Goals of the property language

The property language for the MCC @ Petri Nets is a language designed to allow participation of many tools. It allows to write structural, reachability, CTL, LTL formulæ. It is tightly related to Petri nets, that are the modeling formalism used during the contest.

This language is designed to evolve in the future editions of the contest. It is also designed to integrate with Petri Net Markup Language in the future. To do so, the language is provided as a RelaxNG grammar and an XML Schema generated from it.

Contents

1 Description of the langage

The property language is designed as a RelaxNG grammar. Thus, properties as given in XML. We generate a text equivalent from it, but it is not stable and only given for readability.

```
default namespace = "http://mcc.lip6.fr/"
start = property_set
```

1.1 Property sets

The `property_set` element is the root of the XML representation. It contains several properties.

```
default namespace = "http://mcc.lip6.fr/"
start = property_set

property_set = element property_set {
  property*
}
```

1.2 Properties

A property is composed of several madatory parts: a unique identifier, a textual description of the property, and a formula. It has a set of tags to give a hint about the class of tools that can compute the formula (structural, reachability, CTL or LTL). These hints do *not* say that the formula is expressed exactly in the language given by the hint, but tells that given minor changes, it could be.

A property also contains an optional part for the expected result. It gives the expected value and a textual explanation.

```
property = element property {
  element id {
    xsd:ID
  } &
  element description {
    text
  } &
  element tags {
    element is_structural { xsd:boolean } &
    element is_reachability  { xsd:boolean } &
    element is_ctl { xsd:boolean } &
    element is_ltl { xsd:boolean }
  } &
  element expected_result {
    element value {
      xsd:integer | xsd:boolean
    } &
    element explanation {
      text
    }
  }? &
  element formula {
    formula
  }
}
```

1.3 Formulæ

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

```
formula =
    boolean_formula
  | integer_formula
```

1.4 Boolean formulæ

Boolean formulæ are the majority of available formulæ. We difen them in several parts.

```
boolean-formula = ...
```

1.4.1 Reachability operators

```
boolean-formula =
    ...
  | element invariant {
      boolean-formula
    }
  | element impossibility {
      boolean-formula
    }
  | element possibility {
      boolean-formula
    }
  | ...
```

- `invariant` evaluates to true if its subformula is verified for all states of the system;
- `impossibility` evaluates to true if its subformula is never verified for all states of the system;
- `possibility` evaluates to true if its subformula is verified for some states of the system (at least one).

1.4.2 State operators

These operators are the `A` and `E` operators of CTL.

```
boolean-formula =
    ...
  | element all-paths {
      boolean-formula
    }
  | element exists-path {
      boolean-formula
    }
  | ...
```

1.4.3 Path operators

These operators are the `X`, `G`, `F` and `U` operators of CTL and LTL.

```
boolean-formula =
    ...
  | element globally {
      boolean-formula
    }
  | element finally {
      boolean-formula
    }
  | element next {
      boolean-formula &
      element if-no-successor { xsd:boolean } &
      element steps { xsd:positiveInteger }
    }
  | element until {
      element before {
        boolean-formula
      } &
      element reach {
        boolean-formula
      } &
      element strength {
        "weak" | "strong"
      }
    }
  | ...
```

- `next` evaluates to true if its subformula is verified `steps` states after along the path from the current state; the `if-no-successor` value is the value to return if the successor state does not exist;
- `until` has a `strength` modifier, allowing the weak until.

1.4.4 Petri net operators

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

- `deadlock` evaluates to true if the current state is in dealock (has no successor);
- `is-fireable` evaluates to true if one of the set of transitions given is fireable from the current state.

1.4.5 Boolean operators

These are usual Boolean operators.

```
boolean-formula =
    ...
  | element true  { empty }
  | element false { empty }
  | element negation {
      boolean-formula
    }
  | element conjunction {
      boolean-formula,
      boolean-formula+
    }
  | element disjunction {
      boolean-formula,
      boolean-formula+
    }
  | element exclusive-disjunction {
      boolean-formula,
      boolean-formula+
    }
  | element implication {
      boolean-formula,
      boolean-formula
    }
  | element equivalence {
      boolean-formula,
      boolean-formula+
    }
  | ...
```

1.4.6 Comparison operators

These are integer comparison operators.

```
boolean-formula =
    ...
  | element integer-eq {
      integer-expression,
      integer-expression
    }
  | element integer-ne {
      integer-expression,
      integer-expression
    }
  | element integer-lt {
      integer-expression,
      integer-expression
    }
  | element integer-le {
      integer-expression,
      integer-expression
    }
```

```
  | element integer-gt {
      integer-expression,
      integer-expression
    }
  | element integer-ge {
      integer-expression,
      integer-expression
    }
  | ...
```

1.5 Integer formulæ

An integer formula is an integer expression. The tool must return the integer, that is the result of the expression.

```
integer-formula =
    integer-expression

integer-expression = ...
```

1.5.1 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-product {
    integer-expression,
    integer-expression+
  }

| element integer-difference {
    integer-expression,
    integer-expression
  }

| element integer-division {
    integer-expression,
    integer-expression
  }
  | ...
```

1.5.2 Petri net operators

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

- `place-bound` returns the exact of estimated bound of a set of places; for several places, it means the maximum number of tèkens in all these places at the same time;
- `tokens-count` returns the exact number of tokens in a set of places.

1.6 Places and Transitions

Places and transitions are uniquely identified. The identifiers are those of the PNML file.

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

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

2 Tools

The RelaxNG grammar can be downloaded from `http://mcc.lip6.fr/properties/mcc-properties.rnc` using:

```
wget http://mcc.lip6.fr/properties/mcc-properties.rnc
```

The XML schema file can be downloaded from `http://mcc.lip6.fr/properties/mcc-properties.xsd` using:

```
wget http://mcc.lip6.fr/properties/mcc-properties.xsd
```

2.1 How to generate the XML schema from the RelaxNG grammar?

The Trang tool is able to transform the RelaxNG grammar into an XML Schema. Visit `http://www.thaiopensource.com/relaxng/trang.html` to install this tool.

```
trang -I rnc -O xsd mcc-properties.rnc mcc-properties.xsd
```

2.2 How to generate C++ classes from the XML Schema?

Generation of C++ classes requires Code Synthesis' xsd tool (`http://www.codesynthesis.com/products/xsd/`). This tool converts the XML Schema of the property language to a set of C++ classes, an XML validating parser, and an XML output. This tool is free software, and is available for numerous platforms. It is available at `http://www.codesynthesis.com/products/xsd/download.xhtml`. Parsing and validating XML also requires to install Xerces-C++, available at `http://xerces.apache.org/xerces-c/`.

After installing the `xsd` tool, you have to fix the file `xsd/cxx/zc-istream.txx`:

```
35c35
<       setg (b, b, e);
---
>       this->setg (b, b, e);
```

The conversion from the XML Schema to C++ classes is then performed using the following command:

```
mkdir -p src/cxx/
xsd cxx-tree \
    --generate-serialization \
    --generate-doxygen \
    --generate-ostream \
    --generate-comparison \
    --generate-detach \
    --generate-default-ctor \
    --generate-polymorphic --polymorphic-type-all \
    --namespace-map http://mcc.lip6.fr=mcc \
    --output-dir src/cxx/ \
    --root-element property-set \
    mcc-properties.xsd
```

2.3 How to generate Java classes from the XML Schema?

Conversion from the XML Schema to Java classes requires the Java Architecture for XML Binding (JAXB – `http://jaxb.java.net/`). It is included in recent Java distributions.

To generate the classes, use the following command:

```
mkdir -p src/java/
xjc -d src/java/ -p mcc mcc-properties.xsd
```

It generates a set of Java files in the `java` directory.

2.4 How to generate Python classes from the XML Schema?

The python script generateDS (`http://www.rexx.com/~dkuhlman/generateDS.html`) generates Python code from the XML Schema.

```
mkdir -p src/python/
python generateDS.py -m -f --silence -o src/python/mcc-properties.py mcc-properties.xsd
```

2.5 How to generate C# classes from the XML Schema?

There seems to be also tools for C# developers:
`http://stackoverflow.com/questions/386155/comparison-of-xsd-codegenerators-c`. We did not test them, but are interested by feedback if you use one.