

Design Recovery of Legacy Database Applications based on Possibilistic Reasoning

Jens H. Jahnke and Melanie Heitbreder
AG-Softwaretechnik, Fachbereich 17, Universität Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany;
e-mail: [jahnke|mela]@uni-paderborn.de

Abstract

Industrial database applications often evolve over three or more generations of developers, cover several hundred thousand lines of code and maintain a vast amount of data. A rapidly growing number of companies face the problem that they have to adapt or modernise such existing legacy database applications (LDA) in order to keep up with emerging requirements. The documentation of such LDAs is often obsolete as they have been developed over several generations of programmers. This paper presents an application of possibilistic reasoning to infer the semantic information that is necessary to recover the conceptual design of an LDA. A dedicated, graphical language (called Generic Fuzzy Reasoning Nets) is introduced to specify and customise the applied reverse engineering process. The actual reasoning process is performed by a nonmonotonic inference engine based on fuzzy petri nets which supports lazy execution of expensive analysis operations.

1. Introduction

Industrial database applications often evolve over three or more generations of developers, cover several hundred thousand lines of code and maintain a vast amount of data. A rapidly growing number of companies face the problem that they have to adapt or modernise such existing legacy database applications (LDA) in order to keep up with emerging requirements. These requirements may comprise the integration/federation of an LDA with other information systems (e.g. to allow for data mining) or the migration to new technologies like object-oriented databases or programming languages. A typical problem that complicates such a maintenance task is that the documentation of an LDA has become obsolete and the developers who are familiar with the application have left the project.

The research area of *database reverse engineering* (DRE) focuses on the development of techniques, methodologies and tools to improve human understanding of complex legacy database applications. A major DRE task is to recover the conceptual data structure of an LDA which might be implemented on top of a relational database. Due to its simplicity, the relational data model cannot directly

express abstract modelling concepts like inheritance, aggregation and n-ary associations. However, indicators for such modelling concepts are scattered over all parts of an LDA, i.e. its procedural code, its database schema definition, its data, its obsolete documentation, etc. Furthermore, schemas of LDAs often comprise optimisation structures that might not be easy to recognize. An RE tool should find and weigh all these uncertain (and partially contradicting) indicators according to their credibility in order to reconstruct an up-to-date design. Moreover, users and developers should be considered as a valuable source of information due to their domain knowledge. Existing approaches do not support this kind of analysis process. Those few approaches, e.g. [2,1,6,5,7], which provide means for design recovery of LDAs are not able to deal with uncertainty and contradicting assumptions. Furthermore, in most available DRE tools the employed reverse engineering process can hardly be customised or extended, because it is hard-coded.

In our approach, we use possibilistic reasoning [11,12] to perform the analysis process. The DRE knowledge is specified in a dedicated high-level language called *Generic Fuzzy Reasoning Net* (GFRN) and, thus, it can easily be customised. The actual inference process is performed by evaluating a *fuzzy petri net* (FPN)[10] which has been generated from the GFRN specification.

The remainder of this section describes a small reverse engineering sample scenario in order to take a closer look at the problem domain. Section 2 introduces the GFRN formalism and exemplifies its use to specify DRE knowledge. Section 3 sketches how a GFRN specification is executed based on an FPN. Finally, Section 4 closes with some concluding remarks.

A reverse engineering sample scenario

In the following, we assume that the reader is familiar with the basic concepts of relational database design[13]. Figure 1 shows an example LDA including small details of its procedural code, its database schema and the stored data. The recovered conceptual schema in EER-notation is depicted in Figure 2, while the semantic information which

has to be deduced in order to reobtain this abstraction is shown in Figure 3.

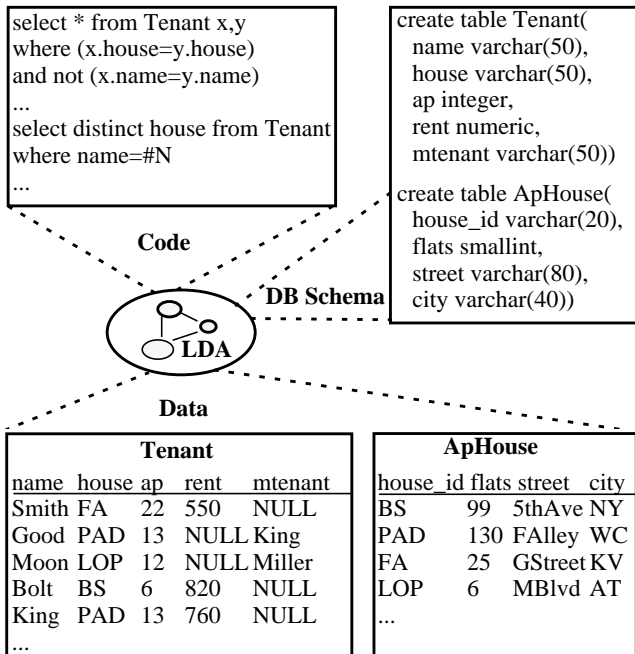


Figure 1. Details of a Legacy Database Application

Often, the schema definition of an LDA does not contain explicit definitions for foreign keys or candidate keys [13]. This is because of the limited functionality of antiquated database management systems. Possible keys can be found by searching the code of the LDA for special patterns (sometimes called *clichés*). Examples for instances of clichés are the two SQL queries in the upper left part of Figure 1. The first cliché is called a *cyclic join* [1]. It selects two entries in table `tenant` with the same value in column `house` but with different values in column `name`. This cliché serves as an indicator that tenants might be distinguishable by their name. On the other hand, the second query is an indicator against the assumption that `name` might be a key of table `tenant`: the column `house` of a tenant with a given name is selected, but the query contains the keyword `distinct`, which is used to avoid multiple elements with the same value in the result of a query. However, experience shows that the first cliché deserves a higher credibility than the second one. The assumption that `name` is a key might be disproved or supported by examining the data in table `tenant`. If there are two rows with the same value in column `name` the assumption must be refuted. On the other hand, if all rows have distinct values in column `name`, the assumed key could gain a greater credibility depending on the extent of the provided data in table `tenant`, e.g. if there are six hundred tenants with unique names there should be greater confidence that the assumption is true than if there are only six tenants.

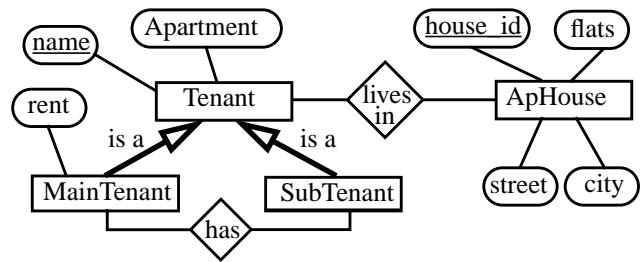


Figure 2. : Conceptual Schema in EER-Notation

In order to detect foreign keys between tables one can search the application code for *join* clichés, like the first query in Figure 1. Further indications for a possible foreign key might be retrieved by checking the similarity of column names with other column names (e.g. columns `house` and `house_id`) or table names (e.g. column `mtenant` and table `tenant`). Obviously, columns which are supposed to have identical meaning should have the same type. For example columns `house` and `house_id` both are of type `varchar` but with a different length. However, as this occurs frequently even for columns with identical meaning, their types should be considered as compatible to a certain degree. Like before, possible foreign keys should be checked against the available data.

Keys:

Tenant(`name`)
ApHouse(`house_id`)

Equivalence Classes:

{Tenant.`mtenant`,Tenant.`name`}
{Tenant.`house`, ApHouse.`house_id`}

Foreign Keys:

Tenant.`mtenant` *references* Tenant.`name`
Tenant.`house` *references* ApHouse.`house_id`

Variants:

Tenant: MainTenant(`name`,`house`,`ap`,`rent`)
SubTenant(`name`,`house`,`ap`,`mtenant`)

Figure 3. Deduced Semantic Information

A further examination of the contents of table `tenant` shows that there seem to be two different variants of tenants: Each row in table `tenant` has either a `NULL`-value in column `rent` or in column `mtenant`. This reveals a hidden inheritance hierarchy. Again, the credibility of this indicator depends on the extent of the available data in table `tenant`.

2. Generic Fuzzy Reasoning Nets

The scenario given above exemplifies the role of heuristics in the DRE domain in order to find and weigh (contradicting) indicators for semantically higher modelling concepts in LDAs. The set of heuristics, respectively the weight of indicators which are used to analyse different LDAs, depends on technical parameters, like the database systems or the programming languages which are used. However, they might as well depend on non-technical parameters, like the personal programming style of

developers. For example it might be that a conscientious developer uses the `select_distinct` keyword only when she expects duplicate tuples in the result set of a query. Thus, a `select_distinct` indicator found in her legacy code deserves a rather high credibility. This is in contrast to the credibility of such an indicator in the code written by a more 'relaxed' developer, who tends to use this keyword in her queries as default, even if it is not necessary. This shows that flexibility in terms of customisability and extensibility is a crucial requirement for the applicability of DRE tools. To meet this requirement we have developed a dedicated high-level formalism to specify and customize DRE knowledge, so-called *Generic Fuzzy Reasoning Nets* (GFRN). This allows us to develop a generic DRE environment that is parameterisable by GFRN specifications and executes the specified analysis process [4,3].

A GFRN is a graphical network of fuzzy predicates (with oval shape) and weighted implications (represented as boxes) which are connected by arcs. We employ necessity-valued possibilistic logic [12], i.e. each weight represents the necessity that the corresponding implication is valid. Negations are represented by arcs with black arrow heads. Arcs can be labelled with formal parameters that can be used to specify constraints for implications. Before we give a formal definition we will illustrate the proposed formalism through an application example: Figure 4 shows a detail of a GFRN specification which aims to detect possible foreign keys in LDAs: Implication i_1 specifies the before mentioned heuristic that a *cyclic join* cliché is a rather credible indicator for a possible key candidate. On the other hand, implication i_2 models our experience that a *select-distinct* cliché over a set of columns v_1 serves as a negative indicator that subsets of v_1 might be keys. Implication i_3 specifies that an assumed key candidate may only be valid if there exists no counterexample in the data of the LDA.

A frequently used heuristic to detect relationships between tables, i.e. to detect equivalent columns in different tables, is to check column names for similarities. Obviously, it is necessary that columns which have identical meaning are type compatible. This heuristic is modelled with Implications i_8 and i_9 . Implication i_7 is an example that even complex rules can be expressed in GFRN specifications. It specifies that pairs of equivalent columns in two different tables are an indicator for an inclusion dependency (IND) over these columns in either direction. At this, `sameTable` is defined to be a boolean function that returns *true* iff all columns in its argument are in the same table. Functions π_1 and π_2 represent the relational projection on the first and the second element of a pair v_3 , respectively. Parameter v_1 is constrained to be a set of pairs v_3 . Finally, function `swap` is defined to exchange the order of each pair in a set of pairs. This reflects the possibility of INDs in both directions.

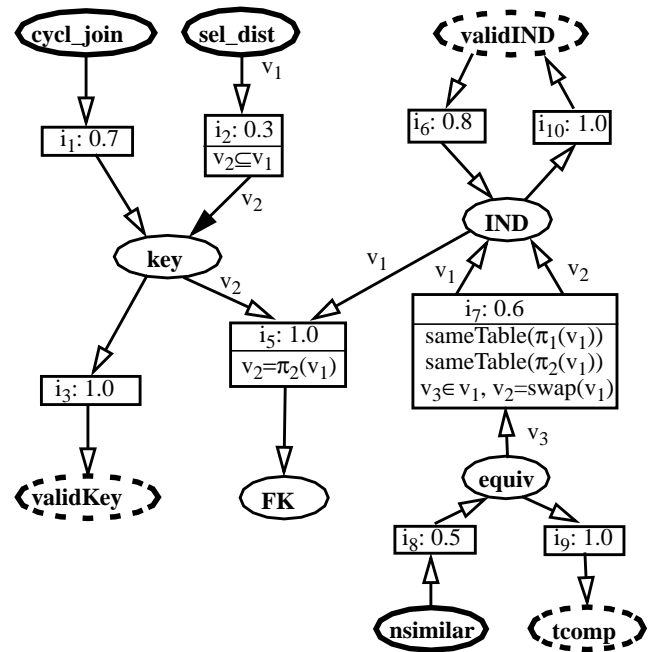


Figure 4. : Generic Fuzzy Reasoning Net

Again, the available data has to be checked whether or not the supposed INDs can be disproved (Implication i_{10}). Implication i_6 specifies that the validation of an assumed IND over a huge amount of available data may further support this assumption. Finally, the presence of a foreign key can be deduced from the existence of an IND over pairs of columns, where the second element of each pair must constitute a key candidate (Implication i_5).

Some fuzzy propositions can be determined by automatic analysis operations, e.g. checking for similar column names, comparing column types, validating assumptions against the available data, searching the code for clichés, etc. In the GFRN formalism, a fuzzy predicate can be bound to an automatic analysis operation. Such predicates are called *axioms* and are represented by bold ovals in the graphical notation. Axioms provide the initial amount of information that is used in the inference process to deduce further fuzzy propositions. However, it is important for the applicability of an analysis tool that expensive analysis operations are performed only on demand, i.e. when the information they provide is really needed. Thus, we distinguish between *strong axioms* and *deferred axioms*. Analysis operations that are bound to strong axioms are performed, before the actual inference process starts. As an example, the strong axiom `nsimilar` is bound to an operation that compares the names of different columns according to a possibility function as a measure of similarity. Such a possibility function could for example be based on string similarity metrics like the Levenshtein-distance [8]. In contrast to strong axioms, the analysis operation of a deferred axiom is performed on demand while the inference process is in

progress. The validation of an IND via the available data (validIND) is an example for an expensive operation that should only be performed for already indicated INDs. In Section 3 we will explain in detail how and when analysis operations of deferred axioms are executed.

Formally, a GFRN is defined by a tuple $(P, F, I, E, cf, A^s, A^d)$, where $P = \{p_1, p_2, \dots, p_z\}$ is a set of *predicates* and $F = \{f_1^{u1}, f_2^{u2}, \dots, f_x^{ux}\}$ is a set of *functions* with arity $uq \in \mathbb{N}$. $I = \{i_1, i_2, \dots, i_m\}$ is a set of *implications*, each implication is a tuple $i_g = (i_g, V_g, K_g)$, with a unique *implication identifier* i_g , a set of *parameters* $V_g = \{v_1, v_2, \dots, v_f\}$, and $K_g = \{k_1, k_2, \dots, k_s\}$, a set of *constraints* over V_g , with $k_x = (w, r, f^u)$, (w_1, w_2, \dots, w_u) where $w, w_1, \dots, w_u \in V_g$, $r \in \{\in, \subseteq, =\}$ and $f^u \in F$. Furthermore, $E = \{e_1, e_2, \dots, e_n\}$, $n \in \mathbb{N}^+$ is a set of *arcs*. Each arc is a tuple $e_g = (l_g, s_g, d_g, \alpha_g)$, with a *location* $l_g = (p_q, (v, V, K)) \in (P \times I)$, a *sign* $s_g \in \{+, -\}$, a *direction* $d_g \in \{\text{antecedent}, \text{consequent}\}$, and an *actualization parameter* $\alpha_g \in V$. The *confidence association* $cf: I \rightarrow]0, 1[$ associates real values to implications. $A^s = \{a^s_1, a^s_2, \dots, a^s_z\}$ is a set of strong axioms, each strong axiom is a tuple $a^s_q = (p_q, \Omega_q)$, with $p_q \in P$ and a function $\Omega_q: \mathcal{P}(U^{nq}) \times [0, 1] \times [0, 1]$, $nq \in \mathbb{N}$. At this, $\mathcal{P}(U^{nq})$, denotes the power set of nq -tuples of elements of a set U . In our formal model, constant function Ω_q represents an analysis operation that is performed before the inference process starts. It delivers a set of fuzzy propositions, where the semantics of each proposition (c, n^+, n^-) is that n^+ and n^- express the necessities that predicate p_q is or is not fulfilled for the constant set of nq -tuples c over the universe of discourse U . Finally, $A^d = \{a^d_1, a^d_2, \dots, a^d_y\}$ is a set of deferred axioms, each deferred axiom is a tuple $a^d_q = (p_q, \omega_q)$, with $p_q \in P$ and a function $\omega_q: \mathcal{P}(U^{nq}) \rightarrow [0, 1] \times [0, 1]$. Function ω_q represents an analysis operation that is performed on demand for a given constant c and it delivers necessity degrees that $p_q(c)$ is true or false.

The semantics of GFRN implications is formally defined by a translation in a set of closed, weighted formulas in first order logic, whereby all formal parameters are implicitly universal quantified. For example implication i_2 is translated to a formula $(\forall v_1)(\forall v_2 \subseteq v_1)(sel_dist(v_1) \Rightarrow \neg key(v_2))$ with weight 0.3. The reader should note that this semantics is unlike the semantics of most fuzzy rule based systems. The difference is that usually rule based systems do not consider the contraposition of a rule, i.e. in such systems a rule *if A then B* does not imply the rule *if not B then not A*. In the GFRN approach contrapositions of implications are considered as this enables nonmonotonic reasoning. Nonmonotonic reasoning is needed for an semi-interactive reasoning process and to defer costly analysis operations.

3. The fuzzy inference engine

We decided to use a fuzzy petri net (FPN) as the inference engine of a GFRN specification as it allows for efficient

evaluation and nonmonotonic reasoning [10]. Generally, a petri net [9] is a directed, bipartite graph with active and passive elements. The particular FPN model used in our approach is an extension of the FPN model described in [10]. The major difference compared to the original FPN model is that our approach supports inconsistent analysis results, i.e. for every proposition d , there are two fuzzy values: the first fuzzy value represents the necessity that d is fulfilled, while the second value represents the necessity that d is not fulfilled. Formally, a FPN is a tuple $N = (Pl, T, A, D, m, b, c, th, s)$ with a set of places Pl , a set of transitions T and a finite set of arcs $A \subseteq (Pl \times T) \cup (T \times Pl)$. Each place is associated to a proposition $d_i \in D$ by a bijective function $b: Pl \rightarrow D$. The fuzzy marking function $m: Pl \rightarrow [0, 1] \times [0, 1]$ assigns a pair of real values to each place, where $m(p_z) = (n^+, n^-)$ is called the *fuzzy belief marking* (FBM) of place p_z , where n^+ and n^- represent the positive and the negative fuzzy belief that $b(p_z)$ is true or false, respectively. Furthermore, a threshold th and a certainty factor c are assigned to each transition $c, th: T \rightarrow [0, 1]$. Finally, $s: A \rightarrow \{+, -\}$ is a function which labels each arc with a sign.

The FPN is evaluated by iteratively executing the following two steps illustrated in Figure 5: the first step generates new *fuzzy truth tokens* (FTT) at the outgoing arcs of each transition, while subsequently the second step computes new FBMs at each place in the FPN.

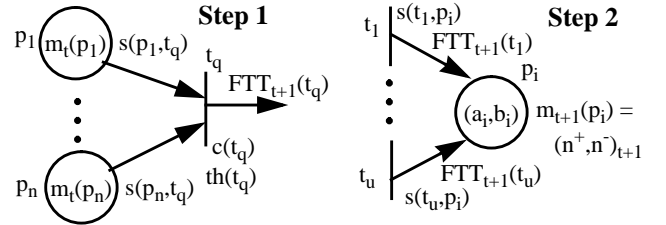


Figure 5. Calculation of Fuzzy Beliefs

The FTT at transition t_q and iteration $t+1$ is obtained as follows: at first, according to the sign of the connecting arc the *absolute fuzzy belief* (AFB) of each place p_z with marking $m_t(p_z) = (n^+, n^-)_t$ in the antecedent of t_q is computed as

$$AFB_{t+1}(p_z, t_q) = \begin{cases} n_t^+ - n_t^-, & \text{if } n_t^+ > n_t^- \text{ and } s(p_z, t_q) = + \\ n_t^- - n_t^+, & \text{if } n_t^- > n_t^+ \text{ and } s(p_z, t_q) = - \\ 0, & \text{else} \end{cases} \quad (\text{eq. 1})$$

The transition t_q is *enabled*, i. e. it may fire with an FTT greater than zero, if the minimum AFB (AFB^{min}) in the antecedent of t_q is greater than its threshold th_q . In this case, the new FTT of t_q is obtained by taking the minimum of the certainty factor c_q and AFB^{min} as shown in the following equations, where the fuzzy-AND operator \wedge is defined by the minimum function:

$$f_{t+1}(t_q) = \begin{cases} c(t_q) \wedge AFB^{min}, & \text{if } AFB^{min} > th(t_q) \\ 0, & \text{else} \end{cases} \quad (\text{eq. 2})$$

$$\text{with } AFB^{min} = \bigwedge_{1 \leq z \leq k} \{AFB_{t+1}(p_z, t_q)\} \quad (\text{eq. 3})$$

The new FB $m_{t+1}(p_i) = (n_{t+1}^+, n_{t+1}^-)$ at each place p_i is given by two equations:

$$n_{t+1}^+ = \begin{cases} \bigvee_{z \in [1, u] | s(t_z, p_i) = +} FTT_{t+1}(t_z), & \text{if } \exists I \leq i \leq u: s(t_z, p_i) = + \\ n_t^+, & \text{else} \end{cases} \quad (\text{eq. 4})$$

$$n_{t+1}^- = \begin{cases} \bigvee_{z \in [1, u] | s(t_z, p_i) = -} FTT_{t+1}(t_z), & \text{if } \exists I \leq i \leq u: s(t_z, p_i) = - \\ n_t^-, & \text{else} \end{cases} \quad (\text{eq. 5})$$

The two iteration steps are repeated until *steady-state* is reached, i.e. no further changes to the FB of any place can be obtained. As our inference mechanism deals with nonmonotonic reasoning, cycles may occur in the FPN, which possibly lead to infinite periodic oscillations of FB values, so-called *limitcycles* [10]. If limitcycles exist in a FPN steady-state cannot be reached. Due to the lack of space we cannot discuss this problem in detail. However, in [10] the authors present an algorithm for detecting and eliminating limitcycles in their FPN model that can be applied to our approach with only minor changes.

When the FPN is in steady-state the analysis results are presented to the reengineer for further investigation. One might want to further examine propositions with a low or inconsistent FBM. As a measure of inconsistency, we have defined the function $incons(Pl) \rightarrow [0, 1]$, which is defined as

$$incons(p_z) = n^+ \wedge n^-, \text{ with } m(p_z) = (n^+, n^-). \quad (\text{eq. 6})$$

This allows the reengineer to query the system for analysis results that have a greater inconsistency than a given value or to examine further propositions with an AFB in a given interval. As soon as the reengineer gains further knowledge about the LDA the inference process can be resumed in order to gain new analysis results. In our approach, interactively added pieces of knowledge that represent the subjective belief of the reengineer are called *weak axioms* and may be refuted or supported by the inference engine. The reverse engineering process described above is illustrated in Figure 6.

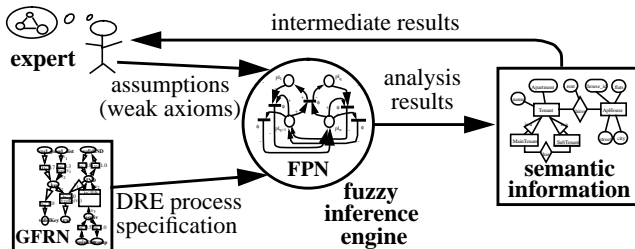


Figure 6. The Reverse Engineering Process

Expansion and evaluation of an FPN based on a GFRN

As our approach allows for lazy execution of (expensive) analysis operations according to intermediate deduction results, the proposed inference process consists of an iteration of expansion and evaluation steps in an FPN based on a given GFRN specification. Figure 7 gives an overview on the proposed inference algorithm: the inference process starts by executing all analysis operations which are assigned to strong axioms in the GFRN. For each fuzzy proposition that is in the result set of such an operation a new FPN place is created. The following steps are performed iteratively, until no further knowledge can be deduced. At first, the reengineer is allowed to add new assumptions about the LDA, interactively. Then, based on this initial knowledge and the defined GFRN implications the FPN is extended by new places.

algorithm inference

begin

create empty FPN;

create new places according to results of analysis operations which are assigned to strong axioms;

Repeat

create new places according to weak axioms

(interactively added assumptions);

expand FPN places forward according to Fig. 8;

expand FPN places backward according to Fig. 8;

perform analysis operations of deferred axioms that are assigned to predicates of newly created places;

expand FPN transitions according to Fig. 9;

Repeat evaluate FPN **Until** steady state according Fig. 5;

Until no further change to the FPN

end

Figure 7. The Inference Algorithm

This is done in a *forward* and a *backward* expansion step, shown in Figure 8. The forward expansion step searches the FPN for fuzzy propositions which are instances of predicates in the antecedent (pl_{q+1}, \dots, pl_z) or consequent (pl_1, \dots, pl_{q-1}) of an implication i in the GFRN. Then, it is checked whether all formal parameters of i are completely determined by the constant values of the found propositions. In this case, it is furthermore checked if all places in the antecedent (pl_{q+1}, \dots, pl_z) carry an $AFB > \epsilon$ according to the sign of the arc that connects the corresponding predicate with implication i in the GFRN. If this test succeeds, the forward expansion step creates places in the FPN for each proposition in the consequent of i (according to the current parameter bindings). At this, the real number $\epsilon \in [0, 1]$ allows the user to define a level for the minimum AFB that has to be fulfilled such that it is worth the effort of further expanding the FPN (and performing further deferred analysis operations, etc.).

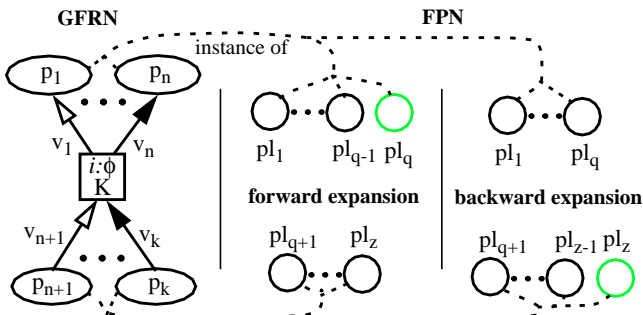


Figure 8. Expansion of FPN places

The right side of Figure 8 illustrates the backward expansion step. Again, it is checked whether all formal parameters of implication i are determined by the propositions associated with (pl_1, \dots, pl_z) . In this case, it is checked if there is at least one place pl_x in the consequent (pl_1, \dots, pl_q) with an FBM $m(pl_x) = (n^+, n^-)$ where $n^+ > \epsilon$ or an inconsistency value $incons(pl_x) > \epsilon$. If this test succeeds the backward expansion step creates places in the FPN for each proposition in the antecedent of i .

After evaluating all deferred axioms according to the parameter binding of the newly created places, in the next step the FPN is expanded by further transitions. Figure 9 illustrates that for every implication in the GFRN at least two transitions in the FPN are created. This is, because every implication entails its contraposition. In Figure 9 the transitions and arcs which represent the contraposition have grey colour. However, there is an important exception from the expansion rule shown in Figure 9: places which have been created due to strong or deferred axioms do not have incoming arcs in the FPN. This is because their FBMs represent definite fuzzy facts and, thus, have to be immutable.

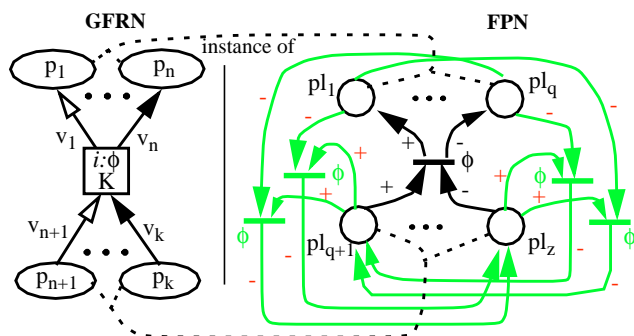


Figure 9. Expansion of FPN transitions

When the FPN is completely expanded, it is evaluated until *steady state* is reached. Then, the inferred analysis result and/or interactively added domain knowledge might iteratively enable further expansion steps. The inference process is finished when no further knowledge can be deduced or annotated.

4. Concluding remarks

In this paper, we show how possibilistic reasoning can be applied to design recovery of legacy database applications. As opposed to existing, mostly hard-coded approaches, we present GFRNs as a dedicated, high-level language to specify RE processes. This approach provides the flexibility for easy customisation and extension of uncertain RE knowledge. We furthermore present a framework for an inference engine that is based on a dedicated class of fuzzy petri nets and allows for lazy evaluation of expensive analysis operations. We currently develop a GFRN editor and a prototype of the inference engine. Both components will be integrated parts of our database migration environment [3]. Currently, the expressiveness of the GFRN formalism is validated by applying them to practical examples as part of industrial projects.

References

- [1] M. Andersson. Extracting an Entity Relationship Schema from a RDB through Reverse Engineering. In *Proc. of 13th Int. Conf. of the ERA, Manchester*. Springer, 1994.
- [2] 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*.
- [3] 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. Conf. on Software Maintenance (ICSM'96)*. IEEE CS, 1996.
- [4] J. H. Jahnke, W. Schäfer, and A. Zündorf. Generic Fuzzy Reasoning Nets as a Basis for Reverse Engineering RDB Applications. In *Proc. of (ESEC/FSE)*. Springer, 1997.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] L. Allison, C. S. Wallace and C. N. Yee. When is a String like a String?. In *AI & Maths*. 1990
- [9] J. L. Peterson. *Petri Net Theory and Modeling of Systems*. Prentice Hall, 1981.
- [10] A. Konar and A. K. Mandal. Uncertainty management in expert systems using fuzzy petri nets. *IEEE Transactions on Knowledge and Data Engineering*, February 1996.
- [11] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1978.
- [12] D. Dubois, J. Lang, and H. Prade. Possibilistic Logic. In D. M. Gabbay et. al. (editors), *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 3, pages 439–513. Clarendon Press, Oxford, 1994.
- [13] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. Benjamin Cummings, Redwood City, CA. 1992