NNF and NNPrF — Fuzzy Petri Nets Based on Neural Network for Knowledge Representation, Reasoning and Learning

Zhou Yi (周 奕) and Wu ShiLin (吴时霖)

Department of Computer Science, Fudan University, Shanghai 200433

Received December 14, 1994; revised July 7, 1995.

Abstract

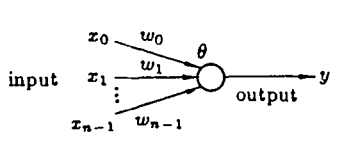
This paper proposes NNF — a fuzzy Petri Net system based on neural network for proposition logic representation, and gives the formal definition of NNF. For the NNF model, forward reasoning algorithm, backward reasoning algorithm and knowledge learning algorithm are discussed based on weight training algorithm of neural network — Back Propagation algorithm. Thus NNF is endowed with the ability of learning a rule. The paper concludes with a discussion on extending NNF to predicate logic, forming NNPrF, and proposing the formal definition and a reasoning algorithm of NNPrF.

Keywords: Fuzzy Petri net system, NNF, NNPrF, neural network, forward reasoning, backward reasoning, learning.

1 Introduction

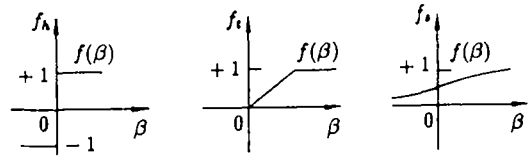
1.1 Basics of Neural Network

A neural network is composed of processing units (corresponding to neurons in human brain) and the connections linking them into a whole. Fig.1 shows the working model of a processing unit. (Note: In Fig.1 x_i is the i-th element of the n-dimension input vector; w_i is the weight of the connection between the i-th input element and the processing unit; θ is the threshold of the processing unit; y is the output of the processing unit.) Fig.2 shows three kinds of usually used function f.



$$y = f(\sum_{i=0}^{n} w_i * x_i - \theta)$$

Fig.1



$$\beta = \sum_{i=0}^{n} w_i * x_i - \theta$$

Fig.2

A lot of neurons are interlinked to make up a neural network. A neural network can work dynamically. To some extent, it is similar to human thinking, and it does

Research supported by the National High-Tech R/D Programme of China.

imitate human brain. A neural network is characterized by its high efficiency, self-organization, fuzzy representation and reasoning, and the ability of self-learning. It has been applied to the domains of pattern recognition, signal processing and knowledge engineering, and has been proved to have good efficiency.

1.2 Shortcomings of the Present Models of Petri Net in Representing Fuzzy Logic

Murata[2] has proposed the ways for representing proposition logic by using P/T nets, representing first-order predicate logic by using Pr/T nets and developed the ways for logic inference. In the reasoning of the P/T net model of proposition logic, if there is a token in a place, then we consider that the statement the place corresponds to is true, else we consider the statement is false. So, given the flowing character of tokens, we can only see that there are no tokens in these places after they have flowed through places. Then we consider the statements corresponding to these places to be false, or we don't know whether the statements are true or false. But actually, these are statements we have conceived as true. This is of course an error in logic. Moreover, the flowing character of tokens brings about the conflict of transition firing. But this conflict doesn't exist in logic inference. So there is inconsistency between proposition logic and P/T net model. This inconsistency also exists between predicate logic and Pr/T net model. And it is brought into the representation of fuzzy logic based on P/T net and Pr/T net models.

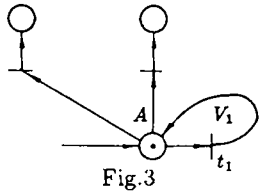


Fig.3

In Fig.3, $v1$ is the belief strength of rule $t1$, $0 \leq v1 \leq 1$, and the truth value of the token in A is x. In the present fuzzy Petri net models, after firing $t1$, $x = x * v1$, then x will become smaller and smaller. This is also an error in logic brought in by the flowing of tokens.

The present fuzzy Petri net models for knowledge representation and reasoning have brought in the concepts of truth values of the conditions and conclusion in a rule and the belief strength of the rules. But when they are to decide which rule will be used, they still insist on that it is only when all of the conditions of a rule are matched that the rule can be used. That is to say, the transition is enabled only when every input place of a transition has tokens. This is a way that equally processes every condition. But in practice, the importance of different conditions of a rule is different. So we shall give different conditions different weights. For example, in the rule "if a thesis demonstrates originality, its proposition is correct, and it is written clearly, then the thesis can be published", it is obvious that the first two conditions are more important than the last one.

So, based on neural networks, we can propose NNF — a new fuzzy Petri net for proposition logic. NNF has token propagation property, and uses weighted fuzzy

logic. From this we extend NNF to first-order predicate logic and form NNPrF — fuzzy Petri net for first-order predicate logic.

2 Concepts, Reasoning and Learning of NNF — A New Fuzzy Petri Net Based on Neural Network for Proposition Logic

2.1 Concepts

2.1.1 Formal Definition of NNF-Net

$\sum = (P, T, F, W, V, \theta, \alpha, Y, M_0)$ is an NNF, iff

(1) (P, T, F) is a finite Petri net. P is a finite set of places, and T is a finite set of transitions. A transition in NNF is something like a neuron in a neural network. F is a finite set of arcs connecting places and transitions. They have the characters of:

$P \cap T = \phi; F \subseteq (P \times T) \cup (T \times P);$

$P \cup T \neq \phi; dom(F) \cup cod(F) = P \cup T,$

where $dom(F) = \{x | y, (x, y) \in F\}$, $cod(F) = \{x | y, (y, x) \in F\}$; $\forall t \in T$, t has only one output place.

(2) $W : F \to [0, 1]$ is an association weight function.
$\forall f \in F$, $W(f) = w_i$, w_i is the weight of arc f, $w_i \in [0, 1]$.

(3) $V : T \to [0, 1]$ is an association function.
$\forall t \in T$, $V(t) = v_t$, v_t is the belief strength of the production rule corresponding to transition t.

(4) $\theta : T \to [0, 1]$ is an association function.
$\forall t \in T$, $\theta(t) = \theta_i$, θ_i is the threshold of transition t.

(5) $\alpha : \forall p \in P$, if there is a token σ in p, then $\alpha(\sigma) = \alpha_i \in [0, 1]$, α_i is the truth value of the token σ. Token σ in p with the truth value of α_i represents that the truth value of the statement corresponding to place p is α_i.

(6) $Y : T \to$ the set of different kinds of output functions, $\forall t \in T$, t is assigned an output function f. Fig.4 shows two kinds of output functions f used in NNF, i.e. f'_t, f'_s.

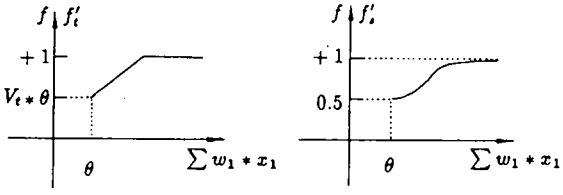


Fig.4. Two kinds of output functions used in NNF.

We introduce two kinds of f in NNF — f'_t, f'_s.

$$f'_t(v_t, \theta, \sum w_i * x_i) = \begin{cases} v_t * \sum w_i * x_i, & 0 < (\sum w_i * x_i - \theta) * v_t \leq 1 \\ 0, & \sum w_i * x_i - \theta \leq 0 \\ 1, & (\sum w_i * x_i - \theta) * v_t > 1 \end{cases}$$

$$f'_s(v_t, \theta, \sum w_i * x_i) = \begin{cases} 1/(1 + e^{-v_t * (\sum w_i * x_i - \theta)}), & \sum w_i * x_i - \theta \geq 0 \\ 0, & \sum w_i * x_i - \theta < 0 \end{cases}$$

(7) $M_0 = (\alpha_0, \ldots, \alpha_i \ldots)$, α_i is the truth value of the token in place P_i. M_0 initializes the distribution of tokens with belief strength in NNF.

2.1.2 Transition Rule

In Fig.5 of transition t, input places are $P_0, \ldots, P_{n-1}$, the truth values of the tokens in them are $x_0, \ldots, x_{n-1}$, the weights of the arcs connecting $P_0, \ldots, P_{n-1}$ to t are $w_0, \ldots, w_{n-1}$, the threshold of t is θ, and the belief strength of t is v_t. The outplace of t is P; the weight of the arc connecting t to p is w.

If $\sum w_i * x_i < \theta$, then t cannot be fired;
else t is fired;

The truth value of the token in P is $y' = \max(y, w * f(\sum w_i * x_i - \theta))$. ($y$ is the truth value of the token in P before firing t. If there is no token in P before firing t, then $y = 0$.) Fig.5 shows the procedure of transition firing.

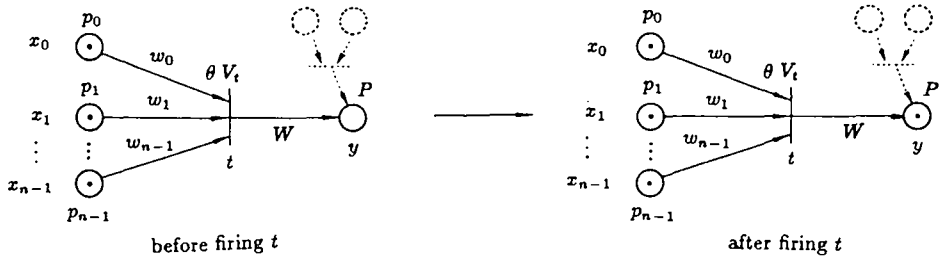


Fig.5

2.1.3 Incidence Matrix

In an NNF-net that has n transitions and m places, its incidence matrix is $C = [C_{ij}]$ $(1 \leq i \leq n, 1 \leq j \leq m)$. Every row of C represents a transition; every column represents a place, $C_{ij} = w(t_i, p_j) - w(p_j, t_i)$.

2.2 NNF Model for Proposition Logic

NNF separates facts from rules. The NNF-net structure only represents rules. In an NNF-net with m places, facts are represented by $M_0 = (\alpha_0, \ldots, \alpha_i, \ldots, \alpha_{m-1})$.

If $\alpha_i = 0$, then there is no token in P_i.

If $0 < \alpha_i \leq 1$, then α_i is the true value of the token in P_i — the truth value of the fact which P_i is corresponding to.

The rules that can be represented in NNF are weighted production rules. Every rule has two parts: conditions and conclusions, which are all weighted conjunctive or disjunctive forms. And we call the two kinds of forms as weighted statement forms. One character of NNF is that it doesn't distinguish weighted conjunctive forms from disjunctive forms, because computations of the truth value of the two kinds of forms are the same. The following is our explanation.

Definition 1. 1. *Every atomic statement P is a weighted statement form, whose truth value is $T(P)$ — the truth value of P.*

2. *Let x be a weighted statement form, then $\sim x$ is also a weighted statement form, which is called "the negation of x", whose truth value is $-T(x)$. So $F(x) = -T(x)$ is called "the false value of x".*

3. *Let $x_1, \ldots, x_n$ be all weighted statement forms and $w_1, \ldots, w_n$ be weights, $\sum w_i = 1$, and $w_i \geq 0$, then*

$$X = w_1x_1 \wedge \cdots \wedge w_nx_n = \wedge_{i=1}^{n} w_ix_i$$

is a weighted statement form, which is called "a weighted conjunctive form of x_is". The truth value of X is $T(X) = \sum w_i * T(x_i)$.

4. *Let x be a weighted statement form, then (x) is also a weighted statement form, which is equal to x. The truth value of (x) is $T((x)) = T(x)$.*

Definition 2. *Let $x_1, \ldots, x_n$ and $w_1, \ldots, w_n$ be the same as Definition 1, then*

$$X = w_1x_1 \vee \cdots \vee w_nx_n = \vee_{i=1}^{n} w_ix_i$$

is "the weighted disjunctive forms of x_is". The false value of X is

$$F(X) = \sum w_i * F(x_i)$$

So, from Definition 1, we can get

$$F(X) = \sum w_i * F(x_i) = \sum w_i * (-T(x_i)) = -\sum w_i * T(x_i) = -T(X)$$

and $T(X) = \sum w_i * T(x_i)$.

From what is discussed above, it can be seen that the computations of the truth value of the weighted conjunctive and disjunctive forms are the same.

In NNF, the only thing that decides whether a transition can be fired is whether the truth value of the condition, a weighted statement form, is larger than θ. So we need only to pay attention to the truth values of conditions and conclusions.

NNF only represents the rules that have the form of:

$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w * A \tag{1}$$

The belief strength of the rule is v_t, the threshold of the rule is θ, and the truth values of conditions $B_0, \ldots, B_{n-1}$ are $x_0, \ldots, x_{n-1}$. Only if $\sum w_i * x_i > \theta$, then we can get conclusion A and the truth value of $A = w * f(v_t, \theta, \sum w_i * x_i)$. (We can select f_t' or f_s' as function f. Fig.4 shows f_t' and f_s'.)

However, we may change any kinds of rules into form (1). For example:
If a rule has the form of:

$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w_1' * A_1 \wedge \cdots \wedge w_n' * A_n,$$

then we can change it into:

$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w_1' * A_1,$$
$$\cdots$$
$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w_n' * A_n.$$

So it can be represented in NNF.
If a rule has the form of:

$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w_1' * A_1 \vee \cdots \vee w_n' * A_n,$$

then we can also change it into:

$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w_1' * A_1,$$
$$\cdots$$
$$w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1} \xrightarrow{v_t,\theta} w_n' * A_n.$$

So it can be represented in NNF.

After that, we don't distinguish the representations of conjunctive forms from those of disjunctive forms in NNF. We use "$\wedge$" to represent conjunction and disjunction.

Fig.6 shows how a rule and the propagation of tokens are represented when the rule is used.

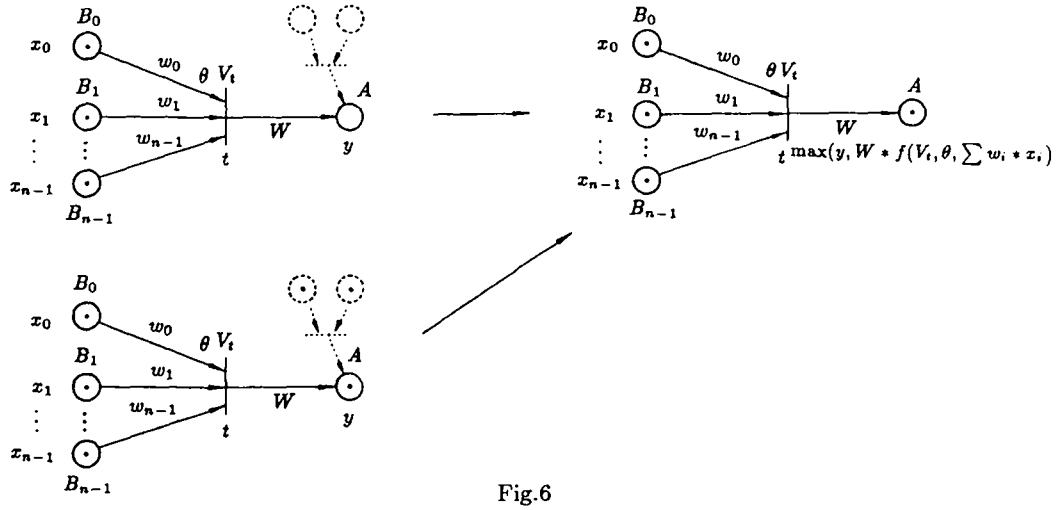


Fig.6

In NNF, we introduce a transition as shown in Fig.7. The transition t represents goal: $w_0 * B_0 \wedge \cdots \wedge w_{n-1} * B_{n-1}$. The truth values of the tokens in $B_0, \ldots, B_{n-1}$ are $x_0, \ldots, x_{n-1}$. If $\sum w_i * x_i > \theta$, then goal transition t can be fired.

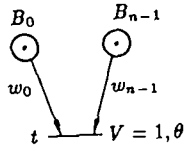


Fig.7

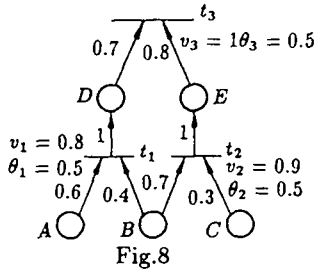


Fig.8

incidence matrix

	A	B	C	D	E	belief strengths	thresholds
$t1$	−0.6	−0.4	0	1	0	0.8	0.5
$t2$	0	−0.7	−0.3	0	1	0.9	0.5
$t3$	0	0	0	−0.7	−0.8	1	0.5
initial state M_0=	(0.6,	0.8	0	0	0.1)		

Fig.9

Example 1. We have a set of weighted production rules:

1) 0.6A, 0.4B $\overset{v=0.8,\theta=0.5}{\longrightarrow}$ D

2) 0.7B, 0.3C $\overset{v=0.9,\theta=0.5}{\longrightarrow}$ C.

The facts are: 0.6A, 0.8B, 0.1E.

The goal is to prove: −0.7D, 0.8E. Let the belief strength of the goal be 1 and the threshold of the goal be 0.5.

The NNF representation of the rules in the example is shown in Fig.8.

The incidence matrix, initial state M_0, belief strengths and thresholds of transitions of the NNF model of the example are shown in Fig.9.

2.3 Reasoning in NNF

The difference between the reasoning used in NNF and that used in the former fuzzy production system is as follows. According to the reasoning of the former fuzzy production system, it is only when all of the conditions of a rule are true that the conclusion of the rule can be true. But according to the reasoning used in NNF, we are not to give up a rule even if one of its conditions is false. We shall, in that case, proceed to process the next condition of the rule. After all the conditions of the rule have been processed, we say: if $\sum w_i * x_i > \theta$ (θ is the threshold of the rule) then the rule is usable; else it is not. So in NNF we can carry on reasoning even when only partial information is available.

2.3.1 Forward Reasoning

NNF is used to represent fuzzy production rules. Facts are separated from rules. They are stored in M_0. When a forward reasoning begins, we put the tokens with truth values stored in M_0 to their corresponding places, then search for transitions that can be fired, and fire them one by one. At the same time, tokens are propagated. The transition firing procedure will not stop, either until the goal transition can be fired (in that case, we say the goal is true), or until the state of NNF cannot be changed with further reasoning, while the goal transition remains unfired (in that case, we say the goal is false).

Algorithm 1. $C_{n\times m}$ is the incidence matrix, the goal transition is corresponding to the n-th row of $C_{n\times m}$; M_0 is the initial state that contains tokens representing facts with truth values, M_0 is an m-dimension vector; Threshold=$[\theta_1, \ldots, \theta_n]$ is the threshold vector of transitions, and $V = [v_1, \ldots, v_n]$ is the belief strength vector of transitions.

$A = C_{n\times m}$; $M_{\text{new}} = M_0$; (see whether goal transition can be fired)
  $X = 0$; conclusion=0; success=false;
  call procedure1(success, n, X, conclusion)
  if success then goto stop;
  MAXStep=MAXNUM; (define the maximum loop number to be MAXStep)
REPEAT
  $M = M_{\text{new}}$; (search for transitions that can be fired, and fire them one by one)
  FOR $i = 1$ to $n - 1$ DO (see whether transition i can be fired)
    $X = 0$; success=false; conclusion=0
    call procedure1(success, i, X, conclusion)
    if success then
M_{new}[conclusion]=MAX(M_{new}[conclusion], A[i, conclusion]$*f(V$[i], threshold[i], X)
    endif (see whether goal transition can be fired)
    $X = 0$; success=false; conclusion=0
    call procedure1(success, n, X, conclusion)
    if success then goto stop;
  ENDFOR
  MAXStep=MAXStep-1;
UNTIL M_{new}=M or Step=0
stop: if success then goal=true
    else goal=false
ENDalgorithm1

procedure1(success, t, X, conclusion) (see whether transition t can be fired)

```
FOR j = 1 to m DO
   if A[t,j] < 0 and Mnew[j] > 0 then X = Mnew[j] * A[t,j] + X
   if A[t,j] > 0 then conclusion=j
ENDFOR
if X >Threshold[t] then success=true;
endprocedure1
```

For example, in applying Algorithm 1 to Example 1, the reasoning can be shown as follows (here we select f_t' as the output function f):

	A	B	C	D	E	belief strengths	thresholds
$t1$	−0.6	−0.4	0	1	0	0.8	0.5
$t2$	0	−0.7	−0.3	0	1	0.9	0.5
$t3$	0	0	0	−0.7	−0.8	1	0.5
M_0=	(0.6	0.8	0	0	0.1)		
			↓ $t1$				
M=	(0.6	0.8	0	0.544	0.1)		
			↓ $t2$				
M=	(0.6	0.8	0	0.544	0.504)		

Given $0.7*0.544+0.8*0.504 > 0.5$, goal transition $t3$ can be fired, and thus we say goal: −0.7D, 0.8E is true.

The proof of the correctness of Algorithm 1:

Algorithm 1 uses forward propagations of tokens to fire the goal transition. There are 3 cases in Algorithm 1:

(1) If the goal is true and can be attained in the defined maximum number of inference steps — MAXStep, then the goal transition can certainly be fired by using Algorithm 1 in NNF. But what is to be noted is that MAXStep should be large enough so that the goal transition can be fired by forward reasoning in MAXStep.

(2) If tokens cannot be propagated further in MAXStep, and the goal transition hasn't been fired, then Algorithm 1 considers that the goal is false. This consideration is right for the goal is actually false.

(3) If the goal transition hasn't been fired after MAXStep forward reasoning, then Algorithm 1 considers that the goal is false.

The definition of MAXStep excludes the endless looping propagation of tokens in the NNF systems as shown in Fig.10.

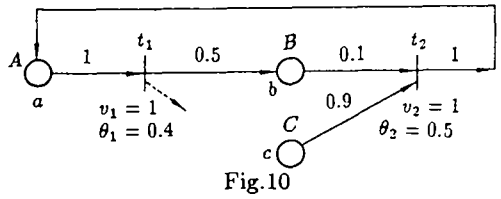


Fig.10

Note: a, b, c are the truth values of tokens in places A, B, C. And before firing $t1$ and $t2$, $a_0 = 0.5, b_0 = 0, c_0 = 1$. We select f_t' as the output function. For the first time, $t1$ is fired, $b_1 = a_0*0.5 = 0.25$, then $t2$ is fired, $a_1 = b_1*0.1+0.9 = 0.925$; for the second time, $t1$ is fired, $b_2 = a_1*0.5 = 0.4625$, then t_2 is fired, $a_2 = b_2*0.1+0.9 = 0.94625$; ...; for the i-th time, $t1$ is fired, $b_i = a_{i-1}*0.5$, then t_2 is fired, $a_i = b_i*0.1+0.9$. So, $a_i = 0.5*0.05^{i-1}+0.9*\sum_{k=0}^{i-2} 0.05^k$.

From what is discussed above, we can see that $t1$ and $t2$ will be fired in endless cycles. Only by defining maximum inference step number, the endless cycles can be cut.

2.3.2 Backward Reasoning

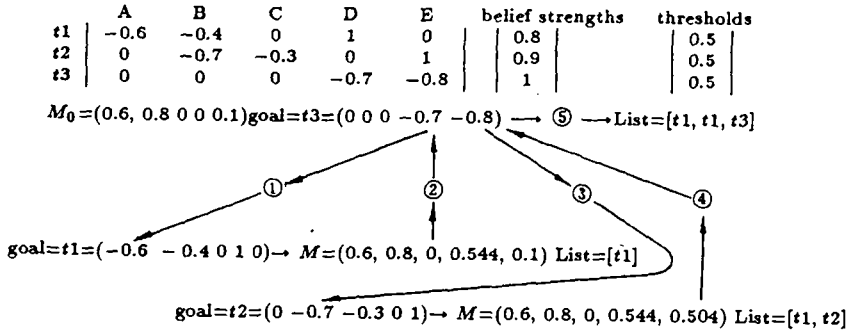
Forward reasoning starts from facts. Backward reasoning starts from the goal. In backward reasoning we divide a goal into some subgoals, then for every subgoal we search for a proving path that can give the subgoal a maximum truth value. Algorithm 2 is a backward reasoning algorithm in NNF. It can get the rules proving the goal and the sequence of the rules used for proving the goal.

Algorithm 2. $C_{n\times m}$ is the incidence matrix, the goal transition is corresponding to the n-th row of $C_{n\times m}$, M_0 is the initial state that contains tokens representing facts with truth values, M_0 is an m-dimension vector, Threshold=$[\theta_1,\ldots,\theta_n]$ is the threshold vector of transitions, and $V=[v_1,\ldots,v_n]$ is the belief strength vector of transitions.

```
A = C_{n×m}; M = M_0; List=Φ; success=false; v = 0;
MAXStep=MAXNUM; (define the maximum inference step number to be MAXStep)
call procedure2(n, v, success, List, M, Step)
if success then "the goal is true"
else "the goal is false"
ENDalgorithm2

Procedure2(goal, v, success, list, current_M, Current_step)
If current_step>0 then
   X = 0;
   FOR j = 1 to m DO
      IF A[goal, j]>0 then conclusion=j
      IF A[goal, j]<0 then
         (search for a rule that can let M[j] get maximum truth value)
         BIG=M[j]; MAXLIST=Φ;
         FOR i = 1 to n − 1 DO
            if A[i, j] > 0 then
               success=false; sublist=Φ; v = 0;
               sub_M=current_M; substep=current_step-1;
               call procedure2(i, v, success, sublist, sub_M, substep)
               if success and v >BIG then
                  BIG=v; MAXLIST=sublist; select_M=sub_M;
               endif
            endif
         ENDFOR
         if MAXLIST≠ Φ then
            current_M[j]=BIG; List=[List|MAXLIST]
         endif
         X = X + BIG * (−A[goal, j]);
   ENDFOR
   IF X >Threshold[goal] then
      success=true;
      v = A[goal, conclusion]*f(V[goal], Threshold[goal], X); List=[List|goal];
   ENDIF
ENDIF
Endprocedure2
```

For example, we apply Algorithm 2 to Example 1. The reasoning is shown as follows (here we select f'_t as the output function f):



Because $0.7*0.544+0.8*0.504 > 0.5$, so goal transition $t3$ can be fired, then we say goal: $-0.7D$, $0.8E$ is true, and the sequence of transitions firing for proving the goal is List=[$t1$, $t2$, $t3$].

The proof of the correctness of Algorithm 2:

Since we have defined the maximum inference step number, Algorithm 2 always can finish.

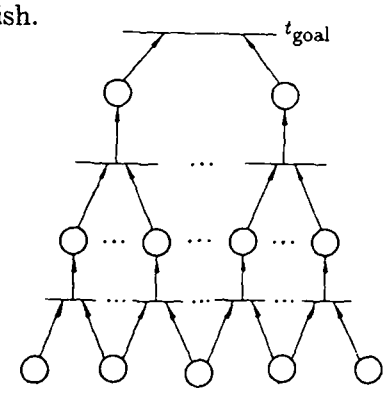


Fig.11

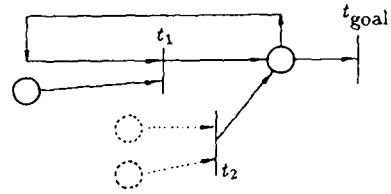


Fig.12

Note: In the case of Fig.12, endless looping will happen in $t1$. When the loop number has reached MAXStep, Algorithm 2 will give up $t1$, and try to get solution by $t2$.

In the cases as shown in Fig.11, it is only when MAXStep is large enough that Algorithm 2 can certainly finish successfully. That is to say, if the goal is actually true, then Algorithm 2 can fire the goal transition and find the best solution. But if the goal is actually false, then the goal transition cannot be fired and Algorithm 2 will report "the goal is false".

In the cases as shown in Fig.12, if MAXStep isn't defined, then backward reasoning will proceed endlessly. In Algorithm 2, the definition of MAXStep prevents proceeding with the endless looping road, and Algorithm 2 will try other solution roads.

2.3.3 Learning in NNF

Since NNF has some characters of neural network, NNF can make use of the ways of weights training common to neural networks and acquire knowledge by modifying the weights of the arcs from places to transitions.

In neural networks, a three-layer perceptron net can be used to realize an m-class

classifier as Fig.13 shows.

$y_l = f(\sum_{k=0}^{n_2-1} w''_{kl} * x''_k - \theta''_l)$, $0 \le l \le m-1$, $x''_k = f(\sum_{j=0}^{n_1-1} w'_{jk} * x'_j - \theta'_k)$, $0 \le k \le n_2 - 1$

$x'_j = f(\sum_{i=0}^{n-1} w_{ij} * x_i - \theta_j)$, $0 \le j \le n_1 - 1$, f_s is selected as output function f. $x_0, \ldots, x_{n-1}$ are n elements of input sample X, while $y_0, \ldots, y_{m-1}$ represent m classes.

In neural networks, the famous Back-Propagation algorithm is used in three-layer perceptron net to acquire knowledge by weights training. In BP algorithm, the difference between desired output and actual output is used to adjust, in the first place, the connection weights between the output layer and the hidden layer right below it, then is propagated backward in the network to layers below the top hidden layer, and is ultimately used to adjust connection weights between the input units and units above them.

In NNF, a transition acts something like a processing unit in neural network. We create an NNF model of perceptron to realize classifier as Fig.14 shows.

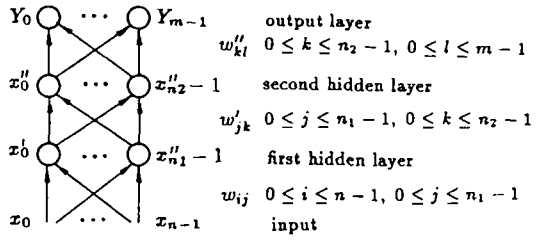


Fig.13

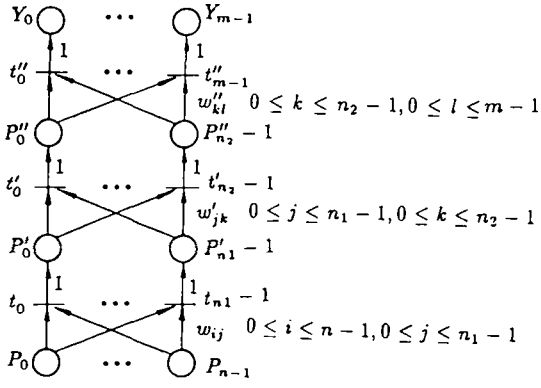


Fig.14

Note: $P_0, \ldots, P_{n-1}$ are input places that represent the characters of input sample. The weight of the arc from a transition to a place is 1. f'_s is selected as the output function. $Y_0, \ldots, Y_{m-1}$ are output places that represent classes. The desired output is that there is only one token in $Y_0, \ldots, Y_{m-1}$. If the token is in Y_i, then the input sample belongs to class i.

First, we put the tokens with truth values that represent the characters of input sample into corresponding places: $P_0, \ldots, P_{n-1}$. Then the truth values of tokens in $P_0, \ldots, P_{n-1}$ are $x_0, \ldots, x_{n-1}$. After that, we apply the forward reasoning algorithm of NNF to the perceptron shown in Fig.14. Then we can get the following.

The truth value of the token in P'_j is

$$x'_j = f'_s(v_j, \theta_j, \textstyle\sum_{i=0}^{n-1} w_{ij} * x_i) \ (0 \le i \le n-1,\ 0 \le j \le n_1 - 1)$$

w_{ij} is the weight of the arc from P_i to t_j. θ_j is the threshold of transition t_j. v_j is the belief strength of t_j.

The truth value of the token in P''_k is

$$x''_k = f'_s(v'_k, \theta'_k, \textstyle\sum_{i=0}^{n_1-1} w'_{jk} * x'_j) \ (0 \le j \le n_1 - 1,\ 0 \le k \le n_2 - 1)$$

w'_{jk} is the weight of the arc from P'_j to t'_k. θ'_k is the threshold of transition t'_k. v'_k is the belief strength of t'_k.

The truth value of the token in Y_l is

$$y_l = f'_s(v''_l, \theta''_l \sum_{i=0}^{n_2-1} w''_{kl} * x''_k) \ (0 \le k \le n_2 - 1,\ 0 \le l \le m - 1)$$

w''_{kl} is the weight of the arc from P''_k to t''_l. θ''_l is the threshold of transition t''_l. v''_l is the belief strength of t''_l.

We can change BP-algorithm of neural network, and apply it to NNF to acquire knowledge by weights training. Algorithm 3 is the BP-algorithm of NNF.

Algorithm 3.

Step 1: Set the thresholds of transitions to small random values in [0, 0.5]; set the weights of arcs from places to transitions to random small values in [0, 1].

Step 2: Input the tokens with truth values that represent the characters of input sample into corresponding places: $P_0, \ldots, P_{n-1}$, and we have already known the desired output: $d_0, \ldots, d_{m-1}$ — the truth values of tokens in $Y_0, \ldots, Y_{m-1}$.

Step 3: Forward reasoning, thus we can get the actual output: $y_0, \ldots, y_{m-1}$ — the truth values of tokens in $Y_0, \ldots, Y_{m-1}$ propagated by forward reasoning.

Step 4: Adjust weights of the arcs from places to transitions. Adjusting weights starts from $Y_0, \ldots, Y_{m-1}$, propagates backward to $P_0, \ldots, P_{n-1}$. That is, from top layer to bottem layer, adjust weights of the arcs from places to transitions.

The weight w_{ij} (the weight of the arc from place i to transition j) at the next time point is $w_{ij}(t+1) = w_{ij}(t) + \eta * \delta_j * x'_i$. j is the sequence-number of the transition j; x'_i is the truth value of the token in place i, x'_i can be the initial input or the result of forward reasoning; η is the gain item, $\eta \in [0, 1]$; δ_j is the error item.

① If j is one of the sequence-numbers of transitions $t''_0, \ldots, t''_{m-1}$, then $\delta_j = y_j * (1 - y_j) * (d_j - y_j)$,

② If j is one of the sequence-numbers of transitions $t'_0, \ldots, t'_{n_2} - 1$ or $t_0, \ldots, t_{n_1} - 1$, then $\delta_j = x'_j * (1 - x'_j) * (\sum_k \delta_k * w_{jk})$.

Note: x'_j is the truth value of the token in the output place of transition j. k ranges over all of the transitions above transition j which the output place of transition j connects to; w_{jk} is the weight of the arc from the output place of transition j to transition k.

Moreover, if we introduce momentum item to BP-algorithm, then the speed of weights training will be accelerated.

So, $w_{ij}(t+1) = w_{ij}(t) + \eta * \delta j * x'_i + \alpha * (w_{ij}(t) - w_i j(t-1))$ $(0 < \alpha < 1)$

Step 5: If the algorithm has reached the desired error precision or has cycled for limited times, then exit; else goto Step 2.

The proof of correctness:

Algorithm 3 is obtained by revising the famous BP-algorithm in neural network to make it runable in the NNF model. So if BP-algorithm can succeed, then Algorithm 3 can succeed too. And we all know that BP-algorithm is successful.

After training weights by using Algorithm 3, we can delete the arcs whose weights are very small from NNF model. Then the places will not connect to all the transitions above them, and the places will connect only to certain transitions above them where the places, a transition above them, and the output place of the transition can form a rule, the transition represents a rule, and the input places of the transition represent conditions of the rule, and the outplace of the transition represents the conclusion of the rule. So, by weights training, NNF can learn the rules of classification.

3 NNPrF

3.1 Concepts

3.1.1 The Formal Definition of NNPrF

$\sum = (P, T, F, W, D, V, \pi, A_P, A_T, A_F, BS, \theta, \alpha, Y, M_0)$ is an NNPrF, iff

(1) P is a finite set of places, T is a finite set of transitions. A transition in NNF is something like a neuron in a neural network, F is a finite set of arcs connecting between places and transitions. They have characters of:

$$P \cap T = \phi;\ F \subseteq (P \times T) \cup (T \times P)$$

$P \cup T \neq \phi$; $dom(F) \cup cod(F) = P \cup T$, $\forall t \in T$, t has only one output place.

(2) $W: F \to [0,1]$ is an association weight function.

$\forall f \in F$, $W(f) = w_i$, w_i is the weight of arc f. $w_i \in [0,1]$.

(3) D: the individual set of NNPrF. Ω is the given set of operators on D.

(4) V: the set of variables of D.

(5) π: the set of dynamic predicates on D.

(6) A_P: $P \to \pi$. $\forall p \in P$, if $A_P(p)$ is an n-ary predicate, then p is called an n-ary predicate.

(7) $A_T: T \to f_D$, f_D is the formula set on D.

$\forall t \in T$, $A_T(t)$ can only be a static predicate or an operator in Ω.

(8) $A_F: F \to S_s$, S_s is the symbolic sum on D.

$\forall p \in P$, if $(t,p) \in F$ or $(p,t) \in F$, then $A_F(t,p)$ or $A_F(p,t)$ is an n-ary symbolic; otherwise $A_F(t,p)$ or $A_F(p,t)$ equals zero.

(9) $BS: T \to [0,1]$ is an association function.

$\forall t \in T$, $BS(t) = v_t$, v_t is the belief strength of the production rule corresponding to transition t.

(10) $\theta: T \to [0,1]$ is an association function.

$\forall t \in T$, $\theta(t) = \theta_i$, θ_i is the threshold of transition t.

(11) α: $\forall p \in P$, if there is a token σ in p, then $\alpha(\sigma) = \alpha_i \in [0,1]$, α_i is the truth value of σ. If token σ has the structure of $(u_1, \ldots, u_m)$ and the truth value of α_i, then token σ in place p represents that the truth value of $A_P(p)$, the n-ary predicate corresponding to p with the parameter vector of $(u_1, \ldots, u_m)$, is α_i.

(12) $Y: T \to$ the set of different kinds of output functions, $\forall t \in T$, t is assigned an output function f. The kinds of output functions used in NNF can be used in NNPrF too. They are shown in Fig.4.

(13) $M_0 = (\ldots, \{\alpha j_i(aj_{i1}, \ldots, aj_{in_j}) | (aj_{i1}, \ldots, aj_{in_j})$ is a token in $P_j\} \ldots)$, $0 < \alpha j_i \leq 1$, αj_i is the true value of $A_p(P_j)(aj_{i1}, \ldots, aj_{in_j})$.

M_0 initializes the distribution of tokens with belief strength in NNF.

Note: A nonempty and finite set D, a nonempty and finite set V have the following relations.

① If symbols in V denote members in D, then we call V the variable set on D, and call the symbols in V the variables on D.

② The variables on D or members in D are called terms on D. If $g^{(n)}$ is an n-ary operator on D, $v_1, v_2, \ldots, v_n$ are terms on D, then $g^{(n)}(v_1, v_2, \ldots, v_n)$ is a term on D. There is no other term on D.

③ An n-ary vector $(v_1, \ldots, v_n)$ $(n \geq 1)$ is called an n-tuple on D if each of its components is a term on D.

④ The formal sum which is formed by the connection of finite number of n-tuples on D with "+" is called the n-ary symbolic sum on D.

⑤ The formulas on D include: $v_1 = v_2$, where v_1 and v_2 are terms on D; $\neg p$, where p is a formula on D; $p \vee q$, where p and q are formulas on D; $(\exists x)p$, where x is a variable on D, and p is a formula on D. There is no other formula on D.

⑥ $p(D) = \{(d_1, \ldots, d_n) | p(d_1, \ldots, d_n)\}$. If $p(D)$ is static, then we call p an n-ary static predicate on D; otherwise, we call p an n-ary dynamic predicate on D.

3.1.2 Transition Rule

In Fig.15, transition is t, input places are $P_1, \ldots, P_n$, and the truth values of the tokens in them are $x_1, \ldots, x_n$, the weights and the symbolic sums of the arcs connecting $P_1, \ldots, P_n$ to t are $w_1, \ldots, w_n$ and $(x1_1, x1_2, \ldots, x1_{m1}), \ldots, (xn_1, \ldots, xn_{mn})$, the threshold of t is θ, the belief strength of t is v_t. The outplace of t is P, the weight and the symbolic sum of the arc connecting t to P is w and $(y_1, \ldots, y_l)$.

If $\sum w_i * x_i \geq \theta$ and $(x1_1, x1_2, \ldots, x1_{m1}), \ldots, (xn_1, \ldots, xn_{mn}), (y_1, \ldots, y_l)$ is unifiable

then t is fired. Apply the unification to the variables, and get the token $(u_1, \ldots, u_l)$ which should be put into the output place P; the truth value of the token $(u_1, \ldots, u_l)$ in P is $y' = \max(y, w * f(v_t, \theta, \sum w_i * x_i))$. ($y$ is the truth value of token $(u_1, \ldots, u_l)$ in P before firing t. If there is no token $(u_1, \ldots, u_l)$ in P before firing t, then $y = 0$.)

else t cannot be fired.

Fig.15 shows the procedure of transition firing.

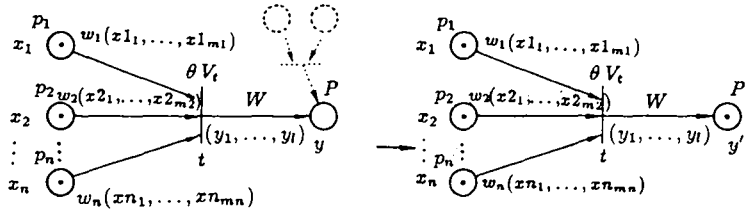


Fig.15

3.1.3 Incidence Matrix

In an NNPrF-net that has n transitions and m places, its incidence matrix is $C = [C_{ij}]$ $(1 \leq i \leq n, 1 \leq j \leq m)$. Every row of C represents a transition, every column represents a place, $C_{ij} = w(t_i, p_j) * A_F(t_i, p_j) - w(p_j, t_i) * A_F(p_j, t_i)$.

3.2 NNPrF Model for Predicate Logic

NNPrF separates facts from rules. The NNPrF-net structure represents rules only. In an NNPrF-net that has m places, facts are represented by $M_0 = (\{\alpha a1_i(a1_{i1}, \ldots, a1_{in1}) | (a1_{i1}, \ldots, a1_{in1})$ is a token in $P_1\}, \ldots, \{\alpha j_i(aj_{i1}, \ldots, aj_{inj}) | (aj_{i1}, \ldots, aj_{inj})$ is a token in $P_j\} \ldots, \{\alpha m_i(am_{i1}, \ldots, am_{inm}) | (am_{i1}, \ldots, am_{inm})$ is a token in $P_m\})$, $0 < \alpha \leq 1$. α is the truth value of its corresponding token — the truth value of a fact.

As in NNF, the rules that can be represented in NNPrF have the form of:

$$w_1 * P_1(x1_1, \ldots, x1_{m1}) \wedge \ldots \wedge w_n * P_n(xn_1, \ldots, xn_{mn}) \xrightarrow{v_t, \theta} w * P(y_1, \ldots, y_l)$$

(v_t and θ are the belief strength and threshold of the rule). The NNPrF model of a rule is shown in Fig.16.

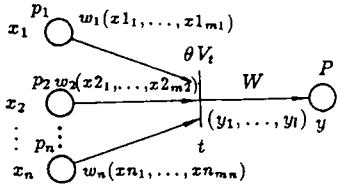


Fig.16

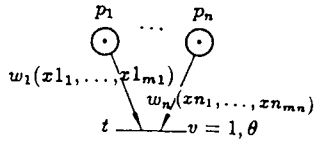


Fig.17

As in NNF, a transition is introduced in Fig.17 to represent goal: $w_1 * P_1(x1_1,\ldots, x1_{m1}) \wedge \cdots \wedge w_n * P_n(xn_1,\ldots,xn_{mn})$. The truth values of the tokens in $P_1,\ldots,P_n$ are $x_1,\ldots,x_n$. If $\sum w_i * x_i > \theta$, then goal transition t can be fired.

Example 2. We have a set of weighted production rules:

1) $0.3 * DOG(x)$, $0.4 * BARK(x)$, $0.3 * JUMP(x) \xrightarrow{v=0.9,\theta=0.5}$ Terrible(x);

2) $1.0 * At(P,w) \xrightarrow{v=0.9,\theta=0.5} At(F,w)$;

3) $0.4 * At(z,A)$, $0.4Man(z) \xrightarrow{v=0.8,\theta=0.5}$ Friendly(z).

The facts are: $1.0{*}DOG(F)$, $0.8{*}BARK(F)$, $0.8{*}JUMP(F)$, $0.9{*}At(P,A)$, $1.0{*}Man(A)$. The goal is to ask: -0.45Terrible(y), $0.45At(y,A)$. Let the belief strength of the goal be 1 and the threshold of the goal be 0.5.

The NNPrF representation of the rules of the example is shown in Fig.18.

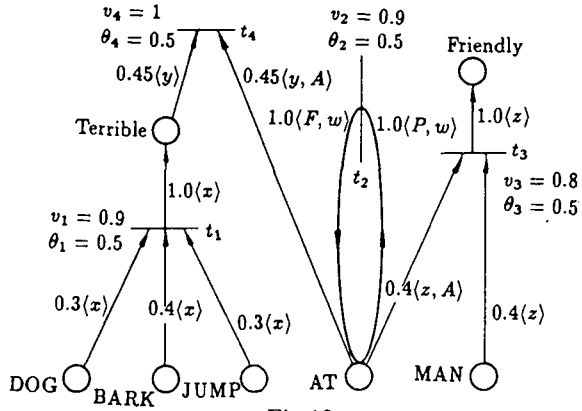


Fig.18

The incidence matrix, initial state M_0, belief strengths and thresholds of transitions of the NNF model of the example are shown in Fig.19.

	Dog	Bark	Jump	Terrible	At	Man	Friendly	Belief strengths	Thresholds
t1	$-0.3\langle x\rangle$	$-0.4\langle x\rangle$	$-0.3\langle x\rangle$	$\langle x\rangle$	0	0	0	0.9	0.5
t2	0	0	0	0	$\langle F, w\rangle - 1.0\langle P, w\rangle$	0	0	0.9	0.5
t3	0	0	0	0	$-0.4\langle x, A\rangle$	$-0.4\langle x\rangle$	$\langle x\rangle$	0.8	0.5
t4	0	0	0	$-0.45\langle y\rangle$	$-0.45\langle y, A\rangle$	0	0	1.0	0.5
M_0	$\{1.0\langle F\rangle\}$	$\{0.8\langle F\rangle\}$	$\{0.8\langle F\rangle\}$	Φ	$\{0.9\langle P, A\rangle\}$	$\{1.0\langle A\rangle\}$	Φ)		

Fig.19

3.3 Forward Reasoning in NNPrF

It differs from NNF in that, in NNPrF, every place probably contains several different tokens, and when firing transitions, we must consider the unification of

variables.

When forward reasoning begins, we put the tokens with truth values stored in M_0 to their corresponding places, then search for transitions that can be fired, and fire them one by one. At the same time tokens are propagated. The transition firing procedure will be stopped, either when the goal transition can be fired in that case, we say the goal is true, or when the state of NNPrF cannot be changed with further reasoning, while the goal transition still cannot be fired, in that case, we say the goal is false.

Algorithm 1′. $C_{n\times m}$ is the incidence matrix, $C = [C_{ij}]$ $(1 \le i \le n,\ 1 \le j \le m)$, $C_{ij} = w(t_i, p_j) * A_F(t_i, p_j) - w(p_j, t_i) * A_F(p_j, t_i)$, the goal transition is corresponding to the n-th row of $C_{n\times m}$. M_0 is the initial state that contains tokens representing facts with truth values. M_0 is an m-dimension vector. Threshold=$[\theta_1, \ldots, \theta_n]$ is the threshold vector of transitions. $V = [v_1, \ldots, v_n]$ is the belief strength vector of transitions.

$A = C_{n\times m}$; $M_{\text{new}} = M_0$; (see whether goal transition can be fired)
  $X = 0$; conclusion=0; success=false; $U = \Phi$;
  call procedure1′(success, n, X, conclusion, U)
  if success then goto stop;
  MAXStep=MAXNUM; (define the maximum loop number to be MAXStep)
REPEAT
  $M = M_{\text{new}}$; (search for transitions that can be fired, and fire them one by one)
  FOR $i = 1$ to $n - 1$ DO (see whether transition i can be fired)
    $X = 0$; success=false; conclusion=0; $U = \Phi$;
    call procedure1′(success, i, X, conclusion, U)
    if success then token $\delta_{\text{output}} = A_F(t_i, p_{\text{conclusion}}) * U$;
the belief strength of δ_{output} is $BS = W(t_i, p_{\text{conclusion}}) * f(V[i], \text{threshold}[i], X)$;
      if M_{new}[conclusion] has already contained δ_{output} then
      the belief strength of δ_{output} in M_{new}[conclusion]
        =MAX(the old belief strength of δ_{output}, BS);
      else
        put $BS * \delta_{\text{output}}$ into M_{new}[conclusion]
      endif
    endif (see whether goal transition can be fired)
    $X = 0$; success=false; $U = \Phi$;
    call procedure1′(success, n, X, conclusion, U)
    if success then goto stop;
  ENDFOR
  MAXStep=MAXStep−1
UNTIL $M_{\text{new}} = M$ or MAXStep=0
stop: if success then goal=true
    else goal=false
ENDalgorithm1′

procedure1′(success, g, X, conclusion, U) (see whether transition g can be fired)
  FOR $j = 1$ to m DO
    if $A_F(p_j, t_g)$ in $A[g, j]$ is unifiable with a token δ in $M_{\text{new}}[j]$ then
      the given belief strength of δ is x_j;
      $X = x_j * W(p_j, t_g) + X$; $U = U \cup$the unifier of $A_F(p_j, t_g)$ and δ;
    if $W(t_g, p_j)$ in $A[g, j] \neq 0$ then conclusion=j
  ENDFOR
  if $X >$Threshold[g] then success=true;
endprocedure1′

For example, we apply Algorithm 1′ to Example 2, the reasoning is shown as follows (here we select f'_t as the output function f):

	Dog	Bark	Jump	Terrible	At	Man	Friendly	Belief strengths	Thresholds
t1	$-0.3(x)$	$-0.4(x)$	$-0.3(x)$	(x)	0	0	0	0.9	0.5
t2	0	0	0	0	$(F,w)-1.0(P,w)$	0	0	0.9	0.5
t3	0	0	0	0	$-0.4(z,A)$	$-0.4(z)$	(z)	0.8	0.5
t4	0	0	0	$-0.45(y)$	$-0.45(y,A)$	0	0	1.0	0.5
M_0	$(\{1.0(F)\}$	$\{0.8(F)\}$	$\{0.8(F)\}$	$\clubsuit)$	$\{0.9(P,A)\}$	$\{1.0(A)\}$	$\clubsuit)$		
			↓ t1						
$M=$	$(\{1.0(F)\}$	$\{0.8(F)\}$	$\{0.8(F)\}$	$\{0.774(F)\}$	$\{0.9(P,A)\}$	$\{1.0(A)\}$	$\clubsuit)$		
			↓ t2						
$M=$	$(\{1.0(F)\}$	$\{0.8(F)\}$	$\{0.8(F)\}$	$\{0.774(F)\}$	$\{\{0.9(P,A)\}, \{0.81(F,A)\}\}$	$\{1.0(A)\}$	$\clubsuit)$		

Given $0.774 * 0.45 + 0.81 * 0.45 > 0.5$ and $U = \{F/y\}$, goal transition $t4$ can be fired, and $y = F$. Then we can get the result: $-0.45\text{Terrible}(F)$, $0.45At(F, A)$.

The proof of correctness:

Algorithm 1′ in NNPrF is extended from Algorithm 1 in NNF. We have proved the correctness of Algorithm 1. So here Algorithm 1′ is correct too.

Like the forward reasoning in NNPrF, the backward reasoning and learning in NNPrF can easily be extended from NNF. But it is beyond the scope of this paper to discuss them.

References

[1] Looney C. Fuzzy Petri nets for rule-based decision making. *IEEE Trans. on Syst., Man.*, 1988, (Jan.): 178-183.

[2] Peterka G, Murata T. Proof procedure and answer extraction in Petri net model of logic programs. *IEEE Trans. on Software Engineering*, 1989, 15(2): 209-217.

[3] Lin Chuang. Application of Petri nets to logical inference of HORN clauses. *Journal of Software*, 1993, 4(4): 33-37.

[4] Zhou Yi, Wu Shilin. The new ways of logic inference of Petri net based on resolution refutation. (accepted by *Chinese Journal of Computers*)

[5] Shen Qing, Tang Lin. The Introduction of Pattern Recognition. Press of the University of National Defense Science and Technology, May 1995.

[6] He Xingui. Knowledge Processing and Expert System. Press of National Defense Industry, Sep. 1990.

Zhou Yi received her B.S. degree from Computer Science Department of Fudan University in 1993. She is currently pursuing her M.S. degree in computer science from Fudan University. Her main research interests include knowledge representation and reasoning, machine learning, computer network, theory of Petri nets and its applications.

Wu Shilin graduated from Mathematical Department, Fudan University in 1957. He is now a Professor of Department of Computer Science, Fudan University. At present, he is the Vice President of Petri Net Community, Chinese Computer Federation. The areas of his research cover computer communication, computer network, theory of Petri net and its applications.