CRML: Common Requirement Modelling Language v1.2

A Language For Verifying Dynamic Requirements of Cyber-Physical Systems

1 Key Concepts

Why?/Motivations

Bridge the gap between the functional view (objective/KPI) and the physical view (constraints) of cyber-physical systems.

What?/Principle

CRML = a formalism to describe and verify requirements that are realistic for systems with strong physical aspects. Realistic requirements mean here:

- *dynamic* to handle interactions of the systems with their environments
- *stochastic* to target indicators that sound reasonable/achievable
- *modular* enough to support the evolution over the multiple operating modes and the multiplicity of constraints coming from the negotiation of the different stakeholders all over the system lifecycle

How to Express a Requirement?

A **requirement** is a 'special' Boolean, called *Boolean4*.

$R =$ *[Where]* *(When)* *(What)* *[How well]*

It is defined as an expression combining up to 4 items:

- a **condition** to be checked (*What*);
- a **time locator** defining when the condition has to be satisfied (*When*);
- (optionally ?!?) a **spatial locator** indicating on which object the condition has to be verified (*Where*);
- (optionally) a **probabilistic target** to indicate with which performance the condition has to be satisfied (*How well*).

How to Verify a Requirement?

The value of a **requirement** at instant t can be:

- `true` if the condition is satisfied over the defined time period;
- `false` if the condition is violated over the defined time period;
- `undefined` if the time locator has not been tested over the test scenario;
- `undecided` if the test scenario has finished before a decision could be made (i.e. before the condition has been violated or before the end of the time locator).

The goal of CRML is to be a pivotal language for verifying requirements by simulation or formal proof.
Simulation is particularly useful for requirements in the continuous-time domain: a simulation model of the tested solution provides the inputs (=`external` variables) of the CRML requirements to assess their value over the test scenarios.

Formal proof requires a translation of CRML requirements into discrete-time logics to target model checkers: time locators are limited to events and requirement's value can thus only be `true or false`.

How to Combine Requirements?

Requirements can be combined according to the algebra defined on the *Boolean4* type.

not	Logical negation			
`b`	`true`	`false`	`undecided`	`undefined`
`not b`	`false`	`true`	`undecided`	`undefined`

and	Logical conjunction			
`b1 and b2`	`true`	`false`	`undecided`	`undefined`
`true`	`true`	`false`	`undecided`	`true`
`false`	`false`	`false`	`false`	`false`
`undecided`	`undecided`	`false`	`undecided`	`undecided`
`undefined`	`true`	`false`	`undecided`	`undefined`

2 Syntax

Notation

`[ expr ]`	optional expression
`{ expr }`	expression repeated one or more times
`expr_1 | expr_2 | ... | expr_n`	n possible alternatives
`'c'`	character c
`"keyword"`	String keyword

Expressions

`[ [type] ident is] [value | external] [; |,]`	expression
`// This is single-line`	comment
`/* This is a multi-line */`	comment

Keywords

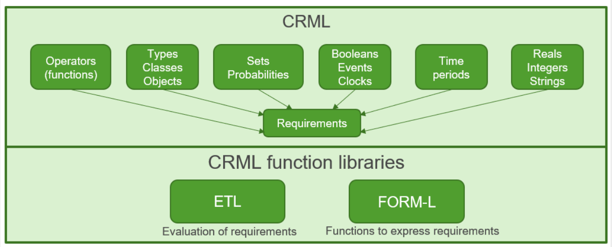
Types
`Boolean, Category, class, Clock, Event, Integer, library, model, Operator, package, Period, Periods, Probability, Real, String, Template, type`

Special values
`false, true, undecided, undefined, time`

Special characters
`(,),[, ], {, }, ,, ;, ., ", ', E, e, //, /*, */, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

Operators
`=, +, -, *, /, <, <=, >, >=, ==, <>, ^, acos, alias, and, asin, associate, at, card, constant, cos, duration, element, else, end, estimator, exp, extends, external, filter, flatten, forbid, if, integrate, is, log, log10, mod, new, not, on, or, parameter, partial, proj, redeclare, sin, start, then, tick, time from, union, variance, while, with`

Architecture



Real Operators

`Real x is decimal_value`	constructor from value
`Real x is new Real n`	constructor from *Integer n*
`x1 + x2`	binary addition
`x1 - x2`	binary subtraction
`+x1`	unary addition
`-x1`	unary subtraction
`x1 * x2`	multiplication
`x1 / x2`	division
`x1^x2`	exponentiation
`sin x1`	sine
`cos x1`	cosine
`asin x1`	inverse sine
`acos x1`	inverse cosine
`exp x1`	exponential
`log x1`	logarithm
`log10 x1`	base 10 logarithm

`if b then x1 else x2`	if *Boolean b* than else
`x1 at c`	value at *Clock c*
`duration b on P`	duration during which *Boolean b* is true over a *Period P*
`e2 - e1`	elapsed duration between two *Event*
`time frome e1`	elapsed physical time from one *Event*

Integer Operators

`Integer n is integer_value`	constructor from value
`Integer n is new Integer x`	constructor from Real x
`n1 + n2`	binary addition
`n1 - n2`	binary subtraction
`+n1`	unary addition
`-n1`	unary subtraction
`n1 * n2`	multiplication
`n1 / n2`	division
`n1^n2`	exponentiation
`if b then n1 else n2`	if *boolean b* than else
`n1 at c`	value of *Integer n1* at *Clock c*
`card c`	number of ticks of *Clock c*

String Operators

`String s is string_value`	constructor from value
`String s is new String x`	constructor from Real|Integer|Boolean x
`s1 + s2`	concatenation

Boolean Operators

`Boolean b is true`	constructor from *true* value
`Boolean b is false`	constructor from *false* value
`Boolean b is undecided`	constructor from *undecided* value
`Boolean b is undefined`	constructor from *undefined* value
`Boolean b is new Boolean c`	constructor from Clock c
`b1 and b2`	conjunction
`b1 or b2`	disjunction
`b1 and e`	conjunction with *event e*
`b1 or e`	disjunction with *event e*
`not b`	negation
`b1 * b2`	filter
`b1 + b2`	accumulation
`integrate b1 on P`	integration over a *Period P*
`b1 == b2`	equality
`if b then b1 else b2`	if *Boolean b* then else
`b1 at c`	value at *Clock c*
`x1 > x2`	strictly greater than for Real | Integer xi
`x1 < x2`	strictly less than for Real | Integer xi
`n1 >= n2`	greater than for Integer xi
`n1 <= n2`	less than for Integer xi
`n1 == n2`	equal to for Integer xi
`n1 <> n2`	different from for Integer xi
`e1 <= e2`	before for Event
`e1 < e2`	strictly before for Event
`e1 >= e2`	after for Event
`e1 > e2`	strictly after for Event

Event Operators

`Event e is new Event b`	constructor from 1st occurence of *Boolean b*
`e1 proj c`	projection on ticks of *Clock c*
`e1 proj(d) c`	bounded projection on ticks of *Clock c* for a duration *Real d*
`e1 + d`	delay of *Real d*
`tick c`	current tick of *Clock c*
`p start`	opening event of *Period p*
`p end`	closing event of *Period p*

Clock Operators

`Clock c is new Clock b`	constructor from *Boolean b*
`c1 proj c2`	projection on ticks of *Clock c2*
`c1 proj(d) c2`	bounded projection on ticks of *Clock c2* for a duration *Real d*
`c1 + d`	delay of *Real d*
`c1 filter cond tick`	filter with *cond* on ticks of *Clock c1*
`c1 and c2`	conjunction
`c1 or c2`	disjunction
`e and c1`	conjunction of *event e* and clock
`c1 and e`	conjunction of clock and event
`e or c1`	disjunction of event and clock
`c1 or e`	disjunction of clock and event
`e1 and e2`	conjunction of *Event* xi
`e1 or e2`	disjunction of *Event* xi

Period Operators

`Period p is [ | ] e1, e2 [ | ]`	constructor from *events*

Periods Operators

`Periods P is {P1, P2, ..., Pn}`	constructor from a set of *periods*
`Periods P is [ | ] c1, c2 [ | ]`	constructor from *clocks*
`P2 while P1`	filter *Periods P2* while *Periods P1*

Probability Operators

`Probability px is new Probability b`	constructor from *Boolean b*
`Probability px is new Probability b at c`	constructor from *Boolean b* at ticks of *Clock c*
`Real y is estimator px`	estimator of *Probability px*
`Real y is estimator variance px`	variance estimator of *Probability px*

3 Libraries

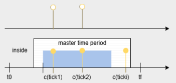
The ETL (= Extended Temporal Logics) library defines custom operators useful for evaluating requirements while the FORML library defines custom operators for expressing requirements with a more user-friendly syntax.

`’this’ is a //test`

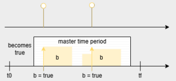
ETL Operators for Clocks

`’this’ is a //test` TO BE UPDATED

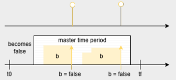
`c ’inside’ p` ....



`b ’becomes true’` ....



`b ’becomes false’` ....



`b ’becomes true inside’ p`	events generated when *Boolean b* becomes true inside a *Period*
`b ’becomes false inside’ p`	events generated when *Boolean b* becomes false inside a *Period*

ETL Operators for Evaluating Boolean Over Periods

`’decide’ b ’over’ p` decide the decision event at instant t of a Period p, could be either the violation of Boolean b or the end of the period

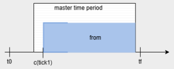
`’evaluate’ b ’over’ p` evaluate the accumulated state of Boolean b over a Period p

`’check’ Boolean b ’over’ Periods P` .... evaluate the accumulated state of Boolean b over a set of Periods P

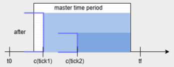
ETL Operators for Combining Boolean

`b1 ’or’ b2`	disjunction of *Boolean*
`b1 ’xor’ b2`	exclusive disjunction of *Boolean*
`b1 ’implies’ b2`	inference of *Boolean*

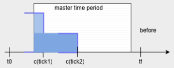
FORML Operators for Expressing Periods



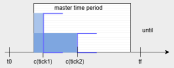
’from’ c ....



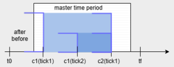
’after’ c ....



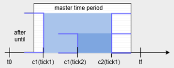
’before’ c ....



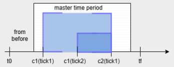
’until’ c ....



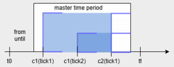
’after’ c1 ’before’ c2 ....



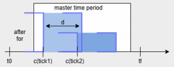
’after’ c1 ’until’ c2 ....



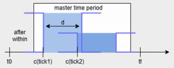
’from’ c1 ’before’ c2 ....



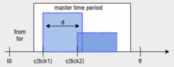
’from’ c1 ’until’ c2 ....



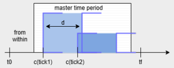
’after’ c ’for’ d ....



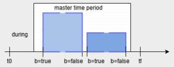
’after’ c ’within’ d ....



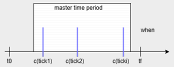
’from’ c ’for’ d ....



’from’ c ’within’ d ....

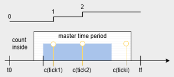


’during’ b ....



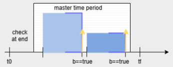
’when’ c ....

FORML Operators for Counting Events

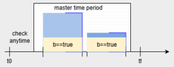


’count’ c ’inside’ p ....

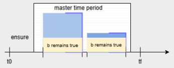
FORML Operators for Expressing Conditions



P ’check at end’ b ....



P ’check anytime’ b ....



P ’ensure’ b ....

P ’check count’ c ’<’ n = ’check’ ({’count’ c ’inside’ P.element < n}) ’over’ P ....

P ’check count’ c ’<’ n = ’check’ b ’over’ P with b = {’count’ c ’inside’ P.element < n} ....

P ’check count’ c ’<=’ n .... checking the number of event occurrences at the end of a time period

P ’check count’ c ’>’ n checking the number of event occurrences at the end of a time period

P ’check count’ c ’>=’ n .... checking the number of event occurrences at the end of a time period

P ’check count’ c ’==’ n .... checking the number of event occurrences at the end of a time period

P ’check count’ c ’<>’ n .... checking the number of event occurrences at the end of a time period

P ’check duration’ b ’>’ d ...... checking the duration of condition Boolean B at the end of a time period

P ’check duration’ b ’<’ d ...... checking the duration of condition Boolean B at the end of a time period

4 Examples

Colors

The built-in keywords are written in blue. User-defined object names are written in orange. Comments are colored in green. Categories names are written in red. TO BE FIXED

Pattern for A Typical Realistic Requirement

```
class TypicalRequirement is ETL union FORM_L
    union {

        type Requirement is Boolean forbid { *,
            +, integrate };

        Boolean inOperation is external;
        Boolean inNormalDomain is external;
        Boolean inBackupDomain is external;
        Boolean inSystemOperatingLife is external
            ;

        Real x, y is external;
        Integer n is external;

        Real p is 0.99;

        /* r1 "During operation, the system
            should stay within its normal domain
            ." */
        Requirement r1 is 'during' inOperation '
            ensure' inNormalDomain;

        // r2 "If the system fails to stay within
             its operating domain, then it
            should not stay outside of its
            normal domain for more than x
            minutes."
        Requirement r2 is 'during' inOperation '
            ensure' (not inNormalDomain 'implies
            ' r2_outside);
        Requirement r2_outside is 'during' [
            inNormalDomain 'becomes false',
            inNormalDomain 'becomes false' + x '
            mn'] 'check at end' b;
        Boolean b is inOperation 'implies'
            inNormalDomain;

        // r3 "The system should not go outside
            its normal domain more than n times
            per year."
        Requirement r3 is 'count' ((b 'becomes
            false') on [b 'becomes false', b '
            becomes false' + 1 'year'] <= n;

        // r4 "If (r1 and r2 and r3) fail, then
            the system should go to its backup
            domain within y minutes as soon as
            the failure is detected."
        Requirement r4 is not (r1 and r2 and r3)
            'implies'
        // R "During system operating life, r1
            and r2 and r3 and r4 should be
            satisfied with a probability of
            success of p%."
        Real prob is estimator Probability (r1
            and r2 and r3 and r4) at
            inSystemOperatingLife 'becomes false
            ';
        Requirement R is 'during'
            inSystemOperatingLife 'check at end
            prob' > p;
};
```

Pumping System with listings

```
'
    // R1: While the system operates, a pump must
      not be started more than twice
Requirement R1 is
    'during' system.inOperation
    'check count' (pump.isStarted 'becomes true')
          '<=' 2;

// R2: At least one hour must separate two
      consecutive pump startups
Requirement R2 is
    'after' pump.isStarted 'for' 1*'h'
    'check count' (pump.isStarted 'becomes true')
          '==' 0;

// R3: While the pump operates, its temperature
      must always stay below 50 C
Requirement R3 is
    'during' pump.isStarted
    'ensure' pump.temp < 50*'degC';
end
'

// R4: While the system operates, after the pump
      temperature rises above 40  C , the
      temperature must not stay above this value
      for a duration of more than 1 mn cumulated
      over the next 15 mn.
Requirement R4 is
    'during' system.inOperation
        'after' pump.temp > 40*degC 'for' 15*'mn'
        'check duration' (pump.temp > 40*'degC')
            '<' 1*'mn';

// R5: While the system operates, there should
      not be a failure with a probability greater
      than 99%
Real pFailure is
    estimator new Probability failure
    at inOperation 'becomes false';
Requirement R5 is
    'during' system.inOperation
    'check at end' pFailure '>' 0.99;
```

Pumping System with Pygmentsfrom

```
# R1: While the system operates, a pump must not be
started more than twice
Requirement R1 is
    'during' system.inOperation
    'check count' (pump.isStarted 'becomes true') '<=' 2;

# R2: At least one hour must separate two consecutive pump
startups
Requirement R2 is
    'after' pump.isStarted 'for' 1*h
    'check count' (pump.isStarted 'becomes true') '==' 0;

# R3: While the pump operates, its temperature must always
stay below 50°C
Requirement R3 is
    'during' pump.isStarted
    'ensure' pump.temp < 50*degC;

# R4: While the system operates, after the pump
temperature rises above 40 °C, the temperature must not
stay above this value for a duration of more than 1 mn
cumulated over the next 15 mn.
Requirement R4 is
    'during' system.inOperation
        'after' pump.temp > 40*degC 'for' 15*mn
        'check duration' (pump.temp > 40*degC) '<' 1*mn;

# R5: While the system operates, there should not be a
failure with a probability greater than 99%
Real pFailure is
    estimator new Probability failure
    at inOperation 'becomes false';
Requirement R5 is
    'during' system.inOperation
    'check at end' pFailure '>' 0.99;
```

TO BE REFORMATTED:CRML v1.2 2023Buffoni et al. 2023, Bouskela et al. 2023

References

[1] Daniel Bouskela et al. "The Common Requirement Modeling Language". In: *Proceedings of the Modelica Conference 2023*. 2023.

[2] Lena Buffoni et al. *Tutorial: CRML A Language for Verifying Realistic Dynamic Requirements*. Tech. rep. MODPROD, 2023. URL: https://github.com/OpenModelica/CRML/tree/main/resources/crml_tutorial.

[3] CRML v1.2. *Specification v1.2 of the Common Requirement Modeling Language*. Tech. rep. ITEA EMBrACE Project, 2023. URL: https://github.com/OpenModelica/CRML/blob/main/language_specification/CRML%20specification_v1.2.pdf.