BCS-FACS

86.1.16

Ian Cottam
Lecture 1

VDM + Mule

Q. What is the Vienna Development Method – VDM?

A. see :-
"Systematic Software Development using VDM"
by CLIFF JONES,
PHI 1986 (i.e. soon!?)

"<u>VDM</u> is a specification technique which is formal <u>enough</u> to support implementation justification."

Cliff Jones.

Character:

- <u>model</u>-orientated;
- techniques <u>motivated</u> by computing applications not (just) mathematical elegance.

- built-in types as, 1st-level, building blocks — <u>sets</u>, <u>maps</u>, <u>sequences</u>, <u>tagged</u> <u>products</u>, recursive <u>tree</u> structures.
- operations defined w.r.t. a <u>state</u> via <u>pre/post-conditions</u>.
- <u>non-determinism</u> handled by <u>relational</u> post-conditions.

e.g. "A compiler dictionary"

The State:

CompDict = seq of Locals

$compdict_0 \triangleq \langle\ \rangle$

Local dictionaries

The State:

Locals = map Id to Attr

The Operations

$STORE_L$ - save attrib. of an id.

$ISIN_L$ - is id declared?

$LOOKUP_L$ - retrieve attrib.

$locals_0 \triangleq [\ ]$

$STORE_L\,(i : Id,\ a : Attr)$

<u>ext</u> <u>wr</u> $ld : Locals$

<u>pre</u> $i \notin \underline{dom}\ ld$

<u>post</u>

$\overleftarrow{ld} \cup [i \mapsto a] = ld$

$ISIN_L\,(i : Id)\, r : \mathbb{B}$

<u>ext</u> <u>rd</u> $ld : Locals$

<u>post</u> $i \in \underline{dom}\ ld \Leftrightarrow r$

$LOOKUP_L\,(i : Id)\, r : Attr$

<u>ext</u> <u>rd</u> $ld : Locals$

<u>pre</u> $i \in \underline{dom}\ ld$

<u>post</u> $r = ld(i)$

CompDict Operations

STORE - save attrib. of an Id.

ISLOC - is Id declared in current block

Lookup - retrieve attribs.

ENTER - start a new scope.

LEAVE - exit a scope level

STORE (i: Id, a: Attr)

<u>ext</u> <u>wr</u> cd : CompDict

<u>pre</u> $cd \neq \langle\rangle \;\wedge$
pre-STORE$_L$(i, a, <u>hd</u> cd)

<u>post</u>
$\exists ld \in Locals \cdot$
post-STORE$_L$(i, a, <u>hd</u> $\overleftarrow{cd}$, ld)
$\wedge \; \langle ld \rangle \frown \underline{tl}\ \overleftarrow{cd} = cd$

ISLOC ... exercise !

LookUp $(i : Id)\, r : Attr$

<u>ext</u> <u>rd</u> $cd : Compdict$

<u>pre</u> $\exists j \in \underline{inds}\ cd \cdot$
$\quad pre\text{-}LookUp_L(i, cd(j))$

<u>post</u>
$\quad \underline{let}\ k = \underline{mins}\ \{ j \mid pre\text{-}LookUp_L(i, cd(j)) \}$

$\quad \underline{in}\ post\text{-}LookUp_L(i, cd(k), r)$

Operations on entire Dictionaries

ENTER
<u>ext</u> <u>wr</u> $cd : CompDict$
<u>post</u> $\langle locals_0 \rangle \frown \overleftarrow{cd} = cd$

LEAVE
<u>ext</u> <u>wr</u> $cd : CompDict$
<u>pre</u> $cd \neq \langle\rangle$
<u>post</u> $\underline{tl}\ \overleftarrow{cd} = cd$

"BIAS"

A model is <u>biased</u> (w.r.t. some given set of OPs) if there exist <u>different</u> <u>elements</u> of the set of objects <u>which</u> <u>cannot</u> <u>be</u> <u>distinguished</u> by any of the operations.

IMPLEMENTATION

ABSTRACT SPEC.

DESIGN SPEC.

⋮

DESIGN SPEC.

⋮

CODE

correctness preserving proofs

Toy E.g.

A Spelling Checker

ABS. state: $A =$ set of Word

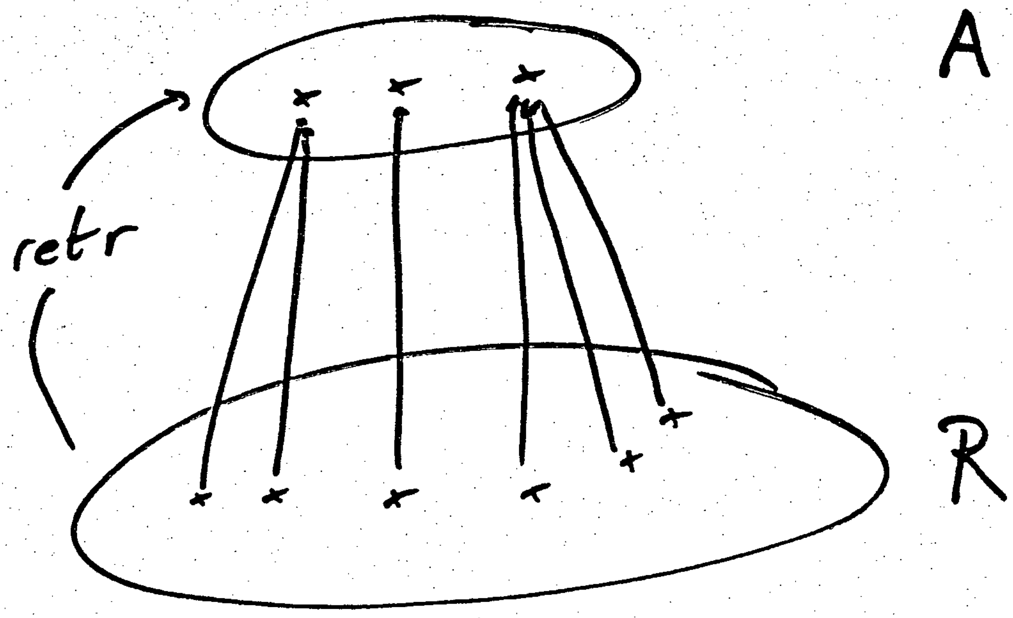
REP. state:

$R =$ seq of Word where

inv-R$(r) \triangleq$

card elems $r =$ len r

Find a 'RETRIEVE' function which is TOTAL and ONTO.



For e.g. retr-A : $R \rightarrow A$

retr-A$(r) \triangleq$ elems r

'ONTO' PROOF
(or ADEQUACY)

<u>from</u> $d \in A$

1 $\underline{elems}\ \langle\rangle = \{\}$

2 $\exists d_1 \in R \cdot retr\text{-}A(d_1) = \{\}$ $\quad$ $\exists$-I(1)

3 <u>from</u> $d \in \underline{set}\ of\ Word,\ w \notin d,\ \exists d_1 \in R \cdot retr\text{-}A(d_1) = d$

?

<u>infer</u> $\exists e_1 \in R \cdot retr\text{-}A(e_1) = d \cup \{w\}$?

<u>infer</u> $\exists d_1 \in R \cdot retr\text{-}A(d_1) = d$ $\quad$ set-ind(2,3)

3.1 <u>from</u> $d_1 \in R,\ retr\text{-}A(d_1) = d$

3.1.1 $\underline{elems}\ d_1 = d$ $\quad$ h3.1, retr-A

3.1.2 $w \notin \underline{elems}\ d_1$ $\quad$ h3, 3.1.1

3.1.3 $d_1 \frown \langle w\rangle \in R$ $\quad$ R, 3.1.2

3.1.4 $\underline{elems}\,(d_1 \frown \langle w\rangle) = \underline{elems}\ d_1 \cup \{w\}$ $\quad$ <u>elem</u>

3.1.5 $retr\text{-}A(d_1 \frown \langle w\rangle) = d \cup \{w\}$ $\quad$ 3.1.1, 3.1.4

<u>infer</u> $\exists e_1 \in R \cdot retr\text{-}A(e_1) = d \cup \{w\}$ $\quad$ $\exists$-I(3.1.5

LPF – a Logic of Partial Functions

$$x \in \underline{\mathrm{dom}}\ m \ \wedge \ m(x) = 42$$

"3-valued"

- $\wedge, \vee$ are commutative
- No "law of excluded middle"
- "Standard" Deduction theorem does not hold.

"MULE" Project

- Alan Wills, Toby Nipkow
- Cliff Jones, Ian Cottam
- Ali Yaghi, Mario Wolczko*

SERC GR/C/05762

October 1982 ≈ October 1985

*MW supported by International Computers Ltd.

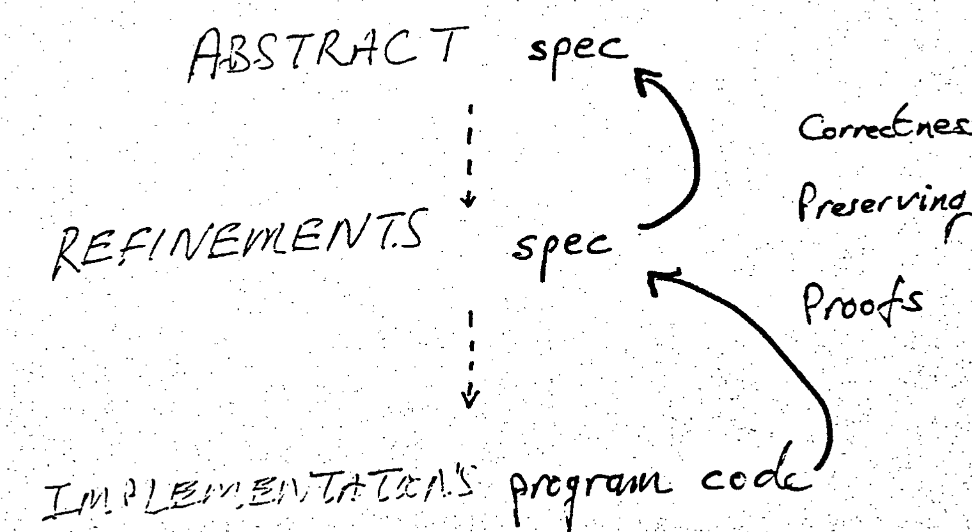
- The need for IPSE for FORMAL Methods.

- Large texts in formal languages (specs., programs, proofs, ---)

- Easy, fast, access to language processors.

- Automatic generation of texts of (unfamiliar) concrete syntax.

- Guiding the automatic generation to satisfy the development <u>method</u> (e.g. VDM)



Technical Requirements of F.M. IPSE

- as per "normal" IPSE
  - Project Database
  - Single, consistent, MMI
  - Version control
  - Project management

but ...

- special tools

e.g.

Proof assistance

- LCF
- Reve
- Boyer-Moore
- Iota
- Gypsy

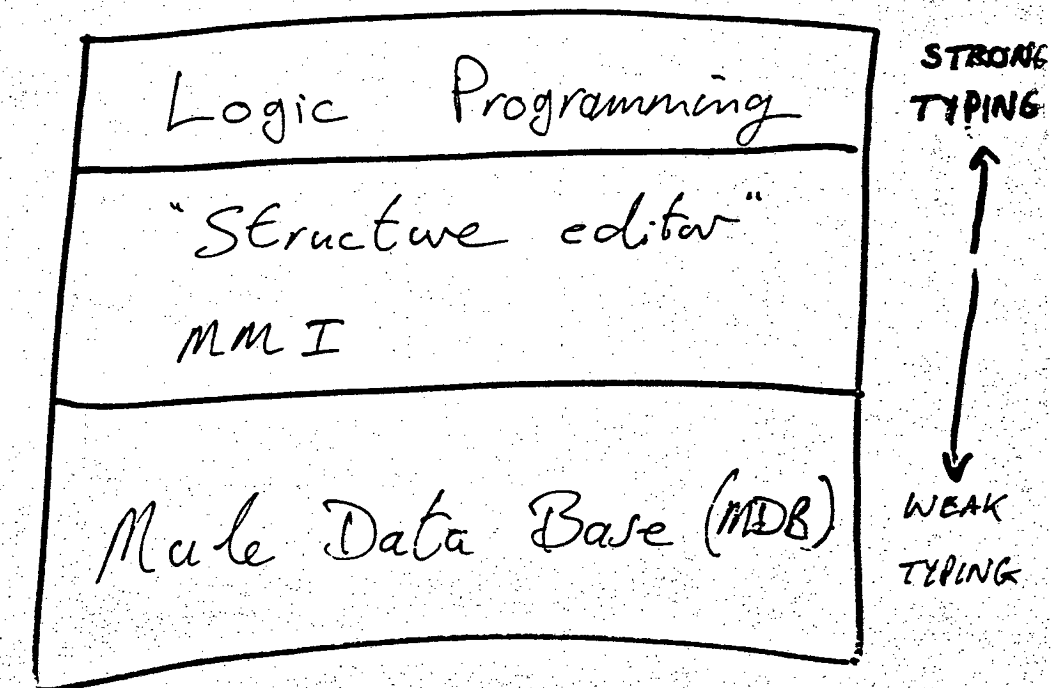
but ...

- more interconnections
- more complex consistency requirements

⟹
- "fine-grained" database
- general graph structure

MULE

– a single-user, work-station based, IPSE for F.M.



DETAILS ...

- <u>MMI</u>
- Data model requirements
- Data objects - <u>MDB</u>
- Data manipulation language - <u>GRAPL</u>
- Problems / Future work.

<u>MMI</u>

- Structure-editor paradigm
  - modifications (and navigation) to data base via syntactical units.

DDL $\propto$ Abstract syntax

- communication via Textual Projections

DB	Screen
Assign (x, 1)	$x := 1$
	$x = 1;$
	$1 \Rightarrow x$

one-many projection schemes

Not necessarily "language"

- VDM
  - formal spec
  - refinement
    - Retr.
    - Proof
    -
  - Implementation

- multiple windows
  - system ops.
  - current window ops
  - –
  - –
  - –

  } DB projection windows

- object selection via block cursors.

For each object

- Proj. scheme
  - Concrete appearance
  - Menu of ops.
  - Edit?
  - Interp. routines

Hybrid Editing

Menu Selection

Text (Parsing)

- **All** transformations invoked via SE operation
- **No** fixed order of work imposed on user

Data Model

- via choice of Abstract Syntax (care needed!)
- User edits instances (i.e. structure constrained by A.S.) ISA relations

"Problem"

Grammars <u>force</u> hierarchy.

Solutions:

× • Extra layer of abstraction "symbol tables"

✓ • Permit Directed GRAPHS (not just trees)

53

Abstract syntax for "Natural Deduction" Proofs.

Theorem :: HYP : <u>list</u> of Lexpr
PROOF: <u>list</u> of Step
CONC : Lexpr

Step = Juststep | Theorem

Juststep :: WHAT : Lexpr
WHY : Rule
HOW : <u>list</u> of Lexpr

Rule = ...

54

Proof

Juststep Theorem

Extend--Front Extend--End

from $E1, E2$

1 \$Proof

infer $E1 \land E2$

from E1, E2

  from E1, E2

    1.1 \$Proof

  1 **infer** $\sim(\sim E1 \vee \sim E2)$

infer $E1 \wedge E2$ 1, $\wedge$-Defn

from E1, E2

  from E1, E2

    1.1 $\sim\sim E1$ $\sim\sim$-I, E1

    1.2 $\sim\sim E2$ $\sim\sim$-I, E2

  1 **infer** $\sim(\sim E1 \vee \sim E2)$ $\sim\vee$-I, 1.1, 1.2

infer $E1 \wedge E2$ 1, $\wedge$-Defn