The τ Model, Formalizing Topic Maps

RobertBarta[*]
Bond University
Faculty of Information Technology
Gold Coast, Australia

GernotSalzer[†]
Technische Universität Wien
Institut für Computersprachen
Vienna, Austria

Abstract

This paper presents a formalization for Topic Maps (TM). We first simplify TMRM, the current ISO standard proposal for a TM reference model and then characterize topic map instances. After defining a minimal merging operator for maps we propose a formal foundation for a TM query language. This path expression language allows us to navigate through given topic maps and to extract information. We also show how such a language can be the basis for a more industrial version of a query language and how it may serve as foundation for a constraint language to define TM-based ontologies.
Keywords: Knowledge Engineering, Semantic Web, Topic Maps

1 Introduction

Topic Maps (TM (Pepper 1999)), a knowledge representation technology alternative to RDF (O. Lassila and K. Swick 1993), have seen some industrial adoption since 2001. Concurrently, the TM community is taking various efforts to define a more fundamental, more formal model to capture the *essence of what Topic Maps are* (Newcomb, Hunting, Algermissen & Durusau 2003, Kipp 2003, Garshol 2004-07-22, Bogachev n.d.). While the degree of formality and the extent of TM machinery varies, all models tend to abstract away from the sets of concepts defined in (Pepper 2000) and use assertions (and topics) as their primitives.

After giving an overview over the current state of affairs, we start with an attempt to conceptually simplify the TMRM (Newcomb et al. 2003) model. From that, a mathematically more rigorous formalization of TMs follows in section 4. Based on maps and elementary map composition we define a path expression language using a postfix notation. While low-level, it forms the basis for querying and constraining topic maps as we point out in section 6. The last section closes with future research directions.

2 Related Work

Historically, Topic Maps, being a relatively new technology, had some deficits in rigor in terms of a defining model. This may be due to the fact that it was more endorsed by industrial players than by the academic community.

Paradoxically, the standardization efforts started out with the syntax (XTM) with only little, informal description of the individual constructs. TMDM (formerly known as SAM) was supposed to fill this role by precisely defining how XTM instances are to be deserialized into a data structure. This is done by mapping the syntax into an infoset model (comparable to DOM) whereby UML diagrams help to illustrate the intended structure as well as the constraints put on it. While such an approach to model definition has a certain appeal for (Java) developers, its given complexity puts it well outside the reach for a more mathematical formalization.

In parallel a fraction within the TM community argued that the TM paradigm can be interpreted on a much more fundamental level if one considers assertions as the basic building blocks, abstracting from the TAO-level which mainly sees topics with their names, occurrences and involvements in associations. This group has developed several generations of the TMRM (Newcomb et al. 2003), the *reference model*. The model therein is mainly based on graph theory mixed with informal descriptions of constraints which cover the resolution of *subject identity*.

Several attempts to suggest an alternative foundational model (Garshol 2004-07-22, Bogachev n.d.) or to formalize TMRM have been made. (Kipp 2003) is successfully using a purely set-theoretic approach to define topic map instances. As all TMRM concepts have been faithfully included, this resulted in a significant set of constraints to be used when reasoning about map instances.

The contribution of this paper we see threefold: Firstly, we believe that TMRM can be reasonably simplified without any loss of generality by the steps outlined in section 3. This is under the assumption that all questions of subject identity are handled outside the model. Secondly, the assertion model seems to be general enough to host conceptually not only TMRM, but also serve as basis for TMDM.

As the TM community now moves to ontology definition languages, retrieval and transformation languages, we contend that the path language which is based on the τ model can serve as semantic fundament.

3 Conceptual Simplification

TMRM's main building blocks are *properties* which are attached to topics and *assertions* which connect topics in various ways.

3.1 Properties

For properties TMRM distinguishes between *subject identifying properties* and *other properties*. The for-

[*]rho@bond.edu.au
[†]salzer@logic.at
Copyright ©2005, Australian Computer Society, Inc. This paper appeared at Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), University of Newcastle, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 43. Sven Hartmann and Markus Stumptner, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

mer can be stand-alone or a combination of other properties; they control—for a given application—under which conditions two topics should be regarded the same.

With the assumption that all identity inducing constraints are best covered by a proper ontology language, we drop this distinction. Also conferred properties can be handled much more flexibly with an ontology language, which allows us to let *conferred* and *builtin* properties collapse.

We abstract further by regarding *properties* just as a special form of binary assertions where the topic plays a role `object` and the property forms the other member of the assertion.

3.2 Assertions

A TMRM assertion stands for a statement between subjects whereby these subjects play certain roles. Such an assertion consists of the subject it is about and a type. Additionally, the players are cast into their respective roles. To be able to reify the fact that a certain topic plays a certain role in an assertion, also this substatement is represented by a another topic (casting).

We observe that any type information for an assertion a can be represented by a second, dedicated assertion b where a plays the `instance` and that type plays the role `class`. A similar consideration applies to casting topics: again, a second, dedicated assertion can be used where the role, the assertion and the player are playing appropriate roles.

Scoping—the restriction of an assertion to a certain context—is clearly a statement about an assertion, so we can represent scoping relations via a further assertion, one which connects the original assertion with the scope itself, again via some predefined roles.

At the end of this process we only have to deal with assertions containing role-player pairs. Assertions have an identity which allows us to use them in other assertions. Topics only exist as focal points and have no explicit property except an identifier.

3.3 Reification

The term *reification* has a long tradition (Sowa 2000) in the knowledge representation community. It has changed its meaning over the years, but it is usually used to describe how humans form concepts and then connect them with the 'real world'. To fully capture the term formally, we would have to adopt a philosophical approach, something which we prefer to avoid for obvious reasons. The question, though, is whether any formalization of TMs can completely ignore reification.

Whenever a statement S is about another assertion A then one of two things could be intended by the author: either (a) S is a statement about the relationship in the 'real world' A is supposed to represent. As an example consider that A is about an employment of a person within an organisation and that we want to qualify in such that "the employment only started in year 2000". Alternatively (b), a statement can be about the assertion within the map itself, such as "this assertion was commented on by user X". In the latter case we treat A as if it were in the 'real world' (inverting somehow the notion of reification by pulling something abstract from a concept space and making it 'real').

Our—pragmatic—approach is that this distinction can (and should) be indicated by the proper form of identifiers. If a topic is supposed to reify a real world concept, then its identifier should be a URI (a locator or a name), in case that the 'real world thing' has one. If that thing is a topic in a topic map, then the author must have a way to address the map as well as the topic within it. If a direct reification is not possible, then the topic's identifier will simply not be a URI. Indirect identification can be achieved via subject indicators attached to the topic or more generally speaking by the context the topic is in.

For assertions we assume that they—as a whole—implicitly reify the relationship they describe. If another assertion makes a reference to an assertion then using the assertion's identifier may thus automatically cover case (a) above. Like with topics, case (b) can be handled by using an identifier which addresses the map and then the assertion within it.

How eventually maps as 'real world' objects are to be addressed is again a matter how identifiers are formed; but this is outside the scope of our model.

4 Formal Maps

In this section we first prepare the grounds by defining identifiers, then we build members and assertions and then finally maps. For presentation, the text here has two layers, one for the formal part and an informal one, shaded grey. The latter is to justify design issues or present examples.

4.1 Identifiers

The set of identifiers, $\mathcal{I}$, contains two sets of objects: names and literals. Literals may be numbers or quoted strings. The set of names, $\mathcal{N}$, is an enumerable collection of atomic objects. *Atomic* means that objects have no other properties than being distinguishable from each other.

In practical situations names may be strings such as URIs. They also may be more complex like XLink or even HyTime pointers. The model only uses the property that they are distinct from each other.

The reason we chose literals to be numbers or strings is simply one of convenience. First, these two basic data types are the most frequently used, and secondly, both have naturally defined an ordering $a \leq b$ on which we can later base sorting.

One issue with selecting a particular set of primitive data types is that of how to represent others, like composite types as one would need for, say, spatial coordinates. We see two approaches: One way is to model the content explicitly with assertions themselves. The other option can be used if the structure of the data is not specifically relevant to a particular application, but has to be kept in a map for archiving purposes. In these situations data can be serialized into a string and treated as such.

Further we assume that $\mathcal{I}$ also contains a small set of predefined identifiers, *id*, *instance*, *class*, *subclass*, *superclass*. By themselves, they are not special. We only single them out to be able to define additional semantics later.

4.2 Members and Assertions

As we are mainly interested in expressing associative relations, we first define a *member* to be a pair $\langle r, p\rangle \in (\mathcal{N} \times \mathcal{I})$, with r being the *role* and p the *player* of the member. An *assertion* a is a finite (possibly empty) set of members. The set of all assertions is denoted by $\mathcal{A}$.

Assertions always have an identity. It is a function $id(a)$ over the set of members of a, whereby we only request that different member sets result in different identities. Obviously, assertions are only equal if they have identical members.

To access the components of an assertion a we define the set $roles(a) = \{r_1, \ldots, r_n\}$ with r_i being the roles in the individual members of a, and the set $players(a) = \{p_1, \ldots, p_n\}$ with p_i being the players in a.

Note that in assertions players are not grouped around a role. If several players play one and the same role, then individual members have to exist for every such player. Also note, that assertions do not have a type component; it is up to a further assertion to establish such a relationship whereby the predefined identifiers *instance* and *class* can be used as roles.

The base model does not impose any restrictions on players and roles. While not necessary for the formalism itself, we might later want to put additional constraints on the form of assertions to only *meaningful* combinations. Examples of such meaningful constraints are "there may be only one player for a particular role" or "in one and the same assertion a particular identifier cannot be used as role *and* as player": $\forall a \in \mathcal{A}, roles(a) \cap players(a) = \emptyset$. Another useful constraint could avoid that the identifier for an assertion appears in that assertion itself: $\forall a \in \mathcal{A}, id(a) \notin (roles(a) \cup players(a))$.

This assertion structure proves to be central to the whole model. It is sufficiently flat as there is no distinction between assertions and properties. The focus on assertions alone also reduces topics to identifiers. Still, the chosen structure seems to incorporate enough of the TM paradigm, in that any number of concepts can be bound together into an assertion and topics—as TMRM mandates—can function as the sole aggregation point for information.

4.3 Maps

We now consider assertions to be atoms from which maps can be constructed. A *map* is a finite (possibly empty) set of of assertions. The set of all maps is denoted by $\mathcal{M}$.

To build bigger maps, we define the *elementary composition*, denoted by $\oplus$, of two maps $m, m' \in \mathcal{M}$, is defined as set union $m \oplus m' = m \cup m'$. We say that m is a *submap* of m' if $m \subseteq m'$.

Note that we have no special merging operation; only exactly identical assertions will be identified. In our setting special-purpose merging, such as TNC (topic name constraint), is split into two phases: first maps are combined using elementary composition and then a second operator is applied to the composite map. That operator will perform a—more or less sophisticated—transformation where all the appropriate merging is done.

As an example we consider a network which hosts several servers, organized into clusters (Table 1). At a particular point in time, servers may be `"up"` or `"down"`.
Accordingly, `macy`, `lacy` and `stacy` are the servers, the first two being in `clusterA`, the other in `clusterB`. While `lacy` is down, `clusterA` is still functional, not so `clusterB` as its only machine is down.

4.4 Primitive Navigation Operators

To navigate through maps and to extract information out of them, we first need to define basic navigation operations within a given map.

In our model we can navigate along roles. One way is to follow a role *outwards* in a given assertion $a \in m$. Given additionally a name r we define the *role-out* operator $a \searrow r = \{p \mid \langle r, p\rangle \in a\}$. It returns all players of a given role in an assertion.

Looking at `a00` in the above example, the expression `a00` $\searrow$ `class` returns the set containing `server` only.

Another option to navigate is to follow a role *inwards*, seen from an assertion's point of view. Given a map m, a name r and an identifier p, we define the *role-in* operator $p \nearrow_m r = \{a \in m \mid \langle r, p\rangle \in a\}$. We omit the reference to m if clear from the context.

To find all assertions in which `clusterA` plays the role `whole`, we can write `clusterA` $\nearrow$ `whole` which evaluates to $\{$`a02`, `a11`$\}$.

The *role-in* operator does not respect the type of assertions. It simply finds all assertions where a particular player plays the given role. However, for practical reasons a refined version of the operator will be defined in section 4.6.

4.5 Subclassing and Instances

To describe (and query) topic maps, we need to express relationships between concepts. While the variety of such relations itself is huge, two special relationships stand out as being fundamental: The *subclass-superclass* relationship is used between classes to form taxonomies (type systems). The *instance-class* relationship is established between an object and the class (or set) the object can be classified into.

Given a map m and names $b, c \in \mathcal{N}$, we define the predicate $subclasses_m(b, c)$ to be true if there exists an $a \in m$ such that both conditions, $a \searrow$ `subclass` $= \{b\}$ and $a \searrow$ `superclass` $= \{c\}$, hold. As the usual interpretation of *subclassing* is that it is transitive, we build the transitive closure ${subclasses_m}^+$ and the transitive, reflexive closure ${subclasses_m}^*$.

Another relationship is *instance of*, abbreviated as $is-a$ which holds if there exists $a \in m$ such that $a \searrow$ `instance` $= \{b\}$ and $a \searrow$ `class` $= \{c\}$.

Mostly we are interested in an *instance-of* relationship which includes the transitive version of subclassing above. $is-a_m{}^*(b, c)$ holds if there exist $a \in m$ such that for some name c' we have $a \searrow$ `instance` $= \{b\}$, $a \searrow$ `class` $= \{c'\}$ and ${subclasses_m}^*(c', c)$.

According to our cluster map the relations $subclasses_m($`server`, `machine`$)$, $is-a_m($`macy`, `server`$)$ and $is-a_m{}^*($`macy`, `machine`$)$ are all true.

The difference between $is-a_m(b, c)$ and $is-a_m{}^*(b, c)$ is that the former only reiterates the information which is already explicit in the map. When querying a map, though, queries should be built more robust: If we ask for "all machines" in a map, then most likely one is also interested in instances of all (direct and indirect) subclasses of "machine".

Table 1: An example map about a computer network

```
a00 = { < instance, macy     >, < class, server       > }
a01 = { < instance, a00      >, < class, isInstance   > }
a02 = { < part, macy         >, < whole, clusterA     > }
a03 = { < instance, a02      >, < class, isPartOf     > }
a04 = { < object, macy       >, < status, "up"        > }
a05 = { < instance, a04      >, < class, hasStatus    > }
...
a10 = { < instance, lacy     >, < class, server       > }
a11 = { < part, lacy         >, < whole, clusterA     > }
a12 = { < object, lacy       >, < status, "down"      > }
...
a20 = { < instance, stacy    >, < class, server       > }
a21 = { < part, stacy        >, < whole, clusterB     > }
a22 = { < object, stacy      >, < status, "down"      > }

a30 = { < subclass, server   >, < superclass, machine > }
a40 = { < instance, clusterA >, < class, cluster      > }
a41 = { < instance, clusterB >, < class, cluster      > }
```

4.6 Typed Navigation

We can use the relation $is-a_m{}^*(b,c)$ to specialize the *role-in* navigation. Given a map m, names r and t and an identifier p the *typed role-in operator* honors additionally an assertion type:

$$p\nearrow_m r\,[t] = \{a \in p\nearrow_m r \mid is-a_m{}^*(id(a),t)\} \quad (1)$$

The obvious difference to the original *role-in* navigation is that we now only consider assertions of the given type to be part of the resulting set.

The expression `clusterA`$\nearrow_m$`whole`[`hasStatus`] is supposed to find all assertions of type `hasStatus` in which `clusterA` is the `whole`. Since there is no such assertion, the result is empty.

A further way to generalize the navigation is to allow as role also all subclasses:

$$a\searrow_m r^* = \{p \mid \exists\langle r',p\rangle \in a\colon subclasses_m{}^*(r',r)\} \quad (2)$$

$$p\nearrow_m r^* = \{a \in m \mid \exists\langle r',p\rangle \in a\colon subclasses_m{}^*(r',r)\} \quad (3)$$

5 Map Path Language

The topic map path language can be used to extract information out of given map. The language will be defined via postfix operators which are applied to (sets of) assertions (or identifiers).

Before we can formally define the individual postfixes and chains of postfixes (*path expressions*) we have to characterize the results of applying postfixes to a set of assertions, such as a map. This is done with a simple algebra based on tuples.

5.1 Tuple Algebra

Our final result of applying a path expression will be a *bag of tuples*. The advantage of tuples are that they can hold composite results. Every tuple represents then one possible result, all of them are organized into a bag. Bags are like sets except that a particular element may appear any number of times. This is convenient if we later want to sort or count the tuples. Otherwise all the usual set operations can be used on bags.

Assertion tuples are elements from the cartesian product $\mathcal{A}^n$ with $\mathcal{A}$ being the set of assertions. Similarily, *identifier tuples* are elements from $\mathcal{I}^n$. We call n the *dimension* of the tuple.

When we organize tuples $t_1,\ldots,t_n$ into a bag, then we denote this as $[t_1,\ldots,t_n]$.

A map $m = \{a_1,\ldots,a_n\}$ can be represented as the tuple bag $[\langle a_1\rangle,\ldots,\langle a_n\rangle]$. Conversely, we can also interpret a tuple bag as map when the tuples it contains are single assertions.

If a bag contains other bags, then the structure can be *flattened out*:

$$[b_1,b_2,\ldots,b_n] = [b_{\mathrm{ij}} \mid b_{\mathrm{ij}} \in b_i \wedge (1 \le i \le n)] \quad (4)$$

During application of path expressions also tuples of bags may be created. Also these can be reduced by building tuples of all combinations of bag elements:

$$\langle b_1,b_2,\ldots,b_n\rangle = b_1 \times b_2 \times \cdots \times b_n \quad (5)$$

Finally, if a tuple only contains a single component, then it is equivalent to that component:

$$\langle b\rangle = b \quad (6)$$

As we have covered all possible constellations which can occur when evaluating path expressions, we can always reduce every result to a bag of tuples. We call this set $\mathcal{B}_\mathcal{I}$.

5.2 Postfixes and Path Expressions

Individual postfixes (as detailed below) can be combined to form chains. The *set of path expressions* $\mathcal{P}_\mathcal{M}$ is defined as the smallest set satisfying the following conditions:

1. The *projection* postfix π_i is in $\mathcal{P}_\mathcal{M}$ for any non-negative integer i.
2. Every identifier from $\mathcal{I}$ is in $\mathcal{P}_\mathcal{M}$.
3. The *role-out* and *role-in* postfixes $\searrow r$ and $\nearrow r$ for a name r are in $\mathcal{P}_\mathcal{M}$.

4. The *positive predicate* postfix [p = q] and the *negative predicate* postfix [p != q] are both in $\mathcal{P}_\mathcal{M}$ for two path expressions p and q. As special cases we also include [p] and [!p].

5. For two path expressions p and q also the *concatenation* $p \cdot q$ is in $\mathcal{P}_\mathcal{M}$. If - from the context - it is clear that two path expressions are to be concatenated, we omit the infix.

6. For two path expressions p and q the *alternation* $p \| q$ is in $\mathcal{P}_\mathcal{M}$.

The application of a path expression p to a map m is denoted by $m \otimes p$.

For this process, first we will reinterpret the map as tuple bag. Then each of the postfixes in p is applied to it. Each such step results in a new bag which will be flattened according to the tuple algebra above. The final bag will be the overall result.

5.2.1 Projection and Identifiers

For both, assertion and identifier tuples, we will use the *projection postfix* to extract a particular j:

$$\langle u_1, \ldots, u_n \rangle \otimes \pi_j = [\langle u_j \rangle] \tag{7}$$

Projection here plays a similar role like in query languages like SQL, except that we here use an index for selection instead of names.

We drop the index 1 in π_1 if it is applied to a tuple with only a single component where then obviously it holds that $\langle u \rangle \otimes \pi = \langle u \rangle$. Such a projection also serves as the *empty postfix*.

In case the path expression is simply an identifier $i \in \mathcal{I}$, then for any u the result is always this identifier:

$$u \otimes i = [\langle i \rangle] \tag{8}$$

5.2.2 Concatenation and Alternation

We define the *concatenation* $\cdot$ of path expressions p and q (given any u) as

$$u \otimes (p \cdot q) = (u \otimes p) \otimes q \tag{9}$$

The syntactic structure of path expressions ensures that u is always a structure for which such an evaluation is defined.

The *alternation* of two path expressions p and q is defined as the union of the result tuple bags of the individual evaluations:

$$u \otimes (p \| q) = u \otimes p \cup u \otimes q \tag{10}$$

5.2.3 Navigation Postfix

Next we define how *role-out* and *role-in* navigation postfixes can be applied to an assertion tuple. We simply apply the navigation to every assertion in the tuple:

$$\langle a_1, \ldots, a_n \rangle \otimes \searrow r = \langle a_1 \searrow r^*, \ldots, a_n \searrow r^* \rangle \tag{11}$$

$$\langle p_1, \ldots, p_n \rangle \otimes \nearrow r = \langle p_1 \nearrow r^*, \ldots, p_n \nearrow r^* \rangle \tag{12}$$

Note that we have used the typed navigation from section 4.6. While not absolutely necessary, it helps to keep path expressions more concise. Note also, that the individual elements of the resulting tuples are bags. Again, the transformation rules of the tuple algebra have to be used to reduce this into a bag of tuples.

5.2.4 Filtering Postfixes

From tuple bags we can filter out specific tuples using predicates. Given a tuple bag $B = [t_1, \ldots, t_k]$ and two path expressions p and q, applying the *positive predicate postfix* [p = q] to B is defined as

$$B \otimes [\ \mathrm{p} = \mathrm{q}\ ] = [t \in B \mid t \otimes \mathrm{p} \cap t \otimes \mathrm{q} \neq \emptyset] \tag{13}$$

If p and q are identical, then we can abbreviate [p = p] with [p].

The result of the positive predicate prefix is that subbag of B for which elements the evaluation of p and q gives at least one common result.

Note that this implements an *exists semantics* as $B \otimes [\ \mathrm{p} = \mathrm{p}\ ]$ is reducable to $[t \in B \mid t \otimes \mathrm{p} \neq \emptyset]$. Only those tuples of B will be part of the result tuple bag if there exists at least one result when p is applied to that tuple.

By introducing *negation* in predicate postfixes, we can also implement *forall semantics*. Given a tuple bag B and two path expressions p and q, we define the *negative predicate postfix* as

$$B \otimes [\ \mathrm{p} \mathrel{!}= \mathrm{q}\ ] = [t \in B \mid t \otimes \mathrm{p} \cap t \otimes \mathrm{q} = \emptyset] \tag{14}$$

If p and q are identical, then we can abbreviate [p != p] with [! p]. In this case the result tuple bag becomes $[t \in B \mid t \otimes \mathrm{p} = \emptyset]$.

A particular tuple will only then be part of the result tuple bag if p applied to it will not render a single value, i.e. *all* evaluations will return no result.

Implicit in the formalism are the logic conjunction and disjunction of predicate postfixes. Obviously, a logical *and* is provided by concatenating two predicate postfixes ([..] $\cdot$ [..]) as the result of the first postfix will be further tested for the second predicate. The logical *or* between predicate postfixes is implicitly given by alternating them ([..]$\|$[..]).

5.3 Evaluation Example

Let us assume that we are looking for the status of the servers in `clusterA`: $[\pi \searrow \texttt{class} = \texttt{isPartOf}] \searrow \texttt{instance}[\pi \searrow \texttt{whole} = \texttt{clusterA}] \searrow \texttt{part} \nearrow \texttt{object} < \pi \searrow \texttt{object}, \pi \searrow \texttt{status} >$

The first predicate selects out all those assertions in the map which have a `class` role where one of the players happens to be `isPartOf`. If we are then looking at these assertions and the player(s) of the role `instance`, then we have effectively selected the assertions of type `isPartOf` from the map.

We consider each of these assertions (in our case these are `a02`, `a11` and `a21`) and filter out those of them which have a `whole` role where one player is `clusterA`. When we continue with `a02` and `a11`, and then follow the `part`, this leads to a bag containing only the names `macy` and `lacy`.

In the next step we investigate where these names are players of the role `object`, so we find a bag with assertions `a04` and `a12`. Here our path splits into two components: the first one navigates to the name of that object, the other to its `status`. The result is then $[\langle \texttt{macy}, \text{"up"} \rangle, \langle \texttt{lacy}, \text{"down"} \rangle]$.

In second example we look at all clusters which are down, i.e. where all machines in that cluster are down. As result we get $[\langle \texttt{clusterB} \rangle]$: $[\pi \searrow \texttt{class} = \texttt{cluster}] \searrow \texttt{instance}[\pi \nearrow \texttt{whole} \searrow \texttt{part} \nearrow \texttt{object} \searrow \texttt{status}! = \text{"up"}]$

6 Querying, Filtering and Constraining of Maps

Maps and path expressions, as presented here, can serve as a basis for more high-level concepts, as they are needed for ontology and knowledge engineering (Fensel, Hendler, Lieberman & Wahlster 2003). The use of path expressions to extract information out of maps leads to the following observations:

Obviously, $\mathcal{P}_\mathcal{M}$ is a (primitive) *language to query* topic maps. Note, though, that $\mathcal{P}_\mathcal{M}$ lacks all facilities to newly create content, such as XML or TM content as described in (Garshol & Barta 2003). A more industrial topic map query language (TMQL) will have to offer content generation language constructs. While it will also provide more concise syntax due to high-level concepts, $\mathcal{P}_\mathcal{M}$ can (and probably will) act as a semantic foundation.

More formally, we can identify a subset of $\mathcal{P}_\mathcal{M}$, the *filters* $\mathcal{F}_\mathcal{M}$, which contains all those queries which return maps:

$$\mathcal{F}_\mathcal{M} = \{q \in \mathcal{P}_\mathcal{M} \mid \forall m \in \mathcal{M}, m \otimes q = [\langle a_1 \rangle \dots \langle a_n \rangle \mid a_i \in m]\} \tag{15}$$

Clearly, the filtered maps are always submaps of the queried map: $m \otimes f \subseteq m$, for $f \in \mathcal{F}_\mathcal{M}$.

Interestingly, $\mathcal{P}_\mathcal{M}$ can also be regarded as primitive *constraint language*: only when the application of a path expression c to a map m renders any result, then the map conforms to the expectations we have set up in c.

If, for instance, we had set up a query which asks for *all weapons of mass destruction* in our running example, then the result would have been the empty bag. Only if the query follows the structure and the vocabulary of the map, then there will be a non-empty result. Equivalently, this is also true the other way round.

Consequently, we can define a *satisfaction relation* $\models \subseteq \mathcal{P}_\mathcal{M} \times \mathcal{M}$ between a path expression c and a map m, such that

$$c \models m \quad \Longleftrightarrow \quad m \otimes c \neq \emptyset \tag{16}$$

Based on this, logical connectives between constraints can be defined.

7 Future Work

While we concentrate in this work on formalizing the structure of topic maps (at least our understanding thereof) and of an expression language to extract information from them, we have not yet studied any properties of $\mathcal{P}_\mathcal{M}$. Specifically, we are interested how path expressions relate to formulas in description logics (Baader, Calvanese, McGuinness, Nardi & Patel-Schneider 2003, *Description Logics Home Page* n.d.), especially in the light that both can be used to model an ontology. A related question is how a path language can be used to express *identity* (apart from the explicit identity given by the topic's identifier).

Finally, in a larger picture, we are interested in connecting maps, constraints, queries and even maybe updates for topic maps in an algebra. When connecting maps, merging as defined by the XTM standard is an issue.

References

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P., eds (2003), *The Description Logic Handbook*.
URL: *http://books.cambridge.org/0521781760.htm*

Bogachev, D. (n.d.), 'TMAssert'.
URL: *http://homepage.mac.com/dmitryv/TopicMaps/TMRM/TMAssert.pdf*

Description Logics Home Page (n.d.).
URL: *http://dl.kr.org/*

Fensel, D., Hendler, J. A., Lieberman, H. & Wahlster, W., eds (2003), *Spinning the Semantic Web*, The MIT Press.
URL: *http://mitpress.mit.edu/catalog/item/default.asp?tid=9182*

Garshol, L. M. (2004-07-22), 'A proposed foundational model for Topic Maps'.
URL: *http://www.jtc1sc34.org/repository/0529.htm*

Garshol, L. M. & Barta, R. (2003), 'JTC1/SC34: TMQL requirements'.
URL: *http://www.isotopicmaps.org/tmql/tmqlreqs.html*

Kipp, N. A. (2003), 'A mathematical formalism for the Topic Maps reference model'. http://www.isotopicmaps.org/tmrm/0441.htm.
URL: *http://www.isotopicmaps.org/tmrm/0441.htm*

Newcomb, S. R., Hunting, S., Algermissen, J. & Durusau, P. (2003), 'ISO/IEC JTC1/SC34, Topic Maps - reference model, editor's draft, revision 3.10'.
URL: *http://www.isotopicmaps.org/tmrm/*

O. Lassila and K. Swick (1993), *Resource Description Framework (RDF) model and syntax specification, Technical report, W3C*, Camo AS.
URL: *http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.html*

Pepper, S. (1999), 'Navigating haystacks, discovering needles', *Markup Languages: Theory and Practice, Vol. 1 No. 4* .

Pepper, S. (2000), 'The TAO of Topic Maps'.
URL: *http://www.gca.org/papers/xmleurope2000/papers/s11-01.html*

Sowa, J. (2000), *Knowledge Representation: Logical, Philosophical and Computational Foundations*, Brooks-Cole, Pacific Grove.