

Generic Fuzzy Reasoning Nets as a Basis for Reverse Engineering Relational Database Applications

Jens H. Jahnke, Wilhelm Schäfer, Albert Zündorf

AG-Softwaretechnik, Fachbereich 17, Universität Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany;
e-mail: [jahnke|wilhelm|zuendorf]@uni-paderborn.de

WWW: http://www.uni-paderborn.de/fachbereich/AG/schaefer/index_engl.html

Abstract. Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing (CIM), or software engineering (SE). However, a competitive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. One approach for seamless integration of object-oriented and relational systems is to migrate from a relational to an object-oriented system. The first step in this migration process is reverse engineering of the legacy database. In this paper we propose a new graphical and executable language called *Generic Fuzzy Reasoning Nets* for modelling and applying reverse engineering knowledge. In particular, this language enables to define and analyse fuzzy knowledge which is usually all what is available when an existing database schema has to be reverse engineered into an object-oriented one. The analysis process is based on executing a fuzzy petri net which is parameterized with the fuzzy knowledge about a concrete database application.

1 Introduction and Related Work

Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing (CIM), or software engineering (SE). Those new requirements are not fulfilled by relational technology [LS88, Mai89]. However, a competitive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. In addition, complex dependencies between those data stored in the different systems might exist and should be maintained. One approach for seamless integration of object-oriented and relational systems is to migrate the data (and the corresponding schema) from a relational to an object-oriented system.

The whole migration process consists of three steps which are (1) the *schema migration process* which maps a relational schema to an equivalent object oriented schema, (2)

the *data migration process* which converts extensions of the relational schema to extensions of the object oriented schema and (3) the *application migration process* which creates a new application program using the object oriented database for every application program that uses the legacy database.

This paper focuses on the first step in the migration process, namely the migration of the schema. A major job to do when migrating a schema is to analyse and reverse engineer the schema. This paper proposes a new approach for analysing existing relational schemas in order to reverse engineer them into object-oriented ones.

Analysing an existing database schema means to analyse the schema definition, the application code written in a so-called embedded SQL (Standard Query Language)-language where the embedding language is e.g. C or COBOL, and the available extensions of the schema. This analysis requires to express possibly uncertain assumptions about the schema which is to be reverse engineered. These assumptions may be partly deducible from the schema definitions and in particular the defined integrity constraints or from the SQL-queries within the application code. They may also only be deduced from the currently available schema extensions. Furthermore, an application has been developed incrementally over time by many developers who are sometimes even not accessible any more and who often did not do a good job on documentation. The result is that integrity constraints may even define contradicting constraints.

In order to retrieve an object-oriented schema an analysis tool should help to combine basic assumptions and to investigate their consequences. As many of those assumptions are uncertain, a tool should be able to deduce different possible consequences and their confidences. By supporting an incremental and interactive verification process, the tool should then let the reverse engineer decide finally which is the best alternative to choose.

Existing approaches [DA87,JK90,SK90,And94,PB94,PKBT94,FV95] do not support this kind of reasoning process, since the knowledge about the reverse engineering process is usually not defined explicitly but hard-coded in a batch-oriented analysis tool. One notable exception is an approach which is based on defining the reverse engineering knowledge in terms of PROLOG-rules [SLGC94]. However this approach as well as all others do not support the explicit (and thus easily changeable) definition and analysis of uncertain knowledge.

The next section gives a more detailed example which kind of semantic information we have to derive and which kind of uncertainty we have to deal with. Section 3 then presents our approach how to capture formally and precisely yet intuitively the (uncertain) reverse engineering knowledge about relational databases. Section 4 describes the inference engine which analyses basic facts about the investigated database and which then infers as much semantic information as possible by applying the reverse engineering knowledge described within Section 3. Within this step the inference engine incorporates and verifies interactively added user knowledge. Section 5 concludes with describing the current state of our work.

2 A Motivating Scenario

Due to its simplicity the relational model lacks the expressive power to explicitly represent high level modelling concepts like, e.g. object relationships, aggregation, and inheritance. However, indicators for these concepts are spread all over the application code, the schema and the extension of a legacy relational database. The focus of reverse engineering is to find such indicators in order to recover information about high level modelling concepts. In [FV95] this process is described as *semantic completion* of a relational schema. Consequently, the recovered information which is the result of this process is denoted as the *semantic information* of a relational schema. This semantic information is the basis for a translation of the relational schema into an object-oriented data model, as described in [FV95].

As an illustrating example consider the cutout of a relational database system represented by its schema (Fig. 1), its extension (Fig. 2) and its application code (Fig. 3). The reverse engineered object model for this example is given in OMT-like notation in Fig. 5. In order to be able to produce this object model the semantic information depicted in Fig. 4 has to be deduced from the legacy database. In the following we will explain how this information is retrieved from the sample database.

First we deduce that there may be two different kinds (*variants*) of persons, due to the fact that every person in the sample extension has either a null-value in the AFR column or in the dep column, but there is no person that has both columns with null value. Due to our domain knowledge about the analysed database application, we name the variant with values in the AFR column `Developer`, while the other variant with `dep` values is denoted as `Manager`.

Attribute `dep` is probably a key of variant `Manager`, as it is declared as an index and it is not-null in this variant and the functional dependencies to all other attributes in `Developer` hold in the extension.

```
create table Project (  
  descr varchar(50) not null,  
  manager varchar(50),  
  size numeric,  
  scheduled datetime)  
create table Person(  
  name varchar(80),  
  email varchar(80),  
  dep varchar(80),  
  AFR real)  
create index Person(dep)  
create table WP(  
  descr varchar(50) not null,  
  proj varchar(50)  
  dev varchar(50),  
  size numeric,  
  scheduled datetime,  
  foreign key proj references Project(descr))
```

Fig. 1: Sample database schema

Furthermore, the first statement in Fig. 3 selects two persons with different name attributes. In [And94] this is called a cyclic exclusion, which serves as an indicator that name is a key of table `Person`. On the other hand, the second select statement, which gets the AFR value of some entry in table `Person` with a specific value for name, includes the keyword *distinct*, which is a negative indicator for name being a key

[And94]. Due to our experience, we know that some programmers tend to use the key-word distinct by default even if not necessary. Thus, we resolve the above conflict by assigning a higher weight to the pro argument than to the counter argument.

Person	name	email	dep	AFR	
	Steve	snoopy	NULL	1.2	
	Brenda	brenda	MW	NULL	
	Martin	mksoft	NULL	1.9	
	Boris	bumbum	NULL	1.4	
	Michael	mike	IS	NULL	
...					
Project	descr	manager	size	scheduled	
	Varlet	Michael	150	05/31/97	
	Merlin	Brenda	320	06/01/98	
	GEN	Michael	80	02/01/97	
...					
WP	descr	project	dev	size	scheduled
	Trafo	Varlet	Martin	30	02/01/97
	PE	Merlin	Steve	20	04/01/97
	GUI	GEN	Boris	10	01/01/97
	Ana	Varlet	Martin	12	03/01/97...

Fig. 2: Database extension

```

select * from Person x,y
where (x.AFR>$A) and (y.AFR>$A)
and not (x.name=y.name)
...
select distinct AFR from Person
where name=$N
...
select email from Person,WP
where scheduled<$E and dev=name
...

```

Fig. 3: Application code

From the observation that tables `Project` and `WP` both have attributes `descr`, `size` and `scheduled` with matching types, it is likely that at-a-time these attributes have identical meaning. According to [FV95] attributes with identical meanings are collected in so-called

equivalence classes. Normally, this indicates an inclusion dependency (IND) between both sets of attributes. However, this assumption can be disproved by analysing the extension in Fig. 2. In fact, there is no tuple in table `Project` and `WP` with common values in these columns. Therefore, we assume that there is a hidden *domain relation* `Task` for all tuples in `Project` and `WP`. At this, the term domain relation is used as the relational analogy for a superclass in the object oriented data model (cf. [FV95]). The deduced inheritance relationship is characterized by inclusion dependencies (ISA-IND) from `Project` and `WP` to the new domain relation.

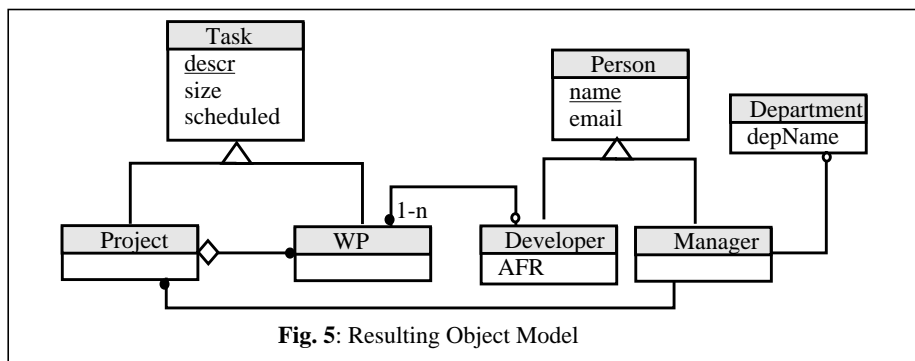
Moreover the third statement of Fig. 3 is a strong indicator that attribute `dev` of table `WP` belongs to the same equivalence class as attribute `name` of table `person`, because the statement includes a join of both tables over these attributes. Thus, we conclude that there exists an IND in either direction between these attributes. However, we already know that there are two different variants of `person`, namely `Developer` and `Manager`. Thus, we have to check for both variants, whether the possible INDs can be disproved by a counterexample in the extension. In fact, the assumption of an IND between variant `Manager` and `WP` is refuted by the given extension, while INDs in both directions between variant `Developer` and `WP` are fulfilled.

According to [FV95], we classify the IND going from table `WP` to table `Developer` as a *key-based* IND, as we assumed that the attributes on its right side built a key in table `Developer`. Such an IND represents a referential integrity constraint and is called an R-IND in [FV95]. Consequently, the supposed inverse IND from `Developer` to `WP` is denoted *inversely key-based* but not key-based. Such INDs are classified as so-called

C-INDs, as they reveal a cardinality constraint for the regarded relationship between WP and Developer, i.e. there is at least one tuple in WP for each tuple in Developer. Together, these two inverse INDs are translated into an association between WP and Developer with cardinality 1-n to 0-1 in the object-oriented model (Figure 5).

<p>Variants: Person: Developer(name,email,AFR) Manager(name,email,dep)</p> <p>Keys: project(descr) WP(descr) Person(name) Manager(dep)</p> <p>Not Null: Project(manager,descr)</p> <p>Equivalence Classes: { WP.descr,Project.descr } { WP.size,Project.size } { WP.scheduled,Project.scheduled } { WP.dev,Developer.name } { WP.proj,Project.descr } { Project.manager,Manager.name }</p>	<p>Domain Relations: Task(descr,size,scheduled)</p> <p>Inclusion Dependencies: R-IND: Project(manager) \subset Manager(name) WP(proj) \subset Project(descr) WP(dev) \subset Developer(name) C-IND: Developer(name) \subset WP(dev) ISA-IND: Project(descr,size,scheduled) \subset Task(descr,size,scheduled) WP(descr,size,scheduled) \subset Task(descr,size,scheduled)</p>
---	---

Fig. 4: Semantic information for the given database schema



The reader should note that due to the lack of space we do not present all parts of the sample database that is investigated in order to derive the depicted semantic information. In addition some information like e.g. the names for the variants Developer and Manager cannot be deduced automatically, but has to be added manually.

3 Generic Fuzzy Reasoning Nets

In order to make reverse engineering knowledge about relational databases accessible for a (semi-)automatic analysis process, it has to be specified in a formal representation. Moreover, it is of great importance that this knowledge is easily adaptable. For example, different legacy database applications may provide different sources of information which has to be considered in the reverse engineering process. Furthermore, the credi-

bility of indicators may differ from one to another developer/company depending on the preferred style of programming and the history of the legacy database application.

Coding the reverse engineering knowledge in a programming language like C or C++ is not feasible, since this approach lacks the desired flexibility. Even if this knowledge is modelled in textual rules (e.g. in PROLOG) extensive specifications might be difficult to understand. Thus, because of its expressive power we developed a graphical formalism, so called *Generic Fuzzy Reasoning Nets* (GFRN) which are described in this section.

Let us consider the following reverse engineering rule R:

if the application code includes a select statement with a *distinct* keyword qualified by a set of attributes *a* **then** it can be deduced with certainty ϕ that *a* and all its subsets do not represent candidate keys.

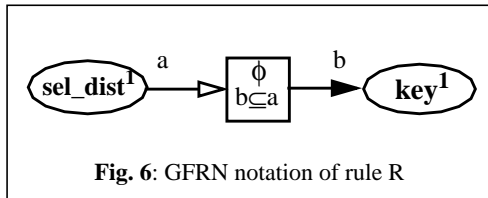


Fig. 6: GFRN notation of rule R

The GFRN notation of our sample rule R is shown in Fig. 6. There are two predicates, sel_dist^1 and key^1 , which are represented by ovals and an implication, which has a rectangular shape. Predicates are labelled with unique names which are indexed by

their arity. Predicates and implications are connected by directed edges. An outgoing edge from a predicate to an implication denotes that this predicate is in the antecedent of the implication, while an ingoing edge from an implication to a predicate shows that the predicate is in the implication's consequent. Negations are represented by black arrow heads. Each edge is labelled with a (list of) formal parameter(s), which serves as argument for the participating predicate. Each implication has an associated confidence factor (ϕ). Furthermore, each implication has a set of constraints over the formal parameters of its in- and outgoing edges. In Fig. 6 the implication has only one constraint which specifies parameter *b* to be a subset of parameter *a*.

The semantic of a GFRN is formally defined by a translation into weighted formulas in first-order logic: Each predicate is translated in a corresponding predicate symbol. Each implication is translated into a pair $\mathcal{F}^{\phi} := (\mathcal{F}, \phi)$, where ϕ is a valuation (usually between 0 and 1) and \mathcal{F} is a closed formula in classical first-order logic [CS80]. The reader should note that in the GFRN language all formal parameters of implications are implicitly quantified with a universal quantifier. For example the implication in Fig. 6 is translated in the following logic formula \mathcal{F} :

$$\mathcal{F}: (\forall a)(\forall b \subseteq a)(sel_dist(a) \Rightarrow \neg key(b))$$

In different approaches to uncertain logic the valuation ϕ can have different semantic. In probabilistic logic [Paa88] ϕ defines a probability measure for the likelihood that \mathcal{F} is fulfilled, while in possibilistic logic [DP88] which is based on the theory of fuzzy sets [Zad78] ϕ represents an upper bound of possibility that \mathcal{F} is true (respectively a lower bound of necessity in necessity valued possibilistic logic [DLP94]).

We choose a possibilistic semantic for ϕ because in our application domain valuations are intuitively chosen by the reverse engineer based on heuristics and assumptions rather than on statistical analysis of precise likelihood measures. Furthermore computation of probabilistic measures is much more expendable, in order to fulfil all axioms of probability theory and as argued we do not need this degree of precision. For a detailed discussion on the pros and cons of probabilistic and possibilistic logic we refer to [DP88].

The knowledge that we used in our reverse engineering scenario (Section 2) is easily modelled in a GFRN as shown in Fig. 7. We already know predicates sel_dist^1 and key^1 from Fig. 6. The fact that a cyclic exclusion might be an indicator for a key constraint is specified by an additional predicate $cycl_excl^1$ and implication i_2 .

According to our experience with legacy database applications we have chosen the confidence of implication i_2 as quite high (0.8) while the confidence of implication i_1 is relatively low (0.3). This is due to our observation that a cyclic exclusion serves as a credible indicator for a key constraint, while the keyword `distinct` is often used in application code when it is not really needed.

Implication i_{10} specifies with confidence 1 that a supposed key constraint may only exist, if it cannot be disproved by a counterexample in the database extension.

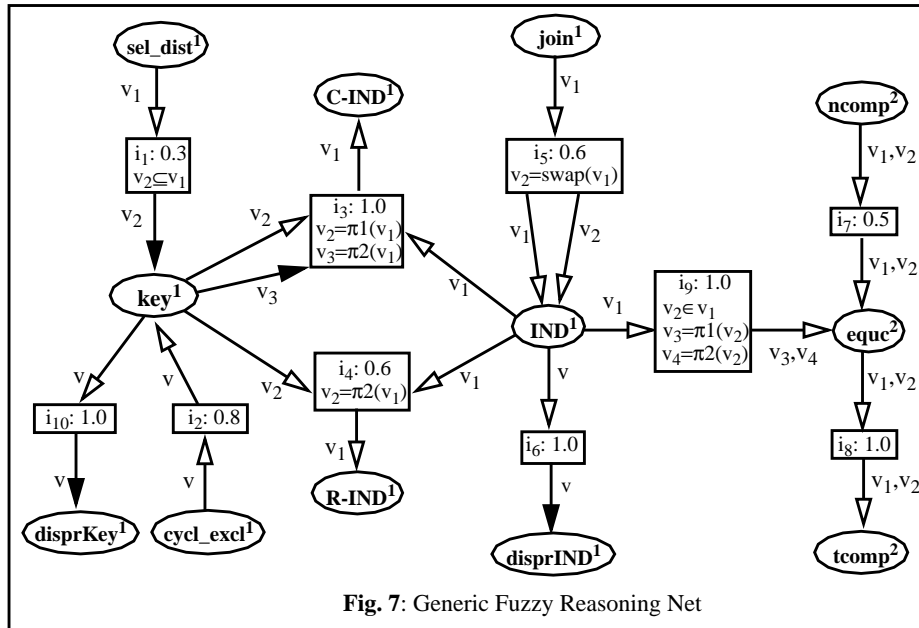


Fig. 7: Generic Fuzzy Reasoning Net

Furthermore, the GFRN includes a predicate $join^1$ which reflects the fact that we are interested in finding joins over attributes of different tables in the application code of the legacy relational database. The argument of predicate $join^1$ is a set of tuples, each tuple representing a pair of compared attributes of both tables. As argued in our reverse engineering scenario such a join statement may reveal an inclusion dependency in one

or the other direction between the participating attributes. This is reflected via implication i_5 which has two outgoing edges to predicate IND^1 . The right edge is labelled with parameter v_2 which is constrained to be the application of function swap on parameter v_1 . Function swap is defined to take a set of pairs and to exchange the element positions of each pair. Consequently, for a given join statement i_5 specifies the conclusion of an IND in both possible directions with a certainty factor of 0.6.

A supposed IND may only reside if it cannot be disproved by an analysis of the extension of the legacy database. This is specified by implication i_6 . Moreover, i_9 strictly implies that every pair of attributes in an IND belongs to the same equivalence class. At this, functions π_1 and π_2 stand for the relational projection on the first and the second element in each tuple.

Another heuristical indicator for a possible equivalence of two attributes is the compatibility of their names (ncomp^2). Furthermore, it is obvious that equivalent attributes must have compatible types, which is specified through tcomp^2 and i_8 . The reader should note that due to the application of fuzzy sets in our approach the notion of type and name compatibility is not as strictly defined as in most other approaches.

Finally, implications i_3 and i_4 serve to classify an IND as either an R-IND, if there is a key constraint for the attributes on the right of the inclusion, or a C-IND if this is not the case and the left hand of the inclusion dependency is assumed to be a key.³

Formally a GFRN is defined by the following 5-tuple.

Definition 1: Generic Fuzzy Reasoning Net

A generic fuzzy reasoning net is defined by a 5-tuple $GFRN := (P, I, E, F, cf)$, where

- $P = \{p_1^{k1}, p_2^{k2}, \dots, p_z^{kz}\}$ is a finite set of unique *predicate symbols*, where $kq \in \mathbb{N}$ denotes the arity of predicate symbol p_q^{kq} .
- $I = \{i_1, i_2, \dots, i_m\}$ is a finite set of *implications*, each implication $i_g \in I$ is a tuple $i_g = (i_g, V_g, K_g)$, with
 - i_g , a unique *implication identifier*,
 - $V_g = \langle v_1, v_2, \dots, v_f \rangle, f \in \mathbb{N}^+$, a list of unique *parameter names* of implication i_g ,
 - $K_g = \{k_1, k_2, \dots, k_s\}, s \in \mathbb{N}$ is a finite set of *constraints* over V_g , with $k_x = (w, r, f^t, \langle w_1, w_2, \dots, w_u \rangle)$ where $w, w_1, \dots, w_u \in V_g, r \in \{\in, \subseteq\}$ and $f^t \in F$.
- $E = \{e_1, e_2, \dots, e_n\}, n \in \mathbb{N}^+$ is a finite set of *edges*, where each $e_g \in E$ is a tuple $e_g = (\chi_g, l_g, s_g, d_g, A_g)$, with
 - χ_g an unique *identifier*,

3. The confidence of implication i_4 has been chosen as 0.6 because according to [FV95] an IND may also represent an inheritance relationship (classified as an ISA-IND), if there exists key constraints for both sides of the IND.

- $l_g: (p_q^{kq}, (t, V, K)) \in (P \times I)$, a *location*,
 - $s_g \in \{+, -\}$, a *sign*,
 - $d_g \in \{\textit{antecedent}, \textit{consequent}\}$, a *direction*, and
 - $A_g = \langle \alpha_1, \alpha_2, \dots, \alpha_{kq} \rangle$, an *actualization vector* of e_g where $\alpha_j = \varepsilon$ or $\alpha_j \in V$ for $1 \leq j \leq kq$.
- $F = \{f_1^{u_1}, f_2^{u_2}, \dots, f_x^{u_x}\}$ is a finite set of unique *function symbols*, where $u_q \in \mathbb{N}$ denotes the arity of function symbol $f_q^{u_q}$.
 - $cf: I \rightarrow (0, 1]$ is a function that associates real values between 0 and 1 to implications.

4 The Fuzzy Inference Engine

A GFRN solves the problem to represent general reverse engineering knowledge formally but yet intuitively accessible. In order to analyse existing applications, this knowledge has to be combined with concrete information about the application's schema, the available extensions, and the available procedural code (cf. the example in Section 2). An inference engine uses the knowledge expressed in a GFRN together with concrete information about an existing application to infer all possible pieces of semantic information about the application (e.g. Fig. 4) together with their confidences. The confidence information is then used by the reverse engineer to decide which are the pieces of semantic information that are used to construct the object-oriented schema.

4.1 Introducing Incomplete Information to the Inference Process

The inference process starts with the defined GFRN and a usually incomplete amount of information about a concrete application. This information is incomplete because it is not feasible to retrieve all facts about the application which may be relevant for deduction *before* the inference process is started. For example it is not practical to determine in advance the set of all possible inter- and intrarelatational dependencies that can or cannot be disproved via the extension of a relational database application, since the number of resulting facts grows exponentially with the size of the analysed schema. The only way out is to start inferencing with incomplete information which may be completed depending on the intermediate results during the inference process (as we will illustrate later).

In our approach we will refer to the initial amount of incomplete information as the *premise*, which consists of a set of facts which we call *axioms*. We distinguish three types of axioms which are handled differently during the inference process.

The first type of axioms are so-called *strong* axioms, since they cannot be disproved during the inference process. Typically, strong axioms are automatically retrieved during an initial investigation of the concrete legacy database application. Revisiting the motivating scenario in Section 2 the information that a select-distinct statement and a cyclic join has been found in the application code will be represented in our approach as two strong axioms with confidence 1.

Another type of axioms are called *weak* axioms. Weak axioms stem from the subjective belief of a reengineer that certain facts may be true with a given confidence. As opposed to strong axioms the confidence of weak axioms may be changed during the inference process. For example let us assume that the reengineer in our sample scenario interactively adds the assumption that attribute `dev` is a key of table `WP` with his/her subjective confidence of 0.5. Given the available extension in Fig. 2 this assumption has to be refuted during the inference process, as the values of attribute `dev` are not unique for each tuple in table `WP`.

Finally, a premise may include a number of *deferred axioms* which are evaluated on an on-demand basis during the inference process. For example, if we consider the GFRN in Fig. 7 it is obvious that axioms over predicates `disprIND`¹ and `tcomp`² should be defined as deferred, because otherwise, as argued above, there would be an enormous number of strong axioms that would have to be created.

Formally, a premise is defined as follows.

Definition 2: Proposition

Let (P, I, E, F, cf) be a GFRN and U the set of constants in our universe of discourse.

A *proposition* d is a tuple $d := (p_q^{kq}, O)$, where

- p_q^{kq} is a predicate symbol,
- $O := \langle o_1, o_2, \dots, o_{kq} \rangle$, is a list of *constants*, $o_i \in U$; $1 \leq i \leq kq$.

Any proposition $d := (p_q^{kq}, \langle o_1, \dots, o_{kq} \rangle)$ is said to be in the extend of P ($d \in \text{ext}(P)$), iff $p_q^{kq} \in P$.

Definition 3: Premise

A *premise* Z_G over a given GFRN $G := (P, I, E, F, cf)$ is defined as a tuple $Z_G := (A_G^s, A_G^w, A_G^d)$:

- $A_G^s := \{a_1^s, a_2^s, \dots, a_n^s\}$, $n \in \mathbb{N}^+$ is a finite set of *strong axioms*, while $A_G^w := \{a_1^w, a_2^w, \dots, a_m^w\}$, $m \in \mathbb{N}^+$ is a finite set of *weak axioms*, where each $a \in A_G^s \cup A_G^w$ is a tuple $a := (d, s, \phi)$
 - $d \in \text{ext}(P)$ is a *proposition*,
 - $s \in \{+, -\}$ is the *sign* of a , and
 - $\phi \in (0, 1]$ is a *confidence*.
- $A_G^d := \{a_1^d, a_2^d, \dots, a_m^d\}$, $m \in \mathbb{N}^+$ is a finite set of *deferred axioms*, where each $a \in A_G^d$ is a tuple $a := (p, s, \phi)$
 - $p \in P$ is a *predicate*,
 - $s \in \{+, -\}$ is the *sign* of a , and
 - $\phi: \text{ext}(P) \rightarrow (0, 1]$ is a *confidence function*.

4.2 Fuzzy Petri Nets

The requirement to deal with incomplete and inconsistent information demands an inference mechanism that allows for nonmonotonic reasoning. We employ fuzzy petri nets (FPNs) [Loo88,KM96] to execute GFRN specifications as they fulfil the above requirement. A further advantage of this approach is that petri nets are an ideal choice for implementing the proposed inference engine, because they can be evaluated efficiently due to their high degree of structural parallelism and pipelining [KM96].

Generally, a petri net [Pet81] is a directed bipartite graph with active and passive elements. In terms of terminology we follow [KM96] and call active elements *transitions*, while passive elements are referred to as *places*.

Definition 4: Fuzzy Petri Net

Given a finite set of propositions D a *fuzzy petri net* (FPN) is defined by a tuple $FPN := (Pl, Tr, C, D, c, th, m, b)$, where

- $(Pl, Tr; C)$ is a net without isolated elements, with
 - $Pl := \{pl_1, pl_2, \dots, pl_{2n}\}$, $n \in \mathbb{N}^+$, is a finite set of *places*,
 - $Tr := \{tr_1, tr_2, \dots, tr_x\}$, $x \in \mathbb{N}^+$, is a finite set of *transitions*,
 - $C \subseteq (Pl \times Tr) \cup (Tr \times Pl)$ a *flow relation*.
- $m: Pl \rightarrow [0, 1]$ is a function that associates real values between 0 and 1 to places.
- $b: \{+, -\} \times D \rightarrow Pl$ is a bijective function that associates signed propositions to places.
- $c, th: Tr \rightarrow [0, 1]$ are functions that associate real values between 0 and 1 to transitions.

According to Definition 4 each place pl in an FPN is marked by a real value ($m(pl)$) between 0 and 1. Each proposition is associated with two places in an FPN: The marking of one place represents the fuzzy belief that the proposition is true, and the marking of the other place represents the fuzzy belief that its negation is true. Furthermore, each transition tr has a confidence ($c(tr)$) and a *threshold* ($th(tr)$).

As an example, Fig. 8 shows a transition with k input places. At a certain point in time t we denote the fuzzy beliefs associated to places pl_1 to pl_k with $m_t(pl_1)$ to $m_t(pl_k)$, respectively. In order to determine, whether transition tr_q is enabled the fuzzy AND-operator (which is defined as the minimum function) is applied on the fuzzy beliefs of all its input places. If the result of this operation is greater than the threshold $th(tr_q)$ then tr_q is said to be enabled. In this case tr_q generates a *fuzzy truth token* $ftt_{t+1}(tr_q)$ for its output arcs which computes to the result of the above AND-operation concatenated with the transitions confidence $c(tr_q)$ via another AND-operation. Otherwise $ftt_{t+1}(tr_q)$ is defined to be zero.

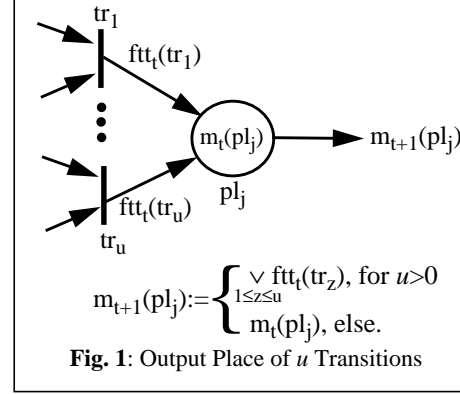
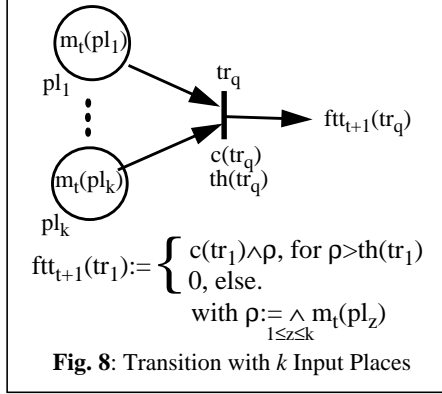


Fig. 9 shows that given a place pl_j with fuzzy belief $m_t(pl_j)$ in the output of u transitions, the new fuzzy belief $m_{t+1}(pl_j)$ is computed by applying the fuzzy OR-operator (which is defined as the maximum function) over the set of fuzzy truth tokens generated by transitions tr_1 to tr_u . In case there is no transition with pl_j in its output ($u=0$), the fuzzy belief for pl_j remains unaltered.

A major difference of the employed FPN formalism in contrast to classical petri nets is that when an enabled transition fires, tokens are not removed from its input places. On the contrary, input tokens are only copied and remain at their original places. This procedure is necessary for logic inference, since the truth of a proposition may imply the truth of several other conditions. Because of this speciality well-known structural conflicts like deadlocks and traps [Pet81] can not occur in an FPN.

In [Loo88], Looney presents an algorithm for belief evaluation in acyclic fuzzy petri nets. This algorithm is extended in [KM96] for FPNs with arbitrary topology. At this, evaluation of fuzzy beliefs is carried out by a number of subsequent belief revision steps. In each belief revision step the fuzzy beliefs at all places in the FPN are updated in parallel according to Fig.8 and Fig. 9. The evaluation process terminates when the FPN has reached *steady-state*. This is the case when the most recently performed belief revision step did not cause a change of any fuzzy belief in the entire FPN.

However, some FPN may exhibit sets of places that show periodically oscillation of their fuzzy beliefs over an infinite number of belief revision steps. In this case it is said that *limitcycle* exists at each of these places. The evaluation algorithm presented in [KM96], which is also utilized in our approach, allows for a detection and elimination of limitcycles.

4.3 Expansion and Evaluation of a FPN based on a GFRN

In the following we will explain how a GFRN with a given premise can be executed by expanding and evaluating a fuzzy petri net. Due to the lack of space we will only give an informal description of the inference algorithm. A formal specification of this algorithm using graph rewriting systems is described in [JSZ97].

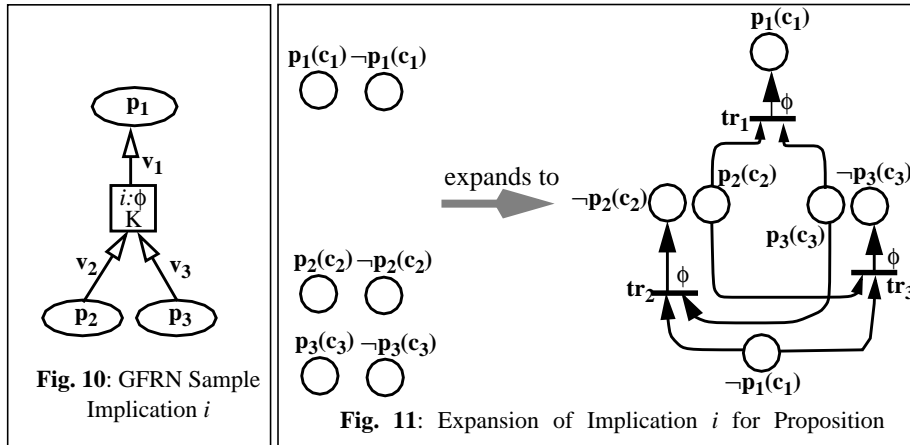
The inference algorithm has two main phases. In the first phase (expansion), an FPN is created according to the specified GFRN and a given premise. In the second phase (evaluation), fuzzy beliefs are computed until steady-state has been reached.

The expansion phase is performed within three steps. In the first step, places for all relevant propositions are created in the FPN. Each proposition in the FPN represents an instance of a certain predicate in the corresponding GFRN with constants as parameters. As defined above, a proposition d is represented by two places: One place $b(+,d)$ carries the fuzzy belief that d is true (we call it the *positive place of d*), while the other place $b(-,d)$ carries the fuzzy belief that d is false (we call it the *negative place of d*).

The first propositions that are created in the FPN are defined by the set of strong and weak axioms in the given premise. Then, iteratively every implication with one or more existing propositions in its antecedent and/or consequent is checked, whether all its formal parameters are completely determined by the actual parameters of these propositions. If this is the case, then all propositions that have not yet been created for this implication are created. Whenever a new proposition has to be created, which is defined by a deferred axiom, the fuzzy belief of this proposition is computed by its confidence function. This expansion step is complete when no further proposition can be created.

In the second expansion step, the FPN is completed by creating transitions between positive and negative places of propositions. The instantiation of an implication in a GFRN consists of at least two transitions in the FPN. This is because every implication also implies a contraposition, i.e. from $a \Rightarrow b$ we can conclude $\neg b \Rightarrow \neg a$. In our approach, the contraposition is considered in the inference process, which allows to refute already made assumptions when certain other facts they imply prove to be false.

Let us consider the sample GFRN implication i in Fig. 10 which has two predicates in its antecedent (p_2, p_3) and one predicate in its consequent (p_1).



The left side of Fig. 11 shows the three pre-existing propositions ($p_1(c_1)$, $p_2(c_2)$) and $p_3(c_3)$. If the formal parameters v_1, \dots, v_3 of implication i are bound to constants c_1, \dots, c_3 such that the constraints K hold, the right side of Fig. 11 shows how implication i is expanded into three transitions with confidence ϕ^4 .

At this, transition tr_1 represents the rule $p_2(c_2) \wedge p_3(c_3) \rightarrow p_1(c_1)$ while the contraposition $\neg p_1(c_1) \rightarrow \neg(p_2(c_2) \wedge p_3(c_3))$ is normalized to $\neg p_1(c_1) \wedge p_3(c_3) \rightarrow \neg p_2(c_2)$, represented by transition tr_2 and $\neg p_1(c_1) \wedge p_2(c_2) \rightarrow \neg p_3(c_3)$ represented by transitions tr_3 .

Generally, the number of transitions that are created for the contraposition is equal to the number of propositions in the antecedent of the expanded implication.

Finally, the third expansion step removes all ingoing edges from propositions that represent strong or deferred axioms from the FPN. This is done in order to fulfil the requirement of Section 4.1 that during the evaluation phase, the beliefs of strong and deferred axioms must not be changed.

The evaluation phase of the inference process starts when the FPN has been completely expanded. In this phase, fuzzy beliefs are evaluated at each place in the FPN (according to Fig. 8 and Fig. 9) until steady state has been reached.

The described inference algorithm is given below.

algorithm inference

begin

Expansion Phase:

For each (strong and weak) axiom create a new proposition;

Repeat

If there exists a set of propositions $d_1..d_n$ with $d_i := (p_i^{ki}, O_i)$

and there exists an implication $i: (t, V, K)$ in the GFRN with $p_1^{kl} .. p_n^{kn}$ in its antecedent or consequent

and all formal parameters $v \in V$ are determined by $d_1..d_n$

then expand all propositions in the antecedent and consequent of i .

If there exists a set of proposition $d_1..d_n$ with $d_i := (p_i^{ki}, O_i)$

and there exists an implication $i: (t, V, K)$ in the GFRN with p_1^{kl} in its consequent and $p_1^{kl} .. p_n^{kn}$ in its antecedent and all formal parameters $v \in V$ are determined by $d_1..d_n$

then expand implication i for proposition d_1 according to Fig. 11

until FPN complete;

remove ingoing edges from strong and deferred axioms;

Evaluation Phase:

Repeat evaluate FPN **until** steady state according to [KM96]

end

We now revisit our example scenario from Section 2 to exemplify how the specified GFRN of Fig. 7 and the defined inference mechanism is used to deduce the semantic information that was used to create the total many-to-one association between the classes WP and Developer in the object-model shown in Fig. 5 with a fuzzy belief of 0.6.

4. By default the threshold of transitions is set to zero at expansion time. It may be increased in order to eliminate limitcycles (cf. [KM96]).

For this example, we assume that it already has been deduced that table `Person` has the two variants `Developer` and `Manager`. We will use `n` as an abbreviation for `Developer.name` and `d` as an abbreviation for `WP.dev`.

At first, an initial automatic investigation of the application code in Fig. 3 retrieves the following three facts (cf. Section 2) which serve as strong axioms with fuzzy belief 1:

Strong axioms: $A^s := \{(\text{cycl_excl}^1, \{n\}, +, 1), (\text{sel_dist}^1, \{n\}, +, 1), (\text{join}^1, \{(n,d)\}, +, 1)\}$

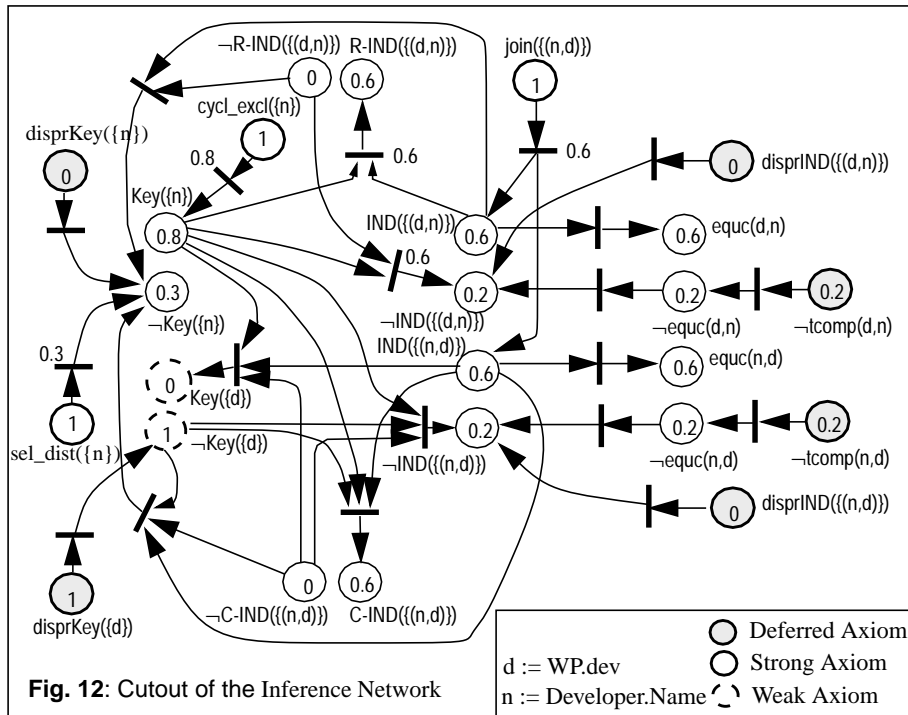
Furthermore, the reengineer has a subjective belief that with confidence 0.5 attribute `dev` is a key of table `WP`. This is represented as a weak axiom.

Weak axiom: $A^w := \{(\text{key}^1, \{d\}, +, 0.5)\}$

Moreover, there is an automatic procedure ϕ_1 that computes the fuzzy belief for compatibility of two given attribute types. Finally, we have automatic procedures ϕ_2 and ϕ_3 which try to disprove supposed key- and IND-constraints by finding counterexamples in the available extension. Since we want these three procedures to be executed on an on-demand basis, we define the following deferred axioms.

Deferred axioms: $A^d := \{(\text{tcomp}^1, +, \phi_1), (\text{disprKey}^1, +, \phi_2), (\text{disprIND}^1, +, \phi_3)\}$

The expansion of the FPN in Fig. 12 starts with the instantiation of the strong and weak axioms, i. e. we create places for the propositions $\text{cycl_excl}(\{n\})$, $\text{sel_dist}(\{n\})$, $\text{join}(\{(n,d)\})$, and $\text{Key}(\{d\})$.



Within the first expansion step, implication i_5 is applied to proposition $join(\{(n,d)\})$. This binds parameter v_1 of implication i_5 to $\{(n,d)\}$. Then the constraint $v_2=swap(v_1)$ binds parameter v_2 to $\{(d,n)\}$. Thus we create places for the two inclusion dependencies $IND(\{(n,d)\})$ and $IND(\{(d,n)\})$. Within the second expansion step, implication i_5 creates the transitions $join(\{(n,d)\}) \rightarrow IND(\{(n,d)\}) \wedge IND(\{(d,n)\})$ and $\neg join(\{(n,d)\}) \rightarrow \neg join(\{(n,d)\})$ and $\neg IND(\{(n,d)\}) \rightarrow \neg join(\{(d,n)\})$. However, the third expansion step removes the latter two transitions because they target a strong axiom.

The just created $IND(\{(n,d)\})$ proposition allows the instantiation of implication i_9 . This creates an equivalence class proposition $equc(n,d)$. This in turn enables the expansion of implication i_8 creating proposition $tcomp(n,d)$. Note that $tcomp(n,d)$ is defined by a deferred axiom. In addition, the expansion of implication i_6 creates proposition $disprIND(\{(n,d)\})$. Altogether, the inclusion dependency will be verified by analysing the type compatibility of the involved attributes and by searching for a counterexample in the database extension. The remainder of the depicted FPN is expanded analogously.

The evaluation of the expanded FPN starts by assigning the fuzzy beliefs of the strong and weak axioms to the corresponding places. Thus, the depicted propositions $cycl_excl$, sel_dist , and $join$ get a fuzzy belief of 1. The positive place of $Key(\{d\})$ gets the fuzzy belief 0.5 based on the reverse engineer's decision.

According to Fig. 8 and Fig. 9, the confidence of transition $cycl_excl(\{n\}) \rightarrow Key(\{n\})$ restricts the maximum fuzzy belief propagated to the positive place of $Key(\{n\})$ to 0.8. Analogously, the negative place of $Key(\{n\})$ gets a fuzzy belief of 0.3 from proposition $sel_dist(\{n\})$. Altogether, we have a higher belief that attribute name is a key of table Developer.

The evaluation of the proposition $disprKey(\{d\})$ detects two tuples of table WP with value Martin in their dev attribute. This counterexample assigns belief 1 to $disprKey(\{d\})$. This is propagated to the negative place of $Key(\{d\})$. Thus, the subjective belief of the reengineer that dev is a key of table WP has been disproved by a deferred axiom.

Furthermore, propositions $IND(\{(n,d)\})$ and $IND(\{(d,n)\})$ get a positive fuzzy belief of 0.6 from $join(n,d)$ and a negative fuzzy belief of 0.2 from the type compatibility test. Then, the fuzzy beliefs of $R-IND(\{(d,n)\})$ and $C-IND(\{(n,d)\})$ compute to 0.6. Together, the R-IND and the C-IND indicate an association between Developer and WP that has the cardinalities shown in Fig. 5 with a confidence of 0.6.

5 Concluding Remarks

In this paper we presented GFRNs as a new expressive graphical and executable formalism to specify uncertain reverse engineering knowledge. GFRNs have been defined formally and a first prototype of the proposed inference engine already exists. The expressiveness of GFRNs is now validated by applying them to practical examples as part of industrial projects.

We currently develop a GFRN editor that facilitates creation and customization of GFRN specifications. This will be used to build a database of reverse engineering knowledge for relational databases.

A major point is that GFRNs and the corresponding inference engine will not be used as stand alone tools. As soon as the inference engine is integrated into our database migration environment [JSZ96], the user may interactively add and verify assumptions about a database and derive all semantic information necessary for the migration to the object-oriented data model. This enables our migration environment to provide sophisticated support for the entire database migration process.

References

- [And94] M. Andersson. Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In *Proc. of the 13th Int. Conference of the Entity Relationship Approach, Manchester*, pages 403–419. Springer, 1994.
- [CS80] J. D. Carney and R. K. Scheer. *Fundamentals of Logic*. Macmillan Publishing Co., Inc., 1980.
- [DA87] K. H. Davis and A. K. Arora. Converting a Relational Database Model into an Entity-Relationship Model. In *Proc. of the 6th Int. Conference of the Entity Relationship Approach, New York*, pages 271–285. North-Holland, November 1987.
- [DLP94] D. Dubois, J. Lang, and H. Prade. *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 449–403. Clarendon Press, Oxford, 1994.
- [DP88] D. Dubois and H. Prade. An introduction to possibilistic and fuzzy logics. In P. Smets, E. H. Mamdani, D. Dubois, and H. Prade, editors, *Non-Standard Logics for Automated Reasoning*, pages 287–326. Academic Press, London, 1988.
- [FV95] C. Fahrner and G. Vossen. Transforming Relational Database Schemas into Object-Oriented Schemas according to ODMG-93. In *Proc. of the 4th Int. Conf. of on Deductive and Object-Oriented Databases 1995*, 1995.
- [JK90] P. Johannesson and K. Kalman. A method for translating relational schemas into conceptual schemas. In F. H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*. ERI, 1990.
- [JSZ96] J. H. Jahnke, W. Schäfer, and A. Zündorf. A design environment for migrating relational to object oriented database systems. In *Proc. of the 1996 Int. Conference on Software Maintenance (ICSM'96)*. IEEE Computer Society, 1996.
- [JSZ97] J. H. Jahnke, W. Schäfer, and A. Zündorf. Specification and implementation of generic fuzzy reasoning nets using programmed graph rewriting systems. Technical report, University of Paderborn, 1997. forthcoming.

- [KM96] A. Konar and A. K. Mandal. Uncertainty management in expert systems using fuzzy petri nets. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):96–105, February 1996.
- [Loo88] C. G. Looney. Fuzzy petri nets for rule-based decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):178–183, February 1988.
- [LS88] C. Lewerentz and A. Schürr. GRAS, a management system for graph-like documents. In *Proc. of the 3rd Int. Conf. on Data and Knowledge Bases*. Morgan Kaufmann, 1988.
- [Mai89] D. Maier. Making database systems fast enough for CAD applications. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 573–582. Addison-Wesley, 1989.
- [Paa88] G. Paass. Probabilistic logic. In P. Smets, E. H. Mamdani, D. Dubois, and H. Prade, editors, *Non-Standard Logics for Automated Reasoning*, pages 213–251. Academic Press, London, 1988.
- [PB94] W. J. Premerlani and M. R. Blaha. An approach for reverse engineering of relational databases. *Communications of the ACM*, 37(5):42–49, May 1994.
- [Pet81] J. L. Peterson. *Petri Net Theory and Modeling of Systems*. Prentice Hall, 1981.
- [PKBT94] J-M. Petit, J. Kouloumdjian, J-F. Boulicaut, and F. Toumani. Using queries to improve database reverse engineering. In *Proc. of 13th Int. Conference of ERA, Manchester*, pages 369–386. Springer, 1994.
- [SK90] F. N. Springsteel and C. Kou. Reverse Data Engineering of E-R Designed Relational Schemas. In *Proc. of Databases, Parallel Architectures and their Applications*, pages 438–440. Springer, March 1990.
- [SLGC94] O. Signore, M. Loffredo, M. Gregori, and M. Cima. Reconstruction of er schema from database applications: a cognitive approach. In *Proc. of 13th Int. Conference of ERA, Manchester*, pages 387–402. Springer, 1994.
- [Zad78] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1978.