TECHNICAL REPORT NO. 606

An Algebra for Triadic Relations

by
Edward L. Robertson

January, 2005

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

An Algebra for Triadic Relations

Edward L. Robertson†
Computer Science Department & School of Informatics
Indiana University
Bloomington IN 40405
edrbtsn@cs.indiana.edu

Abstract

This paper introduces and develops an algebra over triadic relations, that is, relations whose contents are only triples. In essence, the algebra is a variation of relational algebra, defined over relations with exactly three attributes and closed for the same set of relations. Ternary relations are important because they provide the minimal, and thus most uniform, way to encode semantics wherein metadata may be treated uniformly with regular data; this fact has been recognized in the choice of triples to formalize the "Semantic Web". Indeed, algebraic definitions corresponding to certain of these formalisms will be shown as examples.

An important aspect of this algebra is an encoding of triples, implementing a kind of reification. The algebra is shown to be equivalent, over non-reified values, to a restriction of Datalog and hence to a fragment of first order logic. Furthermore, the algebra requires only two operators if certain fixed infinitary constants (similar to Tarski's identity) are present. In this case, all structure is represented only in the data, that is, in encodings represented by these infinitary constants.

1. Introduction and Motivation

There is increasing interest in developing representations where metadata, that is data about data, is treated uniformly with regular data. Relations are the minimal, and thus most uniform, such representations, a fact recognized by C. S. Peirce in 1885[10]. Peirce viewed the data/metadata distinction as semantic, using a uniform syntax. On the other hand, current relational databases, based on Codd' s original formalization of relational databases in terms of Relational Algebra (RA) and Relational Calculus[3], relegated metadata to an essentially syntactic role.[1] Codd's formulation was quite suitable for the applications and technologies of the day,[2] but applications with huge or dynamic schemata cannot be accommodated with fixed metadata. Heterogeneous situations, where diverse schemata represent semantically similar data, illustrate the problems which arise when one person's semantics is another's syntax – the physical "data dependence" that relational technology

† Supported by NSF Grant IDM 82407

[1] The syntactic nature of this distinction is evident in SQL, where data is represented by strings with quotation marks and metadata without.

[2] Another early attempt at a conceptually founded database system, the DEACON project[1], was similar in spirit to much of the material in this paper. Its performance was dismal, however, taking 17 minutes to execute a mildly complicated query on a very small test data.

was designed to avoid has been replaced by a structural data dependence. Hence we see the need to to a simple, uniform relational representation where the data/metadata distinction is not frozen in syntax.

The first issue in developing a relational characterization is the arity (or arities) of the relations. All relations should have the same arity for simplicity and uniformity; any approach with arbitrary (albeit fixed) arities hides semantics in the choice of arity. Unary relations are obviously insufficient and quadratic (4-airy) relations provide only minimally more capacity than triadic relations. Hence the choice is between two and three. Tarski studied binary relations extensively[12], but relation names played a significant, distinct metadata role. Binary relations are sufficient to represent information in a fixed schema, but the names of these relations are inaccessible from the relation contents. Both a benefit and a disadvantage of binary relations is that they are inherently closed in an algebra of unary and binary operators[12].

Ternary (or "triadic") relations were advocated by Peirce[10] and are significant in a variety of knowledge representation contexts. Two such contexts, the Semantics Web and semantic nets, are discussed below as motivation for our formalization. Join operations on triadic relations, on the other hand, must be carefully defined, lest the results increase in arity (joining two triadic relations on a single attribute results in a quintary relation).

As the World Wide Web has grown from presentation of information into management and manipulation of that information, there has been a recognition of the need for description of not only structure but also content of web artifacts. This description is to be achieved via the "Semantic Web". Central to the Semantic Web is a simple, uniform representation mechanism know as RDF ("RDF" was originally an abbreviation for "Resource Descriptor Framework", but RDF has outgrown the narrow technical aspects implied by this name) and central to RDF is a formalization in terms of triples, discussed as "RDF Model Theory"[16]. For example, this same source specifies that a Sequence (`Seq`) is a Type of Class by requiring that the triple (`[rdf:Seq]`, `[rdf:Type]`, `[rdf:Class]`) must hold in any rdfs-interpretation.

Another natural use of triples is to construct semantic nets (or networks), which are used to express the semantics of natural language. Semantic nets are represented by labeled graphs (often called Conceptual Graphs[11]). For example, "John throws the ball", shown diagrammatically in Fig. 1, has a variety of linear (textual) representations:

i. observing that one component is a relation name and placing that name first (Tarski's convention[12], also used in semantic nets), as in `throw(John, ball)`,
ii. observing one component as a dominant object and placing that first (the F-logic convention[7]), as in `John(throw→ball)`, and of course
iii. representing the triple directly (Peirce's convention[10]), as in (`John`, `throw`, `ball`).

The final form is used in this paper as it is most neutral concerning the roles of the various components.

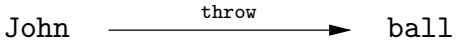


Figure 1

These notations easily extend to "John throws the ball to Mary", decomposed as triples

(John, throw, ball), (John, throw-to, Mary), and (ball, thrown-to, Mary), shown diagrammatically in Fig. 2.

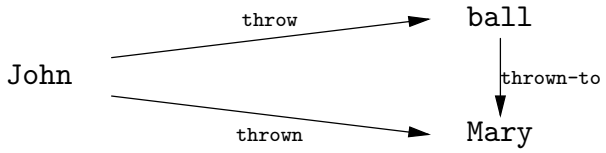


Figure 2

However, the naive labeled graph representation does not work so well with statements about statements, such as "I see John throw the ball". This can be captured better by moving from arrows to triangles, that is, by treating `throw` comparably to `John` or `ball`. To achieve this, "`throw`" is promoted from being an edge label to a full participant in relationships, so that relations are triadic rather than binary. This sentence is thus represented as shown in Fig. 3.

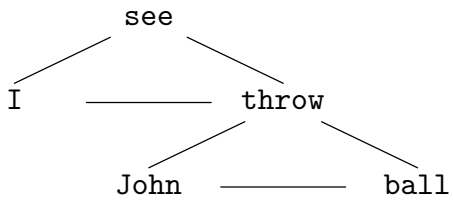


Figure 3

This notation is brittle, however, in that it must distinguish between different occurrences of "`throw`"; otherwise there is no way of preventing the inference, upon adding (`Jane`, `throw`, `stick`), that I see Jane throw a stick, even if that happens out of my eyesight. This difficulty is addressed by reification, which lifts the triple (`John`, `throw`, `ball`) to the single value `John◊throw◊ball`, allowing the representation of "I see John throw the ball" seen in Fig. 4.

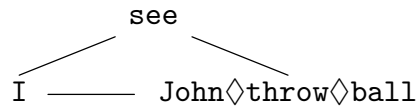


Figure 4

Aspects of this event are then indicated by triples including (`John◊throw◊ball`, `agent`, `John`), extending to include other aspects such as (`John◊throw◊ball`, `to`, `Mary`). Of course this example is still too simplistic – it must at least distinguish the different occasions when John threw the ball, as well as the specific identities of John and the ball. It is from this never-ending quest for differentiation that the notion of value-independent object-id arises.

There are issues concerning reification that deserve further investigation. *Webster's 7th* defines "reify" as "to regard (something abstract) as a material or concrete thing". In our case, the abstract thing is a relationship and its concrete form is a node (or node name). There are two specific choices affecting how reification is expressed – choices which had been implicitly made in the `throw` example above. The first choice is whether the binary relationship or the triple is reified – pictorially, whether the arrow or the triangle is the abstract thing. The second choice concerns whether the name of the new concrete

thing is generated without reference to the component names (anonymously) or the new name is created out of the component names (to coin a word, "nonymously"). In the `throw` example, `John◊throw◊ball` nonymously reifies a triangle. Anonymous reification is essentially object id creation.

We illustrate these differences adapting an example from[15], where the W3C consortium addresses these same issues. In this example, "Chris is diagnosed with cancer" is represented relationally as `diagnosis(Chris,cancer)`, as a triple as (`Chris`, `diagnosis`, `cancer`), and graphically with both variations in Fig. 5. This statement about a diagnosis is itself the subject of a second statement, where the probability of the correctness Chris's diagnosis is said to be high. Reification is needed because of a "statements about statements" situation.

Figure 5

Three different variations of reification of the above statement are shown in Fig. 6. On the left, the relationship (arrow) is reified to the triple (`Chris`, α, `cancer`), where α is a generated anonymous name. Observe that α only identifies a generic relationship; the type of that relationship must be specifically added. In the center is reification as practiced in other Semantic Web discussions (a similar diagram appears in [15]). In this case the relationship between `Chris` and `cancer` is abstracted to a *blank node* (RDF Model Theory uses this term). Observe that this requires a new label, albeit one that may be implicit. On the right is reification as used in this paper, wherein the entire triad relating `Chris`, `diagnosis`, and `cancer` is represented by a single value. As noted above, this last variation reifies a triangle nonymously, in that the three component values are coded into the reification. The textual form (in triples) of the rightmost representation is (`Chris◊diag◊cancer`, `probability`, `high`). For comparison, the XML form of the statement (as it appears in [15]) is given in the Appendix.

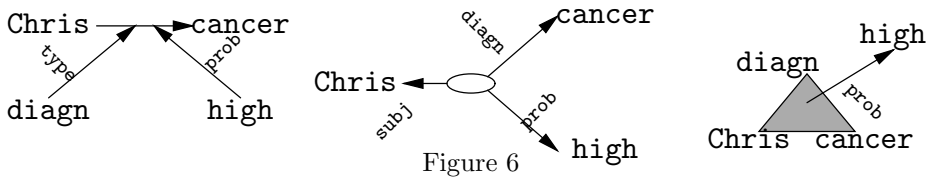


Figure 6

While the representation of information by triples, as discussed above, is well established, there has been no formal mechanism just for manipulating information in that format. Of course any query mechanism will extract information, but the result of this extraction may not be triples. That is, there is no natural way to restrict output of these mechanisms to triples, except by fiat. F-logic[7] has a triadic format at its outset, but it may be used to define higher arity predicates; reification is integrated with F-logic in [17]. Further from a triadic-to-triadic language is relational algebra (RA), although the algebra defined below is essentially a variation of RA which is naturally closed on triadic relations. Closest to a natural triadic-to-triadic query language is a variation of Datalog[13] which

we call "Trilog"; Trilog is defined in Sect. 2 and later used as a standard of comparison.

The remainder of this paper proceeds as follows: Sect. 2 introduces fundamental notation – that is notation not specific to the triadic algebra, Trirel. The definition of the algebra itself is given in Sect. 3. The next section introduces a variant formalization, where all but two operators are replaced by "Tarskian constants"[12] and proves the variant algebra to be equivalent to the original formulation. Although the constants introduced in Sect. 4 are infinite, Sect. 5 shows that the impact of these infinite relations may be controlled. Section 6 proves the analog of Codd's theorem, showing a restricted equivalence between Trirel and Trilog. Finally, Sect. 7 relates this work to the semantic web.

2. Notation

We assume that all values come from a countable fixed domain $\mathcal{D}$. It is, of course, possible to partition $\mathcal{D}$ into types and index the algebraic operators with respect to types, but this leads to unnecessarily cumbersome notation. Hence we globally assume a single domain $\mathcal{D}$. Toward the end of the paper, input values in $\mathcal{D}$ are distinguished from internal values, but all the operators are always over the single domain $\mathcal{D}$.

Lower case letters ($a, b, c, \ldots, x, y, z$) are used as variables over $\mathcal{D}$. Where possible, letters from the beginning of the alphabet are used for manifest values, appearing in the final result, while letters from the end of the alphabet are used for intermediate values. Finally, i, j, and sometimes k are indices for relation coordinates, always with values restricted to $\{0, 1, 2\}$. For example, $*_i$, defined below, actually refers to three operators $*_0$, $*_1$, and $*_2$, which are the same except that they operate on columns 0, 1, and 2 respectively.

The basic structures are sets of triples over $\mathcal{D}$. We refer to these as *triadic relations*. It often aids perspicuity to view these triples in a triangular form. On these occasions (`Chris`,`diagnosis`,`cancer`) is written as $\begin{smallmatrix}\texttt{diagnosis}\\ \texttt{Chris}\quad\texttt{cancer}\end{smallmatrix}$, or more generally (x_0, x_1, x_2) as $\begin{smallmatrix}&x_1&\\ x_0&&x_2\end{smallmatrix}$. Occasionally, d^3 will be used to indicate the triple (d, d, d), for $d \in \mathcal{D}$.

Note the indexing of components in the triple above and its triangular presentation. By convention, indices are interpreted modulo 3, so that position $i + 1$ is always one step clockwise from position i. The benefit of this notation is discussed in the context of $\wr$ and ρ below. The numbering (forgiving the "off by one" quirk) is suggestive of Peirce's *firstness*, *secondness*, and *thirdness*[10].

Set notations serve two purposes in this paper. Occasionally they are used to give definitions of formal constructs; such occasions are always marked with "$\stackrel{\text{def}}{=}$". More often, sets are used for exposition, particularly to provide intuition on the results of evaluating algebraic expressions. When used expositionally, set notation is somewhat informal, without explicit binding of variables (they are all existentially bound over $\mathcal{D}$) and even using "$*$" as a place-holder for unique variables, emphasizing purpose (the traditional "don't care") rather than syntax.

We now turn to Trilog, a restriction of (non-recursive) Datalog[13] to triadic relations. The notions of *extensional database* (EDB) and *intensional database* (IDB) are borrowed

from the discussion of Datalog in [13]. EDB relations are those given as input to the program (or later, algebra). The sets of EDB and IDB relations are disjoint.

2.1 *Definition*: The language *positive Trilog* is the fragment of Datalog subject to the following restrictions:

- *i*: rule bodies are either (*a*) conjuncts of triadic relations, where each variable in the head must also occur in the body, or (*b*) disjuncts of triadic relations, where each variable in the head must also occur in every disjunct,
- *ii*: rule heads are single triadic relations and may not be EDB relations,
- *iii*: definitions may not be recursive,
- *iv*: the reserved word "`result`" designates the result of the computation.

Parameters to relation occurrences, either in the head or body of a rule, may be constants or variables with the usual semantics.

The constraint on occurrences of variables (rule *i*) force Trilog to be *safe*; that is, Trilog programs run on finite EDB always produce finite results.

2.2 *Example*: If the single table `Fact` contains the semantic net information in Fig. 3, then the query "Who saw whom?" is expressed by:

```
result(who,"saw",sub) :- Fact(who,"saw",r) & Fact(sub,r,obj)
```

2.3 *Definition*: Full Trilog is Trilog where negation is allowed in rule bodies. That is, restriction *i*(*a*) is changed to "conjuncts of triadic relations and negations of triadic relations, where each variable in the head occurs in at least one positive relation in the body."

Trilog is of course equivalent to the use of a fragment of first order logic to define ternary predicates, a fragment which has less convenient syntax and safety rules.

Even though the above formulation restricts a single rule body to be purely conjunctive or purely disjunctive, arbitrary Boolean formulae may be represented do to two facts: (1) conversion to disjunctive normal form preserves safeness and (2) all non-output variables in each case of disjunct may be unique.

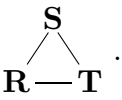
3. Definition of Trirel

The algebra Trirel begins with named variables (the EDB of an expression) and explicitly enumerated sets of triples. Its expressions are built inductively with certain unary, binary, and (a single) ternary operations.

The fundamental operation on triadic relations is a particular three-way join which takes explicit advantage of the triadic structure of its operands. This join of three triadic relations results in another triadic relation, thus providing the closure required of an algebra. This operation first appears as "△" in Longyear[8].

3.1 *Definition*: Let **R**, **S**, and **T** be triadic relations. The *triadic join* of **R**, **S**, and **T**, is defined

$$\mathtt{trijoin}(\mathbf{R},\mathbf{S},\mathbf{T}) \stackrel{\text{def}}{=} \{ {}_{a}{}^{b}{}_{c} : \exists\, x,y,z[{}_{a}{}^{x}{}_{z} \in \mathbf{R} \;\&\; {}_{x}{}^{b}{}_{y} \in \mathbf{S} \;\&\; {}_{z}{}^{y}{}_{c} \in \mathbf{T}]\}$$

An equivalent diagrammatic notation for $\texttt{trijoin}(\mathbf{R}, \mathbf{S}, \mathbf{T})$ is 

.

Of course the trijoin operator could be replicated in higher degrees (if one were interested in the algebra of relations of arity, say, 17). A suitable join operation is

$$\texttt{njoin}(\mathbf{R}_0, \cdots, \mathbf{R}_{k-1}) \stackrel{\text{def}}{=} \{(x_0, \cdots, x_{k-1}) : \exists y_0, \cdots, y_{k-1}[\&_{j=0}^{k-1}\mathbf{R}_j(v_{0,j}, \cdots, v_{k-1,j})]\}$$

where $v_{i,j} = x_j$ if $i = j$ and y_i otherwise. Note that this does not give trijoin; to achieve that, y_{2j-i} is required in the "otherwise" case.

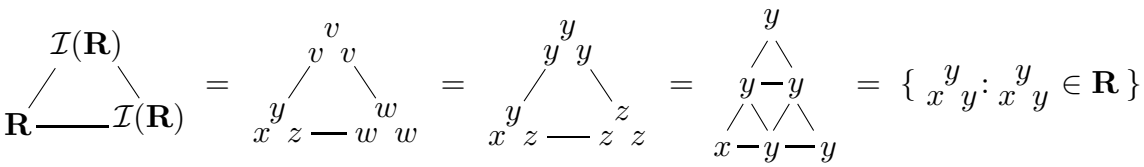
The unary operators of the algebra are $\mathcal{I}$, $\mathcal{D}^3$, ρ, and three $\wr_i$. The motivation for the notation "$\mathcal{D}^3$" becomes apparent in the following section.

3.2 *Definition*:

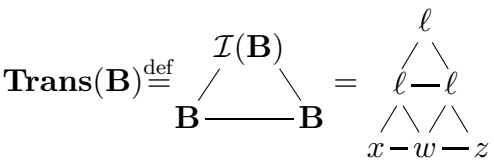
$$\mathcal{I}(\mathbf{R}) \stackrel{\text{def}}{=} \{ \begin{smallmatrix} x & & x \\ & x & \end{smallmatrix} : x \text{ occurs in the active domain of } \mathbf{R}\}$$
$$\mathcal{D}^3(\mathbf{R}) \stackrel{\text{def}}{=} \{ \begin{smallmatrix} y & & \\ & x & z \end{smallmatrix} : x, y, \text{ and } z \text{ occur in the active domain of } \mathbf{R}\}$$

3.3 *Notation*: If we wish to emphasize the coincidence of values that occurs in a tri-join, we will use a notation which collapses pairs of values forced to be equal by the join conditions and makes explicit the structure of the relational operands. Example 3.5 illustrates this convention and motivates the definition of $\mathcal{I}(\mathbf{R})$.

3.4 *Example*:



3.5 *Example*: Returning briefly to the context binary relations, the triple (x, ℓ, z) indicates that the binary relationship $\ell(x, z)$ holds. Assume that $\mathbf{B}$ holds a number of such relation encodings. Then the following computes one transitive step of all these relationships by joining $\mathbf{B}$ with itself, preserving the relationship label (ℓ component):



Note that transitive closure, with its arbitrary iteration of **Trans**, is as impossible in Trirel as it is in RA, and for the very same reasons.

3.6 *Definition*: The (clockwise) *rotation* operator ρ is defined over triadic relations in the expected way: $\rho(\mathbf{R}) \stackrel{\text{def}}{=} \{ \begin{smallmatrix} a & & b \\ & c & \end{smallmatrix} : \begin{smallmatrix} b & & \\ & a & c \end{smallmatrix} \in \mathbf{R} \}$

The conventional numbering of components of a triple, interpreting index expressions modulo 3, is matched by the fact that $\rho^i = \rho^{i+3}$ for any i. For example, assume that $\begin{smallmatrix} y_1 \\ y_0\ y_2 \end{smallmatrix} = \rho^j(\begin{smallmatrix} x_1 \\ x_0\ x_2 \end{smallmatrix})$, then we can state $y_{i+j} = x_i$ for $i \in \{0, 1, 2\}$. The following definition also illustrates this convention.

3.7 *Definition*: The *flip* operators $\wr_i$ fix the i^{th} component of the elements of a triadic relation and interchange the other two.

$$\wr_i(\mathbf{R}) \stackrel{\text{def}}{=} \{\ \begin{smallmatrix} y_1 \\ y_0\ y_2 \end{smallmatrix} : \exists x_0, x_1, x_2 [\begin{smallmatrix} x_1 \\ x_0\ x_2 \end{smallmatrix} \in \mathbf{R}\ \&\ y_i = x_i\ \&\ y_{i+1} = x_{i+2}\ \&\ y_{i+2} = x_{i+1}]\ \}$$

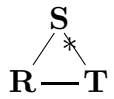
The binary operations of the algebra are the usual set operations $\cap, \cup$, and $-$ (relative complement). Relative complement is the only non-monotone operator in Trirel. Thus the fragment of Trirel that excludes only "$-$" is termed *positive Trirel*.

Sometimes the three equality conditions implied in `trijoin` are too strong, so there are two useful families of operators that introduce "don't-cares", borrowing the "star" notation – note that these operators are not primitive. The $*_i$ operators put don't-cares at i coordinates and $\texttt{trijoin}^*_i$ is a variation of `trijoin` which "breaks the bond" across from corner i.

3.8 *Definition*: $*_i(\mathbf{R}) \stackrel{\text{def}}{=} \texttt{trijoin}(X_0, X_1, X_2)$, where $X_i = \mathcal{D}^3(\mathbf{U})$, $X_{i+1} = \mathbf{R}$ and $X_{i+2} = \mathcal{I}$ and $\mathbf{U}$ is the union of all active domains in the relevant expression.

3.9 *Definition*: (Definitions for subscripts 1 and 2 are symmetric.)

$$\begin{aligned} \texttt{trijoin}^*_0(\mathbf{R}, \mathbf{S}, \mathbf{T}) &\stackrel{\text{def}}{=} \texttt{trijoin}(\mathbf{R}, *_2(\mathbf{S}), \mathbf{T}) \\ &= \{\begin{smallmatrix} b \\ a\ c \end{smallmatrix} : \exists\, w, x, y, z [\begin{smallmatrix} x \\ a\ z \end{smallmatrix} \in \mathbf{R}\ \&\ \begin{smallmatrix} b \\ x\ y \end{smallmatrix} \in \mathbf{S}\ \&\ \begin{smallmatrix} w \\ z\ c \end{smallmatrix} \in \mathbf{T}]\} \end{aligned}$$

The graphical notation equivalent to $\texttt{trijoin}^*_0(\mathbf{R}, \mathbf{S}, \mathbf{T})$ is 

.

All the above are easily definable in RA (or relational calculus). For example, using $\mathbf{R}[i]$ to denote the i^{th} component of $\mathbf{R}$, `trijoin`($\mathbf{R}$,$\mathbf{S}$,$\mathbf{T}$) is simply the RA expression $\Pi_{\mathbf{R}[0],\mathbf{S}[1],\mathbf{T}[2]}(\sigma_{\mathbf{R}[1]=\mathbf{S}[0]\&\mathbf{R}[2]=\mathbf{T}[0]\&\mathbf{S}[2]=\mathbf{T}[1]}(\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T}))$.

On the other hand, the triadic algebra thus-far defined obviously omits operators of standard relational algebra: projection, selection, join, and Cartesian product. Join and Cartesian product have obviously been specialized to `trijoin`. Projection is of course not allowed, since it would break the fact that the algebra is closed on triadic relations. Selection – at least equality selection – is unnecessary because of the equality test implicit in `trijoin`. Using the traditional notation of σ for selection and assuming that $d \in \mathcal{D}$ and that the attributes of $\mathbf{R}$ are named x_0, x_1, and x_2, Example 3.4 implements $\sigma_{x_1=x_2}(\mathbf{R})$ and $\texttt{trijoin}^*_1(\{\ (d, d, d)\ \}, \mathbf{R}, \mathcal{I})$ implements $\sigma_{x_0=d}(\mathbf{R})$.

Inequality selection can of course be derived from relative complement, but it is unseemly to require a non-monotone operator for such an obviously monotone construction. An alternative is to extend Trirel with the operators $\mathcal{N}_j$ (for Not equal), similar to $\mathcal{I}$, such

that $\mathcal{N}_j(\mathbf{R})$ has the j^{th} component different from the other two. Inequality selection is then easily expressed, as in $\sigma_{x_0 \neq x_2}(\mathbf{R}) = \texttt{trijoin}(\mathcal{N}_1(\mathbf{R}), \mathbf{R}, \mathcal{I})$.

The language thus far is too restrictive. This is most easily seen from the diagrammatic perspective. In their graphical form, expressions built from `trijoin`, $\wr$, and ρ are relentlessly planar. Thus it is not possible for these operators to simulate

```
result(x,y,z):- R(x,y,z) & S(xx,yy,zz) &
              Link(x,"L1",xx) & Link(x,"L2",yy) & Link(x,"L3",zz) &
              Link(y,"L1",yy) & Link(y,"L2",zz) & Link(y,"L3",xx) &
              Link(z,"L1",zz) & Link(z,"L2",xx) & Link(z,"L3",yy)
```

This program essentially encodes the Kuratowski graph $K_{3,3}$[6, Thm. 11.13], which is known to be non-planar. Figure 7 shows the connections required in the body of this program, with heavy lines representing `L1 Link`s, dashed lines `L2 Link`s, and dotted lines `L3 Link`s.

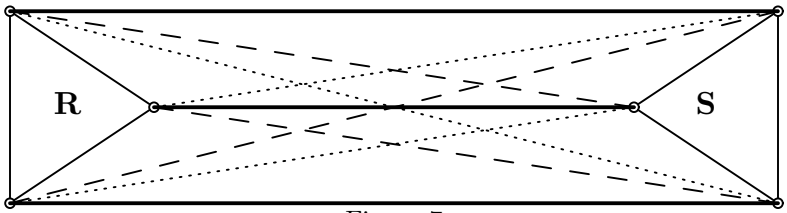


Figure 7

Intersections, the other means to require equality, are of no help because intersections are limited to triads and expressing the above with intersection would require equality across all six coordinates of **R** together with **S**.

The solution to this problem requires encoding triads in **R** and **S** as single values – the same encoding that supports reification. The term "tag" is used to name this encoding because it has an implementation flavor suitable for the algebra, avoiding semantic and philosophical concerns that accompany "reify". Tagging has been extensively studied in the context of binary relations in [14], which provided the seed of this paper.

3.10 *Definition*: The function τ is a *tagging function*, satisfying $\tau\colon \mathcal{D} \times \mathcal{D} \times \mathcal{D} \overset{1-1}{\to} \mathcal{D}$ and π_0, π_1, and π_2 are three *projection functions*, such that $\pi_i\colon \mathcal{D} \to \mathcal{D}$. These functions correspond so that, for arbitrary $x \in \mathcal{D}$ for which the π_i are defined, $x = \tau(\pi_0(x), \pi_1(x), \pi_2(x))$. In addition to being $1-1$, τ may not "have cycles." That is, there is no value a which is itself the value of a (non-trivial) τ expression involving a.

As is evident above, there is no stipulation that the π_i be total. The exact specification of the functions π_i determines whether or not the encoding is well-founded[2], so either case is available under this formalism. The "acyclic" restriction on τ is of course especially important if the encoding is well founded. Consistent with the usage described above, we will interpret subscripts to π_i mod 3.

3.11 *Definition*: The *tagging operator* $\mathcal{T}$ maps triadic relations into their fully tagged versions. That is,

$$\mathcal{T}(\mathbf{R}) \stackrel{\text{def}}{=} \{ \begin{matrix} & \tau(a,b,c) & \\ \tau(a,b,c) & & \tau(a,b,c) \end{matrix} : \begin{matrix} & b & \\ a & & c \end{matrix} \in \mathbf{R} \}$$

Each occurrence of $\mathcal{T}$ expands the active domain. The acyclicity of τ again comes to the rescue, in that the active domain of an expression with k occurrences of τ is within k compositions of $\mathcal{T}$ on the EDB values.

It is of course necessary to be able to access the values of the components of a tag. The *untag* (or *unpack*) operator $\mathcal{U}$ provides this capability. Note that $\mathcal{U}$ cannot unpack a tuple either if that tuple does not have the same values in each component or if that value is not in the range of τ.

3.12 *Definition*: For $i, j \in \{0, 1, 2\}$, $\mathcal{U}_{i,j}(\mathbf{S}) \stackrel{\text{def}}{=}$

$$\{ \begin{matrix} & x_1 & \\ x_0 & & x_2 \end{matrix} : \exists y_0, y_1, y_2 [\tau(y_0, y_1, y_2)^3 \in S \ \& \ \tau(y_0, y_1, y_2) = x_{i+1} = x_{i+2} \ \& \ x_i = y_j] \}$$

3.13 *Definition*: Overloading $\mathcal{T}$ and $\mathcal{U}$, $\mathcal{T}_{i,j}(\mathbf{R}) \stackrel{\text{def}}{=} \mathcal{U}_{i,j}(\mathcal{T}(\mathbf{R}))$ and

$$\mathcal{U}(\mathbf{S}) \stackrel{\text{def}}{=} \begin{matrix} & \mathcal{U}_{1,1}(\mathbf{S}) & \\ \mathcal{U}_{0,0}(\mathbf{S}) & & \mathcal{U}_{2,2}(\mathbf{S}) \end{matrix} = \{ \begin{matrix} & b & \\ a & & c \end{matrix} : \tau(a,b,c) \in \mathbf{S} \}$$

Obviously, $\mathcal{U}(\mathcal{T}(\mathbf{R})) = \mathbf{R}$ (but $\mathcal{T}(\mathcal{U}(\mathbf{R})) \subsetneq \mathbf{R}$ unless $\mathbf{R}$ is already symmetric).

This completes the definition of Trirel. Because all of the above operators introduce only finitely many new values, the following is immediate through an induction on expression construction.

3.14 *Definition*: A Trirel expression $\mathcal{E}$ is *safe* if it always returns a finite result when evaluated over finite EDB (input) relations.

3.15 *Proposition*: Trirel is safe.

The set of operators introduced above is obviously not minimal. We agree with Peirce that "superfluity here, as in many other cases in algebra, brings with it great facility in working."[10, p. 191] The one occasion where this superfluity is more an aesthetic issue than a notational convenience is in having distinct tag and untag operations (instead of having the $\mathcal{T}_{i,j}$, from 3.13, as primitive). Thus reification is a single primitive step rather than a construction, as is done in the binary case in [14].

4. Constants and Operators

The goal of this section is to support the claim that the trijoin operator is indeed "more fundamental" than the other operators. Although trijoin cannot alone express all other operators, it can express all the monotonic operators (that is, all operators except $-$) when used in conjunction with a few *constant* relations. These constants are infinite, but only finite subsets are required in the evaluation of any expression based on Trirel operators. The idea of such constants dates back to Tarski[12], who introduced four binary relations: universality (all pairs), identity (all equal pairs), diversity (all unequal pairs), and empty. He considered these four akin, even though three are infinite and the fourth is as small as a set can be.

This section first defines primitive constants $\triangle_{i,j}$and $\mathcal{A}$. Then it defines a few additional constants in terms of the $\triangle_{i,j}$and $\mathcal{A}$, only using trijoin. Then it proves the major result of this section alluded to above. Finally, it briefly returns to the issue of inequality selection.

4.1 *Definition*: For $i, j \in \{0, 1, 2\}$,

$$\triangle_{i,j} \stackrel{\text{def}}{=} \{ \begin{smallmatrix} x_1 \\ x_0 \; x_2 \end{smallmatrix} : x_0, x_1, x_2 \in \mathcal{D} \;\&\; x_i = \pi_j(x_{i+1}) \;\&\; x_{i+1} = x_{i+2} \}$$

The effect of $\triangle_{i,j}$ is to match up domain values in the i^{th} position of a triangle with the j^{th} position encoded within the other components of the triangle, interpreted as a tag. Relaxing the finiteness restrictions, $\triangle_{i,j} = \mathcal{U}_{i,j}(\mathcal{T}(\mathcal{D}^3))$. For example, $\triangle_{0,1}$ is $\begin{smallmatrix} \tau(x,y,z) \\ y \quad \tau(x,y,z) \end{smallmatrix}$ $= \{(y, t, t) | \exists x, z[t = \tau(x, y, z)]\}$. In a sense, the three tagging sets $\triangle_{i,i}$ are more natural, in that they have the untagged value in the "right place." The sets $\triangle_{i,i+1}$ and $\triangle_{i,i+2}$ are merely rotations of $\triangle_{i,i}$. However, since rotation will subsequently be expressed in terms of the $\triangle_{i,j}$, all these must be considered primitive.

4.2 *Definition*: The *alternative set* $\mathcal{A}$ explicitly expresses two-fold choices:

$$\mathcal{A} \stackrel{\text{def}}{=} \{ \begin{smallmatrix} w \\ u \; v \end{smallmatrix} : w = u \vee w = v \}$$

The notation Trirel$^\infty$ is used for the algebra with only trijoin, relative complement, and the constants $\triangle_{i,j}$and $\mathcal{A}$. Of course, positive Trirel$^\infty$ excludes "$-$". Trirel$^\infty$ is inherently unsafe.

There are a few other infinitary relations that are important. $\mathcal{I}$ is the triadic identity relation (and is more properly written $\mathcal{I}_\mathcal{D}$, the first and only time we shall explicitly indicate domain). $\mathcal{D}^3$ is the "universal" relation: all triples of values in $\mathcal{D}$. The notation $\triangle$ is overloaded to $\triangle_{i,j,j'}$ (used extensively in Sect. 6). Note that only i and not a corresponding i' is required in this extension because the j and j' components go into the i and $i + 1$ positions of the result. The following gives the Trirel definition of these constants.

4.3 *Definition*:

$$\mathcal{I} \stackrel{\text{def}}{=} \{ \begin{smallmatrix} a \\ a \; a \end{smallmatrix} : a \in \mathcal{D} \}, \quad \mathcal{D}^3 \stackrel{\text{def}}{=} \{ \begin{smallmatrix} b \\ a \; c \end{smallmatrix} : a, b, c \in \mathcal{D} \}$$

$$\triangle_{i,j,j'} \stackrel{\text{def}}{=} \{ \begin{smallmatrix} & x_1 & \\ x_0 & & x_2 \end{smallmatrix} : x_0, x_1, x_2 \in \mathcal{D} \;\&\; x_i = \pi_j(x_{i+2}) \;\&\; x_{i+1} = \pi_{j'}(x_{i+2}) \}$$

While these could be postulated as primitive, they can equally well be defined in terms of the various $\triangle$ constants. The following are easily verified.

4.4 *Lemma*:

$$\mathcal{I} = \begin{array}{ccc} & \triangle_{1,0} & \\ \triangle_{0,0} & \text{—} & \triangle_{2,0} \end{array}, \quad \mathcal{D}^3 = \begin{array}{ccc} & \triangle_{1,1} & \\ \triangle_{0,0} & \text{—} & \triangle_{2,2} \end{array}, \text{ and}$$

$$\triangle_{i,j,j'} = \mathtt{trijoin}(X_0, X_1, X_2), \text{ where } X_i = \triangle_{i,j} \;\&\; X_{i+1} = \triangle_{i+1,j'} \;\&\; X_{i+2} = \mathcal{I}$$

The definitions of $*_i$ and $\mathtt{trijoin}_i^*$, which only require $\mathtt{trijoin}$, $\mathcal{I}$, and $\mathcal{D}^3$, carry over from Sect. 3.

4.5 *Theorem*: Let $\mathcal{E}$ be an expression in positive Trirel. Then there is an equivalent $\mathcal{E}^\star$ in positive Trirel$^\infty$.

Proof: The preceding and following lemmas cover all the cases for the operators in Trirel. The theorem follows by simple induction using these lemmas.

4.6 *Lemma*:

$$\mathcal{T}(\mathbf{R}) = \begin{array}{ccccc} & & \triangle_{0,1} & & \\ & \mathbf{R} & & & \\ \triangle_{1,0} & \text{–} & \triangle_{1,2} & \text{——} & \mathcal{I} \end{array} \quad \text{and } \mathcal{T}_{2,j}(\mathbf{R}) = \begin{array}{ccccc} & & \triangle_{0,1} & & \\ & \mathbf{R} & & & \\ \triangle_{1,0} & \text{–} & \triangle_{1,2} & \text{——} & \triangle_{2,j} \end{array}$$

Proof:

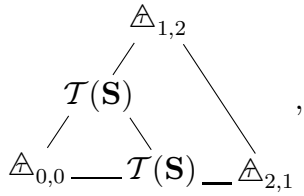
$$\text{Let } \widehat{\mathbf{R}} \stackrel{\text{def}}{=} \begin{array}{ccc} & \mathbf{R} & \\ \triangle_{1,0} & \text{–} & \triangle_{1,2} \end{array} = \begin{array}{ccc} & b & \\ a & \text{———} & c \\ \tau(a,b,c) & \text{-} \tau(a,b,c) & \text{-} \tau(a,b,c) \end{array} = \{ \begin{smallmatrix} & b & \\ \tau(a,b,c) & & \tau(a,b,c) \end{smallmatrix} : \begin{smallmatrix} a & & c \end{smallmatrix} \in \mathbf{R} \}$$

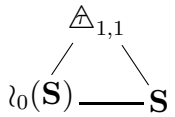
$$\text{then } \begin{array}{ccc} & \triangle_{0,1} & \\ \widehat{\mathbf{R}} & \text{———} & \mathcal{I} \end{array} = \begin{array}{ccccc} & & \tau(a,b,c) & & \\ & b & \text{——} & \tau(a,b,c) & \\ \tau(a,b,c) & \text{–} & \tau(a,b,c) & \text{——} & \tau(a,b,c) \end{array} = \mathcal{T}(\mathbf{R})$$

The proof for $\mathcal{T}_{2,j}$ is almost the same and for the other variants of $\mathcal{T}$ merely a rearrangement and reindexing of the expression.

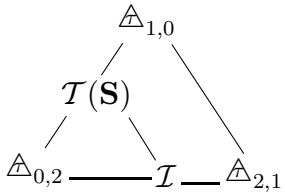
4.7 *Lemma*:

$$\wr_0(\mathbf{S}) = \quad , \qquad \mathcal{U}_{1,1}(\mathbf{S}) =$$





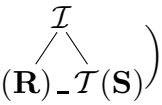
and $\rho(\mathbf{S}) =$



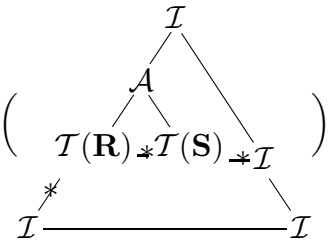
The use of $\wr_1$ in the implementation of $\mathcal{U}$ above guarantees, with the properties of $\mathbb{\Delta}_{0,0}$, that only triples in $\mathbf{S}$ that have the same value in each coordinates are unpacked.

4.8 *Lemma*:

$$\mathbf{R} \cap \mathbf{S} = \mathcal{U}\Big($$



$$\Big) \text{ and } \mathbf{R} \cup \mathbf{S} = \mathcal{U}\Big($$



$$\Big)$$

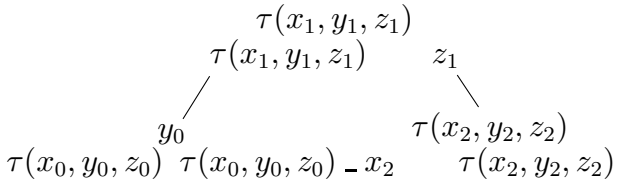
This completes the proof of Theorem 4.5. Extension to relative complement/negation is immediate. Note that these constants are tightly linked to operators that generate finite sub-instances of the respective constants, as illustrated by $\mathcal{I}$ and $\mathcal{I}(\mathbf{R})$. Uses of $\mathbb{\Delta}_{i,j}$, $\mathcal{I}$, *etc.* can be replaced by the corresponding operators; using the constants makes the expressions somewhat more readable.

Finally, Trirel$^\infty$ may be extended with an infinite diversity relation $\mathcal{N}$ (that is, $\mathcal{N} = \mathcal{D}^3 - \mathcal{I}$), in order to support inequality selection in extended positive Trirel$^\infty$. Note that $\mathcal{N}$ cannot be defined in positive Trirel$^\infty$. The addition of $\mathcal{N}$ accounts for all four constants considered essential by Tarski (empty is a constant defined trivially by explicit enumeration).

5. Finiteness of Trirel$^\infty$ Expressions

As was observed in Prop. 3.15, Trirel is safe. However, the presence in Trirel$^\infty$ of the infinite relations $\mathcal{A}$ and the various $\triangleq_{i,j}$ naturally raises the question whether the result of evaluating some particular Trirel$^\infty$ expression is finite or infinite. Obviously finiteness may be dependent up data values; for example, $*_i(\mathbf{R})$ is finite (indeed empty) only when $\mathbf{R}$ is empty. However, it is possible tell whether an expression will always yield finite results; we now sketch an algorithm to do so. The algorithm will presented in three phases – initialization, constraint inferencing, and finiteness inferencing – interwoven with explanation and verification.

There are two key observations: EDB relations impose finiteness restrictions and these restrictions are fully revealed by exploring the expression graph. One potential source of difficulty is the possibility of infinite exploration. For example, consider the expression $\texttt{trijoin}(\triangleq_{1,1},\triangleq_{2,2},\triangleq_{0,0})$, or graphically



A value, say b_0, starting out in the 0 and 1 positions of the top element, gets tagged, as expected, into $\tau(a_0, b_0, c_0)$ in the lower left, for arbitrary a_0 and c_0.[3] It is the continuation of such taggings, into $\tau(\tau(a_0, b_0, c_0), b_2, c_2)$, then into $\tau(a_1, b_1, \tau(\tau(a_0, b_0, c_0), b_2, c_2))$, and so on indefinitely, that appears to cause a problem. However, an instance satisfying this condition could only exist if $b_0 = \tau(a_1, b_1, \tau(\tau(a_0, b_0, c_0), b_2, c_2))$, which is not allowed by Def. 3.10. Such unsatisfiability corresponds to cycles in a dependency graph and thus is easily determined.

Initialization:

Begin with an expression $\mathcal{E}$ in positive Trirel$^\infty$. Constants other than $\triangleq_{i,j}$ and $\mathcal{A}$ are replaced as in Lemma 4.4. To each coordinate m of each relation occurrence in $\mathcal{E}$, assign the symbolic value v_m. These symbolic values are unique except that the two identical coordinates of a $\triangleq_{i,j}$ share the same symbol. Furthermore, v_0, v_1, and v_2 are reserved as labels for the three output nodes of the expression.[4] The symbolic values assigned to EDB coordinates should be distinguishable from coordinates of $\mathcal{A}$ or $\triangleq_{i,j}$ (either directly in the symbol or through some additional bookkeeping); we will call the former *EDB variables*. The symbol "*" may be assigned during subsequent phases; it indicates a don't care (and hence has the potential cause to an infinite result).

There are two sources of constraints: from the relations and from the trijoin operations. A constraint from an occurrence of $\triangleq_{i,0}$ arises between the coordinate corresponding

[3] Value subscripts in this example correspond to variable subscripts in the above graphical expressions. Thus b_0 is the value assigned to y_0, and so on.

[4] These two rules on labels mean that an expression consisting only of an $\triangleq_{i,j}$ cannot be labeled, a trivial case we ignore.

to a single value single value (in position i of $\triangle_{i,0}$), which is labeled v_m, and a corresponding τ term, labeled v_n. The constraint is $v_n = \tau(v_m, *, *)$; constraints for $\triangle_{i,1}$ and $\triangle_{i,2}$ are the obvious permutations. Two constraints arise from an $\mathcal{A}$, but in separate sets of constraints. Say the coordinates of $\mathcal{A}$ are (ℓ, m, n). $\mathcal{A}$ then produces two set of constraints, each a copy of the existing constraints plus $v_m = v_\ell$ in one case and $v_m = v_n$ in the other.[5] Overall this creates several sets of constraints, doubling the number of these sets for each occurrence of $\mathcal{A}$.

The second source of constraints is from `trijoin` operations. A formal definition would induct on subexpressions to define pairs of "subexpression corners". However, an intuitive approach, viewing the expression as a graph as done above, is sufficient. Each edge arising from a `trijoin` and expressing an equality condition between nodes m and n gives rise to the constraint $v_m = v_n$.

Constraint Inferencing:

For each set of constraints (that is, for one distinct choice for each relevant OR arising from an $\mathcal{A}$), iterate the following until no further changes occur. There are two cases, depending upon the form of the constraint. If there are any cases of the first sort, pick one of those.

$v_m = v_n$: If this is a tautology, that is $m = n$, it is just dropped. Assume, without loss of generality, that v_n arises from an EDB relation unless neither do. Replace every occurrence of v_m in other constraints with v_n. Retain the fact that $v_m = v_n$ but exclude it from further inferencing (within this set of constraints).

$v_m = \tau(e_0, e_1, e_2)$: If v_m is an EDB variable, exclude this constraint from further choice in this iteration (but process it in substitutions as described below). If there is another constraint of the form $v_m = \tau(e'_0, e'_1, e'_2)$, the two constraints are coalesced into the single $v_m = \tau(e^\star_0, e^\star_1, e^\star_2)$, where each $e^\star_i$ is defined according to the following table. The rules in the table are considered in the given order (thus, for example, the situation where $e_0 = v_n$ and $e'_0 =$ "$*$" is handled in the second subcase, so that e in the third subcase is never merely "$*$").

e_i	e'_i	$e^\star_i$	action
$*$	e	e	
e	$*$	e	
v_n	e	v_n	add constraint $v_n = e$
e	v_n	v_n	add constraint $v_n = e$
$\tau(-)$	$\tau(-)$	recurse	

Repeat the above coalescing until only one constraint has v_m on the left. Then replace each occurrence of v_m in other constraints by the right hand side. If any substitution creates a circular dependency, that is, if some v_ℓ occurs as the unique variable on the left and also in the expression on the right, immediately terminate with failure - the set of constraints is not satisfiable as discussed above.

5.1 *Lemma*: The above iteration terminates.

[5] This is the same as the observation that $\mathcal{A} = \{\,(x, y, y)\,\} \cup \{\,(y, y, z)\,\}$ and that `trijoin` distributes across union.

Proof: Each iteration removes one variable from consideration; this can only happen finitely often. The fact that no variable may be defined in terms of itself means that the sub-iteration of the second case always terminates.

This first iteration produces, for each variable (except possibly v_0, v_1, and v_2), a symbolic expression which characterizes that variable. Each non-EDB variable occurs at most once on the left hand side of a constraint, while EDB variables may occur several times.

Finiteness Inferencing:

A second iteration now deduces finiteness or non-finiteness. This second iteration is based upon the observation that finiteness can be imposed on v_n in two ways: first when the expression characterizing v_n has only terms known to be finite and second when v_n occurs in an expression characterizing a variable already known to be finite. As an example of this second rule, consider $v_4 = \tau(v_9, *, *)$, where v_4 is an EDB variable. Then the only possible values for v_9 are $\pi_0(v_4)$ as v_4 ranges over the corresponding values in the EDB. Thus v_9 is inferred to be finite. Indeed, an unnecessarily precise characterization of v_n, according to the second rule, could be given as a composition of the various π_j.

To implement the above observations, iterate the following two steps until no further changes occur:

for each variable v_m known to be finite (initially the EDB variables), infer that all variables occurring in the expression characterizing v_m are also finite.

for each expression comprised only of variables known to be finite, infer that the variable(s) characterized by that expression are also finite.

5.2 *Theorem*: A Trirel$^\infty$ expression $\mathcal{E}$ consisting only of EDB relations and $\triangleq_{i,j}$ constants satisfies one of three conditions, independent of the EDB:

i. $\mathcal{E}$ is always finite,
ii. $\mathcal{E}$ is always infinite, or
iii. $\mathcal{E}$ is always infinite or empty.

Proof: Recall that v_0, v_1, and v_2 correspond to the output of $\mathcal{E}$. Case *i* holds when the expressions defining v_0, v_1, and v_2 have only occurrences of variables known to be finite. Case *ii* holds if no variables known to be finite occur. And case *iii* holds otherwise (*i.e.*, the defining expressions has a mixture of finite and infinite atoms).

The above construction has implications for the evaluation of a (positive) Trirel$^\infty$ expression. Any $\mathcal{D}$ value may replace a don't-care (that is, an occurrence of "$*$" in a constraint), so even an infinite result has a finite representation. Furthermore, considering that a don't-care evaluates equal to any $\mathcal{D}$ value allows a finite evaluation, even of an expression with an infinite result.

Theorem 5.2 cannot handle expressions containing $\mathcal{A}$. Such expressions are, in effect, unions of the cases of Thm. 5.2. Consider the expression $\texttt{trijoin}(\mathcal{T}(\mathcal{E}_i), \mathcal{A}, \mathcal{T}(\mathcal{E}_{iii}))$ (encoding the union of sets defined by $\mathcal{E}_i$ and $\mathcal{E}_{iii}$), where $\mathcal{E}_i$ and $\mathcal{E}_{iii}$ match cases *i* and *iii* of the theorem respectively. Given a particular EDB, if $\mathcal{E}_{iii}$ is empty, the entire expression is finite; otherwise it is infinite. However, the algorithm does in fact track when this happens.

5.3 *Theorem*: If $\mathcal{E}$ contains EDB relations, $\triangleq_{i,j}$, of $\mathcal{A}$, the possible cases include *i.–iii.* above plus

iv. $\mathcal{E}$ is equivalent to a union of expressions matching cases *i.–iii.*.

While Thm. 5.3 is less satisfactory than Thm. 5.2, it still provides a definitive characterization of those expressions whose output is always finite The same may be said about expressions that include relative complement ("−"), in that $\mathcal{E}_1 - \mathcal{E}_2$ is certain to be finite if $\mathcal{E}_1$ is. However, it is possible that infinite $\mathcal{E}_1$ and $\mathcal{E}_2$ cancel each other out.

6. Equivalence of Trilog and Trirel

Two formalisms dealing with triadic relations have been discussed thus far: the operational Trirel in detail and the declarative Trilog more cursorily. It would be nice to have exact equivalence between these, paralleling Codd's equivalence of relational algebra and relational calculus, but the presence of tagging makes an exact parallel impossible. However, a restricted equivalence does hold. The first step of this equivalence is to show that Trirel can simulate Trilog.

To ease readability of this section, infinitary constants are are used for their operator forms; such constants are to be replaced as discussed toward the end of Sect. 4.

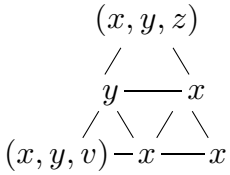
6.1 *Theorem*: For every positive Trilog program `prog`, there exists a positive Trirel expression $\mathcal{E}_{\texttt{prog}}$ that computes `result` of `prog` when given the same EDB.

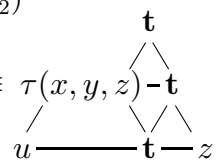
Proof: Because Trilog programs are not recursive, the rules may be ordered such that the relation in the head of each rule does not occur in the body of any preceding rule. This is equivalent to the dependency graph technique of [13]. If a relation `r` occurs in the head of more than one rule, give these occurrences new, unique names and add a new rule defining `r` as the union of all these just introduced relations. Then this ordered, renamed program is translated one rule at a time according to lemmas 6.3 and 6.4, which deal with the two cases of rule formation.

6.2 *Lemma*: The Trirel expression $\mathcal{L}$, defined below, is such that

$$\mathcal{L}(\mathbf{Q}) \;=\; \{\tau(\tau(x,y,v),u,z) : (\tau(x,y,z),u,v) \in \mathbf{Q}\}$$

Proof: The construction of $\mathcal{L}$ is given in a sequence of steps, each step showing first the desired set and then an expression yielding that set. Variable bindings are implicitly global until $\mathbf{Q}$ is introduced in step $\mathcal{L}_5$. The value $\tau(\tau(x,y,z),u,v)$ occurs frequently and is abbreviated by $\mathbf{t}$. The expressions for $\mathcal{L}_2$ and $\mathcal{L}$ shown in graphic form as well as typical algebraic notation as partial explication of the transformations implemented here.

step	desired result	expression
$\mathcal{L}_1$	$\{(\tau(x,y,z),u,v)\}$	$\texttt{trijoin}(\mathcal{T}(\mathcal{D}^3),\mathcal{D}^3,\mathcal{D}^3)$
$\mathcal{L}_2$	$\{(\tau(x,y,v),\tau(x,y,z),x)\}$	$\texttt{trijoin}(\triangleq_{1,1,0},\triangleq_{2,0,1},\mathcal{I}) \equiv$ 

$\mathcal{L}_3$	$\{(\tau(x,y,v),\tau(x,y,z),v)\}$	$\mathtt{trijoin}_0^*(\triangle_{2,2},\mathcal{L}_2,\mathcal{I})$
$\mathcal{L}_4$	$\{(\tau(x,y,v),\mathbf{t},\mathbf{t})\}$	$\mathtt{trijoin}(\mathtt{trijoin}(\mathcal{L}_3,\triangle_{0,0},\mathcal{I}),\mathcal{T}(\mathcal{L}_1),\mathcal{T}(\mathcal{L}_1))$
$\mathcal{L}_5$	$\{(\mathbf{t},u,\mathbf{t}):(\tau(x,y,z),u,v)\in\mathbf{Q}\}$	$\mathcal{T}_{1,1}(\mathbf{Q}\cup\mathcal{T}(\mathcal{L}_1))$
$\mathcal{L}_6$	$\{(\mathbf{t},\mathbf{t},z)\}$	$\mathtt{trijoin}(\mathcal{D}^3,\ \triangle_{2,1},\ \triangle_{2,2})$
$\mathcal{L}$	$\{(\tau(x,y,v),u,z):$ $(\tau(x,y,z),u,v)\in\mathbf{Q}\}$	$\mathtt{trijoin}_1^*(\mathcal{L}_4,\mathcal{L}_5,\mathcal{L}_6)\equiv$ 

6.3 *Lemma*: Let $\mathtt{R}(p_0,p_1,p_2)$ `:-` $\mathtt{S}_1(q_{1,0},q_{1,1},q_{1,2})$ `&` $\cdots$ `&` $\mathtt{S}_k(q_{k,0},q_{k,1},q_{k,2})$ be a Trilog statement defined over a set of variables $\mathcal{V}=\{q_{j,i}:1\le j\le k\&1\le i\le 3\}\cup\{p_i:1\le i\le 3\}$. Then there is an equivalent Trirel expression $\mathcal{E}$ that computes the value of R given instances of the $\mathtt{S}_j$.

Proof: It is sufficient to give $\mathcal{E}_i$, $i\in\{0,1,2\}$, respectively containing triples of the form $(\mathtt{s},\mathtt{s},\mathtt{p})$, where s encodes an assignment to all variables in $\mathcal{V}$ satisfying the rule body and p is an assignment to p_i is consistent with s. With these $\mathcal{E}_i$,

$$\mathcal{E}=\begin{array}{ccc} & \wr_0(\mathcal{E}_1) & \\ \swarrow & & \searrow \\ \wr_1(\mathcal{E}_0) & \text{——} & \mathcal{E}_2 \end{array}$$

Without loss of generality, we assume that each variable occurs at most once as a parameter to any one $\mathtt{S}_j$. If that is not the case, that is, some v occurs more than once as a parameter to some $\mathtt{S}_j$, replace all but one of these v's by new, unique variables (in essence, don't cares) and, in the algebraic expression, intersect $\mathtt{S}_j$ with an expression forcing equality at the respected components.

The first step is to build a structure that encodes the contents of the S's. In the following the notation (x_j,y_j,z_j) is always restricted to tuples in $\mathtt{S}_j$. Define

$Svect_1 \stackrel{\text{def}}{=} *_0(*_2(TAG(\mathtt{S}_1))) = \{(*,\tau(x_1,y_1,z_1),*)\}$

and inductively, for $j>1$,

$Svect_j \stackrel{\text{def}}{=} \mathtt{trijoin}(*_1(*_2(Svect_{j-1})),TAG(\mathtt{S}_j),\mathcal{D}^3)=\{(s,\tau(x_j,y_j,z_j),*):s\in Svect_{j-1}\}.$

Thus $Svect_k$ encodes all structures of the following form, where the "w_j" are distinct place-holder variables:

$$((\cdots((w_0,(x_1,y_1,z_1),w_1),(x_2,y_2,z_2),w_2),\cdots),(x_k,y_k,z_k),w_k).$$

Similarly, we want to construct comparable structures that enforce agreement among the appropriate positions for each distinct variable $v\in\mathcal{V}$. Note that, in the following, v as a superscript to M is the variable name while v in the expository set expressions ranges over the possible values of the variable so named. This possibility is resolved to certainty in M_k^v. Also, $\hat{x}_j$ is v if $q_{j,0}$ is v and is * otherwise. Similar usage holds for $\hat{y}$ and $\hat{z}$. Parallel

to the above definition of $Svect$, define

$$\tilde{M}_j^v \stackrel{\text{def}}{=} *_0(\triangleq_{i,2}) \text{ if } v = p_{j,i},\ \mathcal{D}_3 \text{ otherwise}$$
$$M_0^v \stackrel{\text{def}}{=} \tilde{M}_0^v$$
$$M_j^v \stackrel{\text{def}}{=} \mathcal{L}(*_2(M_{j-1}^v)) \cap \tilde{M}_j^v$$

Thus $M_1^v = \{(*, (\hat{x}_1, \hat{y}_1, \hat{z}_1), v)\}$ and, for $j > 1$, $M_j^v = \{(s, (\hat{x}_j, \hat{y}_j, \hat{z}_j), v) : s \in *_2(M_{j-1}^v)\}$.

Thus M_k^v encodes the set of all structures which agree on the positions where v occurs and

$$\mathcal{E}_i = \mathcal{T}_{2,2}(Svect_k \cap \bigcap_{\substack{v \in V \\ v \neq p_i}} *_2(M_k^v) \cap M_k^{p_i})$$

as required.

6.4 *Lemma*: Let $\mathtt{R}(p_0, p_1, p_2)$:- $\mathtt{S}_1(q_{1,0}, q_{1,1}, q_{1,2}) \vee \cdots \vee \mathtt{S}_k(q_{k,0}, q_{k,1}, q_{k,2})$ be a Trilog statement where $q_{j,i} \in \{p_0, p_1, p_2\}$. Then there is an equivalent Trirel expression $\mathcal{E}$ that computes the value of R given instances of the $\mathtt{S}_j$.

Proof: A union of the $\mathtt{S}_j$, with relevant flips and rotations, suffices. Recall that p_0, p_1, and p_2 must occur within every $\mathtt{S}_j$ of the union.

This lemma completes the proof of Theorem 6.1. Now let us consider the other side of the equivalence issue: the degree to which Trilog can implement Trirel.

Because Trilog is inherently conservative, in that it does not introduce new values, it cannot implement tagging. One might consider restricting the output of Trirel expressions to triples over the active domain of the EDB. However, because all values in Trilog are atomic, it cannot implement any "higher order" algebraic operation, even something as simple as $\mathcal{T}(\mathbf{R}) \cap \mathbf{S}$. Thus the domain restriction must apply to inputs as well as outputs.

To this end, $\mathcal{D}$ is partitioned into $\mathcal{D}_b$ and $\mathcal{D}_t$, for "base domain" and "tagging domain". In particular, $\mathcal{D}_t \stackrel{\text{def}}{=} range(\tau)$ and $\mathcal{D}_b \stackrel{\text{def}}{=} \mathcal{D} - \mathcal{D}_t$. This partition induces a unique tree structure on any element of $\mathcal{D}$. That is, an element x of $\mathcal{D}_t$ is expanded into a node with three subtrees $\pi_0(x)$, $\pi_1(x)$, and $\pi_2(x)$, which are recursively expanded until elements of $\mathcal{D}_b$ are reached for the leaves. For the next theorem, the EDB is assumed to contain only values from $\mathcal{D}_b$ and any values containing elements of $\mathcal{D}_t$ are deleted from the output.

6.5 *Proposition*: Given a Trirel expression $\mathcal{E}$, it is possible to tell whether $\mathcal{E}$ produces no, some, or only values in $\mathcal{D}_b$.

Proof: The proof follows from constructions in the next theorem.

6.6 *Theorem*: For each positive Trirel expression $\mathcal{E}$, there exists a positive Trilog program $\mathtt{prog}_{\mathcal{E}}$ such that `result` of `prog` is the value of $\mathcal{E} \cap \mathcal{D}_b^3$ when given the same input instances over $\mathcal{D}_b^3$.

Proof: Let $\mathbf{S}_1, \cdots, \mathbf{S}_k$ be the occurrences of relations in $\mathcal{E}$; note that one relation may occur as multiple $\mathbf{S}_j$'s. Because unions are handled separately in Trilog, it is necessary to separate the cases whether or not unions occur in $\mathcal{E}$.

Case: $\mathcal{E}$ is a "simple expression" without $\cup$

The tree structure on elements of $\mathcal{D}$ carries over to $\mathcal{E}$. This and the exclusion of $\cup$ imply that, for each coordinate of the result of an expression, tagged values are constructed in one and only one way. Thus a tagged expression (including taggings of taggings) may be simulated with a finite number of Trilog variables.

To begin the construction, each subexpression $\mathcal{F}$ of $\mathcal{E}$ is associated with three sets of labels. At the leaves, these sets are singletons, but they merge moving up the tree. This merging occurs, in programming jargon, by reference and not by value. That is, when sets associated with two positions are merged, the result is not two sets (one for each position) but one set associated with both positions. Consequently, any subsequent merges propagate directly to all positions associated with a set. To be precise, this set association is defined on the recursive structure of subexpressions $\mathcal{F}$ of $\mathcal{E}$, with the following cases:

$\mathcal{F}$ is a leaf, that is $\mathbf{S}_j$ for some j: associate with each parameter position i of $\mathcal{F}$ the set $\{\langle j, i\rangle\}$.

$\mathcal{F} = \hat{\mathcal{F}} \cap \tilde{\mathcal{F}}$: If the tree structures for $\hat{\mathcal{F}}$ and $\tilde{\mathcal{F}}$ do not match exactly, then $\hat{\mathcal{F}} \cap \tilde{\mathcal{F}}$ is empty. Otherwise, for each matching leaf in the trees of $\hat{\mathcal{F}}$ and $\tilde{\mathcal{F}}$, merge the two sets associated with those leaves.

$\mathcal{F}$ is defined using `trijoin` or $\mathcal{I}$: handled similarly in three cases corresponding to the three sides.

$\mathcal{F}$ is defined using $\wr$ or ρ: just reshape the tree structure.

$\mathcal{F}$ is defined using $\mathcal{T}$ or $\mathcal{U}$: these obviously affect the tree structure but do not change the associated sets.

Finally, the results of expression $\mathcal{E}$ are associated with sets. In particular, for each coordinate i of $\mathcal{E}$, examine its tree structure. If that tree is just a base value, it has an associated set and $\langle 0, i\rangle$ is added to that set. If the tree structure is not a base value, then $\mathcal{E}$ will always produce values in $\mathcal{D}_t$, which will be deleted. This observation is also the heart of the proof of Proposition 6.5.

The associated sets index relation coordinates that are joined, since these sets are merged whenever they overlap. Now construct the body of a Trilog rule as a conjunction of the $\mathbf{S}_j$'s. Assign a unique variable to each set and place that variable in each location in that set. That is, if v is assigned to a set containing $\langle j, i\rangle$, the i th parameter of S_j in the conjunction is v. The head of the rule is either `result`, if the expression stands by itself, or is a new, unique name if the expression is a subexpression of a union, as discussed below. In either case, the i th parameter of the head is the variable assigned to the set containing $\langle 0, i\rangle$.

Case: $\mathcal{E}$ contains $\cup$

Each expression will now correspond to a finite union of trees relating to simple expressions. Most algebra operations are accomplished by distributing across $\cup$. For example, if $\mathcal{E}$ is $\hat{\mathcal{E}} \cap \tilde{\mathcal{E}}$, and $\hat{\mathcal{E}}$ and $\tilde{\mathcal{E}}$ correspond to $\widehat{\text{tree}}_1 \cdots \widehat{\text{tree}}_k$ and $\widetilde{\text{tree}}_1 \cdots \widetilde{\text{tree}}_\ell$ respectively, then $\mathcal{E}$

corresponds to

$$\bigcup_{\substack{1\le i\le k\\ 1\le j\le \ell}} \widehat{\text{tree}}_i \cap \widetilde{\text{tree}}_j$$

When a simple subexpression evaluates empty, it is dropped from the union. Each simple subexpression will be encoded in a single Trilog rule as in the preceding case. Adding one additional rule unioning the simple subexpressions completes the construction.

7. Application to RDF

This section briefly considers the application of Trirel to RDF, first to the model theory[16] and then to approaches for querying RDF.

RDF model theory contains variety of closure rules, rules that are applied to close any piece of RDF syntax E. For example, rule rdfs2 states that if E contains both (xxx,aaa,yyy) and (aaa,`[rdfs:domain]`,zzz), then (uuu,`[rdf:type]`,zzz) should be added to E. While the algebra itself does not provide a mechanism for doing updates, it is easy to use the algebra to define the increment to E. In particular, let **C** be {(∗, [`rdfs : range`], [`rdf : type`])} (*i.e.* **C** is defined by applying $*_0$ to an explicitly enumerated constant). Then rule rdfs2 adds to E

$$\wr_0(\begin{array}{ccc} & \wr_0(\text{E}) & \\ \text{E} & & \mathbf{C} \end{array}) = \wr_0(\begin{array}{ccccc} & & zzz & & \\ & & / & \backslash & \\ & aaa & - & [\texttt{rdfs : range}] & \\ / & \backslash & / & & \backslash \\ uuu & - & * & - & [\texttt{rdf : type}] \end{array}) = \begin{array}{ccc} & [\texttt{rdf : type}] & \\ uuu & & zzz \end{array}.$$

Trirel is not sufficient for RDF model theory since the latter includes a transitive closure (rule rdfs5). Any solution that augments Trirel with transitive closure will handle this problem. The fixed arity of Trirel is beneficial here; with relations of arbitrary arity, the question is which pair attributes encodes the binary relationship to be closed. With ternary relations, there is only one "extra" attribute and that in fact is actually useful to label the relationships to be closed, as was seen in **Trans** of example 3.5. Thus we define an operator $\mathfrak{TC}$ (or the three rotations thereof) that computes the full closure of the operation of that example.

7.1 *Definition*: Let **R** be a triadic relation. Then

$$\mathfrak{TC}_1 \stackrel{\text{def}}{=} \{ {}_{x_0}{}^{\ell}{}_{x_k} | \exists x_1, \cdots, x_{k-1} [\&_{i=1}^{k} \, x_{i-1} \, {}^{\ell} \, x_i \in \mathbf{R}]\}$$

$\mathfrak{TC}_0$ and $\mathfrak{TC}_2$ are defined analogously or as rotations of $\mathfrak{TC}_1$.

Note that the above definition has an implicit existential for k or equivalently an unbounded union over sets parameterized by that k. The other common definition of transitive closure, as the fixed point of **Trans**, has a similar unbounded union nature.

Trirel may be used to express the core of SquishQL[9], a proposed language for querying RDF (with variants such as RDQL). In SquishQL queries, the `WHERE` clause is a collection of triplets, forming a template for the specified retrieval; this template immediately maps to a Datalog body. SquishQL not closed on triadic relations, however, in that SquishQL query may return tuples over an arbitrary list of attributes. Thus, similar to the construction of

$\mathcal{L}$ (Lemma 6.2), such arbitrary lists of values may be returned in encoded form. Theorem 6.6 may be used to translate arbitrary SquishQL queries to Trirel. On the other hand, this lack of closure may be considered a problem with SquishQL.

8. Conclusion

This paper has introduced an algebra, Trirel, over triadic relations. An essential characteristic of an algebra is closure – that is, all algebraic operations produce results from the same set as the inputs. Other interesting models of RDF queries are not algebras in this strict sense, even when query results are projected down to exactly three-element tuples. This is because all these models require intermediate constructions with more than three active elements[6] for certain queries. Hence their primitive operations do not collectively specify an algebra.

Trirel surmounts this problem with a mechanism for encoding triples of values in a single value. This mechanism thus supports a kind of reification. However, if the reified values are treated only as internal values (that is, reified values are not allowed in input or output), the encoding provides no additional query capability beyond other formalisms (RA, Datalog, FOL) suitably restricted to triples. This suggests that reification can be introduced where semantic considerations require it, without concern that this introduction would seriously impact the formalism in other ways.

Trirel is fully symmetric, unlike other approaches from databases [9,5] or logic [10,4]. Thus interpretation (*e.g.* mapping to labeled graphs) is entirely by convention, rather than being imposed by the algebra and its operators.

This work also illustrates the well-known tradeoff between complexities of representation and manipulation. That is, given suitable constant relations, the only operators necessary are join and relative complement.

Acknowledgments: The author is grateful for valuable comments and suggestions by Richard Martin and Dirk Van Gucht. He first heard about "triangles" from Chris Longyear[8] of the DEACON project.

An earlier version of this paper appears as "Triadic Relations: an Algebra for the Semantic Web" in the *Proceedings of 2nd International Workshop on Semantic Web and Databases* (SWDB 2004), to be published by Springer Verlag.

[6] The notion of "active element" is merely intuitive, roughly corresponding to the minimum number of variables in relational calculus or Datalog.

Bibliography

1 J. A.Craig, S. C. Berezner, H. C. Carney, and C. R. Longyear. Deacon: Direct english access and control. In *Proc. AFIPS Fall Joint Comput. Conf.*, pages 365–380, 1966.

2 P. Aczel. *Non-well-founded sets*. CSLI Lecture Notes 14. Stanford University, 1988.

3 E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

4 J. M. Dunn. A representation of relation algebras using Routly-Meyer frames. In C. A. Anderson and M. Zelëny, editors, *Logic, Meaning, and Computation*, pages 77–108. Kluwer Academic Publishers, 2001.

5 C. Gutierrez, C. Hurtaddo, and A. Mendelzon. Foundations of semantic web databases. In *ACM Principles of Database Systems*, 2004.

6 F. Harary. *Graph Theory*. Addison-Wesley, 1969.

7 M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. of ACM*, 42(4):741–843, 1995.

8 C. R. Longyear. Further towards a triadic calculus. *Journal of Cybernetics*, 2(1):50–65, (2):7–25, (3):51–78, 1972.

9 L. Miller, A. Seaborne, and A. Reggiori. Three implementations of squishql, a simple rdf query language. In *International Semantic Web Conference (ISWC)*, 2002.

10 C. S. Peirce. On the algebra of logic. *Amer. J. of Math.*, pages 180–202, 1885.

11 J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole, 2000.

12 A. Tarski. On the calculus of relations. *J. of Symbolic Logic*, 6(3):73–89, 1941.

13 J. D. Ullman. *Princ. of Database and KnowledgeBase Systems*, volume I – Fundamental Concepts. Computer Science Press, New York, 1988.

14 D. Van Gucht, L. V. Saxton, and M. Gyssens. Tagging as an alternative to object creation. In J. C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing for Advanced Database Systems*, pages 201–242. Morgan Kaufmann, 1994.

15 W3C. Defining n-ary relations on the semantic web: Use with individuals, 1999. `http://www.w3.org/TR/swbp-n-aryRelations`.

16 W3C. RDF Model Theory, 2002. `www.w3.org/TR/rdf-mt/`.

17 G. Yang and M. Kifer. On the semantics of anonymous identity and reification. In *DBLP 2002*, volume 2519 of *Lecture Notes in Computer Science*, pages 1047–1066. Springer, 2002.

Appendix

The following, from [15], is an XML representation of a variation of the cancer example above, specifically "Christine is diagnosed with breast cancer, with high probability". This illustrates the consequences of needing to identify every node, although identity is explicit rather than anonymous.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
   xmlns="http://protege.stanford.edu/swbp/diagnosis.rdf#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xml:base="http://protege.stanford.edu/swbp/diagnosis.rdf">
<rdfs:Class rdf:ID="Disease"/>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="Diagnosis_Relation"/>
<rdf:Property rdf:ID="has_diagnosis">
<rdfs:domain rdf:resource="#Person"/>
<rdfs:range rdf:resource="#Diagnosis_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="diagnosis_value">
<rdfs:range rdf:resource="#Disease"/>
<rdfs:domain rdf:resource="#Diagnosis_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="diagnosis_probability">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource="#Diagnosis_Relation"/>
</rdf:Property>
<Diagnosis_Relation rdf:ID="Diagnosis_Relation_1">
<diagnosis_probability
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >High</diagnosis_probability>
<diagnosis_value>
<Disease rdf:ID="Breast_Tumor"/>
</diagnosis_value>
</Diagnosis_Relation>
<Person rdf:ID="Christine">
<has_diagnosis rdf:resource="#Diagnosis_Relation_1"/>
</Person>
</rdf:RDF>
```