AVER

avẽr,´ v.t. (-rr)

- to assert, affirm, prove

A verifying enviroment

(Jim Cunningham)

acknowledgements

related projects

workers

as we go

THIS TALK

A semi-historical overview

verification ... automated reasoning

a new computing concept:

define *lpm* =

AVER ROOTS

Constructive Design with Hoare-style axioms

e.g. given nats x, y, $y > 0$

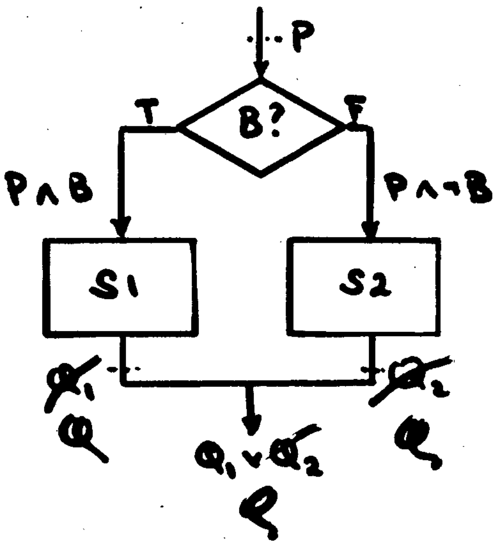
find q such that:

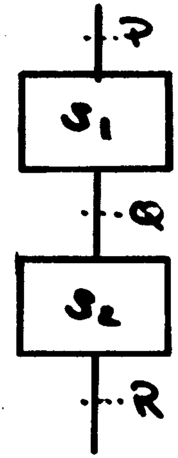
$$\underbrace{(x = q * y + r) \,\&\, 0 \leq r}\ \&\ r < y$$

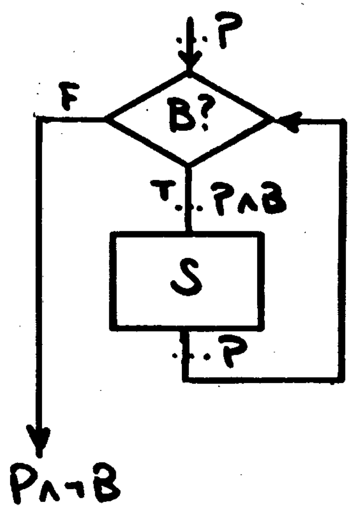
Method: fit post condition to axiom
establish precondition for axiom
force progress

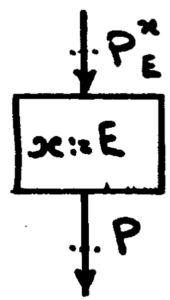
VERIFICATION

(Diagrammatic)









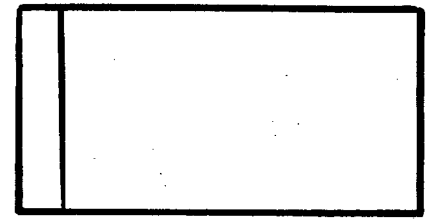
$$x := x+1$$

$$\{x+1>0\} \qquad \{x>0\}$$

PROCESS SPECIFICATION

Antecedent:

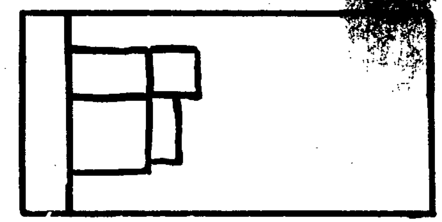
input is:



& not all placed

Consequent

output is:



& all placed

Invariant (the difficult bit is factored out)

if placed then on board
& without overlap
& minimising weighted distance

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

Complementary aspect:

DATA STRUCTURE SPECIFICATION

eg: ordered set

functions

member , $\leq$, min

processes

put , get

A: true

C: member(x)

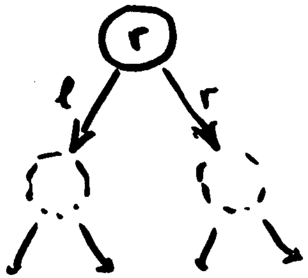
I: $\forall y \; y \neq x \rightarrow$ member(y) preserved

implementations

eg1. tree + functions: root, left, right

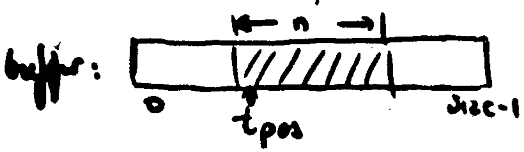
Invariant:

member(x) $\Leftrightarrow$ x = root
$\vee$ x $\leq$ root $\wedge$ left.member(x)
$\vee$ x not $\leq$ root
$\wedge$ right.member(x)



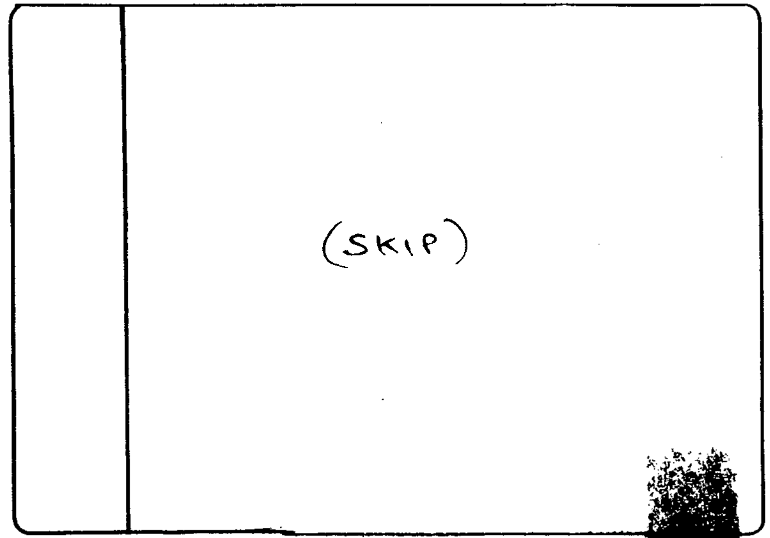
eg2. bounded buffer

+ functions n, size, tpos, buffer



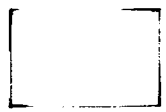
member(x) = ?

BOARD DATA STRUCTURE



includes board dimensions
component placements
maybe connection routings
geometric expressions of invariants

assumes component data structure



type identifier
pin positions
pin types
component dimensions

SYSTEM SPECIFICATION ?

Many processes

Many data structures

– Not a simple combination

e.g. telephone system:

<u>entities</u> such as subscibers

<u>relations</u> which pertain

conversation (x, y)

off-hook (x)

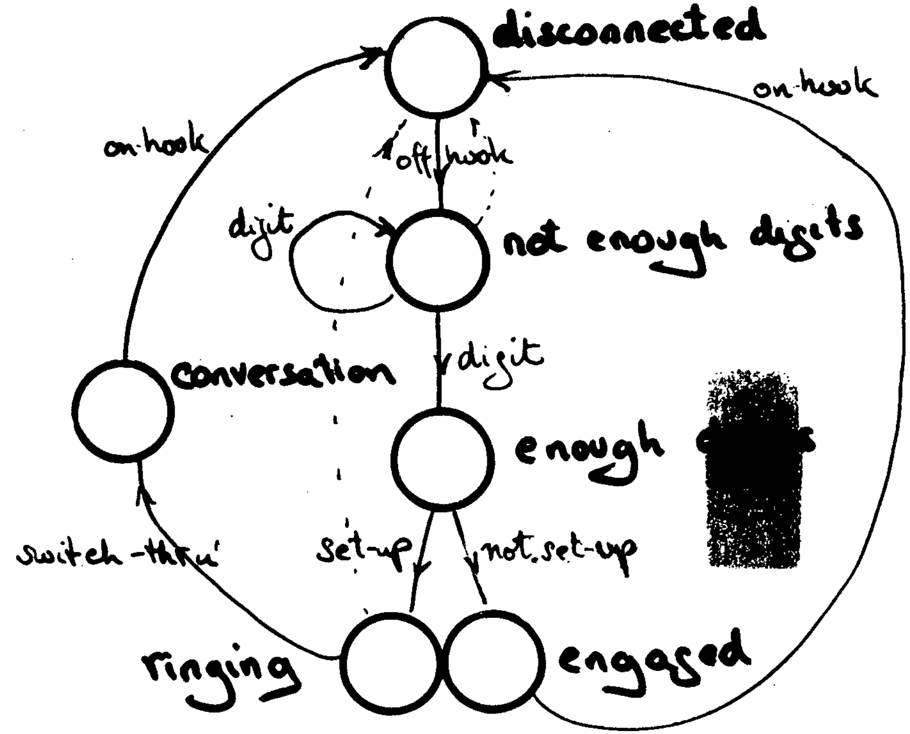
<u>processes</u> / <u>actions</u>

external dial (x, y)

internal switch-through (x, y),

clear-down (x)

Many Processes:



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:

$\forall x, y$: subscriber,
(conversation(x, y) $\rightarrow$
offhook(x) & offhook(y))

Constructive Design Steps

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

eg negate axiom implying
handsets off hook in conversation:

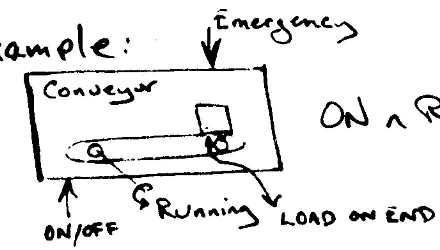
$\Rightarrow$ $\exists x, y$: subscriber
conversation(x, y) & $\neg$offhook(x)
or conversation(x, y) & $\neg$offhook(y)

pre condition for
"clear down" process

APPLICATION : Conveyor System Example

Work by Cunningham & Kramer
on Design of Distributed Systems

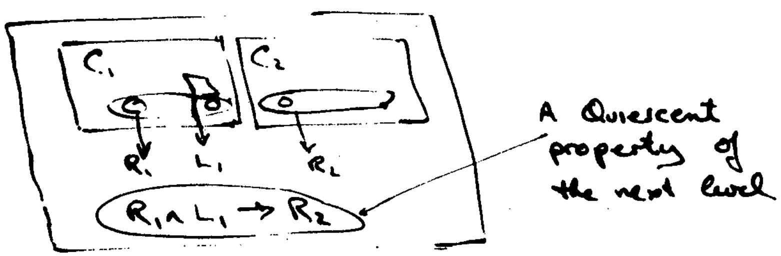
Example:



Quiescent

$ON \wedge R \wedge \overline{E} \vee E \vee OFF$

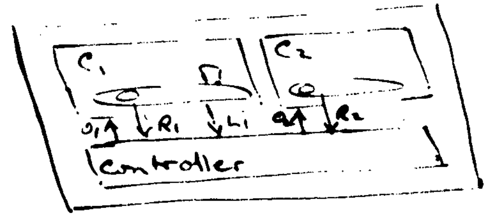
HIERARCHICAL ASPECT :



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 $\Rightarrow$ Precondi[illegible] for
   ac[illegible]

e.g. Suppose $L_1 \wedge R_1 \to R_2$ false

i.e. $\overline{L_1 \wedge R_1} \vee R_2$ false

i.e. $L_1 \wedge R_1 \wedge \overline{R_2}$ true

then do something !
(one term of a disjunctive set.)

3. Hoare logic verifies the actions

i.e. {anything} before
switch on C_2
ensures $\{R_2\}$ 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

$\exists y$ conversation(x, y)

$\sim$ conversation$(x, \text{dialed}(x))$

Requirements for 'Tool' Support

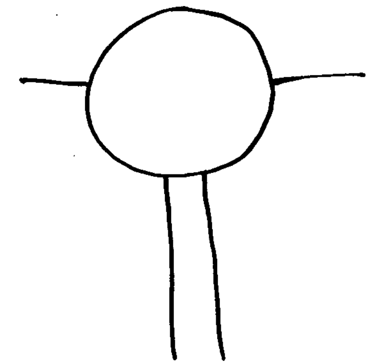
1 Mechanical Aids for any chosen logic
  - certainly for first-order p.c.
  - but real people use equations too
  - choice of style is important for simplicity of understanding

2. A specification database
  - with housekeeping
  - validation possibilities
  - 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 - <u>see later</u>
Composable / Modularisable
Parametrisable



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

<u>axioms</u>

(body above legs) inside arms

}

All things to all persons

Almost without meaning

Unless you provide it by tools.

A Parameterised Structure

(loose domain interpretation
via Scott's Information Structure)

```
def lift = {( floor,
              time,
              request : time # floor # floor → bool,
              at : floor # time → bool
            )
```

type

local definition

```
def service = {(t: time,
                f1: floor, f2: floor
                at
               )
               axioms
               exists t1: time,
               at(f1, t) and at(f2, t1)
               and t ≤ t1
               }

axioms all t: time,
request(t, from, to)
=> exists t1: time,
   service(t1, from, to)
   and t ≤ t1

}
```

```
def lift = temporal_logic & {( floor,
                               request : floor # floor → bool,
                               at : floor → bool
                              )
```

domain intersection combinator

?

domain combinators (also ++)

. . .

visible temporal operators

```
axioms

□ request (from, to) =>
    ◇ service (from, to)

}
```

USEWP at work

AVER INTERFACE

TOP LEVEL A.N. OTHER is D.T. @ IST / ARC & UNIX

FUNCTION PROVIDED IS
- extensible
- menu-driven
- composition OF OTHER TOOLS

Top level menu:

where: in directory

create: create a new definition

modify: modify an existing definition

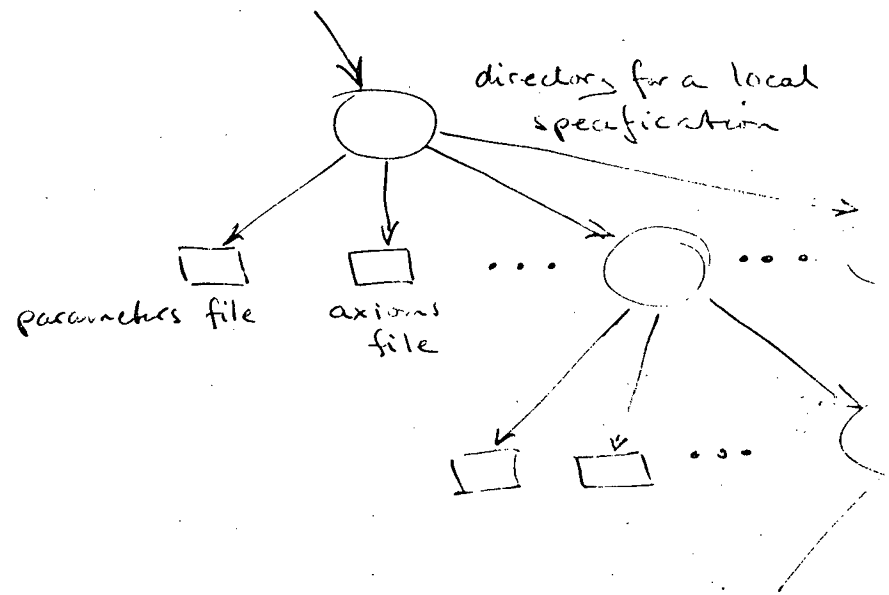
look: inspect a definition

print:

help:

AVER INTERNAL FORM

A UNIX FILE STORE



Structure of Specification

$\sim$ 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 axiomatizing
(LNCS 130)

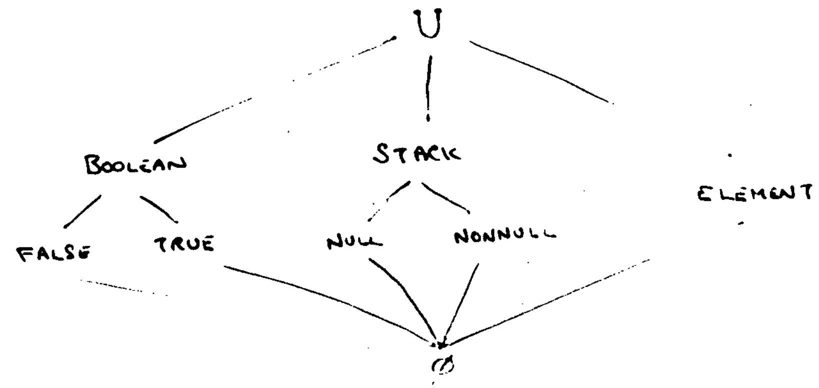
domain equations themselves
(Smyth: Math. Sys. Th. '81)

$\underline{def}$ $uft = char^* ++ id \to uft$!

(unbounded research)

Main Points about the Equational Logic

Simple type lattices based on subsets



Monotonic function templates

eg., for top: NONNULL → ELEMENT implies STACK → ELEMENT

for isempty: NULL → TRUE implies NULL → BOOLEAN

(the templates for each function form a lattice)

Weak equality ($x = x$ only if defined!)

for $D_1 \sqcap D_2 \not\subseteq \emptyset$ $=: D_1 \times D_2 \to$ BOOLEAN

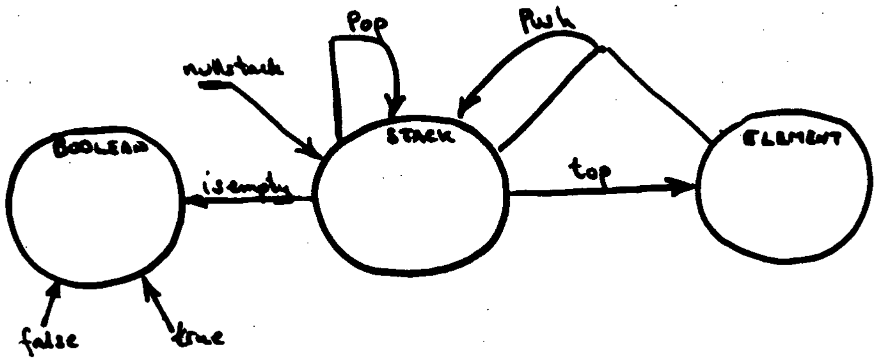
for $D_1 \sqcap D_2 \subseteq \emptyset$ but $D_1 \not\subseteq \emptyset$ and $D_2 \not\subseteq \emptyset$
$=: D_1 \times D_2 \to$ FALSE

for $D_1 \subseteq \emptyset$ or $D_2 \subseteq \emptyset$ $=: D_1 \times D_2 \to \emptyset$

(so equality can only hold non-empty intersecti

AVOIDING COMPLICATION & BIAS

STACK EXAMPLE



I.E. "SYNTAX"

nullstack : STACK
pop : STACK → STACK
push : STACK × ELEMENT → STACK
top : STACK → ELEMENT
isempty : STACK → BOOLEAN

POSSIBLE "SEMANTICS"

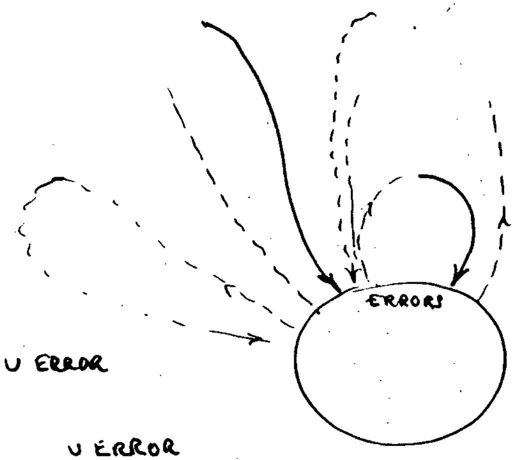
pop(push(s,e)) = s
top(push(s,e)) = e
isempty(nullstack) = true
isempty(push(s,e)) = false
. . .

PROBLEM AREAS

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

∃ stack for which

TOTAL ALGEBRA with ERROR TYPE



∪ ERROR

∪ ERROR

etc.

h

STACK ∪ ERROR → STACK ∪ ERROR
etc.

h
h
h

if $s \neq$ error then
etc.

HIERARCHICAL SORTS

NONNULL → STACK

NONNULL → ELEMENT

NULLSTACK → TRUE
NONNULL → FALSE

•
•
•
•

top (nullstack) = ?

•
•

HIERARCHICAL / SORTED ERRORS

etc.

STACKERROR → STACKERROR

etc.

separate for non-error
& error domain!.

IMPORTANT CONSEQUENCES

RJC:

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.

$$\text{if\_then\_else} : \left\{ \begin{array}{l} \text{bool} \times D_1 \times D_2 \to D_1 \sqcup D_2, \\ \text{true} \times D_1 \times D_2 \to D_1 \\ \text{false} \times D_1 \times D_2 \to D_2 \\ \emptyset \times D_1 \times D_2 \to \emptyset \end{array} \right\}$$

is if then else
the only specialisation
to use domain

THEOREM PROVING

'60 Early Serious Attempts
'63 Hopeless Results
Recognition of Hard Problem
'65 Breakthrough:
Robinsons Resolution Principle for First Order Predicate Calculus
67 –70 Development of Resolution Leading to Prolog inter alia
–73 Realization that R.T.P 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 Knuth & Bendix for Equational Rewriting
79 Tendency for ND & KB to converge
81 Performance Benefit of Typed Systems for both RTP & ND / ER.R.
83 Steps towards combining KB & RTP

Limitations of TP in Practice

"Heavy" intellectual work

Expensive on Computing Resources

Won't cope with some perceptions
of "real" problems
eg. flat problems 50 entities

Useful strategies for few logics

FOPC ERR

3

Strengths

Commercially valuable for
modest flat problems
eg 20-30 var comb. logic
for VLSI

Invaluable for theoretical eg
- small abstractions of problems
& for applications in
small kernel areas
eg. Security

Major T.P. work

U.S.

Boyer-Moore	DRS
Affirm	ERR
L. Livermore	RTP
...	

GERMANY

Markgraf	RTP

UK

LCF	DRS

Minor UK

AverTP	RTP + ERR KB

FRANCE

Rev 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: specialise the TP

Remember TP explodes with poor strategies. N.D not good enough for mortals.

First order Predicate Logic Tools

(Cunningham & Zappacosta SPE '1983)

Basic

Parsing

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

$$(P \vee Q) \wedge (\overline{P} \vee R)$$

$$\frac{\phantom{(P \vee Q) \wedge (\overline{P} \vee R)}}{Q \vee R}$$

2 Unification for Predicate Instances

$$(P(x_1) \vee Q(x_1)) \wedge (\overline{P}(a) \vee R(x_2))$$

Substitution $\sigma = \{a/x_1\}$

$$Q(a) \vee R(x_2)$$

Note

Resolution is Refutation Complete

(dual is proof complete)

$\approx$ a proof can be expressed using resolution

Nobody promises either

i) Proof exists

or ii) TP will find it in bounded time

So strategy is of paramount importance
(& restrictions like horn clauses only)

KEY CONCEPTS

Substitions σ : var $\rightarrow$ terms

extended to Expression $\rightarrow$ Expression

Matching $M(E, E') = \{\sigma \mid \sigma(E) = E'\}$

Unification $U(E_1, E_2) = \{\sigma \mid \sigma(E_1) = \sigma(E_2)\}$

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

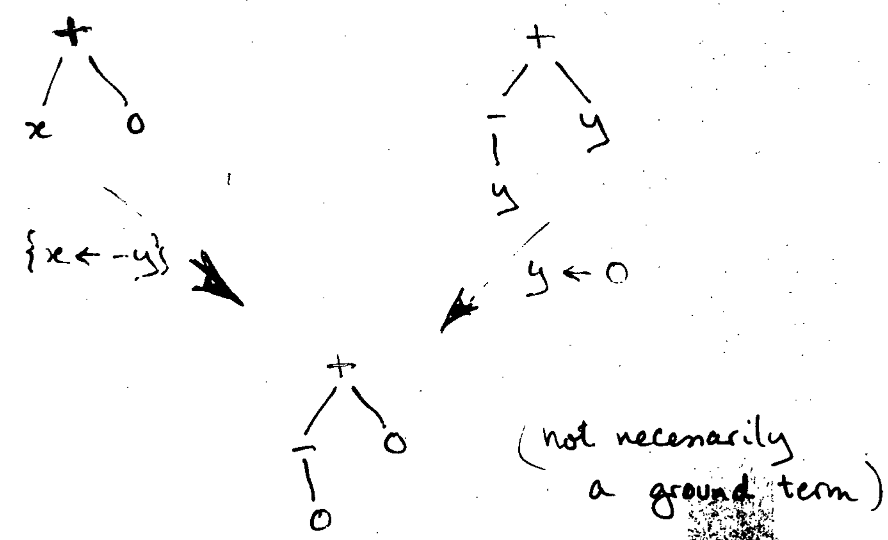
Superposition $S(E_1, E_2) =$

$$U(E_1, E_2') \text{ where } E_2' \hat{\subseteq} E_2$$

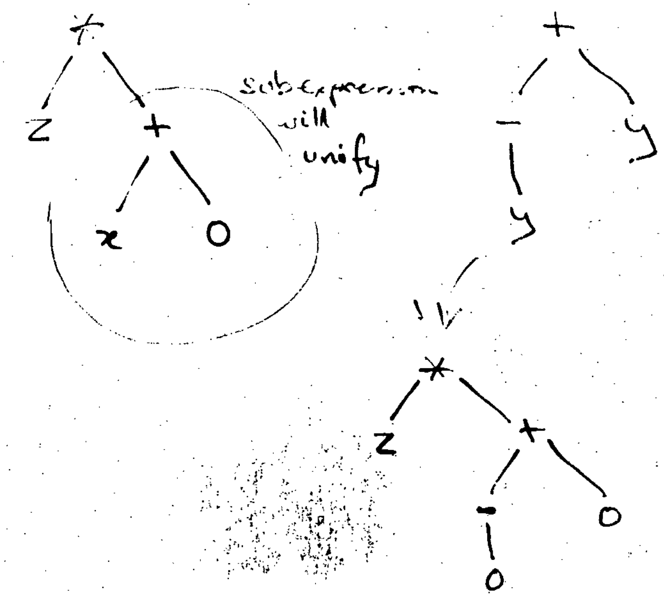
$$\cup$$

$$U(E_1', E_2) \text{ where } E_1' \hat{\subseteq} E_1$$

UNIFICATION



SUPERPOSITION



CRITICAL EXPRESSIONS

— the key to KB & efficient ER

R1: $f(e, x_1) = x_1$

R2: $f(x_1, e) = x_1$

R3: $\underline{f(x_1, x_1)} = e$

R_4: $f(\underline{f(x_1, x_2)}, x_3) = f(x_1, f(x_2, x_3))$

Common Instance of R3 & R4 l.h.s found by <u>superposition</u>

$$f(f(x_1, x_1), x_2)$$

$$f(f(x_1, x_1), x_2) \xrightarrow{R3} f(e, x_2) \xrightarrow{R1} x_2$$

$$f(f(x_1, x_1), x_2) \xrightarrow{R4} f(x_1, f(x_1, x_2))$$

R_5: $x_2 = f(x_1, f(x_1, x_2))$

orientation of critical pair

With just two more critical pairs get: $f(x_1, x_2) = f(x_2, x_1)$

CONFLUENCE

Rewrite rules are ordered to simplify

eg. $f(x_1, f(x_1, x_2)) \rightarrow x_2$

Rewrite system is finitely terminating

(no infinite loops, as with

$x + y = y + x$.)

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)

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

[redacted]

STATE OF OUR ART : Conditionals & Induction

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
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 [redacted] 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.

repclass

```
INIT:              -> 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 **repclass** under the interpretation

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

by showing that [redacted] are satisfied in **repclass**.

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 Associate-Commutative Unification Methods)

   Coming - see Hsiang '83

3. Equation Solving by exploiting Confluence

   "Narrowing"

Kirchners, Lescanne, U. of Nancy

STATE OF THE ART

Bundy / Dick 85

Equational Reasoning by "narrowing"

S1 $\log_2(x+1) + \log_2(x-1) == 3$, $ans(x)$

S2 $\log_2(exp(x,2)-1) == 3$, $ans(x)$

S3 $\log_2(p-1) == 3$, $ans(nsqrt(p))$

S4 $\log_2(p) == 3$, $ans(nsqrt(p+1))$

S5 $x == 3$, $ans(nsqrt(exp(2,x)+1))$

S6 $ans(nsqrt(exp(2,3)+1))$

⋮

S10 $ans(psqrt(exp(2,3)+1))$

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 sythesis 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
- vax 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.