*Information Modelling and Knowledge Bases XXXII*
*M. Tropmann-Frick et al. (Eds.)*


*doi:10.3233/FAIA200824*

# A Natural Logic System for Large Knowledge Bases

Troels ANDREASEN [a], Henrik BULSKOV [a] and Jørgen FISCHER NILSSON [b]
[a] *Computer Science, Roskilde University, Denmark, {troels, bulskov}@ruc.dk*
[b] *Mathematics and Computer Science, Technical University of Denmark, Denmark, jfni@dtu.dk*

**Abstract.** This paper describes principles and structure for a software system that implements a dialect of natural logic for knowledge bases. Natural logics are formal logics that resemble stylized natural language fragments, and whose reasoning rules reflect common-sense reasoning. Natural logics may be seen as forms of extended syllogistic logic. The paper proposes and describes realization of deductive querying functionalities using a previously specified natural logic dialect called NaturaLog. In focus here is the engineering of an inference engine employing as a key feature relational database operations. Thereby the inference steps are subjected to computation in bulk for scaling-up to large knowledge bases. Accordingly, the system eventually is to be realized as a general-purpose database application package with the database being turned logical knowledge base.

**Keywords.** Natural Logic, Deductive Querying, Large Logical Knowledge Bases, Relational Data Base Operations

## 1. Introduction

In a number of papers [1, 2, 3, 4, 5, 6] we have proposed and analyzed a dialect of natural logic intended for knowledge base use. Natural logics are forms of formal logic that appear as stylized fragments of natural language. Therefore, a knowledge base consisting of natural logic sentences can be read and understood right away by domain specialists without understanding of predicate logic, description logic or logic programming.

In the applied dialect, called NaturaLog, one can state implicitly quantified relationships between introduced classes. In particular one can specify formal ontologies by means of the subclass relationship and partonomies by means of parthood relations. Moreover, one can introduce relations *ad libitum*. The natural logic NaturaLog basically covers a fragment of predicate logic with respect to expressivity, see also [7, 8, 9, 10, 11, 12]. Natural logics are further distinguished by their appealing to high level "natural" reasoning rules reminiscent of and departuring from the elementary reasoning rules known from syllogistic logic.

This paper focusses on the engineering of an appropriate inference engine serving diverse forms of deductive reasoning and querying. In order to address the

scaling-up problem encountered as computational intractability this paper proposes an inference engine using relational database operations such as selection and equi-join. Thereby logical consequences of knowledge base sentences are computed and recorded *en masse* rather than in a conventional stepwise fashion. We envisage applications e.g within the life-sciences employing quantitative complex models with thousands of classes structured into ontologies supplemented with a variety of class-class relations.

The paper is organized as follows: After having introduced NaturaLog in section 2, section 3 explains how sentences are encoded in a relational data base relation, and section 4 describes realization of the inference engine by computing and storing of relevant parts of the deductive closure using relational database query operations. Section 5 covers implementation and section 6 covers query issues.

## 2. The NaturaLog Natural Logic

Our natural logic comprises sentences that specify relationships between stated classes of not further described entities. Here we describe a somewhat simplified version of NaturaLog. For the currently full version we refer to [12]. Classes of not further described entities are introduced by common nouns such as cell, betacell, insulin, hormone. In the case of substances such as insulin the entities of the class may be conceived as portions of such substances.

The key class-class relationship is the well-known subclass relationship isa. Thus one can form sentences such as betacell isa cell and insulin isa hormone. The pertinent sentence form $C$ isa $D$ is actually a shorthand of every $C$ isa $D$. This is reminiscent of affirmative categorical sentences in syllogistic logic, cf. [2].

This sentence form is actually a special case of the more general NaturaLog key sentence form

every $C$ $R$ $D$

where $R$ is a linguistically transitive verb (or more generally a verb form) designating a binary relation. This is exemplified by the sentence every betacell produce insulin. Again, the determiner every is optional: betacell produce insulin is shorthand of every betacell produce insulin.

There is also the NaturaLog form

some $C$ $R$ $D$

in which presence of the determiner some is made mandatory.

These natural logic sentence forms have predicate logical construals: [ every ] betacell produce insulin in predicate logic becomes the rather incomprehensible $\forall x(\mathsf{betacell}(x) \rightarrow \exists y(produces(x,y) \wedge \mathsf{insulin}(y)))$, and in description-logical becomes the obscure $\mathsf{betacell} \sqsubseteq \exists \mathsf{produces.insulin}$. The sentence insulin isa hormone is explicated simply as $\forall x(\mathsf{insulin}(x) \rightarrow \mathsf{hormone}(x))$. The subclass relation isa is equipped with built-in rules yielding reflexivity and transitivity.

NaturaLog adopts existential import, meaning that for each mentioned class $C$ there is implicitly what in a predicate logical construal appears as the existential statement $\exists x C(x)$, see for instance [2, 11, 12]. As a consequence, there is no notion of empty class in our natural logic. Two classes are assumed disjoint

if they have no common subclass. Introduction of a common subclass makes the two classes overlapping, since their common subclass is bound to be non-empty. Thus classes at the outset are disjoint. This is in accord with the common default principle for scientific taxonomies and formal ontologies. We refer to [2, 12] for further discussion of the rationale.

NaturaLog in addition to simple classes designated by common nouns importantly affords compound terms comprising modifiers that restrict classes to subclasses. For instance, the sentence [ every ] cell that produce hormone reside-in gland comprises the compound term cell that produce hormone consisting of the class cell attributed the modifier that produce hormone linguistically in the form of a restrictive relative clause that $R$ $D$. The compound term cell that produce hormone thereby designates a subclass of cell. This conforms with the notion of generative ontologies introduced in [13] reflecting the recursive structure of the attached modifiers for generating evermore specialized subclasses.

To this end let us specify the recursively defined syntactic notion of compound concept terms:

> A concept term $Cterm$ is a class name $C$ as introduced above, optionally affixed a modifier taking form of either
> – a restrictive relative clause that $R$ $Cterm$, where $R$ is a verb form, or
> – a prepositional phrase $R$ $Cterm$, where $R$ is a preposition here also designating a binary relation.

Compound modifiers may further be combined by means of logical conjunction in NaturaLog as introduced in [12].

An essential aspect of natural logic is the provision of inference rules reflecting direct "natural" reasoning on the linguistic forms in sentences. It is important to stress that *the computational reasoning enabling deductive querying is accomplished directly on natural logic forms and thus are not conducted by reduction to predicate logic.* This is detailed in section 4. As a preliminary step we explain how NaturaLog sentences are represented in a database relation.

## 3. Encoding of Natural Logic in Data Base Relations

The NaturaLog sentences are to be represented in a database relation forming the knowledge base. In the simple sentences with no compound terms this is done straightforwardly in principle with tuples such as,

kb(..., every, betacell, produce, insulin)

for the sentence [ every ] betacell produce insulin. The tuple is stored in a relation kb and specifies values for attributes QUANT, SUB, REL, OBJ. However, to distinguish given sentences from derived and to separate propositions from definitions two additional attributes MODE and TYPE are added such that the full schema for the kb relation becomes:

kb(MODE, TYPE, QUANT, SUB, REL, OBJ)

where MODE can take the values given and deriv, TYPE can take the values prop and defin and QUANT is either every or some.

Sentences with compound terms are decomposed into simple sentences following the methods detailed in [2, 6, 12], that is, by introduction of appropri-

ate auxiliary class names representing the compound terms. As an example, the above sentence cell that produce hormone reside-in gland gives rise to the auxiliary class name cell-that-produce-hormone which has to be defined by two additional propositions, labeled "defin", giving altogether three propositions, that is, three database tuples

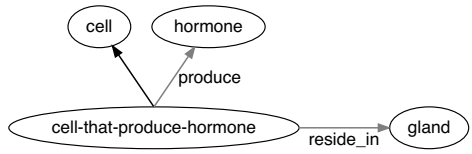


**Figure 1.** Example graph including three sentences

The sentences included in the graph in figure 1 will be represented by the following tuples in the kb-relation:

kb(given, defin, every, cell-that-produce-hormone, isa, cell)
kb(given, defin, every, cell-that-produce-hormone, produce, hormone)
kb(given, prop, every, cell-that-produce-hormone, reside-in, gland)

Notice that in the graph notation unlabelled arrows represent isa. The definitional contributions are logically explicated as "if-and-only-if":

$$\forall x(\mathsf{cell\text{-}that\text{-}produce\text{-}hormone}(x) \leftrightarrow \mathsf{cell}(x) \wedge \mathsf{producehormone}(x))$$

This reduction of recursive compound terms serves easing and streamlining of the computation of inference and query answers.

## 4. Inference through Data Base Query Operations

Let us begin considering a tiny natural logic knowledge base containing solely the two sentences betacell produce insulin and insulin isa hormone. Suppose we wish to know which kind of cells that produce hormone giving this information. To this end we might form the parameterized "query" sentence X produce hormone, aiming at obtaining answers as instantiation of the parameter variable X. However, no answer is obtained. This is because the common-sense expected answer X = betacell necessitates the introduction of inference rules capable of combining information in sentences according to logical consequence principles in the computation.

In [6, 12] we devised a collection of deductive inference rules for decomposed and encoded NaturaLog sentences in a knowledge base. The rules are specified in Datalog. This means that Datalog functions as a metalogic for NaturaLog, where also various forms of deductive querying are stated. Here, instead of Datalog we apply an ordinary relational database query language. The derived sentences are added to the database relation kb. In this way the query-relevant part of the deductive closure is materialized in a iterative pre-compilation involving the knowledge base in its entirety. In the course of this computation new compound terms being potentially relevant to queries are generated. This implies that subsequent deductive query answer computations reduce to mere sentence

retrieval from the knowledge base now being extended with materialized inferred propositions. For instance, for the above example with just two sentences, the inference engine adds the encoded form of the deduced sentence betacell produce hormone using the generalization monotonicity rule (5) described below.

This approach to computation replaces the common goal-driven top-down approach to derivations, known e.g. from logic programming, with a bottom-up pre-computation with caching drawing on the entire knowledge base. Thereby, backtracking computation is avoided at the expense of deriving sentences being irrelevant to the queries at hand. Thus time-consuming query reasoning is traded for compile-time computation of the deductive closure and storage use.

As indicated above, the inference engine is a mechanism that iteratively materializes (adds) new tuples to the kb relation by application of the inference rules. The NaturaLog inference rules are presented below along with examples of inferred propositions, that is, tuples that can be inferred from other tuples in the kb relation. The result of adding all propositions that can be inferred is a limited deductive closure of the knowledge base. For practical purposes this computation should not be performed tuple by tuple. Section 5 exemplify how materialization of inferred tuples can be performed using SQL bulk insertions.

### *4.1. Weakening and Dual Relationship*

As mentioned in section 2, all classes are non-empty (due to existential import), and therefore we have that any default every proposition can be weakened to a some proposition. This is expressed in the weakening rule:

$$\frac{\mathsf{every}\ C\ R\ D}{\mathsf{some}\ C\ R\ D} \tag{1}$$

allowing to make derivations such as:

$$\frac{\mathsf{kb(given,\ prop,\ every,\ betacell,\ produce,\ insulin)}}{\mathsf{kb(deriv,\ prop,\ some,\ betacell,\ produce,\ insulin)}}$$

Mathematically, each relation $R$ possesses an inverse relation $R^{-1}$. Therefore, for strictly logical reasons and given our pervasive principle of non-empty concepts, for $C\ R\ D$ (i.e. every $C$ $R$ some $D$) implicitly, as explained in chapter [12], we also have the dual

$$\mathsf{some}\ D\ R^{-1}\ \mathsf{some}\ C$$

Dual relations need to be explicitly specified in the knowledge base. For this purpose we introduce a database relation inv with the schema:

$$\mathsf{inv}(R,\ R_{inv})$$

Using this we can e.g. specify produced_by to be the inverse of produce:

$$\mathsf{inv(produce,\ produced\_by)}$$

The rule expressing the existence of an inverse is the following:

$$\frac{\mathsf{some}\ C\ R\ D}{\mathsf{some}\ D\ R_{inv}\ C} \tag{2}$$

where $R_{inv}$ is the inverse relation to $R$. As an example we have:

$$\frac{\mathsf{kb(given,\ prop,\ some,\ betacell,\ produce,\ insulin)}}{\mathsf{kb(deriv,\ prop,\ some,\ insulin,\ produce\_by,\ betacell)}}$$

Notice that relation inversion linguistically corresponds to active-to-passive voice switching as covered in more detail in [12].

The weakening (1) and the dual relationship (2) rule in combination implies:

$$\frac{C\ R\ D}{\mathsf{some}\ D\ R_{inv}\ C} \tag{3}$$

An example of application of this rule is:

$$\frac{\mathsf{kb(given,\ prop,\ every,\ betacell,\ produce,\ insulin)}}{\mathsf{kb(deriv,\ prop,\ some,\ insulin,\ produce\_by,\ betacell)}}$$

Observe that for the copula isa we have inv(isa, isa).

### *4.2. Monotonicity Deduction Rules*

A query task appeals implicitly to appropriate deduction rules. This is because a query normally involves propositions that are deducible from the ones given explicitly in the knowledge base. As an example, given the knowledge base propositions insulin isa hormone and betacell produce insulin, the query

X produce hormone

would intuitively yield X = betacell (with multiple answers to be expected in a more comprehensive knowledge base). This is achieved by means of a pair of logical deduction rules known as monotonicity rules in natural logic [7], the inheritance and the generalization rule, which can be stated as follows:

$$\frac{C_{sub}\ \mathsf{isa}\ C \qquad C\ R\ D}{C_{sub}\ R\ D} \tag{4}$$

$$\frac{C\ R\ D \qquad D\ \mathsf{isa}\ D_{super}}{C\ R\ D_{super}} \tag{5}$$

examples of application of these rules are:

$$\frac{\begin{array}{l}\mathsf{kb(given,\ prop,\ every,\ pancreas,\ isa,\ gland)}\\ \quad \mathsf{kb(given,\ prop,\ every,\ gland,\ produce,\ hormone)}\end{array}}{\mathsf{kb(deriv,\ prop,\ every,\ pancreas,\ produce,\ hormone)}}$$

$$\frac{\begin{array}{l}\mathsf{kb(given,\ prop,\ every,\ betacell,\ produce,\ insulin)}\\ \qquad \mathsf{kb(given,\ prop,\ every,\ insulin,\ isa,\ hormone)}\end{array}}{\mathsf{kb(deriv,\ prop,\ every,\ betacell,\ produce,\ hormone)}}$$

The rules are illustrated in figure 2. The inheritance rule provides inheritance to all sub-concepts of a concept C, while the generalization rule admits generalization

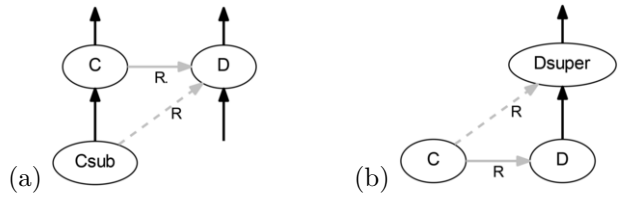


**Figure 2.** Monotonicity rules: (a) inheritance and (b) generalization. Dashed relations are inferred.

of an ascribed property. Notice that the monotonicity rules provide transitivity of isa with the relation R being isa.

The two monotonicity rules above apply to the form every $C$ $R$ $D$. The logic calls for corresponding rules to the form some $C$ $R$ $D$.
Recalling existential import, we get an overlap rule:

$$\frac{\text{every } C \text{ isa } D_1 \qquad \text{some } C \text{ isa } D_2}{\text{some } D_1 \text{ isa } D_2} \tag{6}$$

as well as negative sentences:

$$\frac{\nvdash \text{some } C \ R \ D}{\text{no } C \ R \ D} \tag{7}$$

As a special case we get the non-overlap rule where $R$=isa:

$$\frac{\nvdash \text{some } C \text{ isa } D}{\text{no } C \text{ isa } D} \tag{8}$$

The negative conclusions obtained in rules 7 and 8 by means of negation by non-provability are not added to knowledge base.

### *4.3. Subsumption*

The presence of if-and-only-if definitions in the knowledge base, indicated by the use of definitions for compound terms, calls for the following subsumption rule:

$$\frac{D \text{ isa } D1 \otimes D \ R \ D2 \qquad C \text{ isa } D1 \qquad C \ R \ D2}{C \text{ isa } D} \tag{9}$$

where $\otimes$ indicates the compound term definition as in the decomposition of the form $D$ isa $D1$ that $R$ $D2$. The rule is shown in graph form in figure 3(a). This subsumption rule yields a derived concept inclusion proposition depicted as a dashed arc in figure 3. Due to the if-and-only-if definition of D, it holds that for any concept X such that X isa D1 and X is R related to D2 we must have that X isa D.

An example of application of this rule is :

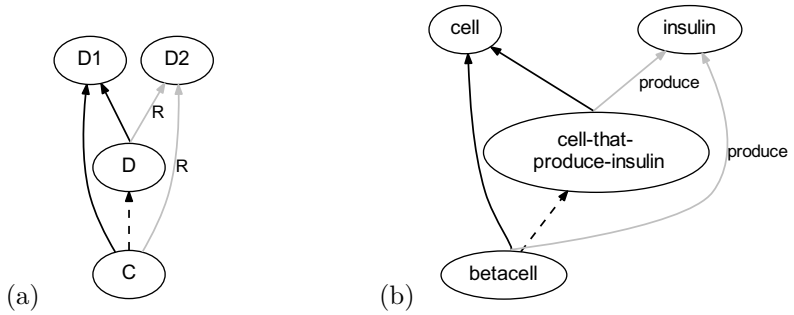


**Figure 3.** The subsumption rule with an example. Dashed relations are inferred.

$$\frac{\begin{array}{r}\mathsf{kb(given,\ defin,\ every,\ cell\text{-}that\text{-}produce\text{-}insulin,\ isa,\ cell)}\\ \mathsf{kb(given,\ defin,\ every,\ cell\text{-}that\text{-}produce\text{-}insulin,\ produce,\ insulin)}\\ \mathsf{kb(given,\ prop,\ every,\ betacell,\ isa,\ cell)}\\ \mathsf{kb(given,\ prop,\ every,\ betacell,\ produce,\ insulin)}\end{array}}{\mathsf{kb(deriv,\ prop,\ every,\ betacell,\ isa,\ cell\text{-}that\text{-}produce\text{-}insulin)}}$$

this example is also shown in graph form in figure 3(b)

### *4.4. Materialization of New Concepts*

All concepts potentially contributing to the answer of a query are to be made explicit in the knowledge base. To achieve this, we now introduce materialization inference rules for integrating new concepts. For instance, from cell and produce and hormone we can construct the constant cell-that-produce-hormone. We distinguish two materialization cases for pairs of definitional arcs and for pairs of non-definitional arcs. The following rule take care of pairs of definitional arcs.

$$\frac{A\ \mathsf{isa}\ B \bigotimes A\ R\ C \qquad C\ \mathsf{isa}\ D}{\langle B\text{-}that\text{-}R\text{-}D\rangle\ \mathsf{isa}\ B \bigotimes \langle B\text{-}that\text{-}R\text{-}D\rangle\ R\ D \qquad A\ \mathsf{isa}\ \langle B\text{-}that\text{-}R\text{-}D\rangle} \tag{10}$$

Where *B-that-R-D* is a new concept positioned as illustrated in figure 4 that also includes a concrete example where the materialized concept is cell-that-produce-hormone. Observe that in the graph rendition pairs of definitional arcs, unlike non-definitional arcs, meet in their starting points.

An example of tuples inferred by materialization for definitional arcs is the following:

$$\frac{\begin{array}{r}\mathsf{kb(given,\ defin,\ every,\ betacell,\ isa,\ cell)}\\ \mathsf{kb(given,\ defin,\ every,\ betacell,\ produce,\ insulin)}\\ \mathsf{kb(given,\ prop,\ every,\ insulin,\ isa,\ hormone)}\end{array}}{\begin{array}{r}\mathsf{kb(deriv,\ defin,\ every,\ cell\text{-}that\text{-}produce\text{-}hormone,\ isa,\ cell)}\\ \mathsf{kb(deriv,\ defin,\ every,\ cell\text{-}that\text{-}produce\text{-}hormone,\ produce,\ hormone)}\\ \mathsf{kb(deriv,\ prop,\ every,\ betacell,\ isa,\ cell\text{-}that\text{-}produce\text{-}hormone)}\end{array}}$$

For the case of non-definitional arcs of concepts we have the following rule.

$$\frac{C\ \mathsf{isa}\ D \qquad C\ R\ E}{\langle D\text{-}that\text{-}R\text{-}E\rangle\ \mathsf{isa}\ D \bigotimes \langle D\text{-}that\text{-}R\text{-}E\rangle\ R\ E \qquad C\ \mathsf{isa}\ \langle D\text{-}that\text{-}R\text{-}E\rangle} \tag{11}$$

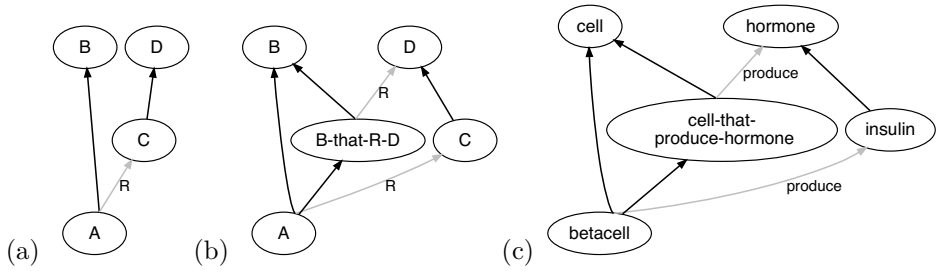


**Figure 4.** Materialization from definitional arcs with example. New concepts B-that-R-D and cell-that-produce-hormone

The rule is shown in graph form in figure 5, that also includes an example. The materialized concepts in the figure are D-that-R-E and cell-that-produce-insulin respectively. The example in figure 5 in inferred database tuple form is the following.

$$\frac{\begin{array}{r}\mathsf{kb(given,\ prop,\ every,\ betacell,\ isa,\ cell)}\\ \mathsf{kb(given,\ prop,\ every,\ betacell,\ produce,\ insulin)}\end{array}}{\begin{array}{l}\mathsf{kb(deriv,\ defin,\ every,\ cell\text{-}that\text{-}produce\text{-}insulin,\ isa,\ cell)}\\ \mathsf{kb(deriv,\ defin,\ every,\ cell\text{-}that\text{-}produce\text{-}insulin,\ produce,\ insulin)}\\ \mathsf{kb(deriv,\ prop,\ every,\ betacell,\ isa,\ cell\text{-}that\text{-}produce\text{-}insulin)}\end{array}}$$

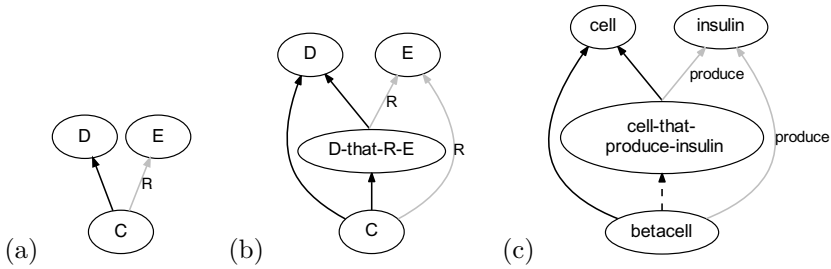


**Figure 5.** Materialization from non-definitional arcs with example. New concepts are D-that-R-E and cell-that- produce-hormone

## 5. Implementation of the Inference Rules by means of Data Base Querying

Execution of queries in a system consisting of a knowledge base and a set of inference rules as presented in the previous section, can be prohibitively ineffective if these rules must be applied while evaluating the queries. Instead, as a preprocess before querying, the knowledge base can be extended with the relevant deductive closure, i.e. assertion of all the expressions that can be inferred from the set of inference rules over that base.
Given the following SQL definition of the database schema to represent the knowledge base kb and the inverse relations inv

```
CREATE TABLE kb(
       mode TEXT,-- 'given' or 'deriv[ed]'
       type TEXT, -- 'prop[osition]' or 'defin[ition]'
       quant TEXT, -- 'every' or 'some'
       sub TEXT, -- subject
       rel TEXT, -- relation
       obj TEXT -- object
);

CREATE TABLE inv(
       rel TEXT, -- relation
       invrel TEXT -- inverse relation
);
```

the deductive closure of the knowledge base kb can be achieved by insertion of all the new tuples that can be inferred from existing tuples using the inference rules, as shown in the previous section.
For the monotonicity rule (4)

$$\frac{C_{sub}\ \mathsf{isa}\ C \qquad C\ R\ D}{C_{sub}\ R\ D}$$

the following SQL statement inserts all tuples inferred from the tuples in kb

```
INSERT INTO kb(
   SELECT 'deriv', 'prop', 'every', kb1.sub, kb2.rel, kb2.obj
   FROM kb AS kb1, kb AS kb2
   WHERE kb1.rel = 'isa'
   AND kb1.obj = kb2.sub);
```

And for the subsumption rule (9)

$$\frac{D\ \mathsf{isa}\ D1 \otimes D\ R\ D2 \qquad C\ \mathsf{isa}\ D1 \qquad C\ R\ D2}{C\ \mathsf{isa}\ D}$$

the following SQL statement inserts all tuples inferred from the tuples in kb excluding the cases where $D$ is equal to $C$ and $D1$ is equal to $D2$.

```
INSERT INTO kb(
   SELECT 'deriv', 'prop', 'every', p1.sub, 'isa', d1.sub
   FROM kb AS d1, kb AS d2, kb AS p1, kb AS p2
   WHERE d1.rel = 'isa'
   AND d1.type = 'defin'
   AND d1.sub = d2.sub
   AND d1.obj <> d2.obj
   AND d2.type = 'defin'
   AND p1.rel = 'isa'
   AND p1.sub = p2.sub
   AND p1.obj = d1.obj
   AND p1.sub <> d1.sub
   AND p2.obj = d2.obj
   AND d2.rel = p2.rel);
```

The remaning inference rules can likewise be expressed as insertion statements. Note, that the insert statements also must take into account not adding tuples that are already present in kb, which is left out here for clarity.
Assuming a set of functions that extends the knowledge base if the inference rules can be applied, for instance, the function `monotonicity_rules()` to apply the two monotonicity rules. An simplistic algorithm applying all the inference rules until all possible inferences are asserted into the knowledge base would look:

```
LOOP
       count <- SELECT COUNT(*) FROM kb;
       EXECUTE inverse_rule();
       EXECUTE monotonicity_rules();
       EXECUTE subsumption_rule();
       EXECUTE materialization_rules();
 WHILE count <> SELECT COUNT(*) FROM kb;
```

Applying this algorithm on the knowledge base in figure 6 will recursively assert all inferred tuples. The first rule to be executed in the algorithm is the inverse rule which add all the inverse edges to the knowledge base as shown in figure 7, where the dotted lines represent the inverted relations in the form some $D$ $R^{-1}$ some $C$ to distinguish from the every form. In the rest of the example figures in this section the inverted edges are left implicit for clarity.

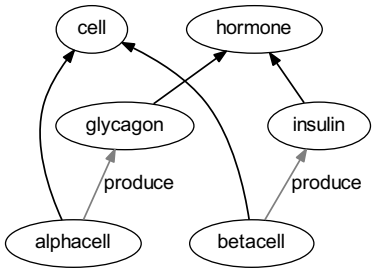


**Figure 6.** Knowledge base with given propositions

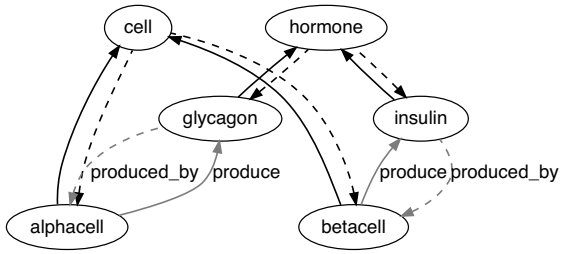


**Figure 7.** Applying the dual relationship rule on the knowledge base from figure 6

Figure 8 shows the result of running one iteration of the algorithm (omitting dual propositions) where the monotonicity rules add the two tuples

kb(given, prop, every, alphacell, produce, hormone)
kb(given, prop, every, betacell, produce, hormone)

and the materialization rules the following tuples

kb(given, prop, every, cell-that-produce-glycagon, produce, glycagon)
kb(given, prop, every, cell-that-produce-glycagon, isa, cell)
kb(given, prop, every, alphacell, isa, cell-that-produce-glycagon)
kb(given, prop, every, cell-that-produce-insulin, produce, insulin)
kb(given, prop, every, cell-that-produce-insulin, isa, cell)
kb(given, prop, every, betacell, isa, cell-that-produce-insulin)

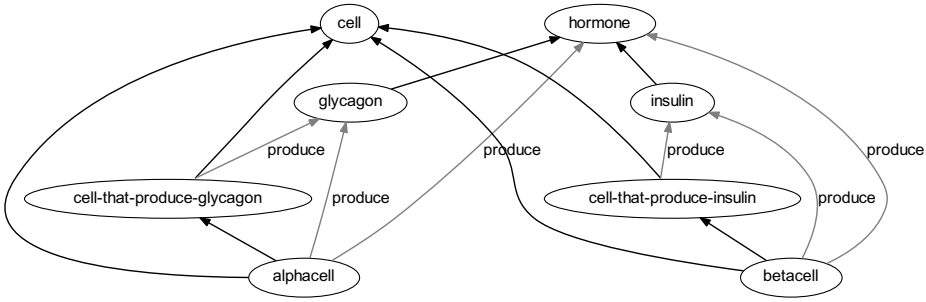


**Figure 8.** The first iteration of the algorithm (dual propositions omitted)

The first iteration adds in total eight new tuples (plus all the inverted edges) to the knowledge base, thus the algorithm will continue with another iteration. In this next iteration the monotonicity rule will add

kb(given, prop, every, cell-that-produce-insulin, produce, hormone)
kb(given, prop, every, cell-that-produce-glycagon, produce, hormone)

and the materialization rules will add one new concept leading to the following tuples

kb(given, prop, every, cell-that-produce-hormone, isa, cell)
kb(given, prop, every, cell-that-produce-hormone, produce, hormone)
kb(given, prop, every, cell-that-produce-insulin, isa, cell-that-produce-hormone)
kb(given, prop, every, cell-that-produce-glycagon, isa, cell-that-produce-hormone)

Here we have shown the first steps of the deductive closure, but the algorithm will continue executing all the inference rules on the knowledge base until no more tuples are added.

## 6. Deductive Querying

Section 4 introduced to the inference rules that apply to a NaturaLog knowledge base and gave examples of how the rules apply to specific tuples. Section 5 exemplified how deductive closure corresponding to the given inference rules can be derived through database updates that iteratively extends the database relation kb by adding new inferred tuples. Below we introduce different types of queries and describe how these can be evaluated by accessing the kb relation.

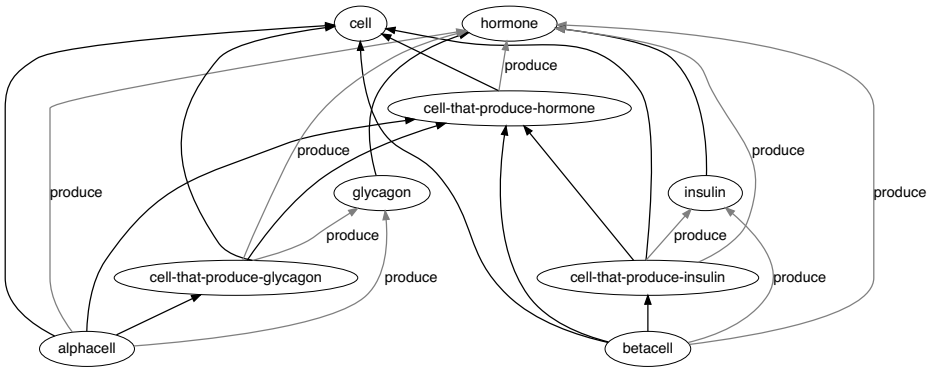


**Figure 9.** The second iteration of the algorithm (dual propositions omitted)

### *6.1. Concept Querying*

A basic query form is an open proposition with one or more free query variables. To pose a query e.g. "what produce insulin" the following parameterized NATURALOG sentence can be used:

X produce insulin

For the knowledge base shown in figure 9 this query would yield {betacell, cell-that-produce-insulin} as possible instantiations for the variable X. The query X produce hormone would lead to the answer {alphacell, cell-that-produce-glycagon, betacell, cell-that-produce-insulin, cell-that-produce-hormone}, and the query betacell produce Y would yield the answer {insulin, hormone}.

Using a variable in the position of the relation provides similarly the possible instantiations. For instance, the query betacell R hormone yields {produce}, while X R hormone leads to the answer {(glycagon, isa), (cell-that-produce-hormone, produce), (cell-that-produce-glycagon, produce), (cell-that-produce-insulin, produce), (insulin, isa), (alphacell, produce), (betacell, produce)}.

Expressions for such concept queries with one or more free variables in SQL are straightforwardly obtained from the proposition form: The first and the last of the queries mentioned above can be expressed in SQL as follows.

```
SELECT sub
 FROM kb
 WHERE rel = 'produce' and obj = 'hormone';

SELECT sub, rel
 FROM kb
 WHERE obj = 'hormone';
```

In the examples above we consider the default proposition form every $C$ $R$ $D$. As noticed, the indicated closures shown in figures 6, 8 and 9 does not include propositions derived from the dual relationship rule that take the form some $D$ $R^{-1}$ $C$. This is to avoid cluttering and maintain readability especially in the two last of these figures. Figure 7, however, includes duals derived from the given propositions in figure 6 and indicates how the knowledge base expands when closuring also with respect to dual relationship. For instance the query X R insulin to the knowledge base in figure 7 would yield, explicating also the quantifier: {(every, betacell, produce), (some, hormone, isa)}.

The SQL expression to retrieve this would simply be:

```
SELECT quant, sub, rel
 FROM kb
 WHERE obj = 'hormone';
```

### *6.2. Commonality Querying*

One of the more sophisticated query forms that NaturaLog affords is commonality querying. The commonality for a pair of stated concepts C and D are the properties they have in common. Considering for instance alphacell and betacell in figure 9 the commonality would be {(produce, hormone), (isa, cell), (isa, cell-that-produce-hormone)}. This can be retrieved by the simple SQL expression:

```
SELECT rel, obj
 FROM kb
 WHERE quant = 'every' AND sub = 'alphacell';
  INTERSECT
 SELECT obj
 FROM kb
 WHERE quant = 'every' AND obj = 'betacell';
```

However, the most interesting contribution to the answer in this case would be the most specific part, that is, {(isa, cell-that-produce-hormone)}. This can also be expressed in SQL in a straightforward manner, such as:

```
SELECT rel, obj
 FROM kb
 WHERE quant = 'every' AND sub = 'alphacell';
  INTERSECT
 SELECT obj
 FROM kb
 WHERE quant = 'every' AND obj = 'betacell'
 GROUP BY obj;
 ORDER BY count(*) DESC LIMIT 1;
```

### *6.3. Pathway Querying*

The entire knowledge base graph forms a road map between all the applied concepts. The introduction of a universal concept at the top of the ontology ensures that all concepts are connected. This concept map can be queried by means of dedicated rules searching pathways in the graph between two stated concepts in the knowledge base. The pathway querying applies the given sentences suplemented with their duals, ignoring other derived propositions. A simple example is given in figure 7.

## 7. Conclusion

In this paper we have described principles and structure for a software system that implements a dialect of natural logic for knowledge bases. This has been done by explaining

how the various inference rules can be realized by relational database query facilities. The derived sentences forming a deductive closure are stored so that querying in the natural logic is conducted as a mere retrieval process. Accordingly, the system is to be implemented as a general-purpose database application. It remains to be seen whether the devised database inference engine principles, is a viable approach when scaling-up knowledge bases.

## References


[1] Troels Andreasen, Henrik Bulskov, Per Anker Jensen, and J. Fischer Nilsson. Computing pathways in bio-models derived from bio-science text sources. In *Proceedings of the IWBBIO International Work-Conference on Bioinformatics and Biomedical Engineering, Granada, April*, pages 217–226, 2014.

[2] J. Fischer Nilsson. In pursuit of natural logics for ontology-structured knowledge bases. In *The Seventh International Conference on Advanced Cognitive Technologies and Applications*, 2015. ISBN 978-1-61208-390-2.

[3] Troels Andreasen, Henrik Bulskov, Per Anker Jensen, and J. Fischer Nilsson. A system for conceptual pathway finding and deductive querying. In *Flexible Query Answering Systems 2015*, pages 461–472. Springer, 2015.

[4] Troels Andreasen, Henrik Bulskov, Per Anker Jensen, and J. Fischer Nilsson. *Partiality, Underspecification, and Natural Language Processing*, chapter A Natural Logic for Natural-Language Knowledge Bases. Cambridge Scholars, 2017.

[5] Troels Andreasen, Henrik Bulskov, Per Anker Jensen, and J. Fischer Nilsson. Pathway computation in models derived from bio-science text sources. In Marzena Kryszkiewicz, Annalisa Appice, Dominik Slezak, Henryk Rybinski, Andrzej Skowron, and Zbigniew W. Ras, editors, *Foundations of Intelligent Systems*, pages 424–434, Cham, 2017. Springer International Publishing. ISBN 978-3-319-60438-1.

[6] Troels Andreasen, Henrik Bulskov, Per Anker Jensen, and J. Fischer Nilsson. Deductive querying of natural logic bases. In Alfredo Cuzzocrea, Sergio Greco, Henrik Legind Larsen, Domenico Saccà, Troels Andreasen, and Henning Christiansen, editors, *Flexible Query Answering Systems*, pages 231–241, Cham, 2019. Springer International Publishing. ISBN 978-3-030-27629-4.

[7] Johan van Benthem. *Essays in Logical Semantics, Volume 29 of Studies in Linguistics and Philosophy*. D. Reidel, Dordrecht, Holland, 1986.

[8] J van Benthem. Natural logic, past and future. In *Workshop on Natural Logic, Proof Theory, and Computational Semantics*, 2011.

[9] Lawrence S. Moss. Syllogistic logics with verbs. *J. Log. Comput.*, 20(4):947–967, 2010.

[10] Ian Pratt-Hartmann and Lawrence S. Moss. Logics for the relational syllogistic. *Review of Symbolic Logic*, 2(4):647–683, 2009.

[11] Gyula Klima. Natural logic, medieval logic and formal semantics. *MAGYAR FILOZÃŞFIAI SZEMLE*, 54 (4):58–75, 2010.

[12] Troels Andreasen, Henrik Bulskov, and Jørgen Fischer Nilsson. Natural logic knowledge bases and their graph form. *Data & Knowledge Engineering*, August 2020. ISSN 0169-023X. doi: https://doi.org/10.1016/j.datak.2020.101848.

[13] Troels Andreasen and J. Fischer Nilsson. Grammatical specification of domain ontologies. *Data Knowl. Eng.*, 48(2):221–230, 2004. URL `http://dblp.uni-trier.de/db/journals/dke/dke48.html`.