Q. What is the Vienna Development Method - VDM?

A. see: - "Systematic Software Development using VDM" by CLIFF JONES, PHI 1986 (i.e. soon!?)
VDM is a specification technique which is formal enough to support implementation justification. " Cliff Jones.

Character:
- model-orientated;
- techniques motivated by computing applications not (just) mathematical elegance.

- built-in types as, 1st-level, building blocks - sets, maps, sequences, tagged products, recursive tree structures.
- operations defined w.r.t. a state via pre/post-conditions.
- non-determinism handled by relational post-conditions.
e.g. "A computer dictionary."

**The State:**

\[
\text{CompDict} = \text{seq of Locals}
\]

\[
\text{compdict}_0 \triangleq \left[ \right]
\]

**Local dictionaries**

**The State:**

\[
\text{Locals} = \text{map Id to Attr}
\]

**The Operations**

\[
\text{STORE}_L - \text{save attrib. of an id}
\]

\[
\text{IS IN}_L - \text{is id declared?}
\]

\[
\text{Lookup}_L - \text{retrieve attrib.}
\]

\[
\text{locals}_0 \triangleq \left[ \right]
\]
\[
\text{STORE}_L (i: Id, a: Attr) \\
\text{ext wr \, ld: Locals} \\
\text{pre \, } i \notin \text{dom \, ld} \\
\text{post} \\
\overline{\text{ld} \cup [i \mapsto a] = ld}
\]

\[
\text{ISIN}_L (i: Id) r: B \\
\text{ext \, rd \, ld: Locals} \\
\text{post} \, i \in \text{dom \, ld} \iff r
\]

\[
\text{Lookup}_L (i: Id) r: Attr \\
\text{ext \, rd \, ld: Locals} \\
\text{pre} \, i \in \text{dom \, ld} \\
\text{post} \, r = \text{ld}(i)
\]
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)
\[\text{ext wr cd: CompDict}\]
\[\text{pre cd \neq <>} \land \text{pre-STORE_E} (i, a, hd cd)\]
\[\text{post \exists ld \in Locals \cdot post-STORE_E} (i, a, hd cd, ld)\]
\[\land <ld>^\text{top cd} = cd\]
ISLOC ... exercise!

Operations on entire Dictionaries

Lookup \((x : Id)_r : Attr\)

\[\text{ext } rd \text{ cd } : \text{Compdict}\]

\[\text{pre } E \text{ je ind } \text{ cd} \cdot \]

\[\text{pre-lookup}_{L}(x, \text{cd}(j))\]

\[\text{post}\]

\[\text{let } k = \text{mins} \{ j \mid \text{pre-lookup}_{L}(x, \text{cd}(j)) \}\]

\[\text{in } \text{post-lookup}_{L}(x, \text{cd}(k), r)\]

ENTER

\[\text{ext wr } \text{cd} : \text{CompDict}\]

\[\text{post } <\text{locals}>^\text{cd} = \text{cd}\]

LEAVE

\[\text{ext wr } \text{cd} : \text{CompDict}\]

\[\text{pre } \text{cd} \neq <>\]

\[\text{post } \text{et } \overline{\text{cd}} = \text{cd}\]
"BIAS"

A model is biased (w.r.t. some given set of OPs) if there exist different elements of the set of objects which cannot be distinguished by any of the operations.
**Toy E.g.: A Spelling Checker**

**ABS. state:** \( A = \text{set of Word} \)

**REP. state:**

\[ R = \text{seq of Word where } \]

\[ \text{inv-}R(x) \uparrow \]

\[ \text{card elems } r = \text{len } r \]

Find a 'RETRIEVE' function which is **TOTAL** and **ONTOT**.

For e.g. \( \text{retr-A : } R \rightarrow A \)

\[ \text{retr-A}(x) \uparrow \text{elems } r \]
3.1 \text{from } d \in R, \text{retr-}A(d_i) = d

\begin{align*}
3.1.1 \quad & \text{elems } d_1 = d & h3.1, \text{retr-}A \\
3.1.2 \quad & w \notin \text{elems } d_1 & h3, 3.1.1 \\
3.1.3 \quad & d_1 \wedge <w> \in R & R, 3.1.2 \\
3.1.4 \quad & \text{elems } (d_1 \wedge <w>) = \text{elems } d_1 \cup \{w\} & \text{elem} \\
3.1.5 \quad & \text{retr-}A(d_1 \wedge <w>) = d \cup \{w\} & 3.1.1, 3.1.4
\end{align*}

\begin{align*}
\text{from } d \in A \\
\text{elems } <> = \{\}
\end{align*}

\begin{align*}
\text{from } d \in \text{set of word}, w \notin d, \\
\text{elems } d_1 = d
\end{align*}

\begin{align*}
\text{infer } \exists e_1 \in R \cdot \text{retr-}A(e_1) = d \cup \{w\} & \text{set-ind}(2,3) \\
\text{infer } \exists d_1 \in R \cdot \text{retr-}A(d_1) = d & \text{set-ind}(2,3) \\
\text{infer } \exists e_1 \in R \cdot \text{retr-}A(e_1) = d \cup \{w\} & \text{set-ind}(3.15)
\end{align*}
LPF - A logic of Partial Functions "MULE" Project

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

\[ x \in \text{dom } m \land m(x) = 42 \]

"3-valued"

- \( \wedge, \lor \) are commutative
- No "law of excluded middle"
- "Standard" Deduction theorem does not hold.

SERC GR/C/05762
October 1982 - October 1985

* MW supported by International Computers Ltd.
- Automatic generation of texts of (unfamiliar) concrete syntax.
- Easy, fast, access to language processors.
- Large texts in formal languages (specs, proofs).
- Guiding the automatic generation to satisfy the development method (e.g. TLA).
- The need for IPSE for formal methods.

IMPLEMENTATION

Abstract spec

Refinements

Concretes

Proofs
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
    - Gipsy
but ...

- more interconnections
- more complex consistency requirements

⇒

- “fine-grained” database
- general graph structure

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

```
Logic Programming

"Structure editor"

MMI

Mule Data Base (MDB)
```
DETAILS ...

- MMI
- Data model requirements
- Data objects - MDB
- Data manipulation language - GRAFL
- Problems / Future work.

MMI
- Structure-editor paradigm
  - modifications (and navigation)
  to database via syntactical units.

DDL = Abstract syntax
  - communication via
  Textual Projections
Not necessarily "language"

DB

Assign

$x \leftarrow 1$

$\rightarrow x := 1$

$x = 1$

$\rightarrow 1 \mapsto x$

one-many projection schemes

Screen

* 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

VDM

formal spec

refinement

Implementation

Retr. Prof.
- Hybrid Editing
  - Menu Selection
  - Text (Parsing)

- Data Model
  - Via choice of Abstract Syntax (core needed!)

- User edits instances
  - (i.e. structure constrained by AS, ISA relations)

- All transformations invoked via SE operation

- No fixed order of work imposed on user
```

"Problem"
Grammars force hierarchy.

Solutions:
- Extra layer of abstraction "symbol tables"
- Permit Directed Graphs (not just trees)

Abstract syntax for "Natural Deduction" Proofs.

Theorem :: HYP : list of Lexpr
          PROOF : list of Step
          CONC : Lexpr

Step = Juststep / Theorem

Juststep :: WHAT : Lexpr
          WHY : Rule
          HOW : list of Lexpr

Rule = ...

```
Proof

Just step Theorem

\[ \text{Extend-Front} \quad \text{Extend-End} \]

\[
\text{proof} \quad E_1, E_2 \\
\text{infer} \quad E_1 \land E_2
\]
\[
\begin{array}{c}
\text{from} \quad E_1, E_2 \\
\begin{array}{c}
\text{from} \quad E_1, E_2 \\
1.1 \quad \text{\$proof} \\
1 \quad \text{infer} \quad \neg (E_1 \lor \neg E_2)
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\text{infer} \quad E_1 \land E_2 \\
1, \land\text{-Def}
\end{array}
\]

\[
\begin{array}{c}
\text{from} \quad E_1, E_2 \\
\begin{array}{c}
\text{from} \quad E_1, E_2 \\
1.1 \quad \neg \neg E_1 \quad \neg \neg I, E_1 \\
1.2 \quad \neg \neg E_2 \quad \neg \neg I, E_2 \\
1 \quad \text{infer} \quad \neg (\neg E_1 \lor \neg E_2) \quad \neg \neg I, 1.1, 1.2
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\text{infer} \quad E_1 \land E_2 \\
1, \land\text{-Def}
\end{array}
\]