Topic Maps—Reference Model, 13250-5 version 6.0

Patrick Durusau Steven R. Newcomb

July 13, 2005

This is a working draft of the Topic Maps—Reference Model. It focuses on the integration of Robert Barta's T^+ (formerly τ^+) model with a prose description of the TMRM. Editorial corrections have been made but the *Foreword*, *Introduction*, *Scope* and *Annex A* have been omitted in order to focus on the parts of the TMRM most affected by the integration of the T^+ model. The version of the T^+ paper used for that integration is dated July 11, 2005 and represents the latest version of that document available to the editors.

There are two types of comments in this draft. Comments by the editors that would NOT appear in a final draft begin: Eds. .

No attempt has been made to comply with ISO format in this draft. Such details, along with the omitted sections, will be added prior to submission of this draft for CD balloting.

The contributions of Robert Barta, Lars Heuer and Gernot Salzer, authors of the T^+ model paper, are gratefully acknowledged. That paper was edited and adapted into the following draft.

Special mention should be made of Robert Barta's efforts to answer numerous questions by the editors concerning the T^+ model.

The editors bear sole responsibility for any errors in the adaptation of that paper.

1 Terms and Definitions

Property A key/value pair appearing in a subject proxy.

Subject Proxy A unit of information that is a set of one or more properties, at least one of which has been defined by its governing Subject Map Disclosure as indicating a subject.

Subject Map A possibly empty set of subject proxies.

Subject Map Disclosure (SMD) A set of rules, disclosed in conformance with the requirements of this International Standard, on the structure and interpretation of subject proxies and their properties.

Eds. The T^+ paper suggests only the term "disclosure." The editors see that as problematic as it does not distinguish between what is being disclosed and the expression of that disclosure. Even though the TMRM does not constrain how expressions of disclosure are made, it is an important distinction to make.

The T^+ paper speaks of patterns in subject maps and it is suggested that "Subject Map Patterns" might be a more useful terms than "Subject Map Disclosure." It naturally lends itself to "Subject Map Pattern Languages" and similar uses.

The TMRM should continue to speak of "disclosure" in the sense used by the T^+ but should also make the distinction between that and the expression of that disclosure.

1.1 Notation

1.2 General Notation

$\in$ member of

$\nearrow$ key-in operator

$\searrow$ key-out operator

$\oplus$ set union

$\|$ or

$\neg$ negation

$\rightarrow$ implication

$\cup$ union

$\cap$ intersection

$|$ the set of all * such that

$\Longleftrightarrow$ logical equivalence

$\subseteq$ subset equals

$\forall$ for all

$\exists$ exists

$\amalg$ product

$\emptyset$ empty set

$[]$ empty sequence

$\sum$ sum

$\models$ logical implication

$\approx$ approximate equality

{} set delimiters (also used as proxy delimiters)

1.3 Subject Map Notation

The following notation is used in the formalism of this standard:

$\mathcal{C}$ Constraint set

$\mathcal{I}$ Set of all subject proxie identifiers (internal)

$\mathcal{K}$ Bag of all keys in key/value pairs

$\mathcal{M}$ Set of all maps

$\mathcal{N}$ Set of natural numbers

$\mathcal{P}$ Set of all properties

$\mathcal{P}_{\mathcal{M}}$ Set of all path expressions

$\mathcal{S}$ Tuple sequence set

$\mathbf{T}$ Set of all types

$\mathcal{T}$ Set of all finite-length tuples

$\mathbf{V}$ Set of values

V^n Set of all tuples with length n

$\mathcal{V}$ Set of values as the result of a path expression

$\mathcal{X}$ Set of all proxies

k A key (in key/value pair)

m A subject map

p A property

s A tuple sequence

$\vec{s}^{\mathcal{P}}$ Ordered tuple sequence

v A value (in key/value pair)

$x \Uparrow_m$ All keys where a given proxy is the value of a property (in map m)

$x \Downarrow$ All the keys of a given proxy

ε Empty postfix

2 Subject Proxies and Properties

A subject proxy (proxy) is a representative for a subject in a subject map. Subject maps consist of proxies, which are themselves composed of properties.

Subjects such as books, cars, love and hate have properties. Statements connecting subjects have exactly the same structure.

Proxies are formally defined as follows: A *subject proxy* (or short *proxy*, is a finite set of properties, $\{p_1, \ldots, p_n\}$, with $p_i \in \mathcal{P}$. For the set of all proxies $\mathcal{X}$ consequently $\mathcal{X} = 2^{\mathcal{P}}$ holds.

A property of a subject proxy consists of a *key*, the means by which a property is addressed, and a *value* which is identified by the *key*.

If the set of all subject proxies in a map is defined as $\mathcal{X}$ and the set of all properties (of subject proxies) is defined as $\mathcal{P}$, then a property can be defined as follows: A property is a tuple that consist of the key k and the value v in the following expression: $\langle k, v \rangle \in (\mathcal{X} \times V)$. The set of all properties of a set of subject proxies is denoted with $\mathcal{P}$.

Properties are labeled values. Values are unconstrained. There must be a disclosed way to combine two (or more) of them when properties have to be combined during viewing multiple subject proxies as one subject proxy. Note that values can also themselves be proxies.

In order to address proxies they all have a system identifier and $\mathcal{I}$ is defined as the set of all identifiers. The mapping between $\mathcal{X}$ and $\mathcal{I}$ is defined by the functions $id : X \mapsto I$ and $id^{-1} : I \mapsto X$, with the constraint that $id(x) = id(x')$ only iff $x = x'$. Proxies and their identifiers are interchangeable.

Informally, maps consists of proxies and these, in turn, consists of properties. A property has two components: a key and a value. The key identifies the property inside a proxy; attached to it is the actual value.

Eds. The final sentence in T^+ for the foregoing comment reads: "As such, keys must be proxies themselves, this implies that there is a recursive relationship between proxies and properties." That appears to confuse indentifier (local system address) with subject address. Is a "key" an identifier for a subject proxy or is it something else?

Eds. We omit the paragraph from the T^+ that begins: "Proxies are identical only if they have identical properties." If two or more subject proxies have different identifiers, they are (in our view) different subject proxies, which can be viewed as one subject porxy. One problem raised by the "identical properties" statement is how to treat references to the different subject proxies. See the discusion of the set union infix below in 3.1 Maps.

$\mathcal{N}$, the set of natural numbers, is used to build the following proxies: $\{\langle \perp, 0 \rangle\}$ with the identifier *instance*, $\{\langle \perp, 1 \rangle\}$ with the identifier *class*, $\{\langle \perp, 2 \rangle\}$ with the identifier *subclass*, and $\{\langle \perp, 3 \rangle\}$ with the identifier *superclass*.

Eds. Question: Do we intend to declare subject proxies here that are inherent in all subject maps? Not an objection. If we are, shouldn't we be up front about it?

Eds. Question: What are the subjects represented by these subject proxies? Note that the only property given for each one shares the key $\perp$ which has the values of 0, 1, 2, 3, for the identifiers, "trivial proxy," "instance," "class," "subclass," and "superclass," respectively.

Is this meant to constrain system identifiers? In other words, does this prohibit any disclosure from using these identifiers? And without any meaningful properties for comparison, how would these proxies ever merge with others?

Note that sometimes in the T^+ language, as in this occassion, are not set delimiters but proxy delimiters and $\langle$ and $\rangle$ are *key* and *value* pair (or property) delimiters.

To be perfectly clear about the proxy declarations:

Identifier	Proxy start	Property start	Key	Property	Property end	Proxy end
instance	{	<	$\perp$	0	>	}
class	{	<	$\perp$	1	>	}
subclass	{	<	$\perp$	2	>	}
superclass	{	<	$\perp$	3	>	}

Proxies do not have a dedicated type component. The predefined identifiers *instance* and *class* are used to declare a type for a proxy. To express that a proxy p is an instance of type q, the following expression would be used, $\{\langle \text{instance}, p \rangle, \langle \text{class}, q \rangle\}$.

To address the keys in the properties of a proxy $x = \{p_1, \ldots, p_n\}$ the function $keys(x) = \{k \mid \exists v, \langle k, v \rangle \in x\}$ is defined. The result is a set, as keys may not

occur more than once in a proxy. The function to access all property values inside a proxy is defined as: $values(x) = \{v \mid \exists k, \langle k, v\rangle \in x\}$. This result is a bag.

Eds. We depart from the T^+ in not allowing keys to occur more than once in a subject proxy. To do otherwise, would be similar to allowing more than one attribute on an element in markup to have the same name. How would any system address keys with the same name in the same subject proxy separately?

Eds. The TMRM does not impose any restrictions on property values. Constraints can be expressed on the properties that may occur within proxies or their values.

2.1 Maps

Proxies are the components from which *subject maps* (or short *maps*, are constructed. A *map* is a finite (possibly empty) set of of proxies. The set of all maps is denoted by $\mathcal{M}$. Combination of two maps,$m, m' \in \mathcal{M}$ is defined as $m' = m \cup m'$. The map m is a *submap* of m' iff $m \subseteq m'$.

Eds. We decline to follow the set union of maps as defined by the T^+ model. Consider the following example:

```
       Proxy
       identifier    property   value     property   value


map1   aaa           name       Durusau   SSN        1231


map2   bbb           name       Durusau   SSN        1231


Elsewhere in map2:

       ccc           rolePlayer bbb (other properties omitted)

Under the set union approach, aaa and bbb should be "consolidated."

The problem is there is no mechanism for updating the reference to
the proxy in map2 with the identifier "bbb."

Nor should the TMRM attempt to specify such a mechanism.
```

```
The viewing rules are already responsible for viewing multiple
proxies as one and handling of references is only one of the issues
that they will have to handle.

Our disagreement with the set union infix is general and continuing.

Should the set union infix be defined by disclosures, our
objection would be resolved.

Note objectionable uses of the set union infix occurs in
equations 12-21 and is implied in the postfix "uniq."
```

Viewing of multiple subject proxies as one subject porxy is a partial function $\bowtie$: $\mathcal{X} \times \mathcal{X} \mapsto \mathcal{X}$ which fullfils two purposes: first it identifies pairs of proxies and then it defines how two such proxies should be viewed as one. Viewing multiple subject proxies in a map m, $m|_{\bowtie}$ is defined by:

$$m|_{\bowtie} = \{x \in m \mid \neg\exists y \in m, x \bowtie y \quad \text{or} \quad y \bowtie x \quad \text{is defined}\} \cup \{x \bowtie y \mid x, y \in m\} \tag{1}$$

Eds. This function is the proper home for all disclosures concerning the viewing of multiple subject proxies as a single subject proxy. The "identical" subject proxy language is unnecessary, as is a default definition of the set union infix operator.

3 Map Navigation

The TMRM defines basic navigation operations to enable the expression of constraints on maps and to support the extraction of information from them.

3.1 Primitive Navigation Operators

Navigation is along keys between proxies and other values. To find all keys in a given proxy the notation $x \Downarrow$ is introduced and based on the function *keys*:

$$x\Downarrow = keys(x) \tag{2}$$

The notation to find all keys where a given proxy is *the value* of a property in a context of map m is:

$$x\Uparrow_m = \{k \mid \exists x \in m, \exists v, \langle k, v\rangle \in x\} \tag{3}$$

To find the value of a given key in a proxy, the *key-out* operator is defined for a given proxy $x \in m$: $x \searrow k = \{v \mid \langle k, v\rangle \in x\}$.

To find all the values of a key, given a map m, a key k and a value v, the *key-in* operator is defined as: $v \nearrow_m k = \{x \in m \mid \langle k, v\rangle \in x\}$. The reference to m will be omitted if that is clear from the context.

3.2 Subclassing and Instances

To describe, constrain and query topic maps, relationships must be expressed between proxies. 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.

Eds. This is the first mention of "object" in the T^+ paper. It is not clear if superclass/subclass is to be applied to; 1. a property; 2. an object like variantName as a subclass of Name; or, 3. man as a subclass of person (superclass/subclass of subjects). Note that the definition is of predicates for superclass and subclass. These definitions do not add properties to the subject proxies declared earlier for superclass, subclass and class. The same is true for instance.

Given a map m and proxies $c, c' \in \mathcal{X}$, the predicate $subclasses_m(c, c')$ is defined to be true if there exists an $x \in m$ such that both conditions, $x \searrow \text{subclass} = \{c\}$ and $x \searrow \text{superclass} = \{c'\}$, hold. Transitive closure is defined as: ${subclasses_m}^+$ and the transitive, reflexive closure as: ${subclasses_m}^*$.

Another relationship between two proxies is *instance of*, abbreviated as $is-a_m(a, c)$ which holds if there exists a $x \in m$ such that $x \searrow \text{instance} = \{a\}$ and $x \searrow \text{class} = \{c\}$. The *instance-of* relationship (which includes the transitive version of subclassing) ${is-a_m}^*(a, c)$ holds if there exist $x, c' \in m$ such that there exists $x \searrow \text{instance} = \{a\}$, $x \searrow \text{class} = \{c'\}$ and ${subclasses_m}^*(c', c)$.

The difference between $is-a_m(a, c)$ and ${is-a_m}^*(a, c)$ is that the former only reiterates the information which is already explicit in the map. The latter enables queries that obtain all direct and indirect subclasses of a particular class.

Eds. Note here that we should say whether or not the TMRM is defining the superclass-subclass and instance-of relationships for all subject maps.

Eds. If these predicates are to be defined and made general for all subject maps, then the definitions need to be properties of the subject proxies defined for these subjects.

3.3 Typed Navigation

All subclasses of a key can also be used for navigation:

$$x \searrow_m k^* = \{v \mid \exists \langle k', v\rangle \in x \colon {subclasses_m}^*(k', k)\} \tag{4}$$

$$v \nearrow_m k^* = \{x \in m \mid \exists \langle k', v \rangle \in x \colon subclasses_m{}^*(k', k)\} \tag{5}$$

Equation 4 enables the navigation to all subclasses of a particular key in a particular subject proxy.

Equation 5 enables the navigation to all subclasses of a key, wherever those are located in a subject map.

4 Path Expressions

Path expressions can be used to extract information out of a given map. The path language is defined via postfix operators which are applied to (sets of) proxies (respectively their identifiers). A simply algebra based upon tuples is defined to support expression of individual postfixes and chains of postfixes (*path expressions*) by characterizing the results of applying postfixes to a set of proxies.

Eds. While the editors do not doubt the power and elegance of the tuple algebra and the following exposition on path expressions and postfixes to express constraints on subject maps, the nature of the result set remains uncertain.

If the result set is a set of subject proxies (our view) then how should the disclosures that govern that set of proxies be described?

4.1 Tuple Algebra

Eds. Note that subject proxies are tuples as are properties. The subject proxy tuple consists of its identifier and the subject proxy. Subject proxies are composed of properties, which are key/value pairs.

A particular path expression can be interpreted as an *expression of interest*, i.e. as a pattern to be identified in a map. Tuples of values are a convenient way to capture the result of one pattern match. All these partial results can then be organized into a tuple sequence.

A single tuple with values from a value set $\mathcal{V}$ is denoted as $\langle v_1, v_2, \ldots, v_n \rangle$. Tuples are identical if all their values in the corresponding positions are.

Eds. It is not clear if the final sentence: "Tuples are identical if all their values in the corresponding positions are." applies to subject proxies which are being treated as tuples. As noted earlier, subject proxies could have the same key/value pairs but different identifiers.

Tuples can be concatenated, simply by collating their values:

$\langle u_1, u_2, \ldots, u_m \rangle \cdot \langle v_1, v_2, \ldots, v_n \rangle = \langle u_1, u_2, \ldots, u_m, v_1, v_2, \ldots, v_n \rangle$. This enables the representation of tuples as products of singleton tuples:

$$t = \prod_{i=1}^{n} \langle v_i \rangle = \langle v_1 \rangle \langle v_2 \rangle \ldots \langle v_n \rangle \tag{6}$$

The index may be omitted if it is clear from the context. The set of all tuples with length n is $\mathcal{V}^n$. The set of all finite-length tuples is denoted with $\mathcal{T} = \mathcal{V}^*$.

When tuples are organized into sequences the single sequence is written

$$s = \sum_{i=1}^{m} t_i = [t_1, \ldots, t_m] \tag{7}$$

if that is *unordered* and $\vec{s}^{?}$ for *ordered sequences*. Sequences behave like bags; individual tuples can appear any number of times, there is no inherent order. All tuples within one tuple sequence must have the same number of values.

Eds. The statement that "sequences behave like bags" is inconsistent with the earlier claim concerning "identical" tuples and the "set" language to this point. Gathering tuples into sequences, which behave like bags, i.e., there is no "identical" test, is different from saying that tuples can be "identical."

All sequences, together with the *empty sequence* [] build the *tuple sequence set* $\mathcal{S}$.

Sets of values can be interpreted as tuple sequences in such a way that every value builds exactly one tuple. For a given set $\{v_1, \ldots, v_n\}$ the tuple sequence $\sum_{i=1}^{n} \langle v_i \rangle$ can be built. This conversion is denoted as $\langle \{v_1, \ldots, v_n\} \rangle$. Under this interpretation, a map $m = \{x_1, \ldots, x_n\}$ can be represented as the tuple sequence $[\langle x_1 \rangle, \ldots, \langle x_n \rangle]$. Conversely, a tuple sequence can be interpreted as a map when the tuples it contains are single proxies.

Tuple sequences can be concatenated

$$\sum s_i + \sum t_j = [s_1, \ldots, s_m, t_1, \ldots, t_n] \tag{8}$$

but only if both operand tuple sequences are ordered, is the result ordered. Otherwise it will be unordered. Indices will be omitted if their range is obvious.

Tuple sequences can also be combined by *multiplying* them. Every tuple of the left operand sequence is concatenated with every other tuple of the right-hand one. The way this is defined below is to cut off the first value of each tuple of the second operand and to combine that with every tuple of the first operand. This is then repeated until the second operand does not have tuples with any value left.

The product between two tuple sequences is defined via

$$\left(\sum_{i=1}^{n} t_i\right)\left(\sum_{j=1}^{m}\langle v_{1\mathrm{j}}, v_{2\mathrm{j}}, \ldots, v_{\mathrm{lj}}\rangle\right) = \left(\sum_{i=1}^{n} t_i \sum_{j=1}^{m}\langle v_{1\mathrm{j}}\rangle\right)\sum_{j=1}^{m}\langle v_{2\mathrm{j}}, \ldots, v_{\mathrm{lj}}\rangle \quad (9)$$

$$\sum_{i=1}^{n} t_i \sum_{j=1}^{m}\langle v_j\rangle = \sum_{i,j=1}^{nm}(t_i\langle v_j\rangle) \quad (10)$$

The resulting sequence is unordered.

N-ary functions can be applied to tuple sequences. Given a function $f : \mathcal{V}^n \mapsto \mathcal{V}$ it can be interpreted as one which takes a value tuple of length n and renders one value. To apply it to a tuple sequence, it is applied to every individual tuple and organize the singleton results back into a sequence:

$$f(\sum t_i) = \sum\langle f(t_i)\rangle \quad (11)$$

4.2 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 *empty* postfix ε is in $\mathcal{P}_\mathcal{M}$.
2. The *projection* postfix π_i is in $\mathcal{P}_\mathcal{M}$ for any positive integer i.
3. Every value from $\mathcal{V}$ is in $\mathcal{P}_\mathcal{M}$. This includes proxy identifiers.

   Given path expressions $p_1, p_2, \ldots, p_n$ and a function $f : \mathcal{V}^n \mapsto \mathcal{V}$ then also $f(p_1, p_2, \ldots, p_n)$ is in $\mathcal{P}_\mathcal{M}$.
4. The postfixes *key-out* and *key-in* $\searrow k$, $\nearrow k$ for a key k, and $\Uparrow$, $\Downarrow$are in $\mathcal{P}_\mathcal{M}$.
5. 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. The special cases [p] and [!p] are included.
6. For two path expressions p and q also the *concatenation* $p \circ q$ is in $\mathcal{P}_\mathcal{M}$. If it is clear from the context that two path expressions are to be concatenated, the infix is omitted.
7. For two path expressions p and q the *alternation* $p \| q$ is in $\mathcal{P}_\mathcal{M}$.
8. The postfix *uniq* is in $\mathcal{P}_\mathcal{M}$.
9. Given an order $\leq$ on tuples, then $sort_{\leq}$ is in $\mathcal{P}_\mathcal{M}$.

The application of a path expression p to a tuple sequence s is denoted by $s \otimes p$.

4.2.1 Identifiers, Projection and Functions

When the empty postfix is applied to a tuple sequence, the result is the empty tuple sequence. When a single value v is applied to any tuple sequence, the result is a tuple sequence containing that value in a single tuple $[\langle v \rangle]$.

The *projection postfix* can be used to extract a certain column j from a tuple sequence:

$$\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle \otimes \pi_j = \sum_{i=1}^{n} \langle v_{\mathrm{ji}} \rangle \tag{12}$$

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

When a function is applied to a tuple sequence, then first all the parameter path expressions are evaluated on every individual tuple. As this evaluation may result in a sequence with any number of tuples, the function will be applied to each of them:

$$\sum_{i=1}^{n} t_i \otimes f(p_1, p_2, \ldots, p_m) = f(\sum_{i=1}^{n} \prod_{j=1}^{m} [t_i] \otimes p_j) \tag{13}$$

4.2.2 Concatenation and Alternation

The *concatenation* of path expressions p and q is defined as

$$s \otimes (p \circ q) = (s \otimes p) \otimes q \tag{14}$$

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

$$s \otimes (p \| q) = (s \otimes p) + (s \otimes q) \tag{15}$$

4.2.3 Filtering Postfixes

Specific tuples can be filtered out from tuple sequences using predicates. Given a tuple sequence s and two path expressions p and q, applying the *positive predicate postfix* [p = q] to s is defined as

$$s \otimes [\ \mathrm{p} = \mathrm{q}\ ] = [t \in s \mid (t \otimes \mathrm{p}) \cap (t \otimes \mathrm{q}) \neq \emptyset] \tag{16}$$

Any existing ordering in s will be maintained. If p and q are identical, then [p = p] can be abbreviated with [p].

The result of the positive predicate prefix is that sub-sequence of s for which elements the evaluation of p and q gives at least one common result.

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

By introducing *negation* in predicate postfixes, *forall semantics* can be implemented. Given a tuple sequence s and two path expressions p and q, the *negative predicate postfix* is defined as

$$s \otimes [\ \mathrm{p}\ !{=}\ \mathrm{q}\ ] = [t \in s \mid (t \otimes \mathrm{p}) \cap (t \otimes \mathrm{q}) = \emptyset] \tag{17}$$

Again, any ordering in s will be honored. If p and q are identical, then $[\ \mathrm{p}\ !{=}\ \mathrm{p}\ ]$ can be abbreviated with $[\ !\ \mathrm{p}\ ]$. In this case the result tuple sequence becomes $[t \in s \mid t \otimes \mathrm{p} = \emptyset]$.

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

Logic conjunction and disjunction of predicate postfixes are implicit in the formalism. Logical *and* is provided by concatenating two predicate postfixes as the result of the first postfix will be further tested for the second predicate. The logical *or* between predicate postfixes is given by alternating them.

4.2.4 Navigation Postfixes

key-out and *key-in* navigation postfixes can be applied to a tuple sequence by applying it to every proxy tuple:

$$\left(\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle\right) \otimes \searrow k = \sum_{i=1}^{n} \prod_{j=1}^{l} \langle v_{\mathrm{ji}} \searrow_m k^* \rangle \tag{18}$$

$$\left(\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle\right) \otimes \nearrow k = \sum_{i=1}^{n} \prod_{j=1}^{l} \langle v_{\mathrm{ji}} \nearrow_m k^* \rangle \tag{19}$$

This evaluation depends on a context map m.

The above process simply iterates over each tuple and compute an intermediate result for this one tuple. This intermediate result is achieved by applying the navigation to each value in the current tuple. This results in a set of values, that is converted into a tuple sequence. All these tuple sequences are multiplied, giving one intermediate result. All these intermediate results are then combined into the result.

A similar approach is used for finding keys:

$$\left(\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle\right) \otimes \Downarrow = \sum_{i=1}^{n} \prod_{j=1}^{l} \langle v_{\mathrm{ji}} \Downarrow \rangle \tag{20}$$

$$\left(\sum_{i=1}^{n}\langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \Uparrow = \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}}\Uparrow_m\rangle \tag{21}$$

4.2.5 Sorting

Based on a given ordering $\leq$ on tuples a tuple sequence $s = \sum t_i$ can be ordered: $\overrightarrow{s}^{?} = \sum t_{i'}$ contains exactly the same tuples and additionally satisfies the constraint $i' \leq j' \iff t_{i'} \leq t_{j'}$

Eds. We have omitted the *uniq* postfix. For reference, that language reads: "When the postfix *uniq* is applied to a tuple sequence, it will compute a tuple sequence in which every tuple from the original sequence occurs exactly once. Iff the sequence was ordered, the result will be ordered as well."

5 Ontological Commitments

Path expressions allow to impose constraints on maps. When applying a path expression to a map, there will only be some non-empty result if the map is somehow aligned with the expectations expressed within the path expression.

5.1 Constraints

A given path expression is regarded as a *constraint* and the *satisfaction relation* $\models \subseteq \mathcal{P}_{\mathcal{M}} \times \mathcal{M}$ between a path expression c and a map m is defined as:

$$c \models m \qquad \iff \qquad [m] \otimes c \neq [] \tag{22}$$

A *constraint set* is then simply a set of path expressions.

5.2 Types

Types are those sets from which property values are taken. All such selected sets are presumed to be disjunct so that for every value its type can be implicitely inferred.

Such collections may be simply sets having no further structure. They also may be abelian groups, i.e. sets together with a binary, commutative operator $+$. The operator would prescribe how values are supposed to be combined in the case of the property occurring in a proxy that is one of several proxies being viewed as one proxy. The types might also be sets with ordering defined on the values; this obviously is necessary if values have to be sorted.

A map $m = \{x_1, \ldots, x_n\}$ *conforms* to a given type T, $m \approx T$, if all property values in m are from T, i.e. $\forall x \in m, values(x) \subseteq T$.

5.3 Disclosures

Given a type T and a constraint set C, the tuple $\langle T, C\rangle$ is a *disclosure of ontological commitment* (or short *disclosure*. The set of all such disclosures will be denoted as $\mathcal{D}$. Combination of disclosures is defined as: $d = \langle T, C\rangle$ and $d' = \langle T', C'\rangle$, $d \oplus d' = \langle T \cup T', C \cup C'\rangle$.

A *disclosure governs a map*, $d \models m$, for $d = \langle T, C\rangle$ if m conforms with T and m also satisfies all constraints in C. The *governance* of a disclosure d, $gov(d)$, is the set of all maps which are governed by d.

Eds.

1. The constraints C consist of path expressions that define the property sets that comprise subject proxies.
2. The types T were termed property classes in TMRM version 5.0.
3. The comparison of subject proxies and how those operations are accomplished corresponds to the $\bowtie$ operator, which we read as being composed of path expressions or C.

Note that the issue of disclosure versus the expression of disclosure is not resolved in the T^+ paper.

6 Conformance

If a Subject Map Disclosure meets the requirements of section 5.1, then it is a conforming Subject Map Disclosure.