Short Report – Milestone 2

Alex Coudray
Kerian Thuillier

September 2019

1 Grammar Specificity

Our grammar let us write Boolean operations in an infix way: (ab) or $!a$. Each operator (except *not* – !) is considered as binary operators working as follows: "(" + *expression* + *operator* + *expression* + ")". Parentheses are a part of the syntax. The operator *not* is an operator with only one argument (an expression).

A constraint of our grammar is that we could only formulate binary operation or simple operation, *i.e.* an operator took only 1 (for *not*) or 2 arguments. Therefore it is possible to write more complex operators from binary ones :

$$a \wedge b \wedge c \rightarrow (a \wedge (b \wedge c))$$

With this syntax we could be sure of operations priority.

Examples:

$$a \wedge b \rightarrow (a \;\; b)$$
$$a \vee b \rightarrow (a \;||\; b)$$
$$a \; exclude \; b \rightarrow (a \;!\; b)$$
$$ab \rightarrow (a \;\; ==> \;\; b)$$
$$ab \rightarrow (a \;\; <=> \;\; b)$$
$$not \; a \rightarrow !a$$

where a, b are Boolean expressions or atomic values.

2 Model Comparison

There are two differences between meta-models **MM1** design by ourselves and **MM2** generated from our concrete syntax.

Firstly, we put in **MM1** two enumerations containing the available operator's types in order to only use valid ones when defining an operation. Whereas, **MM2** do not have those enumeration and considerate operators' types as *String* – which means than any *String* can be a valid operator's type.

Secondly, in **MM1** the *Unop* object (representing simple operation) has a reference links to exactly 1 *Expression*. It is the same for the *Binop* object (representing binary operations) which have two references, each link to exactly 1 *Expression*. While the generated meta-model **MM2** let those references be linked to 0 or 1 *Expression*.

3 Grammar Comparison

Now, let's compare our grammar **GR2** with the generated one **GR1** (grammar generated from meta-model **MM1**).

The first thing that we can note is that even though *BinopType* and *UnopType* are generated, they are not used by **GR1**. We do not understand why, but *Unop* and *Binop* are not generated. So, for $GR1$ an expression can only be an atomic value.

The second one is that each type has the current format :

type_name { attributes_name attributes_values }

which is not really user-friendly for the case of Boolean formulas. Our grammar *GR2* use infix notation for every binary operation, this syntax is closer to the mathematical syntax of Boolean formulas.