tolog for TMQL?

Preliminaries

tolog status

- **Current version is 0.1**
  - can only query associations and type-instance relationship
  - supports and, or, not, and inference rules
  - a proposal for version 1.0 is being developed
- **Three implementations**
  - one in-memory implementation in the OKS
  - one SQL-based implementation also in the OKS
  - one in-memory implementation in TM4J
- **Has been in active use since 2001**
  - is by now well understood
  - a substantial number of people have learned it
  - has proven to be easy to implement, use, and learn
- **Ontopia is *very* pleased with tolog**
  - several customers have chosen us because of it
  - one even chose to use topic maps because of it...

The Datalog inheritance

- **Datalog is a subset of Prolog, used in deductive databases**
- **These were a class of databases that implemented logical inferencing on top of relational databases**
- **A large body of research was done on this late 80s and early 90s**
- **tolog is essentially Datalog for topic maps**
  - this means that this body of research can be applied to tolog
  - we have found several valuable insights in this material already
  - it also means tolog is already familiar to many people

Tutorial

The basics of tolog

- **Is loosely based on the Prolog programming language**
- **Some features also stolen from SQL**
- **Basic feature: matching of predicates against topic map data**
- **Supports querying (selects) on**
  - associations
  - class-instance relationships
- **More features need to be added before it can become TMQL**

tolog query results

- **tolog does querying by matching a query against the data**
- **In this process variables are bound to values**
- **A tolog query result is basically a table with the variables as columns and each set of matches as a row**

Query:
Return all composers and the operas that they composed

composed-by($A : composer, $B: opera)

A	B
Boito, Arrigo	Mefistofele
Boito, Arrigo	Nerone
Catalani, Alfredo	Dejanice
Catalani, Alfredo	Edmea
Catalani, Alfredo	La Falce
Catalani, Alfredo	La Wally
Catalani, Alfredo	Lorelei

Association predicates

- **General form of a predicate:**
  - *assoctype* **(** *player1* **:** *roletype1*, *player2* **:** *roletype2* **)**
- **Association and role types are specified with *topic references*:**
  - use topic id (or another form of reference – described later)
  - e.g., born-in (player1 : person, player2 : place)
- **Players may be specified in two ways:**
  - using a *variable* ($name), meaning: find all matches in this position
  - e.g., born-in ($A : person, $B : place)
  - using a *topic reference,* e.g. the topic id of the player (or another form of topic reference – described later)
  - e.g., born-in (puccini : person, $B : place)

Some simple examples

- **born-in($PERSON : person, $PLACE : place)?**
  - find all person and place role players in born-in associations
- **born-in($PERSON : person, lucca : place)?**
  - find all people born in Lucca
- **born-in(puccini : person, $PLACE : place)?**
  - find all places where Puccini was born (there's only one)
- **born-in(puccini : person, lucca : place)?**
  - was Puccini born in Lucca?
  - will return single empty match (true) or nothing (false)
- **Note: Queries always end with '?'**

Chaining predicates (AND)

- **Predicates can be chained (with implicit ands)**
  - born-in($PERSON : person, $PLACE : place),
    located-in($PLACE : containee, italy : container)?
- **This query finds all the people born in Italy**
  - It first builds a two-column table of all born-in associations
  - Then, those rows where the place is not located-in Italy are removed
- **Any number of predicates can be chained**

Projection

- **Sometimes queries make use of temporary variables that we are not really interested in**
- **The way to get rid of unwanted variables is projection**
- **Syntax:**

  select $variable (, $variable)* from
  <query>?

- **The query is first run, then projected down to the request variables**

Sorting

- **Using the result is sometimes easier if we sort it**
- **Syntax:**

  <select>
  <query>
  order by $variable (, $variable)*?

- **Will sort by *variable* in ascending lexical order**
- **Ascending order is the default**
  - To sort by descending order, append the word 'desc'
- **Note that you can sort by any number of variables**
  - useful when one variable has many equal matches

Making use of OR

- **Or allows us to specify multiple ways of finding results**
- **Find opera premieres by city**
  - { premiere($OPERA : opera, $CITY : place) |
    premiere($OPERA : opera, $THEATRE : place),
    located-in($THEATRE : containee, $CITY : container) } ?
- **This is necessary because for some operas we don't know the theatre, only the city**
  - some *premiere* associations are between operas and theatres
  - others are between operas and cities
- **OR has a higher order of precedence than AND**

The built-in *instance-of* predicate

- **Using select returns topics that play the role represented by the variable (here: $CITY)**
  - This has nothing to do with the types of those topics
  - In our case, some are cities, others are theatres, television stations and even countries!
- **We need to extract just the topics of type "city" from this list**
- **There is a built-in predicate that makes this easy**
- **instance-of has the following form:**
  - instance-of **(** *instance***,** *class* **)**
  - NOTE: the order of the arguments is significant
- **Like players, *instance* and *class* may be specified in two ways:**
  - using a *variable* ($name)
  - using a *topic reference*
  - e.g. instance-of ($A, city)

Counting

- **Projection has an additional feature: counting**
- **If you want to know which city had the most premieres, you can tell tolog to count them**
  - select $CITY, **count**($OPERA) from
    instance-of($CITY, city),
    { premiere($OPERA : opera, $CITY : city) |
      premiere($OPERA : opera, $THEATRE : theatre),
      located-in($THEATRE : containee, $CITY : city) }
    order by $CITY ?
- **This will collapse all rows where the city column is the same, and counts the number of collapsed rows**

Some more predicates

- **In addition to *instance-of* tolog has two other useful predicates:**
- **direct-instance-of (*instance*, *class*)**
  - does not take account of the superclass-subclass relationship, as *instance-of* does
- **$A /= $B**
  - true if the two values are not identical
  - { ..$A.. | ..$A.. }, instance-of($A, bling)

Negation

- **Negation in tolog is a kind of filter**
- **What this means is that it can't generate matches**
- **You must first produce matches, and then remove with *not***
- **People other than composers that were born in Italy:**

  born-in($PERSON : person, $PLACE : place),
  located-in($PLACE : containee, italy : container),
  not(instance-of($PERSON, composer))?

- **Removes all matches where the person is a composer (or a subclass thereof)**

Inference rules

- **Enable the query language to deduce new facts that are implied by the information already in the topic map**
- **For example, if a composer 'X' wrote an opera to a libretto based on a work by writer 'Y', one can imply an “inspired by” relationship**
- **Example:**

```
inspired-by($X, $Y) :-
  composed-by($X : composer, $OPERA : opera),
  based-on($OPERA : result, $WORK : source),
  written-by($WORK : work, $Y : writer).
```

- **Use:**

```
inspired-by($A, hugo)?
```

Ways of referring to topics

- **So far we have always used topic IDs to refer to topics**
  - **topic ID**
    - requires topic to be defined in top document
    - if not, use source locator
- **There are a number of alternatives:**
  - **object ID**
    - syntax: @342231
    - always works, but hard to read and write and not stable across different versions of the topic map and (possibly) the OKS
    - fine for dynamic use in applications
  - **source locator**
    - syntax: s"file.xtm#type" (like fully qualified IDs)
  - **subject indicator**
    - syntax: i"http://psi..." (most stable – independent of internal IDs)
  - **subject address**
    - syntax: a"http://www..." (also stable)

Proposed extensions

Occurrence predicates

- **These allow occurrence types to be used as predicates:**
  - homepage($COMPANY, $URI)?
  - birthdate($PERSON, $DATE)?
- **This also means that string literals are necessary for queries like**
  - birthdate($PERSON, “1973-25-12”)?

Comments

- **For non-trivial rules files one quickly finds a need for comments**
  - introductory text at the beginning of the file
  - explanation of what the different inference rules do
  - commenting out code
- **Proposal**
  - '%' starts a comment which extends to the end of the line
  - '%' inside a string does not start a comment
- **Rationale**
  - this is the Prolog and Datalog syntax for comments

Non-binding clauses

- **Sometimes you want to include a clause to get a particular value, not as an inclusion criterion**
- **We want all companies based in Oslo and their home pages**
  - located-in($COMPANY : located, oslo : location), homepage($COMPANY, $HOMEPAGE)?
- **We won't get companies based in Oslo which have no home page, but that's wrong**
- **Proposed solution:**
  - located-in($COMPANY : located, oslo : location), { homepage($COMPANY, $HOMEPAGE) }?
- **Rationale:**
  - can be interpreted as if there were an empty or branch that always succeeds
  - no extra characters or constructs needed

Introspective queries

- **The constructs provided so far can only be used when all types are known**
- **Queries like the following cannot be formulated**
  - find all association types in this topic map
  - find all role types used in more than one association type
  - find all occurrence types
  - find all topics used as scopes which are not role types
- **To achieve this we propose a set of predicates based on the SAM**
- **Two possible approaches**
  - #1: one predicate per information item type
    - each predicate has one keyword argument per property (almost)
  - #2: one predicate per property (pretty much)
    - each predicate has one or two arguments

Approach #1

- **Find all association types in this topic map**
  - association($TYPE : type)?
- **All role types used in more than one association type**
  - association-role($TYPE : type, $ASSOC1 : association),
    association-role($TYPE : type, $ASSOC2 : association),
    association($ASSOC1 : association, $ATYPE1 : type),
    association($ASSOC2 : association, $ATYPE2 : type),
    $ATYPE1 /= $ATYPE2?
- **All occurrence types**
  - occurrence($TYPE : type)?
- **All topics used as scopes but not as role types**
  - { association($SCOPE : scope) | basename($SCOPE : scope) | ... },
    element($SCOPE, $THEME),
    not(association-role($THEME : type))?

Approach #2

- **Find all association types in this topic map**
  - select $TYPE from type($ASSOC, $TYPE), association($ASSOC)?
- **All role types used in more than one association type**
  - role($ASSOC1, $ROLE1), type($ROLE1, $TYPE),
    role($ASSOC2, $ROLE2), type($ROLE2, $TYPE),
    type($ASSOC1, $ATYPE1), type($ASSOC2, $ATYPE2),
    $ATYPE1 /= $ATYPE2?
- **All occurrence types**
  - select $TYPE from occurrence($TOPIC, $OCC), type($OCC, $TYPE)?
- **All topics used as scopes but not as role types**
  - select $THEME from
    scope($CHARACTERISTIC, $SCOPE),
    element($SCOPE, $THEME),
    not(role($ASSOC, $ROLE), type($ROLE, $THEME))?

Problems with existing tolog

- **Referring to topics with URIs is now very painful**
  - URIs are long and awkward and must now be spelled out in full every time
- **Name collisions**
  - if one of your IDs clash with the built-in predicates you must use URIs
  - if you have a lot of inference rules they can clash with each other, with IDs, and with built-in predicates
- **Flat namespace limits number of predicates**
  - if predicates for strings, numbers, dates, ... are to be introduced chances of collisions increase
  - similarly, having large numbers of inference rules becomes difficult

Solution: prefixes and modules

- **Declaring prefixes which are bound to namespaces solves this**
  - using xtm for "http://www.topicmaps.org/xtm/1.0/core.xtm#" as identifier
    select $TOP from
    xtm:superclass-subclass($TOP : xtm:superclass, $SUB : xtm:subclass),
    not(xtm:superclass-subclass($SUP : xtm:superclass, $TOP : xtm:subclass))?
- **Alternatives for the 'as' part are**
  - identifier: use URI as subject identifier
  - subject: use URI as subject address
  - source: use URI as source locator
  - uri: use URI as prefix for a URI literal
  - module: load rules file from the URI
- **The language can define built-in modules identified by URI**
  - these are treated *as if* they were rules files, but don't need to be loaded
  - instead, query engines can recognize the URIs

A string module?

- **A built-in string module could provide predicates like**
  - string:upper($IN, $OUT), string:lower($IN, $OUT), string:title($IN, OUT)
  - string:concat($IN1, $IN2, $OUT)
  - string:starts-with($STR, $SUB), string:contains($STR, $SUB)
  - string:substring($STR, $OUT, start, end?)
  - string:length($STR, $LEN)
  - string:sub-before($STR, $SUB, $OUT), string:sub-after($STR, $SUB, $OUT)
- **Note that not all arguments here can bind new values**
  - string:length($STR, 5) would logically give all strings of length 5, but should be considered an error unless $STR is bound by some other predicate

More extensions

- **To make this work we'll need**
  - numbers, and a syntax for numeric literals
  - the == operator
  - probably also <, <=, >, >= operators
  - possibly also operators for basic arithmetic (+, -, *, /)
- **Clearly we can, if we want, also put in modules for**
  - regular expressions
  - date operations
  - pretty much anything you can imagine
- **The language is extensible through the addition of modules and predicates**
  - this means we can grow it as we want; the basic model can remain the same
  - it also leaves room for proprietary extension in a controlled way

Modifications

- **Can be done through the addition of predicates which modify the topic map**
- **Must be added with care, as modification introduces time**
  - order of evaluation suddenly matters
- **Delete**
  - instance-of($PERSON, bad-person), delete($PERSON)?
- **Update**
  - basename(london, $NAME), set-value($NAME, 'London')?
- **Addition**
  - add-basename(london, 'Londinum', $NAME), add-theme($NAME, latin)?

The consequences of modules

- **tolog can be made to consist of parts**
  - the language core, defining the evaluation model, the concept of predicates, and the module system
  - modules can be added for different purposes, as needed
  - the topic mappiness of tolog can be made to reside in a particular module
- **This allows great flexibility in the language design**
  - and, not to forget, in the evolution of the language

Weaknesses

- **Not sure how to handle scope**
  - a special / operator on the predicate level?
  - by introducing support for sets?
  - by a special clause at the end: SELECT ... FROM ... IN SCOPE ...?
- **Association syntax is verbose**
  - not clear how to shorten it; convenience rules may be one solution
- **Result sets are not topic maps**
  - *can* add the ability to interpret them as such, however
- **Association role handling is subtle**
  - tricky to get right, understand, and implement
- **Using ID is not the best solution**
  - very concise and natural, but doesn't work in all cases
  - generalization to source locators and prefixes improves on this

Integration in context

- **One of the main reasons to have a query language is to allow its use in various contexts**
  - in languages built on top of the query language (XSLT, Schematron, mapping files, ...)
  - in programming languages etc
- **tolog is not straighforward to integrate in this way**
- **A functional language that returns a result as a set is easier**
  - XPath works this way, which makes it very easy to embed
  - a functional language does not fit topic maps very well, however
- **As will be shown, tolog can be used this way**

Relationship to other standards

The RDF QLs

- **It turns out that most RDF QLs are Datalog-like**
  - not all choose a pure Datalog-like approach; some only have a Datalog core
- **RDQL**
  - SELECT ?givenName
    WHERE (?y, <vCard:Family>, "Smith") ,
    (?y, <vCard:Given>, ?givenName)
    USING vCard FOR <http://www.w3.org/2001/vcard-rdf/3.0#
- **RQL**
- **RIL**
- **???**

tolog can query RDF

- **By adding a new kind of 'as' keyword tolog can query RDF**
- **A cross TM/RDF query:**
  - using foaf for "http://xmlns.com/foaf/0.1/" as rdf
    xc for "http://psi.ontopia.net/xmlconf/#" as indicator
    select $B from
    foaf:mbox($A, "mailto:larsga@ontopia.net"),
    foaf:knows($A, $B),
    foaf:mbox($B, $BMAIL),
    xc:email($BTM, $BMAIL),
    xc:employed-by($BTM : xc:employee, $C : xc:employer),
    xc:homepage($C, "http://www.empolis.com")?
- **Note the use of the email address to do the join across the TM/RDF boundary**

Consequences

- **tolog can be used to do RDF/TM integration in applications**
- **It is technically possible to create a common RDF/TM query language core, maybe even a fully common language**
  - the political issues are something else entirely, of course
- **We can avoid greater RDF/TM incompatibilities than necessary**
  - the two communities can work together, for once
  - less to learn for people dealing with both
- **Implementing tolog on top of RDF is easy**

tolog can query the RM

- **The RM notion of an assertion is very close to the notion of a predicate**
  - the Berlin paper used the term “statement” to explain how topic maps could be mapped to the predicates used to query them
- **This means that the SAM-specific parts of tolog would really be the SAM module**
  - admittedly this depends on how we support scope
- **We can have our cake, and eat it, too!**
  - we can go with SAM now
  - we *could* add an RM module later, when the RM is ready for it
  - the language core and other modules will be common
- **This means we can move forward now, but remain future-proof**

tolog can query RDBMSs

- **A table maps to a predicate, with the field names as role names**
  - using uni for "jdbc:postgresql:net///university" as sql
    select $NAME, $ADDRESS from
      uni:employee($NAME : name, $ADDRESS : address, $DEPID : depid),
      uni:department('research' : name, $DEPID : id)?
- **In SQL, this would be**
  - select NAME, EMPLOYEE.NAME
    from EMPLOYEE, DEPARTMENT
    where DEPARTMENT.NAME = 'research' AND
          DEPARTMENT.ID = EMPLOYEE.DEPID;

tolog – the universal query language

- **In fact, tolog can query anything!**
  - Datalog-like query languages for XML already exist (like BECHAMEL)
- **In truth, it's Datalog that can query anything**
  - tolog is just Datalog adapted to topic maps
- **The benefit is, however, that tolog can turn anything into topic maps**
  - the potential usage area becomes very wide
  - information integration, logical inferencing, ...

ontopia

TMTL

Do we need an XSLT for topic maps?

- **There are several reasons to think so**
  - the most common application of topic maps is to create web portals
  - most topic map applications involve a web interface *somewhere*
  - solutions to this exist, but they are all proprietary
  - visualizing topic maps by programming against an API is *hard*
- **To make topic maps succeed we need to**
  - create something that makes it easy for non-programmers to use TMs
  - create a thriving open source culture for TMs
  - help new technology providers see how to make use of topic maps
- **A standardized language for topic maps -> textual output could do all of this**

Ontopia's Navigator Framework

- **Ontopia has a tool called the Navigator Framework that does this**
  - it *dramatically* simplifies the task of creating web applications with TMs
  - programmers can learn it in a day
  - it is based on JSP, which is inappropriate for a standard
  - it does not make sufficient use of tolog
  - it is too complex and needs a redesign
- **We have created a language we call TMTL to replace it**
  - it solves all the problems described above
  - I implemented it in a single night (roughly 5 hours; 567 LOC)
  - we do *not* offer it commercially at this point
- **We want to show it for two reasons**
  a) it illustrates the idea of embedding a query language in another language
  b) we may want to standardize something like it

TMTL language features

- **Basic workings are like XSLT, except there are no template rules**
- **New predicate introduced: name($TOPIC, $STRING)**
  - selects the most appropriate name for the topic
  - always produces a string, but that may be “[No name]” if none is found
- **<tmtl:page/> wraps the TMTL transformation**
- **<tmtl:if select=”...”>...body...</tmtl:if>**
  - query in 'select' is run, if there is a result the body is executed for the 1st row
  - the values bound by query are available inside the element
- **<tmtl:foreach select=”...”>...body...</tmtl:foreach>**
  - exactly like <tmtl:if>, except body is executed once for each result row
- **In content {$VAR} is used to output**
  - a string, if $VAR is a string or a locator
  - an ID, if $VAR is a topic map object

Example

```
<tmtl:page xmlns:tmtl="http://psi.ontopia.net/tmtl/"> <!-- topic set by context -->
  <tmtl:if select="illustration(%topic%, $PICTURE)?">
    <img align="right" src="{$PICTURE}" height="234"/>  
  </tmtl:if>
  <tmtl:if select=”name(%topic%, $NAME)?”><h1>{$NAME}</h1></tmtl:if>
  <p>Italian composer
    <tmtl:if select="instance-of(%topic%, librettist)?">
      and <a href="librettist.jsp?id={$topic}">librettist</a>
    </tmtl:if>.
    <tmtl:if select="nom-de-plume(%topic%, $NAME)?">
      Also known as {$NAME}.
    </tmtl:if>
    <tmtl:if select="born(%topic%, $DATE)?">
      Born {$DATE}
      <tmtl:if select="born-in(%topic% : person, $CITY : place), name(%CITY%, $NAME)?">
        in <a href="city-region.jsp?id={$CITY}">{$NAME}</a>
      </tmtl:if>.
    </tmtl:if>
```

Optimizations

Reordering clauses

- **The order of clauses is immaterial**
  - the query produces the same result anyway
  - one requirement: not and /= clauses must have all variables bound before you can go there
- **The order affects performance, however**
  - if the first clause produces many matches that means more work for the second clause, and so on...
  - putting a clause that produces few matches first means less work throughout the evaluation
- **The OKS and TM4J in-memory implementations implement this**
  - the technique for doing so is described in the Berlin paper
  - the SQL implementation has no need to do this

Inference rule inlining

- **Inference rules which are not recursive can be inlined**
- **Trivial example is**
  - employed-by($EMPLOYER, $EMPLOYEE) :-
    employment($EMPLOYER : employer, $EMPLOYEE : employee)?
- **When seeing this rule in a query it can be inlined**
- **The same applies to larger rules as well**
  - provided interaction with context is right, and
  - there is no recursion, direct or indirect

Rewriting queries

- **Queries can be rewritten to use implementation-internal predicates in certain situation**
- **This query is likely to be slow in big topic maps**
  - select count($TYPE) from topic($TOPIC), direct-instance-of($TOPIC, $TYPE)?
- **It can be rewritten by the optimizer to use a special predicate**
  - select count($TYPE) from topic-type($TYPE)?
- **The rewritten version is much faster**
  - many inefficiences can be handled in this way
  - basically a clean way to optimize special cases

More techniques

- **Variable merging**
  - merging variables and literals when there are more variables than necessary
- **Or lifting**
  - in some cases predicates can be lifted out of or branches to the main query, avoiding repeated execution

Conclusions

Summary

- **Language has many strengths**
  - already implemented, widely used, well understood, well tried
  - syntax is very concise and regular: very few features
  - easy to implement and easy to learn
  - extensible
  - can handle all the requirements, and some extra (like inferencing)
  - can be implemented efficiently, easy to optimize
  - builds on well-established theory and implementation experience
  - universal query language
- **...and some weaknesses**
  - handling scope is tricky
  - result sets not topic maps
  - awkward use as an embedded language
  - some subtleties

Ontopia's view

- **We are very satisfied with tolog**
  - sufficiently that we think it is a very good candidate for TMQL
- **It needs more work, but that's what the standards process is for**