ASTMA! TUTORIAL

Erica Santosaputri [1]

[1] School of Information Technology, Bond University

EXTENDED TUTORIAL ABSTRACT

This tutorial provides an introduction to AsTMa!, a topic map constraint language. The language allows topic map authors to define the rules to be enforced towards topic map documents. The language uses logic operators and patterns based on the topic map paradigm. This tutorial is prepared for those with basic understand in the Topic Maps concept as well as the topic map authoring language. AsTMa! is bundled with an automated validator to ensure that the topic map document created follows the rules defined.

AsTMa! is built as an extension of AsTMa= (the authoring language), featuring logic operators like NOT, AND, and OR, simple logical quantifiers and regular expressions. AsTMa! also defines open and closed patterns. In open maplet patterns, denoted by brackets pair […], additional information is allowed; while in closed maplet patterns, denoted by reversed brackets pair]…[, an exact match is required. Three types of constraints that can be defined are: existence quantified constraints, conditional quantified constraints, and Boolean quantified constraints.

Existence quantified constraints ensures that there has to be at least one part within the topic map document that satisfies the rules. Example:

Scenario: A topic map is to be constructed with main topic *programming language*s. Within the document, we need to ensure that the language *java* is included, which is a type of *programming language*, and it has a basename *Java Language*. This constraint is written in AsTMa! as:

```
exists [ java (programming-language)
         bn: Java Language ]
```

The brackets pair […] shows that it is an *open* maplet pattern constraint, where additional information is allowed. Therefore the following topic map document (written in AsTMa=) conforms the constraint:

```
java (programming-language)
bn: Java Language
in: Java is a programming language that has been used extensively
```

Associations in topic map may also be constrained. Here is an example of such a constraint:

```
exists ] (compiler-for-the-language)
         compiler: jbuilder
         language: java [
```

The above constraint ensures that the association of type *compiler-for-the-language* exists in the topic map document, and the association *must* have two players (indicated by the closed maplet pattern). The player *jbuilder* plays the role *compiler*, and the other player *java* plays the role *language*, no additional members are allowed here.

Conditional quantified constraints allow constraints to be enforced to submaps rather than the whole topic map document. The keyword used to denote the first condition clause is *forall*, and it has to be ended by an existence quantified constraint with the keyword *exists.*

Here is an example of a conditional quantified constraint:

```
forall [ $a (programming-language)
         bn: /language/i ]
```

```
=> exists [ (compiler-for-the-language)
            compiler: *
            langugage: $a ]
```

Note the use of the variable *$a* in the above constraint. Variables in AsTMa ! must start with a '$' sign and these variables may be used to mediate matched values between clauses. The first clause with the keyword *forall* will try to match all possible submaps, using these submaps the inner clauses will be evaluated.

AsTMa! also allows the use of *structural variables.* These variables will be bound to submaps which satisfy the constraints. Here is an example of structural variables being used:

```
forall $a [ (compiler-for-the-language) ]
=> exists $a ] (compiler-for-the-language)
               compiler: jbuilder | visualj
               language: java [
```

The variable $a will be bound to a submap containing the matched association. The validation of the inner constraint will then be applied to that submap only, not to whole topic map. The inner clause will check whether within the same submap $a, there exists an association as defined by the existence quantified constraint. In this case, the constraint is satisfied.

Boolean quantified constraints allow combination of constraints with using the Boolean operators `AND`, `OR`, and `NOT`. The operator `NOT` may only be used for existence quantified constraints. The boolean operator `NOT` is suitable to forbid specific patterns. The following constraint ensures that within the topic map document a topic or association that follows a particular pattern does not exist:

```
not exists [ * (programming-language)
             bn: /perl/i ]
```

The boolean operators `AND` and `OR` may be used anywhere in between parts of constraints. For example, they can be used in between existence quantified constraints as shown below. Not how the indentation is used to give `or` the higher precedence:

```
forall $a [ (compiler-for-the-language) ]
=> exists $a ] (compiler-for-the-language)
                compiler: jbuilder
                language: java [
   OR
   exists $a ] (compiler-for-the-language)
                compiler: visualj
                language: java [
```

When combined with boolean operators, the individual constraints above will then be processed as one block. The validation mechanism will check whether the Topic Map document given satisfies all (and) or some (or) of the constraints in that one block.

A blank line is used to separate each block of constraints. Each of these blocks of constraints will be processed in turn, so that they are implicitly AND-ed, although they cannot share variables. Depending on the validation implementation it may flag which block of constraint is violated by a particular topic map document.

Summary

AsTMa! is a language to express rules for topic map authors. Rules apply to whole maps, so quantifiers like *forall* and *exists* help to precisely impose constraints based on particular patterns. These patterns are similar to AsTMa= topics and association, but are extended to allow regular expressions where normally topic ids or text is allowed. To fine tune the required form of a topic or an association the knowledge engineer has to define whether the pattern is to be interpreted in a closed, restrictive way, or, alternatively, whether a conforming map can contain more information than is prescribed (open).