International Journal of Computational Intelligence Systems, Vol. 5, No. 1 (February, 2012), 173-196

A LINGUISTIC TRUTH-VALUE TEMPORAL REASONING (LTR) SYSTEM AND ITS APPLICATION TO THE DESIGN OF AN INTELLIGENT ENVIRONMENT

Zhirui Lu [1] Juan Augusto [2] Jun Liu [2] Hui Wang [2]

[1] *School of Computing and Mathematics, University of Ulster, Jordanstown, Shore Road Newtownabbey, BT37 0QB, United Kingdom*

E-mail: Lu-Z1@email.ulster.ac.uk

[2] *School of Computing and Mathematics, University of Ulster, Jordanstown, Shore Road Newtownabbey, BT37 0QB, United Kingdom*

E-mail: j.augusto/j.liu/h.wang@ulster.ac.uk

Received 10 May 2010
Accepted 16 December 2011

Abstract

This paper focuses on a linguistic-valued temporal logic based reasoning formalism for dynamically modelling and merging information under uncertainty in some real world systems where the state of a system evolves over time and the transition through states depends on uncertain conditions. We provide forward and backward reasoning algorithms which, respectively, support simulation and query answering. These algorithms are then explained through several examples based on Smart Homes applications.

Keywords: Logic; Decision-making; Uncertainty; Temporality; Forward Reasoning; Backward Reasoning

1. Introduction

Decision-making is an essential component in many real world systems and has been the subject of research for many years. One of the common problems is how to make the best possible decision based on uncertain or incomplete information in dynamic environments. This problem exists in many applications, such as smart homes, risk management, disaster monitoring and management, weather forecast, and stock market analysis. These real world systems are characterized by uncertainty, where the states and transition between states are not deterministic due to various reasons, and temporality, where the transition between states is dependent not just on the current state but also on previous states. Solving decision-making problems in such systems depends on the formal mechanisms used, and logic is one of them. To handle both temporality and uncertainty, some non-classical logic systems have been extensively investigated, *e.g.*, many-valued/fuzzy logic and temporal logic.

Substantial research has been done to handle uncertainty (Ref. 1, 2, 3, 4, 5, 6, 7, 8) and temporality (Ref. 9, 10, 11, 12, 13, 14, 15) through logic approaches. In real applications, uncertainty and tem-

porality may co-exist, so the ability to reason about both time and uncertainty is very important and desirable in a decision support system. For that reason we have developed a computational system which can handle both elements.

One approach to decision making under uncertainty and temporality is by combining many-valued and temporal logics. There is already work in this approach, which includes (Ref. 16, 17, 18, 19). Detailed review of these work can be found in Section 2. In our recent work (Ref. 20), a novel reasoning framework for decision making under uncertainty and temporality, which integrates many-valued logic (mainly focus on Łukasiewicz logic) and temporal logic (Ref. 21) was presented. Difference to other proposals, our many-valued approach allows the consideration of uncertainty not only on states but also on rules. We adopted a many-valued logic with truth values in the interval [0, 1] by using the Łukasiewicz implication operator, *i.e.*, Łukasiewicz logic $L[0, 1]$ (Ref. 1, 7, 22), where 0 means 'false' and 1 means 'true'. For the temporal logic part, we follow the work of (Ref. 21), which provided a simple stratified causal rules to reason about the dynamic aspects of a system with little computational cost.

Consistently with our previous work, see (Ref. 20), in this paper we assume the truth-value degree is taken from a linguistic term set instead of a numerical value in the interval [0, 1]. This approach is called a linguistic-valued based approach, where the symbolic reasoning is focused on linguistic terms. This approach is more natural than a simple number in [0,1], especially when it is impossible or unnecessary to obtain more accurate values, and it also makes the reasoning qualitative in nature. Its application is beneficial because it introduces a more flexible framework for representing information in a more direct and suitable way when it is not possible to express it accurately. Thus, the burden of quantifying a qualitative concept is eliminated and the systems can be simplified. The basic definitions for syntax, semantic, inference rules, and the related theorems over the defined logic system is provided a theoretical foundation on the algorithms that we have developed.

The paper is organized as follows. In Section 2, some relevant work is reviewed. In Section 3, two simple but realistic scenarios are provided. Section 4 gives an overview of our combined propositional logic system including inference rules. The forward and backward reasoning algorithms are detailed in Sections 5 and 6 respectively, along with the corresponding solutions for the scenarios introduced in Section 3. Section 7 summarizes our contributions.

2. Related Work

Logic is used in most intellectual activity, but is studied primarily in the disciplines of philosophy, mathematics, and computer science (Ref. 23). However, after developing in many years, many elements have been introduced into logic, such as uncertainty and temporality. Accordingly classical logic has also been extended into non-classical logics to handle such different elements, to solve more specific real world problem, such as fuzzy logic (Ref. 23, 24, 25, 26, 27, 28) and temporal logic (Ref. 13, 29, 30, 31), etc.

2.1. Fuzzy Logic

As mentioned previously, uncertainty and temporality are two important features of decision-making tasks. Uncertainty means that, due to some reasons, the information available may not be always accurate, and noise may occur during data transmission. Consequently, the collected information may be uncertain. To handle the uncertainty through logic approaches, many works have been done by researchers such as many-valued logic (including fuzzy logic and lattice-valued logic), which, among others, can be referred to (Ref. 1, 2, 3, 4, 5, 6, 7, 25).

Since not all the real decision-making problems can be represented only with 'true' or 'false' by logic systems, Łukasiewicz provided the three-valued ('true', 'unknown', 'false') logic system and its implication calculi (Ref. 32), which introduced a new research area for logic, logic with uncertain information. Fuzzy logic is one of such logic systems used to handle the uncertain information in decision-making problem. "Fuzzy logic" provides two different meanings, wide (*FLw*) and narrow (*FLn*). Ac-

cording to Zadeh's work (Ref. 24, 27), it provided the following useful distinction:"In a narrow sense, *FLn* is a logic system which aims at a formalization of approximate reasoning. In this sense, *FLn* is an extension of many-valued logic, $\cdots$." Hajek indicated that the calculi of the many-valued logic (Ref. 5) is the base of fuzzy logic in the narrow sense (Ref. 3).

One of very important fuzzy logic (*FLn*) branches is the Łukasiewicz truth-functional logic (Ref. 33). This logic inherits the calculi from Łukasiewicz implication, and extends it from three values into [0,1] interval. Łukasiewicz logic is a well known many-valued logic studied in numerous papers on fuzzy and many-valued logic. Pavelka (Ref. 1) incorporated internal truth value in the language, established a fuzzy propositional logic system whose truth value set is an enriched residuated lattice and proved a lot of important results about its axiomatizability. More importantly, he showed that the only natural way of formalizing fuzzy logic (or the only axiomatizable fuzzy logics) for truth values in the unit interval [0, 1] or on a finite chain is by using the Łukasiewicz implication operator or some isomorphic forms of it. A logic defined by a semantic is axiomatizable if there is a set of axioms and inference rules which are sound and complete with respect to that semantics, *i.e.*, syntactic truth equals semantical truth. Not only developing the *FLn*, many researchers also did their work in Fuzzy Sets Theory (*FST*) (Ref. 2, 25, 34, 35).

2.2. Temporal Logic

Temporal logic is one of many mechanisms available to reason about time. Literature in this area is vast and we are not aiming to provide an exhaustive review of this concept here, for some samples of the various systems available see (Ref. 9, 10, 11, 12, 13, 14, 15). One well known system to reason about time is Allen's Interval Temporal Logic (*ITL*), and provided seven relations and their inverses between them (Ref. 29, 30, 36):

1. $Before(i_1, i_2)$: the end of i_1 is previous to the beginning of i_2.
2. $Meets(i_1, i_2)$: i_2 begins exactly when i_1 finishes.
3. $Overlap(i_1, i_2)$: i_1 starts before i_2 and i_1 finishes before i_2.
4. $Starts(i_1, i_2)$: i_1 starts simultaneously with i_2 but it finishes sooner.
5. $During(i_1, i_2)$: i_1 starts after and ends before i_2.
6. $Finishes(i_1, i_2)$: i_1 starts later than i_2 but they finishes at the same time.
7. $Equal(i_1, i_2)$: i_1 starts and finishes at the same time as i_2.

In Allen's work (Ref. 36), a transitivity table is provided, which showed how to infer the temporal relations between time intervals *i* and *k*, where there are relations between *i* and *j* and the relation between *j* and *k* from the knowledge base. This basic relations between intervals used by Allen has been extended by (Ref. 37), which introduced a new concept called *semi-intervals*. The transitivity table extended by the author contained 29 relations.

Another type of temporal logic is based on the use of temporal operators (Ref. 38):

- *X* next: represented as $\bigcirc s$, which means *s* has to hold at the next time state.
- *G* always: represented as $\Box s$, which means *s* has to hold at all time states.
- *F* eventually: represented as $\diamondsuit s$, which means *s* has to hold at some time states.
- *U* until: represented as $s\ U\ s_1$, which means *s* has to hold until s_1 holds.
- *R* release: represented as s $\Re$ s_1, which means at the first position in which *s* is true (if such position occurs), s_1 ceases to be true; it is required to be true until release occurs.

One common problem to most of these well known options is computational complexity, their expressiveness comes at a cost. The system we propose in a later section is mainly focused on the operator "next" and is designed to be simple and computationally less demanding.

2.3. Fuzzy and Temporal Logic

Real world applications often require de combination of uncertain and temporal reasoning, therefore researchers have proposed several options that can accommodate both concepts. One such approach is the combination of fuzzy logic and temporal logic.

For example, Escalada-Imaz (Ref. 16) provided a many-value temporal reasoning method which extended the truth value of a state to indicate the uncertainty of an element for real-time control systems. In the uncertain part, it used the many-valued calculi to handle the uncertain information and provided an interval [0,1] to indicate the truth value degree of state, where 0 means false and 1 means truth. The calculation of the truth value degree of the state in rules should be followed the many-valued calculi. In temporality part, it implemented a valid period for state, and suggested that the valid period of conclusion part in the rule should only be the interaction of the valid period in condition part, which is one of interval temporal logic applications. In this reasoning, it assumed that rules are always true, without any uncertainty, however, in some real world problems, the provided rules may not always be trusted, such that, it may occur uncertainty in rules as well when an expert is unable to establish/predict a precise correlation between premise and conclusion but only with degrees of certainty.

Cárdenas Viedma *et al.* (Ref. 17) extended temporal constraint logic (*TCL*) into fuzzy temporal constraint logic (*FTCL*) which allows *FTCL* to handle uncertainty of time representation such as 'approximately 90 minutes', such that, within the implementation of fuzzy set into time representation, it improves the ability of system for handling the decision-making problem which contains uncertain time issue, however, the uncertainty of states and rules was not the main issue and not addressed in (Ref. 17).

There are some other combined logic systems for real-time decision-making problem, such as, Mucientes *et al* (Ref. 18) suggested a fuzzy temporal reasoning method for robot control; and Schockaert and De Cock (Ref. 19) extended classic time interval as used in temporal reasoning into fuzzy-time interval to make the reasoning system more flexible, the proposed system is similar to the one in (Ref. 17), but it provided more mathematical definitions, lemmas and theorems which built up a theoretical foundation.

The brief literature review above shows that some of decision-making systems combining with fuzzy logic and temporal logic still have some weakness because they may face to different problems and requirements. Some of them only considered the uncertainty of state and extend the truth value of state and gave the valid time interval for conclusion part, some of them just focused on the uncertainty of time issue. However, from the real world experience, we know that the rule may not always be true in some special case, such that, the uncertainty may not only exist in states; it still possibly exists in rules. How to handle such a problem is one of the main objectives in this paper. In addition, those works always considered the numerical truth-value degree, which also limited the applicability of the system for qualitative reasoning and decision making, this, however, is quite common in human decision making process.

3. Problem Description: Two Realistic Scenarios

In this section, two scenarios are presented to illustrate the problem considered in this paper. The proposed reasoning framework aims to help deal with such problem within uncertain and time constrained situations to make rational decisions.

3.1. Scenario 1: Health Care in Smart Home Environment

A Smart Home can be described as a house that is supplemented with technology, for example sensors and devices, in order to increase the range of services provided to its occupants reacting in an intelligent way relies on data gathered by sensors, having to deal with the storage, retrieval, and processing of uncertain and time constrained data (Ref. 39).

We assume that a system is used to monitor the status of a patient to decide whether the patient's activities are normal or if help is needed

(Ref. 40). We consider the following states: *inBed* representing the sensor detecting the patient state is in that specific position, *inBedTime* specifies the time the patient is expected to be in bed, *standing*/*moving*/*sitting*/*laying* representing that activity of the patient has been detected, *mt20u* is used to capture the passage of 20 units of time, *safe*/*undesirable* represent the belief that the person is in one of those situations. *carerVisits*/*carerCalls* represents that currently the carer has been required to take one of those actions, *carerGivesOK* is the input given by the carer through the system interface denoting s/he has checked or assisted the patient. All of such states and rules given below will contain a truth value, and we define the truth value set $V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$, which indicates the levels of trust: {v_0=*false*, v_1=*almost false*, v_2=*probably false*, v_3=*unknown*, v_4=*probably true*, v_5=*almost true*, v_6=*true*}.

In this scenario, we assume that the status of the occupant is 'safe' and 'desirable' at start. It is bed time, the occupant goes to bed and sit there. The sensor catches that the occupant sit in bed for a long time (20 units time setting in this scenario), which is considered may be an indication of a problem, then the system will call the carer. After the carer checked the situation, he/she gives an answer to the system whether the situation is or not undesirable. We represent this with a sequence of events and causal rules as follows:

Independent States: $\pm inBed$, $\pm standing$, $\pm moving$, $\pm sitting$, $\pm laying$, $\pm inBedTime$, $\pm mt20u$, $\pm carerGivesOK$, $\neg carerVisits$, $\neg carerCalls$

Dependent States:
$\pm safe, \pm undesirable, +carerVisits, +carerCalls$

Same-time Rules:

Stage 1:

1. $(\neg inBed \wedge standing \rightarrow safe), v_6$
2. $(\neg inBed \wedge moving \rightarrow safe), v_6$
3. $(\neg inBed \wedge sitting \rightarrow \neg safe), v_6$
4. $(\neg inBed \wedge laying \rightarrow \neg safe), v_6$
5. $(inBed \wedge \neg inBedTime \rightarrow undesirable), v_6$
6. $(inBed \wedge sitting \wedge inBedTime \rightarrow safe), v_6$
7. $(inBed \wedge sitting \wedge inBedTime \wedge mt20u \rightarrow undesirable), v_6$

Stage 2:

8. $(\neg safe \rightarrow carerVisits), v_6$
9. $(undersirable \rightarrow carerCalls), v_6$

Next-time Rules:

(10) $(\neg safe \wedge carerGivesOK \rightarrow \bigcirc safe), v_6$

(11) $(undesirable \wedge carerGivesOK \rightarrow \bigcirc \neg undesirable), v_6$

Events:

- occurs($ingr(\neg inBed)$, 1*, v_5)
- occurs($ingr(sitting)$, 5*, v_4)
- occurs($ingr(inBed)$, 5*, v_6)
- occurs($ingr(sitting)$, 7*, v_5)
- occurs($ingr(mt20u)$, 28*, v_5)
- occurs($ingr(carerGivesOK)$, 31*, v_6)

Initial Setting:
I_c={$(inBed, 0, v_0)$, $(standing, 0, v_0)$, $(sitting, 0, v_0)$, $(moving, 0, v_0)$, $(laying, 0, v_0)$, $(mt20u, 0, v_0)$, $(inBedTime, 0, v_5)$, $(carerGivesOK, 0, v_0)$, $(safe, 0, v_6)$, $(undesirable, 0, v_0)$, $(carerVisits, 0, v_0)$, $(carerCalls, 0, v_0)$} and $\lambda = v_3$.

In this scenario, we wonder whether the system will call the carer automatically if the system catches that the patient is not safe or undesirable, and after the carer visits and provides '*OK*' information to the system, the system should reset the status of the patient back to safe or desirable.

Notice that: $(\neg inBed \wedge standing \rightarrow safe), v_6$ represents that the truth value of rule $\neg inBed \wedge standing \rightarrow safe$ is v_6; occurs($ingr(\neg inBed)$, 1*, v_5) represents that, there is an event occurs between time slot 1 and 2, which changes the truth value of $\neg inBed$ into v_5, and $ingr(\neg inBed)$ is to denote an ingression to $\neg inBed$; $(inBed, 0, v_0)$ represents that, the truth value of *inBed* at $time = 0$ is v_0; and λ is the trigger level.

3.2. Scenario 2: Cooker Monitoring in Smart Home Environment

Consider again a scenario where the task is to model a kitchen monitored by sensors in a Smart Home system. Let us assume the cooker is on (*cookerOn*, a sensor detecting cooker being activated), but the motion sensor is not activated ($\neg atKitchen$, *atKitchen* is a sensor detecting location of the patient in the kitchen). If no motion is detected after more than a number n units of time (here to simplify we assume n=3, *umt3u*), then we consider the cooker is unattended (*cu*). In this case, at the next unit of time, the alarm will be on (*alarmOn*) to notify the user. In this scenario, cooker and user are both monitored by sensors, but there may be a problem with these sensors which can not return accurate information about the status of cooker or position of the user, and some causal relationships may not be always certain. For example, due to the likely malfunction of sensor *atKitchen*, we can only assume that the patient is in the kitchen with *e.g.*, 80% certainty, or with 'high' confidence. What we want to know is whether the alarm being on or off can be automatically inferred under such uncertain and dynamic situation. The proposed reasoning framework aims to help dealing with such uncertain and time constrained situations to make rational decisions. To identify the truth-valued level of the status of states, we still used the 7 tiered truth-value set V given in Scenario 1.

According to the above description, we can have the following assumptions for the scenario:

Independent States:

$cookerOn, \pm atKitchen, umt3u$

Dependent States:

$\pm cu, \pm hazzard, \pm alarmOn, \neg umt3u, \neg cookerOn$

Same-time Rules

Stage 1:

1. $(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu), v_6$
2. $(\neg cookerOn \rightarrow \neg alarmOn), v_6$
3. $(\neg cookerOn \rightarrow \neg hazzard), v_6$
4. $(\neg cookerOn \rightarrow \neg umt3u), v_6$
5. $(\neg cookerOn \rightarrow \neg cu), v_6$

Stage 2:

6. $(cu \rightarrow alarmOn), v_6$
7. $(cu \rightarrow hazzard), v_6$

Next-time Rules

(8) $(alarmOn \rightarrow \bigcirc(\neg cookerOn)), v_6$

Events:

- occurs($ingr(atKitchen)$, 0*, v_5)
- occurs($ingr(cookerOn)$, 0*, v_5)
- occurs($ingr(\neg atKitchen)$, 1*, v_5)
- occurs($ingr(umt3u)$, 4*, v_5)

Initial Setting:

$I_c = \{(cookerOn, 0, v_0), (atKitchen, 0, v_0), (umt3u, 0, v_1), (cu, 0, v_1), (hazzard, 0, v_1),$ (alarmOn, 0, v_1)$\}$ and $\lambda = v_3$.

This scenario is used to test whether the system will turn on the alarm and shut down the cooker automatically if the cooker is unattended.

4. Outline of Linguistic-valued Temporal Propositional Logic Systems

This section briefly outlines the formal linguistic-valued temporal propositional logic framework as introduced in (Ref. 39, 41), only some necessary concepts and notations are reviewed, more details please referred to (Ref. 20).

4.1. Syntax and Semantics

Following the notations given in (Ref. 21), we assume a set of atomic states, denoted as S, and $s_1, s_2, \cdots, s_n \in S$. We also assume a set of rules, denoted as R, characterizing relationships amongst states of the system. We provide $Q = S \cup R$ to refer to as the full set of all states and rules. Each atomic state contains two status, each positive atomic state s being paired with its negation $\neg s$. $s_1 \wedge s_2$ denotes the (non-atomic) state holds if and only if s_1 and s_2 both hold. S_I denotes the set of the independent atomic states which does not depend on other states and S_D

denotes the set of the dependent atomic states. An independent state can only be initiated by the occurrence of initiating events. We also propose a set of time slot, denoted as T, and $t_1, t_2, \cdots, t_n \in T$, and every time slot is independent to each other. In this system, it only considers two kinds of rules:

Same-time rules: $s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow s$ and

Next-time rules: $s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow \bigcirc s$,

where each s_i is an atomic state and $s \in S_D$. $s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow s$ represents the influence of $s_1 \wedge s_2 \wedge \cdots \wedge s_n$ over s, and $s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow \bigcirc s$ represents delayed (next time) influence of $s_1 \wedge s_2 \wedge \cdots \wedge s_n$ over s. Same-time rules are required to be stratified, which means they are ordered in such a way that the states in a rule are either independent or dependent on states which are heads of rules in previous levels of the stratification, see more details on this in (Ref. 21). Time is represented as a discrete series of atomic instances, labelled by natural numbers (Ref. 20).

The semantic definition follows a universal algebraic point of view, as in (Ref. 1, 7, 22).

Definition 1. Let X be a set of propositional variables, $TY = V \cup \neg, \rightarrow$ be a type with $ar(\neg) = 1$, $ar(\rightarrow) = 2$ and $ar(a) = 0$ for every $a \in V$. The propositional algebra of the many-valued propositional calculus on the set of propositional variables is the free T algebra on X and is denoted by *LP(X)* (Ref. 20).

Note that V and *LP(X)* are the algebras with the same type TY, where $TY = V \cup \neg, \rightarrow$. Moreover, note that $\wedge$, $\vee$, and $\otimes$ can all be expressed by $\neg$ and $\rightarrow$, so $p \wedge q, p \vee q, p \otimes q \in LP(X)$ if $p, q \in LP(X)$. In addition, notice that $Q = S \cup R \subseteq LP(X)$.

Definition 2. Let T be a time set and V be a truth value set, if a mapping $\gamma : LP(X) \times T \rightarrow V$ satisfies the following properties:

1. For any $\alpha \in V, t \in T$, $\gamma(\alpha, t) = \alpha$;
2. γ is a propositional algebra homomorphism with respect to the first argument;
3. For any $t_1, t_2 \in T$, if $t_1 \neq t_2$, then $\gamma(*, t_1) \neq \gamma(*, t_2)$, where * means any state belong to S.

Then γ is called a temporal valuation of *LP(X)*.

Definition 3. Let $p \in LP(X)$, and $\alpha \in V$. If there exists a temporal valuation γ such that $\gamma(p) \geqslant \alpha$ at time t, then p is said to be α-satisfiable at time t. If $\gamma(p) \geqslant \alpha$ for every temporal valuation γ of *LP(X)* at time t, then p is said to be valid with the truth-value level α at time t. If α is equal to the maximum value of V at time t, then p is valid in time t.

4.2. Łukasiewicz Linguistic Calculi and Temporal Logic

In this paper, we present a reasoning framework for decision making under uncertainty and temporality, which integrates fuzzy logic (mainly based on Łukasiewicz linguistic truth-value logic) and temporal logic (*LTL*, mainly based on the work in (Ref. 22)). Different from other research works, this reasoning scheme allows the consideration of uncertainty not only on states but also on rules. We apply the linguistic computational symbolic approach, acts by direct computation on linguistic values (Ref. 41, 42, 43, 44, 45), indicates that the set of linguistic truth value as V and $v_0, v_1, \cdots, v_m \in V$. Łukasiewicz logic is applied because of its axiomatizability. According the truth-value algebra is called as Łukasiewicz linguistic truth-valued implication algebra and is defined as follows (Ref. 41).

Definition 4. Let $V = \{v_i\}$, where $i = 0, \cdots, m-1, m$, be a finite and totally ordered linguistic term set. Any label, v_i, represents a possible value for a linguistic variable. Moreover, L must have the characteristics below:

1. The set is ordered: $v_i \leqslant v_j$ if $i \leqslant j$;
2. There is a negation operator: $(\neg v_i) = v_j$ such that $j = m - i$;
3. There is a maximization operator: $Max(v_i, v_j) = v_j$ if $v_i \leqslant v_j$;
4. There is a minimization operator: $Min(v_i, v_j) = v_i$ if $v_i \leqslant v_j$;
5. There is an implication operator, : $\rightarrow$: $V \times V \rightarrow V$ defined by $v_i \rightarrow v_j = v_{min(m, m+j-i)}$, where $i, j \in \{0, ..., m\}$;

6. There is a product operator (called Łukasiewicz product): $: \otimes : V \times V \to V$ defined by $v_i \otimes v_j = v_{max(0,i+j-m)}$.

$(V, \leqslant, \to, \neg)$, in short V, forms an algebra called as Łukasiewicz linguistic truth-valued implication algebra (Ref. 41).

Notice that, we say $v_k = v_i \to v_j = v_{m+j-i}$, which means that for a linguistic value v_k, where $k = m + j - i$, $v_i, v_j, v_k, v_m \in V$ and $i, j, k = 0, 1, \cdots, m$.

The temporal logic element applied in our reasoning framework is simpler than those approaches cited above. We follow the work of (Ref. 21), which provided a simple stratified causal rule.

4.3. Inference Rules for Reasoning Algorithms

This section explains how reasoning with uncertainty in a dynamic system is performed in our system. The definitions below cover all the possible ways the system can use to compute the value of a state based on other states and incoming events (which can also alter the truth value of states). Assume that $s_1, s_2, ..., s_n \in S$, $t_1, t_2, ..., t_m \in T$, $v_0, v_1, v_2, ..., v_r \in V$, and $s_1 \wedge s_2 \wedge ... \wedge s_n \to s$, $s_1 \wedge s_2 \wedge ... \wedge s_n \to \bigcirc s \in R$, and set the truth value of state s_i to be v_i, and the truth value of rule $s_1 \wedge s_2 \wedge ... \wedge s_n \to s$, and $s_1 \wedge s_2 \wedge ... \wedge s_n \to \bigcirc s$ to be v_τ and v_δ respectively.

Definition 5. Same-time Rule (R_s):

$$(s_1 \wedge s_2 \wedge ... \wedge s_n \to s,\ v_\tau)$$

which means that if s_1 and s_2 and ... and s_n holds, then the truth-value level stating that s holds at the same time slot is v_τ.

Definition 6. Same-Time Linguistic-Valued Temporal Modus Ponens Rule (*s-LTMP*):

$$\frac{((s_1,t,v_1),...,(s_n,t,v_n)),(s_1 \wedge s_2 \wedge ... \wedge s_n \to s, v_\tau)}{(s,t,v_\alpha \otimes v_\tau)}$$

where $v_\alpha = Min(v_1, v_2, ..., v_n), v_\tau \in V$.

Definition 7. Backward Same-Time Linguistic-Valued Temporal Rule (*s-BLTR*):

$$\frac{Query(s,t,v_\beta),(s_1 \wedge s_2 \wedge ... \wedge s_n \to s, v_\tau)}{(Min(s_1,t,v_{r-\tau+\beta}),\cdots,Min(s_n,t,v_{r-\tau+\beta}))}$$

where $v_\beta, v_\tau, v_r \in V$, $v_\beta \leqslant v_\tau$ and v_r is the maximum value in V.

Definition 8. Next-time Rule (R_n):

$$(s_1 \wedge s_2 \wedge ... \wedge s_n \to \bigcirc s,\ v_\delta)$$

which means that if $s_1 \wedge s_2 \wedge ... \wedge s_n$ holds, then the truth-value level stating that s holds at the next time slot is v_δ. The symbol '$\bigcirc$' indicates that the change on the truth value of state s will be delayed and take effect in the next time unit.

Definition 9. Next-Time Linguistic-Valued Temporal Modus Ponens Rule (*n-LTMP*):

$$\frac{((s_1,t,v_1),...,(s_n,t,v_n)),(s_1 \wedge s_2 \wedge ... \wedge s_n \to \bigcirc s, v_\delta)}{(s,t+1,v_\alpha \otimes v_\delta)}$$

where $v_\alpha = Min(v_1, v_2, ..., v_n), v_\delta \in V$.

Definition 10. Backward Next-Time Linguistic-valued Temporal Rule (*n-BLTR*):

$$\frac{Query(s,t,v_\beta),(s_1 \wedge s_2 \wedge ... \wedge s_n \to \bigcirc s, v_\delta)}{(Min(s_1,t-1,v_{r+\beta-\delta}),...,Min(s,t-1,v_{r+\beta-\delta}))}$$

where $v_\beta, v_\delta, v_r \in V$, $v_\beta \leqslant v_\delta$ and v_r is the maximum value in V.

Unlike dependent states, independent states can only be initiated by the occurrence of the initiating events. We write $(Ingr(s), t, v_\alpha)$ to denote an ingression to s, at t, with the truth-value $v_\alpha \in V$.

Definition 11. Assume events occur instantly and an instant between t and $t+1$ is denoted as t^*. $Occurs(Ingr(s), t^*, v_\alpha)$ is used to express an event occurrence such as:

$$Occurs(Ingr(s), t^*, v_\alpha) \leftrightarrow (s,t,v_\beta) \wedge (s,t+1,v_\alpha),$$

where $s \in S_I$ and $v_\alpha, v_\beta \in V$.

Definition 12. Event Occurrence Rule (*EOR*):

$$\frac{(s,t-1,v_\beta), Occurs(Ingr(s), t^*, v_\alpha)}{(s,t,v_\alpha)}$$

Definition 11 and 12 provide a calculus for event, such that, while an event occurs, then the truth value of state should be changed by *EOR*.

Definition 13. (**Persistency rule**) An atomic state is said to follow the persistency rule if there is no event or rule that can be applied to affect the truth value of state s at time t, and at time $t+1$, it inherits the same truth value as that one at time t, i.e., $(s,t,v_\beta) \to (s,t+1,v_\beta)$, where $s \in S$ and $v_\beta \in V$.

Definition 14. Forward Persistency Rule (*FPR*):

$$\frac{(s,t-1,v_\beta), Persistence(s,t-1,v_\beta)}{(s,t,v_\beta)}$$

The *FPR* is used to handle the state which is not affected by any event or rule in a specific time slot, such that, from the inference rule, we can see that it just inherits the truth value from the previous time slot.

Definition 15. Backward Persistency Rule (*BPR*):

$$\frac{Query(s,t,v_\beta), Persistence(s,t-1,v_\beta)}{Min(s,t-1,v_\beta)}$$

From the above definitions, there are four different forward Modus Ponens inference rules and three different backward inference rules in our logic system. At a meta-logical level the following priority applies in their application:

- Forward: $EOP >> s-LTMP >> n-LTMP >> FPR$,
- Backward: $EOP >> s-BLTR >> n-BLTR >> BPR$

which means they are considered from left to right until one can be applied.

4.4. Logical Calculus

According to the definition of the Łukasiewicz linguistic truth-valued implication algebra, *s-LTMP* and *n-LTMP*, it shows that if we have a rule $(s_1 \wedge s_2 \wedge ... \wedge s_n \rightarrow s, v_\tau)$, and we also know $v_\alpha = Min(v_1, v_2, ..., v_n), v_\tau \in V$, where $v_1, v_2, ..., v_n$ is the truth value of $s_1, s_2, ..., s_n$ at time t, then according to the Łukasiewicz linguistic truth-valued implication algebra, the truth value of s should be $v_\alpha \otimes v_\tau$, and such calculi process we called it forward reasoning calculi. However, according to $s-BLTR$ and $n-BLTR$, what it requires is that, we have known the result (v_β) of the conclusion (s) and the rule is given $(s_1 \wedge s_2 \wedge ... \wedge s_n \rightarrow s, v_\tau)$, then we want to know what is the feasible condition $v_\alpha = Min(v_1, v_2, ..., v_n)$ of the rule to match such case, then this calculi process is called backward reasoning calculi. Hence, we have the following theorem for the backward reasoning calculi according to the backward calculation of Łukasiewicz linguistic truth-valued implication algebra to support such calculi process:

Theorem 1. *Suppose* $< V = \{v_0, v_1, \ldots, v_m\}, \wedge, \vee, \neg, \rightarrow, \otimes >$ *is a Łukasiewicz implication algebra on a finite chain truth value set* V, $v_\alpha, v_\beta, v_\theta \in V$, *and* $v_\alpha \otimes v_\beta = v_\theta$, *where* v_α *is the truth value of condition (body of rule),* v_β *is the truth value of rule, and* v_θ *is the truth value of queried state (head of rule). Then we can have* $v_\alpha \leqslant v_{m-\beta+\theta}$ *and* $v_\theta \leqslant v_\beta$.

Proof.

1. According to the definition 4, we know that

   $$v_\theta = Max\{v_0, v_{\alpha+\beta-m}\},$$

   so if we only consider the index calculation, then we have

   $$\theta = Max\{0, \alpha+\beta-m\}.$$

   Such that,

   $$\theta = \{ \begin{array}{c} 0, if, \alpha+\beta-m \leqslant 0 \\ \alpha+\beta-m, otherwise \end{array}$$

   If $\alpha+\beta-m \leqslant 0$, then $\theta = 0$, then $\alpha+\beta-m-\theta \leqslant 0$. Hence $\alpha \leqslant m-\beta+\theta$;

   If $\alpha+\beta-m \geqslant 0$, then $\theta = \alpha+\beta-m$.
   Hence $\alpha = m-\beta+\theta$;

   Therefore, $\alpha \leqslant m-\beta+\theta$.

2. Because $v_\alpha \otimes v_\beta = v_\theta \leqslant v_\alpha \wedge v_\beta$, so $v_\theta \leqslant v_\beta$.

□

Within the definition of inference rules, the Łukasiewicz linguistic truth-valued implication algebra and its backward reasoning calculi, we could have the definition of the deduction for our reasoning as below:

Definition 16. Let $A \in F_V(LP(X))$ be a fuzzy premise set, and $C \in LP(X)$. An α-temporal deduction of C from A is defined as follows:

1. At a specific time t, an α-deduction of C from A at time t, denoted as (ω,t), is a finite sequence in the following form:

   $$(\omega,t) : (C_1,t,\alpha_1),(C_2,t,\alpha_2),\ldots,(C_n,t,\alpha_n),$$

   where $C_n = C$, $\alpha_n = \alpha$. For each i, $1 \leqslant i \leqslant n$, $C_i \in LP(X)$, $\alpha_i \in V$, and

   (a) If $C_i \in LP(X)$ and $\alpha_i = A(C_i)$ at time t; or

   (b) If there exist $j,k < i$, such that $C_i = s-LTMP(C_j,C_k)$ and $\alpha_i = \alpha_j \otimes \alpha_k$ at time t.

   (c) If there exists an α-deduction of C from A at t, then denoted it as $A \vdash_{(t,\alpha)} C$.

2. If there exists a finite sequence in the following form:

   $$(\omega,t) : (C_1,t,\alpha_1),(C_2,t,\alpha_2),\ldots,(C_n,t,\alpha_n),$$

   where $C_n = \bigcirc C$, $\alpha_n = \alpha$. For each i, $1 \leqslant i \leqslant n$, $\alpha_n = \alpha$, and $\alpha_i \in V$. If $i \neq n$, (C_i,t,α_i) is the same as the above same time process, and for $i = n$,

   (a) If $C \in LP(X)$ and $\alpha = A(C)$ at time *t+1*, or,

   (b) If there exist $j,k < n$, such that $C_n = n-LTMP(C_j,C_k)$ and $\alpha_n = \alpha_j \otimes \alpha_k$ at time *t+1*.

   Then it is called an α-deduction of C from A at time *t+1*, denoted it as $A \vdash_{(t,\alpha)} \bigcirc C$.

3. In case there is no event or rule changes the truth value of a state at time t.

   (a) If there is an event Occurs(C, (t-1)*, α), where $C \in LP(X)$ and $\alpha = A(C)$, occurs between time *t-1* and time t, then $C \in LP(X)$ and $\alpha = A(C)$ at time t.

   (b) If $C \in LP(X)$ and C is not taken effect by the same-time rule, next-time rule, or event occurring, and $\alpha = A(C)$ at time t-1, then it should follow Persistence rule, such that, we will have $C \in LP(X)$ and $\alpha = A(C)$ at time t.

Here we set $Ded(A)(C) = \vee\{\alpha; A \vdash_{(t,\alpha)} C\} = \vee\{B(C)$; B can be deduced from A at time t$\}$.

4.5. Soundness and Completeness Theorems

Theorem 2. *Soundness: Let $A \in F_V$*(LP(X)), *$\alpha \in V$, and $C \in LP(X)$. If there exists an α-temporal deduction of* C *from* A *in the following form:*

$$(C_1,t,\alpha_1),(C_2,t,\alpha_2),\ldots,(C_n,t,\alpha_n),$$

where $C_n = C$ or $\bigcirc C$, $\alpha_n = \alpha$. For each i, *$1 \leqslant i \leqslant n$, $C_i \in LP(X)$, and $\alpha_i \in V$, then for any temporal valuation γ in* LP(X), *γ satisfies* A *at time t_n implies that $\gamma(C) \geqslant \alpha$ at time t_n.*

Theorem 3. *Completeness: Let $A \in F_V$*(LP(X)), *$\alpha \in V$, and $C \in LP(X)$. If* C *is an α-logical consequence of* A *at time* t, *then there exists an α-temporal deduction of* C *from* A *in the following form:*

$$(C_1,t,\alpha_1),(C_2,t,\alpha_2),\ldots,(C_n,t,\alpha_n),$$

where $C_n = C$ or $\bigcirc C$, $\alpha_n = \alpha$. For each i, *$1 \leqslant i \leqslant n$, $C_i \in LP(X)$, and $\alpha_i \in V$.*

According to the soundness and completeness theorems, it shows that although the system is simple, it is sound and complete, such that, both of them provide a theoretical foundation for us to implement such a system into a program (e.g. Prolog).

The above soundness and completeness theorem mainly provided a support for the forward deduction, based on it, we have the soundness theorem for backward reasoning calculi given as follow.

Theorem 4. *(Soundness Theorem for Backward Reasoning Calculi) Suppose we have sets of independent states (S_I), dependent states (S_D), same-time rules (R_s), next-time rules (R_n), events (*E*), initial setting (I_c), time (*T*) and truth value levels (*V*), then we define $S = S_I \cup S_D$, and $FL = S_I \cup S_D \cup R_s \cup R_n \cup I_c \cup E$. Let $A \in FL$, $s \in S$, $t \in T$ and $v_\alpha \in V$. If* s *is an v_α-logical consequence of* A *at time* t *(Query(s,t,v_α)), then there exists an v_α-temporal deduction of* s *from* A *as following form:*

$$(s_1,t_1,v_{s_1}),(s_2,t_2,v_{s_2}),\cdots,(s_1,t_n,v_{s_n})$$

where $C_n = s$ *or* $\bigcirc s$, $t_n = t$, *and* $v_{s_n} = v_\alpha$. *For each* i, $1 \leqslant i \leqslant n$, $s_i \in S$, $v_{s_i} \in V$.

Proof. (Similar proof in (Ref. 20)) □

As the continuation of work (Ref. 20, 39), the aim of this paper is to provide an executable reasoning algorithm, includes forward and backward reasoning algorithms, for prediction and enquiry purpose, for decision-making system to solve the real-world decision making problem. Within this system, it should be able to solve the dynamic decision making problem which contains uncertain states and rules. We would like to build up a generic algorithm, which allows users insert their owned problem and situation and returns a reliable result which may lead the most acceptable choice.

The reasoning system is actually a linguistic truth-valued approximate reasoning system based on a linguistic-valued logic in which the truth degree of an assertion is a linguistic value in Łukasiewicz linguistic-valued algebra. For example, in daily life, when people are asked to assess the degree of a person being "Old", they usually give a verbal answer like very or quite true rather than a numerical answer such as 0.3 or 0.6. A key insight behind the linguistic-valued logic scheme is that we can use natural language to express a logic in which the truth values of propositions are expressed as linguistic value, such as using true, very true, less true, false, *etc.* instead of a numerical scale. It is expected that such an approach could reduce approximation errors that could be caused in quantification into numerical values and also will treat vague information in its true format, i.e., achieve reasoning with words.

5. Forward Reasoning Algorithm

Our forward reasoning algorithm allows the user to consider a knowledge base, including states, rules, events, and an initial setting. With such input, the system will strictly follow this knowledge base by time to calculate all the possible results and to list them in a table as the output. Through the forwarding reasoning algorithm, users can simulate their assumptions for decision-making problems, or to provide real data to the system to predict the possible outcome of a real problem. This function provides a way for users to verify their assumptions, or predict possible results, which could help them with their decision-making.

The classical forward reasoning algorithm for stratified causal theory (Ref. 21) only provides two possible statuses for a state: *true* or *false*, and it also assumes that all the rules in the knowledge base are equally true. This section extends the classical forward reasoning algorithm into a linguistic-valued one which allows users to do the prediction with uncertain information, attached both to states and rules.

Input:

- a stratified set of same-time rules (R_s),
- a set of next-time rules (R_n),
- a set of truth value V with r level $V = \{v_0, v_1, ..., v_{r-1}\}$,
- an initial setting (I_c), which are specified by determining $(s_i, 0, v_i)$ with $s_i \in S_I$, $v_i \in V$ and $(r_j, 0, v_j)$ with $r_j \in R$, $v_j \in V$,
- an event occurrence list (E), which is a set of formulae in the form $Occurs(Ingr(s), t*, v_i)$.

Output: a history of the values of all states up to a time t.

We say a rule is live if it can be applied. A threshold λ is assumed and used to determine if a state has supportive evidence of holding (when its degree of truth is in $[\lambda, v_{r-1}]$) or not (when its degree of truth is in $[v_0, \lambda)$). We compute the resulting history as follows:

1. At $t = 0$, set all the truth value of states as the declaration in Initial Setting. Apply any live same-time rules under the order of increasing stratification level. Once all possible same-time rules were applied, then apply any next-time rule.
2. For $t = 1, 2, .3,$
   (a) For each $Occurs(Ingr(s)$, (t-1)*, $v_i)$ that arrives, if $(s, t-1, v_j)$ holds then assert (s, t, v_i), where $v_i, v_j \in V$.
   (b) For each independent state s, if $(s, t-1, v_i)$ holds and (s, t, v_j) was not asserted, then assert (s, t, v_i). This is

called 'applying persistence' to the state s, where $v_i, v_j \in V$.

i. For k=1, 2, 3, , apply any live same-time rule of Stage k

ii. Apply persistence: For any co-k-dependent state s, if $(s, t-1, v_j)$ has not been asserted, and $(s, t-1, v_i)$, then assert $(s, t-1, v_i)$, where $v_i, v_j \in V$.

(c) Apply any live next-time rule.

Example 1. The algorithm given above provides a methodology of simulation. We now apply such algorithm into the scenarios in Section 3. Consider the scenario 1, suppose the user wants to know the projection of values into the future for this scenario until $t = 33$, then the system will return the results shown in the Table 1. We explain this results below:

Explanation:

$t = 0$: According to the initial setting, the truth value of *inbedTime* and *safe* should be v_5 and v_6, the rest should be set to v_0.

$t = 1$: Since nothing happens, so all the truth value of states should be kept the same as in $t = 0$ according to the persistency rule.

$t = 2$: occurs(ingr($\neg$*inBed*), 1*, v_5), then the truth value of *inBed* should be v_1, the change of *inBed* does not trigger any rule, so the rest of states keeps the same value as in $t = 1$, by the persistency rule.

$t = 3$: Since nothing happens, all the truth value of states should be kept the same as in $t = 2$ according to the persistency rule.

$t = 4$: Since nothing happens, all the truth value of states should be kept the same as in $t = 3$ according to the persistency rule.

$t = 5$: Since nothing happens, all the truth value of states should be kept the same as $t = 4$ according to the persistency rule.

$t = 6$: Event occurs(ingr(*inBed*), 5*, v_6) and occurs(ingr(*sitting*), 3*, v_4) occur, then the status of *inBed* is changed into v_6 by *EOR* and the truth value of *sitting* is changed to be v_4. Such changes activate the rule $(inBed \wedge sitting \wedge inBedTime \rightarrow safe, v_6)$, so according to the *s-LTMP*, we have the $Min\{v_{inBed}, v_{inBedTime}, v_{sitting}\} = v_4$ and $v_r = v_4$, then the truth value of *safe* should be $v_4 \otimes v_6 = v_4$. All other states remain the same as in $t = 5$.

$t = 7$: Since nothing happens, all the truth value of states should be kept the same as in $t = 6$ according to the persistency rule.

$t = 8$: occurs(ingr(*sitting*), 7*, v_5), so it changes the truth value of *sitting* to be v_5. Such change does activate the rule $(inBed \wedge sitting \wedge inBedTime \rightarrow safe, v_6)$, such that, the truth value of *safe* should change into v_5 by *s-LTMP*. The rest of states remain the same as in $t = 7$.

From $t = 9$ to $t = 28$: Since nothing changes during these time slots, all the status of states remains the same by the persistency rule.

$t = 29$: According to occurs(ingr(*mt20u*), 28*, v_5), then the truth value of *mt20u* is v_5. Since $(inBed \wedge sitting \wedge inBedTime \wedge mt20u \rightarrow undesirable, v_6)$ is triggered, then the status of *undesirable* is changed into v_5 by *s-LTMP*. Similarly, for $(undersirable \rightarrow carerCalls, v_6)$, the truth value of *carerCalls* should changed into v_5. The rest remains the same as in $t = 27$.

$t = 30$: Since nothing changes during these time slots, all the status of states remains the same by the persistency rule.

$t = 31$: Since nothing changes during these time slots, all the status of states remains the same by the persistency rule.

$t = 32$: There is an event, occurs(ingr(*carerGivesOK*), 31*, v_6), then the status of *carerGivesOK*

Table 1. Scenario 1 by Forward Reasoning

	0	1	2	3	4	5	6	7	8	...	29	...	32	33
inBed	v_0	v_0	v_1	v_1	v_1	v_1	v_6	v_6	v_6	...	v_6	...	v_6	v_6
standing	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_0	...	v_0	v_0
moving	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_0	...	v_0	v_0
sitting	v_0	v_0	v_0	v_0	v_0	v_0	v_4	v_4	v_5	...	v_5	...	v_5	v_5
laying	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_0	...	v_0	v_0
inbedTime	v_5	v_5	v_5	v_5	v_5	v_5	v_5	v_5	v_5	...	v_5	...	v_5	v_5
mt20u	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_5	...	v_5	v_5
safe	v_6	v_6	v_6	v_6	v_6	v_6	v_4	v_4	v_5	...	v_5	...	v_5	v_5
undesirable	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_5	...	v_5	v_1
carerVisits	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_0	...	v_0	v_0
carerCalls	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_5	...	v_5	v_5
carerGivesOK	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	v_0	...	v_0	...	v_6	v_6

is set to be v_6 by *EOR*. Such change activates a next-time rule, $(undesirable \wedge carerGivesOK \rightarrow \bigcirc\neg undesirable, v_6)$, then according to *n-LTMP*, the truth value of $\neg undesirable$ is v_5, because the truth value of $\neg undesirable$ is equal to $v_5 \otimes v_6 = v_5$. Such that the truth value of *undesirable* can be represented as $v_{undesirable} = v_{max} - v_{\neg undesirable} = v_{(6-5)} = v_1$ at $t = 33$.

$t = 33$: Since a next-time rule activated at $t = 32$, then the truth value of *undesirable* change into v_1. The rest remains the same as in $t = 32$.

This simulation shows all the procedure over time, in case the user only wants to know the process until $t = 33$, the simulation is stopped at $t = 33$ and provides a table which shows the result more clearly.

Example 2. Consider scenario 2, suppose the user wants to know the simulation of this scenario until $t = 6$, then the system will return the result as in Table 2.

Table 2. Scenario 2 by Forward Reasoning

	0	1	2	3	4	5	6
cookerOn	v_0	v_5	v_5	v_5	v_5	v_5	v_1
atKitchen	v_0	v_5	v_1	v_1	v_1	v_1	v_1
umt3u	v_1	v_1	v_1	v_1	v_1	v_5	v_1
cu	v_1	v_1	v_1	v_1	v_1	v_5	v_1
hazzard	v_1	v_1	v_1	v_1	v_1	v_5	v_1
alarmOn	v_1	v_1	v_1	v_1	v_1	v_5	v_1

Explanation:

$t = 0$: According to the initial setting, we have $(cookerOn, 0, v_0)$, $(atKitchen, 0, v_0)$, $(umt3u, 0, v_1)$, $(cu, 0, v_1)$, $(hazzard, 0, v_1)$, $(alarmOn, 0, v_1)$, then the truth value of *cookerOn*, *atKitchen*, *umt3u*, *cu*, *hazzard* and alarm are set to be v_0, v_0, v_1, v_1, v_1, and v_1.

$t = 1$: Since occurs($ingr(atKitchen)$, 0*, v_5) and occurs($ingr(cookerOn)$, 0*, v_5), then the truth value of both *atKitchen* and *cookerOn* is set to be v_5, but neither of them triggers any rule, then the rest of states remain the same status as in $t = 0$ by the persistency rule.

$t = 2$: For occurs($ingr(\neg atKitchen)$, 1*, v_5), then we have the truth value of *atKitchen* is back to v_1, and the rest keeps the same as in $t = 1$.

$t = 3$: Since nothing changes during these time slots, all the status of states remains the same by the persistency rule.

$t = 4$: Since nothing changes during these time slots, all the status of states remains the same by the persistency rule.

$t = 5$: According to the event occurs($ingr(umt3u)$, 4*, v_5), then the truth value of *umt3u* is changed to be v_5 by *EOR*, and rule (1) is activated. Since $Min(v_{cookerOn}, v_{\neg atKitchen}, v_{umt3u}) = v_5$, and the

truth value of rule (1) is v_6, the truth value of *cu* is $v_5 \otimes v_6 = v_5$ at $t = 5$ by *s-LTMP*. Since the change of *cu*, then it triggers rules (6) and (7), which changes the truth value of *alarmOn* and *hazzard* into v_5 by *s-LTMP* also. The change of *alarmOn* also triggers the next-time rule (8), such that we should have the truth value of *cookerOn* to be v_1 at $t = 6$ by *n-LTMP*.

$t = 6$: Since the next-time rule (8) was triggered at $t = 5$, then we have (*cookerOn*, 6, v_1), the rules (2), (3), (4) and (5) are activated, and we have the truth value of *umt3u*, *cu*, *hazzard*, and *alarmOn* changed back to v_1 by *s-LTMP*.

From the above two scenarios, we can see the forward reasoning algorithm will search time by time, such that, it provides a full result of all states from $t = 0$ to the time specified by the users.

Notice that, we assume that the uncertainty could also exist in rules, if we change the uncertain value associated with a rule, the final result may be different, some examples follow.

Example 3. Assume the truth value of rule $\neg cookerOn \rightarrow \neg cu$ is v_5, then the final result will be as presented in Table 3.

Table 3. An Example with Uncertainty of Rule(5) in Scenario 2

	0	1	2	3	4	5	6
cookerOn	v_0	v_5	v_5	v_5	v_5	v_5	v_1
atKitchen	v_0	v_5	v_1	v_1	v_1	v_1	v_1
umt3u	v_1	v_1	v_1	v_1	v_1	v_5	v_1
cu	v_1	v_1	v_1	v_1	v_1	v_5	v_2
hazzard	v_1	v_1	v_1	v_1	v_1	v_5	v_1
alarmOn	v_1	v_1	v_1	v_1	v_1	v_5	v_1

Example 4. Assume the truth value of rule $\neg kitchen \wedge cookerOn \wedge umt3u \rightarrow cu$ is v_5, then the final result should be as presented in Table 4.

Table 4. An Example with Uncertainty of Rule(1) in Scenario 2

	0	1	2	3	4	5	6
cookerOn	v_0	v_5	v_5	v_5	v_5	v_5	v_2
atKitchen	v_0	v_5	v_1	v_1	v_1	v_1	v_1
umt3u	v_1	v_1	v_1	v_1	v_1	v_5	v_2
cu	v_1	v_1	v_1	v_1	v_1	v_4	v_2
hazzard	v_1	v_1	v_1	v_1	v_1	v_4	v_2
alarmOn	v_1	v_1	v_1	v_1	v_1	v_4	v_2

The two examples above show that although there is only one rule containing uncertainty compared to the original setting, the result could be different. In example 3, since the truth value of rule (5) changed into v_5, it only affect the final output of *cu*. However, for the second example, although it only changes the truth value of rule (1), such change has an effect in several states. According to these two examples, we can see that, the uncertainty of rules play a very important role in our system. While setting up such uncertainty of rules in real world problems, we must be very careful to avoid large amount of information loss.

6. Backward Reasoning Algorithm

This section explains the process used to answer specific queries from users in our reasoning system. Sometimes, users may only want to focus on some particular states under certain specific conditions. In those circumstances, using forward reasoning, which fully explores the evolution process of the problem will incur in unnecessary computation. Hence, a backward reasoning algorithm can be used more effectively to trace the particular state without exploring the whole searching space.

The backward reasoning algorithm takes a query as a starting point and then reason backwards in time from the goal to the facts that sustain the conclusion. The advantage of this strategy is that focused by a goal (a specific query) it only explores part of the Knowledge Base (*KB*) which is related and necessary to answer the query, such that it avoids the system exploring the whole Knowledge Base which is computationally inefficient. It leads the whole searching process and is more accurate and efficient than the forward reasoning mode. The cost of saving

time depends on the type of *KB*. Some have many small trees because there are many rules which are not connected with each other, some *KB*s generate few bigger trees because many rules are interlinked with each other.

To explain the backward reasoning algorithm, we first introduce three main concepts, Supporting Tree (*ST*), Activation time list of *ST* (*AcTimes*), and searching process.

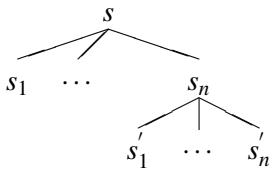
6.1. Supporting Trees

In this section, a tree-like representation of the Knowledge Base will be given and the example of how to build a tree will also be provided.

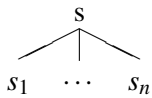
Definition 17. A supporting tree of state s (ST_s) should include the following properties:

- The head of the tree is the queried state (s).
- All the states in the leaves are independent states.
- The states in the same level is in 'AND' relationship.
- The link between two different levels is a rule, and 'parent' is the head of the rule, whereas, 'children' is the body of the rule.

This can be defined informally: Each $Query(s,t,v)$ has associated one or more *ST* with $s \in S_D$, all non-leave nodes belong to S_D and leave nodes belong to S_I.



For each

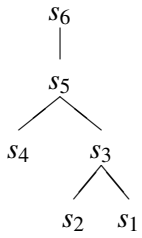


it means there is $r \in R$, such that, $s_1 \wedge \cdots \wedge s_n \rightarrow s$.

Example 5. Same-time Rule Tree Structure. Suppose $s_1, s_2, s_3, s_4, s_5, s_6 \in S$, and we have a truth value set V. Assume that we have the following rules, $(s_1 \wedge s_2 \rightarrow s_3, v_{\beta 1})$, $(s_3 \wedge s_4 \rightarrow s_5, v_{\beta 2})$, and $(s_5 \rightarrow s_6, v_{\beta 1})$, then we can express it as the following supporting tree structure:

$$\text{T}=\{(s_1 \wedge s_2 \rightarrow s_3, v_{\beta 1}),\ (s_3 \wedge s_4 \rightarrow s_5, v_{\beta 2}),\ (s_5 \rightarrow s_6, v_{\beta 1})\},$$

which can be represented as in the following figure:

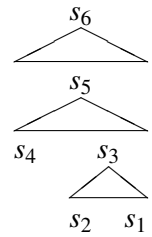


Same-time Rule Tree Structure

Example 6. Next-time Rule Tree Structure. Suppose $s_1, s_2, s_3, s_4, s_5, s_6 \in S$, and we have a truth value set V. Assume that we have the following rules, $(s_1 \wedge s_2 \rightarrow \bigcirc s_3, v_{\beta 1})$, $(s_3 \wedge s_4 \rightarrow \bigcirc s_5, v_{\beta 2})$, and $(s_5 \rightarrow \bigcirc s_6, v_{\beta 1})$, then we can express it as the following supporting tree structure:

$$\text{T}=\{(s_1 \wedge s_2 \rightarrow \bigcirc s_3, v_{\beta 1}),\ (s_3 \wedge s_4 \rightarrow \bigcirc s_5, v_{\beta 2}),\ (s_5 \rightarrow \bigcirc s_6, v_{\beta 1})\},$$

which can be represented as in the following figure:



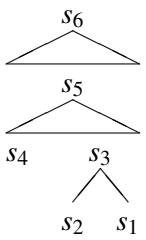
Next-time Rule Tree Structure

So, with the definitions of the tree structure of the same-time rule and next-time rule, then we can express the *ST* with both same-time rule and next-time rule.

Example 7. Same-time Rule and Next-time Rule Tree Structure. Suppose $s_1, s_2, s_3, s_4, s_5, s_6 \in S$, and a truth value set V. Assume that we have the following rules, $(s_1 \wedge s_2 \rightarrow s_3, v_{\beta 1})$, $(s_3 \wedge s_4 \rightarrow \bigcirc s_5, v_{\beta 2})$, and $(s_5 \rightarrow \bigcirc s_6, v_{\beta 1})$, then we can express it as the following supporting tree structure:

$$\text{T}=\{(s_1 \wedge s_2 \rightarrow s_3, v_{\beta 1}),\ (s_3 \wedge s_4 \rightarrow \bigcirc s_5, v_{\beta 2}),\ (s_5 \rightarrow \bigcirc s_6, v_{\beta 1})\},$$

which can be represented as in the following figure:



Same-time Rule and Next-time Rule Tree Structure

6.2. Activation Time List

The list of activation times when there are event occurrences can be used to show the search on meaningful times only. To create *AcTimes* list, it needs to know two parts of the Knowledge Base: the list of the independent states in *ST* and the event list in the Knowledge Base. We say the independent states in *ST* are the activation points of *ST* because *ST* is only activated by those independent states, every time the truth value of those independent states changes, then the specific *ST* is activated. Therefore, with the list of independent states of specific *ST* and the event list, we can know the activation time of that *ST*. We will use the following terminology:

'Obtain list *AcTimes* of Activation Times' means *AcTimes* is the decreasing order list that is obtained from merging the decreasing ordered lists of activation times for ST_s and $ST_{\neg s}$, denoted as $AcTimes_s$ and $AcTimes_{\neg s}$ For example if the rules in ST_s are triggered by events occurring at 3, 1 and 5 then this forms a list [5, 3, 1] and if the rules in $ST_{\neg s}$ are triggered by events occurring at 2, 2 and 5 then this forms a list [5, 2, 2]. The result of *AcTimes* list is [5, 5, 3, 2, 2, 1].

't becomes the next available time closest to time in *AcTimes*' means t is made max(*AcTimes*) and that time is extracted from the *AcTimes* list. In the previous example $t = 5$ and the remaining *AcTimes* list is [5, 5, 3, 2, 2, 1]. Notice that, the initial setting time should always be included in the activation time list, in the previous example, if we assume that the initial setting time is $t = 0$, then the final outcome of the list *AcTimes* should be [5, 3, 2, 2, 1, 0].

6.3. Detailed Backward Reasoning Algorithm

Suppose there is a query about s given as $Query(s,t,v_\alpha)$, then:

Step 1

(a) Form all trees,

(b) Find those trees supporting s is above threshold and those trees supporting s is below threshold,

(c) According on whether the winner *ST* has same sign than the query, or not, then returns 'true' or 'false', respectively

(d) If there is more than one *ST* winner,then choose one according to a domain dependent or domain independent heuristic.

Step 2

(a) Collect the list I_S of independent states of *ST*;

(b) Create the list *AcTimes* of *ST* by using the list I_S and events from the Knowledge Base;

(c) Use s as start, and search *ST* backwards until it reaches activation points (independent states) of *ST*;

Step 3: Return the answer of the query.

With such basic idea and structure of the algorithm, we can build the full backward reasoning algorithm as follows.

Notation: suppose that we want to know whether $Query([\neg]s,t,v_\alpha)$ is true or not (i.e., whether $([\neg]s,t,v_\beta) : v_\beta \geqslant v_\alpha$ or not). Here we use $[\neg]s$ as an abbreviation for s or $\neg s$. We represent with λ a $v_i \in L$, one of many possible linguistic values which is considered the minimum level of credibility for a proposition (a 'credibility threshold'). For a tree to be activated each rule $(s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow s, v_\tau)$ or $(s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow \bigcirc s, v_\delta)$ in the tree should be such that for all $s_i : (s_i,t,v_{\beta_i}), \in L$, $v_\theta = Min_i(v_{\beta_i}) \geqslant v_{r+\alpha-\tau} \geqslant \lambda$ or $v_\theta = Min_i(v_{\beta_i}) \geqslant v_{r+\alpha-\delta} \geqslant \lambda$ respectively, $i = 1, \cdots, n$ (in fact assume $v_\theta = Min_i(v_{\beta_i})$), to find the suitable v_θ supporting a given state s

such that $Query([\neg]s,t,v_\alpha)$ is true, it requires $v_\alpha \geqslant v_\theta \otimes v_\tau$ following the *n-LTMP* rule, it leads to the condition that is $v_\theta \geqslant v_{r+\alpha-\delta}$.

Notice that same-time rules are requested to be cycle free and 'stratified'. Each cycle generated by a query at time t will have a finite number of iterations until they are evaluated at 0. Another thing needs to be noticed is that if the supporting tree is only made of one independent state (use s as an example), then the ST should only be the state itself, which means $ST_s = [s]$.

The full algorithm is:

Input:

- the sets of independent and dependent states: S_I, S_D
- a set of non-cyclic Same-Time Rules, R_s
- a set of Next-Time Rules, R_n
- a set of truth value degree V, where $V = \{v_1, v_2, \cdots, v_r\}$
- a set of facts called initial conditions (I_c) provides the truth values of states in $S_I \cup S_D$ at time=0
- a set of known events, E, describing state ingression
- a credibility threshold λ
- $Query([\neg]s,t,v_\alpha)$

Output:

whether $Query([\neg]s,time,v_\alpha)$ is true or not and an explanation for the answer.

1. IF $([\neg]s,time,v_\beta) \in I_c$ or $occurs(ingr([\neg]s),(time-1)*,v_\beta) \in E$

   THEN IF $v_\beta \geqslant v_\alpha$

   THEN answer 'true' and give fact as explanation.

   ELSE answer 'false' and give fact as explanation.

2. ELSE ($Query([\neg]s,time,v_\alpha)$ must be inferred by deduction, possibly by persistency rule)

   Obtain the sets of trees ST_s and $ST_{\neg s}$

   Obtain list I_S of Independent States for the sets of ST_s or $ST_{\neg s}$

   Obtain list $AcTimes$ of Activation Times for the sets of ST_s or $ST_{\neg s}$

   Set $t = Max(AcTimes)$ and $t \leqslant time$

   Select the first ST_s or $ST_{\neg s}$ from the sets of trees ST_s or $ST_{\neg s}$

   REPEAT

   (a) IF $ST_s \cup ST_{\neg s} \neq \emptyset$

   (i) find a main rule in ST_s or $ST_{\neg s}$

   A. IF r: $(s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow [\neg]s, v_\tau) \in R_s$ and $v_\tau \geqslant v_\alpha$ THEN

   * for each $s_i \in S_I$: $Query(s_i,t,v_\theta)$ where $v_\theta = v_{r+\alpha-\tau}$, or

   ** for each $s_d \in S_D$: $Query(s_d,time,v_\theta)$ where $v_\theta = v_{r+\alpha-\tau}$

   B. IF r:$(s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow \bigcirc[\neg]s, v_\delta) \in R_n$ and $v_\delta \geqslant v_\alpha$ THEN

   * for each $s_i \in S_I$: $Query(s_i,t',v_\theta)$ where t' is max($AcTime \leqslant time$-1) and $v_\theta = v_{r+\alpha-\delta}$, or

   ** for each $s_d \in S_D$: $Query(s_d,time-1,v_\theta)$ where $v_\theta = v_{r+\alpha-\delta}$

   (ii) IF $[\neg]s$ can be proved true from a tree

   THEN answer 'true' ['false'] and use the tree as the explanation

   ELSE THEN SELECT next ST_s or $ST_{\neg s}$ from the sets of trees ST_s or $ST_{\neg s}$

   (b) ELSE THEN $Query([\neg]s,t'',v_\alpha)$, where $t'' = max(AcTime < t)$

   UNTIL ($[\neg]s$ can be proved)

Notice that, according to our definitions, if the truth value of rules (v_τ or v_δ) is smaller than the queried value (v_α), then the reasoning system will consider result from these rules as false by default.

Theorem 5. *Termination of all queries to the backward reasoning algorithm is guaranteed.*

Proof.

1. Basic Case: $Query(s,0,v_\alpha)$ is answered by the I_c.

   Hypothesis: assume for $t > 0$, it always finishes (t is a finite number). Then for $time = t+1$ (*time* is finite),

2. $Query(s,time,v_\alpha)$ the algorithm can proceed only as in any of the following cases:

   (a) There is an event from E, which is $Occur(ingr([\neg]s),(time-1)*,v_\beta)$, if $v_\beta \geqslant v_\alpha$, then answer 'true', otherwise, answer 'false' and the algorithm terminates.

   (b) There is a same-time rule

   $$s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow s, v_\tau$$

   because same-time rules are cycle-free and stratified, then there are no loops, therefore, each *ST* will be finite and the leaves of that tree can be checked for satisfaction through (1) or (2b).

   (c) There is a next-time rule

   $$s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow \bigcirc s, v_\delta,$$

   the answer will depend on queries, $Query(s_i,t',v_\theta)$ if $s_i \in S_I$, where $t' = Max(AcTimes \leqslant time-1)$, or $Query(s_i,time-1,v_\theta)$ if $s_i \in S_D$, where $i = 1,2,\cdots,n$ and $v_\theta = v_{r+\alpha-\delta}$ and by the induction hypothesis they all finish.

   (d) Neither (a), (b), or (c) applies, then assumes that the last occurrence of an event that changed s occurred in the past, the value of $Query(s,time,v_\alpha) = Query(s,t",v_\alpha)$, where $t" = Max(AcTimes)$.

Notice both (c) and (d) always change the truth value of states at an earlier time. Hence, if there is no explicit data, then the algorithm eventually reaches 0 where the query answer is provided by I_c. Reaching 0 can not be jumped over as time in our system is discrete and always jumped to a legal decreasing time, where $Occurs(s_i,t*,v_\beta)$ until it eventually reaches 0 and stops decreasing. □

6.4. Application of the Algorithm to a Practical Case

In this section, we apply the backward reasoning algorithm to the scenarios mentioned in Section **??**.

Scenario 1: the user wants to know what will be returned as feedback about the following queries:

- Is it probably the case the patient is in bed at time=6?
- Is it highly possible the patient is safe at time=4?

To answer the questions given above, we have the following queries for each question:

1. $Query(inBed,6,v_4)$

2. $Query(safe,4,v_5)$

1. For the first query, the algorithm asks to search the event list in Step(1), then we have an event occurs(ingr(*inBed*), 5*, v_6), we know that $v_6 > v_4$, which means that the system will return the result as 'true', given $occurs(ingr(inBed),5*,v_6)$.

2. For the second query, by Step(1), there is no event indicating. either *safe* or $\neg safe$ occurs at time=3*. In Step(2), we build up the following supporting trees of *safe* and $\neg safe$: $ST_{safe} = \{T1,T2,T3,T4,T5\}$ and $ST_{\neg safe} = \{T6,T7\}$ where:

T1=[$\neg inBed \wedge standing \rightarrow safe, v_6$].
T2=[$\neg inBed \wedge moving \rightarrow safe, v_6$].
T3=[$inBed \wedge sitting \wedge inBedTime \rightarrow safe, v_6$].
T4=[$(\neg inBed \wedge sitting \rightarrow \neg safe, v_6)$, $(\neg safe \wedge carerGivesOK \rightarrow \bigcirc safe, v_6)$].
T5=[$(\neg inBed \wedge laying \rightarrow \neg safe, v_6)$, $(\neg safe \wedge carerGivesOK \rightarrow \bigcirc safe, v_6)$].
T6=[$\neg inBed \wedge sitting \rightarrow \neg safe, v_6$].
T7=[$\neg inBed \wedge laying \rightarrow \neg safe, v_6$].

- Obtain the $ST_{safe} = T1$, then create the $I_S = [\neg inBed, standing]$, according to the event list, we have $AcTimes = [2,0]$. So, at Step(i), search the main rule of $T1$ and go to Step(A). Since $(\neg inBed \wedge standing \rightarrow safe, v_6) \in R_s$ and $v_6 > v_5$ at Step(A), and because $\neg inBed$ and *standing* are independent states, Step(B*), produces queries *Query*($\neg inBed$, 2, v_5) and *Query*(*standing*, 2, v_5).
  - For $\neg inBed$, an event occurs at *time*=1*, occurs(ingr($\neg inBed$), 1*, v_5), so the answer to this query is 'true'.
  - Since *Query*(*standing*, 2, v_5) returns 'false', given $(standing, 0, v_0)$ and no event changes that, then it does not support (*standing*, 2, v_5). The searching process is shown as Table 5.

Table 5. The Searching Process of T1

	0	1	2	3	4
inBed			$v_1(< v_5?)$		
standing	v_0		$v_0(\geqslant v_5?)$		
moving					
sitting					
laying					
inbedTime					
mt20u					
safe					$(\geqslant v_5?)$
undesirable					
carerVisits					
carerCalls					
carerGivesOK					

- Then the system should obtain $ST_{safe} = T2$, which is not successful for similar reasons.
- Obtaining $ST_{safe} = T3$, then it creates $I_S = [inBed, sitting, inBedTime]$ and $AcTimes = [2,0]$, and searches the main rule of $T3$: $(inBed \wedge sitting \wedge inBedTime \rightarrow safe, v_6) \in R_s$, but $v_6 > v_5$, and since *inBed*, *sitting* and *inBedTime* are independent states, Step(B*), produces queries *Query*(*inBed*, 2, v_5), *Query*(*sitting*, 2, v_5) and *Query*(*inBedTime*, 2, v_5).
  - Since *Query*(*inBed*, 2, v_5) returns 'false', given occurs$(ingr(\neg inBed), 1*, v_5)$ and no event changes that, then it does not support (*inBed*, 2, v_5). So T3 can not support that $(safe, 4, v_5)$. The searching process is given as Table 6.

Table 6. The Searching Process of T3

	0	1	2	3	4
inBed			v_1		$v_1(\geqslant v_5?)$
standing					
moving					
sitting					
laying					
inbedTime					
mt20u					
safe					$(\geqslant v_5?)$
undesirable					
carerVisits					
carerCalls					
carerGivesOK					

- If we consider instead the system obtains $ST_{safe} = T4$, then the list of independent states is $I_S = \{inBed, sitting\}$ and the activation time list is $AcTimes = [2,0]$. The main rule of $T4$ is $(\neg safe \wedge carerGivesOK \rightarrow \bigcirc safe, v_6) \in R_s$. In Step(B). Since $v_6 > v_5$, and *carerGivesOK* is independent state, it creates a new query *Query*$(carerGivesOK, 2, v_5)$ at Step(B*); because $\neg safe$ is a dependent state, then it creates a new query *Query*$(\neg safe, 2, v_5)$ at Step(B**).
  - For *Query*$(carerGivesOK, 2, v_5)$, there is no event to consider in Step(1) about *carerGivesOK* or $\neg carerGivesOK$.
  - In Step(2), since $\pm carerGivesOK$ are independent states, from the event list and I_c, we can infer the latest change of *carerGivesOK* is $(carerGivesOK, 0, v_0) \in I_c$. So, *Query*(*carerGivesOK*, 2, v_5) returns 'false'.

  Since *Query*$(carerGivesOK, 2, v_5)$ is 'false', then $T4$ can not support that $(safe, 4, v_5)$. The searching process is shown as Table 7.

Table 7. The Searching Process of T4

	0	1	2	3	4
inBed					
standing					
moving					
sitting					
laying					
inbedTime					
mt20u					
safe				$(\leqslant v_1?)$	$(\geqslant v_5?)$
undesirable					
carerVisits					
carerCalls					
carerGivesOK	v_0			v_0 $(\geqslant v_5?)$	

- The procedure to obtain $ST_{safe} = T5$ is similar to $ST_{safe} = T4$, which can not support $(safe, 4, v_5)$ either.
- If we consider $ST_{\neg safe} = T6$ to prove $Query(safe, 4, v_5)$ is false, we just need to prove that $Query(\neg safe, 4, v_2)$ is true.

  After obtaining $ST_{\neg safe} = T6$, then the independent state list should be $I_S = [\neg inBed, sitting]$ and the activation time list is $AcTimes[2, 0]$. So, by using the main rule of $T6$, $(\neg inBed \wedge sitting \rightarrow \neg safe, v_6)$, and given $v_6 > v_2$ we use Step(A). Since, both $\neg inBed$ and *sitting* are independent states, by Step(A*), we have two new queries: $Query(\neg inBed, 2, v_2)$ and $Query(sitting, 2, v_2)$.

  - For $Query(\neg inBed, 2, v_2)$, by Step(1) there is no event about $\neg inBed$ or *inBed*. Since $\pm inBed$ are independent states, according to the event list and I_c, the latest change is so $occurs(ingr(\neg\ inBed), 1^*, v_5)$, and $v_5 > v_2$ such that, it returns 'true'.
  - For $Query(sitting, 2, v_2)$, according to the I_c, we have $(sitting, 0, v_0) \in I_c$, by Step(1), it returns 'false'.

  Since $Query(sitting, 4, v_2)$ returns 'false', then $T6$ can not support $(\neg safe, 4,\ v_2)$. The searching process is shown as Table 8.
- The procedure for obtaining $ST_{\neg safe} = T7$ is similar to $ST_{\neg safe} = T6$, which can not support $(\neg safe, 4, v_2)$ either.

Table 8. The Searching Process of T6

	0	1	2	3	4
inBed			v_1 $(< v_4?)$		
standing					
moving					
sitting	v_0		$(\geqslant v_4?)$		
laying					
inbedTime					
mt20u					
safe					$(\leqslant v_4?)$
undesirable					
carerVisits					
carerCalls					
carerGivesOK					

- Since no tree supports $Query(safe, 4, v_5)$ or $Query(\neg safe, 4, v_2)$, the system investigates whether there is an event earlier on which can influence this state, in the absence of such evidence, it reaches 0, when, according to I_c, we have $(safe, 0, v_6) \in I_c$, so it supports $Query(safe, 4, v_5)$ by persistency rule.

Such that, the system will return the final result is: 'true, by $(safe, 0, v_6) \in I_c'$.

Scenario 2: the user wants to know what will be returned as feedback about the following queries:

1. Is the cooker probably off at time=1?
2. Is the cooker probably unattended at time=5?

To answer the questions given above, we have the following queries:

1. $Query(\neg cookerOn, 1, v_4)$
2. $Query(cu, 5, v_4)$

1. For the first query, at Step (1), there is no event about $\neg cookerOn$ or $cookerOn$ at $time = 1$. At Step(2), we obtain the $ST_{\neg cookerOn} = T1$ and $ST_{cookerOn} = T2$, such that:

   T1=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$, $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$]

   T2=[$cookerOn$]

   We create $I_S = \{\neg atKitchen, cookerOn, umt3u\}$, obtain the activation time list $AcTimes = [1, 0]$, then set $t = 1$.

- Select T1 as the main tree and the main rule of T1 is $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$, because $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6) \in R_n$ and $v_6 > v_3$, then we go to Step(B). Such that, we have a new query which is $Query(alarmOn, 0, v_4)$
  - For $Query(alarmOn, 0, v_4)$, there is an initial setting about $alarmOn$ in set I_c, which is ($alarmOn$, 0, v_1), and $v_1 < v_4$, such that, according to Step(1), it returns 'false' to the previous step.

  Since $Query(alarmOn, 0, v_4)$ returns 'false', T1 can not support $(\neg cookerOn, 1, v_4)$, then Select T2. The searching process is shown as Table 9.
- $T2 = [cookerOn] \neq \emptyset$. Since $cookerOn$ is independent state. According to the event list, the latest change is an occurrence occurs($ingr(cookerOn)$, 0^*, v_5), and $v_5 > v_2$, then returns 'true'. Hence, T2 is supported , then $Query(\neg cookerOn, 1, v_4)$ returns 'false'.

Table 9. The Searching Process of $Query(\neg cookerOn, 1, v_4)$

	0	1
cookerOn		$(\leqslant v_2?)$
atKitchen		
umt3u		
cu		
hazzard		
alarmOn	v_1 $(\geqslant v_4?)$	

So the final feedback for such simple query is:
'false, T2=[*cookerOn*], occurs(*ingr*(*cookerOn*), 0*, v_5)'

(2) For The second query, $Query(cu, 5, v_4)$, at Step (1), there is no event about cu or $\neg cu$ at $time = 5$. At Step(2), we obtain the $ST_{cu} = T1$ and $ST_{\neg cu} = T2$, such that,

T1=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$];
T2=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$, $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$, $(\neg cookerOn \rightarrow \neg cu, v_6)$]

We create the $I_S = \{\neg atKitchen, cookerOn, umt3u\}$, and obtain the activation time list as $AcTimes = [5, 2, 1, 0]$, then set $t = 5$.

- Select T1 as the main tree, which provides the main rule is $(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6) \in R_s$, and $v_6 > v_4$, according to Step(A*), then we produce new queries which are $Query(\neg atKitchen, 5, v_4)$, $Query(cookerOn, 5, v_4)$, and $Query(umt3u, 5, v_4)$.
  - For $Query(\neg atKitchen, 5, v_4)$, because it is independent state, and the closest instance of the event to $t = 5$ is occurs(*ingr*($\neg atKitchen$), 1*, v_5), where $v_5 > v_4$, then it returns 'true'.
  - For $Query(cookerOn, 5, v_4)$, because it is independent state, and the closest instance of the event to $t = 5$ is occurs(*ingr*(*cookerOn*), 0*, v_5), where $v_5 > v_4$, then it returns 'true'.
  - For $Query(umt3u, 5, v_4)$, because it is independent state, and the closest instance of the event to $t = 5$ is occurs(*ingr*(*umt3u*), 4*, v_5), where $v_5 > v_4$, then it returns 'true'.

  Since $Query(\neg atKitchen, 5, v_4)$, $Query(cookerOn, 5, v_4)$ and $Query(umt3u, 5, v_4)$ all return 'true', then T1 supports $Query(cu, 5, v_4)$ and returns 'true'. The searching process is given as Table 10.

Table 10. The Searching Process of $Query(cu, 5, v_4)$

	0	1	2	3	4	5
cookerOn		v_5				$(\geqslant v_4?)$
atKitchen			v_5			$(\geqslant v_4?)$
umt3u						v_5 $(\geqslant v_4?)$
cu						$(\geqslant v_4?)$
hazzard						
alarmOn						

- If we consider $ST_{\neg cu} = T2$ to prove $Query(cu, 5, v_4)$ is false, we just need to prove that $Query(\neg cu, 5, v_3)$ is true.

  Select T2 as the main tree, which provides $(\neg cookerOn \rightarrow \neg cu, v_6)$ as the main rule, then according to Step(A**), we have a new query $Query(\neg cookerOn, 5, v_3)$.

  - For $Query(\neg cookerOn, 5, v_3)$, for Step (1), there is no event about $\neg cookerOn$ or $cookerOn$ at $time = 5$. At Step(2), we obtain the $ST_{\neg cookerOn} = T3$ and $ST_{cookerOn} = T4$, such that,

    T3=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$, $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$]
    T4=[*cookerOn*]

    Select T3 as the main tree and $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6) \in R_n$ as the main rule, according to Step(B**), it creates a new query $Query(alarmOn, 4, v_3)$.

    - For $Query(alarmOn, 4, v_3)$, for Step (1), there is no event about *alarmOn* or $\neg alarmOn$ at $time = 4$. At Step(2), we obtain the $ST_{alarmOn} = T5$ and $ST_{\neg alarmOn} = T6$, such that,

      T5 = [$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$]
      T6 = [$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$, $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$, $(\neg cookerOn \rightarrow \neg alarmOn, v_6)$]

      Select T5 as the main tree and $(cu \rightarrow alarmOn, v_6) \in R_s$ as the main rule, according to Step(A**), it creates a new query that $Query(cu, 4, v_3)$.

      - For $Query(cu, 4, v_3)$, at Step (1), there is no event about cu or $\neg cu$ at $time = 4$. At Step(2), we obtain the $ST_{cu} = T7 = T1$ and $ST_{\neg cu} = T8 = T2$, such that, according to T1, we obtain the independent state as $I_S = \{\neg atKitchen, cookerOn, umt3u\}$ and the activation time list as $AcTimes =$

$[2,1,0]$, then it creates new queries as $Query(\neg atKitchen, 2, v_3)$, $Query(cookerOn, 2, v_3)$ and $Query(umt3u, 2, v_3)$ by Step(A*).

1) For $Query(\neg atKitchen, 2, v_3)$, because it is independent state, and the closest event of it to $t = 2$ is occurs($ingr(\neg atKitchen)$, 1*, v_5), where $v_5 > v_3$, then it returns 'true'.
2) For $Query(cookerOn, 2, v_3)$, because it is independent state, and the closest event of it to $t = 1$ is occurs($ingr(cookerOn)$, 0*, v_5), where $v_5 > v_3$, then it returns 'true'.
3) For $Query(umt3u, 2, v_3)$, because it is independent state, and the closest event of it to $t = 0$ is $(umt3u, 0, v_1)$, where $v_1 < v_3$, then it returns 'false'.

Since $Query(umt3u, 2, v_3)$ returns 'false', then T1 does not support $Query(cu, 4, v_3)$ and returns 'false'.

Since $Query(cu, 4, v_3)$ returns 'false', then T5 does not support $Query(alarmOn, 4, v_3)$ and returns 'false'.

Since $Query(alarmOn, 4, v_3)$ returns 'false', then T3 does not support $Query(\neg cookerOn, 5, v_3)$ and returns 'false'.

Since $Query(\neg cookerOn, 5, v_3)$ returns 'false', then T2 does not support $Query(\neg cu, 5, v_3)$. The searching process is given as Table 11.

Table 11. The Searching Process of $Query(\neg cu, 5, v_3)$

	0	1	2	3	4	5
cookerOn		v_5	$(\geqslant v_3?)$			$(\leqslant v_3?)$
atKitchen			$v_5(\geqslant v_3?)$			
umt3u	v_1		$(\geqslant v_3?)$			
cu					$(\geqslant v_3?)$	$(\leqslant v_3?)$
hazzard						
alarmOn					$(\geqslant v_3?)$	

Therefore, according to the Table 10 and 11, the answer for the query $Query(cu, 5, v_4)$ should be: 'true, by T1=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$], occurs($ingr(cookerOn)$, 0*, v_5), occurs($ingr(\neg atKitchen)$, 1*, v_5), occurs($ingr(umt3u)$, 4*, v_5)'.

For the next query, we want to illustrate a finite loop, which may occur in some cases. If we have the following query, $Query(\neg cookerOn, 3, v_3)$, we apply the following procedure:

(3) For the $Query(\neg cookerOn, 3, v_3)$, at Step(1), there is no event about $\neg cookerOn$ or $cookerOn$ at $time = 3$. At Step(2), we obtain $ST_{\neg cookerOn} = T1$ and $ST_{cookerOn} = T2$, such that:

T1=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$, $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$]

T2=[$cookerOn$]

We create $I_S = \{\neg atKitchen, cookerOn, umt3u\}$, and obtain the activation time list $AcTimes = [2,1,0]$, then set $t = 2$.

- Select T1 as the main tree, which provides the main rule is $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$. Because $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6) \in R_n$ and $v_6 > v_3$, according to Step(B**), then we produce a new query which is $Query(alarmOn, 2, v_3)$.
  - For $Query(alarmOn, 2, v_3)$, there is no event about $alarmOn$ or $\neg alarmOn$ at $time = 2$ at Step(1). At Step (2), for obtaining the $ST_{alarmOn} = T3$ and $ST_{\neg alarmOn} = T4$:

    T3=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$]

    T4=[$(\neg atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu, v_6)$, $(cu \rightarrow alarmOn, v_6)$, $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$, $(\neg cookerOn \rightarrow \neg alarmOn, v_6)$]

    To explain the loop case, we only focus on T4. We create $I_S = \{\neg atKitchen, cookerOn, umt3u\}$, and obtain the activation time list $AcTimes = [2,1,0]$, then set $t = 2$ and select T4 as the main tree, and the main rule of T4 is $(\neg cookerOn \rightarrow \neg alarmOn, v_6)$. Because $(\neg cookerOn \rightarrow \neg alarmOn, v_6) \in R_s$ and $v_6 > v_3$, then go to Step(A). By Step(A**), we produce a new query which is $Query(\neg cookerOn, 2, v_3)$.
    - For the $Query(\neg cookerOn, 2, v_3)$, there is no event about $\neg cookerOn$ or $cookerOn$ at time=2, then go to Step (2). At Step (2), for obtaining the $ST_{\neg cookerOn} = T1$ and $ST_{cookerOn} = T2$. Then, we create $I_S = \{\neg atKitchen, cookerOn, umt3u\}$ and obtain the activation time list $AcTimes = [2,1,0]$. After that, set $t = 2$ and select T1 as the main tree, which provides the main rule is $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6)$. Because $(alarmOn \rightarrow \bigcirc(\neg cookerOn), v_6) \in R_n$ and $v_6 > v_4$, then go to Step(B). According to Step(B**), we produce a new query which is $Query(alarmOn, 1, v_3)$.

The above process will run again and reaches $Query(alarmOn, 0, v_3)$, and there is an initial setting

about *alarmOn* in set I_c, which is (*alarmOn*, 0, v_1), and $v_1 < v_3$, such that, according to Step(1), it returns 'false' to the previous step. The searching process is given as Table 12.

Table 12. The Searching Process of $Query(\neg cookerOn, 3, v_3)$

	0	1	2	3
cookerOn		$(< v_3?)$	$(< v_3?)$	$(< v_3?)$
atKitchen				
umt3u				
cu				
hazzard				
alarmOn	v_1 $(\geqslant v_3?)$	$(\geqslant v_3?)$	$(\geqslant v_3?)$	

Since the final step returns 'false', then the whole loop returns 'false' to the query, such that, the result for this process is 'false' and the system needs to select another *ST* to prove the query. From this example, we can see that because all the same-time rules have been stratified, a loop can only be created between next-time rules. However, the loop will always be decreased by processing next-time rules, such that, eventually it will reach the initial condition set, and, the system always returns a specific feedback to the user.

6.5. Discussion

Compared to forward reasoning algorithm, the backward reasoning algorithm does not cover all states and rules all the time. It only focuses on the states and rules which are related to the queried state, such that, it saves a lot of time and memory during the running procedure and it is more efficient than forward reasoning algorithm. Both algorithms return an answer for any possible query. We have illustrated how both, forward and backward reasoning, algorithms work and how they provide different benefits to the users.

7. Conclusion

This paper presents a combined logical system to reason with uncertainty and time as well as two algorithms that provides two different strategies: forward and backward reasoning. The forward reasoning algorithm simulates a typical decision-making process, which contain an initial set of facts, and find all the entailed conclusions up to the time slot specified by the users. The backward reasoning is constructed over three main concepts, supporting tree (*ST*), activation time list (*AcTimes*) and searching process. Unlike forward reasoning, the main focus of backward reasoning is to analyse the specific queries from users, and return a result, which contain 'true' or 'false' to a certain degree, and an explanation. With these two algorithms, this system can be used to predict the status of a decision-making process and trace the status of the specific state with the initial assumptions provided by the users.

Two scenarios extracted from a Smart Home system illustrated how the system works. For the simulations, the results of both scenarios clearly show the status of states in each time slot. We applied the backward reasoning algorithm to both scenarios, providing four queries in two scenarios, which included the following different cases: pure same-time rule, pure next-time rule, multi rules and *ST*s, and loop. The backward reasoning system returned a reasonable result to each query.

8. Reference

1. J. Pavelka, "On fuzzy logic I, II, III," *Z. Math. Logik Grundlagen Math*, **25**, 45–52, 119–134, (1979).
2. V. Novák, I. Perfilieva and J. Mockor, *Mathematical Principles of Fuzzy Logic*, Kluwer, Dordrecht, (2000).
3. P. Hájek, *Metamathematics of Fuzzy Logic*, Kluwer, Dordrecht, (1998).
4. S. Gottwald, "A treatise on many-valued logics," *Studies in Logic and Computation*, **9**, (2001).
5. L. Bolc and P. Borowik, *Many-Valued Logics, 1. Theoretical Foundations*, Springer, Berlin, (1992).
6. R. Cignoli, I. D'Ottaviano and D. Mundici, *Algebraic foundations of many-valued reasoning*, Kluwer, Dordrecht, (2000).
7. Y. Xu, D. Ruan, K. Qin and J. Liu, *Lattice-valued Logic: An Alternative Approach to Treat Fuzziness and Incomparability*, Springer-Verlag, (2003).
8. L. Martinez, D. Ruan and F. Herrera, "Computing with words in decision support systems: An overview on models and applications," *International Journal of Computational Intelligence Systems*, **3**(4), 382–395, (2010).
9. A. Galton, "Temporal logics and their applications," *Academic Press Professional, Inc.*, San Diego, CA, USA, (1987).
10. E. A. Emerson, *Temporal and modal logic*, 995–1072, (1990).
11. D. Gabbay, I. Hodkinson and M. Reynolds, *Temporal Logic (Volume 1): Mathematical Foundations and Computational Aspects*, Oxford University Press, (1994).
12. Y. Venema, *Temporal logic*, Blackwell Publishers, (2001).
13. J. Augusto, "A general framework for reasoning about change," *New Generation Comput.*, **21**(3), (2003).
14. J. Augusto, "Temporal reasoning for decision support

in medicine," *Artif. Intell. Med.*, **33**(1), 1–24, (2005).
15. F. Kröger and S. Merz, *Temporal Logic and State Systems (Texts in Theoretical Computer Science. An EATCS Series)*, Springer Publishing Company, Incorporated, (2008).
16. G. Escalada-Imaz, "A temporal many-valued logic for real time control systems," *In AIMSA'00: Proceedings of the 9th International Conference on Arti cial Intelligence*, pages 91–100, London, UK. Springer-Verlag, (2000).
17. M. Cárdenas Viedma, R. Morales and I. Sánchez, "Fuzzy temporal constraint logic: a valid resolution principle," *Fuzzy Sets Syst.*, **117**(2), 231–250, (2001).
18. M. Mucientes, R. Iglesias, C. Regueiro, A. Bugarín and S. Barro, "A fuzzy temporal rule-based velocity controller for mobile robotics," *Fuzzy Sets Syst.*, **134**(1), 83–99, (2003).
19. S. Schockaert and M. De Cock, "Temporal reasoning about fuzzy intervals," *Artif. Intell.*, **172**(8-9), 1158–1193, (2008).
20. Z. Lu, J. Liu, J. Augusto and H. Wang, "A many-valued temporal logic and reasoning framework for decision making," *Computational Intelligence in Complex Decision Systems*, 127–148, (2010).
21. A. Galton and J. Augusto, "Strati ed causal theories for reasoning about deterministic devices and protocols," *In TIME '02: Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)*, 52, Washington, DC, USA. IEEE Computer Society, (2002).
22. J. Liu, D. Ruan, Y. Xu and Z. Song, "A resolution-like strategy based on a lattice-valued logic," *IEEE Trans. on Fuzzy Systems*, **11**(4), 560–567, (2003).
23. J. Welton, *A Manual of Logic*, W.B. Clive, (1922).
24. L. Zadeh, "Approximate reasoning based on fuzzy logic," *In IJCAI'79: Proceedings of the 6th international joint conference on Arti cial intelligence*, 1004–1010, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc, (1979).
25. V. Novák, "First-order fuzzy logic," *Studia Logica*, **46**(1), 87–109, (1984).
26. M. Gupta, "Book review: Fuzzy sets, fuzzy logic, and fuzzy systems (selected papers by professor Lotfi A. Zadeh)," *J. Intell. Fuzzy Syst.*, **6**(4), 497–498, (1998).
27. L. Zadeh, "Fuzzy sets," *Information and Control*, **8**, 338–353, (1965).
28. G. Klir and B. Yuan, *Fuzzy sets an fuzzy logic: Theory and Applications*, Prentice-Hall PTR, (1995).
29. J. Allen, "Time and time again: The many ways to represent time," *Intl J. of Intelligent Systems*, **6**(4), 1–14 (1991).
30. J. Allen and G. Ferguson, "Actions and events in interval temporal logic," *J. Log. Comput.*, **4**(5), 531–579, (1994).
31. J. Augusto, H. Wang and J. Liu, "Situation assessment during disaster management," *International Journal of Computational Intelligence Systems*, **1**(3), 237–247, (2008).
32. J. Łukasiewicz, "O logice trówartosciowej (on three-valued logic)," *Ruch Filozoficzny*, **5**, 170–171, (1920).
33. G. Gerla, *Fuzzy Logic : Mathematical Tools for Approximate Reasoning (Trends in Logic)*, Springer, (2001).
34. P. Hájek, "Fuzzy logic and arithmetical hierarchy," *Fuzzy Sets Syst.*, **73**(3), 359–363, (1995).
35. S. Gottwald, "Fuzzy sets and fuzzy logic: the foundations of application - from a mathematical point of view," *Friedr. Vieweg and Sohn Verlagsgesellschaft mbH*, Wiesbaden, Germany, (1993).
36. J. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, **26**(11), 832-843, (1983).
37. C. Freksa, "Temporal reasoning based on semi-intervals," *Artif. Intell.*, **54**(1-2), 199–227, (1992).
38. Linear-Time Temporal Logic (1998). *Stanford, http://www-step.stanford.edu/tutorial/temporal-logic/temporal-logic.html*, Link checked November 22, 2009.
39. Z. Lu, J. Liu, J. Augusto and H. Wang, "Multi-valued temporal reasoning framework for decision-making," *In In Proceeding of the 8th International FLINS Conference*, 301–306, (2008).
40. J. Augusto, P. McCullagh, V. McClelland and J. Walkden, "Enhanced healthcare provision through assisted decision-making in a smart home environment," *In In Proc. of the 2nd Workshop on Artificial Intelligence Techniques for Ambient Intelligence (AITAmI07)*, Hyderabad, India, (2007).
41. J. Liu, L. Lopez, Y. Xu and Z. Lu, "Automated reasoning algorithm for linguistic valued Łukasiewicz propositional logic," *In ISMVL '07: Proceedings of the 37th International Symposium on Multiple-Valued Logic*, 29, Washington, DC, USA. IEEE Computer Society, (2007).
42. R. Yager, "An approach to ordinal decision making," *International Journal of Approximate Reasoning*, **12**, 237–261, (1995).
43. F. Herrera and E. Herrera-Viedma, "Linguistic decision analysis: Steps for solving decision problems under lingusitic information," *Fuzzy Sets and Systems*, **115**, 67–82, (2000).
44. L. Martinez, "Sensory evaluation based on linguistic decision analysis," *International Journal of Approximate Reasoning*, **44**(2), 148–164, (2007).
45. M. Delgado, J. Verdegay and M. Vila, "On aggregation operations of linguistic labels," *International Journal of Intelligent Systems*, **8**(3), 351–370, (1993).