BCS - FACS
85. 12. 17

Ian Cottam
Lecture 2
Graph + Rapid Prototyping
Data objects - MDB

Model

BRM

Directed Graphs

General query criteria

Specific application constrained queries
MDB supports:

- Labelled, directed, graphs
  - nodes/arcs labelled by entities
  - nodes may contain values
  - one-one (nodes, entities)
  - many-one (nodes, values)
  - many-one (arcs, entities)

'Node' and 'Entity' are synonymous.

- Lists
- Sets
- Value maps
DATA MANIPULATION LANGUAGE

"GRAPL"

Experimental

VHLL for database (MDE)

Query and Update

- Prolog
- Typol (Mender)
• support activities beyond the context-free
  - type checking
  - formula transformation
  - proof checkers

Q. Why Prolog-like?
need...

- logical inferences on MBD
- (subgraph) pattern-match
- symbolic manipulations
- ability to handle incomplete data (e.g., syntax graph with "holes")

[+ not much time to design/ "re-invent"]
Graph • Prolog

• almost identical ("to our surprise")
• all Graph predicates are typed.
• concept type inheritance.
• several trivial details.
person :: A: Age, N: Name;
student :: person + C: Dept, U: Univ

young (person (?AGE,-))

?- young (student(3,-,-,-,-)).

yes
ABSTRACT SYNTAX

Formula

Formula :: TERM:: Term;
Term :: EXPR:: Expr;
Expr = vari|true|false|undef|
or|neg|impl|turnstile;

vari :: VAR:: Id;
true :: ; false :: ; undef :: ;
or :: 01:: Expr, 02:: Expr;
and :: A1:: Expr, A2:: Expr;
impl :: A:: Expr, C:: Expr;
neg :: NOT:: Expr;
turnstile :: LHS:: Expr, RHS:: Expr;
( * Graph prog. A.S. *)

check :: T: Term;
copy :: INE: Expr, INB: List,
      OUTE: Expr, OUTE: List;
taut :: E: Expr, L: List;
 falsify :: E: Expr, L: List;
evau :: E: Expr, V: Expr;
isin :: I: Id, V: Expr, L: List;
writeVal :: L: List;

Triple :: I: Id, V: Expr, L: List
List :: Triple | nil;
      nil :: ;
Prolog 3:

\[
\text{check ( Term(?E) ) } \leftarrow \\
\text{copy (?E, nil, ?C )} \\
\text{taut (?C, ?B) .}
\]

\[
\text{check ATTACH TO Term WITH "Tautology check" .}
\]
taut (?E, ?B) <=
not (falsify (?E, ?B)),
write (<'TAUTOLOGY!' >),
writeln.

falsify (?E, ?B) <=
eval (?E, false),
write (<'False with valuation' >),
writeln.
write Eval (?B).

(* ditto for eval (?E, undef) *)
eval (true, e).

 eval (false, false).

 eval (undef, undef).

 eval (vri (?v), ?v).

 eval (and (?E1, -), false) <=
 eval (?E1, false).

 eval (and (-, ?E2), false) <=
 eval (?E2, false).

 eval (and (?E1, -), undef) <=
 eval (?E1, undef).

 eval (and (?E1, ?E2), undef) <=
 eval (?E2, undef).

 eval (and (?E1, ?E2), true) <=
 eval (?E1, true), eval (?E2, true).
(* ditto *)

\[
eval \begin{cases} & \text{or} \\
& \text{impl} \\
& \text{turnstile} \\
& \text{neg}
\end{cases}
\]
copy (var(?I), ?ENV, var(?V), ?ENV)
<= isin (?I, ?V, ?ENV), !.
copy (var(?I), ?ENV, var(?V),
   C triple (?I, ?V, ?ENV))
<= copy (?E1, ?O1, ?C1, ?O1),
(* ditto for copy of or, impl,
   turastle and neg *)
\text{isin}(?I, \text{\texttt{\textbar triple}(?I, ?V, -)})\). \\
\text{isin}(?I, ?V, \text{\texttt{\textbar triple}(-, -, ?L)}) \leftarrow \\
\text{isin}(?I, ?V, ?L). \\

\text{writeVal}(\text{\texttt{\textbar triple}(?I, ?V, ?L)}, \\
\text{write}(\langle ?I, '=' , ?V \rangle), \\
\text{writeln}, \\
\text{writeVal}(?L). \\

\text{writeVal}(\text{\texttt{\textbar}})
Limitations of Current Ver.

- cannot update database.
  - need to assert/retract

- sub-typing ("inheritance")
  is too slow.
  - need smart implementation

- all Graph output goes to a monitor window (not current)
  - needs fixing.
Rapid Prototyping of VDM Specifications.
Why?

- early customer feedback:
  - would this functionality satisfy your requirements?

- helps specifier/designer:
  - is spec. consistent, computable, etc.? (primitive theorem proving assistance)
DIFFER!

- Encourages test-case approach to correctness.

Management reluctant to have staff "wasting time" proving theorems if Spec/Prototype passes all Q.A.'s tests.
Credits: IDC [redacted] White + Shirley McAteer

Goal
The automatic (or, at least semi-automatic) generation of a prototype implementation from a formal spec.

(N.B. not possible for all VDM specs.)
PROBLEMS

• \( \text{past} \)
  \[ \exists i \in \mathbb{N} \cdot \]
  \[ \neg \exists j \in \mathbb{N} \cdot j > 0 \]
  (* there is a largest value in the set of naturals *)

• \( \forall i \in \mathbb{N} \cdot \text{even}(i) \lor \overline{\text{odd}(i)} \)
  \[ \text{univ. quant. over infinite set.} \]
Fundamental Requirement

A person writing an abstract formal spec. in VDM should not have to know about or be concerned about the prototyping process.

⇒ better for prototype generation to fail than spec. is "infected".
VDM spec.

"Translator"

LIB. of Prolog defs.
for VDM ops
- sets
- bags
- maps
- sequences
- etc.

Prolog program

Prolog Interpreter
**Implicit defn. of seq. concat.**

**VDM:** concat: seq of X × seq of X → seq of X

**post-concat** \((e_1, e_2, r)\)
\[
\begin{align*}
\text{len } r &= \text{len } e_1 + \text{len } e_2 \\
\text{r}(1 \ldots \text{len } e_1) &= e_1 \\
\text{r}(\text{len } e_1 + 1 \ldots \text{len } r) &= e_2
\end{align*}
\]

**Prolog:**
\[
\text{concat} \((L_1, L_2, R)\) :-
\begin{align*}
\text{len}(L_1, \text{LEN}_L), \text{len}(L_2, \text{LEN}_L2), \\
\text{LEN}_R &= \text{LEN}_L + \text{LEN}_L2, \text{len}(R, \text{LEN}_R) \\
\text{subl}(R, 1, \text{LEN}_L, \text{RF}), \text{RF} &= L_1, \\
\text{mid} &= \text{LEN}_L + 1, \text{subl}(R, \text{mid}, \text{LEN}_R, \text{RB}), \\
\text{RB} &= L_2.
\end{align*}
\]
<table>
<thead>
<tr>
<th>VDM</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>sets</td>
<td>lists (unique)</td>
</tr>
<tr>
<td>sequences</td>
<td>lists</td>
</tr>
<tr>
<td>maps</td>
<td>lists of 2-tuples</td>
</tr>
<tr>
<td>bags</td>
<td>(unique 2nd el)</td>
</tr>
<tr>
<td>State</td>
<td>lists of 2-lists</td>
</tr>
<tr>
<td>Variables</td>
<td>(&quot;map x to N&quot;)</td>
</tr>
</tbody>
</table>
VDM post-conditions

\[ \text{post\_op}(\ldots) \rightarrow \text{state}(\text{Ocd}) \]

translated post-cond

\[ \text{translated\ post\-cond}(\text{post}(\text{Ocd}, \ldots, \text{New})) \]

react \[ (\text{state}(\_)) \]

asserta \[ (\text{state}(\text{New})) \]
c.q. The **TEST** from the **Equivalence Relation** problem.

**VDM:** Part = set of set of Pno

\[
\text{Test} (p_1, p_2 : \text{Pno}) \vdash \text{DB} \\
\text{ext } \text{rd } p : \text{Part} \\
\text{post } r \iff \exists s \in \overline{p}. p1 \in s \land p2 \in s
\]

**Prolog:**
\[
\text{Test} (p_1, p_2) : - \\
\text{state}(s), \text{member}(s, p), \\
\text{member}(p_1, s), \text{member}(p_2, s).
\]

(* No state change *)
The EQUATE op.

VDM

\[
\text{EQUATE } (P_1, P_2 : \text{Pno}) \\
\text{ext wr } P : \text{Part} \\
\text{post } P = \\
\{ s \in P | P_1 \epsilon S \land P_2 \epsilon S \} \\
\cup \{ \bigcup \{ s \in P | P_1 \epsilon S \lor P_2 \epsilon S \} \}
\]
Current Implementation

- Semi-automatic translation, based on a VDM-editor generated via the C.N.E. ALOEGEN program.

A.S.

Unparsing \( \triangle \) Prolog
Schemes
Future Plans

- Mule-based system
  - fully automatic
  - action routines in
    C after producing

- General investigation of
  Rapid Prototyping
  within IPSE 2.5 Alvey
  project.
Mule:

- single-user, workstation-based, "IPSE"

- ICL / Perg Systems PERQ I
  - UNIX
  - Pascal (UNIX-Portable)

- VDM version
Major Influences:

- Gandalf  
  CMU (USA)

- Mentor  
  INRIA (France)

- PSG  
  Darmstadt (Germany)

+ "Workstation
corporate concepts  
Xerox PARC (USA)
Related Work

Zs. Farkas, P. Szeredi, and E. Santane-Toth,
"LDM - a program specification support system."
Proc. 1st Int. Conf. on Logie Prog.
Marseille, 1982.
A. How to make graph links?
- MMI pointing device
- "Dynamic syntax" for text I/P
- Explicit action routine in some DBL.
\[ E_1 \xrightarrow{E_2} E_3 \]

\[ E_1 \xrightarrow{E_2} S = \{ F_1, \ldots, F_n \} \]

\[ E_1 \xrightarrow{E_2} L = \{ F_1, \ldots, F_n \} \]

\[ E_v \xrightarrow{VS} \{ F_1, \ldots, F_n \} \]