Knowledge coding methods for rule-based expert systems

PETR POLACH, JAN VALENTA, VACLAV JIRSIK
Department of Control and Instrumentation
Brno University of Technology
Kolejni 2906/4, 612 00 Brno
CZECH REPUBLIC
polachpetr@phd.feec.vutbr.cz, jirsik@feec.vutbr.cz
http://www.uamt.feec.vutbr.cz

Abstract: This paper gives an overview of knowledge representation methods that are currently being implemented for use in a hybrid expert system shell that has been under development at the Department of Control and Instrumentation, BUT. Two approaches are discussed – a diagnostic and a planning expert system knowledge base coding. A new method suitable for diagnostic rule-based expert systems with automated weight tuning, deriving from neural networks, is proposed. Next, an approach to modeling knowledge bases using Petri Nets is discussed and inference engine operations are compared for both diagnostic and planning expert system.

Key-Words: Expert systems, Knowledge Engineering, Knowledge base tuning, Resla, Petri Nets

1 Introduction

Expert systems are interactive computer programs for decision support. With the goal of reducing costs, speeding up the decision making process and making it available no matter what the time or place used, they substitute human experts [12], [14]. Expert systems (ES) in general utilize numerous methods for representing expert knowledge – that is acquiring the problem-solving heuristic from the actual human expert, coding it in a proper way and of course using it later for the expert system operation. Every knowledge base needs to be thoroughly tuned and verified so that the results given in any computer consultation session are as coherent with the human expert advice as possible [3], [5], [17].

Even though the basic structure of the knowledge base can be built quite easily (it is usually not difficult to determine the basic terms of the problem domain and to link them up based on mutual dependencies), common sense or even expert knowledge may not be enough to set up the weights of such links or may not lead to finding all the necessary connections. The process of creating and tuning a knowledge base can therefore be very long, costly and it is not guaranteed that the results will be satisfactory. Many times this leads into limited use of expert systems.

Our goal is to aid the process of knowledge base creation, tuning and deployment by presenting appropriate SW tools and methods. During our previous work we created an experimental modular expert system shell [17] and we will now describe the basics of knowledge representation methods that are implemented (or are currently being implemented) into it. Section 2 of this papers deals with a new methods for diagnostic systems – RESLA [25]. This method stems from the similarities between rule networks and artificial neural networks. An algorithm for network result error propagation thus *tuning* the network's weights is introduced. Sections 4 and 5 deal with the usage of Petri nets in expert systems. The inference engine based on Petri nets operation is described and examples are given therein.

2 RESLA method

Here we propose a knowledge base structure that allows for direct tuning of rule weights using a modified back-propagation algorithm. The knowledge base is created in a form of an oriented acyclic graph where the first layer nodes stand for user responses (inputs from an analyzed process), the intermediate layer nodes are used for expressing mutual dependencies and supportive hypothesis (i.e. intermediate consultations results) and the output layer members represent the consultation result-hypothesis.

The RESLA method is a supervised learning algorithm that uses the error function gradient for modifying rule weights. As with neural network back-propagation algorithm, it is necessary to have a set of patterns with a set of relevant responses. Different are the "transfer" functions of the rules (1). The condition of rule transfer function differentiability applies. y_j^n is the output of the *j*th rule in the *n*th layer, w_j^n is its weight, $\mathbf{x}$ is the rule antecedent vector and $\Theta(\mathbf{x})$ is the AND/OR function.

$$y_j^n = w_j^n \cdot \Theta(\mathbf{x}) \tag{1}$$

The standard AND, OR functions are non differentiable, therefore we replace them with the t-norm (2) or s-norm (3) respectively. y is the function output, x_i are input variables.

$$y = \prod_i x_i \tag{2}$$

$$y = \bigcup(x_1, x_2) = x_1 + x_2 - x_1 \cdot x_2 \tag{3}$$

The rules used within a knowledge base are defined as follows:

- Logical product (4)

$$y_j^n = w_j^n \cdot \prod_{i=1}^{N} \left(\lambda_j^n \left(y_i^{n-1}\right)\right), \tag{4}$$

where w_j^n is the weight of jth rule in the nth layer, y_i^{n-1} is the output of the ith rule in the $(n-1)$th layer, N is the number of jth rule inputs and $\lambda_j^n = 1 - x$ for a negated link; $\lambda_j^n = x$ otherwise.

- Logical sum (5)

$$y_j^n = w_j^n \cdot \overset{N}{\underset{i=1}{OR}}\left(\lambda_j^n \left(y_i^{n-1}\right)\right), \tag{5}$$

where $\overset{N}{\underset{i=1}{OR}}(\mathbf{x})$ is a recursive function

$$\overset{N}{\underset{i=1}{OR}}(\mathbf{x}) = \bigcup\left(x_1, \bigcup\left(x_2, \ldots \bigcup(x_{i-1}, x_i)\right)\right). \tag{6}$$

The aggregation function (7) provides for the composition of individual rule- sub-tree influences into the probability values of the goal node (goal hypothesis).

$$y_j^n = w_j^n \cdot \sum_{i=1}^{N} \left(\gamma_i^n \left(y_i^{n-1}\right)\right), \tag{7}$$

where y_j^n is the output of the jth rule in the nth (output) layer. $\gamma_j^n = -x$, for a negated link; $\gamma_j^n = x$ otherwise.

At the beginning of weight optimization, all the weights $\mathbf{w}$ are set randomly from $\langle 0; 1\rangle$. In every step, one model input is submitted. The knowledge base as a whole executes projection $\Psi: \mathbf{x} \to \mathbf{y}$. $\mathbf{y} = \Psi(\mathbf{x})$, where $\mathbf{y}$ is the output vector, $\mathbf{x}$ is the input vector, Ψ is the knowledge base system function. The error function is then

$$\mathbf{e} = \mathbf{d} - \mathbf{y} = \mathbf{d} - \Psi(\mathbf{x}). \tag{8}$$

$\mathbf{d}$ is the vector of desired outputs. The immediate quadratic error for one rule output (from (1) and (8)) is then

$$e_j = \frac{1}{2}\sum_{p=1}^{M}\left({}_p d_j - {}_p w_j^n \cdot {}_p y_j\left(\mathbf{w}_j, {}_p \mathbf{y}_j^{n-1}, \lambda_j\right)\right)^2 = \\ = \frac{1}{2}\sum_{p=1}^{M}\left({}_p d_j - {}_p w_j^n \cdot {}_p \Theta_j\left(\lambda_j^n\left({}_p y_j^{n-1}\right)\right)\right)^2, \tag{9}$$

where M is the number of examples belonging to the jth output, p is the respective sample. Similarly, the error for the aggregation rule can be determined. The quality criterion is

$$\varepsilon = E\{e_j\}. \tag{10}$$

The weights are set according to the delta rule

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu_k \nabla e, \tag{11}$$

μ_k is the learning constant (determining the speed of adaptation process), k is step index. (12) shows the rule (11) rewritten for individual knowledge base components

$$w_j^n(k+1) = w_j^n(k) - \mu_k \frac{\partial e}{\partial w_j^n}(k). \tag{12}$$

Since the rule function is $y_j^n = w_j^n \cdot \Theta_j(\mathbf{x})$, we can express the gradient of the error function according to the respective weight w as

$$\frac{\partial \varepsilon_j^n}{\partial w_j^n} = \frac{\partial \varepsilon_j^n}{\partial y_j^n} \cdot \frac{\partial y_j^n}{\partial w_j^n} = \frac{\partial \varepsilon_j^n}{\partial y_j^n} \cdot \alpha_j^n, \tag{13}$$

here α_j^n denotes the rule's inner "potential". For the output layer, we have

$$\frac{\partial \varepsilon_j^n}{\partial w_j^n} = \frac{1}{2} 2\left(d_j - w_j \cdot \Theta(\mathbf{x}_j)\right)\left(-\Theta(\mathbf{x}_j)\right) = -\left(d_j - y_j^{n_0}\right) \cdot \alpha_j^{n_0} \tag{14}$$

Similarly, the gradients can be expressed for the aggregation rules and for intermediate layers (15)

$$\frac{\partial \varepsilon_j^{n-1}}{\partial y_j^{n-1}} = \sum_{y_j^{n-1} \to \Theta_j^n} \frac{\partial \varepsilon_j^n}{\partial y_j^n} \cdot w_j^n \cdot \frac{\partial \Theta_j^n\left(\lambda_j^n\left(y_j^{n-1}\right)\right)}{\partial y_j^{n-1}}. \tag{15}$$

The rule weights for the *n*th (output) layer are set according to (16), for intermediate layers according to (17).

$$w_j^{n_0}(t+1) = w_j^{n_0}(t) + \mu_t \alpha_j^{n_0} \left(y_j^{n_0} - d_j^{n_0} \right) \quad (16)$$

$$w_j^{n-1}(t+1) = \\ = w_j^{n-1}(t) + \mu_t \alpha_j^{n-1} \sum_{y_j^{n-1} \to \Theta_j^n} \frac{\partial \varepsilon_j^n}{\partial y_j^n} \cdot w_j^n \cdot \frac{\partial \Theta_j^n \left(\lambda_j^n \left(y_j^{n-1} \right) \right)}{\partial y_j^{n-1}} \quad (17)$$

For individual rule and link types, substitute expressions for $\frac{\partial \Theta_j^n \left(\lambda_j^n \left(y_j^{n-1} \right) \right)}{\partial y_j^{n-1}}$ were found.

- The AND rule with a positive (non-negated) link to its input variable

$$\frac{\partial \Theta_j^n \left(\lambda_j^n \left(y_j^{n-1} \right) \right)}{\partial y_j^{n-1}} = \prod_{\substack{i=0 \\ i \neq j}}^{N} \left(\lambda_j^n \left(y_j^{n-1} \right) \right) \quad (18)$$

- The AND rule with a negated link to its input variable

$$\frac{\partial \Theta_j^n \left(\lambda_j^n \left(y_j^{n-1} \right) \right)}{\partial y_j^{n-1}} = -\prod_{\substack{i=0 \\ i \neq j}}^{N} \left(\lambda_j^n \left(y_j^{n-1} \right) \right) \quad (19)$$

- The OR rule with a positive link to its input variable

$$\frac{\partial \Theta_j^n \left(\lambda_j^n \left(y_j^{n-1} \right) \right)}{\partial y_j^{n-1}} = 1 - \bigcup_{\substack{i=0 \\ i \neq j}}^{N} \left(\lambda_i^n \left(y_i^{n-1} \right) \right) \quad (20)$$

- The OR rule with a negated link to its input variable

$$\frac{\partial \Theta_j^n \left(\lambda_j^n \left(y_j^{n-1} \right) \right)}{\partial y_j^{n-1}} = \bigcup_{\substack{i=0 \\ i \neq j}}^{N} \left(\lambda_i^n \left(y_i^{n-1} \right) \right) - 1 \quad (21)$$

The adaptation algorithm executes as follows:

1. **Initialize weights**. Set weights $w_1 \cdots w_n$ randomly for all nodes in the knowledge base.
2. **Load pattern**. Pick a previously unused pattern and set it on the net input layer.
3. **Compute error value**. Using (8) compute the error value of each output. Using (10) compute the overall error of the network.
4. **Adapt the weights**. Using (16) compute new weight values for the output layer. Compute new weight values for the intermediate layers according to (17).
5. **Repeat** steps 2 to 4 for each individual pattern in the pattern set.
6. **Exit** if the overall error value is lesser than required or maximum step count is reached.

2.1 RESLA experimental results

The method was tested on a knowledge base executing three logical functions. The output knowledge base nodes Y_{and}, Y_{or}, Y_{xor} modeled three logical functions, each of them having three input arguments X_1, X_2, X_3:

- $Y_{\text{and}} = X_1 \wedge X_2 \wedge X_3$
- $Y_{\text{or}} = X_1 \vee X_2 \vee X_3$
- $Y_{\text{xor}} = \left(\left(\left(\overline{X_1} \wedge X_2 \right) \vee \left(X_1 \wedge \overline{X_2} \right) \right) \wedge X_3 \right) \vee \\ \vee \left(\left(\left(\overline{X_1} \wedge X_2 \right) \vee \left(X_1 \wedge \overline{X_2} \right) \right) \wedge \overline{X_3} \right)$

Figure 1 shows the graphical representation of the experimental knowledge base.

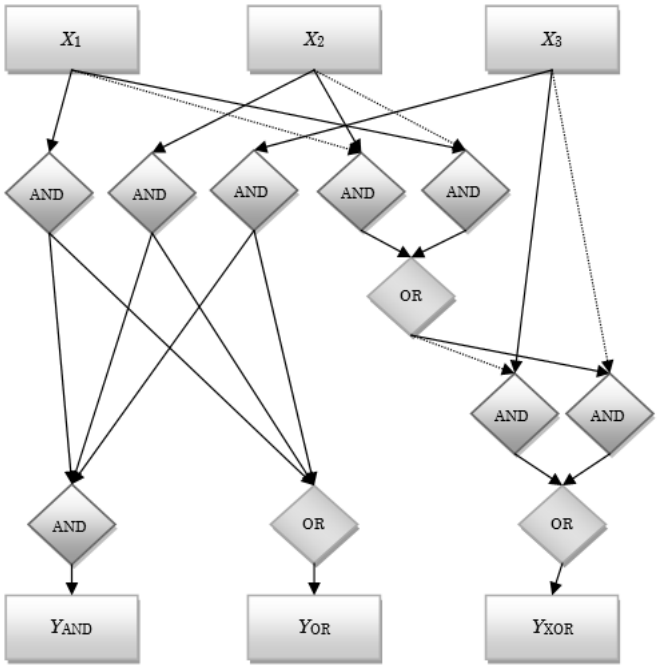

Fig. 1 Knowledge base for Y_{AND}, Y_{OR} and Y_{XOR}

The following figures show the development of the rule weights in the adaptation process.

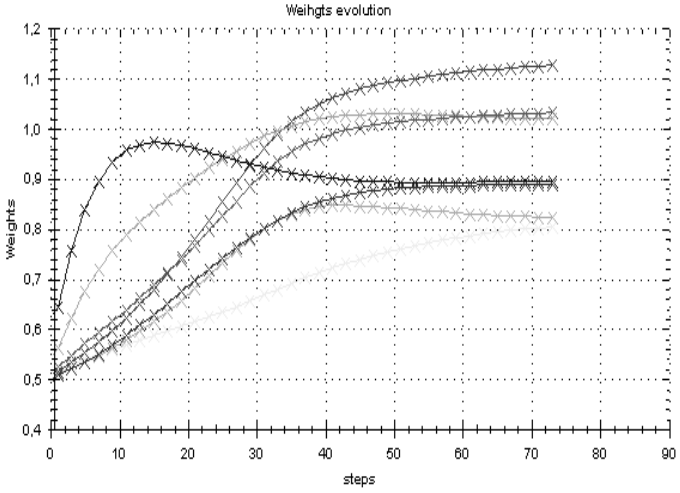

Fig. 2 $\mu_k = 0{,}1$

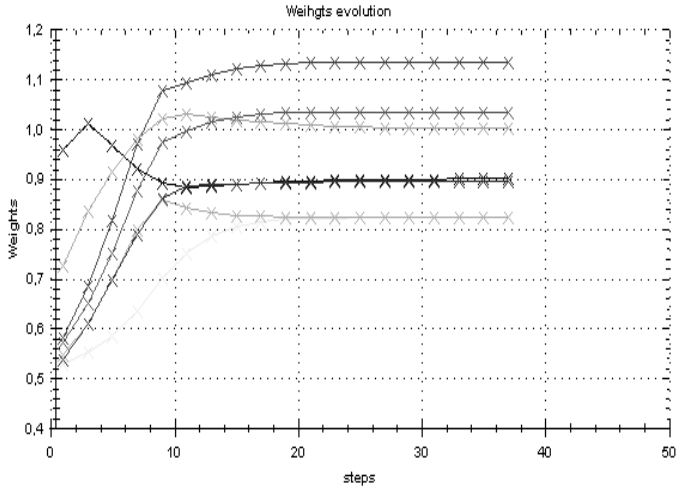

Fig. 3 $\mu_k = 0{,}4$

Figures 4 and 5 compare the development of the global average error for the knowledge base weights.

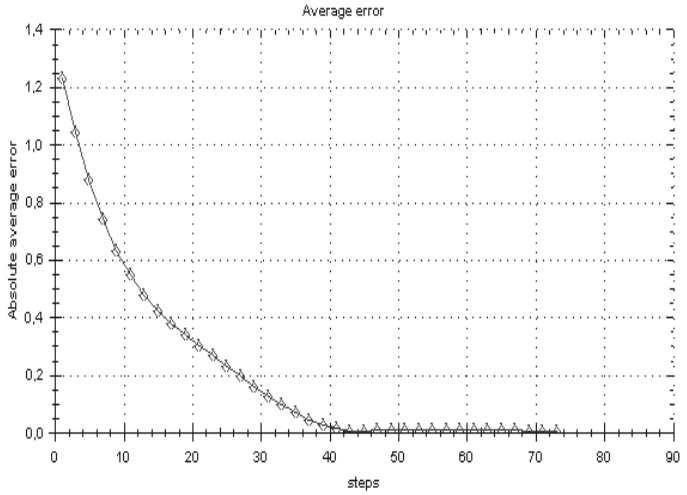

Fig. 4 $\mu_k = 0{,}1$

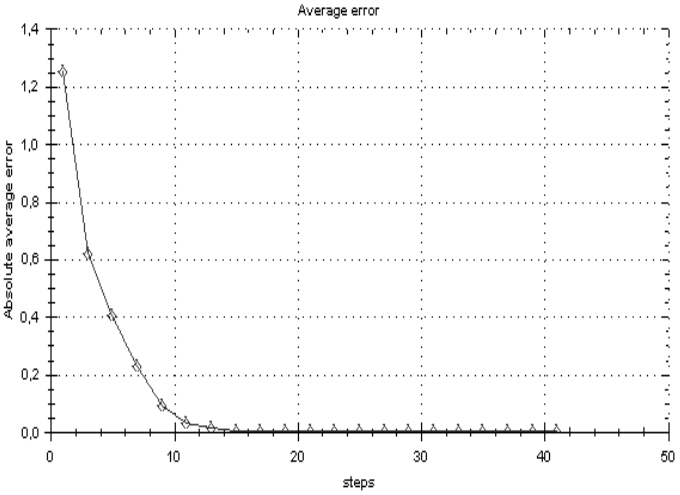

Fig. 5 $\mu_k = 0{,}4$

3 Petri Nets

Petri Nets (PN) are a significant tool for modeling parallel systems and discrete time systems. Thanks to their ability to express parallelism, synchronicity and event causality they are fit for modeling systems such as communication protocols, computer and database systems etc. Petri nets are a class of mathematical models that enable us to describe control flows and information dependencies inside any modeled system [3], [5]. They provide a basis for variant purposes in knowledge engineering: knowledge representation, reasoning engines, knowledge acquisition and verification [13]. Using Petri Nets the structuring of knowledge within the rule bases is achieved. It helps to clearly express the relationships among individual rules thus helping the experts and knowledge engineers in creating and modifying (*tuning*) the knowledge bases. They allow for design of an efficient inference algorithm. In applications where real-time performance is of crucial importance, the concurrency among rules activation (that Petri nets naturally model) may be a great advantage.

3.1 PN definition

The basic elements of any basic or high-level (i.e. *colored*, *timed* or *synchronized* [2]) Petri nets are:

- *Places* (in graphical representation usually denoted by circles) representing conditions
- *Transitions* (rectangles or strokes) representing events
- *Oriented arcs* connecting places and transitions
- *Tokens*

A Petri net is a directed bi-parite graph with two types of vertices – places and transitions. The immediate state

of the system is represented by its *marking*, i.e. the number of tokens in each place.

We describe a Petri Net as a sextuplet N (22)

$$N = (P, T, F, W, K, M_0) , \qquad (22)$$

where P and T are disjoint $T \cap P = \varnothing$ sets of places and transitions respectively. $F \subseteq (P \times T) \cup (T \times P)$ is a binary relation – flow relation of the net N denoting the position and orientation of the arcs. Notation $W : F \to \{1, 2, 3, \ldots\}$ rates the arcs in the graph denoting their *weights*, $K : P \to \{0, 1, 2, 3, \ldots\} \cup \{\infty\}$ specifies the capacities of places and $M_0 : P \to \{0, 1, 2, 3, \ldots\} \cup \{\infty\}$ is the *initial marking* denoting the state of the network before the execution (firing) of any transition, $\forall p \in P : M_0(p) \le K(p)$. All the sets in the sextuplet N in (22) will be considered ordered so that e.g. the *first* place p_1 from the set of all places P has its capacity denoted by the first cell of vector $K(p_1)$.

Similarly, $M_i = (p_{1i}, p_{2i}, p_{3i}, \ldots)$ is a vector of numbers stating the token count in each place of the network in its i-th state. It is the marking of the Petri net in the i-th state.

A Petri net *incidence matrix* A (23) describes the relation between the net's initial state M_0 and its final state M_f by denoting the influence of firing transitions on net places' token counts. A cell a_{xy} in (23) shows the change of number of tokens p_x in place X after executing (firing) the transition t_y.

$$A = \begin{array}{c} \\ p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_n \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & \ldots & \ldots & t_m \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \ldots & \ldots & a_{1m} \\ a_{21} & & & & \vdots \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ a_{n1} & \ldots & \ldots & \ldots & a_{nm} \end{bmatrix} \end{array} \qquad (23)$$

Consider a net shown in Fig. 6. Here $P = \{p_1, p_2, p_3\}$, $K = (\infty, \infty, \infty)$, $T = \{t_1, t_2\}$, $F = \{[p_1, t_1], [p_2, t_1], [p_3, t_2], [t_1, p_3], [t_2, p_1], [t_2, p_2]\}$ $W = (3,1,1,1,2,1)$, $M_0 = (4,1,0)$ and A is (24).

$$A = \begin{bmatrix} -3 & 2 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (24)$$

Figure 6 shows the net in its initial state M_0 and in two subsequent states M_1 and M_2 after firing the transitions t_1 and t_2 respectively. Firing an *enabled* (*executable*) transition carries the network into a new state removing tokens from the transition's input places, adding tokens into the output places. This is expressed as (25) and (26)

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \qquad (25)$$

$$(4,1,0) \xrightarrow{t_1} (1,0,1) \xrightarrow{t_2} (3,1,0) . \qquad (26)$$

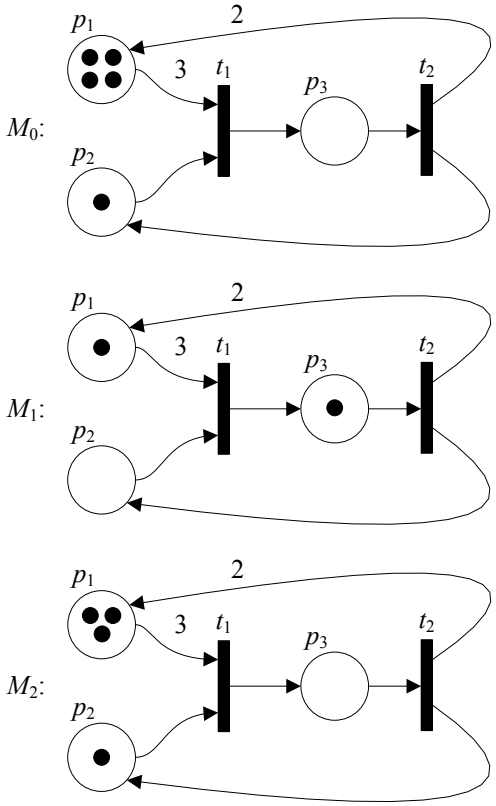

Fig. 6 A simple Petri Net in three consequent states

A transition t is enabled and can be fired when the condition (27) holds true.

$$\forall [p_{\text{in}}, t], [t, p_{\text{out}}] \in F : M(p_{\text{in}}) \ge W([p_{\text{in}}, t]) \wedge \\ \wedge K(p_{\text{out}}) - M(p_{\text{out}}) \ge W([t, p_{\text{out}}]) \qquad (27)$$

Just like in Boolean rule systems, it may be convenient to use a negation of a rule antecedent in Petri net-based system as well. Condition (27) says that a transition is enabled $\Leftrightarrow$ all the input places hold at least the number of tokens denoted by the input arc weight. The 'negation' of a rule antecedent then requires that the place holds fewer tokens than denoted by the input arc weight. In Petri Nets, this is indicated by the presence of

an *inhibitor*. In order to use inhibitors we need to extend the definition of Petri Nets (22) so that all transition input arcs $[p_x, j_y] \in F$ carry the information about the type of arc actually used. We do that by denoting them as triplets $[p_x, j_y, b]$, where $b = 1$ indicates the presence of an inhibitor, while $b = 0$ indicates the presence of a regular arc.

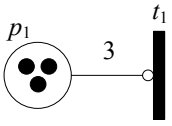

Fig. 7 A transition disabled by an inhibitor

For Petri Nets with inhibitors, condition (27) extends to (28). Fig. 7 shows a part of a Petri Net that is described by the following: $p_1 \in P$, $t_1 \in T$, $[p_1, t_1, 1] \in F$, $M(p_1) = 3$, $W([p_1, t_1, 1]) = 3$.

$$\forall [p_{\text{in}}, t, b], [t, p_{\text{out}}] \in F: \quad \begin{cases} b = 0: & M(p_{\text{in}}) \geq W([p_{\text{in}}, t, b]) \\ b = 1: & M(p_{\text{in}}) < W([p_{\text{in}}, t, b]) \end{cases} \wedge \quad (28)$$
$$\wedge \left(K(p_{\text{out}}) - M(p_{\text{out}}) \geq W([t, p_{\text{out}}]) \right)$$

For the examples presented in this paper we will stick to definition (22) for a Petri Net without inhibitors. Nevertheless, it is trivial to extend our considerations using (28). Note that every state M_i shown in Fig. 8 has only one transition enabled. Note also that, unlike regular arcs, inhibitors do not remove any tokens from the input place when the respective transition is fired $(1,0,1,1) \xrightarrow{t_2} (3,1,0,1) \xrightarrow{t_1} (0,0,1,2)$.

The following sections 4 and 5 deal with usage of Petri Nets in the different cases of diagnosis and planning. The task of a diagnostic expert system is to effectively interpret the data (facts) in order to provide an evaluation of a set of given final hypotheses thus selecting the likeliest, i.e. the one the corresponds the best with the real data (facts). It is the process of re-evaluation of partial and goal hypotheses (stored in the knowledge base as a fixed problem-model) that the Petri Nets may be used for in this case.

On the other hand, by planning we understand solving a problem where there is no set of hypotheses to be evaluated (proven or rejected). Usually, an initial state of a system is given together with a desired goal. The Petri Nets can here be used for finding a sequence of operations taking the system from its initial state to the desired one.

For many applications the modern expert systems don't rely on purely simple Boolean rules. They do handle the uncertainty simple production rules cannot cope with. Among the advantages that support the use of Petri Nets in expert systems (good analysis methods, attractive graphical formalism, structured representation, …) is also the fact that many researchers have dealt with the extension of Petri Nets formalisms in order to incorporate a way to handle the uncertainty directly. Fuzzy Petri Nets were introduced in several suitable modifications [4], [13], [24].

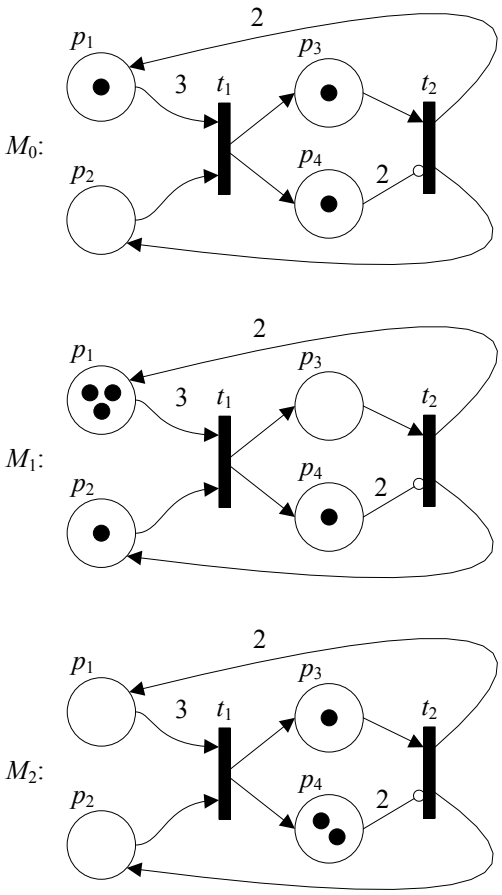

Fig. 8 A Petri Net with an inhibitor

4 Petri Nets in diagnostic expert systems

Here we propose a method for creating diagnostic expert system knowledge bases using a modified Petri net. The task of a diagnostic expert system is, given a set of *goal hypotheses* (i.e. a set of possible diagnosis conclusions), to find the one that is true for the examined problem case (i.e. to find a conclusion that corresponds to the facts ascertained during the consulting session).

Each place in the Petri net models a hypothesis that can be proven (tokens present) or rejected. Firing a transition means validating a rule. We take advantage of

the Petri net capability to express control flows and construct the knowledge bases so that only those rules that directly influence the inference process at a given moment are examined. We achieve that by structuring the knowledge base so that the inference control information is an integral part of rule antecedents. The inference engine (its operation is outlined in Fig. 12) then examines only antecedents (or antecedent sub-trees) of those rules which have previously been activated by the presence of an *inference control token* in one of the input places.

A general production rule `IF` *a* `THEN` *c*, where *a* is the rule *antecedent* and *c* is the rule *consequent*, can be modeled with a Petri net as shown in Fig. 9.

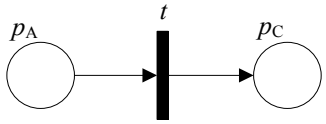

Fig. 9 A simple production rule in PN

For each rule: $P_{\mathrm{A}} = \{p_{\mathrm{A}1},\ldots,p_{\mathrm{A}n}\} \subset P$ is the set of rule antecedents and $P_{\mathrm{C}} = \{p_{\mathrm{C}1},\ldots,p_{\mathrm{C}m}\} \subset P$ is the set of consequents. Validating a rule means firing the transition t (i.e. removing tokens from input places, adding tokens to output places).

Using a weighted Petri net we can model the following types of rules. AND (Fig. 10 left), OR (Fig. 10 right). Clearly, constructing an OR rule means creating a structure of n transitions each with one input place $p_{\mathrm{A}i} \in P_{\mathrm{A}}$ thus creating a sub-net of n AND rules each of them having one antecedent and a full set of consequents P_{C}.

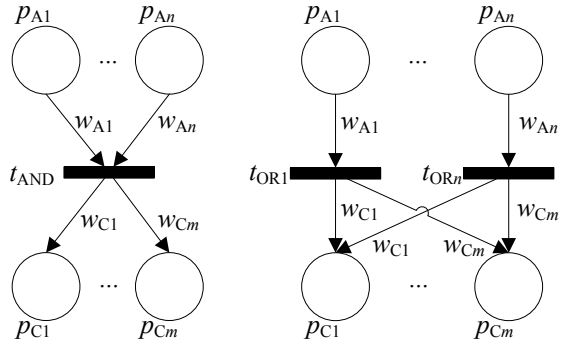

Fig. 10 AND, OR rules in PN

When validating a rule, we follow this procedure (29) for modifying the net's (marking):

$$\begin{aligned}&\textbf{IF}\\&\forall p_{\mathrm{A}i} \in P_{\mathrm{A}} : M(p_{\mathrm{A}i}) \geq w_{\mathrm{A}i}\\&\textbf{THEN}\\&\forall p_{\mathrm{A}i} \in P_{\mathrm{A}} : M(p_{\mathrm{A}i}) = M(p_{\mathrm{A}i}) - w_{\mathrm{A}i}\\&\forall p_{\mathrm{C}j} \in P_{\mathrm{C}} : M(p_{\mathrm{C}j}) = M(p_{\mathrm{C}j}) + w_{\mathrm{C}j}\end{aligned} \tag{29}$$

For OR rules in a net where $\forall p \in P : K(p) = \infty$ this approach leads to accumulating individual antecedent *contributions* to the validation of the consequent – if more than one antecedent hold true, firing all enabled OR rule sub-transitions results into addition of a number of tokens denoted by the sum of weights of arcs connecting all enabled sub-transitions with the respective consequent place. This feature can be used for constructing *weighted OR rules* and can be extended further by introducing limited place capacities and transition priorities.

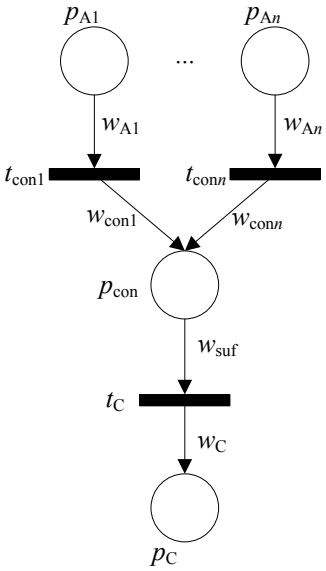

Fig. 11 A weighted OR rule

Consider a rule modeled by the net shown in Fig. 11. Here, the consequent p_{C} is confirmed when all the token count contributions from antecedents holding true collected in intermediate place p_{con} sum up to a count equal or greater than the value of *sufficiency threshold* w_{suf}.

$$p_C \Leftarrow \sum_{i:M(p_{\mathrm{A}i}) \geq w_{\mathrm{A}i}} w_{\mathrm{con}i} \geq w_{\mathrm{suf}} \tag{30}$$

The described structure may be useful when a diagnosis needs to be made based on various *symptoms* or *attributes* and it is known that the symptoms may be present in various combinations (and even various

strengths or severities). Each symptom/attribute that is observed (i.e. the corresponding $M(p_A) > 0$) causes an addition of w_A tokens into the intermediate place (p_{con} in Fig. 11). When enough symptoms are present, i.e. the total number of generated tokens exceeds the sufficiency threshold, the consequent is confirmed.

If it is known that only a certain number of symptoms needs to be present in order to be able to conclude the diagnosis with p_C and it is costly to perform all the tests needed for verifying the validity of all antecedents, we introduce transition priorities so that the least costly verifiable antecedents get examined first. If the rule is constructed so that $\sum_i w_{con/} > w_{suf}$ then the inference engine stops examining the antecedents as soon as $M(p_{con}) \geq w_{suf}$ omitting the verification of antecedents with the lowest priority if higher priority antecedents have been observed as holding true.

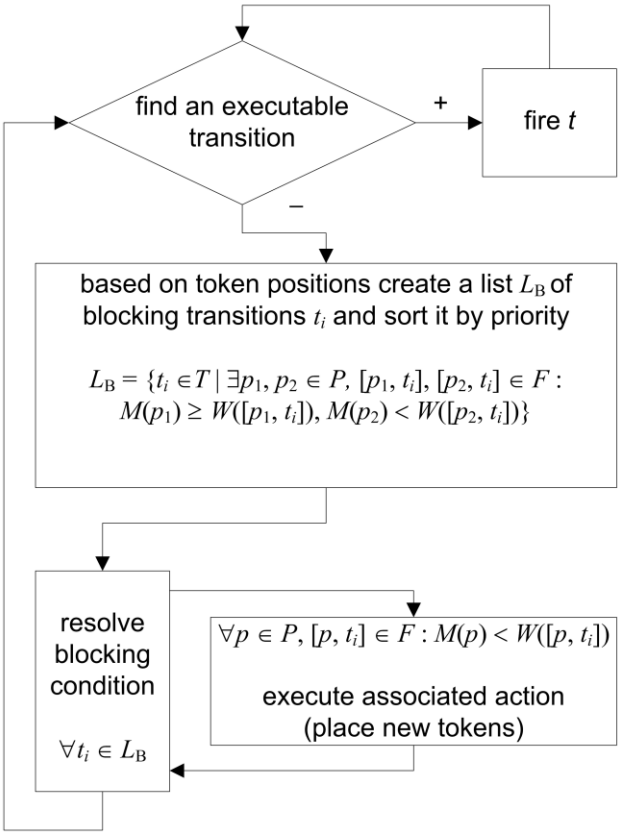

Fig. 12 The inference process during diagnosis

The rule base can easily be extended so that more complex reasoning is possible. If an attribute/symptom must be present in order to be able to conclude with p_C although the presence of the attribute itself is not sufficient enough, we simply create a direct arc connecting the necessary antecedent place and the consequent transition t_C. Similarly, when a symptom (possibly from a different part of the knowledge base) must not be present, we create an inhibitor. Consider the case shown in Fig. 13. Here, p_{A1}, p_{A2}, p_{A3}, p_{A4} are symptoms supporting the diagnosis p_C. p_{A1} must be observed if p_C is to be confirmed. Also p_{A2} must hold true or both of p_{A3} and p_{A4} must hold true. p_{D1} must be rejected.

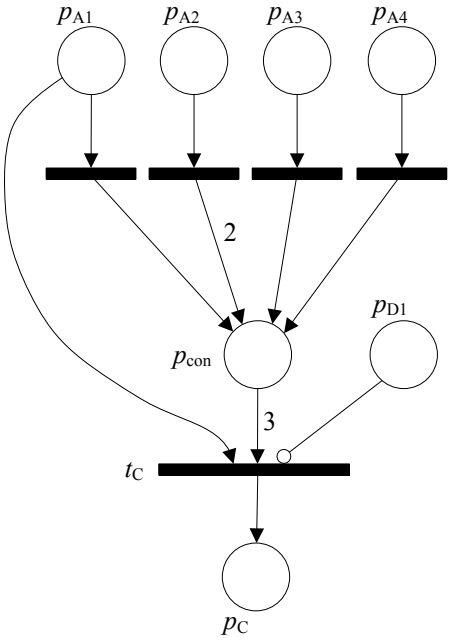

Fig. 13 On the use of a weighted rule

Note that the place p_{A1} represents a fireability condition for two different transitions. It is therefore required that this place has at least as many tokens as the sum of all weights of outgoing arcs when the statement p_{A1} holds true. This can be achieved by setting a correct weight of arcs connecting into this place.

In order to speed up the reasoning process, for each unexamined transition the depth of every antecedent sub-tree is calculated and the shallowest trees (possibly consisting of just one place) are examined first.

4.1 Place associated actions

The expert system requires gathering data (facts) during the inference process so that it can assess rule antecedents. We therefore extend the definition (22) of a Petri Net so that each place can be associated with an action. Among these actions are:

- ask a question,
- query a database,
- collect data from a measuring system,
- etc.

When a certain step in the reasoning process is reached (a place receives tokens) the corresponding action is executed and a list of potentially firable (currently blocked) transitions is created. The transitions from this list are blocked due to the absence of tokens in input places (rule antecedents). The inference engine examines these places executing associated actions. Based on the callback data (e.g. user responds 'yes' and a token is placed) the inference process moves on along the knowledge base network.

4.2 A diagnosis example

In Fig. 14 we demonstrate the use of Petri nets for creating a knowledge base. The aim is to reason about the cause of a TV set malfunction. The knowledge base consists of 5 goal hypotheses and a number of intermediate places used for step-by-step evaluation of the problem case. The possible conclusions (and respective places in the knowledge base) are:

- **screen** is defective (p_{screen})
- **antenna** is not connected (p_{antenna})
- **sound** processing circuits are defectional (p_{sound})
- **power cord** is unplugged (p_{cord})
- **fuse** is blown (p_{fuse})

Among the places are also antecedents of the production rules. In this case, each of the antecedent places has a possible user response associated with it. When the reasoning branches (inference engine examines the blocking conditions as shown in Fig. 12) the user is presented with a question (associated to the inference flow control places) and a list of these responses and is asked to pick one. Based on this pick a token is placed into the respective place thus releasing the blocking condition and enabling the respective transition. The inference flow control places and questions associated with them are:

- p_{q1} : 'Does the TV set work at all when switched on?'
- p_{q2} : 'Is the power cord plugged into the socket properly?'
- p_{q3} : 'Does the screen show anything at all?'
- p_{q4} : 'Do you hear any sound?'

All of these are yes/no questions. The place p_{q1Y} models the user's response 'yes' to the question from p_{q1}. The place p_{q1N} is the 'no' answer. When an answer is picked by the user, the respective place receives a token.

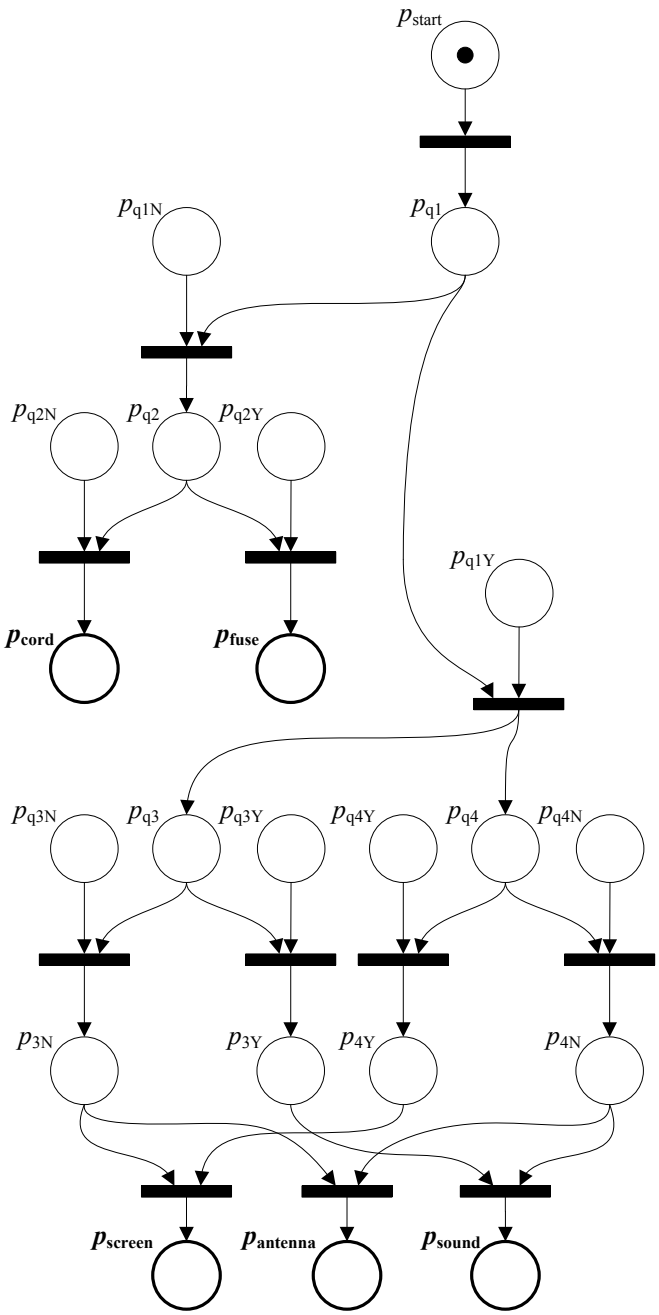

Fig. 14 A knowledge base for TV set malfunction diagnosis

The goal hypotheses places are printed in bold in Fig. 14. The inference process starts by placing a token into the place p_{start} and continues by firing transitions following the rules as discussed in section 3 and 4 and resolving the blocking conditions using the place associated actions.

When a token is present in one of the goal hypothesis places and no more transitions can be fired, the consultation ends.

5 Petri Nets for planning

By planning we understand finding a way (a path) that, when followed, transforms a system from its initial state to a desired state. A path is a sequence of *actions* (applied *operators*). The role of a human expert here is to provide the set of valid operators/actions that can be applied onto the system (any individual part or a group of parts) and specify the conditions under which the operators may be used.

During planning (i.e. state search) we examine the incidence matrix A. Following the rules for Petri nets [2] the algorithm looks for executable transitions and fires them in each 'current' planning step c thus transforming the network into state c+1, i.e. from marking M_c into M_{c+1}. The inference engine compares the resulting state M_{c+1} with the desired final state M_f (ending the process if M_f is reached) and with the members of the set of previously achieved states – the history M_H^* (going back to last branching in order to prevent deadlocks if necessary).

The knowledge base contains a set of transitions representing operations that may be applied on individual parts of the system (places of the Petri Net) in order to achieve the desired state M_f. The system (problem task) itself is modeled as a set of Petri Net places. The expressing capabilities of such a model are given by the type of Petri Net used. In the simplest case of a P/T Petri Net, the only attribute of any part of the system (and the only condition that limits the possibility to use an operation selected from the knowledge base) is a positive integer denoting the count of tokens. I.e. the system is described by the marking M of the net. The employability of operations proposed by an expert in the knowledge base is purely dependant on the rules for firing transitions. The inference is error-trial driven. If a need for modeling more complex systems arises, higher lever Petri Nets (e.g. colored) may be used.

5.1 The knowledge base

In the current version of our expert system, the knowledge base is a plain text file that holds the list of allowed operations. They are, in fact, transitions written in an expected format.

At the start of the inference process (planning), the expert systems parses the knowledge base file and generates a Petri Net from the set of places P describing the examined problem by generating the sets of transitions T and arcs F so that in each planning step, i.e. marking M_c, all operations may be executed (all kinds of transitions may be fired). The knowledge base records determine the number of input places for a transition, the number of output places and individual arc weights. A knowledge base KB is a set of operations OP defined as (31), (32).

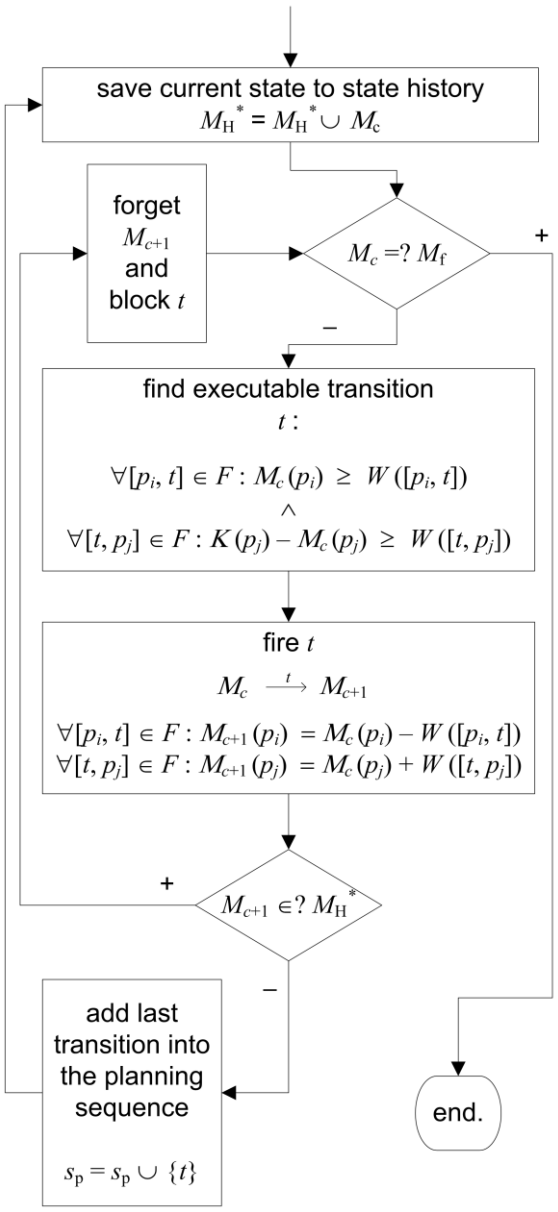

Fig. 15 The inference process during planning

$$OP = (n_{\text{pin}}, n_{\text{pout}}, W_{\text{pin}}, W_{\text{pout}}) \quad (31)$$

$$KB = \{OP_1, \ldots, OP_\varphi\} \quad (32)$$

Where n_{pin} and n_{pout} are the numbers of input and output places respectively, W_{pin} and W_{pout} are the vectors of arc weights, $n_{\text{pin}}, n_{\text{pout}} \in \{1,2,3,\ldots\}$,

$$\left|W_{\text{pin}}\right| = n_{\text{pin}} \ . \quad (33)$$

(33) applies for n_{pout} and W_{pout} similarly. The Petri Net model is generated as follows:

1. load operation $OP \in KB$,
2. given the n_{pin}, generate set IN of all n_{pin}-variations of P,

$$IN = \{[p_{\text{in}1}, \ldots, p_{\text{in}l}] : p_{\text{in}} \in P, p_{\text{in}a} \neq p_{\text{in}b}\} \quad (34)$$

3. given the n_{pout}, generate set OUT of all n_{pout}-variations of P,

$$OUT = \{[p_{\text{out}1}, \ldots, p_{\text{out}m}] : p_{\text{out}} \in P, p_{\text{out}a} \neq p_{\text{out}b}\} \quad (35)$$

4. create a net where each individual n_{pin}-tuplet $[p_{\text{in}1}, \ldots, p_{\text{in}l}] \in IN$ is connected to each n_{pout}-tuplet $[p_{\text{out}1}, \ldots, p_{\text{out}m}] \in OUT$ through a separate transition t_{OP} modeling the operation OP via n_{pin} input arcs with weights W_{pin} and n_{pout} output arcs with weights W_{pout}.

5. repeat for all operations from the knowledge base

It is clear that this 'blind' approach is an open end in the current implementation. It poses a combinatory problem since $|IN| \cdot |OUT|$ transitions must be generated for each operation in the knowledge base. Loading a knowledge base KB generates ω (36) new transitions in the model.

$$\omega = \frac{\rho \cdot |P|! \cdot |P|!}{(|P| - n_{\text{pin}})! \cdot (|P| - n_{\text{pout}})!} \quad (36)$$

For simple reasoning algorithms that work with a model constructed using just low level P/T nets with weighted arcs where there are no other limiting conditions that could drive the process of generating transitions, it is useful to extend (34) and (35) so that only such transitions get generated that can at all be fired during the session. Obviously, the relation between the place capacities and arc weights can be used here. We extend (34) with the condition (37) so that we can write (38) for input places and arcs and similarly (39) for output places and arcs.

$$\begin{aligned} \neg\exists p_{\text{in}} \in [p_{\text{in}1}, \ldots, p_{\text{in}l}] : K(p_{\text{in}a}) < W_{\text{in}}(a) \\ \neg\exists p_{\text{out}} \in [p_{\text{out}1}, \ldots, p_{\text{out}m}] : K(p_{\text{out}b}) < W_{\text{out}}(b) \end{aligned} \quad (37)$$

$$IN = \{[p_{\text{in}1}, \ldots, p_{\text{in}l}] : p_{\text{in}a} \neq p_{\text{in}b}, K(p_{\text{in}c}) \geq W_{\text{in}}(c)\} \quad (38)$$

$$OUT = \{[p_{\text{out}1}, \ldots, p_{\text{out}m}] : p_{\text{out}a} \neq p_{\text{out}b}, K(p_{\text{out}c}) \geq W_{\text{out}}(c)\} \quad (39)$$

Generally, each operation OP can be extended by adding a set of conditions C that limit the generation of transitions e.g. based on properties of places so that $OP = (n_{\text{pin}}, n_{\text{pout}}, W_{\text{pin}}, W_{\text{pout}}, C_{\text{OP}})$ and transition t_{OP} is generated if $C_{\text{OP}} = f_{\text{C}}([p_{\text{in}1}, \ldots, p_{\text{in}l}], [p_{\text{out}1}, \ldots, p_{\text{out}m}])$ holds true. We will demonstrate this on an example in the following section 5.2.

In order to maintain the generality of the inference engine it may be useful to implement the set KB in terms of *function objects* so that the inference engine (of a presumably *empty* expert system) does not need to deal with a complicated proprietary syntax. The knowledge engineer may then as well use the respective standard high level programming language to formulate transition-generation conditions C_{OP} for each operation.

Another way of optimizing the transition generation is regarding the cells of the vectors of arc weights W_{pin}, W_{pout} as functions $w = f_{\text{w}}(p, K(p), M_{\text{c}}(p))$. The feature for blocking the generation of dead transitions (37 to 39) is obviously disabled for these transitions as at the moment of transition generation the generating engine cannon assess the attribute $M_{\text{c}}(p)$. Not only does this approach reduce the number of generated transitions, for some task it may be useful for describing the problem domain in a simpler and readable way.

5.2 A planning problem example

Let us consider the following problem: We are given a set of jars with known capacities and with a known amount of water pre-filled in them. Our task is to fill the jars with a given amount of water. We are not given anything to use for measuring the water volumes, the only information is the full capacity of the jars and the volumes of pre-filled water. At any time we can pour out all the water from any of the jars or can fill up any of the jars from the tap or can pour the water over from one jar to another. For simplicity, we will reduce this task to a specific example. Two jars are given, one with the capacity of 5 liters, 3 liters for the other one. The desired volumes of water left in the jars after the task has been solved are 3 and 2 liters. Jar 1 is empty, Jar 2 is filled with 3 liters.

Considering the previous we can model this problem for solving in our expert system by introducing:

$$\begin{aligned} P &= \left(p_{\text{jar1}}, p_{\text{jar2}}, p_{\text{source}}, p_{\text{sink}}\right) \\ K &= \left(5,3,\infty,\infty\right) \\ M_0 &= \left(0,3,\infty,\infty\right) \\ M_{\text{f}} &= \left(2,3,\infty,\infty\right) \end{aligned} \tag{40}$$

Note that the places p_{source} and p_{sink} exist too. Since the current state of the system is denoted by its marking (section 3) and the observed state of the system of jars is actually given by the number of liters in each jar, it is clear that the number $M(p)$ of tokens in each place corresponds to the amount of water in each respective jar. p_{source} is where we take our water from, p_{sink} is where we dispose of it.

Given the allowed operations from the problem assignment, we could describe the knowledge base as (41).

$$KB = \left\{OP_{\text{PourOut}}, OP_{\text{FillUp}}, OP_{\text{PourOver}}\right\} \tag{41}$$

OP_{PourOut} represent the operation of pouring out all the water from a specified jar, OP_{FillUp} stands for filling the jar up from a water source, OP_{PourOver} represents pouring over a beforehand unspecified amount of water between two jars based on the remainder of water in one jar and the capacity left in the other jar.

$$OP_{\text{PourOut}} = \begin{pmatrix} n_{\text{pin}} = 1, \\ n_{\text{pout}} = 1, \\ W_{\text{pin}} = \left(f_{\text{w1}}\left(M\left(p_{\text{in1}}\right)\right)\right), \\ W_{\text{pout}} = \left(1\right), \\ C_{\text{OP}} = f_{\text{C}}\left(p_{\text{out1}}\right) \end{pmatrix} \tag{42}$$

$$OP_{\text{FillUp}} = \begin{pmatrix} n_{\text{pin}} = 1, \\ n_{\text{pout}} = 1, \\ W_{\text{pin}} = \left(1\right), \\ W_{\text{pout}} = \left(f_{\text{w1}}\left(M\left(p_{\text{out1}}\right), K\left(p_{\text{out1}}\right)\right)\right), \\ C_{\text{OP}} = f_{\text{C}}\left(p_{\text{in1}}\right) \end{pmatrix} \tag{43}$$

$$OP_{\text{PourOver}} = \begin{pmatrix} n_{\text{pin}} = 1, \\ n_{\text{pout}} = 1, \\ W_{\text{pin}} = \left(f_{\text{w1}}\left(M\left(p_{\text{in1}}\right), M\left(p_{\text{out1}}\right), K\left(p_{\text{out1}}\right)\right)\right), \\ W_{\text{pout}} = W_{\text{pin}} \end{pmatrix} \tag{44}$$

We can only fill up the jars from the water source, therefore $f_{\text{C}}\left(p_{\text{in1}}\right) \Leftrightarrow p_{\text{in1}} = p_{\text{source}}$ and we can dispose of the water only in a place 'out of the modeled system', $f_{\text{C}}\left(p_{\text{outl}}\right) \Leftrightarrow p_{\text{outl}} = p_{\text{sink}}$.

The operation OP_{PourOut} can be described as: Take all the water and remove it from the jar. I.e. the current number of tokens needs to be removed. Thus $f_{\text{w1}} = M(p_{\text{in1}})$.

The operation OP_{FillUp} means: Whatever the current volume of water in a respective jar is, fill it up. Thus we add as many tokens as it is needed to use up the whole capacity of the modeling PN place, $f_{\text{w1}} = K(p_{\text{out1}}) - M(p_{\text{out1}})$.

For OP_{PourOver}, $f_{\text{w1}} = \min\left[M(p_{\text{in1}}), K(p_{\text{out2}}) - M(p_{\text{out2}})\right]$, as we can only pour over as much water as it is available in the jar we pour from. At the same time we can only pour as much water as the capacity left in the jar we pour the water to.

Since the state of source and sink places are not significant, we can model both of those as having unlimited capacity as well as unlimited stock of tokens. This way we do not run into any problems and may pick an arbitrary weight for the arcs leading to/from them. Here we set $w = 1$.

Fig. 16 shows the model after knowledge base has been loaded and transitions have been generated – initial state M_0.

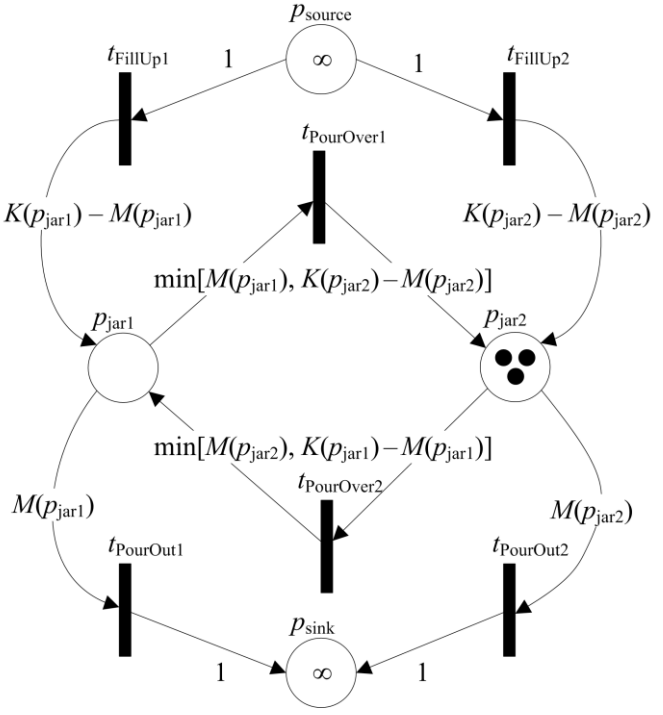

Fig. 16 The inference process during planning

The generated relation F is (45), PN vector W (46) and incidence matrix for this Petri Net is (47).

$$F = \{[p_{\text{jar1}}, t_{\text{PourOut1}}], [t_{\text{PourOut1}}, p_{\text{sink}}], [p_{\text{jar2}}, t_{\text{PourOut2}}], [t_{\text{PourOut2}}, p_{\text{sink}}], [p_{\text{source}}, t_{\text{FillUp1}}], [t_{\text{FillUp1}}, p_{\text{jar1}}], [p_{\text{source}}, t_{\text{FillUp2}}], [t_{\text{FillUp2}}, p_{\text{jar2}}], [p_{\text{jar1}}, t_{\text{PourOver1}}], [t_{\text{PourOver1}}, p_{\text{jar2}}], [p_{\text{jar2}}, t_{\text{PourOver2}}], [t_{\text{PourOver2}}, p_{\text{jar1}}]\} \quad (45)$$

$$\begin{aligned} W &= (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) \\ w_1 &= M(p_{\text{jar1}}), w_3 = M(p_{\text{jar2}}), \\ w_2 &= w_4 = w_5 = w_7 = 1, \\ w_6 &= K(p_{\text{jar1}}) - M(p_{\text{jar1}}), \\ w_8 &= K(p_{\text{jar2}}) - M(p_{\text{jar2}}), \\ w_9 &= w_{10} = \min[M(p_{\text{in1}}), K(p_{\text{out2}}) - M(p_{\text{out2}})] \\ w_{11} &= w_{12} = \min[M(p_{\text{in2}}), K(p_{\text{out1}}) - M(p_{\text{out1}})] \end{aligned} \quad (46)$$

$$A = \begin{bmatrix} -w_1 & 0 & w_6 & 0 & -w_9 & w_{10} \\ 0 & -w_3 & 0 & w_8 & w_{12} & -w_{11} \\ 0 & 0 & -1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (47)$$

5.3 Planning sequence and optimality

From (46) and since K is constant, it is clear that $A = f(M)$. In each step c we:

- compute $A_c = f(M_c)$ as a means to look for possible changes in the states of the system
- build the set $T_{\text{ec}} = \{t_{ec,1}, \ldots, t_{ec,\lambda}\}$ of all transitions that are enabled and their firing may transform the state of the system into one of $|T_{\text{e}}| = \lambda$ future states $M_{c+1,1\ldots\lambda}$ that are collected $M_c \xrightarrow{T_{ec}} \{M_{c+1,1}, \ldots, M_{c+1,\lambda}\}$ in a separate set.
- $\forall M_{c+1,i} \in \{M_{c+1,1}, \ldots, M_{c+1,\lambda}\}$ branch the search and continue in one path until another branch or M_{f} or an previously examined state $M_{\text{h}} \in M_{\text{H}}^*$ is reached.

For each marking M_{h} that is reached during the planning process, the *depth* denoting the number of steps the inference engine needed for reaching M_{h} is also stored. In case a marking M_{h} is reached that had already been reached previously during the planning process, further planning is stopped. Stored and current depth values are compared and if the current depth is lower (i.e. the last sequence leading into the particular state is shorter than the one previously found) then the previously found sequence leading into M_{h} is forgotten.

This behavior leads to finding the shortest sequence of operations for transforming the system from M_0 to M_{f}. Applying other optimality criterions is an open end at the moment. One could replace the 'shortest path' rule by e.g. evaluation of a cost function assigned to each transition.

The result of the planning process is a firing sequence s_{p} denoting the sequence of operations taken from the knowledge base. If no solution was found (i.e. all the possible marking development paths were examined without having reached the M_{f}), then $s_{\text{p}} = \varnothing$.

In the case of the planning problem described in this example, the resulting planning sequence could be (48).

$$s_{\text{p}} : M_0 \xrightarrow{t_{\text{PourOut2}}} M_1 \xrightarrow{t_{\text{FillUp1}}} M_2 \xrightarrow{t_{\text{PourOver1}}} M_{\text{f}} \quad (48)$$

$$\begin{aligned} M_0 &= \{0,3,\infty,\infty\} \\ M_1 &= \{0,0,\infty,\infty\} \\ M_2 &= \{5,0,\infty,\infty\} \\ M_{\text{f}} &= \{2,3,\infty,\infty\} \end{aligned} \quad (49)$$

There are other possible outcomes even for this simple case, the resulting sequence is dependent on the order of operations as they are defined in *KB* for this implementation.

6 Conclusion

In this paper we have proposed a method suitable for automatic tuning of rule weights in network-based knowledge bases for diagnostic expert systems (section 2).

We have also discussed the usage of Petri Nets in both diagnostic and planning expert systems and shown an approach for using them to code knowledge. Reasoning algorithms were described as well. Simple examples were given in sections 4 and 5. The use of Petri nets for diagnostic expert systems does not bring many assets unless high level Petri nets are used. For planning expert systems they seem to be a very promising tool.

Future research will be focused on extending the expert planning Petri net based approach, especially on optimal planning.

Acknowledgement

This work has been supported by the Czech Republic Ministry of Education under the project No. MSM0021630529.

References:

[1] Brachman R. J., Levesque H. J., *Knowledge representation and reasoning*, Morgan Kaufmann, San Francisco, 2004, ISBN 1-55860-932-6

[2] David R., Alla H., *Discrete, Continuous and Hybrid Petri Nets*, Springer – Verlag, Berlin, 2005, 524p, ISBN 3-540-22480-7

[3] Faltus V.: Petri Nets for dynamic crossroad control modeling, *Automatizace*, vol. 02-48, 2005, pp 88 – 91, ISSN 0005-125X

[4] Chen S., Weighted fuzzy reasoning using weighted fuzzy Petri nets, *IEEE Transactions on knowledge and data engineering*, vol. 14, no. 2, IEEE, 2002, pp 386-397, ISSN 1041-4347

[5] Fischer Michael D., *Expert systems and anthropological analysis*, University of Kent, Canterbury UK, 2007, [cited on May 15, 2010], available from <http://lucy.ukc.uk/bicaweb/b4_/expert.html>

[6] Geva S., Wong M.T., Orlovski M., Rule extraction from trained artificial neural network with functional dependency, *First International Conference on Knowledge-based Intelligent Electronic Systems*, Adelaide, Australia, 1997, pp 559-564, ISBN 0-7803-3755-7

[7] Giarratano J., Riley G., *Expert systems – principles and programming (third edition)*, PWS Publishing, Boston, ISBN 0-534-95053-1

[8] Jackson P., *Introduction to Expert Systems*. Pearson Addison Wesley, 1999, ISBN 0-201-87686-8

[9] Jirsík V., Valenta J., Expert system for vehicle selection, *Automatizace*, vol. 5, 2006, ISSN 0005-125X

[10] Kamruzzaman S. M., Islam M., Extracting symbolic rules from artificial neural networks, *Proceedings of world academy of science, engineering and technology*, vol. 10, 2005, ISSN 1307-6884

[11] Kasabov N., *Foundations of neural networks, fuzzy systems and knowledge engineering*, MIT Press, London, 1996, ISBN 0-262-11212-4

[12] Koutsojannis C., Tsimara M., Nabil E. HIROFILOS: A Medical Expert System for Prostate Diseases, *Proceeding of the 7th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*, WSEAS, Wisconsin, USA, 2008, pp 254-259, ISBN 978-960-474-049-9

[13] Lee J., A Fuzzy Petri Net-based expert system and its application to damage assessment of Bridges, *IEEE Transactions on systems, man and cybernetics – Part B*, vol. 29, no. 3, IEEE, 1999, ISSN 1083-4419

[14] Matamaros A. et al., Expert System for the Preeclampsia Prevention Program, *Proceedings of the 4th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*, WSEAS, Wisconsin, USA, 2005, pp 146-149, ISBN 960-8457-38-6

[15] Million P., *Petri nets for expert systems*, thesis, Brno University of Technology, Czech rep., 2010, 74 pgs.

[16] Polách P., Knowledge representation methods for expert systems, *Proceedings of the IEEE Workshop Kraliky 2009*, VUT, Brno, pp 213-216, ISBN 978-80-214-3938-5

[17] Polách P., Valenta J., Jirsík V., Hybrid Expert System Shell, *Proceedings of the 4th European Computing Conference*, WSEAS Press, USA, 2010, pp 148-153, ISBN 978-960-474-178-6

[18] Polách P., Jirsík V., Expert system knowledge base creation and tuning support – NPS32 graphical add-on, *Inteligentní systémy pro praxi*, AD&M, Ostrava, 2007, pp 43-46, ISBN 978-80-239-8245-9

[19] Prapakorn N., Chittayasothorn S., An RDF-based Distributed Expert Systems, *WSEAS Transactions on Computers*, vol. 8, no. 5, WSEAS, USA, 2009, pp 788-798, ISSN 1109-2750

[20] Rattanaprateep C., Chittayasothorn S., A Frame-based Object-Relational Database Expert System Architecture and Implentation, *Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, WSEAS, Wisconsin, USA, 2006, pp 327-332, ISSN 1790-5109

[21] Rogulj N., Papić V., Pleština V., Development of the Expert System for Sport Talents Detection, *Proceedings of the 7th WSEAS International Conference on Automation & Information,* Budapest, Hungary, 2006, pp 7-10, ISBN 960-8457-46-7

[22] Turban E., Aronson J. E., *Decision support systems and intelligent systems (sixth edition)*, Prentice Hall, New Jersey, 2001, ISBN 0-13-089465-6

[23] Said A. F., Rafea A., El-Beltagy S. R., Hassan H., Automatic Generation of Explanation for Expert Systems Implemented with Different Knowledge Representations, *WSEAS Transactions on Systems*, vol. 8, no. 1, WSEAS, USA, 2009, pp 55-64, ISSN 1109-2777

[24] Scarpelli H., Gomide F., Yager R. R., A reasoning algorithm for high-level fuzzy Petri Nets, *IEEE Transactions on fuzzy systems*, vol. 4, no. 3, IEEE 1996, pp 282-294, ISSN 1063-6706

[25] Valenta J., *Automated weight tuning for rule-based knowledge bases*, doctoral thesis, VUT, BRNO, 2009