A fuzzy Petri net model for intelligent databases

Burcin Bostan-Korpeoglu *, Adnan Yazici

Department of Computer Engineering, Middle East Technical University, TR-06531 Ankara, Turkey

Received 31 May 2006; accepted 16 August 2006
Available online 18 September 2006

Abstract

Knowledge intensive applications require an intelligent environment, which can perform deductions in response to user queries or events that occur inside or outside of the applications. For that, we propose a fuzzy Petri net (FPN) model to represent knowledge and the behavior of an intelligent object-oriented database environment, which integrates fuzzy, active and deductive rules with database objects. The behavior of a system can be unpredictable due to the rules triggering or untriggering each other (non-termination). Intermediate and final database states may also differ according to the order of rule executions (non-confluence). In order to foresee and solve problematic behavior patterns, we employ a static rule analysis on the FPN structure that provides easy checking of the termination property without requiring any extra construct. In addition, with our proposed fuzzy inference algorithm, we guarantee confluent rule executions. The techniques and solutions provided in this study can be used in various complex systems, such as weather forecasting applications, environmental information systems, defense applications.

() 2006 Elsevier B.V. All rights reserved.

Keywords: Object-oriented database; Knowledge-base; Fuzziness; Active database; Fuzzy Petri net; Static rule analysis; Termination; Confluence

1. Introduction

Knowledge intensive applications require an intelligent environment which can perform deductions [19,8]. In such an application, there can be two reasons for deduction; one of them is user queries and the other is events occurring inside or outside of the system. To handle these deduction requirements, we propose a fuzzy Petri net (FPN) model to represent knowledge and the behavior of an intelligent object-oriented database environment. Our architecture fulfills the requirements of complex real world applications such as weather forecasting, environmental monitoring, defense applications. For example, in the weather forecasting application there exists huge amounts of data related to the atmospheric elements, such as pressure, temperature, humidity, coming from sensors connected to weather stations. Since a couple of changes taking place at the
same time are the indications of some forthcoming events, the system needs to perform deductions in order to determine the possible results of these changes in the environment. For example, high temperature, low pressure and high humidity in certain location and time may trigger heavy rain, which can be represented as a fuzzy rule in our model. A particular value change on an atmospheric element often triggers multiple fuzzy rules, which should be executed concurrently and require a fuzzy inference mechanism. In addition there could be state changes in the application environment, such as seasonal changes. In winter, we expect weather events like snow, freeze, etc., while in spring we expect rain, hail or shower. The application should be intelligent enough to handle these state changes, i.e., give more importance to some rules or prune some others according to the state. There may also be user queries which require deductions. These queries may be on the current values of the elements of the application domain. For example, in a weather forecasting application, we can have a query like “Which cities are very hot?”. We may also want to know the future trends of the elements of the application domain. For example, we may ask queries like “What is the expected temperature change in Paris within 12 hours?” or “What is the expected weather event in Paris these days?”. We can increase such examples for many other real world applications. In order to satisfy the requirements of such knowledge intensive applications, we integrate fuzzy, active and deductive rules with database objects. That is, our intelligent database environment allows objects to perceive dynamic occurrences or user queries after which they produce new knowledge or keep themselves in a consistent, stable, and up-to-date state; thus, performing intelligent behavior.

In literature, it has been argued that integration of active and deductive paradigms into a unique homogeneous framework is an important and challenging goal [32]. There have been a number of studies dealing with the problem of defining a unified semantics for deductive and active rules. In that, there exist two research directions for integration: some of them [32,4] try to extend deductive databases to support active behavior, while others [12] study how deductive rules can be implemented by means of active rules. For example, Zaniolo [32] uses a non-monotonic extension of logical clauses, which includes negation and aggregates under $XY$-stratification semantics. In that, active rules are expressed by means of built-in predicates which implement basic update operations. Bayer and Jonker [4] specify a framework for supporting triggers in the context of deductive databases. While event specification is defined on insertion or deletion of facts to predicates and on their composition, condition specification uses extended datalog with negation and built-in predicates. A different line of research is presented by Ceri and Widom [12], which uses production rules to physically maintain intentional data defined by the deductive rules. In that study, active rules are derived automatically in order to maintain intentional data when extensional relations are updated by users. In our study, users can express active and deductive rules independently in their traditional form. However, we consider deductive rules as special cases of active rules, where we use abstract kind of events. Therefore, internally all rules are of active kind and their processing is modelled with a fuzzy Petri net.

We use fuzzy Petri nets (FPNs) to represent knowledge and to model the behavior of the system. Also, we check the properties of the system, i.e., perform static rule analysis, using our FPN. Petri nets, in general, are considered as a graphical and mathematical modeling tool. They are powerful in modeling information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel and non-deterministic [24]. Several kinds of Petri nets have been investigated as tools for representing rules in knowledge based systems. The main advantage of using Petri nets in rule-based systems is that they provide a structured knowledge representation in which the relationships between the rules in a knowledge base are easily understood, and they render a systemic inference capability [15]. Considering uncertain and imprecise knowledge existing in various knowledge intensive applications, the degree of truth of rules and facts represented in a knowledge base is expressed as a real number in interval $[0,1]$. Fuzzy Petri nets (FPNs) are formed [15] to handle such fuzzy expressions.

Using FPNs to model fuzzy rule based reasoning provides a couple of advantages [21,22]:

- FPNs’ graphical representation can help experts to construct and modify fuzzy rule bases.
- They can model the dynamic behavior of fuzzy rule-based reasoning. Evaluation of markings is used to simulate the dynamic behavior of the system. The explanation of how to reach conclusions is expressed through the movements of tokens in FPNs.
FPNs eliminate the need to scan all the rules. They improve the efficiency of fuzzy rule-based reasoning by using transitions and arcs to connect fuzzy rules as a net-based structure.

FPNs’ analytic capability can help with checking properties of a modeled system to gain deeper insights into the system.

There have been a number of studies using FPNs. Chen et al. [14] consider a representation of fuzzy production rules with certainty factors. The reasoning algorithm used in that study determines whether there exists an antecedence–consequence relationship between two propositions. If this is the case, the degree of truth of the consequent proposition is evaluated from that of antecedent propositions. In a later work by Chen [13], the authors extend their previous work with a weighted fuzzy Petri net (WFPN) model, where certainty factors, truth values of propositions and weights of propositions are represented by fuzzy numbers. However, in their work [14,13] only exact matching is allowed. A similar approach taken by Manoj et al. [23] modify Chen et al.’s [14] fuzzy reasoning algorithm after finding out that it does not work with all types of data. Chun and Bien [15] study algebraic forms of a state equation of the FPN, which are systematically derived using a matrix representation. They use state equations to perform both forward and backward reasoning. However, they change the firing rule of conventional Petri nets. A different FPN model is studied by Bugarin and Barro [10], who present the execution of a knowledge base by using a data driven strategy based on the sup-min compositional rule of inference. However, since the arrangement of the linking transitions and the applied algorithm depend on the initial marking, their work is not appropriate for large systems. Finally, Scarpelli et al. [29] propose a high level fuzzy Petri net (HLFPN) for modeling fuzzy reasoning based on compositional rule of inference. Their forward reasoning algorithm consists of the extraction of a subnet from an entire net. However, after extracting the subnet, it is not allowed to have concurrent inference.

All these studies model only the behavior of their systems by using FPNs. However, due to the unstructured and unpredictable nature of rule processing, rules can be difficult to program and the behavior of a system can be complex and sometimes unpredictable. In an active database, rules may trigger and untrigger each other, and the intermediate and final states of the database can depend upon which rules are triggered and executed in which order. In order to determine these undesirable behavior patterns of the rule base, static rule analysis should be performed [1]. Such an analysis involves identifying certain properties of the rule base at compile-time, which gives the programmer an opportunity to modify the rule base. However, none of the studies mentioned above does static rule analysis.

Two important and desirable properties of the active rule behavior are termination and confluence. These properties are defined for user-defined changes and database states in a given rule set.

- **Termination**: A rule set is guaranteed to terminate if, for any database state and initial modification, rule processing does not continue forever (i.e., rules do not activate each other indefinitely).
- **Confluence**: A rule set is confluent if, for any database state and initial modification, the final database state after rule processing is unique, i.e., it is independent of the order in which activated rules are executed.

Static analysis techniques only give sufficient conditions for guaranteeing the property searched for. For example, the identification of potential non-termination in a set of rules indicates the possibility of infinite loops at run time, while the identification of potential non-confluence indicates that a rule base may exhibit non-deterministic behavior.

The main objective of this paper is to present a model and technique to handle knowledge-intensive fuzzy active database applications. In order to do that, we utilize fuzzy Petri nets (FPNs). Compared to the previous studies existing in literature, we extend the FPN with a number of new features. First, we can model any rule type (active or deductive), unlike previous works which do not model active rules. Secondly, we not only model the composition of conditions but also the composition of events. Note that, since the previous studies using FPNs cannot model active rules, they do not consider event composition. Additionally, we extend the functionalities of transitions. That is, we render them capable of performing fuzzification, event/condition composition, concurrent execution, and combination in addition to the sup-min composition. Fourth, since we use the features of colored Petri nets and there exists a rich set
of token types, parameter passing is provided through the FPN. This provides values of conditions and actions to be calculated from the parameters of events. Finally, we check the properties of our system using the FPN structure. This has not been considered by any other study that utilizes FPNs. Our FPN structure already contains the Triggering Graph information and supports the static analysis of the rule base. That is, we can perform the termination analysis easily by just using the FPN. Therefore, there is no need to do extra work to construct Triggering Graph as required by other studies [1,11,16,18,30], which use structures other than FPN for studying rule analysis. In addition, while performing termination analysis, we consider event compositions. We also guarantee confluent rule execution with the fuzzy inference algorithm that we introduce in this paper.

The organization of this paper is as follows: In Section 2, colored Petri nets are briefly described as background. In Section 3, our FPN model for fuzzy rule-based reasoning is presented. We show how we map the fuzzy rules to FPN and how we construct the FPN. In Section 4, after giving the assumptions and understandings in termination analysis, we explain the details of how we perform termination analysis on our FPN model. Section 5 describes how we guarantee confluence in our model with our fuzzy inference algorithm. Finally, we include conclusions and future work in Section 6.

2. Colored Petri nets

Petri nets are represented as directed graphs with two type of nodes (places and transitions) connected by arrows that specify the direction of information flow. After Petri nets’s first introduction by Petri [28], there has been a number of extensions including colored Petri nets [24], fuzzy Petri nets [15], etc. In this paper, we will briefly introduce colored Petri nets as a background.

To facilitate the formal specification and analysis of the structure, information flow, control and computation on systems, colored Petri nets (CPNs) have been introduced [24]. Formally a CPN [17] is a structure \( (P, T, A, N, C, CF, E, G, M_0) \), where

(i) \( P, T, A \) are the same as those in (ordinary) Petri nets.
(ii) \( N : A \to (P \times T \cup T \times P) \) is a node function.
(iii) \( C \) is a finite and non-empty set of data types (called color types). The set of color types determines the types, operations and functions that can be used in the net inscriptions.
(iv) \( CF : P \to C \), is a token type (color) function. It maps each place to a token type in \( C \).
(v) \( E : A \to \text{Expression } E(a) \) of type \( C(p(a)) \), is the arc function. It maps each arc to an expression called an arc expression such that

\[
\forall a \in A : \quad \text{Type}(E(a)) = CF(p(a))_{MS} \land \text{Type}(Var(E(a))) \subseteq C
\]

where \( p(a) \) is the place of \( N(a) \) and \( CF_{MS} \) denotes set of all multisets over \( CF \). In this mapping the expression \( E(p,t) \) is the name of a variable associated with the arc from input place \( p \) to transition \( t \), and the expression \( E(t,p) \) is associated with the transformation performed by transition \( t \) on its inputs to produce an output for place \( p \).
(vi) \( G : T \to \text{Boolean expression} \) is a guard function. It is defined from \( T \) into expressions such that

\[
\forall t \in T : \quad \text{Type}(G(t)) = \text{Bool} \land \text{Type}(Var(G(t))) \subseteq C
\]

(vii) \( M_0 \) is an initialization function. It is defined from \( P \) into expressions such that

\[
\forall p \in P : \quad \text{Type}(M_0(p)) = CF(p)_{MS} \land Var(M_0(p)) = \phi
\]

In contrast to (ordinary) Petri nets [6,7], in which a token represents a typeless fragment of information, CPNs can carry complex information. They provide data typing (color sets) and sets of values of a specified type for each place. An arbitrary assignment of tokens to places is called marking. A particular marking specifies the state of a system being modeled with the net. Tokens are used to define the execution of a net. Places represent storage for input or output. Transitions represent activities (transformations). Every time an input place of the net is marked, whether the corresponding transition(s) can fire has to be checked.
A guard is a boolean expression on a transition $t$ which must be satisfied before $t$ can fire. A transition $t$ can fire if and only if all of its input places are marked with at least $E(p,t)$ tokens, and the guard function $G(t)$ is satisfied (if there is any). The firing of a transition $t$ removes $E(p,t)$ tokens from each input place $p$ of $t$ and adds $E(t,p)$ tokens to each output place $p$ of $t$. Thus, when transition $t$ fires, the subsequent marking satisfies the following equation:

$$M'(p) = \begin{cases} M(p) + E(t,p), & \forall p \in \text{output places} \\ M(p) - E(p,t), & \forall p \in \text{input places} \\ M(p), & \text{otherwise} \end{cases}$$

where $M(p)$ is the marking before transition $t$ fires and $M'(p)$ is the marking after transition $t$ fires. The above transition firing rule is shown in Fig. 1 with an example. In that, variables are denoted by $x, y, z, \ldots$ and the constants are by $a, b, c, \ldots$. The arc labels dictate how many and which kind of colored tokens will be removed or added to the net places. The transition $t$ is said to be enabled if there are enough tokens of the right color in each input place of $t$ if the guard function (here $x = z$) is satisfied.

A recent work by Kwon [20] propose a modeling methodology to represent and analyze a context-aware agent-based system by using CPNs as a method of capturing the dynamics of contextual changes in the system.

3. Fuzzy Petri nets for fuzzy rules

We introduce the following fuzzy Petri net (FPN) structure to model the fuzzy rules: $(P, P_s, P_e, T, TF, TRTF, A, I, O, TT, TTF, AEP, PR, PPM, TV)$, where

(i) $P$ is a finite set of fuzzy places. Each place has a property associated with it, in which
- $P_s \subseteq P$ is a finite set of input places for primitive events.
- $P_e \subseteq P$ is a finite set of output places for actions or conclusions.
(ii) $T$ is a finite set of fuzzy transitions. They use the values provided by input places and produce values for output places.
(iii) $TF$ is a finite set of transition functions, which perform activities of fuzzy inference.
(iv) $TRTF : T \rightarrow TF$ is transition type function, mapping each transition $\in T$ to a transition function $\in TF$.
(v) $A \subseteq (P \times T \cup T \times P)$ is a finite set of arcs for connections between places and transitions. Connections between the input places and transitions $(P \times T)$ and connections between the transitions and output places $(T \times P)$ are provided by arcs. In that
- $I : P \rightarrow T$ is an input mapping.
- $O : T \rightarrow P$ is an output mapping.

![Fig. 1. Firing a Petri net.](image-url)
(vi) TT is a finite set of \textit{fuzzy token (color) types}. Each token has a linguistic value (i.e., low, medium and high), which is defined with a membership function.

(vii) \(\text{TTF}: \mathcal{P} \rightarrow \text{TT} \) is \textit{token type function}, mapping each fuzzy place \(p \in \mathcal{P}\) to a fuzzy token type \(\in \text{TT}\). A token in a place is characterized by the property of the place and a level to which it possesses that property.

(viii) \(\text{APR}: \text{Arc} \rightarrow \text{expression}\), is \textit{arc expression function} mapping each arc to an expression, which carries the information (token values).

(ix) \(\text{PR}\) is a finite set of \textit{propositions}, corresponding to either events or conditions or actions/conclusions.

(x) \(\text{PPM}: \mathcal{P} \rightarrow \mathcal{PR}\), is a \textit{fuzzy place to proposition mapping}, where \(|\mathcal{PR}| = |\mathcal{P}|\).

(xi) \(\text{TV}: \mathcal{P} \rightarrow [0,1]\) is \textit{truth values of tokens} \((\mu_{i})\) assigned to places. It holds the degree of membership of a token to a particular place.

A token value in place \(p_i \in \mathcal{P}\) is denoted by \(\text{TV}(p_i) \in [0,1]\). If \(\text{TV}(p_i) = \mu_i, \mu_i \in [0,1]\) and \(\text{PPM}(p_i) = d_i\). This states that the degree of the truth of proposition \(d_i\) is \(\mu_i\). A transition \(t_i\) is enabled if \(\forall p_i \in I(t_i), \mu_i > 0\). If this transition \(t_i\) is fired, tokens are removed from \(I(t_i)\) and a token is deposited onto each of the output places \(O(t_i)\). Since we provide parameter passing, the token value of an output place \(p_k \in O(t_i)\) is calculated from that of the input places \(I(t_i)\) using the transition function \(\text{TF}_i\), where \(\text{TF}_i = \text{TRTF}(t_i)\). This token’s membership value to the place \(p_k\), (i.e., \(\mu_k = \text{TV}(p_k)\)), is part of the token and gets calculated within the transition function \(\text{TF}_i\), where \(\mu_k = \text{TF}_i(I(t_i))\).

\textbf{Example.} The fuzzy deductive rule \(\text{IF } d_i \text{ and } d_j \text{ and } d_m \text{ THEN } d_k\) can be modeled as shown in Fig. 2. In this example, \(\text{PPM}(p_i) = d_i, \text{PPM}(p_j) = d_j, \text{PPM}(p_m) = d_m, \text{PPM}(p_k) = d_k, \text{TV}(p_i) = \mu_i = 0.5, \text{TV}(p_j) = \mu_j = 0.4\) and \(\text{TV}(p_m) = \mu_m = 0.6\). Since \(\mu_i > 0, \mu_j > 0\) and \(\mu_m > 0\), transition \(t_n\) is enabled and fired. Tokens are removed from \(I(t_n)\), which are \(p_i, p_j, p_m\) and deposited onto \(O(t_n)\), which is \(p_k\). Suppose that the transition function of \(t_n\), which is \(\text{TF}_n = \text{TRTF}(t_n)\), is defined as a \textit{min} operator. Then the truth value of the output token (membership degree) is calculated as \(\text{TV}(p_k) = \text{TF}_n(I(t_n)) = \min(\mu_i, \mu_j, \mu_m) = \mu_k = 0.4\).

![Fig. 2. Firing the fuzzy Petri net.](image-url)
3.1. Models of transitions and places

In a study by Pedrycz and Gomide [27], a generalization of fuzzy Petri net is described with triangular norms (\(t\)- and \(s\)-norms) as computational models of the logical connectives. We will show that, that generalization coincides with our fuzzy Petri net model with some conditions of the marking of the places and the parameters of the net.

The conditions for firing a transition, i.e., the degree of its firing \(z\) defined in \([0,1]\) is given as

\[
    z = T^n_{i=1}((\tau_i \rightarrow x_i)sw_i)
\]

This firing condition describes a conjunctive type of firing in which all the input places (their markings \(x_1, x_2, \ldots, x_n\)) are sought as AND-wise contributors. \(\tau_i\) are used to characterize the threshold level modulating the strength of firing coming from the \(i\)th input place. The weight factor \(w_i\) is used to discriminate the input places with respect to their contribution to the overall level of firing. Once the transition has been fired, each input place of the transition decreases its current level of tokens \(x_i\) moving the marking down to the next level, \(x_i(\text{next})\), where

\[
    x_i(\text{next}) = x_i/z
\]

which depends upon the complement of the produced level of firing \(\bar{z}\). At the same time each associated output place \(y_j\) upgrades its amount of tokens yielding the marking,

\[
    y_j(\text{next}) = y_jsz
\]

Those markings always hold the equalities \(x_i(\text{next}) \leq x_i\) and \(y_j(\text{next}) > y_j\). The disjunctive (OR-wise) form of firings is described as

\[
    z = S^n_{i=1}((\tau_i \rightarrow x_i)tw_i)
\]

In our model,

(i) we have \(r_i = 1\) and \(w_i = 0\) for all \(i = 1, 2, \ldots, n\) which means

\[
    z = T^n_{i=1}(1 \rightarrow x_i) = T^n_{i=1}x_i
\]

where \(x_i = TV(p_i)\). In order for a firing to occur, \(TV(p_i)\) for all \(i = 1, 2, \ldots, n\) must be greater than 0. Otherwise, \(z = 0\) and no firing occurs.

(ii) If there happens a firing, \(\bar{z} = 0\), then \(x_i(\text{next}) = x_i\), \(0 = 0\), meaning input places loses tokens. At the same time output place gains token, \(y_j(\text{next}) = y_jsz\), where \(z > 0\).

(iii) If no firing happens, \(\bar{z} = 1\), then \(x_i(\text{next}) = x_i\), \(1 = 1\), \(y_j(\text{next}) = y_jsz = y_j\) meaning that no changes occur to the status of input and output places.

3.2. Mapping fuzzy rules to fuzzy Petri net

We define our rules, which can be either active or deductive type, in the following form:

\[
\text{<ON event list <event threshold>>}
\text{IF condition list <EC coupling>}
\text{THEN action/conclusion list <CA coupling>}
\]

where the parts inside the \(< >\) means they are optional. If the \(ON\) part is omitted, it becomes a deductive rule. We use the abstract kind of events for the deductive rules so that internally all rules are of active type [8]. If a rule defined is an active one and the threshold value is omitted, and exact matching with a value of 1 is assumed. If coupling modes are omitted in an active rule, they are assumed as immediate. Here all coupling modes are assumed as immediate. Using the above form, an active rule \(R_i\) can be defined as
Ri:

ON ei <event threshold>
IF ci
THEN ai

where $e_i$ corresponds to either a primitive or a composite event. If it is a composite one, it can be constructed from conjunctions and/or disjunctions of primitive/composite events. The same applies to $c_i$, which can either be a primitive or a composite condition. On the other hand, the action/conclusion part $a_i$ can only contain conjunctions of actions/conclusions.

Before going into the details of mapping fuzzy rules to the FPN structure, we shall explain what a scenario means. A scenario is a set of rules of a database state. We partition the rule space and call each partition a scenario. There can be only one active scenario at a time. Switching between the scenarios is determined by the user. During the fuzzy inference cycle, we calculate the strength of events for the rules fired. This calculation is affected by the current active scenario. Considering a weather forecasting application, we would have a set of rules for expected weather and weather events. Our scenarios are determined by the seasons as summer, fall, winter, spring, in which we partition the rule space according to seasons for the expected weather and weather events. For example, in summer we expect a clear or clear few weather. This means that any rule similar to the rule of clear or clear few weather will have a tendency to fire when we are in summer (i.e., summer scenario is on). We give two formulations which are used in the fuzzy inference in the example given in Section 6: the Strength of an event $e$, for rule $r$ within the scenario $s$ is calculated using the formula

$$
\text{strength}(e, r, s) = \mu_s(r) \cdot \mu_{ef}(\text{value}(e_t))
$$

which uses scalar multiplication, where value($e_t$) is the value of the event (either fuzzy or crisp) detected, $\mu_{ef}$ is the membership function of the fuzzy event $e_f$ and $\mu_s(r)$ is the similarity of the rule $r$ to the current scenario $s$.

The formula of $\mu_s(r)$ is defined as

$$
\mu_s(r) = \max(\min(\max(A_s, A_r), \max(C_s, C_r)) \cdot RLV_{rs} / RLV_{max})
$$

where $A_s \in S$, $\forall A_r \in R_r$, $C_s \in S$, $\forall C_r \in R_r$, $RLV_{rs}$, $RLV_{max} \in S$. In that $A$ and $C$ correspond to the antecedent and consequent of a rule. The antecedent is composed of event and condition whereas the consequent is composed of action/conclusion. Here $A_s$ is the antecedent of a current scenario meta rule and $A_r$ is the antecedent of $R_r$ (the rule to be evaluated), $C_s$ is the consequent of a meta rule in the scenario and $C_r$ is the consequent of the rule to be evaluated, $RLV_{rs}$ is the relevance value of the meta rule to the current scenario and $RLV_{max}$ is the maximum of those relevance values.

Returning to the mapping of rules to the FPN structure, we place the constituents of our rules onto the FPN places and transitions. Here is the list of items that we consider while doing this mapping:

1. $\mathbb{PR} = \{pr_1, pr_2, \ldots, pr_n\}$ is a finite set of propositions. A proposition can be any of the following:
   a. primitive events,
   b. fuzzy primitive events,
   c. fuzzy composite events (conjunctions/disjunctions of fuzzy primitive/composite events),
   d. similarities of rules to the current scenario,
   e. fuzzy primitive conditions,
   f. fuzzy composite conditions (conjunctions/disjunctions of fuzzy primitive/composite conditions),
   g. clipped actions/conclusions and
   h. for each concurrent rule set
      • a combined action, and
      • a fuzzy action.

2. $\mathbb{PPM}: \mathbb{P} \rightarrow \mathbb{PR}$, is a mapping from places to propositions where $|\mathbb{PR}| = |\mathbb{P}|$. For each proposition $pr_i$, $i = 1, \ldots, n$, let $p_i$ be a place in the FPN. Some of these places are specialized as $P_s$ and $P_e$.
   a. $P_s \subset \mathbb{P}$ is a finite set of input places corresponding to primitive events. They are the starting places in the fuzzy inference mechanism.
b. $P_e \subset P$ is a finite set of output places, which correspond to fuzzified actions/conclusions. They are the final places in the fuzzy inference mechanism.

3. $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions. While executing transitions, we go through the inference steps. Transitions use the values provided by input places and produce values for output places by using the specific purpose functions. Their functionality can be listed as:
   a. fuzzification of primitive events,
   b. fuzzification of primitive conditions,
   c. providing fuzzy event/condition composition (conjunctions/disjunctions of fuzzy primitive/composite events/conditions),
   d. providing a pass from event to condition evaluation;
      - calculates strengths of events,
      - checks the rule threshold,
      - provides input for the condition evaluation,
   e. finding clipping values for the individual rules by using the event and condition match factors of each rule,
   f. combining the outcome of concurrent actions/conclusions,
   g. determining the overall fuzzy action,
   h. triggering events (marking primitive event places).

4. $A \subseteq (P \times T \cup T \times P)$ is a finite set of arcs for connections between places and transitions. Arcs can be expressed in matrix forms. For this, output and input incidence matrices are used. Since the number of arcs between a transition and a place is either 1 or 0, both matrices are binary matrices.

5. $\text{AEF}: A \rightarrow \text{expression}$, is a mapping for each arc to an expression, which can be one of the following:
   a. an input arc expression for arcs $P \times T$, consisting of the variables of the input place of the arc, which carries information of an input place token,
   b. output arc expression for arcs $T \times P$, consisting of variables to be assigned to output places.

6. $\text{TV}: P \rightarrow [0,1]$ is a mapping for the truth values of propositions ($\mu$) assigned to places. It can be any of the following (only one of them for each place);
   a. 1 (one) for the crisp event,
   b. truth values of the fuzzy primitive events/conditions,
   c. truth values of the composite events/conditions,
   d. similarities of rules to the current scenario,
   e. strengths of events,
   f. clipping values for actions/conditions,
   g. 1 (one) for the combined actions/conclusions, and
   h. a truth value for the overall fuzzy action/conclusion.

7. $\text{TT}$ is a finite set of token(color) types.

8. $\text{TTF}: P \rightarrow \text{TT}$ is a token type function, mapping each place $P$ to a token type $T$.

Fig. 3 shows how we realize the steps of fuzzy inference using the FPN structure. Inference starts at primitive events and ends at combined fuzzy actions/conclusions. Between them, we fuzzify primitive events, obtain compositions of events, trigger conditions, calculate their compositions, reach to the actions for each rule. After combining concurrent actions we reach to the end places, which are combined actions/conclusions. If these end places cause the generation of new events, related primitive events are marked. Passing from one state to another is achieved through the execution of some specific purpose functions.

3.3. Constructing the FPN

We follow an approach in which FPN is constructed directly from given rule definitions. Construction algorithm takes the rules specified in a predefined syntax as input. Then it generates the corresponding FPN structure automatically. More specifically, first the rule definitions are obtained from the user. For each rule, a rule object is created, and the event, condition and action parts of the rule are examined. In that, related FPN places such as primitive events, fuzzy primitive events, etc., are created. Then, the
fuzzy inference groups, which are the concurrent rule sets that get triggered at the same time, are determined. Finally, transitions are constructed over these FPN places. The pseudocode of the algorithm is given in Algorithm 1.

**Algorithm 1. Construct_FPN algorithm**

```
Begin
While there are still some rules do
    Create a rule object
    Examine Event, Condition, Action parts
    For each one of them do
        Create related FPN places
    end for
end while
Construct fuzzy inference groups
Construct transitions
End
```

During the transitions construction, some specific purpose functions are assigned to transitions for value passes between the places. For example, we assign an AND operator to some transitions where required for the conjunctions of distinct events. The transitions provide the inference steps to be realized with the help of these functions. In transitions construction phase, we also decide which action execution or condition evaluation generates new events, i.e., triggers new rules. This is done by performing unification of condition and action calls with event specifications. Our unification principle depends on having the same fuzzy attributes of the same fuzzy domain in the related FPN places. But they may have different fuzzy linguistic terms. For these cases, a transition is added from related condition/action FPN places to the generated primitive event in the FPN (event generating transition).

The rules in the same fuzzy inference group are determined according to their action parts. Rules are in the same fuzzy inference group if action parts unify (their fuzzy attribute domains are the same but they may use different fuzzy linguistic terms). For each fuzzy inference group a combined action and a fuzzy action FPN places are created.

During the FPN construction, the attributes of the rule objects are updated to hold the related links on FPN. These attributes are $PN_{\text{primitiveevent}}, PN_{\text{fuzzyprimitiveevent}}, PN_{\text{fuzzycompositeevent}}, PN_{\text{event}}, PN_{\text{rule}}, PN_{\text{strengthenedevent}}, PN_{\text{fuzzyprimitivecondition}}, PN_{\text{fuzzycompositecondition}}, PN_{\text{condition}}, PN_{\text{clippedaction}}, PN_{\text{combinedaction}}$ and $PN_{\text{fuzzyaction}}$. In addition, each rule object holds the list of rules in the same fuzzy inference group in its $fuzzyinferencegroup$ attribute. Also, each FPN place has a $rule_set$ attribute in order to hold the rule objects that uses the FPN place. Fig. 4 shows the place types that we use in the FPN.

4. Termination analysis

The termination for a rule set is guaranteed if rule processing always reaches a state in which no rule is triggered. Several methods have been proposed in the literature to perform termination analysis. One of them is building a triggering graph by considering the type of triggering events and events generated by the execution of rule actions [1,11,18].

Formally, a triggering graph (TG) [1] is a directed graph $\{V, E\}$, where each node in $V$ corresponds to a rule in the rule base and each edge $r_i \rightarrow r_j$ in $E$ means that the action of the rule $r_i$ generates events that trigger $r_j$. If there are no cycles in the triggering graph, then processing is guaranteed to terminate. TG, however, fails to take into account the details of the interaction between the conditions and the actions of potentially non-terminating rules. That is, although TG has a cycle, it may be the case that, when a rule in the cycle gets triggered, the condition of that rule is not true. As a result, the rule is not executed and the cyclic chain is broken. Consider for example the following rule:
R1:
ON update to attribute A of T
IF new value A > 10
THEN set A of T to 10

The TG for the rule base involving $R_1$ contains a cycle, as the action of $R_1$ updates the attribute $A$ of $T$, which in turn triggers $R_1$. However, non-termination does not result as the action of $R_1$ assigns $A$ a value for which the condition of $R_1$ never becomes true. It is to overcome this limitation in TGs that activation graphs have been introduced.

An activation graph (AG) [2] is built upon the semantic information contained in the rule conditions and actions. It is a directed graph $\{V, E\}$, where each node in $V$ corresponds to a rule in the rule base, each edge
\( r_i \rightarrow r_j (i \neq j) \) in \( E \) means that the action of the rule \( r_i \) may change the truth value of the condition of \( r_j \) from false to true, and each edge \( r_i \rightarrow r_i \) means that the condition of \( r_i \) may be true after the execution of its own action. If there are no cycles in the AG, then processing is guaranteed to terminate. Some studies [3] rely mostly on the AG while making termination analysis.

Other studies [2] try to detect potential non-termination by using both TG and AG together where a rule set can only exhibit non-terminating behavior when there are cycles in both the triggering and the activation graphs that have at least one rule in common.

Returning to the example given above, the AG for rule \( R_1 \) contains no cycle, because its condition cannot be true after the execution of its action. Thus, even though the TG contains a cycle, the execution of the rule terminates.

While TG arcs are syntactically derivable, it is very difficult to determine precisely the arcs of an AG. In Baralis et al. [2], it is assumed that conditions and actions are both represented by relational algebra expressions. Unfortunately, for the host languages that are not based on relational algebra or relational calculus, it is difficult to infer the truth value of a condition from the imperative code of a rule’s action [16]. Due to this fact, most of the studies [16,18,30,25,26] including ours [9] consider only triggering graph information during termination analysis.

The existing studies on rule analysis only deal with simple rule languages; i.e., languages that support only a limited set of constructs for specifying active behavior. If the rules become complex (like having composite events, supporting complex conditions/actions, etc.), their analysis also becomes complex. The reason is that there are additional elements on which the triggering of a rule depends. For example, a rule defined on a complex event is triggered only when all component events occur. Therefore, compared to a simple rule language, it is more difficult to decide when rules may generate an infinite loop during execution. That is, in a system having only primitive events, an edge in the triggering graph indicates that one rule can generate an event that can in turn trigger another. However, when a rule \( r \) has a composite event, it may be that no other single rule in the rule base can trigger \( r \), but that a subset of the rules together may be able to trigger \( r \). Only a very few studies consider compositions while performing termination analysis [16,30].

4.1. An Algorithm for termination analysis on the FPN

Before we check the termination property of the system, we construct the FPN, and during the transition construction we obtain the triggering graph information from the event generating transitions. Note that an


**Algorithm 2.** Termination analysis algorithm

```plaintext
Begin

/*Construct_FPN maps rules to FPN in which unification is used. The constructed FPN also holds the triggering graph information.*/
Construct_FPN();

/*determine_true_cycles function calculates the cyclic paths, it eliminates the false cycles via considering event compositions*/
if(NULL == (true_cycle_list = determine_true_cycles())) then
    Termination guaranteed;
else
    inform_user(true_cycles_list);
end if
End
```

In order to determine whether there exists a cycle or not, all the rules inside that cycle are examined. If there exists a composite event in a rule taking place in the cycle, all composing events of that rule must be triggered within the cycle. Once the algorithm detects a cyclic path, it may not be a true cycle due to one or more rules in the cycle having composite events. That is, not all events in a composite event may be triggered. Such cases are determined in our termination analysis. Algorithm 2 finds the cycles and informs the user. Thus, finds out if the system terminates.

### 4.1. How to determine cycles and true cycles

For this, we use the following data structures:

a. D is an \( m \times m \) matrix of rules. Each of its \([i][j]\) entries holds the connectivity information between rules \( r_i \) and \( r_j \) (or whether there exists an event generating transition from \( r_i \) to \( r_j \)). \( D[i][j] \) is 1, if rule \( r_i \) is connected to rule \( r_j \) with one event generating transition.

b. L is an \( m \times m \) matrix of rules. Each of its \([i][j]\) entries holds a linked list of rules that are in the path from rules \( r_i \) to \( r_j \) inclusive.

Let \( D_k \) be an \( m \times m \) matrix holding the matrix multiplication result of \( k \) number of \( D \) matrices. It holds the connectivity information at \( k \) edges (or event generating transitions) distance. Assuming there is a cycle, if we have number_of_rules rules in the rule base, we can have at most number_of_rules distinct event generating transitions to go through in our FPN. (We start from a rule and go through all rules, then return to the starting rule, which is the path \( r_1, r_2, \ldots, r_{(k-1)}, r_k, r_1 \), having \( k \) connecting edges.) If the \( i \)th diagonal at \( D_k \) holds a value greater than 0(zero), there is a cycle in \( k \) steps (or \( k \) event generating transitions distance). Notice that all rules taking place in that cycle have values greater than 0(zero) in their diagonal entry. Comparing with the previous matrices which indicates a cyclic path, if the same diagonal elements having values greater than 0 are obtained at some \( k \) steps, the matrix multiplication should be terminated. Or if 0(zero) is obtained for all entries of a matrix, again the matrix multiplication should be terminated. This means that all edges have been consumed and there is no further edge to go through, which is an indication of no cyclic behavior.

By the time these \( D \) matrices have been calculated, a linked list of rules which have been gone through the path is held in the \( L \) matrix. This means that \( l[i][j] \) holds a linked list of rules which is passed through the path from rules \( r_i \) to \( r_j \). If the \( i \)th diagonal at \( D_k \) holds a value greater than 0(zero), we can obtain the cyclic path
elements at $l_k[i][j]$. Now it is easy to find true cycles by checking the $[i][j]$ entry of $L_k$. If all the rules in the path have only primitive events, this is certainly a true cycle. Otherwise (if any of them has a composite event) the rule set of the primitive events of the rules is examined. If they are all included in the cycle, again the cycle is a true cycle. Otherwise it is not a true cycle. If there is at least one true cycle, the user is informed about the situation together with the rules taking place inside the cycle. Then it is the user’s choice to change the definitions of these rules. After that, the termination analysis is repeated.

4.2. Example

Suppose there are the following two rules:

R1:
ON (e11 and e21) threshold R1
IF c1
THEN a1
R2:
ON e12 threshold R2
IF c2
THEN a2

Fig. 5. Dependencies and the triggering graph for the example.

Fig. 6. Fuzzy Petri net of the example.
In these rules $a_1$ unifies with crisp event $e_1$ (which has $e_{11}$ and $e_{21}$ as its fuzzy primitive events) and $a_2$ unifies with crisp event $e_2$ (which has $e_{12}$ as its fuzzy primitive event). The dependencies and the triggering graph are shown in Fig. 5. Dashed arcs show the partial triggering due to composite events and solid arcs show the total triggering. Fig. 6 shows the FPN constructed according to this rule set.

For this example, $D_1$ and $L_1$ are as follows:

$$D_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad L_1 = \begin{bmatrix} 1,1 & 1,2 \\ 2,1 & 0 \end{bmatrix}$$

Since $D_1[1][1]$ entry has the value 1, this means that there is a cycle. After checking $r_1 \cdot PN\text{\_primitive\_event\_rule\_set}$, rules $r_1$ and $r_2$ are found. Since the cyclic set {1,1} does not contain $r_2$, this is not a true cycle. Therefore it is excluded. Then $D_2$ is calculated and at the same time $L_2$ is obtained

$$D_2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \quad L_2 = \begin{bmatrix} 1,1,1;1,2,1 & 1,1,2 \\ 2,1,1 & 2,1,2 \end{bmatrix}$$

Since $D_2[1][1]$ entry has a value greater than 0, there is a possibility of a cycle. $r_1 \cdot PN\text{\_primitive\_event\_rule\_set}$ is the rules $r_1$ and $r_2$. Since the cyclic set {1,2,1} contains both $r_1$ and $r_2$, it is a true cycle. The user is informed about the cyclic situation together with the rules taking place in the cycle. The rules inside the cycle are $r_1$ and $r_2$. So, termination analysis returns the rules $r_1$ and $r_2$. This means that they are part of the cycle and need to be changed in order to break the cycle.

4.3. The Algorithm for determining the true cycles

The algorithm, whose pseudocode given below, finds the true cycles by considering event compositions.

**Algorithm 3. Determine_true_cycles algorithm**

```plaintext
Begin
  calculate $L_1$
  $k = 1$
  while (($D_k \neq 0$) and ($k \leq \text{number of rules}$)) do
    for each rule $r_i$ do
      //i is the index of the rule $r_i$ in the $D$ matrix
      if $D_k[i][i] \geq 1$ then
        if rules in the $L_k[i][i]$ are distinct then
          if none of the rules contains composite event then
            return cyclic rules ($L_k[i][i]$)
          else
            if all rule sets of the primitive events of the rules (which have composite event) are included in the $L_k[i][i]$ then
              return cyclic rules ($L_k[i][i]$)
            end if
          end if
        end if
      end if
    end for
    $k = k + 1$
    calculate $D_k$
    calculate $L_k$
  end while
  return no cycle
End
```
Computational complexity of the determining true cycles algorithm: In Algorithm 3, in worst case, we have to process the while loop \( n \) times (\( n \) is the number of rules). In each cycle, we have to obtain the matrix multiplications of two matrices each of which is \( n \) by \( n \). Matrix multiplication requires \( n^3 \) operations. So totally determining true cycles requires \( n^4 \) operations, \( O(n^4) \).

Lemma 4.3.1. Let there be \( k \) rules in the rule base. The algorithm determining the true cycle decides on termination in at most \( k \) steps.

Proof. If we have \( k \) rules in the rule base, we can have at most \( k \) distinct edges to go through if there is cyclicity. This is due to the fact that if there are \( k \) nodes, there may be at most \( k \) edges connecting these nodes in a circular path (considering that the nodes \( n_1, n_2, \ldots, n_{(k-1)}, n_k, n_1 \) can be gone through and this path has \( k \) edges).

Lemma 4.3.2. Let there be \( k \) rules in the rule base and all rules have primitive events. If there is no cyclic behavior, \( D_n \) matrix becomes 0 (zero) at finite number of steps which is at most \( k \).

Proof. If we have \( k \) rules in the rule base and at the same time if all rules have primitive events, there can be at most \( k - 1 \) distinct edges to go through if there is no cyclicity. This is due to the fact that if there are \( k \) nodes, there may be at most \( k - 1 \) edges connecting these nodes (considering that the nodes \( n_1, n_2, \ldots, n_{(k-1)}, n_k \) can be gone through and this path has \( k - 1 \) edges). In these cases \( k \)th edge goes nowhere and the \( D_n \) matrix becomes 0 (zero).

Lemma 4.3.3. Let rule \( r_i \) have a composite event. Although the \( D_n[i][i] \) entry is not 0 (zero) at some steps, it does not guarantee non-termination.

Proof. If \( r_i \) has a composite event, although the \( D_n[i][i] \) entry is not 0 (zero) (there is a cyclic triggering of the \( r_i \)), all the rules related to triggering the events of the \( r_i \) may not be taking place in the cyclic path. If this is the case, there is actually no true cycle.

5. Confluence analysis

On each execution of the scheduling phase of rule processing, multiple rules may be triggered, and hence be eligible for execution. A rule set is confluent if the final state of the database does not depend on which eligible rule has been chosen for execution.

The rule execution process can be described by the notions of rule execution state and rule execution sequence. Consider a rule set \( R \). A rule execution state \( S \) has two components: 1. a database state \( d \), and 2. a set of triggered rules \( R_T \subset R \). When \( R_T \) is empty, no rule is triggered and the rule execution state is quiescent. A rule execution sequence consists of a series of rule execution states linked by (executed) rules. A rule execution state is complete if its last state is quiescent.

Confluence can be defined in terms of execution sequences. A rule set is confluent if, for every initial execution state \( S \), every complete rule execution sequence beginning with \( S \) reaches the same quiescent state. Then confluence analysis requires the exhaustive verification of all possible execution sequences for all possible initial states. This technique is clearly unfeasible even for a small rule set.

A different approach to confluence analysis is based on the commutativity of rule pairs. Two rules \( r_i \) and \( r_j \) commute if, starting with any rule execution state \( S \), executing \( r_i \) followed by \( r_j \) produces the same rule execution state as executing \( r_j \) followed by \( r_i \).

If all pairs of rules in a rule set \( R \) commute, any execution sequences with the same initial state and executed rules have the same final state. Furthermore, under the assumption of commutativity, two sequences with the same initial execution state must have the executed rules. Based on these properties, it is possible to state a sufficient condition to guarantee confluence of a rule set: A rule set \( R \) is confluent if all pairs of rules in \( R \) commute [2].
Confluence may be guaranteed by imposing a total ordering on the active rule set [2]. Consider a prioritized rule set \( R \). If a total ordering is defined on the rules in \( R \), when multiple rules are triggered only one rule at a time is eligible for evaluation. Then, rule processing is always characterized by a single rule execution sequence, which yields a unique final state, and confluence is guaranteed.

5.1. Guaranteeing confluent executions in our FPN model

Before analyzing the confluence property of the FPN structure, we shall give our fuzzy inference algorithm based on the FPN. For this, we use some data structures for the internal representation of the FPN.

a. \( M \) is an \( m \times 1 \) column vector of places \( p_i \), where \( i = 1, \ldots, m \). Each of its entries holds a data structure of two elements:
   - the 1st element holds the number of tokens (\textit{current_marking}),
   - the 2nd element holds a linked list of \textit{current_marking} many token values, which works with FIFO queueing mechanism (\textit{token_values}).

b. \( N \) is an \( n \times 1 \) column vector of transitions \( t_j \), where \( j = 1, \ldots, n \). Each of its entries holds a transition function. Each transition function uses the head elements of the input places \textit{token_values} and produces an output to be added to the tail of the output places \textit{token_values}.

c. \( C^+ = (c^+_{ij}) \) and \( C^- = (c^-_{ij}) \) represent the output incidence matrix and input incidence matrix, respectively, where \( c^+_{ij} \) is 1 if there is an arc from the transition \( t_j \) to place \( p_i \) and \( c^-_{ij} \) is 1 if there is an arc from the place \( p_i \) to the transition \( t_j \) and their values are 0 if there is no connection.

Algorithm 4 gives the pseudocode of our inference algorithm based on FPN.

\textbf{Computational complexity of the inference algorithm:} In Algorithm 4, in worst case, we have 1(one) rule in each inference group, and each rule gets triggered. So, the outermost while loop should be performed \( n \) times. In each cycle we go over the transitions (let we have \( t \) number of transitions), and for each transition, we go over the places 2(two) times, one for the input places, one for the output places. So a total of \( 2 \times n \times t \times p \) operations gets performed, which is \( O(ntp) \).

\begin{algorithm}
\textbf{Algorithm 4. Inference algorithm}
\begin{algorithmic}
\Input a start place \( P_{si} \in P_s \)
\Output a set of end places \( P_e \) as the result of the inference
\Begin
Find the index of the start place \( P_{si} \) in \( M \) vector
Increase the number of tokens in the start place
Add a new token to the tail of the start place
\If the rules triggered are concurrent
Set the active rule to any one of the rules triggered
\Else
Order the triggered rules according to their scenario similarity
\EndIf
\While ordered rules (i.e., linked list of active rules) are not consumed yet
Check the transitions in \( N \) where the current place is connected as an input in input incidence matrix \( c^-_{ij} \)
\For each of those transitions
\Repeat
\If If the rules that uses the transition is a subset of the ordered rules’ head
\Then
Check the other input places of the transition in \( c^-_{ij} \)
\If they also have tokens to use
\EndIf
\EndIf
\EndWhile
\End
\end{algorithmic}
\end{algorithm}
Lemma 5.1.1. The inference algorithm based on FPN guarantees confluent rule execution.

Proof. In the inference algorithm that we give above, if the rules triggered are in the same fuzzy inference group, their execution is carried out at the same time and the total outcome is computed. On the other hand, if the rules triggered are not in the same fuzzy inference group, total ordering is achieved (i.e., rules are executed in an order) according to their $\mu_s(r)$ (the similarity to the current scenario) values. Therefore, the confluent rule execution is guaranteed. □

6. An application example: weather forecasting

Our proposed model can be applied into a number of real life applications, including traffic conjunction management, battlefield simulation, environmental monitoring and weather forecasting. We choose a portion of weather forecasting application as a case study here. In weather forecasting applications, the atmospheric elements that we consider are pressure, temperature, humidity, wind and cloudiness. Our rules consider the change on these parameters, such as value changes, direction changes or velocity changes, etc. There exists two level forecast in our model. At the first level, the expected weather can be one of clear, clear few, clouded instable or clouded stable. At the second level, we determine the expected weather event according to the output of the first level forecast together with the newly changing parameters on the atmospheric elements. In the second level, expected weather event can be any of rain, shower, snow, hail or fog.

In order to put the effect of seasons, we partition our rules of weather according to seasons. That is, each season functions like a scenario for us. By this way, we give more emphasis on some rules according to the season. Table 1 partitions our rules of expected weather and expected weather event according to seasons.

We have the following rules for the expected weather:

R1: (CLEAR WEATHER)
\begin{align*}
on & \text{pressure_change_direction is increasing,} \\
& \text{wind_direction is changing to [west OR northwest]},
\end{align*}

<table>
<thead>
<tr>
<th>Season scenarios</th>
<th>Expected weather</th>
<th>Expected weather event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summer</td>
<td>Clear, clear few</td>
<td>–</td>
</tr>
<tr>
<td>Winter</td>
<td>Clouded stable</td>
<td>Rain, fog, snow</td>
</tr>
<tr>
<td>Spring, fall</td>
<td>Clouded instable</td>
<td>Rain, hail, shower</td>
</tr>
</tbody>
</table>

Table 1
humidity_change_direction is decreasing,
cloud_cover is changing to broken sky,
cloud_base_change_direction is increasing
IF pressure_change_velocity is fast,
previous wind_direction was south,
humidity_change_velocity is fast,
previous cloud_cover was overcast,
cloud_base_change_velocity is fast
THEN expected_weather is "CLEAR"

R2: (CLEAR_FEW WEATHER)
ON pressure_change_direction is increasing,
cloud_cover is changing to [broken sky OR few],
cloud_base_change_direction is increasing,
humidity_change_direction is decreasing,
wind_direction is changing to [north OR northwest]
IF pressure_change_velocity is slow,
wind_value is breeze,
previous wind_direction was [south OR southwest],
previous cloud_cover was [overcast OR cloudy]
THEN expected_weather is "CLEAR_FEW"

R3: (CLOUDED_INSTABLE WEATHER)
ON pressure_change_direction is decreasing,
humidity_change_direction is increasing,
wind_value is changing to [medium_strong OR strong],
cloud_orientation is changing to vertical,
cloud_base_change_direction is decreasing,
temperature_change_direction is increasing
IF pressure_change_velocity is fast,
humidity_change_velocity is fast,
previous wind_value was [breeze OR medium_strong],
THEN expected_weather is "CLOUDED_INSTABLE"

R4: (CLOUDED_STABLE WEATHER)
ON pressure_change_direction is decreasing,
humidity_change_direction is increasing,
temperature_change_direction is increasing,
cloud_base_change_direction is decreasing,
cloud_orientation is changing to horizontal
IF pressure_change_velocity is slow,
humidity_change_velocity is slow,
temperature_change_velocity is slow,
cloud_base_change_velocity is slow,
w wind_direction is [southeast OR south]
THEN expected_weather is "CLOUDED_STABLE"

And we have the following rules for the expected weather event:

R5: (RAIN)
ON expected_weather is clouded_stable,
wind_direction is changing to [south OR southwest],
wind_value is changing to medium_strong,
cloud_color is changing to grey
IF temperature_value is above 3 Celsius degrees,
previous wind_direction was north,
wind_direction_change_velocity is slow,
previous wind_value was breeze,
wind_value_change_velocity is slow,
previous cloud_color was white,
THEN expected_weather_event is "RAIN"

R6: (SHOWER)
ON expected_weather is clouded_instable,
cloud_color is changing to dark_grey
IF previous wind_value was calm,
previous cloud_color was grey
THEN expected_weather_event is "SHOWER"

R7: (SNOW)
ON expected_weather is clouded_stable,
wind_direction is changing to north,
cloud_color is changing to grey
IF temperature is below 0 Celsius degrees,
previous wind_direction was south,
previous cloud_color was white
THEN expected_weather_event is "SNOW"

R8: (HAIL)
ON expected_weather is clouded_instable,
cloud_color is changing to dark
IF temperature is very_high,
previous wind_value was breeze,
previous cloud_color was grey,
THEN expected_weather_event is "HAIL"

R9: (FOG)
ON expected_weather is clouded_stable,
IF wind_value is [calm OR breeze]
THEN expected_weather_event is "FOG"

6.1. Example 1: forecasting the expected weather (active rule example)

Suppose that we are in the Winter season and we have the sensor values for times t₁ and t₂ given at Table 2 for the weather related attributes of a city, where t₁ < t₂. The results after the fuzzification of these values considering the linguistic terms taking place inside the event/condition parts of the rules are also listed in the table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Sensor values</th>
<th>Fuzzification results for event/condition evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>At time t₁</td>
<td>At time t₂</td>
</tr>
<tr>
<td>Wind value</td>
<td>6</td>
<td>6.5</td>
</tr>
<tr>
<td>Cloud orientation</td>
<td>0</td>
<td>22.5</td>
</tr>
<tr>
<td>Cloud base</td>
<td>7200</td>
<td>1200</td>
</tr>
<tr>
<td>Wind direction</td>
<td>255</td>
<td>210</td>
</tr>
<tr>
<td>Humidity</td>
<td>50</td>
<td>57</td>
</tr>
<tr>
<td>Temperature value</td>
<td>-4.5</td>
<td>-1</td>
</tr>
<tr>
<td>Pressure value</td>
<td>1003.5</td>
<td>1000</td>
</tr>
<tr>
<td>Cloud cover</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
### 6.1.1. Steps of the inference mechanism

1. **Fuzzify the event:**
   
   Evaluate $R_1$:
   
   $\mu_{\text{pressure\_change\_direction\_increasing}}(-3.5) = 0.$
   
   $\mu_{\text{wind\_direction\_west/northwest}}(210) = 0.$
   
   $\mu_{\text{humidity\_change\_direction\_decreasing}}(+7) = 0.$
   
   $\mu_{\text{cloud\_cover\_broken\_sky}}(3) = 1.$
   
   $\mu_{\text{cloud\_base\_change\_direction\_increasing}}(-5600) = 0.$
   
   $\min(0, 0, 0, 1, 0) = 0,$ $R_1$ fails.

   Evaluate $R_2$:
   
   $\min(0, 1, 0, 0, 0) = 0,$ $R_2$ fails.

   Evaluate $R_3$:
   
   $\min(1, 1, 0.5, 0.5, 1, 1) = 0.5.$ Since $0.5 > 0,$ $R_3$ gets fired.

   Evaluate $R_4$:
   
   $\min(1, 1, 1, 1, 0.5) = 0.5.$ Since $0.5 > 0,$ $R_4$ gets fired.

2. **Calculate the strength of event and check it with the rule threshold:**

   For this $\mu_{\text{winter}}(r_i)$ values for the rules are calculated using the Formula (2) in Section 3.2.

   $\mu_{\text{winter}}(R_3) = 0,$ $\mu_{\text{winter}}(R_4) = 1$

---

![Fig. 7. Solving Example 1 using FPN.](image-url)
and using the formula (1), event strengths are calculated as:
for $R_3$: $0 \times 0.5 = 0$,
for $R_4$: $1 \times 0.5 = 0.5$.
Threshold values for the rules are given as 0.1, only $R_4$ succeeds this step.

3. **Find the condition match factor:**
Evaluate $R_4$

- $\mu_{\text{pressure change velocity slow}}(-3.5) = 0.5$.
- $\mu_{\text{humidity change velocity slow}}(+7) = 0.5$.
- $\mu_{\text{temperature change velocity slow}}(+3.5) = 0.5$.
- $\mu_{\text{cloud base change velocity slow}}(-5600) = 0.5$.

So $\min(0.5,0.5,0.5,0.66) = 0.5$. Since $0.5 > 0$, $R_4$ passes this step.

4. **Find the clipping values:**
Upto this point only $R_4$ succeeds. Its antecedent matching degrees is $0.5 \times 0.5 = 0.25$.
So clipping value of 0.25 is used for the action part of $R_4$, which is an expected weather of *clouded_stable*.

5. **Find the fuzzy action:**
Maximum of the clipping values are taken, since we have $R_4$, $\max(0.25) = 0.25$. This means an expected weather of *clouded_stable* is forecasted.

Fig. 7 shows part of the FPN used for this example.

6.2. **Example 2: forecasting the expected weather event (active rule example)**

Suppose that after this forecast some more changes occur as listed in Table 3.

6.2.1. **Steps of the inference**

1. **Fuzzify the event:**
   Evaluate $R_5$:
   $\min(0.125,0,0,0.5) = 0$, $R_5$ fails.
   Evaluate $R_6$:
   $\min(0.0625,0) = 0$, $R_6$ fails.
   Evaluate $R_7$:
   $\min(0.125,0.5,0.5) = 0.125$. Since $0.125 > 0$, $R_7$ gets fired.
   Evaluate $R_8$:
   $\min(0.0625,0) = 0$, $R_8$ fails.
   Evaluate $R_9$:
   $\min(0.125) = 0.125$. Since $0.125 > 0$ $R_9$ gets fired.

### Table 3
Sensor values at times $t_2$ & $t_3$ ($t_2 < t_3$) and their fuzzification results according to the linguistic terms used in the event/condition parts of the rules

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Sensor values</th>
<th>Fuzzification results for event/condition evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>At time $t_2$</td>
<td>At time $t_3$</td>
</tr>
<tr>
<td>Wind direction</td>
<td>210</td>
<td>337.5</td>
</tr>
<tr>
<td>Wind value</td>
<td>6.5</td>
<td>6</td>
</tr>
<tr>
<td>Cloud color</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Pressure value</td>
<td>1000</td>
<td>1003.5</td>
</tr>
</tbody>
</table>
2. **Calculate the strength of event and check it with the rule threshold:**
   For this $\mu_{\text{winter}}(r_i)$ values for the rules are calculated using the Formula (2) in Section 3.2.
   $\mu_{\text{winter}}(R_7) = 1$, $\mu_{\text{winter}}(R_9) = 1$
   and using the Formula (1), event strengths are calculated as:
   for $R_3$: $1 \times 0.125 = 0.125$,
   for $R_4$: $1 \times 0.125 = 0.125$.
   Threshold values for the rules are given as 0.1, therefore $R_7$ and $R_9$ succeed this step.

3. **Find the condition match factor:**
   Evaluate $R_7$:
   $\min(1, 0.66, 0.5) = 0.5$. Since $0.5 > 0$, $R_7$ passes this step.
   Evaluate $R_9$:
   $\min(1) = 1$. Since $1 > 0$, $R_9$ passes this step.

4. **Find the clipping values:**
   Upto this point only $R_7$ and $R_9$ succeed. Their antecedent matching degrees are
   for $R_7$, $0.125 \times 0.5 = 0.0625$.
   for $R_9$, $0.125 \times 1 = 0.125$.
   So clipping value of 0.0625 is used for the action part of $R_7$, which is an expected weather event of snow
   and clipping value of 0.125 is used for the action part of $R_9$, which is an expected weather event of fog.

---

**Fig. 8. Solving Example 2 using FPN.**
5. **Find the fuzzy action:**

Maximum of the clipping values are taken, since we have $R_7$ and $R_9$, $\max(0.0625, 0.125) = 0.125$. This means an expected weather event of fog is forecasted.

Fig. 8 shows part of the FPN used for this example.

6.3. **Example 3 – Finding the expected weather (deductive rule example)**

Suppose we have the following deductive rules in the system

R1: (**CLEAR WEATHER**)

IF pressure_change_direction is increasing, wind_direction is changing to [west OR northwest], humidity_change_direction is decreasing, cloud_cover is changing to broken sky, cloud_base_change_direction is increasing pressure_change_velocity is fast, previous wind_direction was south, humidity_change_velocity is fast, previous cloud_cover was overcast, cloud_base_change_velocity is fast

THEN expected_weather is "CLEAR"

R2: (**CLEAR_FEW WEATHER**)

IF pressure_change_direction is increasing, cloud_cover is changing to [broken sky OR few], cloud_base_change_direction is increasing, humidity_change_direction is decreasing, wind_direction is changing to [north OR northwest] pressure_change_velocity is slow, wind_value is breeze, previous wind_direction was [south OR southwest], previous cloud_cover was [overcast OR cloudy]

THEN expected_weather is "CLEAR_FEW"

R3: (**CLOUDED_INSTABLE WEATHER**)

IF pressure_change_direction is decreasing, humidity_change_direction is increasing, wind_value is changing to [medium_strong OR strong], cloud_orientation is changing to vertical, cloud_base_change_direction is decreasing, temperature_change_direction is increasing pressure_change_velocity is fast, humidity_change_velocity is fast, previous wind_value was [breeze OR medium_strong],

THEN expected_weather is "CLOUDED_INSTABLE"

R4: (**CLOUDED_STABLE WEATHER**)

IF pressure_change_direction is decreasing, humidity_change_direction is increasing, temperature_change_direction is increasing, cloud_base_change_direction is decreasing, cloud_orientation is changing to horizontal pressure_change_velocity is slow, humidity_change_velocity is slow,
temperature_change_velocity is slow,
cloud_base_change_velocity is slow,
wind_direction is [southeast OR south]
THEN expected_weather is "CLOUDED_STABLE"

These deductive rules related to expected weather are converted to the following active rules internally, whose events are abstract kind:

R1: (CLEAR WEATHER)
ON raise (expected_weather)
   IF pressure_change_direction is increasing,
       wind_direction is changing to [west OR northwest],
       humidity_change_direction is decreasing,
       cloud_cover is changing to broken sky,
       cloud_base_change_direction is increasing
       pressure_change_velocity is fast,
       previous wind_direction was south,
       humidity_change_velocity is fast,
       previous cloud_cover was overcast,
       cloud_base_change_velocity is fast
THEN expected_weather is "CLEAR"

.... etc

Suppose that after time $t_2$ a query comes like $expected_weather(X)$. We use the values given at Table 2 for evaluation of this query.

Since they make no sense, we omit the steps until finding fuzzy match value.

1. Find the condition match factor:
   Evaluate $R_1$
   $\mu_{pressure\_change\_direction\_increasing}(-3.5) = 0$.
   $\mu_{wind\_direction\_west/northwest}(210) = 0$.
   $\mu_{humidity\_change\_direction\_decreasing}(+7) = 0$.
   $\mu_{cloud\_cover\_broken\_sky}(3) = 1$
   $\mu_{cloud\_base\_change\_direction\_increasing}(-5600) = 0$.
   .... etc.
   $\min(0,0,0,1,0,\ldots) = 0$, $R_1$ fails.
   Evaluate $R_2$:
   $\min(0,1,0,0,0,\ldots) = 0$, $R_2$ fails.
   Evaluate $R_3$:
   $\min(1,1,0.5,0.5,1,1,0.5,0.2,1) = 0.2$. Since $0.5 > 0$, $R_3$ gets fired.
   Evaluate $R_4$:
   $\min(1,1,1,0.5,0.5,0.5,0.66,0.5,0.5) = 0.5$. Since $0.5 > 0$, $R_4$ gets fired.

2. Find the clipping values:
   For this, the condition match factor is multiplied by the strength of event. For the abstract kind of events $\mu_{event\_strength}$ is always 1.
   for $R_1$: $1 \times 0.2 = 0.2$ (for the expected weather of clouded instable),
   for $R_2$: $1 \times 0.5 = 0.5$ (for the expected weather of clouded stable).

3. Find the combined fuzzy action:
   We take the maximum of the clipped actions: $\max(0.2,0.5) = 0.5$, which means expected weather of clouded stable is chosen.

Fig. 9 shows part of the FPN used for this example.
7. Conclusion

In this paper, we introduce a FPN model for fuzzy rule-based reasoning. This includes the transformation of fuzzy rules into FPN, together with their reasoning. We can model active and deductive rules along with the compositions (either event or condition) in our FPN. Besides, the functionalities of transitions are extended so that they are capable of performing some required computations in addition to the sup-min composition provided by other studies. Since we employ colored Petri nets, parameter passing is provided through the Petri net. This provides the values of conditions and actions to be calculated from the parameters of events.

By constructing the FPN, we inherently obtain the triggering graph information. Therefore, without performing additional work, we can easily check the termination property of our system. When the assumptions and theories regarding the termination analysis were put forward, event compositions were neglected by the previous studies [1–3,11,18]. On the other hand, we can handle compositions by the structures already provided by our FPN. In addition, our fuzzy inference algorithm working on our FPN assures confluent rule executions. This assurance comes from the fact that our inference algorithm provides a total order within the rules, using the similarity of the rules to the current active scenario.

The previous studies using FPNs do not model active rules. Also, they do not investigate the properties of their systems by using the FPNs. We aim to fill these gaps and propose a fuzzy Petri net model to represent knowledge and the behavior in an intelligent object-oriented database environment which integrates fuzzy, active and deductive rules with database objects. Fig. 10 compares the previous studies using FPNs with our study.

We think that the FPN introduced in this study can be used to model sensor network applications. As pointed out in [31], sensor networks require a computational model for handling events, event detections,
and action triggers. In many sensor network applications, data coming from sensor nodes may not be precise [5,31]. Sensor readings include uncertainty due to noise, receipt of the sensor to environmental conditions, possible failure, etc. In addition, exact queries may not be the type of queries that can accurately convey the demands of all users of sensor network applications. Therefore, fuzzy and approximate queries such as “give an alarm to the regions when there is a strong possibility of flooding because of high level of rainfall”, need to be supported. This means that a model to be used in such an application should have a capability of reasoning under uncertain and fuzzy information. Our fuzzy Petri net model can process uncertain sensory data and handle fuzzy and approximate queries. Furthermore, the techniques and solutions provided in this paper for the static analysis of rule bases can be utilized to eliminate the undesirable behaviors of the modeled system.

Our study can be extended in several ways. Currently, in our model, switching between the scenarios is determined by the user. However, a model can be developed for automatic switching between the scenarios. Furthermore, we assume all coupling modes being as immediate. The model can be extended with other coupling modes. Finally, temporal dimension can be incorporated to the model.

References

Burcin Bostan-Korpeoglu received her Ph.D. degree in Computer Engineering from Middle East Technical University, Ankara, Turkey, in 2005. She received her M.S. and B.S. degrees also from the Department of Computer Engineering of Middle East Technical University, in 1997 and 1994 respectively. She worked as a research assistant at the same department between 1994 and 1997. Since 1997 she has been working at the Payment Systems Department of the Central Bank of Turkey. Between April 1998 and June 1999, she has participated in the analysis, design and development of the second generation Electronic Funds Transfer System (EFT) and the Electronic Securities Transfer System of Turkey, at the Logica CMG company, UK, London. Her research interests include fuzzy database modeling, object-oriented databases, intelligent database systems and sensor network databases.
Adnan Yazıcı is currently a Professor of the Department of Computer Engineering at Middle East Technical University, Ankara, Turkey. He received his Ph.D. in Computer Science 1991 from the Department of Electrical Engineering and Computer Science (EECS) at Tulane University, New Orleans, USA. He had been a visiting professor in the department of EECS at Tulane University between 1998 and 2000. His current research interests include intelligent database systems, fuzzy database modeling, database indexing, spatio-temporal databases, multimedia & video databases, and sensor network databases. Prof. Yazıcı has published more than 90 international technical papers and co-authored a book titled as Fuzzy Database Modeling (by Springer Verlag) and edited an LNCS on Computer and Information Sciences. He is a senior member of IEEE and has received one of the Young Investigator Awards bestowed by the Parlar Foundation, for the year 2001. He is the conference co-chair of IEEE Conference on Data Engineering (ICDE 2007) and was the conference and PC chair of 18th International Symposium of Computer and Information Sciences (ISCIS 2003) and PC co-chair of ISCIS 1996. He serves as a committee member and reviewer in various international journals and conferences.