

Towards a Static Check of FMUs in VDM-SL

Nick Battle, Casper Thule, Cláudio Gomes, Hugo Daniel Macedo

17th Overture Workshop, Oct 2019

Work Outline

- VDM-SL model of FMI static semantics (configuration)
- Builds on 2016 work by Mirran, Peter J & Kenneth
- Extended to cover the whole of FMI 2.0
- Incorporated into a tool to analyse FMU files
- Tested against the FMI Cross-Check Repository

VDM Modelling

- XML/XSD easiest to model as records + invariants:

```
<ScalarVariable
  name="h"
  valueReference="0"
  causality="output"
  variability="continuous"
  initial="exact">
  <Real
    start="1"
    declaredType="Position"/>
</ScalarVariable>
```

```
ScalarVariable ::
  name           : NormalizedString
  valueReference : nat
  causality      : [Causality]
  variability    : [Variability]
  initial        : [Initial]
  variable       : Real | Integer | ...

inv sv ==
  ...;

Real ::
  declaredType : [NormalizedString]
  min          : [real]
  max          : [real]
  start        : [real]

inv r ==
  ...;
```

VDM Modelling

- Split invariants out as validation functions:

```
Real ::  
  declaredType    : [NormalizedString]  
  min             : [real]  
  max             : [real]  
  start          : [real];
```

```
isValidReal: Real +> bool  
isValidReal(...) ==  
  (max <> nil and min <> nil =>  
    max >= min)  
and  
  (start <> nil =>  
    (min <> nil => min <= start)  
    and  
    (max <> nil => max >= start));
```

VDM Modelling

- Use sets to get around McCarthy logic:

```
isValidReal: Real +> bool
isValidReal(...) ==
{
  -- If max and min defined, max is >= min
  max <> nil and min <> nil =>
    max >= min,

  -- If start and min defined, min <= start
  start <> nil and min <> nil =>
    min <= start,

  -- If start and max defined, max >= start
  start <> nil and max <> nil =>
    max >= start
}
= {true};
```

VDM Modelling

- Use VDM *annotations* for clean error handling:

```
isValidReal: Real +> bool
isValidReal(...) ==
{
  -- @OnFail("2.2.7 max %s is not >= min %s", max, min)
  max <> nil and min <> nil =>
    max >= min,

  -- @OnFail("2.2.7 start %s is not >= min %s", start, min)
  start <> nil and min <> nil =>
    min <= start,

  -- @OnFail("2.2.7 start %s is not <= max %s", start, max)
  start <> nil and max <> nil =>
    max >= start
}
= {true};
```

VDM Modelling

- Annotation comments can be extensive:

```
isValidCoSimulation: [CoSimulation] +> bool
isValidCoSimulation(cs) ==
  cs <> nil =>
    cs.sourceFiles <> nil =>
      /* @OnFail("4.3.1 CoSimulation source file names are not unique: %s",
        let files = cs.sourceFiles in
          { files(a).name | a, b in set inds files &
            a <> b and files(a).name = files(b).name } )

        The file names within the sequence of source files listed for the
        CoSimulation must be unique. This is determined by checking that the
        set of names is the same size as the length of the list of files.
      */
      ( len cs.sourceFiles = card { f.name | f in seq cs.sourceFiles } );
```

VDMCheck Tool

- To be useful, the model needs to drive an “FMU checking tool”
 - XML extracted from FMU file with unzip
 - A SAX parser generates a VDM-SL “fmu” value from FMU XML
 - Parser adds XML line numbers for *@OnFail* messages
 - Generated VDM-SL “fmu” value combined with model types/functions
 - Execute *isValidFMIModelDescription(fmu)* automatically with “-e”
 - *@OnFail* lists any problems found
- Process wrapped in a bash script, *VDMCheck.sh*
- Works on FMU ZIP files, or raw XML
- Similar to existing FMU Compliance Checker (*fmuCheck -x*)

VDMCheck Tool

\$ VDMCheck.sh

Usage: VDMCheck.sh [-v <VDM outfile>] <FMU or modelDescription.xml file>

\$ VDMCheck.sh WaterTank_Control.fmu

No errors found.

\$ VDMCheck.sh modelDescription.xml

2.2.7 Causality/variability/initial/start <input>/<continuous>/nil/nil invalid at line 6

2.2.7 ScalarVariables["v1"] invalid at line 6

2.2.1 ScalarVariables invalid

2.2.8 Outputs should be omitted at line 10

Errors found.

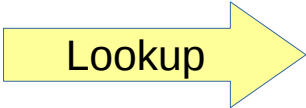
<https://github.com/INTO-CPS-Association/FMI2-VDM-Model/releases>

VDMCheck Tool

- Trivial fields can have complicated semantics!

Real ::

```
declaredType : [NormalizedString]
min           : [real]
max          : [real]
start        : [real];
```

Lookup 

Real ::

```
min : [real]
max : [real];
```

“max >= min required”

“max >= start >= min req.”

“If not defined, the min/max is the largest negative/positive number that can be represented on the Machine.”

“The value[s] defined in the [declaredType] TypeDefinition [are] used as default.”

isValidReal: Real +> bool

isValidReal(...) ==

```
(max <> nil and min <> nil =>
  max >= min)
```

and

```
(start <> nil =>
```

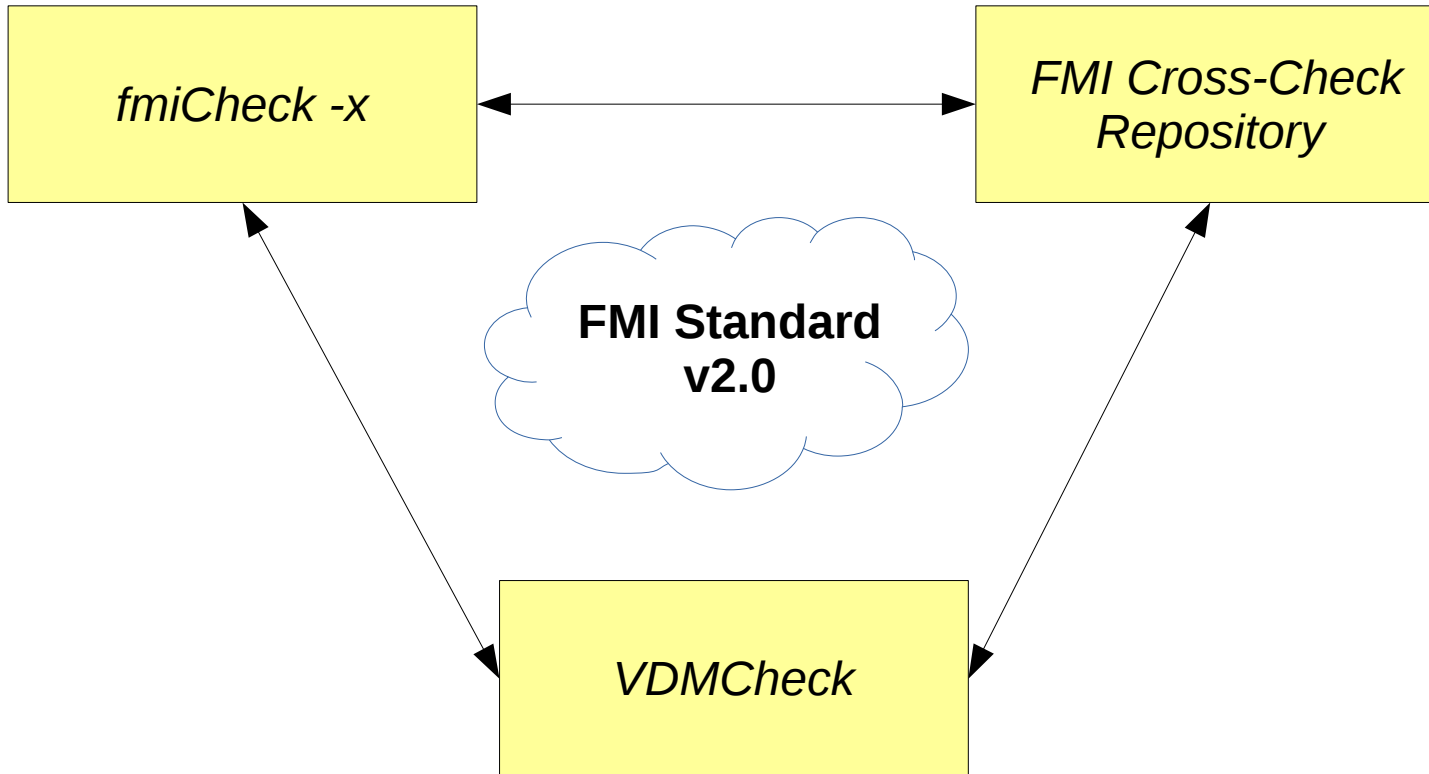
```
(min <> nil => min <= start)
```

and

```
(max <> nil => max >= start));
```

VDMCheck Tool

- But which static semantics is correct?



VDMCheck Tool

- FMI Cross-Check Repository has 692 FMU examples
- *VDMCheck* and *fmucCheck -x* executed on all of them:

Problem Found:	VDMCheck	fmucCheck -x
None	294 (42%)	530 (77%)
Missing <i>ModelStructure InitialUnknowns</i>	118	0
Invalid structured <i>ScalarVariable</i> names	123	123
Invalid <i>ModelStructure Derivatives</i>	124	27
Invalid <i>ScalarVariable</i> attributes	37	12
Invalid aliases	56	0
Invalid “reinit” flag	24	0
Real “unit” not defined in <i>UnitDefinitions</i>	14	0
Invalid <i>ModelStructure Outputs</i>	13	0
Unsorted <i>InitialUnknowns</i>	4	0

Future Work

- Continue to try to establish *intended* static semantics
- Link annotation comments directly to FMI Standard?
- Use code generation for efficiency?
- Extend model to cover FMI Standard *dynamic* (API) semantics
- Modelling of co-simulations of many FMUs
 - Maestro JSON to VDM-SL conversion started
 - Model defines initialization process and algebraic loops
- Migrate model to cover FMI v3.0?