VDM

(Vienna Development Method)

Affiliation:

Andreas Müller – 0555284

Bachelor Presentation – “Technical Mathematics”

Johannes Kepler Universität – Linz, Austria

e-mail: a_m@gmx.at

VDM

- Collection of techniques for
  - modeling
  - specification
  - and design

  of computer based systems

- Origins: IBM laboratories in Vienna

- VDM-SL standardized

Content

History

- 3 phases
  - Origin of VDM (1970s)
  - Rigorous Proofs (1980s and 90s)
  - Formalisation: Tools support

Origin of VDM

- Roots: programming language definition
  - definition of PL/I
- Proofs
  - 1968: Peter Lucas – equivalence of programming language concepts
- 1975: dispersal of the group
  - Different approaches

Rigorous specification and proof

- 1980s: from definition language to development ‘method’
- Standardization process gathered momentum
  - VDM symposia →FME → FM Symposia
- VDM-SL: emphasis on implicit style of operation specification

Example – „biased queue“

$$Queueb :: s : Qel^* \\ \quad i : \mathbb{N}$$

where

$$inv\text{-}Queueb(mk\text{-}Queueb(s,i)) \quad \triangleq \quad i \leq \mathbf{len}\, s$$

$ENQUEUE\ (e: Qel)$

$\mathbf{ext\ wr}\ s\ :\ Qel^*$

$\mathbf{post}\ s = \overleftarrow{s} \frown [e]$

$DEQUEUE\ ()\ e: Qel$

$\mathbf{ext\ rd}\ s\ :\ Qel^*$

$\quad \mathbf{wr}\ i\ :\ \mathbb{N}$

$\mathbf{pre}\ i < \mathbf{len}\, s$

$\mathbf{post}\ i = \overleftarrow{i} + 1 \wedge e = \overleftarrow{s}\,(i)$

taken from
Dines Björner, Matrin C. Henson -
“Logics of Specification Languages”

Proof Obligations

- Invariants, preconditions, post-conditions
  - Arbitrarily complex logical expressions
  - In general: model may not be correct

  →Proof Obligations

- Satisfiability Obligation

$$\forall\ \overleftarrow{qb} \in \mathtt{Queueb}\ .\ \mathtt{pre-DEQUEUE}\ (\overleftarrow{qb})\ \Rightarrow$$
$$\exists\ \mathtt{qb} \in \mathtt{Queueb}, \mathtt{e} \in \mathtt{Qel}\ .\ \mathtt{post-DEQUEUE}\left(\overleftarrow{qb},\ \mathtt{qb},\ \mathtt{e}\right)$$

Rigorous Proof

from	$\overleftarrow{qb} \in Queueb \,.\, pre\text{-}DEQUEUE\,(\overleftarrow{qb})$	
1	**let** $i = \overleftarrow{i} + 1$	
2	**let** $qb = mk\text{-}Queueb\,(\overleftarrow{s}, i)$	
3	$\overleftarrow{i} <$ **len** $\overleftarrow{s}$	h2
4	$i \leq$ **len** $\overleftarrow{s}$	$\mathbb{N}$ / 3 / 1
5	$inv\text{-}Queueb\,(qb)$	4 / 2 / $inv\text{-}Queueb$
6	$qb \in Queueb$	5 / $Queueb$
7	**let** $e = \overleftarrow{s}\,(i)$	
8	$e \in Qel$	7 / 4 / **len**
9	$i = \overleftarrow{i} + 1 \bigwedge e = \overleftarrow{s}\,(i)$	$\bigwedge$-I (1, 7)
10	$post\text{-}DEQUEUE\left(\overleftarrow{qb}, qb, e\right)$	$post\text{-}DEQUEUE$ (9)
infer	$\exists\, qb \in Queueb, e \in Qel \,.\, post\text{-}DEQUEUE\left(\overleftarrow{qb}, qb, e\right)$	$\exists$-I (6, 8, 10)

Rules of inference

- Proof is not formal
  - Couldn‘t be checked by a machine
- Justifications
  - Data type properties
  - Symbols defined elsewhere
  - Rules of inference

$$\boxed{\exists - \mathbf{I}}\ \frac{s \in X ; E(s/x)}{\exists x \in X \,.\, E(x)} \qquad \boxed{\wedge - \mathbf{I}}\ \frac{E_1 ; \ldots ; E_n}{E_1 \wedge \ldots \wedge E_n}$$

Formalization

- Work on tool support
  - Prolog-based animation of a VDM model
  - SpecBox
    - Syntax checking
    - Basig semantic checking
  - VDM Toolbox
  - VDMTools
    - Most robust of tools for VDM-SL

Tools support

- Tool development & industrial engagement
  - Different capabilities
- Afrodite project
  - Object-oriented extensions
  - Real-time extensions

  →VDM++
- 2004: VDMTools sold to CSK (Japan)
  - Develop and promote the toolset

Content

1. Historical context
2. VDM
3. A proof framework for VDM (mural)
4. VDMTools

VDM

- Functions
- Operators
- Set notations
- Composite objects
- Invariants

Functions

- Defining explicit functions
  - Example

    $$square : \mathbb{Z} \to \mathbb{N}$$
    $$square(i) \quad \underline{\triangle} \quad i * i$$

  - Conditions

    $$abs : \mathbb{Z} \to \mathbb{N}$$
    $$abs(i) \quad \underline{\triangle} \quad \textbf{if } i < 0 \textbf{ then } \Leftrightarrow i \textbf{ else } i$$

  - Usage of “**let**”

    $$absprod : \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}$$
    $$absprod(i, j) \quad \underline{\triangle}$$
    $$\quad \textbf{let } k = i * j \textbf{ in}$$
    $$\quad \textbf{if } k < 0 \textbf{ then } \Leftrightarrow k \textbf{ else } k$$

Functions

- Implicit definition
  - What is to be computed (not how)

- Maximum function: $maxs(\{3,7,1\}) = 7$

- Specification: $maxs\ (s\colon \mathbb{N}\text{-}\mathbf{set})\ r\colon \mathbb{N}$
  $\mathbf{pre}\ s \neq \{\,\}$
  $\mathbf{post}\ r \in s \wedge \forall i \in s \cdot i \leq r$

Functions

- Pre-condition: $\mathit{pre\text{-}maxs}\colon \mathbb{N}\text{-}\mathbf{set} \rightarrow \mathbb{B}$
  – Assumptions about arguments
  – Partial function

- Post-condition: $\mathit{post\text{-}maxs}\colon \mathbb{N}\text{-}\mathbf{set} \times \mathbb{N} \rightarrow \mathbb{B}$

- Notice that: $\mathit{pre\text{-}maxs}(\{3,7,1\}) \;\Leftrightarrow\; \mathbf{true}$

  $\mathit{post\text{-}maxs}(\{3,7,1\},7) \;\Leftrightarrow\; \mathbf{true}$

Meaning of implicit specification

- Informally, such a specification requires that, to be correct with respect to the specification, a function must - when applied to arguments (of the right type) which satisfy the pre-condition - yield a result (of the right type) which satisfies the post-condition.

Implicit functions

- Each implementation may yield another result

$$
\begin{aligned}
& arbs\ (s\colon \mathbb{N}\text{-}\mathbf{set})\ r\colon \mathbb{N} \\
& \mathbf{pre}\ s \neq \{\,\} \\
& \mathbf{post}\ r \in s
\end{aligned}
$$

- Quantifiers
  - Avoid recursion required in direct definition

$$
\begin{aligned}
& gcd\ (i\colon \mathbb{N}_1, j\colon \mathbb{N}_1)\ r\colon \mathbb{N}_1 \\
& \mathbf{pre\ true} \\
& \mathbf{post}\ is\text{-}common\text{-}divisor(i, j, r) \wedge \\
& \qquad \neg\, \exists s \in \mathbb{N}_1 \cdot is\text{-}common\text{-}divisor(i, j, s) \wedge s > r
\end{aligned}
$$

Advantages

- direct description of (multiple) properties which are of interest to the user
- characterizing a set of possible results by a post-condition
- explicit record (by Boolean expression) of the pre-condition
- less commitment to a specific algorithm
- provision of a name for the result

Operations

- Implicit specification
- Functions: fixed mapping
  - Input → Output
  - double(2) = 4
- Operations: hidden state to record values
  - e.g.: Subsequent sum
  - **sum(2) = 2**, sum(99) = 101, **sum(2) = 103**,…

Example: Calculator (1)

- Load operation

$$
\begin{array}{l}
LOAD\ (i{:}\,\mathbb{N}) \\
\textbf{ext wr } reg\ :\ \mathbb{N} \\
\textbf{post } reg = i
\end{array}
$$

- Convention: CAPITAL letters
- First line: similar to functions

Example: Calculator (2)

- Load operation

$$
\begin{array}{l}
LOAD\ (i{:}\,\mathbb{N}) \\
\textbf{ext}\ \ \textbf{wr}\ \ reg\ :\ \mathbb{N} \\
\textbf{post}\ \ reg = i
\end{array}
$$

- Second part:
  - External access (ext): read (rd) or read/write (wr)
  - Name followed by type
- Post-condition:
  - Truth valued function (parameter, ext. variables)

Example: Calculator (3)

- Show operation

$$\begin{array}{l} SHOW\ ()\ r\colon \mathbb{N} \\ \textbf{ext rd}\ reg\ :\ \mathbb{N} \\ \textbf{post}\ r = \overleftarrow{reg} \end{array}$$

- Alternative definitions

$$\begin{array}{l} SHOW\ ()\ r\colon \mathbb{N} \\ \textbf{ext rd}\ reg\ :\ \mathbb{N} \\ \textbf{post}\ r = reg \end{array}$$

$$\begin{array}{l} SHOW\ ()\ r\colon \mathbb{N} \\ \textbf{ext wr}\ reg\ :\ \mathbb{N} \\ \textbf{post}\ reg = \overleftarrow{reg} \land r = reg \end{array}$$

Example: Calculator (4)

- Preconditions
  - Omitted preconditions are assumed to be **true**

- Divide operation

$$
\begin{array}{l}
\mathit{DIVIDE}\ (d\colon \mathbb{N})\ r\colon \mathbb{N} \\
\mathbf{ext\ wr}\ \mathit{reg}\ :\ \mathbb{N} \\
\mathbf{pre}\ d \neq 0 \\
\mathbf{post}\ d * r + \mathit{reg} = \overleftarrow{\mathit{reg}} \wedge \mathit{reg} < d
\end{array}
$$

Set notations

- Set
  - Unordered collection of distinct objects
  - Values marked by braces e.g. {a,b}

- Forming of sets
  - Enumeration of their elements
  - Set comprehension $\{i \in \mathbb{Z} \mid 1 \leq i \leq 3\} = \{1, 2, 3\}$
  - Intervals $\{i, \ldots, k\} = \{j \in \mathbb{Z} \mid i \leq j \leq k\}$
  - Empty set {}

Set notations

- The “-set” constructor
  - Applied to a known set
  - Yields a set of sets
    - For finite sets: power set
- Cardinality: **card** operator
  - **card** {} = 0

Partitions

- Set is partitioned $\Leftrightarrow$ split into disjoint subsets

$$Partition = \{\, p \in (\mathbb{N}\text{-}\mathbf{set})\text{-}\mathbf{set} \mid inv\text{-}Partition(p) \,\}$$

$$inv\text{-}Partition : (\mathbb{N}\text{-}\mathbf{set})\text{-}\mathbf{set} \to \mathbb{B}$$
$$inv\text{-}Partition(p) \quad \triangleq \quad is\text{-}prdisj(p) \wedge \{\,\} \notin p$$

$$is\text{-}prdisj : (\mathbb{N}\text{-}\mathbf{set})\text{-}\mathbf{set} \to \mathbb{B}$$
$$is\text{-}prdisj(ss) \quad \triangleq \quad \forall s_1, s_2 \in ss \cdot s_1 = s_2 \vee is\text{-}disj(s_1, s_2)$$

- Example

$$\{p_a, p_b\} \subseteq Partition$$
$$p_a = \{\{1\}, \{2\}\}$$
$$p_b = \{\{1, 2\}\}$$

Composite Objects

- make-functions
  - Given appropriate values for fields
  - Yields value of composite type
- Example: *Datec*

$$mk\text{-}Datec\colon \{1,\ldots,366\} \times \mathbb{N} \to Datec$$

- Types are disjoint

$$mk\text{-}Fahrenheit\colon \mathbb{R} \to Fahrenheit$$
$$mk\text{-}Celsius\colon \mathbb{R} \to Celsius$$

Decomposing Objects

- Selectors
  - Functions to yield component values
  - Signature:

$$day\colon Datec \to \{1,\ldots,366\}$$
$$year\colon Datec \to \mathbb{N}$$

  - e.g.

$$day(mk\text{-}Datec(7, 1979)) = 7$$
$$year(mk\text{-}Datec(117, 1989)) = 1989$$

- Several other ways of decomposing objects

Decomposing Objects

- Notation for defining local values

$$\textbf{let } i = \cdots \textbf{ in } \cdots i \cdots$$

- Using selectors

$$inv\text{-}Datec : Datec \to \mathbb{B}$$
$$inv\text{-}Datec(dt) \quad \triangleq \quad is\text{-}leapyr(year(dt)) \lor day(dt) \leq 365$$

- Using extension of **let**

$$inv\text{-}Datec(dt) \quad \triangleq$$
$$\textbf{let } mk\text{-}Datec(d, y) = dt \textbf{ in } is\text{-}leapyr(y) \lor d \leq 365$$

- Shorter

$$inv\text{-}Datec(mk\text{-}Datec(d, y)) \quad \triangleq \quad is\text{-}leapyr(y) \lor d \leq 365$$

Decomposing Objects

- Case construct
  - If -then-else-Notation

$$
\begin{aligned}
&\mathit{norm\text{-}temp} : (\mathit{Fahrenheit} \cup \mathit{Celsius}) \to \mathit{Celsius}\\
&\mathit{norm\text{-}temp}(t) \quad \triangleq \quad \textbf{if } t \in \mathit{Fahrenheit}\\
&\qquad\qquad\qquad\quad\ \ \textbf{then let } \mathit{mk\text{-}Fahrenheit}(v) = t \ \textbf{ in } \mathit{mk\text{-}Celsius}((v \Leftrightarrow 32) * 5/9)\\
&\qquad\qquad\qquad\quad\ \ \textbf{else } t
\end{aligned}
$$

  - “cases”-Notation

$$
\begin{aligned}
\mathit{norm\text{-}temp}(t) \quad \triangleq \quad & \textbf{cases } t \textbf{ of}\\
& \mathit{mk\text{-}Fahrenheit}(v) \to \mathit{mk\text{-}Celsius}((v \Leftrightarrow 32) * 5/9),\\
& \mathit{mk\text{-}Celsius}(v) \qquad \to t\\
& \textbf{end}
\end{aligned}
$$

Defining composite types

- Definition of a composite type

$$
\begin{array}{l}
\textbf{compose } Datec \textbf{ of} \\
\qquad \begin{array}{lcl} day & : & \{1, \ldots, 366\}, \\ year & : & \mathbb{N} \end{array} \\
\textbf{end}
\end{array}
$$

  - Name of composite object
    - compose *[name]* of
  - Variable number of fields
    - [field-name]: [field-type],
  - end

Defining composite types

- If a value is never decomposed by a selector

$$\textbf{compose}\ Celsius\ \textbf{of}\ v : \mathbb{R}\ \textbf{end} \rightarrow \textbf{compose}\ Celsius\ \textbf{of}\ \mathbb{R}\ \textbf{end}$$

- 'is composed of'

$$Name :: \cdots$$
$$Name = \textbf{compose}\ Name\ \textbf{of}\ \cdots\ \textbf{end}$$

- Names for types

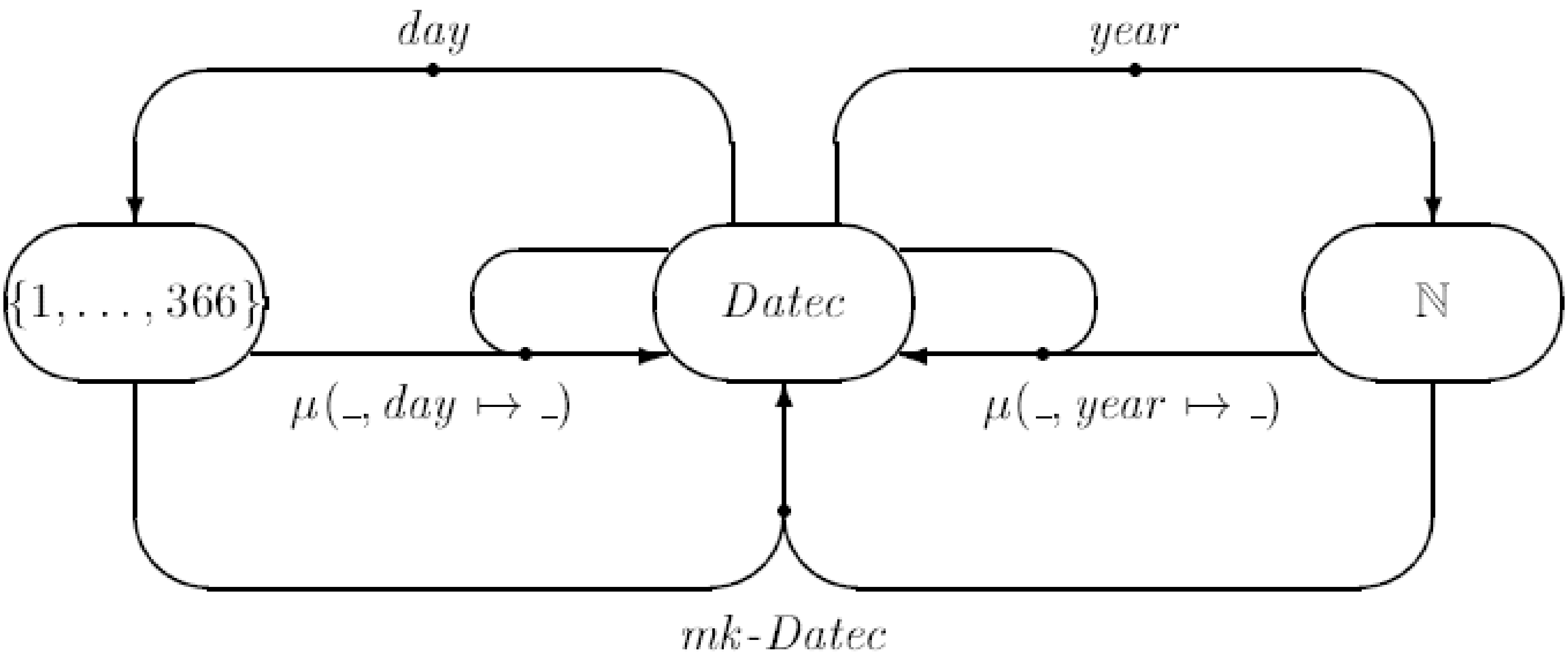
$$\begin{aligned} Datec :: day &: Day \\ year &: Year \end{aligned} \qquad \begin{aligned} Day &= \{1, \ldots, 366\} \\ Year &= \mathbb{N} \end{aligned}$$

Modifying composite objects

- The μ function
  - Create composite values which differ only in one field
  - e.g.

$$
\begin{aligned}
&dt = mk\text{-}Datec(17, 1927)\\
&\mu(dt, day \mapsto 29) = mk\text{-}Datec(29, 1927)\\
&\mu(dt, year \mapsto 1937) = mk\text{-}Datec(17, 1937)
\end{aligned}
$$

ADJ diagram (Datec)



Data type invariants

- Truth-valued functions to record restrictions
- Part of the type definition (Keyword: **inv**)

$$\begin{aligned} Datec \; :: \; & day \; : \; Day \\ & year \; : \; Year \end{aligned}$$

$$\mathbf{inv}\ (mk\text{-}Datec(d, y)) \triangleq is\text{-}leapyr(y) \vee d \leq 365$$

- Defines the set

$$\{mk\text{-}Datec(d, y) \mid d \in Day \wedge y \in Year \wedge inv\text{-}Datec(mk\text{-}Datec(d, y))\}$$

- where

$$inv\text{-}Datec(mk\text{-}Datec(d, y)) \quad \triangleq \quad is\text{-}leapyr(y) \vee d \leq 365$$

Content

1. Historical context
2. VDM
3. A proof framework for VDM (mural)
4. VDMTools

mural

- Developed as part of IPSE 2.5 project
- User-guided proofs
  - Convincing arguments for conjectures made on VDM models
- Users need expertise in structuring a formal proof
  - User completes proofs
- Book-keeping and selection of applicable rules

Constants and Expressions

- Symbols
  - Variables
    - Collections of values
  - Constants
    - Value and type constructors ({} = empty set,…)
    - Fixed arity (x,y): x…value arguments, y…type arguments
  - Binders
    - Introduce and bind new variables (quantifiers,…)

Constants and Expressions

- Expression
  - Variable symbol
  - Constant symbol (correct number of arguments)
  - Binder
    - Binding a variable in another expression

Rules of inference

- Inference rules
  - Above: Hypothesis
  - Below: Conlcusion

$$\boxed{\_+1\text{ - form}}\ \frac{n:\mathbb{N}}{(n+1):\mathbb{N}}$$

- Axiom
  - “Ax” to the right of the rule

$$\boxed{0\text{ - form}}\ \frac{}{0:\mathbb{N}}\,\mathrm{Ax}$$

Proofs

- Arguments from hypotheses to conclusion
- Organized into blocks
  - from
  - infer
- Inference steps are numbered lines
  - Formula
  - Justification

Proofs

- Example:
  - Conjecture

$$\boxed{\text{Conj1}}\ \frac{ns\colon \mathbb{N}^*}{[0] \curvearrowright ns\colon \mathbb{N}^*}$$

  - Proof

	from	$ns\colon \mathbb{N}^*$	
1		$0\colon \mathbb{N}$	0-form
2		$[0]\colon \mathbb{N}^*$	singl-form(1)
	infer	$[0] \curvearrowright ns\colon \mathbb{N}^*$	$\curvearrowright$-form(2,h1)

Proofs

- Sub-proofs

	from $P \Rightarrow Q;\ Q \Rightarrow R$	
1	from P	
1.1	Q	modus ponens(1.h1,h1)
2	infer R	modus ponens(1.1,h2)
	infer $P \Rightarrow R$	deduction(1)

Proofs

- Syntactic definition of constants

$$e_1 \wedge e_2 \triangleq \neg(\neg e_1 \vee \neg e_2)$$

$$\downarrow$$

	$\ldots$	
5	$\neg((A \wedge B) \vee \neg C)$	
6	$\neg(\neg(\neg A \vee \neg B) \vee \neg C)$	unfolding(5)
7	$(\neg A \vee \neg B) \wedge C$	folding(6)
	$\ldots$	

Theories

- Theory is a collection of
  - Constants
  - Binder definitions
  - Derived results (+ proofs)
- Theory store
  - Collection of theories
- Advantages
  - Reuse
  - Limits the scope

Example: mural proof (1)

- Inference rule: $$\boxed{\forall\text{-I}}\ \frac{y{:}\,A \vdash_y P(y)}{\forall x{:}\,A \cdot P(x)}$$

- mural
  - Status: 'unproved'
  - Selection: proof display opens

from $y{:}\,A \vdash_y P(y)$

$\cdots$

infer $\forall x{:}\,A \cdot P(x)$ $\langle$?? justify ?? $\rangle$

Example: mural proof (2)

- Conclusion line flagged ‘unjustified’
- User is free to decide how to approach
  - Backwards from the goal
  - Forwards from the knowns
- Tools to search theory store for applicable rules
- Expert
  - Choose a specific rule
  - Tool manages the pattern-matching

Example: mural proof (3)

- User chooses to work backwards
  - Definition of ∀ as ¬ ∃

$$
\begin{array}{lll}
 & \textbf{from } y{:}\, A \vdash_y P(y) & \\
 & \quad \ldots & \\
\text{a} & \quad \neg\exists y{:}\, A \cdot P(y) & \langle\ ??\ \text{justify}\ ??\ \rangle \\
 & \textbf{infer } \forall x{:}\, A \cdot P(x) & \text{folding(a)}
\end{array}
$$

Example: mural proof (4)

- Rules with sequent hypothesis → sub-proof

	from $y: A \vdash_y P(y)$	
b	**from** $z: A$	
	$\ldots$	
	infer $\neg(\neg P(z))$	$\langle$?? justify ?? $\rangle$
a	$\neg \exists y: A \cdot P(y)$	$\neg$-$\exists$-I(b)
	infer $\forall x: A \cdot P(x)$	folding(a)

Example: mural proof (5)

- Proof is completed by forward reasoning within the sub-proof

	from $y : A \vdash_y P(y)$	
1	**from** $z : A$	
1.1	$P(z)$	sequent h1 (1.h1)
	infer $\neg(\neg P(z))$	$\neg\neg$-I(1.1)
2	$\neg \exists y : A \cdot P(y)$	$\neg$-$\exists$-I(1)
	infer $\forall x : A \cdot P(x)$	folding(2)

Content

1. Historical context
2. VDM
3. A proof framework for VDM (mural)
4. VDMTools

VDMTools

- Works with VDM++
  - Extended version of VDM-SL
  - Object-orientated
- Features
  - Syntax checking
  - Type checking
  - Code generation (Java, C++)

VDM++

```
class <class-name>
  instance variables
    ...
  types
  values
  functions
  operations
    ...
  thread
    ...
  sync
    ...
end <class-name>
```

Internal object state

Definitions

Dynamic behaviour

Synchronization control

Resources

- Literature
  - Dines Bjørner, Martin C. Henson – “Logic of Specification Languages”
  - Cliff B. Jones – “Systematic Software Development using VDM”
  - Peter Gorm Larsen – “VDM++ Tutorial”
- Software
  - VDMTools – www.vdmtools.jp/en/