VDM recursion in Isabelle/HOL

Leo Freitas and Peter Gorm Larsen
21st Overture Workshop, Lubeck March 2023
Model based design with VDM

- **Modelling aspects**
  - Abstract data types with invariant and ordering,
  - Function definitions with specification (pre/post/measures),
  - State rich definitions with statements, variable frames and operations.

- **Symbolic execution**
  - Interpretation of mathematical properties of systems (e.g. possibly with OO and RT features)
  - Limited type scope (e.g. mostly without explicit type bindings; no “x : nat”)

- **Testing:**
  - Combinatorial traces
  - Code coverage

- **Verification:**
  - PO generation
  - Quick check (PO disproof) within given domains
  - Various proof support attempts since Jones’ Mural theorem prover halted
  - No symbolic counter example (no model-based / unbounded counter examples)
VDM proof support history since Mural (1991)


2. Karabotsos (deep) embedding of VDM (LPF) in Isabelle/HOL (2005)
   ○ https://spectrum.library.concordia.ca/id/eprint/8505/1/MR10289.pdf

3. Freitas & Woodcock soundness argument/result for VDM proofs in other logics (2008)
   ○ Unifying Theories of Undefinedness [https://doi.org/10.3233/978-1-58603-976-9-311]
   ○ Linking Z and VDM: (Z semi-classical) logic prover for VDM (LPF) theorems [https://ieeexplore.ieee.org/document/4492887]

   ○ Hand-crafted proof tactics [https://dl.acm.org/doi/10.1145/1774088.1774608]

5. Freitas & Whiteside (shallow) embedding of VDM in Isabelle/HOL (2014)
   ○ VDM theorems proved in Z and Isabelle/HOL [https://link.springer.com/chapter/10.1007/978-3-319-06410-9_20]

6. Freitas automated (shallow) embedding of VDM in Isabelle/HOL with proof crafting support (2021)
   ○ VDM toolkit project [https://github.com/leouk/VDM_Toolkit]
VDM to Isabelle translation strategy 101

- Translation strategy started (2010) as part of the AI4FM project (2010-2014)
  - Attempt to identify proof strategy reuse across provers (Isabelle, ZEves) and models (VDM and Z)
  - Technical report on a translation strategy for most of VDM as shallow embedding
  - Undergraduate course on translating VDM to Isabelle (manually) (2012-2022)

- VDM LPF
  - Pragmatic approach similar to VDMJ’s: a L-R logic (e.g. subset of LPF) with “possible” typing rules

- VDM data types
  - Sets, sequences, maps and records map almost directly to Isabelle libraries
  - Numeric types cannot be directly translated given VDM widening-type rules (e.g. "0 - x" becomes int for x:nat)
  - Set and sequence comprehensions are easy; map comprehension is fiendish

- VDM functions
  - Direct translation as Isabelle definitions for non-recursive functions
  - Recursive functions support quite limited at first (e.g. only for VDM sequences)
VDM Toolkit Project

- Initiative to coalesce various VDM-related developments (e.g. libraries, experiments, etc.) since 2010
- VDM to Isabelle translator (vdm2isa plugin)
  - VDMToolkit: Isabelle proof engineering and support for VDM translation and proof automation
  - exu: VDM style checker and specification reordering (see next talk)
  - vdm2isa: Syntax-driven VDM to Isabelle translator
  - isapog: VDM PO translator and proof script / strategy predictor
  - vdm2isa-lsp: VSCode LSP (editor) + DAP (debugger) integration of plugins
- VDM ANTLR
  - VDM syntax formal definition for parsing, printing, translations, etc.
- VDM Libraries
  - CSV, ISQ, Order, Z-Relations, Dense ranges, Logging, Binary and Matrix arithmetic, General support, etc.
- VDM Annotations
  - Specification profiling; user defined theorems; user defined proof attributes, hints and witnesses.

https://github.com/leouk/VDM_Toolkit
Isabelle recursion principles

- Primitive recursion
  - Reduction rules per type constructors; not possible to have pattern matching

- Total function with automatic proof
  - Implemented on top of primitive recursion with extended pattern matching and non-constructive types
  - Proof obligations:
    - Pattern compatibility (i.e. is the pattern given matchable to input type?)
    - Pattern completeness (i.e. are patterns given exhaustive?)
    - Recursion termination (i.e. are recursive calls well-founded?)

- Total (partial) function with user defined proof
  - Copes with any type and extended pattern as well as partial (non-terminating) functions
  - Partial functions require abstract domain predicates assumptions (psimps-rules) everywhere!
  - Pattern compatibility and completeness proofs are mandatory and with reasonable automation support
  - Termination proofs rely on knowledge of the "Size Change Termination" (SCT) principle(s)!
Extending translation of VDM recursion to Isabelle

- Translation of recursive functions restricted over VDM sequence parameters only
- Recursive functions over nat parameters are (surprisingly) non-trivial!
- Users requested support for sets, maps, and of course, nat!

AKA: PGL’s Napkin @ ISoLA22 :-)

\[
\text{Sum} : \text{set} \rightarrow \text{nat} \\
\text{Sum}(S) = \begin{cases} 
0 & \text{if } \emptyset \subseteq S \\
\text{Sum}(S) & \text{if } \text{words}(S) = 7 \\
\text{Sum}(S) \cup \text{Sum}(S) & \text{otherwise}
\end{cases}
\]
VDM-recursion Isabelle-translation caveats

- Isabelle ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$) types are defined constructively through different embeddings
  - $\mathbb{N}$: defined inductively over two constructors (e.g. zero and suc $n$)
  - $\mathbb{Z}$: defined algebraically as a quotient type between two $\mathbb{N}$ (e.g. positive and negative parts)
  - $\mathbb{Q}$: defined algebraically as a quotient type between two $\mathbb{Z}$ (e.g. numerator and denominator parts)
  - $\mathbb{R}$: defined algebraically as a "vanishing" Cauchy sequence quotient type

- VDM type widening rules forces the use of maximal type for translation
  - In VDM, for a $x$:nat, "0-x" becomes "-x" of type int. In Isabelle this is "0:$\mathbb{N}$"!
  - Translation encode VDM nat as VDMNat ($\mathbb{Z}$) and $\mathbb{Q}$ as $\mathbb{R}$
  - Isabelle recursion must be constructive (i.e. will require various transformations for non $\mathbb{N}$)

- VDM recursion over sets and maps are finite; Isabelle sets are infinite and axiomatic
  - Isabelle requires a constructive well-formed recursive relation; some of which are inferrable
  - Isabelle will impose well-formed proof obligations for sets and map domain's finiteness

- VDM measures are not expressive enough for certain recursion patterns (e.g. ack, recursive types, etc.)
  - Some (complex) recursive measures *must* be relational
VDM $\mathbb{N}$-factorial example caveats

Recursive VDM factorial
- Trivial recursive measure

Isabelle $\mathbb{N}$ factorial
- Automatically discovered measure

Isabelle VDMNat ($\mathbb{Z}$) factorial
- Requires user to prove termination

```plaintext
fun fact' :: $\langle \mathbb{N} \Rightarrow \mathbb{N} \rangle$ where $\langle \text{fact'} \; n = (\text{if} \; n = 0 \; \text{then} \; 1 \; \text{else} \; n \times (\text{fact'} \; (n \cdot 1))) \rangle$

fun fact :: $\langle \text{VDMNat} \Rightarrow \text{VDMNat} \rangle$ where "fact \; n = (\text{if} \; n = 0 \; \text{then} \; 1 \; \text{else} \; n \times (\text{fact} \; (n - 1)))"

Unfinished subgoals:
(a, 1, <):
1. $\forall n. \; n \neq 0 \Rightarrow |n - 1| < |n|
(a, 1, <=):
1. $\forall n. \; n \neq 0 \Rightarrow \text{nat} \; |n - 1| \leq \text{nat} \; |n|
```
Translation recipe for basic types

- Translate (pre/post) specifications
  - Follows usual VDM translation strategy yet creating definition sets

- Translate recursive definition itself
  - Flag controls whether to try Isabelle discovered proofs (e.g. “fun”) or user defined (e.g. “functions”)

- Infers recursive relation from VDM AST
  - Flag controls whether to generate lemma about well-formedness of inferred recursive relation
  - Presumes inferred relation is within largest well-formed relation from lower bound (e.g. for $\mathbb{N}$ bound is 0)

```
definition largest_wf_int_rel :: "\mathbb{Z} \Rightarrow (\mathbb{Z} \times \mathbb{Z}) set" where
  "largest_wf_int_rel d = {{z', z}. d \leq z' \wedge z' < z}"
```

- Sets up pattern consistency, completeness and termination proofs
  - Pattern proofs are almost always found by sledgehammer (unless wicked patterns or mutually recursive calls)
  - Termination proof presumes inferred relation is within largest well-formed relation; up to the user otherwise
  - Harder recursive patterns will require users to define recursive relation as VDM annotations (@IsaMeasure, @Witness)
Isabelle Demo (or see paper theory sources)

1. Recursion for constructively defined basic types ($\mathbb{N}$)
2. Recursion for non-constructively defined basic types ($\mathbb{Z}$)
3. Recursion for constructively defined structured types (seq)
4. Recursion for non-constructively defined structured types (set and maps)
5. Complex recursion patterns (e.g. Ackerman, Permutation, Takeuchi, etc.)
6. Simple mutual recursion (e.g. odd and even)
7. Complex mutual recursion (e.g. N-Queens, Sudoku solvers)


Discussion

- **Results**
  - Semi-automated (with proof support) translation recipes/strategies for:
    - VDM some numeric types (nat, nat1, int)
    - VDM structured types (sets, sequences, maps)
  - Automation caters for most common VDM recursive situations
    - Decreasing nat/int, set (or map domain), sequence

- **Limitations**
  - Recursive VDM types (e.g. VDM records for say linked lists)
  - General upper bound for inferred recursive relations works for nat, int and (simple) sets only.

- **Complexities (e.g. will require user-defined auxiliary lemmas and their proofs)**
  - Recursion over user-defined types
  - Recursion where recursive relation is outside predefined space
  - Mutual recursion will necessitate handling Isabelle union types
Discussion

- **Wish list**
  - completeness of VDM patterns to enable handling FMI models
  - including Isabelle/LSP back end IDE run in background to attempt proofs automatically
  - implement complex (and mutual) recursion templates (i.e. POC for now)
  - VSCode code lenses integration (e.g. akin to jUnit testing)
  - ????
  - ????