

The Property Language Manual

Model Checking Contest @ Petri Nets 2020

Contact:

The MCC formula team,
[Loïc Jezequel](#), [Emmanuel Paviot-Adet](#)

Contents

1	Introduction	3
1.1	Scope	3
1.2	Categories of the contest	3
1.3	Outline	4
2	The BNF grammar of the property language	4
2.1	Subcategory <i>GlobalProperties</i>	5
2.2	Subcategory <i>QLDetection</i>	5
2.3	Subcategory <i>StableMarkingDetection</i>	5
2.4	Subcategory <i>Liveness</i>	5
2.5	Subcategory <i>1Safe</i>	6
2.6	Subcategory <i>UpperBounds</i>	6
2.7	Subcategories <i>ReachabilityFireability</i> and <i>ReachabilityCardinality</i>	6
2.8	Subcategories <i>LTLFireability</i> and <i>LTLCardinality</i>	6
2.9	Subcategories <i>CTLFireability</i> and <i>CTLCardinality</i>	7
3	XML format of the property language	8
3.1	Property sets	8
3.2	Properties	8
3.3	Formulæ	9
3.3.1	CTL state operators	9
3.3.2	Path operators	9
3.3.3	Boolean operators	9
3.3.4	Atomic propositions	10
3.4	Integer expressions	10
3.4.1	Arithmetic operators	10
3.4.2	Token-counting integer operator	10
3.5	Bound formulæ	10
3.6	Places and transitions	11

1 Introduction

1.1 Scope

This document is intended as a guide for the developers of the tools participating in the Model Checking Contest @ Petri Nets, edition 2020. It specifically contains:

- The definition of the property language used in the contest this year.
- The definition of the language subset presented in each category of the contest.

1.2 Categories of the contest

The Model Checking Contest @ Petri Nets compares the competing verification tools on six categories of verification goals:

1. State Space generation
2. Global properties analysis
3. Upper bound analysis
4. Reachability analysis
5. LTL analysis
6. CTL analysis

To maximize tool participation, we have further divided some of these categories into subcategories, where only restricted kinds of formulas will be found:

Category	Subcategory	Description
State Space generation	StateSpace	Counting the number of states in a marking graph
Global properties analysis	GlobalProperties	Existence of deadlock states
	QLDetection	Quasi-liveness detection
	StableMarkingDetection	Detection of stable markings
	Liveness	Liveness detection
	1Safe	1-safeness detection
Upper bound analysis	UpperBounds	Computing the exact upper bound of a set of places
Reachability analysis	ReachabilityFireability	Boolean combinations of checks of firability of transitions
	ReachabilityCardinality	Boolean combinations of comparisons of numbers of tokens
LTL analysis	LTLFireability	LTL with atomic propositions on firability of transitions
	LTLCardinality	LTL with atomic propositions on number of tokens
CTL analysis	CTLFireability	CTL with atomic propositions on firability of transitions
	CTLCardinality	CTL with atomic propositions on numbers of tokens

Each subcategory designates the name of an XML file providing, for every benchmark, a set of formulæ for tools competing in this subcategory to solve. Most of the subcategories proposed this year were already proposed last year. Only some global properties analysis are new.

Most of the formulas presented to tools have been randomly generated. Unfortunately, most of the benchmarks submitted to the contest came without example verification properties. Writing them by hand is a very time consuming task (there are currently more than 250 models!). This is a very unfortunate situation but we have not found any way to overcome it.

1.3 Outline

This document presents the specification language in, conceptually, two steps.

- We first use a BNF grammar to describe the overall specification language. This grammar is also used to formally define the class of formulas that will be presented to the tools participating on each subcategory of the contest ([Section 2](#)).
- Second, the document defines the XML syntax of the files that will actually be passed to the tools ([Section 3](#)).

2 The BNF grammar of the property language

Every property presented to a tool participating in any subcategory of the contests can be derived from the following general grammar of the property language:

```

⟨formula⟩ ::= ⟨boolean-formula⟩
           | ⟨bound-formula⟩

⟨boolean-formula⟩ ::= A ⟨boolean-formula⟩
                  | E ⟨boolean-formula⟩
                  | G ⟨boolean-formula⟩
                  | F ⟨boolean-formula⟩
                  | X ⟨boolean-formula⟩
                  | ⟨boolean-formula⟩ U ⟨boolean-formula⟩
                  | ¬ ⟨boolean-formula⟩
                  | ⟨boolean-formula⟩ ∧ ⟨boolean-formula⟩
                  | ⟨boolean-formula⟩ ∨ ⟨boolean-formula⟩
                  | ∃ p ⟨boolean-formula⟩
                  | ∀ p ⟨boolean-formula⟩
                  | ∃ t ⟨boolean-formula⟩
                  | ∀ t ⟨boolean-formula⟩
                  | ∃ x ⟨boolean-formula⟩
                  | ∀ x ⟨boolean-formula⟩
                  | ⟨atom⟩

⟨atom⟩ ::= deadlock
         | is-fireable( $t_1, \dots, t_n$ )
         | ⟨integer-expression⟩ ≤ ⟨integer-expression⟩
         | ⟨integer-expression⟩ = ⟨integer-expression⟩

⟨integer-expression⟩ ::= ⟨integer-expression⟩ + ⟨integer-expression⟩
                       | ⟨integer-expression⟩ − ⟨integer-expression⟩
                       | Integer constant
                       | x
                       | token-count( $p_1, \dots, p_n$ )

⟨bound-formula⟩ ::= place-bound( $p_1, \dots, p_n$ )

```

The property language considers two kinds of formulas: *Boolean formulas*, evaluating to a Boolean value, and *bound formulas*, which evaluate to a natural number. Only the subcategory *UpperBounds* employs *bound formulas*, all other subcategories use only *Boolean formulas*.

Operators **A** and **E** are the standard CTL operators. Operators **G**, **F**, **X**, **U** are the standard LTL operators. We define that the **X** operator evaluate to false if no successor state exists. A *Boolean formula* can also contain Boolean combinations of Boolean formulas. The \exists and \forall operator quantify over places, transitions or an integer value x . Atomic propositions are either

- **deadlock**, asking that the current state enables no transition,
- **is-fireable**(t_1, \dots, t_n), which holds iff either t_1 , or t_2 , or \dots or t_n are enabled at the current state, or
- (in)equalities of the form $\langle expr \rangle = \langle expr \rangle$ or $\langle expr \rangle \leq \langle expr \rangle$, where $\langle expr \rangle$ is an integer expression.

Integer expressions can contain additions and subtractions of *Integer constants*, always taken from the set $\{1, 2, 3\}$, and the following operator:

- **token-count**(p_1, \dots, p_n), which returns, for the current marking, the exact number of tokens contained in the set $\{p_1, \dots, p_n\}$ of places.

Bound formulas are formed only by atoms **place-bound**(\cdot), whose semantics is defined as follows:

- The atom **place-bound**(p_1, \dots, p_n) evaluates to some natural number c so that c is the maximum number of tokens that any reachable marking can put in all n places p_1, p_2, \dots, p_n at the same time.

Note that, in fact, in the current edition of the MCC no subcategory of the contest will contain formulas using additions and subtractions of integer expressions. These operators remain here only for future use.

This grammar defines a large class of formulæ. Each subcategory of the contest contains formulas of only some specific fragment of this grammar. The following subsections define the specific syntax of the formulæ that tools participating in a given subcategory shall expect to be provided with.

[Section 3](#) presents the correspondence between the above BNF grammar and the syntax of the XML files that will actually be passed to the tools.

2.1 Subcategory *GlobalProperties*

Tools participating in this subcategory will only find one formula to solve for every benchmark:

$\langle formula \rangle ::= \langle boolean-formula \rangle$

$\langle boolean-formula \rangle ::= \mathbf{E F deadlock}$

2.2 Subcategory *QLDetection*

Tools participating in this subcategory will only find one formula to solve for every benchmark:

$\langle formula \rangle ::= \langle boolean-formula \rangle$

$\langle boolean-formula \rangle ::= \forall t \mathbf{E F is-fireable}(t)$

2.3 Subcategory *StableMarkingDetection*

Tools participating in this subcategory will only find one formula to solve for every benchmark:

$\langle formula \rangle ::= \langle boolean-formula \rangle$

$\langle boolean-formula \rangle ::= \exists p \exists x \mathbf{A G token-count}(p) = x$

2.4 Subcategory *Liveness*

Tools participating in this subcategory will only find one formula to solve for every benchmark:

$\langle formula \rangle ::= \langle boolean-formula \rangle$

$\langle boolean-formula \rangle ::= \forall t \mathbf{A G E F is-fireable}(t)$

2.5 Subcategory *1Safe*

Tools participating in this subcategory will only find one formula to solve for every benchmark:

$\langle \text{formula} \rangle ::= \langle \text{boolean-formula} \rangle$

$\langle \text{boolean-formula} \rangle ::= \forall p \text{ A G token-count}(p) \leq 1$

2.6 Subcategory *UpperBounds*

Observe that this is the only subcategory where formulas evaluate to something else than a Boolean value, in particular to a natural number. See the beginning of this section for the semantics of the atom **place-bound**(·).

$\langle \text{formula} \rangle ::= \langle \text{bound-formula} \rangle$

$\langle \text{bound-formula} \rangle ::= \text{place-bound}(p_1, \dots, p_n)$

2.7 Subcategories *ReachabilityFireability* and *ReachabilityCardinality*

Both subcategories contain formulas produced using the same grammar, the only difference being the atomic propositions allowed in each one. The grammar is the following:

$\langle \text{formula} \rangle ::= \langle \text{boolean-formula} \rangle$

$\langle \text{boolean-formula} \rangle ::= \text{E F } \langle \text{state-formula} \rangle$
 $\quad \quad \quad | \text{A G } \langle \text{state-formula} \rangle$

$\langle \text{state-formula} \rangle ::= \neg \langle \text{state-formula} \rangle$
 $\quad \quad \quad | \langle \text{state-formula} \rangle \wedge \langle \text{state-formula} \rangle$
 $\quad \quad \quad | \langle \text{state-formula} \rangle \vee \langle \text{state-formula} \rangle$
 $\quad \quad \quad | \langle \text{atom} \rangle$

Now, the subcategory *ReachabilityFireability* only contains **is-fireable**(·) atoms:

$\langle \text{atom} \rangle ::= \text{is-fireable}(t_1, \dots, t_n)$

while the formulas in the *ReachabilityCardinality* subcategory uniquely contain integer inequalities:

$\langle \text{atom} \rangle ::= \langle \text{integer-expression} \rangle \leq \langle \text{integer-expression} \rangle$

$\langle \text{integer-expression} \rangle ::= \text{Integer constant}$
 $\quad \quad \quad | \text{token-count}(p_1, \dots, p_n)$

Recall that the *Integer constant* will be a number in $\{1, 2, 3\}$.

2.8 Subcategories *LTLFireability* and *LTLCardinality*

Both subcategories contain formulas produced using the same grammar, the only difference being the atomic propositions allowed in each one. The grammar is the following:

$\langle \text{formula} \rangle ::= \langle \text{boolean-formula} \rangle$

$\langle \text{boolean-formula} \rangle ::= \text{A } \langle \text{path-formula} \rangle$

$$\begin{aligned}
\langle \text{path-formula} \rangle & ::= \mathbf{G} \langle \text{path-formula} \rangle \\
& | \mathbf{F} \langle \text{path-formula} \rangle \\
& | \mathbf{X} \langle \text{path-formula} \rangle \\
& | \langle \text{path-formula} \rangle \mathbf{U} \langle \text{path-formula} \rangle \\
& | \neg \langle \text{path-formula} \rangle \\
& | \langle \text{path-formula} \rangle \wedge \langle \text{path-formula} \rangle \\
& | \langle \text{path-formula} \rangle \vee \langle \text{path-formula} \rangle \\
& | \langle \text{atom} \rangle
\end{aligned}$$

Now, the subcategory *LTLFireability* only contains **is-fireable**(\cdot) atoms:

$$\langle \text{atom} \rangle ::= \mathbf{is-fireable}(t_1, \dots, t_n)$$

while the formulas in the *LTLCardinality* subcategory only contain integer inequalities:

$$\langle \text{atom} \rangle ::= \langle \text{integer-expression} \rangle \leq \langle \text{integer-expression} \rangle$$

$$\begin{aligned}
\langle \text{integer-expression} \rangle & ::= \text{Integer constant} \\
& | \mathbf{token-count}(p_1, \dots, p_n)
\end{aligned}$$

2.9 Subcategories *CTLFireability* and *CTLCardinality*

Both subcategories contain formulas produced using the same grammar, the only difference being the atomic propositions allowed in each one. The grammar is the following:

$$\begin{aligned}
\langle \text{formula} \rangle & ::= \langle \text{boolean-formula} \rangle \\
\langle \text{boolean-formula} \rangle & ::= \mathbf{A} \langle \text{path-formula} \rangle \\
& | \mathbf{E} \langle \text{path-formula} \rangle \\
& | \neg \langle \text{boolean-formula} \rangle \\
& | \langle \text{boolean-formula} \rangle \wedge \langle \text{boolean-formula} \rangle \\
& | \langle \text{boolean-formula} \rangle \vee \langle \text{boolean-formula} \rangle \\
& | \langle \text{atom} \rangle \\
\langle \text{path-formula} \rangle & ::= \mathbf{G} \langle \text{boolean-formula} \rangle \\
& | \mathbf{F} \langle \text{boolean-formula} \rangle \\
& | \mathbf{X} \langle \text{boolean-formula} \rangle \\
& | \langle \text{boolean-formula} \rangle \mathbf{U} \langle \text{boolean-formula} \rangle
\end{aligned}$$

The subcategory *CTLFireability* only contains **is-fireable**(\cdot) atoms:

$$\langle \text{atom} \rangle ::= \mathbf{is-fireable}(t_1, \dots, t_n)$$

while the formulas in the *CTLCardinality* subcategory contain integer inequalities:

$$\langle \text{atom} \rangle ::= \langle \text{integer-expression} \rangle \leq \langle \text{integer-expression} \rangle$$

$$\begin{aligned}
\langle \text{integer-expression} \rangle & ::= \text{Integer constant} \\
& | \mathbf{token-count}(p_1, \dots, p_n)
\end{aligned}$$

3 XML format of the property language

During the competition, tools will be provided, for every benchmark and every subcategory of the contest, with one XML file containing the set of formulas to solve for that subcategory and that benchmark.

The purpose of this section is defining the XML Schema describing the structure of the XML files containing the formulas. In fact, we describe a RelaxNG representation of the XML Schema.

Each XML file will be provided together with a plain text file describing the formula in a more readable way. The format of this text file is not guaranteed to be preserved in future editions of the contest. This file is just provided for readability.

As an introducing example, consider the following property, given in XML (left) and the corresponding to a set of formulas containing only one formula (right) generated by the grammar in [Section 2](#).

```
<?xml version="1.0"?>
<property-set xmlns="http://mcc.lip6.fr/">
  <property>
    <id>Dekker-PT-010-test1</id>
    <description>
      Automatically generated formula.
    </description>
    <formula>
      <exists-path>
        <until>
          <before>
            <negation>
              <integer-le>
                <integer-constant>1</integer-constant>
                <tokens-count>
                  <place>p3_0</place>
                </tokens-count>
              </integer-le>
            </negation>
          </before>
          <reach>
            <is-fireable>
              <transition>withdraw_4_8</transition>
            </is-fireable>
          </reach>
        </until>
      </exists-path>
    </formula>
  </property>
</property-set>
```

$E((\neg(1 \leq \text{token-count}(p3_0))) \cup \text{is-fireable}(\text{withdraw_4_8}))$

What follows is the description of the RelaxNG grammar describing the XML files provided to the tools.

3.1 Property sets

The `property-set` element is the root of the XML representation. It contains one or more properties.

```
default namespace = "http://mcc.lip6.fr/"
start = property-set

property-set = element property-set {
  property*
}
```

3.2 Properties

A property is composed of three mandatory parts: a unique identifier, a textual description of the property, and the formula itself.

```
property = element property {
  element id {
    xsd:ID
  } &
  element description {
    text
  } &
  element formula {
    formula
  }
}
```


3.3 Formulæ

Formulæ are the body of properties. They define what is expected to hold on the model. Formulæ are currently of two main types: formulæ that return integers, and formulæ that return Booleans.

In the following, for each rule of the grammar defining formulæ we give the RelaxNG representation (on the left) as well as the corresponding part of the BNF grammar (on the right).

<pre>formula = boolean-formula bound-formula</pre>	$\langle formula \rangle$	$::= \langle boolean-formula \rangle$ $ \langle bound-formula \rangle$
--	---------------------------	--

3.3.1 CTL state operators

These operators correspond to the **A** and **E** operators of CTL.

<pre>boolean-formula = ... element all-paths { boolean-formula } element exists-path { boolean-formula } ...</pre>	$\langle boolean-formula \rangle$	$::= \mathbf{A} \langle boolean-formula \rangle$ $ \mathbf{E} \langle boolean-formula \rangle$ $ \dots$
--	-----------------------------------	---

3.3.2 Path operators

These operators are the **G**, **F**, **X**, and **U** operators of CTL and LTL Recall that the `next` operator should evaluate to false if no successor state exists.

<pre>boolean-formula = ... element globally { boolean-formula } element finally { boolean-formula } element next { boolean-formula & } element until { element before { boolean-formula } & element reach { boolean-formula } & } ...</pre>	$\langle boolean-formula \rangle$	$::= \mathbf{G} \langle boolean-formula \rangle$ $ \mathbf{F} \langle boolean-formula \rangle$ $ \mathbf{X} \langle boolean-formula \rangle$ $ \langle boolean-formula \rangle \mathbf{U} \langle boolean-formula \rangle$ $ \dots$
---	-----------------------------------	---

3.3.3 Boolean operators

These are usual Boolean operators.

<pre>boolean-formula = ... element negation { boolean-formula } element conjunction { boolean-formula, boolean-formula+ } element disjunction { boolean-formula, boolean-formula+ } ...</pre>	$\langle boolean-formula \rangle$	$::= \neg \langle boolean-formula \rangle$ $ \langle boolean-formula \rangle \wedge \langle boolean-formula \rangle$ $ \langle boolean-formula \rangle \vee \langle boolean-formula \rangle$ $ \dots$
---	-----------------------------------	---

3.3.4 Atomic propositions

There are three types of atomic propositions. Recall that `deadlock` evaluates to true if the current state is a deadlock (has no successor) and `is-fireable` evaluates to true if one of the set of transitions given is fireable from the current state.

```
boolean-formula =
  ...
  | element deadlock { empty }
  | element is-fireable {
    transition+
  }
  | element integer-le {
    integer-expression,
    integer-expression
  }
  | ...
```

$\langle \text{boolean-formula} \rangle ::= \langle \text{atom} \rangle \mid \dots$

$\langle \text{atom} \rangle ::= \text{deadlock}$
 $\quad \mid \text{is-fireable}(t_1, \dots, t_n)$
 $\quad \mid \langle \text{integer-expression} \rangle \leq \langle \text{integer-expression} \rangle$

3.4 Integer expressions

3.4.1 Arithmetic operators

These are usual arithmetic operators for integers.

```
integer-expression =
  ...
  | element integer-constant {
    xsd:integer
  }
  | element integer-sum {
    integer-expression,
    integer-expression+
  }
  | element integer-difference {
    integer-expression,
    integer-expression
  }
  | ...
```

$\langle \text{integer-expression} \rangle ::= \text{Integer constant}$
 $\quad \mid \langle \text{integer-expression} \rangle + \langle \text{integer-expression} \rangle$
 $\quad \mid \langle \text{integer-expression} \rangle - \langle \text{integer-expression} \rangle$
 $\quad \mid \dots$

3.4.2 Token-counting integer operator

```
integer-expression =
  ...
  | element tokens-count {
    place+
  }
  | ...
```

$\langle \text{integer-expression} \rangle ::= \text{token-count}(p_1, \dots, p_n)$
 $\quad \mid \dots$

Recall that

- `tokens-count` returns the exact number of tokens in a set of places.

3.5 Bound formulæ

```
bound-formula =
  element place-bound {
    place+
  }
```

$\langle \text{bound-formula} \rangle ::= \text{place-bound}(p_1, \dots, p_n)$

Recall that

- for a set of places, `place-bound` returns some natural number c so that c is the maximum number of tokens that any reachable marking can put in all places of the set.

3.6 Places and transitions

Places and transitions are uniquely identified. The identifiers occurring at the formula XML files are the `id`'s present in the PNML model.

```
place =  
  element place {  
    xsd:IDREF  
  }  
  
transition =  
  element transition {  
    xsd:IDREF  
  }
```