Aver

Aver, v.t. (err)
- to assert, affirm, prove

A verifying environment

acknowledgements
related projects
workers

as we go

(Jim Cunningham)
This Talk

A semi-historical overview

verification ... automated reasoning

a new computing concept:

define \( lpm = \)

Ayer Roots

Constructive Design

with Hoare-style axioms

e.g., given nats \( x, y, z > 0 \)

find \( q \) such that:

\[ x = q \times y + r \land 0 \leq r < y \]

Method: fit post condition to axiom

establish precondition for axiom

force progress
**Verification**

(Diagrammatic)

1. **Process Specification**
   - **Antecedent:**
     - Input is:
     - & not all placed
   - **Consequent:**
     - Output is:
     - & all placed

   **Invariant** (the difficult bit is factored out.):
   - if placed then on board
   - & without overlap
   - & minimizing weighted distance

   (now make this invariant a property of the board data structure)

   \[
   x := x + 1 \quad \{ x + 1 > 0 \} \quad \{ x > 0 \}
   \]
Complementary aspect:

**DATA STRUCTURE SPECIFICATION**

- **functions**
  - member, ≤, min

- **processes**
  - put, get
  - A: true
  - C: member(x)

**Implementation**

- eg1. tree + functions: root, left, right

  **Invariant:**
  
  member(x) \(\Rightarrow\) x = root
  
  \(\forall x \in \text{root} \land \text{left}. \text{member} \)
  
  \(\forall x \notin \text{root} \land \text{right}. \text{member}(x)\)

**eg2. bounded buffer**

+ **functions**

  - n, size, tpos, buffer

- **buffer**

  ![Buffer Diagram]

**BOARDS DATA STRUCTURE**

- **(skip)**

- includes
  - board dimensions
  - component placements

- maybe
  - connection routings

- geometric expression of invariants

- assumes
  - component data structure

- type identifier

- pin positions

- pin types

- component dimensions
**System Specification?**

Many processes

- Not a simple combination

*Telephone System:*

- **Entities** such as subscribers
- **Relations** which pertain
  - conversation \((x, y)\)
  - off-hook \((x)\)

*Processes / Actions*

- **External** dial \((x, y)\)
- **Internal** switch-through \((x, y)\),
  - clear-down \((x)\)

**Many Processes:**

- **Disconnected**
- **Not enough digits**
- **Setup**
- **Not set-up**
- **Conversation**
- **Digit**
- **Engaged**
- **Ringing**

Asynchronous behaviour

Distributed implementation

Partial / Local views
Invariants for Distributed Systems

Process properties difficult for a system like an O/S or a telephone system.

But invariant properties /axioms are easier to focus on. E.g., for a telephone system:

\[ \forall x, y : \text{subscriber}, \]
\[ \text{conversation}(x,y) \rightarrow \text{offhook}(x) \land \text{offhook}(y) \]

Constructive Design Steps

deny axioms (negate invariants)
find disjunctive normal forms terms are preconditions for processes to restore status quo

E.g., negate axiom implying handsets off hook in conversation:

\[ \Rightarrow \exists x, y : \text{subscriber} \]
\[ \text{conversation}(x,y) \lor \neg \text{offhook}(x) \]

pre condition for “clear down” process
APPLICATION: Conveyor System Example

Work by Cunningham & Kramer on Design of Distributed Systems

Example: Emergency Quiescent

CONVEYOR
ON \land R \land E \lor E \lor \neg OFF

ON/OFF
Running
LOAD ON END

Hierarchical Aspect:

A Quiescent property of the next level

Real Example (60 pages)
Quiescents simpler than Dynamics
Covered 90% of Specified Properties

DISTRIBUTED DESIGNS

1. Quiescents of whole = Conjunction of Quiescents of parts with common variables bound

2. Invalid Quiescent \implies Precondition for action
   e.g. Suppose \( L \land R \land R \) false
   \( \land \land \land \) false
   \( \land \land \land \) true
   then do something!
   (one term of a disjunctive set)

3. Hose logic verifies the actions
   \( \{ \text{anything} \} \) before
   switch on \( C \)
   ensures \( \{ C \} \) etc.
REALITY

Details rather complex

process verification still:

- guarded non-determinism
- concurrency without interference in
  Owicki-Gries sense

but "natural" exploitation of
first order logic proofs.

- reasoning by contradiction
- normalised forms
- even skolemisation

By conversation(\(x, y\))
~ conversation(\(x, \text{dialled}(x)\))

REQUIREMENTS FOR 'TOOL' SUPPORT

1. Mechanical Aids for any chosen logic
   - certainly for first-order P.C.
   - but real people use equations too
   - choice of style is important
   for simplicity of understanding

2. A specification database
   - with housekeeping
   - validation possible
   - and ways of re-using
     specifications

3. A hierarchical specification language
   - the formal basis was unclear,
     but appeared to transcend
     details of first order style
AVER Hierarchical Specification Language

Hierarchical abstractions
Axiomatic / Assertional

Declarative / functional

Strongly typed - see later

Composable / Modularisable

Parameterisable

All things to all persons

Almost without meaning

Unless you provide it by tools.

\begin{verbatim}
define daddy = \{<body>, arms, legs\} \{ \text{parameters (normally typed)} \}

\text{axioms}

(body above legs) inside arms
\end{verbatim}

A Parameterised Structure

(loose domain interpretation via Scott's Information Structure)
AVER INTERFACE

Top level menu:

create: create a new definition
modify: modify an existing definition
look: inspect a definition
print:
help:

where: in directory

FUNCTION PROVIDED IS
extensible
menu-driven
Composition
of other tools

A UNIX FILE STORE

Structure of Specification

Structure of file store

(Implementation by Gordon Gallacher)
Main Features in Practice

Hierarchical domain structures are very rich and slippery, but admit many notions of inference, and considerable economy of expression.

Special logics include:
- first order equational logic over hierarchically typed partial algebras
  (Cunningham & Dick Acta '85)
- program logics built up by
  Dijkstra or Goldblatt axiomatization
  (LNC'83 130)
- domain equations themselves
  (Suszko: Math. Syst. Th. '81)
  \[ \text{def } uft = \text{char}^* + \text{id} \rightarrow uft \]
  (unbounded research)

Main Points about the Equational Logic

Simple type lattice, based on subsets

Diagram:
- \( U \)
- \( \text{BOOLEAN} \) \( \rightarrow \) \( \text{STACK} \)
  - \( \text{FALSE} \)
  - \( \text{TRUE} \)
  - \( \text{NULL} \) \( \rightarrow \) \( \text{NONNULL} \)
  - \( \text{ELEMENT} \)

Monotonic function templates
- \( \text{NONNULL} \rightarrow \text{ELEMENT} \) implies \( \text{STACK} \rightarrow \text{ELEMENT} \)
- for empty: \( \text{NULL} \rightarrow \text{TRUE} \) implies \( \text{NULL} \rightarrow \text{BOOLEAN} \)
- (the templates for each function form a lattice)

Weak equality \( x = x \) only if defined!
- for \( D, D_2 \neq \emptyset \) \( = : D \times D_2 \rightarrow \text{BOOLEAN} \)
- for \( D, D_2 \neq \emptyset \) but \( D_1 \neq \emptyset \) and \( D_2 \neq \emptyset \) \( = : D \times D_2 \rightarrow \text{FALSE} \)
- for \( D \neq \emptyset \) or \( D_2 \neq \emptyset \) \( = : D \times D_2 \rightarrow \emptyset \)
- (so equality can only hold non-empty intersection)
AVOIDING COMPLICATION & BIAS

STACK EXAMPLE

L.E. "SYNTAX"
nullstack : STACK
pop : STACK → STACK
push : STACK + ELEMENT → STACK
top : STACK → ELEMENT
isempty : STACK → BOOLEAN

POSSIBLE "SEMANTICS"

pop(push(s,e)) = s
top(push(s,e)) = e
isempty(nullstack) = true
isempty(push(s,e)) = false

PROBLEM AREAS

top(nullstack) = ?
pop(nullstack) = nullstack ?
push(pop(s), top(s)) = s ?

TOTAL ALGEBRA
WITH ERROR TYPE

STACK ERROR → STACK ERROR

if s error then

...
Hierarchical Sorts

NONNULL -> STACK
NONNULL -> ELEMENT
NULLSTACK -> TRUE
NONNULL -> FALSE

top(nullstack) = ?

Hierarchical/launched tracing
STACKERROR -> STACKERROR

prepares for non-error a error domain.
IMPORTANT CONSEQUENCES

1. Natural treatment of \[ \div 0 \], top (nullstack),
etc. as undefined
   (the original purpose)

2. More syntactic processing
   in theorem proving
   (\& less semantic processing)
   (there is a finite set of m.g.u.'s)

3. Conditionals can be processed properly,
   \[
   \text{if-then-else: } \begin{cases}
   \text{bool x } D_1 \times D_2 \rightarrow D_1 \cup D_2, \\
   \text{true x } D_1 \times D_2 \rightarrow D_1, \\
   \text{false x } D_1 \times D_2 \rightarrow D_2, \\
   \emptyset \times \(D_1 \times D_2) \rightarrow \emptyset ?
   \end{cases}
   \]

THEOREM PROVING

- '60 Early Serious Attempts
- '63 Hopeless Results
- '65 Recognition of Hard Problem
- '65 Breakthrough
  Robinson Resolution Principle for First Order Predicate Calculus
- '67 Development of Resolution
- '70 Leading to Prolog inter alia
- '73 Realization that RTP for FoPc
  weak if equality included
  & hard to help interactively
- '75 Trend (in reaction) to interactive
  Natural Deduction
  \& Rewriting Systems
- '77 Recognition of earlier ('70)
  breakthrough by Keith & Bender
  for Equational Rewriting
  Tendency for ND \& KS to converge
- '79 Performance Benefits of Clipped Systems
  for both RTP \& ND /ERR
- '81 Steps towards combining KS \& RTP
- '83
Limitations of TP in Practice

"Heavy" intellectual work
Expenditure on Computing Resources
Wont cope with some perceptions of "real" problem!
Eg. flat problems + entities

Useful strategies for few logics

Record ERR

Strengths

Commercially valuable for
numerical interest problems
by 20-30 var usable logic for VLSI

Involved for theory, also
small abstraction of problems
for applications in
small kernel w/ var
Eg. Security

Major T.P. work

US
Soger-Moore DRS
Affirm ERR
L.livernois RTP

GERMANY
Makigraf RTP

UK
LCF DRS

Minor UK
AvnoTP RTP + ERR KB

FRANCE
Red et.al. ERR KB
Open

Can "real" problems be sufficiently stratified to give tractable TP (at local level)?

How do we integrate TP with Reasoning about Specifications?
- partial answer: specialize the TP

Remember TP explodes with poor strategies. Not good enough for mortals.

First order Predicate logic tools
(Cunningham & Zappacosta, SPE 1983)

Basic
- Passing
- Renaming
- Closing wrt free variables
- Standardizing connector sets
- Skolemize
- Normalize to DNF etc
- Simplify, Factorise

Complex
- Resolution Provers with performance issues
Resolution Principle

Two Steps:
1. Classical Logic
   Conjunctive Normal Form

\[(P \lor Q) \land (\neg P \lor R)\]

\[Q \lor R\]

2. Unification for Predicate Instances

\[(P(x_1) \lor Q(x_1)) \land (\neg P(a) \lor R(x_2))\]

Substitution \(\sigma = \{a/x_1\}\)

\[Q(a) \lor R(x_2)\]

Note:
Resolution is Refutation Complete

\[(\text{dual is proof complete})\]

\[\therefore \text{ a proof can be expressed using resolution}\]

Nobody promises either

i) Proof exists

\[\therefore TP \text{ will find it in bounded time}\]

So strategy is of paramount importance (where restrictions like horn clauses only)
**KEY CONCEPTS**

**Substitutions**  
$\sigma : \text{var} \rightarrow \text{terms}$  
extended to $\text{expression} \rightarrow \text{expression}$

**Matching**  
$M(\bar{e}, \bar{e}') = \{ \sigma | \sigma(\bar{e}) = \bar{e}' \}$

**Unification**  
$U(\bar{E}, \bar{E}_2) = \{ \sigma | \sigma(\bar{E}_1) = \sigma(\bar{E}_2) \}$

**Coercion**  
$C(\bar{E}, t) = \{ \sigma | \sigma(\bar{E}) \text{ is of type } t \}$

**Superposition**  
$S(\bar{E}_1, \bar{E}_2) = U(\bar{E}_1, \bar{E}_2') \quad \text{where } \bar{E}_2' \equiv \bar{E}_2$

$U(\bar{E}_1, \bar{E}_2') \quad \text{where } \bar{E}_1' \equiv \bar{E}_1$

$U(\bar{E}_1', \bar{E}_2) \quad \text{where } \bar{E}_1' \equiv \bar{E}_1$
CRITICAL EXPRESSIONS

- the $V_2 \to KB$ & efficient ER

R1: $f(e, x_1) = x_1$
R2: $f(x_1, e) = x_1$
R3: $f(x_1, x_1) = e$
R4: $f(f(x_1, x_2), x_3) = f(x_1, f(x_2, x_3))$

Common instance of R3 & R4 L.H.S found by superposition

\[
f(f(x_1, x_1), x_2)
\]

R3

\[
f(e, x_2)
\]

R1

\[
f(x_1, f(x_1, x_2))
\]

R4

Each expression has a normal form so equivalence is decidable

The Knuth-Bendix algorithm uses critical pairs and ordering to seek a confluent set from given axioms but it may not exist or may be infinite.

CONFLUENCE

Rewrite rules are ordered to simplify

\[f(x_1, f(x_1, x_2)) \to x_2\]

Rewrite system is finitely terminating

(No infinite loops, e.g. with $x + (y + z) = (x + y) + z$)

With just two more critical pairs get: $f(x_1, x_3) = f(x_2, x_1)$
Confluent sets make equational reasoning directed and decidable but Superposition and critical pair generation is a powerful inference step in itself.

A Confluent Set for Stack: a: stack, b: element

pop (nullstack) ⇒ nullstack
top (push (a, b)) ⇒ b
pop (push (a, b)) ⇒ a
push (pop (a), top (a)) ⇒ a

4. Equivalence is a relation that is:
   - reflexive: a equiv a
   - symmetric: a equiv b if and only if b equiv a
   - transitive: a equiv c if a equiv b and b equiv c

In a software system that is to be implemented:
S is a finite set of items some of which may be equivalent,
E is a set of pairs of equivalent items from S,
EQUIV is a triadic predicate on E x S x S such that:

EQUIV(e, s1, s2) is true if and only if from a given set e of pairs of equivalent items it also follows that s1 equiv s2

Consider the formal specification:

intuitive

INIT: E x S x S → E
ENTER: E x S x S → E
EQUIV: E x S x S → Bool

(1) EQUIV(INIT, s, t) = (s same as t)
(2) EQUIV.ENTER(e, s, t, u, v) =
    EQUIV(e, u, v) or (EQUIV(e, s, u) and EQUIV(e, t, v))
    or (EQUIV(e, s, v) and EQUIV(e, t, u))

a State the properties of EQUIV which should be proved in order to validate the above formal specification.

Consider also another specification in which equivalence is represented by equivalence classes, each class containing elements of E which are equivalent to each other, with a representative element selected for each class.

reclass

INIT: E x S x S → E
ENTER: E x S x S → E
REP: E x S → S

(3) REP.INIT(s) = s
(4) REP.ENTER(e, s, t, u) =
    if (REP(e, t) = REP(e, u)) then REP(e, s)
    else REP(e, u)

b Verify the representation of intuitive by reclass under the interpretation:

(5) EQUIV(e, s, t) = (REP(e, s) = REP(e, t))

by showing that are satisfied in reclass.
STATE OF THE ART: Knuth/Bendix Methods

1. Horn Clause Logic with Equality
   Hsiang, ICALP '83
   Paul, Eurocal '85

2. Full First Order Predicate Logic with Equality (by refutation)
   (requires Associative-Commutative Unification Methods)
   Coming - see Hsiang '83

3. Equation Solving by exploiting Confluence
   "Narrowing"
   Kirchner, Lescanne, U. of Nancy

STATE OF THE ART

Equational Reasoning by "narrowing"

s1
\[ \log_2(x+1) + \log_2(x-1) = 3 , \]
\[ \text{ans}(x) \]

s2
\[ \log_2(\exp(x, 2) - 1) = 3 , \]
\[ \text{ms}(x) \]

s3
\[ \log_2(p-1) = 3 , \]
\[ \text{ans}(\text{nsqrt}(p-1)) \]

s4
\[ \log_2(p) = 3 , \]
\[ \text{ans}(\text{nsqrt}(p+1)) \]

s5
\[ x = 3 , \]
\[ \text{ans}(\text{nsqrt}(\exp(2, x)+1)) \]

s6
\[ \text{ans}(\text{nsqrt}(\exp(2, 3)+1)) \]

20/30 rules for log, exp, nsqrt etc.
these are the lemmas
STATE OF THE ART

Program Synthesis by narrowing

Axioms & Rules (>100)
for substitution
formal expressions
weakest preconditions
simple arithmetic
lists, sets, etc.

No synthesis results
(but trivial verification)

Some theorems we didn't know before

REALITY

Theorem proving tools
- started earlier & better developed
- still a bit clumsy
- no panacea, but already useful

Specification system
- still very experimental
- unstable because of semantic issues
- var version new & flakey

Integration & Interface
- intermediate forms for tool/db interaction
- relation between proof structure & specification structure unclear (combinators seem to help)
- human interfaces not yet good enough for outsiders.