity hypothesis** for Q, **uniformity hypothesis** for I:

Q_1 = emptyq, $I_1 = c_{11}{}^2$;
Q_2 = append(emptyq,$c_{21}{}^2$), $I_2 = c_{22}{}^2$;
...
Q_n = append^{n-1}(emptyq,$c_{n1}{}^2$),$c_{n2}{}^2$),...),$c_{nn-1}{}^2$), $I_n = c_{nn}{}^2$.

A3: no constraint, no variable.

A4: a constraint on Q: isempty(Q)=true solved by Q=isempty; no constraint on I, **uniformity hypothesis** for I.

Q = emptyq, I = c^4.

A5: a constraint on Q: isempty(Q)=false solved by Q=append(X,J); no constraint on X, **regularity hypothesis** on X; no constraint on I and J, **uniformity hypothesis** on I and J.

?- isempty(Q, false)

Q_1 = append(emptyq,$c_{11}{}^5$), $I_1 = c_{12}{}^5$;
Q_2 = append(append(emptyq,$c_{21}{}^5$),$c_{22}{}^5$), $I_2 = c_{23}{}^5$;
...
Q_n = append^n(emptyq,$c_{n1}{}^5$),$c_{n2}{}^5$),...),$c_{nn}{}^5$), $I_n = c_{nn+1}{}^5$.

A6: constraints on Q and I: isempty(Q)=true and isempty(append(Q,I))=false, solved with Q=emptyq, for any I (uniformity hypothesis). Q and I are instantiated with similar values than for A4.

A7: constraints on Q and I: isempty(Q)=false and isempty(append(Q,I))=false, solved with Q=append(X,J), for any I. Q and I are instantiated with similar values than for A5.

fig.4 Instantiation sets generated for the queue specification

```
specif sorted-list =
use bool, int
sort list;
operations

        el :                    -> list;     /* empty-list constructor */
        ap :       list * int   -> list;     /* append constructor */
        sorted :   list         -> bool;
        insert :   list * int   -> list;     /* defined for a sorted list */

preconditions
/* The operation insert is used to insert an integer in a sorted list and to get as a result a
sorted list. */
    pre(insert,L,X) = (sorted(L) = true)
axioms
    A1: sorted(el) = true;
    A2: sorted(ap(el,X)) = true;
    A3: le(X,Y) = true  => sorted(ap(ap(L,X),Y)) = sorted(ap(L,X));
    A4: le(X,Y) = false => sorted(ap(ap(L,X),Y)) = false;
    A5: insert(el,X) = ap(el,X);
    A6: le(X,Y) = true  => insert(ap(L,X),Y) = ap(ap(L,X),Y);
    A7: le(X,Y) = false => insert(ap(L,X),Y) = ap(insert(L,Y),X);
where
    L: list; X,Y: int;
end sorted-list ;
```

fig. 5: Specification of sorted lists

New hypothesis

σ infinite $\Rightarrow$ "regularity-like" hypothesis

$|\sigma_i|$ = length of σ_i

$$\forall \sigma_i \text{ s.t. } |\sigma_i| \leq k,\ \sigma_i(A) \Rightarrow \forall \sigma_i \in \sigma,\ \sigma_i(A)$$

new algorithm:

let φ, k increasing functions, n level of the test

- generate via G all the instanciations σ_i of complexity $\leq \varphi(n)$
- For each σ_i, generate the instanciations of the resulting equation using algorithm 1 with input $k(n)$

$\Rightarrow$ T_n is acceptable

G $\neq$ PROLOG

le (x,y) = true & sorted (append (l,x) = true
$\Rightarrow$ insert (append (l,x), y) =
append (append (l,x), y)

? sorted (append (l,x), true), le $(x,y,$ true).
$\Rightarrow$ L = empty, X = 0, Y = _ ;
L = empty, X = s(0), Y = s(_) ;
L = empty, X = s(s(0)) ; Y = s(s(_)) ;
...

G = SLOG ?

logic interpreter based on narrowing.

- equality is handled $\Rightarrow$ functions definitio
- global backtracking

? sorted (append (l,x)) = true), le (x,y) = true,
le (complexity (l), n) = true,
le (complexity (x), n) = true,
le (complexity (y), n) = true.

$\Rightarrow$ l = empty, X = 0, Y = _ ;
⋮
l = empty, X = $s^{n-1}(0)$, Y = $s^{n-1}(\_)$;
l = append (empty, 0), X = 0, Y = _ ;

```
/* complexity(L) = 1 and complexity of X and Y ≤ 3 */
L = el,   X = 0,                      Y = _;
L = el,   X = succ(0),                Y = succ(_);
L = el,   X = succ(succ(0)),          Y = succ(succ(_));

/* complexity(L) = 2 and complexity of X and Y ≤3 */
L = ap(el,0),                 X = 0,               Y = _;
L = ap(el,0),                 X = succ(0),         Y = succ(_);
L = ap(el,succ(0)),           X = succ(0),         Y = succ(_);
L = ap(el,0),                 X = succ(succ(0)),   Y = succ(succ(_));
L = ap(el,succ(0)),           X = succ(succ(0)),   Y = succ(succ(_));
L = ap(el,succ(succ(0))),     X = succ(succ(0)),   Y = succ(succ(_));

/* complexity(L) = 3 and complexity of X and Y ≤ 3 */
L = ap(ap(el,0),0),                                X = 0,               Y = _;
L = ap(ap(el,0),0),                                X = succ(0),         Y = succ(_);
L = ap(ap(el,0),succ(0)),                          X = succ(0),         Y = succ(_);
L = ap(ap(el,succ(0)),succ(0)),                    X = succ(0),         Y = succ(_);
L = ap(ap(el,0),0),                                X = succ(succ(0)),   Y = succ(succ(_));
L = ap(ap(el,0),succ(0)),                          X = succ(succ(0)),   Y = succ(succ(_));
L = ap(ap(el,succ(0)),succ(0)),                    X = succ(succ(0)),   Y = succ(succ(_));
L = ap(ap(el,0),succ(succ(0))),                    X = succ(succ(0)),   Y = succ(succ(_));
L = ap(ap(el,succ(0)),succ(succ(0))),              X = succ(succ(0)),   Y = succ(succ(_));
L = ap(ap(el,succ(succ(0))),succ(succ(0))),        X = succ(succ(0)),   Y = succ(succ(_));
```

success or failure?
or the oracle problem

two values of X are computed:

$insert_X(append_X(\sim)\sim) = CL1$

and $append_X(append_X(\sim)\sim) = CL2$

We are faced with two "concrete lists"
Are they equal at the abstract level?

$\Rightarrow$ old problem!

- implement eq_X (interpretation of equality)

- give Distinguishing Set of operations with the specification [Kamin]

$first_E(CL1) = first_E(CL2)$

$first_E(remove_E(CL1)) = first_E(remove_E($

... etc

heavy and not always computable!