Topological Sorting VDM-SL for Isabelle/HOL translation

Leo Freitas and Nick Battle
21st Overture Workshop, Lubeck March 2023

VDM to Isabelle translation strategy (201)

- Each VDM module becomes a corresponding Isabelle theory, including DEFAULT
- VDM imports become Isabelle imports; what is not exported become Isabelle hidden constants
- VDM types and expressions, through the `VDMToolkit.thy`, become corresponding Isabelle types
  - Differences must be catered for on per type/expression basis
  - +95% types+expr handled (e.g. except for certain unions, recursive records, incompatible coercions, etc.)
- VDM functions and specification (pre/post/inv) become Isabelle definitions
  - Definition sets are created to enable layered definition unfolding, hence taming proof complexity
  - Recursive functions become Isabelle function package definitions with termination proof lemmas
- VDM state and operations become Isabelle definitions in a monadic style
  - State parameters are made explicit and behave akin to a monad
  - Statement translation strategy exist but is yet to be fully implemented (about 15% only).

Key difficulties of VDM translation (in general?)

1. Module dependencies and definition order
   - Isabelle imposes "declaration before use" and every theory on a file of the same name
   - VDM permits "declaration after use" and resolves forward references through multiple passes
   - Multiple VDM module-less (e.g. DEFAULT) specification leads to single file (e.g. FMU models example)
   - **Definitions and dependencies had to be reordered by the user**
   - This becomes awkward where invariants and other specification refers to auxiliary functions.
   - **Prevented readily translation of legacy models.**
2. Type unions are over-expressive and allows convoluted situations (see paper)
3. Certain VDM patterns allow for highly abstract (and powerful) invariants that are difficult to handle
4. Target differences lead to error prone scenarios (e.g. Isabelle lists are 0-based, maps are total, etc.)

Historical developments

- EMV2 model (150+ modules, XML linked, 60+ KLOC VDM) issues (TC: +4m, IN: +15m)
  - Led to creation of VDM annotations (e.g. dependencies, profiling, printf, specific failure, etc.)
  - Verbose TC output on its multiple passes and all forward reference dependency warnings
  - Minimal passes led to faster times, which demanded least dependencies by chasing forward dep. warnings
- Topological sorting of module dependencies and DOT file output for large developments
  - VDMJ plugin suggesting the user “optimal” (least dependant) **module orders**
- Isabelle translation requires declaration before use; could we reuse topological sorting?
  - *Eureka!*: apply topological sort per module definition as well as across modules (!)
- Fine tune specific topological sort needs for translation
  - Consider both type and function definition spaces
  - Specialised visitor searches across the AST

VDM recursive cycles situations

1. Recursive cycles in modules (e.g. diamond module dependencies)
   - VDMJ type checker uses multiple passes to determine type correctness
   - VDMJ POG generates POs for unknowable situations (e.g. narrower union/nil type results)
   - FMI Rule Model field selection over `[FMIVariable]`: PO on selected union having requested field
2. Recursive cycles in functions (and operations) (e.g. recursively defined functions like `factorial`)
   - VDM likes users to provide recursive measure- functions to ensure termination; warns users if one is not given
   - Function (type instantiation and) application detects mutually recursive cycles that might not terminate
   - VDM measures are a function from inputs to nat; some measures are impossible to write in the language as-is
     - Various common recursive patterns that involve application in its parameters (e.g. `ack(m-1, ack(m, n-1))`)
3. Recursive cycles in types (e.g. `LinkedList :: x: nat next: LinkedList inv l == f(l)`)
   - VDMJ type checker separates the environment of types from the environment of functions definitions
   - Type invariants might also be further defined by auxiliary recursive functions

exu **plugin - VDM style checker**

- Originally a VDM style (extended) type checker in preparation for Isabelle translation
  - Traverses the VDM AST looking for specific constructs
  - Checks call dependencies per function (including pre/post/inv) per module
  - Checks duplicated pattern-kind use (e.g. f(mk_R(x), mk_R(y)))

- Currently operates on a extendable set of inner (git-style) commands
  - sort : topologically sorts all module definitions to enforce declaration before use order
  - graph : prints the dependency DAG between all definitions
  - check : performs AST structural and call dependency checks

- Before module definition topological sorting was not “essential” (but useful) for Isabelle translation

exu's **algorithm**

1. Collect all named definitions
2. Process non-function (e.g. type) space dependencies recursively
   a. Visits all type definitions
   b. Creates any missing inv_T calls, for all declared types T, recursively;
   c. Links type and function spaces dependencies (e.g. invariant with aux. function);
3. Process function definition space dependencies recursively
   a. Visits all function bodies and specifications
   b. Ignores recursive calls (VDMJ handles those);
   c. Links function named dependencies;
4. Topological sort:
   a. Checks whether topological sort is needed;
   b. Applies Kahn's algorithm for DAG sorting of top-level (starting) names;

exu's **algorithm**

5 Module reconstruction:

a. Organise top-level names to separate type from function name-spaces;
b. Reorder module definitions;
c. Optionally re-typechecks module;

- Discovered a case where module reordering gave different type checking results!
  - Ordered definitions lead to VSCode "red squiggles" type error!

```
v0 = mk_(mk_(1,2), mk_(1,2));
v1 = cases v0: mk_(mk_(x,y), mk_(y,z)), mk_(x, y, z) -> x+y+z end;
```

  - Unordered definition VSCode says nothing, VDMJ/DAP complains!

```
v1 = cases v0: mk_(mk_(x,y), mk_(y,z)), mk_(x, y, z) -> x+y+z end;
v0 = mk_(mk_(1,2), mk_(1,2));
```

exu's **algorithm**

- Algorithm properties and invariants
  - Algorithm degenerates when attempting to sort already sorted modules
    - Can make the outcome unsorted!
  - Mutually independent names might get reordered
  - Organised names must account for all original names and be within sorted names
  - Original subset sorted and original = organised
- This extra check involves a further pass over AST, potentially leading to LSP lag
- In practice this is yet to manifest (e.g. we tested on FMI and EMV2 models)

VSCode integration

- exu is embedded in VSCode as an LSP (i.e. VSCode editor) extension
  - Users see “red/yellow” squiggles for exu’s errors and warnings
- exu’s options are embedded in VDM-VSCode settings
- Exu provides an extra command within LSP for profiling
  - This is useful for large / complex models with multiple models
- So far the integration causes no perceptible lag even for large specifications
- **Plan is to integrate within VDM-VSCode release cycle**

LSP + console demo (if you are feeling brave) :-)

- LSP VSCode demo
- Command line detailed output
- Dot file output rendering

Industrial-scale examples

- FMI rules (and other) models
- Spook language
- UNOS lung transplant model
- **Community driven effort / suggestions (e.g. extensions based on demand/request)**