The Property Language Manual

Model Checking Contest @ Petri Nets 2016

Contact:

The MCC formula team,
César Rodríguez, Loïg Jezequel, Emmanuel Paviot-Adet

v. Feb. 3, 2016

Contents

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 2016. 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 four categories of verification goals:

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

To maximize tool participation, we have further divided the last 4 categories into 8 subcategories, where only restricted kinds of formulas will be found:

Subcategory	Description
UpperBounds	Computing the exact upper bound of a set of places
ReachabilityDeadlock	Existence of deadlock states
ReachabilityFireability	Boolean combinations of propositions checking firability of transitions
ReachabilityCardinality	Boolean combinations of propositions comparing the number of tokens in places
LTLFireability	Full LTL with atomic propositions checking firability of transitions
LTLCardinality	Full LTL with atomic propositions comparing the number of tokens in places
CTLFireability	Full CTL with atomic propositions checking firability of transitions
CTLCardinality	Full CTL with atomic propositions comparing the number of tokens in places

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 8 subcategories proposed this year were already proposed last year.[1]

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).

[1] With respect to previous year we have removed three subcategories, all those named **Simple*, see instructions from the MCC'15. We have also unified into the category UpperBounds the subcategories where upper bounds of places needed to be computed.

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*⟩
| ⟨*atom*⟩

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

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

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

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. Atomic propositions are either

- **deadlock**, asking that the current state enables no transition,
- **is-fireable**$(t_1, \ldots, t_n)$, which holds iff either t_1, or t_2, or ... or t_n are enabled at the current state, or
- inequalities of the form ⟨*expr*⟩ ≤ ⟨*expr*⟩, where ⟨*expr*⟩ 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**$(p_1, \ldots, p_n)$, which returns, for the current marking, the exact number of tokens contained in the set $\{p_1, \ldots, p_n\}$ of places.

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

- The atom **place-bound**$(p_1, \ldots, p_n)$ 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 $p_1, p_2, \ldots, p_n$ 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 *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**$(\cdot)$.

⟨*formula*⟩ ::= ⟨*bound-formula*⟩

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

2.2 Subcategory *ReachabilityDeadlock*

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

⟨*formula*⟩ ::= ⟨*boolean-formula*⟩

⟨*boolean-formula*⟩ ::= **E F deadlock**

2.3 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:

⟨*formula*⟩ ::= ⟨*boolean-formula*⟩

⟨*boolean-formula*⟩ ::= **E F** ⟨*state-formula*⟩
| **A G** ⟨*state-formula*⟩

⟨*state-formula*⟩ ::= $\neg$ ⟨*state-formula*⟩
| ⟨*state-formula*⟩ $\wedge$ ⟨*state-formula*⟩
| ⟨*state-formula*⟩ $\vee$ ⟨*state-formula*⟩
| ⟨*atom*⟩

Now, the subcateory *ReachabilityFireability* only contains **is-fireable**$(\cdot)$ atoms:

⟨*atom*⟩ ::= **is-fireable**$(t_1, \ldots, t_n)$

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

⟨*atom*⟩ ::= ⟨*integer-expression*⟩ $\leq$ ⟨*integer-expression*⟩

⟨*integer-expression*⟩ ::= *Integer constant*
| **token-count**$(p_1, \ldots, p_n)$

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

2.4 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:

⟨*formula*⟩ ::= ⟨*boolean-formula*⟩

⟨*boolean-formula*⟩ ::= **A** ⟨*path-formula*⟩

⟨*path-formula*⟩ ::= **G** ⟨*path-formula*⟩
| **F** ⟨*path-formula*⟩
| **X** ⟨*path-formula*⟩
| ⟨*path-formula*⟩ **U** ⟨*path-formula*⟩
| ⟨*atom*⟩

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

⟨*atom*⟩ ::= **is-fireable**$(t_1, \ldots, t_n)$

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

⟨*atom*⟩ ::= ⟨*integer-expression*⟩ $\leq$ ⟨*integer-expression*⟩

⟨*integer-expression*⟩ ::= *Integer constant*
| **token-count**$(p_1, \ldots, p_n)$

2.5 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:

⟨*formula*⟩ ::= ⟨*boolean-formula*⟩

⟨*boolean-formula*⟩ ::= **A** ⟨*path-formula*⟩
| **E** ⟨*path-formula*⟩
| $\neg$ ⟨*boolean-formula*⟩
| ⟨*boolean-formula*⟩ $\wedge$ ⟨*boolean-formula*⟩
| ⟨*boolean-formula*⟩ $\vee$ ⟨*boolean-formula*⟩
| ⟨*atom*⟩

⟨*path-formula*⟩ ::= **G** ⟨*boolean-formula*⟩
| **F** ⟨*boolean-formula*⟩
| **X** ⟨*boolean-formula*⟩
| ⟨*boolean-formula*⟩ **U** ⟨*boolean-formula*⟩

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

⟨*atom*⟩ ::= **is-fireable**$(t_1, \ldots, t_n)$

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

⟨*atom*⟩ ::= ⟨*integer-expression*⟩ $\leq$ ⟨*integer-expression*⟩

⟨*integer-expression*⟩ ::= *Integer constant*
| **token-count**$(p_1, \ldots, p_n)$

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 ((¬ (1 ≤ **token-count**(p3_0))) U **is-fireable**(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).

```
formula =
    boolean_formula
  | bound_formula
```

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

3.3.1 CTL state operators

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

```
boolean_formula =
    ...
  | element all_paths {
      boolean_formula
    }
  | element exists_path {
      boolean_formula
    }
  | ...
```

⟨*boolean-formula*⟩ ::= **A** ⟨*boolean-formula*⟩
| **E** ⟨*boolean-formula*⟩
| ...

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.

```
boolean_formula =
    ...
  | element globally {
      boolean_formula
    }
  | element finally {
      boolean_formula
    }
  | element next {
      boolean_formula &
  | element until {
      element before {
        boolean_formula
      } &
      element reach {
        boolean_formula
      } &
    }
  | ...
```

⟨*boolean-formula*⟩ ::= **G** ⟨*boolean-formula*⟩
| **F** ⟨*boolean-formula*⟩
| **X** ⟨*boolean-formula*⟩
| ⟨*boolean-formula*⟩ **U** ⟨*boolean-formula*⟩
| ...

3.3.3 Boolean operators

These are usual Boolean operators.

```
boolean_formula =
    ...
  | element negation {
      boolean_formula
    }
  | element conjunction {
      boolean_formula,
      boolean_formula+
    }
  | element disjunction {
      boolean_formula,
      boolean_formula+
    }
  | ...
```

⟨*boolean-formula*⟩ ::= ¬ ⟨*boolean-formula*⟩
| ⟨*boolean-formula*⟩ ∧ ⟨*boolean-formula*⟩
| ⟨*boolean-formula*⟩ ∨ ⟨*boolean-formula*⟩
| ...

3.3.4 Atomic propositions

There is 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
    }
  | ...
```

⟨*boolean-formula*⟩ ::= ⟨*atom*⟩ | ...

⟨*atom*⟩ ::= **deadlock**
| **is-fireable**$(t_1, \ldots, t_n)$
| ⟨*integer-expression*⟩ $\leq$ ⟨*integer-expression*⟩

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
    }
  | ...
```

⟨*integer-expression*⟩ ::= *Integer constant*
| ⟨*integer-expression*⟩ + ⟨*integer-expression*⟩
| ⟨*integer-expression*⟩ − ⟨*integer-expression*⟩
| ...

3.4.2 Token-counting integer operator

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

⟨*integer-expression*⟩ ::= **token-count**$(p_1, \ldots, p_n)$
| ...

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+
    }
```

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

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
    }
```