Relational Completeness of a Query Language for Databases of Labelled Objects

Tadeusz Pankowski

Chair of Control, Robotics and Computer Science, Poznań University of Technology, Poznań, Poland
pankowsk@sol.put.poznan.pl

Abstract

The paper deals with the problem of modelling and querying information in schemaless databases of labelled objects. Labelled objects represent uniformly data from both traditional databases and semistructured data stored in the WWW. It was shown that the proposed rule-oriented query language PathLog based on path calculus is at least relational complete. Some relevant problems concerning partial orderings on labelled objects as well as integrity constraints in databases of labelled objects are discussed.

Keywords: semistructured data, data integration, labelled objects, completeness of query languages

1. Introduction

Labelled objects are used for uniform representation of data integrated from various heterogeneous sources such as traditional databases and semistructured data stored in the WWW sources (created according to e.g. XML or HTML standards). The most popular model for representing semistructured data by means of labelled object is the Object Exchange Model (OEM) [2, 11] which was used in the Tsimmis [5] and the Lore [2] projects. Several query languages for databases of labelled objects have been proposed [2, 6, 14, 15].

In [7] we have proposed PathCal (Path Calculus) as a formal query language, based on predicate calculus, as well as its extension PathLog (allowing, like Datalog, to specify rules and programs). The PathCal takes advantages of the concept of paths for querying databases of (partially) labelled objects.

The main goal of this paper is to show that PathLog is at least relational complete. It means that in the case when a relational database is represented as a database of labelled objects, then operations of relational algebra may be expressed by means of PathLog rules or programs. The importance of such property of the query language is that in the wrappers/mediators architecture for integrating heterogeneous data [9, 10], the relational views of relational sources integrated in the system are still preserved.

The structure of the paper is as follows. In Section 2 we review a notion of labelled objects. The general definition of labelled objects [2, 11, 7] is restricted to atomic objects and to such complex objects which states are either disjunctive or conjunctive sets of identifiers. The restriction allows us for further investigating, in Section 3, a partial ordering on objects. In Section 4 the syntax and semantics of PathCal is described. In Section 5 we introduce PathLog as an extension to PathCal supporting rules, programs and recursion. In particular it allows to control structuring of answer to PathCal queries. Transformation of relational algebra into PathLog programs is described in Section 6. Section 7 concludes the paper.

2. Databases of labelled objects

In comparison to classical approaches, databases of labelled objects, which are devoted to store semistructured data [1, 13], are schemaless. In such databases, there is no separate schema describing data structures. Instead, data are kept together with labels supporting the interpretation of
data. In [7], we have proposed the PLO (Partially Labelled Objects) data model as an extended variant of OEM, in which partial labelling is allowed. In this paper, however, we restrict ourselves to objects which are totally labelled, and which states are atomic constants and disjunctive or conjunctive sets of identifiers.

**Definition 1.** Let \( D \) be a set of atomic constants, \( OID = AID \cup SID \cup TID \) be a set of object identifiers (where \( AID, SID \) and \( TID \) are sets of: atomic, set and tuple object identifiers, respectively), and \( L \) be a set of labels. Labelled objects are defined as follows:

1. If \( i \in AID, \ L \in L \) and \( a \in D \), then \((i, L, a)\) is an atomic labelled object.

2. If \( i \in SID, \ L \in L \) and \( \{i_1, \ldots, i_n\} \) is a finite subset of \( OID \), then \((i, L, \{i_1, \ldots, i_n\})\) is a set labelled object; \( \{i_1, \ldots, i_n\} \) is referred to as disjunctive set of identifiers.

3. If \( i \in TID, \ L \in L \) and \( \{i_1, \ldots, i_n\} \) is a finite subset of \( OID \), then \((i, L, [i_1, \ldots, i_n])\) is a tuple labelled object; \([i_1, \ldots, i_n]\) is referred to as conjunctive set of identifiers.

According to the definition above, every labelled object is a triple of the form \((i, L, val(i))\), where \( i \) is a unique object identifier, \( L \) is a label of the object (the label express the meaning of the object, every object has exactly one label), and \( val(i) \) determines object state; \( val(i) \) is either an atomic constant or a finite set of object identifiers.

A set \( O \) of labelled objects is consistent if for every identifier occurring in \( O \) there is a labelled object in \( O \) with this identifier.

**Example 1.** The following set of labelled objects is consistent:

```
1  Biblio  {2,11} 
2  Doc    [3, 4, 7, 10] 
3  Title  „Active Database Systems“
4  Editor [5, 6] 
5  FirstName „Jennifer“
6  Name   „Widom“
7  Editor [8, 9] 
8  FirstName „Stefano“
9  Name   „Ceri“
10 Year  1996 
11 Doc   [11, 12, 13, 14] 
12 Title  „Introduction to Active Database Systems“
13 Author [5, 6] 
14 Author [8, 9] 
15 Source {2} 
```

In the above example, \((1, Biblio, \{2, 11\})\) is a set labelled object (or a disjunctive labelled object). Its state enumerates references to objects being components of the object. The labelled object \((2, Doc, [3, 4, 7, 10])\) is a tuple object (or a conjunctive object). Its state consists of references to objects constituting the description of the object.
3. Partial ordering on labelled objects

In the universe of labelled objects, we will need an operation for joining objects. The join of two tuple objects is analogous to the join in relational data model. In order to join two set objects, a “dual” approach to the join operation is needed [3, 4, 8]. The foundation for the definition of join gives the theory of partial orderings and the lattice theory. In general, the result of join is an object not less specific (has no less information) than arguments of the operation.

We will use three well-known partial ordering relations, which have been used in the study of the semantics of programs [12], feature structures [4], and value-oriented complex objects [3, 8]. The orderings under considerations are:

- partial ordering for disjunctive sets, \( \leq^\# \) (Smyth’s ordering):
  \[ X \leq^\# Y \iff \forall y \in Y. \exists x \in X. x \leq y; \]

- partial ordering for conjunctive sets, \( \leq^\$ \) (Hoare’s ordering):
  \[ X \leq^\$ Y \iff \forall x \in X. \exists y \in Y. x \leq y; \]

- partial ordering for convex sets, \( \leq^+ \) (Plotkin’s ordering):
  \[ X \leq^+ Y \iff X \leq^\# Y \land X \leq^\$ Y. \]

Note, that according to the Definition 1 states of set objects are disjunctive sets, whereas states of tuple objects are conjunctive sets.

The following theorem [7] introduces a partial ordering into a consistent set of labelled objects.

**Theorem 1.** Let \( o = (i, L, X) \) and \( o' = (i', L', X') \) be labelled objects. The relation \( \leq^O \) such that:

- if \( o \) and \( o' \) are atomic objects, then \( o \leq^O o' \iff i = i', L = L' \land X = X'; \)
- if \( o \) and \( o' \) are disjunctive objects, then \( o \leq^O o' \iff L \leq^\# L' \land X \leq^\# X'; \)
- if \( o \) and \( o' \) are conjunctive objects, then \( o \leq^O o' \iff L \leq^\$ L' \land X \leq^\$ X'; \)
- otherwise \( o \) and \( o' \) are not comparable;

and

\[ i \leq^\# i' \iff o \leq o', \]

\[ L \leq^\# L' \iff I(L) \leq^+ I(L), \] where \( I(L) \) is the set of identifiers of objects labelled with \( L \),

is a partial ordering relation on the set of labelled objects. ■

If \( o \leq^O o' \), we will say that \( o \) is more general than \( o' \) (or that \( o \) subsumes \( o' \) ), or that \( o' \) is more specific than \( o \); similarly for identifiers and labels.

Further on, we will be interested in the join operation on tuple objects and tuple object identifiers. The partial ordering on object identifiers induces a lattice structure into the set of tuple object identifiers: \( \mathcal{T} \) denotes the universal upper bound, i.e. \( i \leq \mathcal{T} \), for every \( i \in \text{TID} \), and \( i \oplus^\# i' \) returns the least upper bound (join) of \( i \land i' \). Analogously, for set object identifiers and labels. The other lattice operation (meet) may be defined as well [7, 8].
**Theorem 2.** Let \( o = (i, L, t) \) and \( o' = (i', L', t') \) be tuple labelled objects. Then the object:
\[ o'' = o \oplus o' = (i \oplus I, L \oplus L', t \oplus t'), \]
where
\[ t \oplus t' = \text{toupper}(t \cup t' \cup \{ i \oplus I \mid i \in t, i' \in t', i \oplus I \neq T \}), \]
and the functions \( \text{toupper}(X) \) removes out from \( X \) all subsuming elements:
\[ \text{toupper}(X) = X - \{ x \in X \mid \exists x' \in X. x' \neq x \wedge x \leq x' \}, \]
is the least upper bound (join) of \( o \) and \( o' \). ■

The following example illustrates joining of tuple objects and simultaneously shows how this operation corresponds to cross product in relational data model.

**Example 2.** Consider the following two relations \( R \) and \( S \) and their representations by means of a set of labelled objects:

<table>
<thead>
<tr>
<th>R</th>
<th></th>
<th>S</th>
<th>1</th>
<th>R [2, 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>2</td>
<td>A a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>3</td>
<td>B b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( R \times S \)

<table>
<thead>
<tr>
<th>T</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>T [2, 3, 8, 9]</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
</tr>
</tbody>
</table>

The conditions: \( T = R \oplus L S, 10 = 1 \oplus I 7, 11 = 4 \oplus I 7 \) are database constraints and have to be inserted into the database system. In particular, \( R \leq L T \) and \( S \leq L T \) are satisfied, since for \( I(R) = \{1, 4\}, I(T) = \{10, 11\} \), we have \( \{1, 4\} \leq \{10, 11\} \) and \( \{1, 4\} \leq \{10, 11\} \). ■

4. **PathCal**

Query languages for labelled objects are often based on notion of *paths* [2, 8, 11]. For the PLO data model we have developed a query language PathCal (*Path Calculus*), which semantics is described by means of algebraic operations on path and sets of paths [7].
A path in PathCal is a sequence

\[(i_1, i_2, \ldots, i_n), n \geq 1,\]

of distinct identifiers such that \(i_{k+1} \in \text{val}(i_k), k = 1, 2, \ldots, n-1,\) i.e. the successor belongs to the state of its predecessor. The composition of paths \(p\) and \(p', p.p',\) is defined iff the concatenation of sequences \(p\) and \(p'\) is a path. On sets \(P\) and \(P'\) of paths the following operations are defined:

- composition: \(P.P' = \{p.p' | p \in P, p' \in P',\) and \(p.p'\) is defined\},
- elongation: \(P? = \{p.i | p \in P, i\) is an object identifier, and \(p.i\) is defined\},
- closure: \(P* = P \cup \{p.p' | p \in P, \) \(p'\) is an arbitrary path, and \(p.p'\) is defined\},
- iteration: \(P+ = P \cup \{p.p' | p \in P, p' \in P+,\) and \(p.p'\) is defined\}.

PathCal is a language for defining and querying databases of (partially) labelled objects.

Among terms of the PathCal we distinguish:

1. **Identity** terms: \(i := x | x := \text{NewId}(), i \oplus i,\) where \(x\) is an identifier variable and \(\text{NewId}()\) is a function returning a new identifier.

2. **Path** terms: \(p := i | v | p.p',\) where \(i\) is an identity term, \(v\) is a path variable.

3. **Set-of-path** terms: \(P := L | \{p_1, \ldots, p_n\} | \text{val}(i) | P.P' | P? | P* | P+ | \{p | \varphi\},\) where \(L\) is a label, \(p_i\) is a path term, \(\text{val}(i)\) is a function denoting state of the complex object identified by \(i, P.P'\) is composition, and \(P?, P*, P+\) are elongation, closure, and iteration, respectively; \(\{p | \varphi\}\) is a query returning all paths for which the formula \(\varphi\) is satisfied.

Formulas of PathCal are defined according to the following rule:

\[
\varphi ::= \text{val}(i) = a | \text{val}(i) = s | P(p) | p = a | p = p | p = = p | p \in p | p \subseteq p
\]

\[
\varphi = \neg \varphi | \varphi \lor \varphi' | \varphi \land \varphi' | \varphi \Rightarrow \varphi'
\]

\[
\exists v \varphi | \forall v \varphi,\) where \(v\) is a path variable free in \(\varphi.\)

**Example 3.** The following PathCal formula defines the set of labelled objects representing tables \(R\) and \(S\) from Example 2:

\[
R(x_1 := \text{NewId}()) \land \text{val}(x_1) = \{x_2 := \text{NewId}(), x_3 := \text{NewID}()\} \land
\]

\[
\land A(x_2) \land \text{val}(x_2) = a \land B(x_3) \land \text{val}(x_3) = b \land \ldots \land
\]

\[
\land S(x_7 := \text{NewId}()) \land \text{val}(x_7) = \{x_8 := \text{NewId}(), x_9 := \text{NewID}()\} \land
\]

\[
\land E(x_8) \land \text{val}(x_8) = c \land C(x_9) \land \text{val}(x_9) = b. \quad \square
\]

An interpretation \(I\) of PathCal assigns with every label \(L\) a set \(I(L)\) of object identifiers. We assume, that \(I(L) \cap I(L') = \emptyset\) for \(L \neq L'.\) A path expression determines the set of paths, e.g. a label \(L\) determines \(I(L).\) For example, Biblio.Doc.Title determines the set \{(1, 2, 3), (1, 11, 12),\} since 1 is identifier of an object labelled by Biblio, 2 is identifier of an object labelled by Doc, and 2 \(\in \text{val}(1) = \{2, 3\}.

**Example 4.** The query „Get documents written by Ceri”, has the following formulation in PathCal:

\[
\{b.d.t | \exists a, n (\text{Biblio.Doc.Title}(b.d.t) \land \text{Doc.Author.Name}(d.a.n) \land n = „Ceri“)\}.
\]
The answer to the query, evaluated against the set of labelled objects from Example 2, consists of one path: \{(1, 11, 12)\} and determines the following set of objects:

1. Biblio \{11\}
2. Dok \{12\}
3. Tytul „Introduction to Active Database Systems”

In the above examples we have used the equality symbol “=” for comparing states of objects. Additionally, an identity symbol “≡” for comparing object identifiers may be used.

**Definition 2.** Let \(\omega\) be a valuation of variables, \(a\) be an atomic constant, \(v\) be a path variable, and \(p\) be a path. The satisfaction relation \(\models\) for (atomic) formulas is defined as follows:

\[
\begin{align*}
\omega \models v = a & \iff \text{val(last}(\omega(v))) = a; \\
\omega \models v = p & \iff \text{val(last}(\omega(v))) = \text{val(last}(p)), \\
\omega \models v \equiv p & \iff \text{last}(\omega(v)) = \text{last}(p).
\end{align*}
\]

The satisfaction relation for composed formulas is defined as usually [7, 16].

In general, it should be possible to use the answer to a query in subsequent queries. Thus, the answer should be inserted into the database (possible as a temporary object or materialised view). Such a result, however, as that from Example 4 contradicts uniqueness of object identity – there is already an object with identifier 1 in the database with the state \{2, 11\}.

To cope with the problem of inserting answers to queries back into the database, we will add to PathCal a mechanism, which allows taking control over structuring of answers to queries. In this way we obtain the language PathLog, which will be presented in the following section.

**5. PathLog**

First, let us discussed motivations underlying PathLog. As was stated in the end of the previous section, we must be careful with inserting answers to queries back into the database. Thus, some objects determined by the answer must be treated as new objects created from the base (original) ones. In order to regard the answer as a set of new objects, we have to:

- determine identifiers, states and labels for new objects,
- define relationships between new objects and their base objects, i.e. objects from which they have been created.

In the Example 4 we could create three new objects: \(o_1\), \(o_2\), and \(o_3\) (from base objects with identifiers 1, 11 and 12, respectively). As their identifiers we can choose subsequent integers (greater than the maximal existing identifier), so we can assume: \(id(o_1) = 16\), \(id(o_2) = 17\), \(id(o_3) = 18\), i.e.:

\[
\begin{align*}
16 & \quad \text{Biblio} \quad \{17\} \\
17 & \quad \text{Doc} \quad \{18\} \\
18 & \quad \text{Title} \quad „Introduction to Active Database Systems”
\end{align*}
\]

The state of each new object is either atomic constant or it satisfies the following conditions:

- each identifier belonging to the state identifies an object existing in the database (an old or a new object);
- the set of paths, which can be determined from new created objects is isomorphic with the set of paths determined by the base objects satisfying the query.
The mentioned isomorphism will be denoted by $\equiv$ and is defined as follows:

- if $i$ is an identifier occurring in the answer and it is the base identifier for $i'$, then $i' \equiv i$, i.e. for every query there is an isomorphic mapping between the set of identifiers returned by the query and the set of identifiers generated by the query;
- two paths $p$ and $p'$ are isomorphic, $p' \equiv p$, if $\text{len}(p) = \text{len}(p')$, and the corresponding elements are isomorphic, i.e. $p'[k] \equiv p[k]$, for every $k$, $1 \leq k \leq \text{len}(p)$;
- two sets $P$ and $P'$ of paths are isomorphic, $P' \equiv P$, if for every path $p \in P$ there is a path $p' \in P'$ isomorphic with $p$, and for each path $p' \in P'$ there is $p \in P$ isomorphic with $p'$.

In our example, $\{(16, 17, 18)\}$ is isomorphic with $\{(1, 11, 12)\}$. Note however, that the considered set of generated objects is not what we expect. It is so because their labels are the same as those of base objects. Thus, the subsequent evaluation of the same query (from Example 4) would return the set consisting of two paths: $\{(1, 11, 12), (16, 17, 18)\}$. To avoid such anomalies, new labels for new objects should be given. A mechanism for defining new labels is supported by rules of PathLog. For the query under consideration we can specify the rule:

$$
\text{Biblio1.Doc1.Title1} \leftarrow \{b.d.t \mid \exists a, n \ (\text{Biblio.Doc.Title}(b.d.t) \land \\
\land \text{Doc.Author.Name}(d.a.n) \land n = \text{"Ceri"})\}.
$$

The new created objects will have labels occurring in the head of the rule (on the left-hand side of the symbol $\leftarrow$). Then, the set of new generated objects, which can be included into the database, is the following:

- 16 Biblio1 {17}
- 17 Doc1 [12]
- 18 Title1 „Introduction to Active Database Systems”

We can see, however, that not all labels occurring in the answer must be new. In the case of title we could leave the old label Title rather than introducing the new label Title1. But then, to avoid the problem discussed above, we must leave also the old object (with the identifier 12) rather then to create a new one.

In general, if in the head of the rule occurs an already existing label, then every object corresponding to it and belonging to the answer is the original object from the database. For such labels we do not create new objects.

Note, that an old label, if occurs in the head, can be only in the last position. Any other old label following it would be redundant. The situation when a new label proceeds an old one leads to inconsistencies, because an identifier of a new created object should be inserted into the state of an old object. So we would have a contradiction – the same object would have two different states: one while considered as an element of the database and another while considered as an element of the answer.

Finally, the correct form of the rule expressing the discussed query is the following:

$$
\text{Biblio1.Doc1.Title} \leftarrow \{b.d.t \mid \exists a, n \ (\text{Biblio.Doc.Title}(b.d.t) \land \\
\land \text{Doc.Author.Name}(d.a.n) \land n = \text{"Ceri"})\},
$$

which determines objects:

- 16 Biblio1 {17}
- 17 Doc1 [12]
- 12 Title „Introduction to Active Database Systems”
Note that the object identified by 12 is an original object from the database. Such an answer can be included into the database (e.g. as materialised view) and can be used in subsequent queries.

In PathLog, like in Datalog [16], we can define rules and programs. In the following part of the paper, by *label term* we will understand an expression of the form $L_1. ... . L_n$, where $n \geq 1$, and each $L_k$ is a label which either exists in the database (i.e. in the alphabet of PathCal) or is a new label which should be assigned to new objects (and should be inserted into alphabet of PathCal).

**Definition 3.** Let $\{p \mid \varphi\}$ be a query in PathCal, and $H$ be a label term. Additionally, we assume that every label occurring in $H$, or every label except for the last one, is in PathCal and that the length of $H$ is not greater than the shortest paths belonging to the answer. Then the expression:

$$H \leftarrow \{p \mid \varphi\}$$

is called a *rule* in PathLog. $H$ is a *head*, and $\{p \mid \varphi\}$ is a *body* of the rule.

Examples of rules were discussed in the beginning of this section. From rules we can build *programs*:

**Definition 4.** A *program* in PathLog is a sequence of rules

$$H_1 \leftarrow \{p_1 \mid \varphi_1\}$$

... 

$$H_m \leftarrow \{p_m \mid \varphi_m\},$$

where all heads $H_k$, $1 \leq k \leq m$, have the same first component (label). ■

A program is non-recursive if there is not any new label in the body of any rule. Otherwise a program is recursive. In this paper we restrict ourselves to non-recursive programs.

Evaluation of a rule $H \leftarrow \{p \mid \varphi\}$, where $H = L_1. ... . L_n$ is carried out in two steps:

1. Calculation of the set of paths constituting the answer to the query $\{p \mid \varphi\}$.
2. Determining the set of objects, possibly creating new objects, according to the head $H$.

Calculation of a head of a PathLog rule consists in determining an interpretation $\Gamma'$ of labels occurring in $H$, i.e. sets $\Gamma'(L_k)$ of object identifiers, for each $1 \leq k \leq n$, and in determining values of the function $\text{val}(i)$ for every identifier $i \in \Gamma'(L_k)$. The resulting set of objects (consisting of old and eventually new objects) will be called a *set of objects determined by $H$*. The set is calculated by means of the following algorithm:

**Algorithm 1.**

**Input:** A database and a rule $L_1. ... . L_n \leftarrow \{p \mid \varphi\}$. Let $L = \{L_1. ... . L_N\}$, where $N = n - 1$, if the last label $L_n$ belongs to PathCal, and $N = n$ otherwise.

**Output:** An interpretation $\Gamma'$ for every label $L \in L$ and a state $\text{val}'(i)$ for every $i \in I(L)$. The new interpretation of the language is $I = I \cup \Gamma'$.

**Steps:**

Assume $\Gamma'(L_k) = \emptyset$, for each $1 \leq k \leq N$.

For every path $p = (i_1, ... . i_m)$ belonging to the answer of the query $\{p \mid \varphi\}$, $m \geq N$, do:

- choose a new identifier $i'_k$ and set the isomorphism between $i'_k$ and $i_k$,

  $i'_k \equiv i_k$. 


• insert $i’_k$ into the set $\Gamma’(L_k)$:
  $\Gamma’(L_k) = \Gamma’(L_k) \cup \{i’_k\}$,
• assume
  $\text{val}(i’_k) = \emptyset$.
• if $k > 1$, then insert $i’_k$ into the set $\text{val}(i’_{k-1})$,
  $\text{val’}(i’_{k-1}) = \text{val’}(i’_{k-1}) \cup \{i’_k\}$.
• if $k = N$ and $N = n - 1$, then insert $i_n$ into the set $\text{val}(i’_N)$,
  $\text{val’}(i_N) = \text{val’}(i_N) \cup \{i_n\}$;
• if $k = N$ and $N = n$, then
  $\text{val’}(i’_N) = \text{val}(i_n)$. ■

In case of PathLog programs, the interpretation for every new label is calculated separately for every rule. Next, the union is calculated. In order to preserve isomorphism we must avoid the situation when two different derived identifiers have the same base identifier. Thus, while calculating subsequent rule we must prove whether for the base identifier under consideration a new identifier was already chosen (in the range of analysed query).

The following algorithm determines interpretation of labels and states for objects within PathLog programs.

Algorithm 2.
Input: A PathLog program:

\[ H_1 \leftarrow \{ p_1 \mid \phi_1 \} \]
\[
\ldots
\]
\[ H_m \leftarrow \{ p_m \mid \phi_m \} \]

where $L = \{L_1, \ldots, L_M\}$ is a set of all new labels occurring in heads of rules..

Output: An interpretation $I'$ for every label $L \in L$ and a state $\text{val’}(i)$ for every $i \in I(L)$. The new interpretation of the language is $I = I \cup I’$.

Steps:
For every rule $L_1, \ldots, L_m \leftarrow \{ p \mid \phi \}$ in the program do:
1. Let $N = n$, if $L_m$ does not belong to the alphabet of PathCal, and $N = n - 1$ otherwise.
2. For every path $p = (i_1, \ldots, i_m), m \geq N$, belonging to the answer to the query $\{ p \mid \phi \}$, do:
   for each $k, 1 \leq k \leq N,$ do:
   if in $\Gamma’(L_k)$ there is no identifier isomorphic to $i_k$, then
   - choose a new identifier $i’_k$ and set the isomorphism between $i’_k$ and $i_k$,
     $i’_k \equiv i_k$;
   - insert $i’_k$ into the set $\Gamma’(L_k)$,
     $\Gamma’(L_k) = \Gamma’(L_k) \cup \{i’_k\}$;
   - take,
     $\text{val’}(i’_k) = \emptyset$;
   - if $k > 1$, then insert $i’_k$ into the set $\text{val}(i’_{k-1})$,
     $\text{val’}(i’_{k-1}) = \text{val’}(i’_{k-1}) \cup \{i’_k\}$;
   - if $k = N$ and $N = n - 1$, then insert $i_n$ into the set $\text{val}(i’_N)$,
     $\text{val’}(i_N) = \text{val’}(i_N) \cup \{i_n\}$;
   - if $k = N$ and $N = n$, then
     $\text{val’}(i’_N) = \text{val}(i_n)$. ■
6. Transformation of relational algebra into PathLog programs

In order to prove that PathLog is not less expressive than relational algebra, we will show that every relational algebra operation can be transformed into a PathLog rule or program. First, we will show how a relational database can be transformed into a database of labeled objects.

Let \( R \) be a relation name (a predicate) of type \( \{ A_1, ..., A_n \} \), where \( A_k, 1 \leq k \leq n \) are attributes. Let \( t \) be a tuple of type \( \{ A_1, ..., A_n \} \), i.e. \( t = [A_1 : a_1, ..., A_n : a_n] \). A relation of type \( \{ A_1, ..., A_n \} \) is a finite set of tuples of this type. An interpretation \( I \) bounds relational name with the relation of the same type (extension of this name), \( I(R) = \{ t_1, ..., t_N \} \). In other words, the set \( \{ t_1, ..., t_N \} \) of tuples constitutes an extension of \( R \), if the formula \( R(t_1) \land ... \land R(t_N) \) is true.

Every element \( t = [A_1 : a_1, ..., A_n : a_n] \) of the extension \( I(R) \) will be represented by the following set of labeled objects:

\[
tr(t, R) = \{ (x := \text{NewId()}, R, \{ x_1 := \text{NewId()}, ..., x_n := \text{NewId()} \}), (x_1, A_1, a_1), ..., (x_n, A_n, a_n) \},
\]

For the whole extension, we have:

\[
\text{TR}(I(R)) = tr(t_1, R) \cup ... \cup tr(t_N, R).
\]

Transformation of relational algebra operations:

1. Extension of relational name:

\[
\rho_R = L \leftarrow \{ x \mid R(x) \}.
\]

The answer consists of the set of identifiers corresponding to tuples belonging to \( I(R) \).

Let \( E' \) and \( E'' \) are expressions of relational algebra which are represented in PathLog by programs \( \rho_{E'} \) and \( \rho_{E''} \), respectively. Additionally, let \( L' \) and \( L'' \) are labels occurring in the first position in heads of rules constituting \( \rho_{E'} \) and \( \rho_{E''} \). Let us consider complex relational operations involving \( E' \) and \( E'' \).

2. Selection with a condition of the form \( A = a \). Let \( E = \sigma_{A = a}(E') \), then:

\[
\rho_E = L \leftarrow \{ x \mid \exists y L'.A(x, y) \land y = a \}.
\]

An identifier \( x \in I(L') \) belongs to the answer (in fact, \( x \) is the base identifier for an identifier isomorphic with it and inserted into the answer) if there is an identifier \( y \in I(A) \) such that \( y \in \text{val}(x) \) and \( y \) identifies object which state is the atomic constant \( a \).

Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>R</td>
<td>[5, 6]</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>a</td>
</tr>
</tbody>
</table>

The selection \( \sigma_{A = a}(R) \) is represented by the following PathLog rule:

\[
R1 \leftarrow \{ x \mid \exists y R.A(x, y) \land y = a \},
\]

which evaluation produces the set of identifiers \( \{7, 8\} \), where \( 7 \equiv 1, 8 \equiv 4 \). It determines:
3. Selection with a condition of the form $A = B$. Let $E = \sigma_{A = B}(E')$, then:

$$\rho_E = L \leftarrow \{x \mid \exists y, z \ (L'.A(x, y) \land L'.B(x, z) \land y = z)\}.$$

**Example**

<table>
<thead>
<tr>
<th>R</th>
<th>1 R [2, 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 A a</td>
</tr>
<tr>
<td></td>
<td>3 B b</td>
</tr>
<tr>
<td></td>
<td>4 R [5, 6]</td>
</tr>
<tr>
<td></td>
<td>5 A a</td>
</tr>
<tr>
<td></td>
<td>6 B a</td>
</tr>
</tbody>
</table>

The selection $\sigma_{A = B}(R)$ is represented by the PathLog rule:

$$R1 \leftarrow \{x \mid \exists y, z \ R.A(x, y) \land R.B(x, z) \land y = z\},$$

which determines the set of objects (with $7 \equiv 4$):

<table>
<thead>
<tr>
<th>7 R1 [5, 6]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 A a</td>
</tr>
<tr>
<td>6 B a</td>
</tr>
</tbody>
</table>

4. Projection. Let $E = \pi_{\{A_1, \ldots, A_n\}}(E')$. It can be represented by the following PathLog program

$$\rho_E = L.A_1 \leftarrow \{x, y \mid L'.A_1(x, y)\},$$

$$\cdots$$

$$L.A_n \leftarrow \{x, y \mid L'.A_n(x, y)\}.$$

**Example**

<table>
<thead>
<tr>
<th>S</th>
<th>1 S [2, 3, 4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 A a</td>
</tr>
<tr>
<td></td>
<td>3 B b</td>
</tr>
<tr>
<td></td>
<td>4 C c</td>
</tr>
<tr>
<td></td>
<td>5 S [6, 7, 8]</td>
</tr>
<tr>
<td></td>
<td>6 A a</td>
</tr>
<tr>
<td></td>
<td>7 B d</td>
</tr>
<tr>
<td></td>
<td>8 C c</td>
</tr>
<tr>
<td></td>
<td>9 S [10, 11, 12]</td>
</tr>
<tr>
<td></td>
<td>10 A a</td>
</tr>
<tr>
<td></td>
<td>11 B b</td>
</tr>
<tr>
<td></td>
<td>12 C d</td>
</tr>
</tbody>
</table>

Projection $\pi_{\{A, C\}}(S)$ is represented by:

$$S1.A \leftarrow \{x, y \mid S.A(x, y)\}$$

$$S1.C \leftarrow \{x, y \mid S.C(x, y)\},$$

which generates the following set of paths: $\{(13, 2), (14, 6), (15, 10), (13, 4), (15, 8), (15, 12)\}$, where $13 \equiv 1, 14 \equiv 5, 15 \equiv 9$. From this we obtain:
5. Renaming. Let $E = \delta_{A/B}(E')$, where $E'$ is an expression of type $\tau = \{A_1, \ldots, A_n, A\}$, and $B$ is not in $\tau$. Then the renaming is represented by the following PathLog program:

$$\rho_E =$$

$$(LA_1 \leftarrow \{x.y \mid L'.A_1(x.y)\},$$

$$\ldots$$

$$LA_n \leftarrow \{x.y \mid L'.A_n(x.y)\},$$

$$LB \leftarrow \{x.y \mid L'.A(x.y)\}.$$ 

Note that the last rule in the program renames $A$ to $B$.

Example

<table>
<thead>
<tr>
<th>R</th>
<th>1</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

Renaming $\delta_{B/C}(R)$ is represented by the PathLog program:

$$R1.A \leftarrow \{x.y \mid R.A(x,y)\},$$

$$R1.C \leftarrow \{x.y \mid R.B(x,y)\},$$

which generates the answer: $\{(7, 2), (8, 5), (7, 10), (8, 11)\}$, where $7 \equiv 1, 8 \equiv 5, 10 \equiv 3, 11 \equiv 6$, and the set of objects:

7 R1 [2, 10]
2 A a
10 C b
8 R1 [5, 11]
5 A a
11 C a

6. Cross product. Let $E = E' \times E''$. Then:

$$\varphi_E = L \leftarrow \{x \oplus y \mid L'(x) \land L''(y)\}.$$ 

To each pair $(i', i'')$ of identifiers satisfying the formula $L'(i') \land L''(i'')$ the operator $\oplus$ assigns an identifier $i$ (the join of $i'$ and $i''$). Additionally, $\text{val}(i) = \text{val}(i') \cup \text{val}(i'')$ (Theorem 2). Thus, the answer includes (up to isomorphism) all such identifiers $i$, for which there are identifiers $i'$ and $i''$ satisfying $L'(x) \land L''(y)$ and such that $i = i' \oplus i''$. 
**Example**
The cross product $R \times S$, against the database from Example 2, is expressed by the PathLog rule:

$$T \leftarrow \{ x \oplus y \mid R(x) \land S(y) \}.$$  

The result set of objects and relevant integrity constraints were discussed in Example 2.

7. **Join.** Let $E = E' \bowtie E''$. Every join can be represented by means of renaming, cross product and selection.

8. **Intersection.** In relational model the intersection of $E'$ and $E''$ is defined when $E'$ and $E''$ have the same type $\tau$. Let $\tau = \{ A_1, ..., A_n \}$. Then $E = E' \cap E''$ can be represented by the PathLog rule:

$$\rho_E = L \leftarrow \{ x \mid L'(x) \land \exists y L''(y) \land \\
\exists u, v (L'.A_1(x,u) \land L'''.A_1(y,v) \land u = v) \land \\
\ldots \land \\
\exists u, v (L'.A_n(x,u) \land L'''.A_n(y,v) \land u = v) \}.$$  

The expression may be represented by the following sequence of operations:

- renaming: $E_1 = \delta_{\{A_1, ..., A_n\}/\{B_1, ..., B_n\}}(E'') = \delta_{A_1/B_1}(\ldots (\delta_{A_n/B_n}(E''))\ldots)$;
- cross product: $E_2 = E' \times E_1$;
- selection: $E_3 = \sigma_{A_1 = B_1} \wedge \ldots \wedge \sigma_{A_n = B_n}(E_2) = \sigma_{A_1 = B_1}(\ldots (\sigma_{A_n = B_n}(E_2))\ldots)$;
- projection: $E = \pi_{\{A_1, ..., A_n\}}(E_3)$.

All these operations can be transformed into rules or programs in PathLog.

**Example**

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>B</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>A</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>B</td>
<td>b</td>
</tr>
</tbody>
</table>

Intersection $R \cap S$ is represented by the rule:

$$Q \leftarrow \{ x \mid R(x) \land \exists y S(y) \land \\
\exists u, v (R.A(x,u) \land S.A(y,v) \land u = v) \land \\
\exists u, v (R.B(x,u) \land R.B(y,v) \land u = v) \},$$  

or as a sequence of following operations:

a) renaming:

- $S1.C \leftarrow \{x,y \mid S.A(x,y)\}$,
- $S1.D \leftarrow \{x,y \mid S.B(x,y)\}$,

b) cross product:

$$T \leftarrow \{ x \oplus y \mid R(x) \land S1(y) \},$$
c) selection:
T1 ← \{x | \exists y, z. T.A(x, y) \land T.C(x, z) \land y = z\},
T2 ← \{x | \exists y, z. T1.B(x, y) \land T1.D(x, z) \land y = z\},

d) projection:
Q.A ← \{x, y | T2.A(x, y)\},
Q.B ← \{x, y | T2.B(x, y)\}.

9. Difference. Let \(E'\) and \(E''\) be of the same type \(\tau = \{A_1, ..., A_n\}\). The difference \(E = E' - E''\) is represented by the rule:

\[
\rho_E = L ← \{x | L'(x) \land \forall y (L''(y) \Rightarrow \\
\exists u, v (L'.A_1(x, u) \land L''.A_1(y, v) \land u \neq v) \lor \\
\text{...} \lor \\
\exists u, v (L'.A_n(x, u) \land L''.A_n(y, v) \land u \neq v))\}.
\]
or

\[
\rho_E = L ← \{x | L'(x) \land \neg \exists y, u_1, v_1, ..., u_n, v_n (L''(y) \land \\
\land L'.A_1(x, u_1) \land L''.A_1(y, v_1) \land u_1 = v_1 \land \\
\text{...} \land \\
\land L'.A_n(x, u_n) \land L''.A_n(y, v_n) \land u_n = v_n)\}.
\]

Example

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

The difference \(R - S\) has the following representation:

\[
R1 ← \{x | R(x) \land \neg \exists y, u_1, v_1, u_2, v_2 (S(y) \land \\
\land R.A(x, u_1) \land S.A(y, v_1) \land u_1 = v_1 \land \\
\land R.B(x, u_2) \land R.B(y, v_2) \land u_2 = v_2)\}.
\]

that produces:

<table>
<thead>
<tr>
<th></th>
<th>R1 [2, 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A a</td>
</tr>
<tr>
<td>3</td>
<td>B b</td>
</tr>
</tbody>
</table>

10. Disjoint union. Let \(E'\) and \(E''\) be of the same type. The expression \(E = E' \cup E''\) has the following representation in PathLog:

\[
\rho_E = L ← \{x | L'(x) \lor L''(x)\}.
\]

The answer consists of identifiers belonging to the extensions of \(L'\) or \(L''\).
Example
The disjoint union $T = S \cup R$ is represented by the PathLog rule:

$$T \leftarrow \{ x \mid S(x) \lor R(x) \}.$$ 

For objects from the previous example, we have:

10  $T \ [2, 3]$
2    $A \ a$
3    $B \ b$
11  $T \ [5, 6]$
5    $A \ c$
6    $B \ b$
12  $T \ [8, 9]$
8    $A \ c$
9    $B \ b$

11. Union. Let $E'$ and $E''$ be of the same type. The union of $E'$ and $E''$ can be expressed by combination of difference, disjoint union and intersection:

$$E = E' \cup E'' = E' \cup E'' - E' \cap E''.$$

7. Conclusion

The main goal of the paper was to show that the proposed rule-based query language PathLog based on path calculus is at least relational complete. PathLog is a formal language for querying schemaless databases of labelled objects. First, we define a universe of labelled objects consisting of atomic and complex objects. Among complex objects we have distinguished disjunctive (or set) objects and conjunctive (or tuple) objects. The criterion of distinguishing is the semantics of data represented by the objects and, following this, a partial ordering relation reflecting the specialisation/generalisation relationships between them. As a query language for databases of labelled objects we proposed PathCal, which is a predicate language utilising a concept of paths. Then we extend PathCal to PathLog, which allows defining rules and programs involving PathCal queries. Finally, we have shown that operations of relational algebra can be expressed by means of PathLog rules and programs.

8. References


