D2.1.1 – Modelica extensions for properties modeling
Part IVb: FORM-L and Modelica: syntax and relationships

WP2.1 – Properties modelling language
WP2 – Properties modelling and Safety

MODRIO (11004)

Version 3.0

Date April 19, 2016

Authors Alfredo Garro, Andrea Tundis University of Calabria, Italy

Martin Otter DLR SR, Germany

Accessibility : PUBLIC

Revision

Version	Authors	Description
2016.04.19	Martin Otter	Final clean-up
2015.02.26	Alfredo Garro, Andrea Tundis (University of Calabria)	The document has been improved by introducing Section 9, Check Properties Set, with both a list of FORM_L operators and a mapping to the related Modelica operators, as well as by updating the document according to the implementation of several operators.
2014.09.25	Alfredo Garro, Andrea Tundis (University of Calabria)	The document has been improved through restructuring the table by splitting it in sub-tables according to their content as well as by introducing a textual description for each aspect.
2014.07.23	Martin Otter (DLR)	Considerably improved + operators implemented in prototype Modelica library added.
2014.07.12	Alfredo Garro, Andrea Tundis (University of Calabria)	First version (List of FORM_L operators and proposed mapping to Modelica operators)

Executive summary

This document provides more detailed information to deliverable D2.1.1 Part IV. It contains a detailed description how to express FORM_L language constructs (Thuy 2016) in a reasonable similar way in Modelica, or express it with Modelica language constructs as Modelica functions or blocks. All elements that are not specially marked are either already available in Modelica 3.2 (with the extensions sketched in D2.1.1 Part IV) or are provided in the accompanying Modelica library Modelica_Requirements 0.6.

Contents

Note

- FORM-L constructs not yet implemented are marked in light blue.
- Operators that are unclear, are marked in yellow.
- The numbers in parenthesis (e.g. 12.3), are the chapter numbers of the FORM_L specification.
- Modelica functions start with a lower case letter (= operators without memory). Functions can be called with positional arguments (e.g. “card(b)”).
- Modelica blocks start with an upper case letter (= operators with memory). Blocks can only be called with named arguments (e.g. “After(u=b)”).
- In many cases the input argument of the Modelica function/block is called “u” (as usual in the Modelica Standard Library). The question is whether this should be changed (in some cases to “event”, if the rising edge of the input is used; or in some cases to “condition”).

1. Introduction

This document aims at providing a summarization of the main concepts that the FORM-L language has introduced as well as their mapping on the Modelica language. In particular, the following Tables (from Table 1 to Table 7) summarize the aspects that have been covered such as properties, requirements, assumptions, guards, etc. The first column of each table shows the FORM_L syntax, the second column the corresponding Modelica constructs, and in the third column additional comments are provided for a better comprehension.

2. Properties, Requirements, Assumptions, Guards

In Table 1 are reported the main concepts that have been identified for modeling systems properties: (i) Requirements, which are properties that MUST be satisfied; (ii) Assumptions, which are properties that are supposed to be satisfied; (iii) Guards, which are properties that state the conditions that must be satisfied for a model to be valid (see Figure 1).

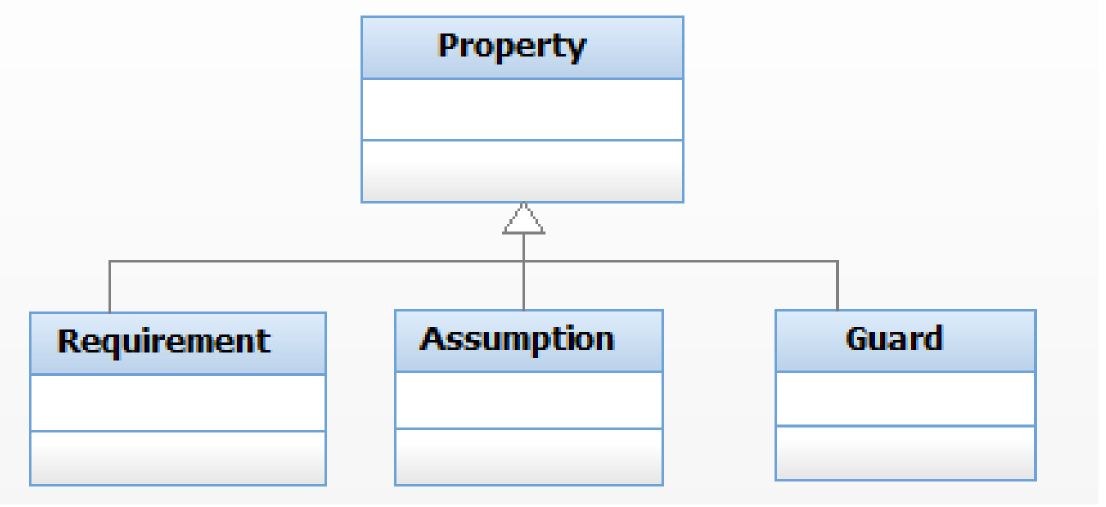

Figure 1: Semantic of the Properties in FORM-L

Operator mapping		Description (Modelica)
FORM-L	Modelica	
Properties, Requirements, Assumptions, Guards (6.)		
Boolean b;	***Boolean*** b;	Boolean variable with values false or true.
property p;	***Property*** p;	Enumeration with values Violated, Undecided, Satisfied
required property r = p;	***Requirement*** r(property = p);	Block where property is monitored. The information about a violated or untested property is written on a log file (for further post-processing) and at the end of the simulation a summary is printed to the output window.
assumed property a = p;	***Assumption a(property*** a = p);	Block where property is monitored. The information about an untested property is written on a log file (for further post-processing) and at the end of the simulation a summary is printed to the output window. If property a = Violated, an assert stops the simulation. This is also reported on the log file.
guard property g = p;	***assert(p <> Property.Violated, “descr.”);***	An assertion is raised if property p is Violated and the text “descr” is printed as error message (so the simulation is stopped with an error)

Table 1: Properties, Requirements, Assumptions, Guards from FORM-L to the Modelica language

3. Basic Language Constructs

In Table 2 the mapping among the basic constructs of the FORM-L language and those already offered by the Modelica language is reported.

Operator mapping		***Description (Modelica)***
FORM-L	***Modelica***	
Basic Language Elements		
`function`	`variable`	Available Modelica language constructs
`real`	***`Real`***	
`integer`	***`Integer`***	
`fixed`	***`parameter`***	
`constant`	***`constant`***	
`real v initially 0.0 end v;`	`Real v(start=0, fixed=true);`	
`sum {S in Steps |S.DemandToBPS};`	`sum(S.DemandToBPS for S in Steps);`	
`when not(Powered) then 0.0 else Power;`	`if not Powered then 0.0 else Power;`	

Table 2: Basic Language Elements from FORM-L to the Modelica language

4. Conditions

Table 3 reports the matching about conditions that in FORM-L represent expressions or functions that can be evaluated over time and can assume three values: true, false or undefined.

Operator mapping		Description (Modelica)
FORM-L	Modelica	
Conditions (4.)		
not(condition) condition **or** condition condition **and** condition	**not** condition condition **or** condition condition **and** condition;	Built-in operators of Modelica on Boolean variables. (3-valued logic is planned to be supported only in a very limited form by a few operators: Requirement, during).
condition **xor** condition condition **nand** condition condition **xnor** condition	**xor**(u1,u2) nand(u1,u2) xnor(u1,u2)	xor/nand/xnor of two Booleans.
condition **excl** condition	**excl**(u1,u2)	= a and not b
first(condition)	**First**(u=…);	Until condition has a falling edge (true -> false), the block returns u. Otherwise, it returns false.
count(condition)	**Count**(u=…);	Returns how often the Boolean input has a rising edge (changes from false to true)

Table 3: Conditions from FORM-L to the Modelica language

5. Set Operators

In Table 4 a set of Boolean and arithmetic operators that allow managing and making easier operation on vectors are reported.

Operator mapping		***Description (Modelica)***
FORM-L	***Modelica***	
Set Operators (12.2-12.5)		
card `set`	***card***`(b);`	Returns the number of elements of a Boolean vector b that are true
first`(set)`	***first***`(b);`	= b[1]
last`(set)`	***last***`(b);`	= b[end]
or`{p` ***in*** `pumps | p.isActive}`	***exists***`({p.isActive` ***for*** `p` ***in*** `pumps});`	Returns true if at least one element of a Boolean vector is true ('or' of all elements)
and`{p` ***in*** `pumps | p.p_a >p_cavitate}`	***forall***`({p.p_a > p_cavitate` ***for*** `p` ***in*** `pumps});`	Returns true if all elements of a Boolean vector are true ('and' of all elements)"
xor`{p` ***in*** `pumps | p.isActive}`	***oneTrue***`({p.isActive` ***for*** `p` ***in*** `pumps});`	Returns true if exactly one element of a Boolean vector is true ('xor' of all elements)

Table 4: Set Operators from FORM-L to the Modelica language

6. Continuous Time Locators

In Table 5 the definition of Continuous Time Locators in FORM-L, that follow a Linear Temporal Logic (LTL) with the additional notion of sliding time windows, is reported.

Operator mapping		*Description (Modelica)*
FORM-L	***Modelica***	
Continuous Time Locators (7.3)		
During `condition` ***check*** `signal`	***implies***`(condition, check);`	Returns check, as long as the condition is true, otherwise returns true (2-valued logic return value)
	during`(condition, check);`	As long as the condition is true, returns Violated if check=false and Satisfied if check=true. Otherwise Undecided is returned (3-valued logic return value)
	during3`(condition, check);`	As long as the condition is true, returns check (= Violated, Undecided or Satisfied). Otherwise Undecided is returned (3-valued logic check and return value)
duringAny `duration` ***check*** `signal`	***duringAny***`(duration=..., check=...);`	Return true if check has been true for at least the given duration
duringAny `Dt` **check duration**`(Off) >= dt`	***duringAccumulated***`(interval=Dt, duration=dt, check=Off);`	If input signal check is cumulatively true for the defined duration within a sliding time window of length interval, true is returned, otherwise false.

after event	***After***(u=…);	After the Boolean input u has a rising edge, the output remains true, otherwise it is false.
after event1 ***untilNext*** event2	***AfterUntil***(u1=…,u2=…);	After the Boolean input u1 has a rising edge and until the Boolean input u2 has a rising edge, the output remains true, otherwise it is false
after event ***for*** duration	***AfterFor***(u=…, duration=...);	After the Boolean input u has a rising edge for a duration interval length, the output remains true, otherwise it is false
after event ***within*** duration		
after signal ***becomes*** true ***for*** duration		In Modelica the same as AfterFor(..).
until event	***Until***(u=...);	The output remains true, until the Boolean input u has a rising edge. Afterwards it is false.
every duration1 ***for*** duration2	***Every***(interval= …, duration = …);	Return true during every interval for a duration length. Otherwise return false.

Table 5: Continuous Time Locators from FORM-L to the Modelica language

7. Combining/Transforming Continuous Time Locators

FORM-L allows deriving complex Continuous Time Locators by combining the existing ones except sliding time windows that can be neither transformed nor combined directly. Such constructs are summarized in Table 6.

Operator mapping		***Description (Modelica)***
FORM-L	***Modelica***	
Combining/Transforming Continuous Time Locators (7.4)		
not(nsCTL) nsCLT1 **or** nsCTL2 nsCTL1 **and** nsCTL2 nsCTL **xor** nsCTL2 nsCTL **excl** nsCTL2	see above “Conditions (4.)”	nsCTL: non-sliding Continuous Time Locators In Modelica the Boolean operators from above (not, or, and, xor) can be used (and have the same effect).
nsCTL + duration	**DelayedRising**(u=.., duration=..);	Provide the input as output exactly delayed by duration. If time less than duration the initial value yStart hold (with a default of false).
nsCTL ***trunc*** duration	???	Semantics not clear

Table 6: Combining/Transforming Continuous Time Locators from FORM-L to the Modelica language

8. Discrete Time Locators

A Discrete Time Locator defines one or more positions in time and have not any notion of duration time but are associated to an event. When the event occurs then something can happen as a consequence. In Table 7 the main constructs defined in FORM-L and the related syntax in Modelica language are reported.

Operator mapping		***Description (Modelica)***
FORM-L	***Modelica***	
Discrete Time Locators (8.2)		
`when condition becomes true`	`edge(condition);`	Available Modelica language construts
`when condition becomes false`	`c = not condition; edge(c);`	
`when condition changes`	`change(condition);`	
`when condition becomes true check c`	`WhenRising(condition=…, check=c);`	When condition has a rising edge, the return value is Satisfied (if check=true) or Violated (if check=false). The return value is kept, until condition has a rising edge again. Before the first rising edge, Undecided is returned.
`when condition becomes false check c`	`WhenFalling(condition=…, check=c);`	When condition has a falling edge, the return value is Satisfied (if check=true) or Violated (if check=false). The return value is kept, until condition has a falling edge again. Before the first falling edge, Undecided is returned.
`when condition changes check c`	`WhenChanging(condition=…, check=c);`	When condition has a changing edge, the return value is Satisfied (if check=true) or Violated (if

		check=false). The return value is kept, until condition has a changing edge again. Before the first changing edge, Undecided is returned.

Table 7: Discrete Time Locators from FORM-L to the Modelica language

9. Check Properties Set

In Table 8 the definition of a set of other Check Properties operators is reported.

Operator mapping		***Description (Modelica)***
FORM-L	***Modelica***	
Check Properties Set		
`DuringMin condition check signal`	`MinDuration(condition=…,check=…,durationMin=…);`	In every true condition phase, check must be true for at least the defined duration.
`DuringMax condition check signal`	`MaxDuration(condition=…,check=…,durationMax=…);`	In every true condition phase, check must be true for at most the defined duration.
`DuringBand condition check signal`	`BandDuration(condition=…,check=…,durationMin=…, durationMax=…);`	In every true condition phase, check must be true at least the defined durationMin and at most the defined durationMax.
`RisingZero condition check signal`	`NoRising(condition=…,check=…,nRising=…);`	In every true condition phase, the number of check rising edges must

		be zero.
RisingFixed condition ***check*** signal	**FixedRising**(condition=…,check=…,nRising=…);	In every true condition phase, a defined number of check rising edges must occur.
RisingMax condition ***check*** signal	**MinRising**(condition=…,check=…,nRisingMin=…);	In every true condition phase, a minimum number of check rising edges must occur
RisingMin condition ***check*** signal	**MaxRising**(condition=…,check=…,nRisingMax=…);	In every true condition phase, the number of check rising edges must be bounded.
RisingBand condition ***check*** signal	**BandRising**(condition=…,check=…, nRisingMin=…, nRisingMax=…);	In every true condition phase, a minimum number of check rising edges must occur and the number of check rising edges must be bounded.

Table 8: Check Properties Set operators

10. Conclusion

The document presented a summary of the concepts provided by the FORM-L language along with the related syntax in the Modelica language. Mapping tables are used to show relation from each FORM-L construct and the related Modelica language syntax.

References

Thuy Nguyen (2016): D2.1.1 – Part III: *Formal Requirements Modelling Language FORM-L*, MODRIO deliverable, April 2016.