International Journal on Artificial Intelligence Tools
Vol. XX, No. X (2012) 1–36
© World Scientific Publishing Company

A SYSTEM TO REASON ABOUT UNCERTAIN AND DYNAMIC ENVIRONMENTS

ZHIRUI LU, JUAN AUGUSTO, JUN LIU, HUI WANG

School of Computing and Mathematics, University of Ulster, Shore Road, Newtownabbey, Co. Antrim BT37 0QB, Northern Ireland, UK
Lu-Z1,@email.ulster.ac.uk; [jc.augusto, j.liu, h.wang]@ulster.ac.uk

ASIER AZTIRIA

University of Mondragon, Basque Country, Spain.
aaztiria@eps.mondragon.edu

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Decision-making on uncertain and dynamic domains is still a challenging research area. This paper explores a solution to handle such complex decision making based on a combined logic system. We provide an explanation of our reasoning system focused on the algorithms and their implementations. The reasoning system is based on a multi-valued temporal propositional logic which we use as the foundation for the implementation of simulation/prediction and query answering tools. This system is available for users to represent knowledge and to refine these systems to debug them and to try different problem solving strategies. We provide examples to illustrate how the system can be used including a problem based on a real smart environment.

Keywords: Decision making; uncertainty; temporality; reasoning, multi-valued logics.

1. Introduction

Decision-making under uncertainty and temporality is an important open problem in the field of decision making. This problem has two main dimensions: uncertainty and temporality. By focusing on different aspects of the problem (to suit different types of applications), we can identify different approaches to this problem: decision making with uncertain temporality, decision making with uncertain information in a certain temporal environment, and decision making with uncertain information in uncertain temporal environment.

The approaches may be probabilistic or logical or a mixture of both. Logic-based formalisms play a very important role in artificial intelligence and have made significant contributions to both the theory and application of artificial intelligence. A logically consistent theoretical structure can provide a rational foundation and support tool to explain and justify empirical observations and rational decision making. Only through an exploration of the underlying logic can we ascertain the consistency and completeness of our analyses. For analysing uncertain information, multi-valued logic [1, 2, 3, 4], fuzzy logic and lattice-valued logic [5, 6, 7, 8, 9], were introduced and investigated in the literature. To handle temporal information, temporal logic was introduced in early 20th century. Temporal logic has different branches, two of which are interval temporal logic (*ITL*) and linear temporal logic (*LTL*) [10, 11, 12, 13, 14]. However, in real world applications, uncertainty and temporality may not appear separately; actually, they co-exist in most of real cases. To handle uncertainty and temporality at the same time in decision making, researchers have tried to combine different logics, such as fuzzy logic and interval temporal logic [15, 16, 17,18], fuzzy logic and linear logic [19, 20].

Our research focuses on the logical approach aiming at handling uncertainty and temporality in an integrated way. Accordingly, we extended a temporal reasoning framework introduced by Galton and Augusto [21, 22] so that the representation and reasoning with uncertainty is allowed within the system. Uncertainty is introduced in the form of a multi-valued logic and reasoning framework, by means of the Łukasiewicz logic [5, 9, 23]**.** This new combined logic system, called Multi-Valued Temporal Propositional Logic (*MTPL*) system [24, 25, 26] has been proved to be a sound and complete logic system. *MTPL* also allows uncertain information to be represented with either a numerical truth value in the [0, 1] interval or a linguistic truth value in a qualitative nature. Based on this sound and complete formal logical framework *MTPL*, this paper provides a detailed explanation of its reasoning system focused on the algorithms and their implementations. In the following sections, we will provide a review of related work in combined logic based reasoning systems in handling both uncertainty and temporality (Section 2). We then briefly introduce *MTPL* in Section 3. This is followed by the reasoning forward and backward algorithms along with illustrative examples (Section 4). Then we discuss aspects of the implementation of the proposed reasoning system and we explain its functionality through examples (Section 5). We then illustrate some of the potential of the system by explaining one of its applications on assisting the improvement of a Knowledge Base (KB) that is used for decision-making on an Intelligent Environment (Section 6). This is followed by our conclusions.

2. Related work

The technical literature reports on a number of conceptual frameworks which aim at handling both temporality and uncertainty in complicated decision-making problems. We focus here on logic-based approaches.

2.1 Logic Systems for Temporality with Uncertainty

Temporality with uncertainty as a research area mainly focuses on how to handle an 'approximate' time description in decision-making problems. For example, a message might say, "It takes Fred about 20 minutes to get to work" [17], where the time description for his journey is not exact, and may be shorter or longer than 20 minutes. The length may depend on many reasons such as traffic jams or delays in public transport. This type of uncertainty of temporality is important in real-world decision-making.

2.1.1 ITL with Uncertainty

The most widely cited and well-known approach to embed temporality in a logic system was the one proposed by Allen following on work published first in philosophy. Allen's temporal reasoning framework provided an interval-based temporal logic (ITL) [10]. In his work, Allen considered 13 possible simple relations between two different time intervals using starting and ending points of the time interval. This seminal work suffered however of serious implementation problems (computational complexity and absolute knowledge requirements). While working to solve the problem of uncertainty in *ITL*, researchers have taken various approaches: some of them focused on the uncertainty of the time interval and some on the uncertainty of different time relations.

Manaf and Beikzadeh [17] introduced a combination of fuzzy logic and interval temporal logic, which focuses on time interval with uncertainty. Fuzzy temporal membership function was introduced to handle this kind of case. This combined logic implements a fuzzy set concept in interval temporal logic. This new combined system could be used to analyze a decision-making problem between two imprecise time descriptions, for example: "It takes Fred about 20 minutes to get to work. John leaves for work 5 minutes later than Fred and arrives a few minutes earlier than Fred. Is it possible for Fred to arrive at work earlier than John?". A similar idea was also discussed by Schockaert and Cock [16] who suggested introducing fuzzy intervals into temporal reasoning through 'fuzzy time intervals' which is used to handle uncertainty at the starting and ending points of a time interval (including the 13 relations from Allen). Raha and Ray [29] also provided an approximate temporal reasoning approach which incorporated the fuzzy set concept into temporal reasoning. They provided different algorithms for the combined reasoning and used probability theory to construct the calculation of the uncertainty of time.

Rather than handling the uncertainty of the time interval, some researchers focused on the uncertainty of relations between time intervals in *ITL*. Aboelela and Douligeris [15] suggested a new reasoning model for fuzzy temporal reasoning that redefined the relation between time intervals based on Allen's work. In their work, they pointed out that there was uncertainty among different relations. For example, the 'precede' relation by Allen's definition could be expressed as a 'fuzzy before' relation, based on the description of the problem and the level of fuzziness. The authors also assume there might be a fuzzy range between time points and relations between time intervals might not be always constant.

Godo and Vila [28] introduced possibilistic temporal reasoning, based on fuzzy temporal constraints. They provided a medical example to illustrate the suggested reasoning, which defined a tuple FUZZDIST(BEGIN(OFP), END(OFP), Π) to indicate the fuzzy time interval.

Ryabov and Trudel [34] suggested combining *ITL* with probabilistic representations. Under their definitions, the uncertainty of time interval could be expressed by probability which allow to calculate the probability of different ranges of time intervals and the probability of their relations, a probabilistic temporal interval network (*PTIN*).

Tawfik and Neufeld [36] combined ITL with a Bayesian network. This model was used in order to analyze the probability of something happening within a given time under certain conditions. To handle the probability of independent elements, the authors introduced events and effects to indicate all other sources and provided probability to construct the conditional probability table for a Bayesian network.

2.1.2 TL with Uncertainty

The interval-based approach described in the previous section is part of only one trend to represent and reason with time which refers to time more explicitly. Other authors and technical fields favour more implicit ways to represent temporality. Temporal Logics (TL) which are based on this style typically include operators like *G* for 'it is always true that …', *F* for 'it is eventually true that …', and *U* for '… is true until … becomes true'. Notice that whilst in Allen's proposal on may indicate an interval of time passing in more absolute terms, for example, 'Friday 11th of November, 2011' with the U operator an interval is relative to other events which may not necessarily have absolute time attached. Operators-based temporal logical frameworks are also important and popular in computer science. See Ref. 16 for a comparison on these two approaches to temporality.

A combination of operator based temporal logics with uncertainty has been applied in many areas [19, 20, 37, 38]. Zhang *et al.* [20] introduced a combined logic called fuzzy propositional modal logic (*FPML*), in which semantics are associated by fuzzy Kripke semantics. This work introduced the concept that there is a probability for each event in *FPML*, which is written as $\langle \varphi, n \rangle$; for example, <□*φ,* 0.7>, which means there is a hypothesis that the probability that □*φ* will happen is equal to or greater than 0.7. The theory provided fuzzy reasoning based on *FPML*, which was used to handle the uncertainty of constraints, where *n* is called believable degrees and the constraint is called a fuzzy constraint.

Similarly, Viedma *et al.* [38] proposed another combined logic to analyze *TL* with uncertainty, which they called fuzzy temporal constraint logic. This is a first-order logic based on temporal constraint logic and a fuzzy constraint network.

Palshikar [39] suggested another combination of fuzzy logic and *TL* that mainly focused on the relations between atoms. It extended fuzzy proposition logic (*FPL*) into fuzzy propositional linear temporal logic (*FzPLTL*). This combination assumed that the underlying model of time is a timeline consisting of a finite number of linearly ordered time instants. Palshikar also introduced a finite fuzzy temporal interpretation, a total function associating a fuzzy truth value from the set [0, 1]. The fuzzy truth-value set was used in the computation of fuzzy logical connectives.

Fuzzy temporal logic could be used in robot control systems, as presented by Lamine and Kabanza [19]. In this application, they combined linear temporal logic, which mainly focused on four *TL* connectives—*next*, *always*, *eventually* and *until*—and the fuzzy logical connective *negative*. It introduced a *weight* concept to indicate the uncertainty in the system, which was formed in $\langle p, \pi \rangle$, where *p* is an infinite sequence of world states and *π* is a real-valued function that evaluates propositions in states.

2.2 Logic Systems for Handling Uncertain Information in Dynamic Environments

Another type of combination of uncertainty and temporality in logic based approaches is handling decision-making problems with uncertain information in a dynamic environment. Different from temporality with uncertainty, this focuses on dealing with uncertain information with a determined temporal property. Probabilistic logic and fuzzy logic are two well-known logics used to deal with uncertain information analysis; these can be combined with different styles of temporal logics to deal with temporality.

In the case of robot control, Mora and Sanchez [40] introduced a real-time fuzzy logic-based navigation controller for mobile robot control which was used to represent uncertain information from sensors collecting data in real time. This controller allowed the decision-making system in the robot to analyze uncertain information at every instant while collecting information in real time.

Mucientes *et al.* [41] used the same method of combining fuzzy logic and temporal logic for a robot system, but they proposed a different decision-making model for the robot velocity controller, called fuzzy temporal rule-based velocity controller. The core of the controller implemented an explicit decision model for knowledge representation and reasoning called fuzzy temporal rules. Based on the research above, Mucientes and Bugarin [42] introduced a new system called the quantified fuzzy temporal rules model, which was set up to analyze a human leg pattern recognition task using data from the robot's sensors. This model was applied to a robot system to detect leg movements.

For another real-time control system, Escalada-Imaz [18] suggested a many-valued temporal reasoning method that extended the truth value of a state to indicate the uncertainty of an element for real-time control systems. This system implemented a valid area (only the interaction area of the condition states) of the conclusion part of the rule, which is an interval temporal logic application. The paper defined a literal (T, V, s), which means that the truth value of state s in time interval T is V, and there are two inference rules to the combined logic, *Intervals Rule* (*IR*) and *Temporal Multivalued Modus Ponens* (*TMMP*). Escalada-Imaz also pointed out that the truth value could be numeric or linguistic.

Compared with the above reviewed logic systems, the logic system MTPL presented in this paper aims to be generic (not application dependent) and also simple and efficient. For example, several of the proposals above are tailored to a specific field (e.g., robotics), or target rich temporal knowledge representations (e.g., time intervals related in various forms), or assume that rules always have the same level of credibility. Our approach is bottom up. We are developing a theory which starts with simple but essential ingredients and we are building more complex concepts on top guided by efficiency concerns. Many of the systems above starts with very complex languages and never reach implementation. Our system is more modest in expressiveness but has a formal theory [24,25,26] which naturally leads to algorithms facilitating simulation and explanatory query answering. These algorithms are implemented and provide system designers and developers a tool which has restricted but well defined expressiveness and a reasonably efficient implementation which is useful in practical cases, as we illustrate later on.

3. Some Basic Concepts and Notations about *MTPL*

The sound and complete formal logic framework of *MTPL* has been detailed in [24, 25, 26], this paper will be focused on the detailed explanation of the reasoning system, so this section reviews some basic concepts and notations considered in our reasoning system, for more details about *MTPL* the reader is referred to [24, 25, 26].

In *MTPL*, it defined a atomic state set S, a rule set R, a truth value set V, a discrete set T of atomic intervals (called *time slots*), and an event set E. Each element $s \in S$ comes in pairs. Each state has two statuses: positive (s) and negative ($\neg s$), we will use ±s as a simplifying notation to denote both s and $\neg s$. Given that S is the set of all atomic states (both positive and negative), these states can be partitioned into two classes, *independent* and *dependent*. *MTPL* uses S_I and S_D to refer to the set of independent and dependent states respectively, where $S_I \cap S_D = \varnothing$ and $S_I \cup S_D = S$. An independent state does not depend causally on other states' holding at the same time, whereas a dependent state can do so [21]. Moreover, the status of independent states can only be changed or initiated by the *initial setting* or *events* (from E). We call dependent state s *co-independent* if $\neg s$ is independent.

Aside from atomic states, the logic system also introduces the following non-atomic state: $s_1 \wedge s_2$, where $s_1, s_2 \in S$; this state holds if and only if both s_1 and s_2 hold. We say a state s holds means the status of state s is true and state $\neg s$ holds means the status of state s is false. To represent that state s holds at time t in *MTPL*, a predicate is introduced: *HoldsAt*(s, t), or (s, t). The rule for negative states is defined as:

$\neg$ *HoldsAt*(s, t) $\leftrightarrow$ *HoldsAt*($\neg s$, t) is shortened as $\neg (s, t) \leftrightarrow (\neg s, t)$

As mentioned above, an independent state can only be affected by the initial setting and events. An event is used to model all the impingements on the system from external resources. All such events are modelled as *ingression* of state s and notated as *ingr*(s). In *MTPL*, an event can only appear in the instant between two vicinal time slots, such as t-1 and t, and this instant is denoted as (t-1)*. To represent an event's occurrence, the predicate *Occurs* is introduced; for example, if an event occurs that makes state s hold at the instant between time t-1 and time t, then it can be represented as *occurs*(*ingr*(s), (t-1)*). This predicate can thus be defined as follows:

$(\neg s, t\text{-}1) \wedge (s, t)$ is shortened as *occurs*(*ingr*(s), (t-1)*), where $s \in S_I$.

The rule set R defines two subsets of rules to represent the temporal environment, same-time rules (R_s) and next-time rules (R_n), where $R_s \cup R_n = R$, which are defined as follows:

Same-time rules: $s_1 \wedge s_2 \wedge \ldots \wedge s_n \to s$;
Next-time rules: $s_1 \wedge s_2 \wedge \ldots \wedge s_n \to \mathrm{O}s$,

where $s_i \in S$ (i=1,…, n) is an atomic state and $s \in S_D$; $s_1 \wedge s_2 \wedge \ldots \wedge s_n \to s$ represents the influence of $s_1 \wedge s_2 \wedge \ldots \wedge s_n$ over s, and $s_1 \wedge s_2 \wedge \ldots \wedge s_n \to \mathrm{O}s$ represents delayed (next time) influence of $s_1 \wedge s_2 \wedge \ldots \wedge s_n$ over s.

Same-time rules must be *stratified*, i.e., they are ordered in such a way that the states in the body of a rule are either independent or dependent on states that are heads of rules in previous levels of the stratification [21]. This is explained as follows:

(1) A Stage k rule is a rule $s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow s$, where each s_i is at most (k-1)-*dependent*, and at least one s_i is (k-1)-dependent. In this case s is said to be k-*dependent*.

(2) All independent states are considered *0-dependent*.

Since all rules in R_s are stratified, every rule in R_s has a number k, which is used to indicate the stage of rules.

Compared with the exiting combined logic systems, in *MTPL* system, the truth value can be represented in both numerical and linguistic values, i.e., the truth value set V can be in [0, 1] interval or a finite ordered linguistic truth value set. In addition, *MTPL* allows the truth value not only existing in states, but also in rules, such that, while represented a status of state, it provided a tuple (s, t, v), which means the truth value of state s is v at *time t*, where $s \in S$, $t \in T$, $v \in V$; and a tuple (r, v) is used to represent the truth value of rule r is v, where $r \in R$, $v \in V$.

Many-valued logic within *MTPL* is based on Łukasiewicz logic. Łukasiewicz logic is a well-known many-valued logic studied in numerous papers on fuzzy and many-valued logic. More importantly, Pavelka [5] showed that the only natural way of formalising 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. Moreover, Liu *et al.* [23] introduced the linguistic truth-valued Łukasiewicz algebras in order to model linguistic values set and their operations using the Łukasiewicz algebra and logic system, about linguistic truth valued algebra and the corresponding logic system can be further referred to [43].

The truth-valued field of *MTPL* was defined as a Łukasiewicz implication algebra. In the following, Lukasiezicz algebra defined on the interval [0, 1], on a finite truth-value set, and on the linguistic truth-value set are given respectively.

Definition 3.1 Suppose a truth value set V=[0, 1], and let $L = < V, \wedge, \vee, \neg, \rightarrow, \otimes >$; $v_\alpha, v_\beta \in V$. If the operations are defined as follows:

$$v_\alpha \wedge v_\beta = Min(v_\alpha, v_\beta),$$
$$v_\alpha \vee v_\beta = Max(v_\alpha, v_\beta),$$
$$\neg v_\alpha = 1 - v_\alpha,$$
$$v_\alpha \rightarrow v_\beta = Min(0, 1 - v_\alpha + v_\beta),$$
$$v_\alpha \otimes v_\beta = Max(0, v_\alpha + v_\beta - 1).$$

Then L is called the Łukasiewicz implication algebra on [0, 1], where $\rightarrow$ is called the Łukasiewicz implication and $\otimes$ is called the Łukasiewicz product.

Definition 3.2 Suppose an ordered finite truth value set $V = \{v_0, v_1, ..., v_m\}$, and let L=<V, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\otimes$>; $v_\alpha, v_\beta \in V$. If the operations are defined as follows:

$$v_\alpha \wedge v_\beta = Min(v_\alpha, v_\beta),$$
$$v_\alpha \vee v_\beta = Max(v_\alpha, v_\beta),$$

$$\neg v_{\alpha} = v_{m-\alpha},$$
$$v_{\alpha} \rightarrow v_{\beta} = Min(0, v_{m-\alpha+\beta}),$$
$$v_{\alpha} \otimes v_{\beta} = Max(v_0, v_{\alpha+\beta-m}).$$

Then L is called the Łukasiewicz implication algebra on a finite chain V. Suppose V={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 short for $V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$. Then L=<V, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\otimes$> is called a Łukasiewicz linguistic truth-valued algebra on an ordered linguistic truth value set V.

A many-valued logic with truth-values defined in a Łukasiewicz linguistic truth-valued algebra on an ordered linguistic truth value set is called a linguistic truth-valued logic. A key insight of linguistic-valued based approaches reflects the use of "words" as computational variables, i.e., the symbolic approach acts by the direct computation, manipulation and reasoning on the available linguistic terms in natural languages [43]. Its application is beneficial because it introduces a more flexible qualitative framework for representing the 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. *MTPL* reasoning framework has a distinct advantage of handling both quantitative (truth-values in [0, 1]) and qualitative (linguistic truth-values) in quantitative and qualitative decision making.

The propositional calculus definition of MTPL follows a universal algebraic point of view [5, 6]. For more detailed definitions and introductions about language, syntax and semantic of MTPL the reader is referred to [24, 25, 26]

Definition 3.3 Let X be a set of propositional variables, L is a truth-valued algebra in Definition 3.1 or Definition 3.2, $TY=L\cup\{\neg, \rightarrow\}$ be a type with $ar(\neg)=1$, $ar(\rightarrow)=2$ and $ar(a)=0$ for every $a\in L$. The propositional algebra of the many-valued propositional calculus of MTPL on the set of propositional variables is the free T algebra on X and is denoted by $LP(X)$.

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

Definition 3.4 Let T be the time set. If a mapping γ: $LP(X) \times T\rightarrow L$ satisfies the following properties:

(1) For any $\alpha\in L$, $t\in T$, $\gamma(\alpha, t)=\alpha$;

(2) $\gamma$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 $\gamma(*, t_1) \not\equiv \gamma(*, t_2)$, where * means any state belong to S.

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

Definition 3.5 Let $p \in LP(X)$, $\alpha \in L$. If there exists a temporal valuation γ such that $\gamma(p) \geq \alpha$ at time t, then p is said to be α-satisfiable at time t. If $\gamma(p) \geq \alpha$ 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 $\alpha = 1$ at time t, then p is valid in time t.

One improvement of *MTPL* comparing with other combined logics is the inference rules systems of *MTPL* not only provide forward calculus, but also a backward calculus, i.e., we implemented two reasoning strategies, forward and backward inference. Such systems were supported by the soundness and completeness theorems [25] in *MTPL*. It means that, while implementing *MTPL* into a reasoning system to handle decision-making problems with uncertain information in certain temporal environment, it is able to provide both forward and backward calculation. So, the proposed reasoning system can provide simulation/prediction function based on forward calculation, and query answering function based on backward calculation, all supported by the *MTPL* system.

4. Reasoning Algorithms

Section 3 introduced the basic concepts that are considered in our system. This system has been implemented and tried in a variety of scenarios. This section will explain the algorithms we used to transform our theory into a useful tool and the next section will show how the system can be used in some of those scenarios. Our multi-valued temporal propositional logic is used to represent knowledge over domains which are dynamic and uncertain. This knowledge representation is declarative, a collection of rules and facts which embeds the relation between different concepts in the domain represented as well as the temporal relation between them and the degree of confidence on the knowledge over these relationships. We implemented two reasoning strategies, forward and backward inference. This section will explain how the forward and backward reasoning algorithms work and how they can be used. To explain the basic concepts associated with these algorithms we will use the following example:

<u>Scenario 1</u>:

Independent state: $\pm s_1$, $\neg s_2$, $\neg s_3$, $\neg s_4$
Dependent State: s_2, s_3, s_4
Same-time Rule: $(s_1 \rightarrow s_2, 1)$, $(s_2 \rightarrow s_3, 1)$
Next-time Rule: $(s_3 \rightarrow \mathrm{O}s_4, 1)$
Initial Setting: $I_c = \{(s_1, 0, 0), (s_2, 0, 0), (s_3, 0, 0), (s_4, 0, 0)\}$
Event: $occurs(ingr(s_1), 3^*, 1)$

4.1. *Forward Reasoning Algorithm*

This feature of our system allows users to simulate the behaviour of the system and explore the effect of different assumptions. This is an important aid to support decision-making by informing the decision makers of the possible consequences of the possible courses of action available.

This algorithm is an extension of a previous forward reasoning algorithm for stratified causal theory [21, 22] which only allowed reasoning based on two truth values: *true* or *false* and it assumed all the rules to have the same strength. This algorithm takes into consideration a more flexible and general representation of the world which includes time and also uncertainty associated with both, states and rules.

Input:

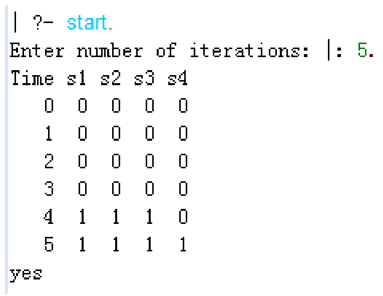
- a stratified set of same-time rules (R_s),
- a set of next-time rules (R_n),
- a set of truth value V with $m+1$ level $V=\{v_0, v_1, ..., v_m\}$, or V=[0,1].
- an initial setting (I_c), which are specified by determining $(s_i, 0, v_i)$ with $s_i \in S$, $v_i \in V$ and (r_j, v_j) with $r_j \in r$, $v_j \in V$,
- an event occurrence list (E), which is a set of formulae in the form $occurs(ingr(s), t^*, v)$.

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

Notice: A threshold λ is assumed and used to determine if a state has supportive evidence of holding (when its degree of truth is in $[\lambda, v_m]$) or not (when its degree of truth is in $[v_0, \lambda)$). We compute the resulting history as follows:

- At t=0, set all the truth value of states as stated in the Initial Setting declaration. Apply any live same-time rule on increasing level of stratification. Once all possible same-time rules were applied, then apply next-time rules.
- For t=1, 2, .3,…
  - (i) For each $occurs(ingr(s), (t-1)^*, v_i)$, if $(s, t-1, v_j)$ holds then assert (s, t, v_i), where $v_i, v_j \in V$.
  - (ii) For each independent state s, if $(s, t-1, v_j)$ holds and (s, t, v_i) was not asserted, then assert (s, t, v_j). This is called 'applying persistence' to the state s, where $v_i, v_j \in V$.
  - (iii) For k=1, 2, 3, …,
    - (1) Apply any live same-time rule of stage k
    - (2) Apply persistence: For any co-k-dependent state s, if (s, t, v_i) has not been asserted, and $(s, t-1, v_j)$, then assert (s, t, v_j), where $v_i, v_j \in V$.
  - (iv) Apply any live next-time rule.

Running the forward reasoning algorithm over Scenario 1 to explore the evolution of the system up to time=5 provides the results shown in Figure 1.



```
| ?- start.
Enter number of iterations: |: 5.
Time s1 s2 s3 s4
   0  0  0  0  0
   1  0  0  0  0
   2  0  0  0  0
   3  0  0  0  0
   4  1  1  1  0
   5  1  1  1  1
yes
```

Fig. 1. Forward reasoning algorithm applied to Scenario 1 for time=5

This figure shows the evolution of the truth values of all the states defined in scenario 1. The simulation shows the values for all states from the initial time 0 until time 5. The ingression to state s_1 after time=3, the change of value in s_1 affects triggers two same-time rules causing immediate effect in s_2 and s_3. Nothing affects s_4 at time-4. States s_2, s_2 and s_3 remain unchanged at time=5 whilst s_4 is affected by the delayed effect of the next-time rule which was activated at time=4 by s_3.

4.2. *Backward Reasoning Algorithm*

Whilst a forward reasoning strategy provides the foundation for a simulation tool there are situations where we need to explore a system in a different way, for example we may need to focus on a specific state and time during the evolution of the system. In these situations the enquiring process is more goal-directed and it explores the evolution of the system only in that part of the process that contributes to a specific output. This enquiry in our system takes the form of answering a query and this section explains the strategy and associated concepts followed to answer those queries.

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 a specific conclusion directly related to the query. The advantage of this strategy is that it only explores part of the Knowledge Base which is necessary to answer the query instead of the more comprehensive coverage of states time by time as it is needed in the forward reasoning approach.

We explain the whole backward reasoning strategy in three stages by introducing two auxiliary concepts: *Supporting Tree* (*ST*), and *Activation time list of ST* (*AcTimes*), before explaining the *search process*.

4.2.1. *Supporting Trees*

Definition 4.1: A *supporting tree of state s* (ST_s) is a structure such that:

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

For example, a tree with the structure shown in Figure 2 represents a rule $r \in R$, such that, $s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow s$.

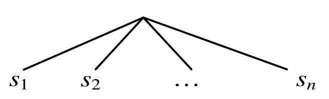


Fig. 2. A Supporting Tree Structure

A tree with the structure shown in Figure 3 represents two rules: $s'_1 \wedge s'_2 \wedge \ldots \wedge s'_n \rightarrow s_n$ and $s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow s$.

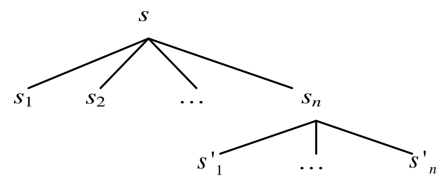


Fig. 3. A Multiple Supporting Tree Structure

We use the concept of supporting tree within our search strategy, each $Query(s,t,v)$ with $s \in S_D$ can have associated one or more *ST*s, where all non-leaf nodes belong to S_D and all leaf nodes belong to S_I:

Example 4.1: 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: ($s_1 \wedge s_2 \rightarrow s_3 , v_\alpha$), ($s_3 \wedge s_4 \rightarrow s_5 , v_\beta$), and ($s_5 \rightarrow s_6 , v_\theta$), then we can express it as the supporting tree structure:

T={($s_1 \wedge s_2 \rightarrow s_3 , v_\alpha$), ($s_3 \wedge s_4 \rightarrow s_5 , v_\beta$),($s_5 \rightarrow s_6 , v_\theta$)} with $v_\alpha, v_\beta, v_\theta \in V$.

Example 4.2: 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, ($s_1 \wedge s_2 \rightarrow \text{O}s_3 , v_\alpha$), ($s_3 \wedge s_4 \rightarrow \text{O}s_5 , v_\beta$), and ($s_5 \rightarrow \text{O}s_6 , v_\theta$), then we can express it as the following supporting tree structure:

T={($s_1 \wedge s_2 \rightarrow \text{O}s_3 , v_\alpha$), ($s_3 \wedge s_4 \rightarrow \text{O}s_5 , v_\beta$), ($s_5 \rightarrow \text{O}s_6 , v_\theta$)} with $v_\alpha, v_\beta, v_\theta \in V$.

Example 4.3: 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 $V \supseteq \{ v_{\beta 1} , v_{\beta 2} , v_{\beta 3} \}$. Assume that we have the following rules, ($s_1 \wedge s_2 \rightarrow s_3 , v_\alpha$), ($s_3 \wedge s_4 \rightarrow s_5 , v_\beta$), and ($s_5 \rightarrow \text{O}s_6 , v_\theta$), then we can express it as the following supporting tree structure:

T={($s_1 \wedge s_2 \rightarrow s_3 , v_\alpha$), ($s_3 \wedge s_4 \rightarrow s_5 , v_\beta$),($s_5 \rightarrow \text{O}s_6 , v_\theta$)} with $v_\alpha, v_\beta, v_\theta \in V$.

4.2.2. *Activation Time List*

The list of activation times holds the times where meaningful event occurred. This focuses the search only on meaningful stages of the evolution of the system out of a potentially long list of times where something occurred. To create the *AcTimes* list, two parts of the Knowledge Base are used: the list of the independent states in *ST* and the event list *E*. The independent states in *ST* are called *activation points* of a given *ST* because the rules that form the *ST* can be triggered or 'activated' by those independent states. Once we know which independent states are part of an *ST* we can collect from the KB the times where they are activated (restricted to the time of the query).

By 'Obtain list *AcTimes* of Activation Times' we will mean *AcTimes* is defined as the decreasing order list of times in a tree. This can be generalized in the case of two trees by obtaining a list *AcTimes* that results of merging the two activation times lists for ST_s and $ST_{\neg s}$, denoted as $AcTimes_s$ and $AcTimes_{\neg s}$ respectively. For example if the rules in ST_s are triggered by events occurring at times 1, 3 and 5, then this forms a list [5, 3, 1]; and if the rules in $ST_{\neg s}$ are triggered by events occurring at times 2, 2 and 5 then this forms a list [5, 2, 2]. The result of the combined *AcTimes* list is [5, 5, 3, 2, 2, 1]. Notice that 0 is always included in *AcTimes* because implicitly or explicitly values for all states are set at the intended beginning of the system, hence for this example, *AcTimes*=[5, 3, 2, 2, 1, 0].

By '*t* becomes the next available time closest to time in *AcTimes*' we will mean *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, 3, 2, 2, 1, 0].

4.2.3. *Backward Reasoning Algorithm*

The general idea is as follows. Suppose there is a query about state *s* given as $Query(s,t,v_\alpha)$, then:

Step 1:

(i) Form all relevant trees (an independent state is a minimal tree),

(ii) Find those trees supporting that *s* is above threshold and those trees supporting that *s* is below threshold,

Step 2: Choose a tree and

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

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

(v) Use *s* as start, and search *ST* backwards until it reaches activation points (independent states) of *ST*; for each dependent state run this process recursively

Step 3:

(vi) According on whether the winner *ST* has same sign (i.e., above/below threshold) as the query, or not, then returns 'true' or 'false', respectively

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

(viii) Return the answer of the query.

Before explaining the most detailed version we need to introduce some notation. Suppose that we want to know whether $Query([\neg]s,t,v_\alpha)$ is true or not (i.e., whether $([\neg]s,t,v_\beta): v_\beta \geq v_\alpha$ or not). Here we use $[\neg]s$ as an abbreviation for s or $\neg s$. We represent the threshold by $\lambda \in V$, one of many possible linguistic values or a truth value in interval [0,1], which is considered as the minimum level of credibility for a proposition (a 'credibility threshold'). For a tree to be activated each rule $(s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow s,\ v_\beta)$ or $(s_1 \wedge\ s_2 \wedge \ldots \wedge s_n \rightarrow \mathrm{O}s,\ v_\tau)$ in the tree should be such that for all s_i:

(1) (s_i, t, v_i), $v_i \in V$, $v_\theta = Min_i(v_i) \geq v_{m+\alpha-\beta} \geq \lambda$, or
$v_\theta = Min_i(v_i) \geq v_{m+\alpha-\tau} \geq \lambda$, where, i=1,…, n, and $V = \{v_0, v_1, ..., v_m\}$;

or

(2) (s_i, t, v_i), $v_i \in V$, $v_\theta = Min_i(v_i) \geq 1 + v_\alpha - v_\beta \geq \lambda$, or
$v_\theta = Min_i(v_i) \geq 1 + v_\alpha - v_\tau \geq \lambda$, where, i=1, …, n, and V=[0, 1].

Inferences in our system are performed based on a set of inference rules which are then used to guide the implementation of our algorithms. The structure of those rules dictate the way the final value of credibility attached to an inference. We cannot describe the whole theory here so we will list the different ways of computing the values and refer the reader to [26] for a more detailed explanation of the logical machinery used by our system.

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. Also to be noticed is that if the supporting tree is only made of one independent state, say s, then the ST should only be the state itself, i.e., $ST_s = \{s\}$. The full backward reasoning algorithm is provided below:

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 $V = \{v_0, v_1, ..., v_m\}$ or V=[0,1].
- a set of facts called initial conditions I_c provides the truth values of states in $S = S_I \cup S_D$ at time=0
- a set of known events, E, describing state ingressions.
- a credibility threshold $\lambda \in V$.
- $Query([\neg]s, t, v_\alpha)$.

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

```
1. IF $([\neg]s, t, v_\mu) \in I_c$ or $occurs(ingr([\neg]s), (t-1)^*, v_\mu) \in E$
   THEN IF $v_\mu \geq v_\alpha$
        THEN answer 'true' and give fact as explanation.
        ELSE answer 'false' and give fact as explanation.
2. ELSE ($Query([\neg]s, t, v_\alpha)$ must be inferred by deduction, or 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 $time$=$Max(AcTimes)$ and $time \leq t$.
```

```
    REPEAT
    (a) IF $ST_s \cup ST_{\neg s} \neq \varnothing$, THEN Select the first tree from $ST_s$ or $ST_{\neg s}$.
      (i) find a main rule $r$ in the tree
        (A) IF $r$: $(s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow s, v_\beta) \in R_s$ and $v_\beta \geq v_\alpha$ THEN
          * For each $s_i \in S_I$ : $Query(s_i, time, v_\theta)$ where $v_\theta \geq v_{m+\alpha-\beta}$ if
            $V = \{v_0, v_1, \ldots, v_m\}$ or $v_\theta \geq 1 + v_\alpha - v_\beta$ if $V$=[0,1], or
          ** For each $s_d \in S_D$ : $Query(s_d, t, v_\theta)$ where $v_\theta \geq v_{m+\alpha-\beta}$ if
            $V = \{v_0, v_1, \ldots, v_m\}$ or $v_\theta \geq 1 + v_\alpha - v_\beta$ if $V$=[0,1].
        (B) IF $r$: $(s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow \text{O}s, v_\tau) \in R_n$ and $v_\tau \geq v_\alpha$ THEN
          * For each $s_i \in S_I$ : $Query(s_i, t', v_\theta)$ where $t' = Max(AcTimes \leq t-1)$
            and $v_\theta \geq v_{m+\alpha-\tau}$ if $V = \{v_0, v_1, \ldots, v_m\}$ or $v_\theta \geq 1 + v_\alpha - v_\tau$ if $V$=[0,1],
            or
          ** For each $s_d \in S_D$ : $Query(s_d, t-1, v_\theta)$ where $v_\theta \geq v_{m+\alpha-\tau}$ if
            $V = \{v_0, v_1, \ldots, v_m\}$ or $v_\theta \geq 1 + v_\alpha - v_\tau$ if $V$=[0,1]
      (ii) IF $[\neg]s$ can be proved true from a tree
        THEN answer 'true' ['false'] and use the tree as the explanation
        ELSE select next tree from the sets of trees $ST_s$ or $ST_{\neg s}$.
    (b) ELSE $Query(s_i, t'', v_\alpha)$, where $t'' = Max(AcTimes < time)$
  UNTIL ($[\neg]s$ can be proved)
```

Given same time rules cannot be cyclic and next time rules decrease time until an answer is reached (worst case at t=0), then we have the following result.

Theorem 4.1 Termination of all queries to the backward reasoning algorithm is guaranteed.

Proof: Basic Case: $Query([\neg]s, 0, v_\alpha)$ is answered by the I_c. Hypothesis (1): assume the thesis is valid for $t > 0$, it always finishes (t is a finite number). Then for $time$=t+1 (time is finite): assume a $Query([\neg]s, t, v_\alpha)$, then the algorithm can proceed only as in any of the following cases (2):

(a) There is an event from E, which is $occurs(ingr([\neg]s), (t-1)^*, v_\mu) \in E$, if $v_\mu \geq v_\alpha$, then answer 'true', otherwise, answer 'false' and the algorithm terminates.

(b) There is a same-time rule $(s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow s, v_\beta)$, 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 $(s_1 \wedge s_2 \wedge \ldots \wedge s_n \rightarrow \text{O}s, v_\tau)$, the answer will depend on queries, $Query(s_i, t', v_\theta)$ if $s_i \in S_I$ where $t' = Max(AcTimes \leq t-1)$ or $Query(s_d, t-1, v_\theta)$ if $s_d \in S_D$, where $d = 1, \ldots, n$ and $v_\theta \geq v_{m+\alpha-\tau}$ if $V = \{v_0, v_1, \ldots, v_m\}$ or $v_\theta \geq 1 + v_\alpha - v_\tau$ if V=[0,1], and by the induction hypothesis they all finish.

(d) Neither (2a), (2b), or (2c) applies, then assumes that the last occurrence of an event changed s occurred in the past, the value of $Query(s, t, v_\alpha) = Query(s_i, t'', v_\alpha)$, where $t'' = Max(AcTimes)$.

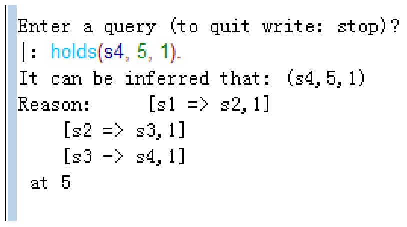
Notice both (c) and (d) always change the truth value of states at an earlier time. Hence, if there is no explicit data, then the algorithm eventually reaches 0 where the query answer is provided by I_c. Reaching 0 cannot be jumped over as time in our system is discrete and always jumped to a legal decreasing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

Figure 6 shows the result of running the backward reasoning algorithm with a simple example: the query "*holds*(s_4, 5, 1)?", where we ask if the truth value of s_4 at time=5 is equal or exceeds 1 (which in this case represents a value $v_1 \in V$)

5. System Implementation

The previous section provided algorithms for both forward and backward reasoning strategies. These algorithms are implemented in Prolog and as such they can be run stand alone or be integrated to other programs, e.g. in Java, which can call our program as a reasoning module through a call to one of the main predicates. This section provides a deeper insight in the implementation and use of the system.

5.1 Implementation of Key Concepts



```
Enter a query (to quit write: stop)?
|: holds(s4, 5, 1).
It can be inferred that: (s4,5,1)
Reason:     [s1 => s2,1]
    [s2 => s3,1]
    [s3 -> s4,1]
 at 5
```

Fig. 6. The result of simple example running with backward reasoning algorithm

First we would like to provide some details of the implementation of some core concepts of the system. The program itself, without any knowledge base added, is several pages long so we focus on a few predicates which implement some key concepts we mentioned above. We omit secondary predicates. The full program code is available under request to the authors.

Stratify is a predicate to stratify all the rules from stage 0 to stage *n*. *stage*_0 is used to gather independent states, *stage*_1 is used to gather rules in the knowledge base which depend on stage_0 states, *stage*_n is used to gather rules in the knowledge base which depend on states belonging to previous levels and achieves that through a recursive call.

```
% stratify(-ISL, -SER_SSRL, +SL).
% Given the Independent States List, ISL, and a Next-time
% Rules plus Same-Time Rules List, SER_SSRL, the procedure
% find the Stratified Same-Time Rules List, SL.
```

```
stratify(ISL, [], SL).
stratify(ISL, SER_SSRL, SL) :-
        stage_0(ISL, S0L),
        stage_1(ISL, SER_SSRL, SER_SSRL2, S1L),
        appendL(S0L,S1L, S01L),
        stage_n(SER_SSRL2, S01L, 2, SL).
stage_0([], []).
stage_0([IS|T], [[[true => IS, _], 0]|S0T]):-  stage_0(T, S0T).
stage_1(ISL, [], [], []).
stage_1(ISL, [[R, RV]|RT], SER_SSRL2, [[[R, RV] ,1]|S1T]):-
        bodyIRInISL(R, ISL),
        stage_1(ISL, RT, SER_SSRL2, S1T).
stage_1(ISL, [R|RT], [R|R2T], S1L):-
        stage_1(ISL,RT,R2T,S1L).
stage_n([], Labeled, _, Labeled).
stage_n(ToBeLabeled, SoFarLabeled, Stage, Labeled):-
        s_n(ToBeLabeled, SoFarLabeled, Stage, NewToBeLabeled,
         NewLabeled),
        appendL(SoFarLabeled, NewLabeled, Last),
        NewStage is Stage+1,
        stage_n(NewToBeLabeled, Last, NewStage, Labeled).
```

The predicate *getTrees* generate a list of supporting trees for a particular state. In this predicate, *getRulesForState* is used to collect all the rules in knowledge base containing *State* as head and place them in a list 'Rules'; *getDependentsList* is used to find out all dependent states in the body of rules in 'Rules' and list them in 'DL'; *getNextLevelTrees* is used to find out all the supporting trees of dependent states in 'DL' and list them in 'AllTrees'. *mergeTrees* will complete the final step, to merge all the supporting trees in 'AllTrees' and list them in 'Trees' which collects all supporting trees of *State*.

```
% getTrees(+State, +IS, +SERLSL, -Trees) find all the trees that
% support State and add to each rule of the tree the set of
% independent states used in that rule.
getTrees(State, IS, [], []).
getTrees(State, IS, SERLSL, Trees):-
   getRulesForState(State, SERLSL, Rules),
   getDependentsList(Rules, IS, DL),
   getNextLevelTrees(DL, IS, SERLSL, AllTrees),
   mergeTrees(AllTrees, Rules, IS, Trees).
```

The Predicate *addTimetoTrees* is a predicate to search all the activation times of all supporting trees from *getTrees*. *addTime* is used to find out all the event occurring time slots of independent states in 'NewTrees' and list them in 'SomeTreesWithTimes';

getOnlySupportedTrees is used to pick up all the supporting trees containing activation time list and list them into 'TreesWithTimes'.

```
% addTimetoTrees(+TreesWithISDS, +IS, +Time, -LT)
% Add a list of times to each 3-upla
% [Tree, ListofISforTree, ListOFDSForTree] giving a 4-upla
% [Tree, ListofISforTree, ListofDSforTree, ListofTimesforISinTree]
% The process is done from 0 up to Time, both inclusive and the
% resulting list expanded of updated trees is LT.  All trees which
% do not have attached at least one time greater or equal to 0
% where all IS are true, must be eliminated.
addTimetoTrees([], IS, Time, []).
addTimetoTrees(Trees, IS, Time, TreesWithTimes):-
   addTimePlace(Trees, NewTrees),
   addTime(NewTrees, IS, -1, Time, [], SomeTreesWithTimes),
   getOnlySupportedTrees(SomeTreesWithTimes, TreesWithTimes), !.
```

5.2 Illustration through Scenarios

Now let us introduce a scenario that will be used to illustrate how to use the system and to show how the system reacts to different types of queries. These samples will be later complemented in the next section showing how the system can be used with more complex engineering problems.

<u>*Scenario 2:*</u> consider the task is to model a kitchen monitored by sensors in a Smart Home system [44, 45]. Let us assume the cooker is on (*cookerOn* represents a sensor detecting cooker being activated), but the motion sensor is not activated ($\neg$ *atKitchen*, *atKitchen* is a sensor detecting location of a person in the kitchen). If no motion is detected after more than a number of *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 occupant. In this scenario, the cooker and the occupant are both monitored by sensors, but sensors are not always reliable and rules may not always capture reality literally so there are elements of uncertainty that have to be taken into account in the representation of the problem. For example, due to the likely malfunction of sensor *atKitchen*, we can only assume that the house occupant is in the kitchen with *e.g.*, 80% certainty, or with 'high' (but not full) 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 set $V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$ identifies the truth-valued level of the status of states interpreted as: {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}. We assume the following description for the scenario:

Independent States: *cookerOn*, ±*atKitchen*, *umt3u*
Dependent States: ±*cu*, ±*hazzard*, ±*alarmOn*, $\neg$ *umt3u*, $\neg$ *cookerOn*
Same-time Rules:
Stratification Level 1 Rules:

$(\neg\, atKitchen \wedge cookerOn \wedge umt3u \rightarrow cu,\, v_6)$
$(\neg\, cookerOn \rightarrow \neg\, alarmOn,\, v_6)$
$(\neg\, cookerOn \rightarrow \neg\, hazzard,\, v_6)$
$(\neg\, cookerOn \rightarrow \neg\, umt3u,\, v_6)$
$(\neg\, cookerOn \rightarrow \neg\, cu,\, v_6)$

Stratification Level 2 Rules:

$(cu \rightarrow alarmOn,\, v_6)$
$(cu \rightarrow hazzard,\, v_6)$

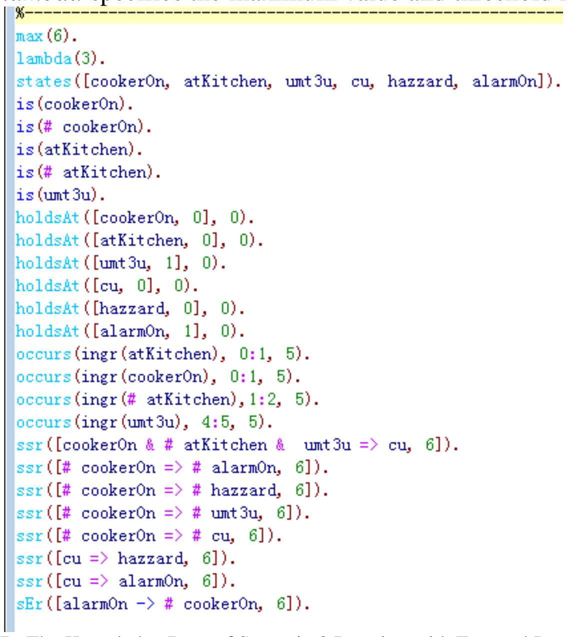
Next-time Rules:

$(alarmOn \rightarrow \mathrm{O}(\neg\, cookerOn),\, v_6)$

Events: $occurs(ingr(atKitchen), 0{:}1, v_5)$
$occurs(ingr(cookerOn), 0{:}1, v_5)$
$occurs(ingr(\neg atKitchen), 1{:}2, v_5)$
$occurs(ingr(umt3u), 4{:}5, v_5)$

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 $\lambda = v_3$.

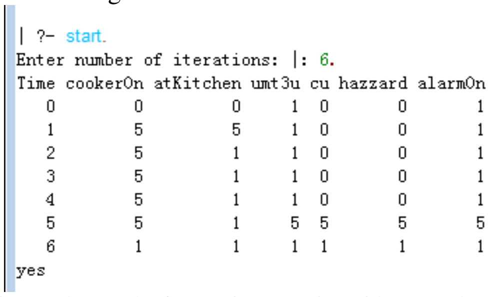
This scenario is used to test whether the system is capable to turn on the alarm and shut down the cooker automatically when the cooker is unattended for a period of time judged to be unsafe. The Forward Reasoning algorithm is run over Scenario 2 as depicted in Fig. 7, *max* and *lambda* specifies the maximum value and threshold respectively.



```
%---------------------------------------------------------
max(6).
lambda(3).
states([cookerOn, atKitchen, umt3u, cu, hazzard, alarmOn]).
is(cookerOn).
is(# cookerOn).
is(atKitchen).
is(# atKitchen).
is(umt3u).
holdsAt([cookerOn, 0], 0).
holdsAt([atKitchen, 0], 0).
holdsAt([umt3u, 1], 0).
holdsAt([cu, 0], 0).
holdsAt([hazzard, 0], 0).
holdsAt([alarmOn, 1], 0).
occurs(ingr(atKitchen), 0:1, 5).
occurs(ingr(cookerOn), 0:1, 5).
occurs(ingr(# atKitchen),1:2, 5).
occurs(ingr(umt3u), 4:5, 5).
ssr([cookerOn & # atKitchen &  umt3u => cu, 6]).
ssr([# cookerOn => # alarmOn, 6]).
ssr([# cookerOn => # hazzard, 6]).
ssr([# cookerOn => # umt3u, 6]).
ssr([# cookerOn => # cu, 6]).
ssr([cu => hazzard, 6]).
ssr([cu => alarmOn, 6]).
sEr([alarmOn -> # cookerOn, 6]).
```

Fig. 7. The Knowledge Base of Scenario 2 Running with Forward Program

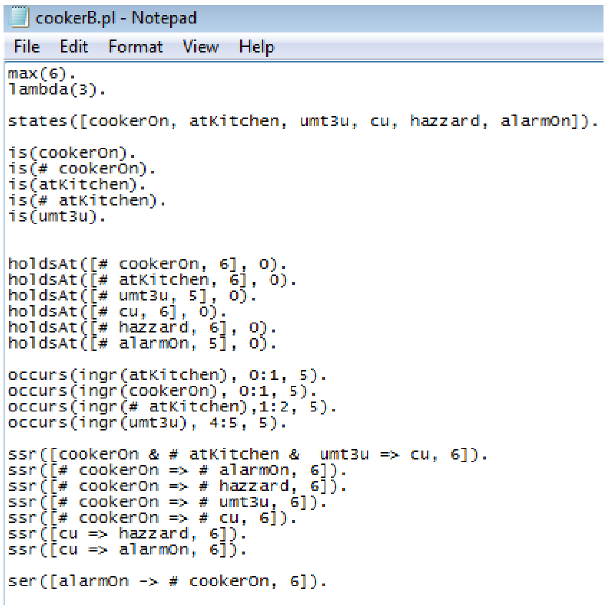
Invoking the main predicate triggers a prompt requesting the maximum time slot to be investigated and that will be enough for the forward program to roll up all the states and show the result as shown in Figure 8.



```
| ?- start.
Enter number of iterations: |: 6.
Time cookerOn atKitchen umt3u cu hazzard alarmOn
   0        0         0     1  0       0       1
   1        5         5     1  0       0       1
   2        5         1     1  0       0       1
   3        5         1     1  0       0       1
   4        5         1     1  0       0       1
   5        5         1     5  5       5       5
   6        1         1     1  1       1       1
yes
```

Fig. 8. The Result of Scenario 2 Running with Forward Program

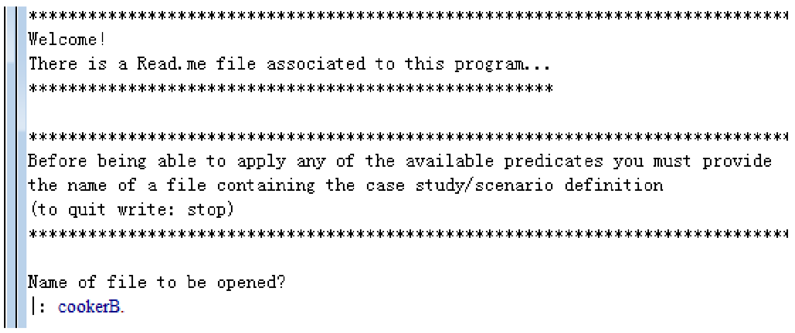
The Backward Program needs a slightly different procedure as we need to specify a query. We also want to illustrate queries which relate to different type of rules that can be used in our system. Figure 9 offers a revised KB, notice the changes in the "holdsAt" predicates.



```
max(6).
lambda(3).

states([cookerOn, atKitchen, umt3u, cu, hazzard, alarmOn]).

is(cookerOn).
is(# cookerOn).
is(atKitchen).
is(# atKitchen).
is(umt3u).


holdsAt([# cookerOn, 6], 0).
holdsAt([# atKitchen, 6], 0).
holdsAt([# umt3u, 5], 0).
holdsAt([# cu, 6], 0).
holdsAt([# hazzard, 6], 0).
holdsAt([# alarmOn, 5], 0).

occurs(ingr(atKitchen), 0:1, 5).
occurs(ingr(cookerOn), 0:1, 5).
occurs(ingr(# atKitchen),1:2, 5).
occurs(ingr(umt3u), 4:5, 5).

ssr([cookerOn & # atKitchen &  umt3u => cu, 6]).
ssr([# cookerOn => # alarmOn, 6]).
ssr([# cookerOn => # hazzard, 6]).
ssr([# cookerOn => # umt3u, 6]).
ssr([# cookerOn => # cu, 6]).
ssr([cu => hazzard, 6]).
ssr([cu => alarmOn, 6]).

ser([alarmOn -> # cookerOn, 6]).
```

Fig. 9. The Knowledge Base of Scenario 2 Running with Backward Program

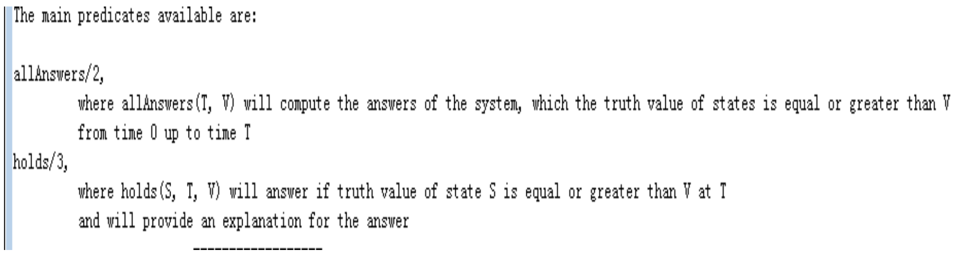
The program will ask the name of the file containing the scenario specification (knowledge base, initial setting and event list) as shown in Figure 10.



```
******************************************************************************
Welcome!
There is a Read.me file associated to this program...
*****************************************************

******************************************************************************
Before being able to apply any of the available predicates you must provide
the name of a file containing the case study/scenario definition
(to quit write: stop)
******************************************************************************

Name of file to be opened?
|: cookerB.
```

Fig. 10. Loading Knowledge Base of Scenario 2 Running with Backward Program

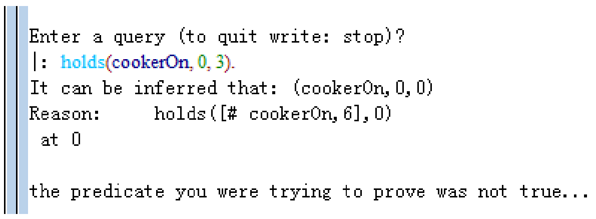
The program will then offer the two main options available to the user (see Figure 11). Notice the first one is really a way to the same that the forward reasoning option does and it is mainly for the development team to check that simulations in the backward program produce the same outcome than what we can retrieve with multiple queries in the backward reasoning strategy. The interesting option here is the predicate *holds/3* which allows making queries.



```
The main predicates available are:

allAnswers/2,
        where allAnswers(T, V) will compute the answers of the system, which the truth value of states is equal or greater than V
        from time 0 up to time T
holds/3,
        where holds(S, T, V) will answer if truth value of state S is equal or greater than V at T
        and will provide an explanation for the answer
                        ------------------
```

Fig. 11. The Explanations of Functions in Backward Program

For example, if the user wants to know if the cooker is on or not at the beginning, then *holds*(*cookerOn*, 0, 3) provides the feedback. as shown in Figure 12.

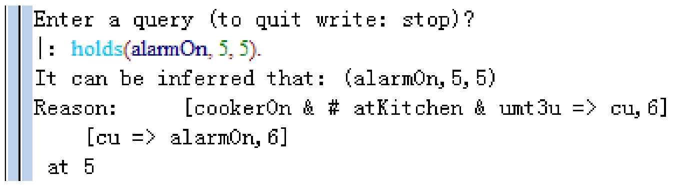


```
Enter a query (to quit write: stop)?
|: holds(cookerOn, 0, 3).
It can be inferred that: (cookerOn,0,0)
Reason:     holds([# cookerOn,6],0)
 at 0

the predicate you were trying to prove was not true...
```

Fig. 12. The Result and Explanation of Query *holds*(*cookerOn*, 0, 3) Running with the Knowledge Base of Scenario 2 and Backward Program

Notice the query challenges the system to prove 'the cooker was on' has strength of level 3 at time 0, i.e. it is unknown whether the cooker was or not on at the beginning. The answer of the system is that it is not true that it was unknown, as the system has a fact to offer to support the conclusion that in fact the cooker was off at 0 (i.e., it knows with strength v_6=true that ¬ cookerOn was the case).

If we wants to know whether we can believe with high confidence that the alarm is on at 5, then we can use the following query: *holds*(*alarmOn*, 5, 5) and the program will provide feedback as shown in Figure 13.

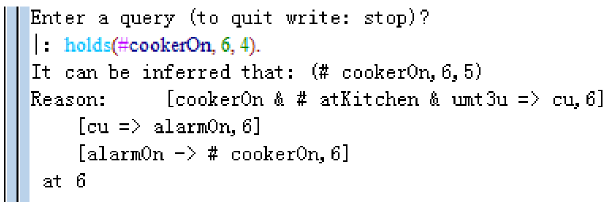


```
Enter a query (to quit write: stop)?
|: holds(alarmOn, 5, 5).
It can be inferred that: (alarmOn,5,5)
Reason:     [cookerOn & # atKitchen & umt3u => cu,6]
    [cu => alarmOn,6]
 at 5
```

Fig. 13. The Result and Explanation of Query *holds*(*alarmOn*, 5, 5) Running with the Knowledge Base of Scenario 2 and Backward Program

which indicates that the system proved that is the case and offers two rules which define the support tree to prove that *alarmOn* can be believed with confidence level of v_5 at 5.

If we want to find out whether we can be highly confident the cooker is off at 6, then we can use *holds*(*#cookerOn*, 6, 4) to obtain feedback as shown in Figure 14.



```
Enter a query (to quit write: stop)?
|: holds(#cookerOn, 6, 4).
It can be inferred that: (# cookerOn,6,5)
Reason:     [cookerOn & # atKitchen & umt3u => cu,6]
    [cu => alarmOn,6]
    [alarmOn -> # cookerOn,6]
 at 6
```

Fig. 14. The Result and Explanation of Query *holds*(*#cookerOn*, 6, 4) Running with the Knowledge Base of Scenario 2 and Backward Program

which indicates the system is capable to prove (even with higher confidence than what we requested, e.g., v_4='probably true') that the cooker is not on at 6.

6. Engineering a Smart Home

This section explains how the system can be used to engineer a Smart Home system. The scenario depicts activities of daily living from a real Smart Home [46] which can be used to monitor activities of daily living:

(i) Make a phone call. The participant moves to the dining room, looks up a specific number in the phone book, dial the number, and listen to the message. The recorded message provides cooking directions, which the participant summarises on a notepad.

(ii) Wash hands. The participant moves into the kitchen sink and washes his/her hands in the sink, using hand soap and drying their hands with a towel.

(iii) Cook. The participant cooks a pot of oatmeal according to the directions obtained. To cook the oatmeal the participant must measure water, pour the water into a pot and boil it, add oats, then put the oatmeal into a bowl with raisins and brown sugar.
(iv) Eat. The participant takes the oatmeal and a medicine container to the dining room and eats the food.
(v) Clean. The participant takes all of the dishes to the sink and cleans them with water and dish soap in the kitchen.

Sensors (see Table 1) distributed in the kitchen monitor these activities, through the use of cooking materials.

Table 1: categories of sensors for kitchen usage scenario

I01- I05:	item sensors for oatmeal, raisins, brown sugar, bowl, measuring spoon
I06:	medicine container sensor
I07:	pot sensor
I08:	phone book sensor
D01:	cabinet sensor
AD1-A/B	water sensors
AD1-C:	burner sensor
asterisk:	phone usage

The following sections illustrate how the system can be used to refine the knowledge base that can be used to command the decision making of the monitoring module for the smart home described above.

6.1. *Initial Knowledge Base*

We start with a specification of the problem which includes the use of sensors, e.g., RFID tags and readers, which can be used to identify the different objects and elements used in the kitchen like oatmeal, raisins, sugar, medicine, pot, spoon, bowl, book, cabinet, water tap A, water tap B, burner, and phone. The aim of this case study is to build up a knowledge base containing states and rules through the description of the scenario, and use the reasoning system to examine such knowledge base with a real dataset with a log of sensor activations. The developing team can then improve the knowledge base to make it more appropriate as part of the monitoring module of the Smart Home system. We represent the states triggered by the different sensors as follows:

oatmealS stands for the sensor monitoring the container of oatmeal.
raisinsS stands for the sensor monitoring the container of raisins.
sugarS stands for the sensor monitoring the container of sugar.
medicineS stands for the sensor monitoring the container of medicine.
potS stands for the sensor monitoring the pot.
spoon stands for the sensor monitoring spoon.
bowl stands for the sensor monitoring bowl.
book stands for the sensor monitoring book.
cabinet stands for the sensor monitoring cabinet.
waterA stands for the sensor monitoring water tap A.

waterB stands for the sensor monitoring water tap B.
burner stands for the sensor monitoring burner.
phone stands for the sensor monitoring phone.

Other situations are also represented with states:

oatmealR is used to represent the occupant has put some oatmeal into the bowl, and it is ready to cook.
raisinsR is used to represent the occupant has put some raisins into the bowl, and it is ready to cook.
sugarR is used to represent the occupant has put some sugar into the bowl, and it is ready to cook.
medicineR is used to represent the occupant has put some pills from the container of medicine and it is ready to be taken.
potFillWithWater means that the occupant has filled the pot full with water and is ready to be boiled.
waterBoiled means the water has been boiled.
food means the food is ready.
eat means the occupant starts to have his/her dinner.
full means the occupant has finished his/her dinner.
clean means the occupant starts washing bowl, pot, and spoon, and put them back to the cabinet.
processFinished is used to represent that the whole cooking and dining process has been finished.

According to the description of the scenario we initially considered the following rules:

Same-Time Rules:

$book \rightarrow \neg processFinished$
$oatmealS \wedge cabinet \rightarrow oatmealR$
$raisinsS \wedge cabinet \rightarrow raisinsR$
$sugarS \wedge cabinet \rightarrow sugarR$
$medicineS \wedge cabinet \rightarrow medicineR$
$potS \wedge waterA \rightarrow potFillWithWater$
$burner \wedge potFillWithWater \rightarrow waterBoiled$
$food \rightarrow \neg oatmealR$
$food \rightarrow \neg raisinsR$
$food \rightarrow \neg sugarR$
$food \rightarrow \neg potFillWithWater$
$food \rightarrow \neg waterBoiled$
$medicineR \wedge food \rightarrow eat$
$processFinished \rightarrow \neg full$
$processFinished \rightarrow \neg clean$

Next-Time Rules:

$book \rightarrow \text{O} phone$
$\neg phone \rightarrow \text{O} \neg book$
$spoon \wedge bowl \wedge oatmealR \wedge raisinsR \wedge sugarR \wedge waterBoiled \rightarrow food$

$eat \rightarrow \mathrm{O}\neg food$
$eat \rightarrow \mathrm{O}\neg medicineR$
$eat \rightarrow \mathrm{O}\,full$
$full \rightarrow \mathrm{O}\neg eat$
$full \rightarrow \mathrm{O}\,clean$
$clean \wedge \neg cabinet \wedge \neg oatmealS \wedge \neg raisinsS \wedge \neg medicineS \wedge \neg waterA \wedge \neg waterB \wedge \neg burner \rightarrow \mathrm{O}\,processFinished$

Independent State:
±oatmealS, ±raisinsS, ±sugarS, ±bowl, ±spoon, ±medicineS, ±potS, ±*book*, ±waterA ±waterB, ±phone, ±burner, ±cabinet, full, clean, *processFinished*, ¬ food, ¬ medicineR
Dependent State:
±eat, food, ±potFillWithWater, ±oatmealR, ±raisinsR, ±sugarR, ±waterBoiled, medicineR, ¬ full, ¬ clean, ¬ *processFinished*

The sensor log includes values in the range 0-100 and in our case these values are mapped to the truth value interval [0, 1]. We assume the initial truth value of all states is 0 except *processFinished* to be 1, and the truth value of all rules is 1, such that, we have the following:

Initial Setting:
I_c={(*oatmealS*, 0, 0), (*raisinsS*, 0, 0), (*sugarS*, 0, 0), (*medicineS*, 0, 0), (*potS*, 0, 0), (*spoon*, 0, 0), (*bowl*, 0, 0), (*book*, 0, 0), (*cabinet*, (*waterA*, 0, 0), (*waterB*, 0, 0), (*burner*, 0, 0), (*oatmealR*, 0, 0), (*raisinsR*, 0, 0), (*sugarR*, 0, 0), (*medicineR*, 0, 0), (*potFillWithWater*, 0, 0), (*waterBoiled*, 0, 0), (*food*, 0, 0), (*phone*, 0, 0), (*eat*, 0, 0), (*full*, 0, 0), (*clean*, 0, 0), (*processFinished*, 0, 1)}

Same-Time Rules:
($book \rightarrow \neg processFinished$, 1)
($oatmealS \wedge cabinet \rightarrow oatmealR$, 1)
($raisins \wedge cabinet \rightarrow raisinsR$, 1)
($sugars \wedge cabinet \rightarrow sugarR$, 1)
($medicines \wedge cabinet \rightarrow medicineR$, 1)
($pots \wedge waterA \rightarrow potFillWithWater$, 1)
($burner \wedge potFillWithWater \rightarrow waterBoiled$, 1)
($food \rightarrow \neg oatmealR$, 1)
($food \rightarrow \neg raisinsR$, 1)
($food \rightarrow \neg sugarR$, 1)
($food \rightarrow \neg potFillWithWater$, 1)
($food \rightarrow \neg waterBoiled$, 1)
($medicineR \wedge food \rightarrow eat$, 1)
($processFinished \rightarrow \neg full$, 1)
($processFinished \rightarrow \neg clean$, 1)

Next-Time Rules:

$(book \rightarrow \bigcirc phone, 1)$
$(\neg phone \rightarrow \bigcirc \neg book, 1)$
$(spoon \wedge bowl \wedge oatmealR \wedge raisinsR \wedge sugarR \wedge waterBoiled \rightarrow \bigcirc food, 1)$
$(eat \rightarrow \bigcirc \neg food, 1)$
$(eat \rightarrow \bigcirc \neg medicineR, 1)$
$(eat \rightarrow \bigcirc full, 1)$
$(full \rightarrow \bigcirc \neg eat, 1)$
$(full \rightarrow \bigcirc clean, 1)$
$(clean \wedge \neg cabinet \wedge \neg oatmealS \wedge \neg raisinsS \wedge \neg medicineS \wedge \neg waterA \wedge \neg waterB \wedge \neg burner \rightarrow \bigcirc processFinished, 1)$

At this stage, the whole knowledge base has been built up, including independent and dependent states, same-time rules, next-time rules, and initial setting. Moreover, to test it, we need to use real dataset to generate events shown as below, so that we can simulate the whole process through our reasoning system and see whether the feedback matches what we expect.

Events:

$occurs(ingr(book), 0{:}1, 1).$	$occurs(ingr(\neg phone), 2{:}3, 1).$
$occurs(ingr(cabinet),6{:}7, 1).$	$occurs(ingr(oatmealS), 7{:}8, 1).$
$occurs(ingr(raisinsS), 7{:}8, 1).$	$occurs(ingr(\neg oatmealS), 7{:}8, 1).$
$occurs(ingr(spoon), 7{:}8, 1).$	$occurs(ingr(\neg cabinet), 8{:}9, 1).$
$occurs(ingr(waterB), 9{:}10, 1).$	$occurs(ingr(\neg waterB), 10{:}11, 1).$
$occurs(ingr(potS), 10{:}11, 1).$	$occurs(ingr(cabinet),11{:}12, 1).$
$occurs(ingr(waterA),11{:}12, 1).$	$occurs(ingr(oatmealS), 11{:}12, 1).$
$occurs(ingr(\neg cabinet), 13{:}14, 1).$	$occurs(ingr(cabinet),19{:}20, 1).$
$occurs(ingr(\neg cabinet),20{:}21, 1).$	$occurs(ingr(cabinet),22{:}23, 1).$
$occurs(ingr(\neg cabinet),22{:}23, 1).$	$occurs(ingr(cabinet),23{:}24, 1).$
$occurs(ingr(sugarS),24{:}25, 1).$	$occurs(ingr(\neg sugarS),25{:}26, 1).$
$occurs(ingr(bowl),26{:}27, 1).$	$occurs(ingr(\neg oatmealS), 27{:}28, 1).$
$occurs(ingr(\neg raisinsS), 27{:}28, 1).$	$occurs(ingr(\neg cabinet),28{:}29, 1).$
$occurs(ingr(cabinet),32{:}33, 1).$	$occurs(ingr(\neg cabinet),33{:}34, 1).$
$occurs(ingr(\neg waterA),43{:}44, 1).$	$occurs(ingr(cabinet), 49{:}50, 1).$
$occurs(ingr(sugarS), 50{:}51, 1).$	$occurs(ingr(medicineS), 50{:}51, 1).$
$occurs(ingr(\neg cabinet),52{:}53, 1).$	$occurs(ingr(burner),55{:}56, 1).$
$occurs(ingr(\neg burner),57{:}58, 1).$	$occurs(ingr(cabinet),67{:}68, 1).$
$occurs(ingr(\neg medicineS), 67{:}68, 1).$	$occurs(ingr(\neg cabinet),68{:}69, 1).$
$occurs(ingr(burner),68{:}69, 1).$	$occurs(ingr(\neg burner),88{:}89, 1).$

The result in Figure 15 shows that the system considers the whole process is done at time=70. However, the sensor caught the burner was on at time 72, and turned off by occupant at time=89, so this simulation uncover a problem in our representation of the world. Since at time=57 the food was ready, it means there was nothing to cook at time=72. In fact, the whole process by that portion of the dataset was finished after time 89, which again highlighted a mismatch.

6.2. *Revised Knowledge Base*

Thanks to our reasoning system we were able to uncover a problem and to reformulate the system. Our analysis guided by the tool led us to identify that at time 56, the burner was only open for 1 time slot, but in fact it was 9 second in the real dataset. By common sense, we know that a pot of water can not be boiled in 9 second. To better match the real data we added a new state to indicate more explicitly the burning time of water (*burningOver*30*Sec*), such that, we have the new rules base:

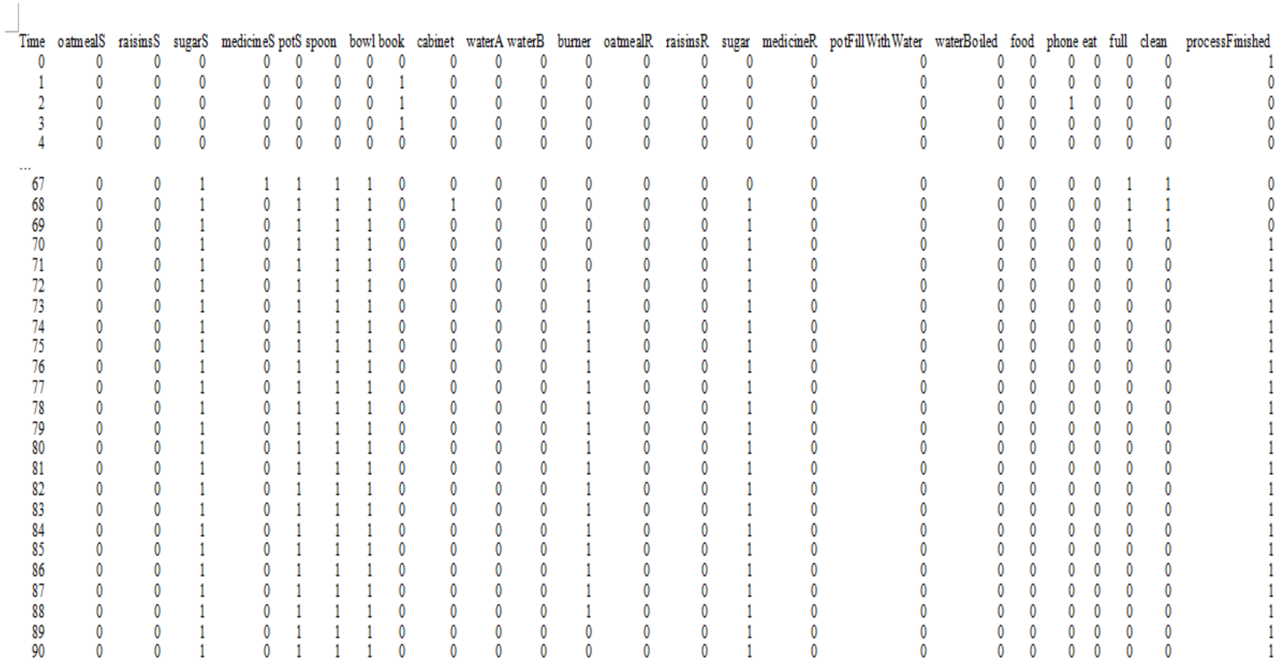
Same-Time Rules:

$(book \rightarrow \neg processFinished, 1)$
$(oatmealS \wedge cabinet \rightarrow oatmealR, 1)$
$(raisins \wedge cabinet \rightarrow raisinsR, 1)$
$(sugars \wedge cabinet \rightarrow sugarR, 1)$
$(medicines \wedge cabinet \rightarrow medicineR, 1)$
$(pots \wedge waterA \rightarrow potFillWithWater, 1)$
$(potFillWithWater \wedge burningOver30Sec \rightarrow waterBoiled, 1)$
$(food \rightarrow \neg oatmealR, 1)$
$(food \rightarrow \neg raisinsR, 1)$
$(food \rightarrow \neg sugarR, 1)$
$(food \rightarrow \neg potFillWithWater, 1)$
$(food \rightarrow \neg waterBoiled, 1)$
$(food \rightarrow \neg burningOver30Sec, 1)$
$(medicineR \wedge food \rightarrow eat, 1)$
$(processFinished \rightarrow \neg full, 1)$
$(processFinished \rightarrow \neg clean, 1)$

Next-Time Rules:

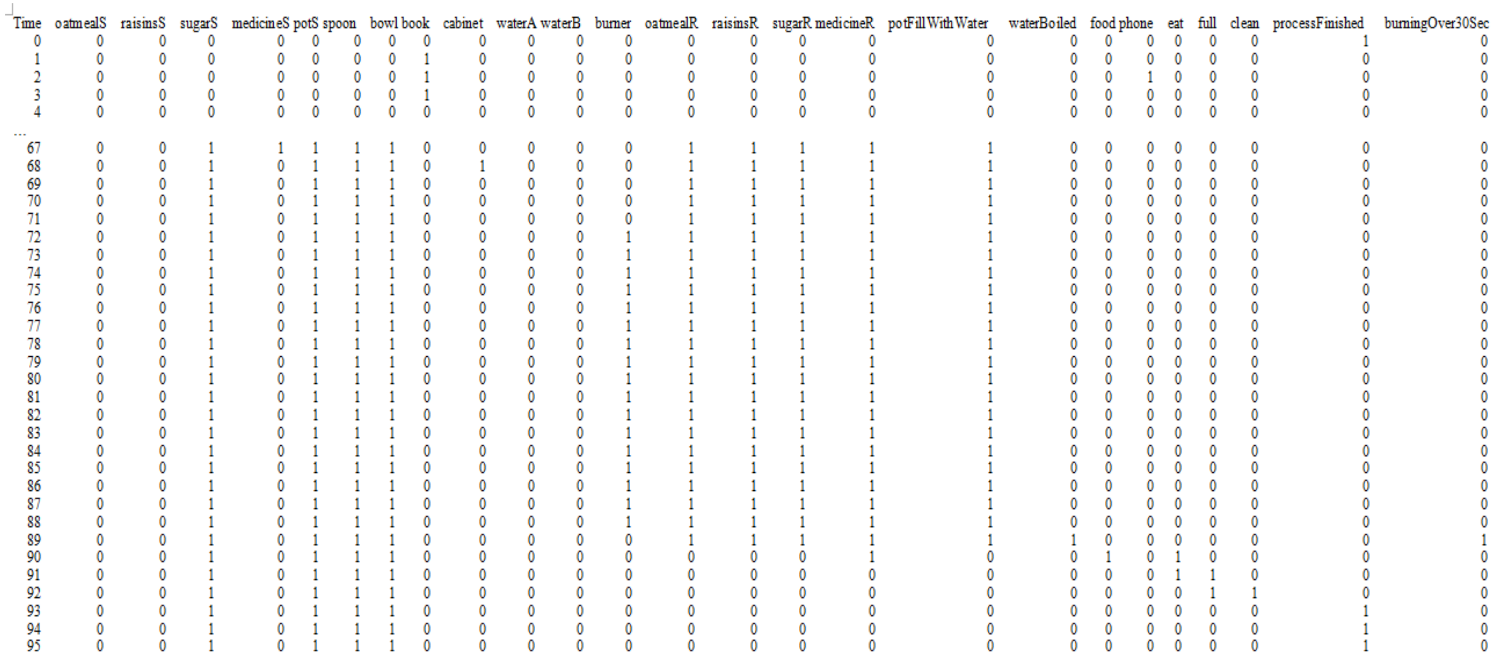
$(book \rightarrow \text{O}\, phone, 1)$
$(\neg phone \rightarrow \text{O} \neg book, 1)$
$(spoon \wedge bowl \wedge oatmealR \wedge raisinsR \wedge sugarR \wedge waterBoiled \rightarrow \text{O} food, 1)$ $(eat \rightarrow \text{O} \neg food, 1)$
$(eat \rightarrow \text{O} \neg medicineR, 1)$
$(eat \rightarrow \text{O} full, 1)$
$(full \rightarrow \text{O} \neg eat, 1)$
$(full \rightarrow \text{O}\, clean, 1)$
$(clean \wedge \neg cabinet \wedge \neg oatmealS \wedge \neg raisinsS \wedge \neg medicineS \wedge \neg waterA \wedge \neg waterB \wedge \neg burner \rightarrow \text{O}\, processFinished, 1)$

where *burningOver*30*Sec* is an independent state and ¬ *burningOver*30*Sec* is a dependent state. Moreover, there is a new event, *occurs*(*ingr*(*burningOver*30*Sec*),88:89, 1), added to the event list. The result of using the new knowledge base is shown in Figure 16, and this matched the real case, which completed the whole process at time 93.



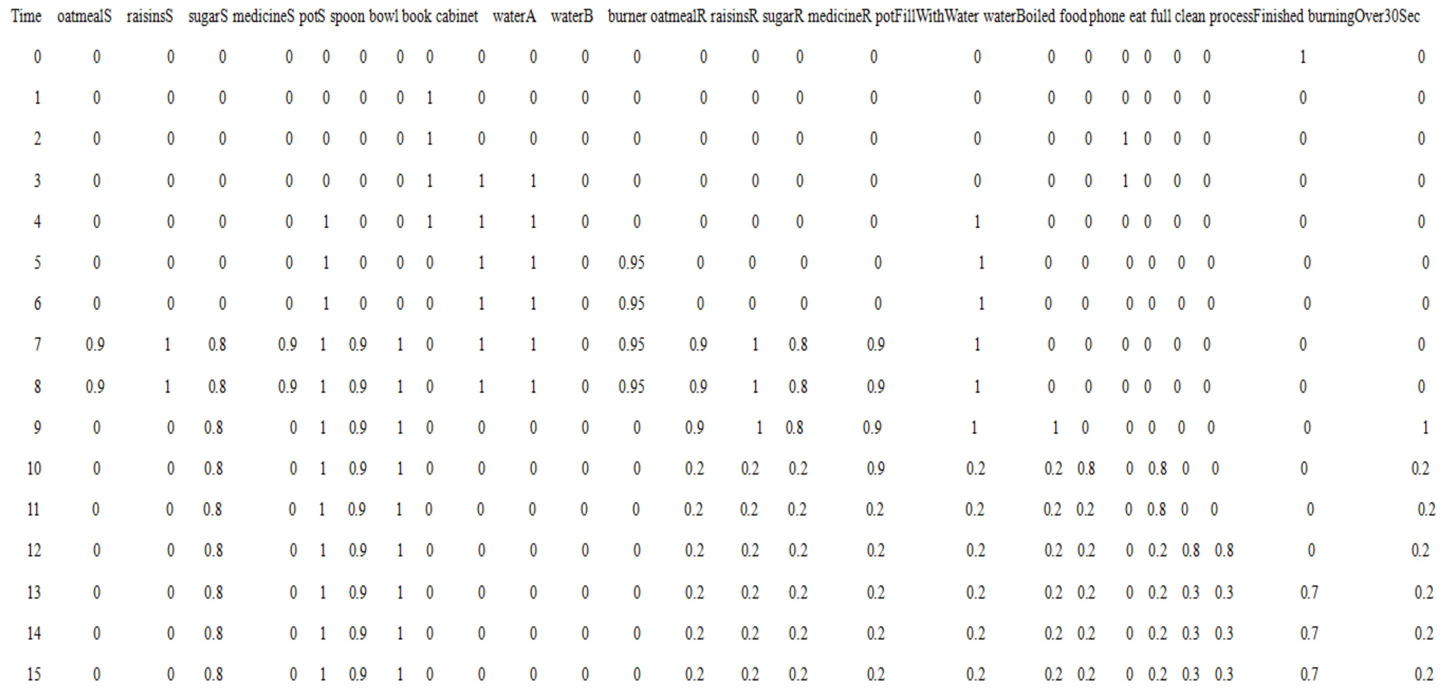
Time	oatmealS	raisinsS	sugarS	medicineS	potS	spoon	bowl	book	cabinet	waterA	waterB	burner	oatmealR	raisinsR	sugar	medicineR	potFillWithWater	waterBoiled	food	phone	eat	full	clean	processFinished
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
…																								
67	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
68	0	0	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
69	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
70	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
71	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
72	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
73	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
74	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
75	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
76	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
77	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
78	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
79	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
80	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
81	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
82	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
83	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
84	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
85	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
86	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
87	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
88	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
89	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
90	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Fig. 15. The result of running original knowledge base with real data in forward program



Time	oatmealS	raisinsS	sugarS	medicineS	potS	spoon	bowl	book	cabinet	waterA	waterB	burner	oatmealR	raisinsR	sugarR	medicineR	potFillWithWater	waterBoiled	food	phone	eat	full	clean	processFinished	burningOver30Sec
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...																									
67	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
68	0	0	1	0	1	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
69	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
70	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
71	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
72	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
73	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
74	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
75	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
76	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
77	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
78	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
79	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
80	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
81	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
82	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
83	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
84	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
85	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
86	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
87	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
88	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
89	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1
90	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0
91	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
92	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
93	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
94	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
95	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Fig. 16. The result of running improved knowledge base with real data in forward program



Time	oatmealS	raisinsS	sugarS	medicineS	potS	spoon	bowl	book	cabinet	waterA	waterB	burner	oatmealR	raisinsR	sugarR	medicineR	potFillWithWater	waterBoiled	food	phone	eat	full	clean	processFinished	burningOver30Sec
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	1	1	0	0.95	0	0	0	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1	1	0	0.95	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0.9	1	0.8	0.9	1	0.9	1	0	1	1	0	0.95	0.9	1	0.8	0.9	1	0	0	0	0	0	0	0	0
8	0.9	1	0.8	0.9	1	0.9	1	0	1	1	0	0.95	0.9	1	0.8	0.9	1	0	0	0	0	0	0	0	0
9	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.9	1	0.8	0.9	1	1	0	0	0	0	0	0	1
10	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.2	0.2	0.2	0.9	0.2	0.2	0.8	0	0.8	0	0	0	0.2
11	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0	0.8	0	0	0	0.2
12	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0	0.2	0.8	0.8	0	0.2
13	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0	0.2	0.3	0.3	0.7	0.2
14	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0	0.2	0.3	0.3	0.7	0.2
15	0	0	0.8	0	1	0.9	1	0	0	0	0	0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0	0.2	0.3	0.3	0.7	0.2

Fig. 17. The result of running improved knowledge base with artificial uncertain information data in forward program

6.3. *Artificial Data Test*

The real dataset given above only contains Boolean values, and now we want to test the improved knowledge base considering uncertain information as well. In this case study, we assume that V=[0, 1] and the meaning of value in [0, 1] is that the closer the value to 0 the closer to the concept of being 'false' and the closer to 1 the closer in meaning to being 'true'. We used forward and backward reasoning to investigate how these changes to the representation affected the outcomes and the decisions associated with the environment under investigation.

Initial Setting:

I_c={(*processFinished*, 1, 0), ($\neg$ *oatmealS*, 1, 0), ($\neg$ *raisinsS*, 1, 0), (*sugarS*, 1, 0), ($\neg$ *medicineS*, 1, 0), ($\neg$ *potS*, 1, 0), ($\neg$ *spoon*, 1, 0), ($\neg$ *bowl*, 1, 0), ($\neg$ *book*, 1, 0), ($\neg$ *cabinet*, 1, 0), ($\neg$ *waterA*, 1, 0), ($\neg$ *waterB*, 1, 0),($\neg$ *burner*, 1, 0),($\neg$ *oatmealR*, 1, 0), ($\neg$ *raisinsR*, 1, 0), ($\neg$ *sugarR,* 1, 0), ($\neg$ *medicineR*, 1, 0), ($\neg$ *potFillWithWater*, 1, 0), ($\neg$ *waterBoiled*, 1, 0), ($\neg$ *food*, 1, 0), ($\neg$ *phone*, 1, 0), ($\neg$ *eat*, 1, 0), ($\neg$ *full*, 1, 0), ($\neg$ *clean*, 1, 0), ($\neg$ *burningOver*30*Sec*, 1, 0)}

Events:

occurs(*ingr*(*book*), 0:1, 1).
occurs(*ingr*(*waterA*), 2:3, 1).
occurs(*ingr*(*potS*), 3:4, 1).
occurs(*ingr*(*raisinsS*), 6:7, 1).
occurs(*ingr*(*oatmealS*), 6:7, 0.9).
occurs(*ingr*(*medicineS*), 6:7, 0.9).
occurs(*ingr*($\neg$ *burner*), 8:9, 1).
occurs(*ingr*($\neg$ *oatmealS*), 8:9, 1).
occurs(*ingr*($\neg$ *medicineS*), 8:9, 1).
occurs(*ingr*($\neg$ *cabinet*), 8:9, 1).
occurs(*ingr*($\neg$ *phone*), 3:4, 1).
occurs(*ingr*(*cabinet*), 2:3, 1).
occurs(*ingr*(*burner*), 4:5, 0.95).
occurs(*ingr*(*spoon*), 6:7, 0.9).
occurs(*ingr*(*sugarS*), 6:7, 0.8).
occurs(*ingr*(*bowl*), 6:7, 1).
occurs(*ingr*(*burningOver30Sec*), 8:9, 1).
occurs(*ingr*($\neg$ *raisinsS*), 8:9, 1).
occurs(*ingr*($\neg$ *waterA*), 8:9, 1).

and let the rule:

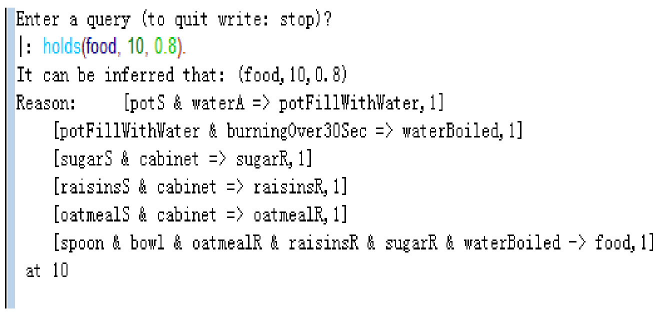
$$(clean \wedge \neg cabinet \wedge \neg oatmealS \wedge \neg raisinsS \wedge \neg medicineS \wedge \neg waterA \wedge \neg waterB \wedge \neg burner \rightarrow \mathrm{O}\, processFinished, 1),$$

changed into:

$$(clean \wedge \neg cabinet \wedge \neg oatmealS \wedge \neg raisinsS \wedge \neg medicineS \wedge \neg waterA \wedge \neg waterB \wedge \neg burner \rightarrow \mathrm{O}\, processFinished, 0.9),$$

The result from the system given in Figure 17 above shows that if the sensors are reasonably reliable then the simulation shows the system is still capable to draw the expected inferences with high level of confidence. After building up the improved knowledge base, users can use the query answering function from the reasoning system to trace the status of a particular state at specific time slot. Such that, while facing a large knowledge base containing huge amount of information, the system does not need to explore the whole knowledge base to search the answer of the query. For example, while using the knowledge base and event list provided above, we can enquiry the system whether food is ready at time 10 with high confidence (over 80%) then they can enter *holds*(*food*, 10, 0.8), and the system will provide the answer shown in that Figure.

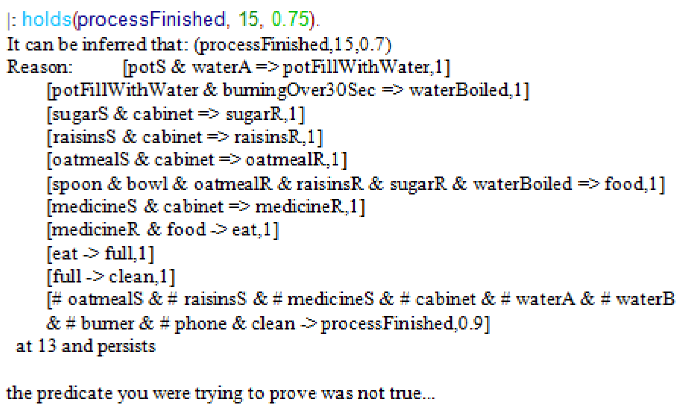
Figure 18 shows the query *holds*(*food*, 10, 0.8) is true, since the truth value of *food* is 0.8 at *time*=10 and also explains how it reached that conclusion. If users want to know if the process can be trusted with high confidence (more than 75% for example) to be finished at time 15, then they can enter a query *holds*(*processFinished*, 15, 0.75) and the system will provide the feedback shown in that Figure.



```
Enter a query (to quit write: stop)?
|: holds(food, 10, 0.8).
It can be inferred that: (food,10,0.8)
Reason:     [potS & waterA => potFillWithWater,1]
    [potFillWithWater & burningOver30Sec => waterBoiled,1]
    [sugarS & cabinet => sugarR,1]
    [raisinsS & cabinet => raisinsR,1]
    [oatmealS & cabinet => oatmealR,1]
    [spoon & bowl & oatmealR & raisinsR & sugarR & waterBoiled -> food,1]
 at 10
```

Fig. 18. The result of query *holds*(*food*, 10, 0.8)

The result shown in figure 19 inform us that the truth value of *processFinished* is only 0.7 at *time*=15, which means that the query *holds*(*processFinished*, 15, 0.75) is not successful and also provides details of how it reached that conclusion.



```
|: holds(processFinished, 15, 0.75).
It can be inferred that: (processFinished,15,0.7)
Reason:       [potS & waterA => potFillWithWater,1]
     [potFillWithWater & burningOver30Sec => waterBoiled,1]
     [sugarS & cabinet => sugarR,1]
     [raisinsS & cabinet => raisinsR,1]
     [oatmealS & cabinet => oatmealR,1]
     [spoon & bowl & oatmealR & raisinsR & sugarR & waterBoiled => food,1]
     [medicineS & cabinet => medicineR,1]
     [medicineR & food -> eat,1]
     [eat -> full,1]
     [full -> clean,1]
     [# oatmealS & # raisinsS & # medicineS & # cabinet & # waterA & # waterB
     & # burner & # phone & clean -> processFinished,0.9]
 at 13 and persists

the predicate you were trying to prove was not true...
```

Fig. 19. The result of query *holds*(*processFinished*, 15, 0.75)

Our experimentation with the tool showed that it provides a valuable contribution on examining the possible behaviours of the system being developed. Both Forward and Backward reasoning modalities offer complementary benefits. The explicit consideration of a temporal dimension (even if simple to favour efficiency), and uncertainty in the facts and the rules allowed us to consider essential features of the type of systems we are interested in, for example, intelligent environments as those we have considered along this article.

7. Conclusions

A multi-valued temporal propositional logic based reasoning system was presented in this paper and several aspects of its implementation and use were explained. The system includes two parts, forward and backward reasoning algorithms. The forward reasoning algorithm provides simulation/prediction function, and backward reasoning algorithm provides query answering function.

Section 3 illustrated the use of the program. It also provided explanations of the implementation of some key predicates for the methods used in forward and backward reasoning algorithms, including the concepts of "rule stratification", "supporting trees" and "activation time list".

Section 4 provided a scenario to show how to use the reasoning system to build up a knowledge base and improve it, moreover, how the system allows users to test their assumptions with artificial data in the knowledge base. The simulation/predication and query answering functions can help users to understand the final outcomes of particular decision-making problems given different assumptions. In this case, the reasoning system is able to help users to improve their knowledge bases or other settings of the real problems. Hence, users can have a clear and good picture of what may happen given different decision makings for a particular problem.

Overall we have obtained a system which allows representation and reasoning with uncertainty and time, important concepts in real world systems. These concepts were incorporated to a limited extent to balance expressiveness and efficiency whilst still ensuring important meta-theoretical properties were maintained. We understand these features are complex and can be still improved in a variety of ways. The many-valued logical system in MTPL was focused only on Lukasiewicz logic because of its distinct feature in terms of axiomatizablility, the use of other fuzzy logical systems replacing Lukasiewicz logical system could be explored in the future work to extend the utility of the reasoning algorithms. The current system does not allow a comfortable handling of intervals and only the propositional fragment has been considered at theoretical level. Our next steps will aim at building on the capabilities achieved in the current system. Still we would like to highlight the versatility of the system which can be used to reason over a variety of important scenarios. We have illustrated its use in this article mainly on safety and automation areas but it can be applied to a wide range of practical scenarios.

References

1. N. Rescher. *Many-Valued Logic*, McGraw Hill: New York. 1969.
2. L. Bolc, and P. Borowik. *Many-Valued Logics, 1. Theoretical Foundations*, Springer: Berlin, 1992.
3. J. Béziau. What is many-valued logic? *In Proc. 27th Int. Symposium on Multiple-Valued Logic*, (U.S.A., Los Alamitos, 1997), pp. 117–121.
4. S. Gottwald. Many-valued logic, in *Philosophy of Logic*, eds. Jacquette, D., Gabbay, D., Thagard, P. and Woods, J. Elsevier B.V.: U.K., 2006.
5. J. Pavelka. On fuzzy logic I: multiple-valued rules of inference, II: enriched residuated lattices and semantics of propositional calculi, III: semantical completeness of some multiple-valued propositional calculi. *Zeitschr. F. Math. Logik und Grundlagend. Math.*, vol. 25, pp. 45-52, 119-134, 447-464, 1979.
6. P. Hajek. *Metamathematics of Fuzzy Logic*, Kluwer: Dordrecht. 1998.
7. V. Novak, I. Perfilieva and J. Mockor. *Mathematical Principles of Fuzzy Logic*, Kluwer: Dordrecht. 2000.
8. G. Gerla. *Fuzzy logic: Mathematical Tools for Approximate*, Kluwer Academic Publishers: Boston, 2001.
9. Y. Xu, D. Ruan, K. Qin and J. Liu. *Lattice-valued Logic: An Alternative Approach to Treat Fuzziness and Incomparability*, Springer-Verlag: Heidelberg, New York. 2003.
10. J. Allen. Maintaining knowledge about temporal intervals. *CACM* **26**(11):832-843, 1983.
11. J. Allen, and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation, Special Issue on Actions and Processes*, **4:**531-579, 1994.
12. J. Augusto. The logical approach to temporal reasoning, *Artificial Intelligence Review* **16**(4):301-333, 2001.
13. J. Augusto. A general framework for reasoning about change. *New Generation Computing* **21**(3):209-247, 2003.
14. V. Rybakov. Logical consecutions in discrete linear temporal logic. *Journal of Symbolic Logic* **70**(4):1137–1149, 2005.
15. E. Aboelela, and C. Douligeris. Fuzzy temporal reasoning and model for event correlation in network management. *24th Annual IEEE International Conference on Local Computer Networks* (LCN'99), 18-20th October 1999, Lowell, USA, pp. 150-159.
16. S. Schockaert and M. Cock. Temporal reasoning about fuzzy intervals. *Artificial Intelligence* **172**:1158-1193, 2008.
17. M. Mucientes, R. Iglesias, C. Reguerio, A. Bugarin and S. Barro. A fuzzy temporal rule-based velocity controller for mobile robotics. *Fuzzy Sets and Systems* **134**:83-99, 2003.
18. G. Escalada-Imaz. A temporal many-valued logic for real time control systems. In *Proceedings of the 9th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, eds. Cerri, S. and Dochev, D., 20th-23rd September 2000, Varna, Bulgaria, pp. 91-100.
19. K. Lamine and F. Kabanza. Using fuzzy temporal logic for monitoring behaviour-based mobile robots. *In Proceeding of the IASTED International Conference on Robotics and Applications*, eds. Hamza, M., 14th-16th August 2001, Honolulu, USA, pp. 116-121.
20. Z. Zhang, Y. Sui, C. Cao and G. Wu. A formal fuzzy reasoning system and reasoning mechanism based on propositional modal logic. *Theoretical Computer Science* **368**:149-160. 2006.

21. A. Galton and J. Augusto. Stratified causal theories for reasoning about deterministic devices and protocols. *In TIME'02: Proc. 9th Int. Symposium on Temporal Representation and Reasoning* (Manchester, UK, 7-9 July, 2002), pp. 52-54.
22. A. Galton. Causal Reasoning for Alert Generation in Smart Homes, *Designing Smart Homes*, eds. Augusto, J. and Nugent, C., Springer-Verlag: Germany, pp 57-70, 2006.
23. J. Liu, L. Lopez Martinez, Y. Xu and Z. Lu. Automated reasoning algorithm for linguistic valued Łukasiewicz propositional logic. *In Proceedings of the 37th International Symposium on Multiple-Valued Logic*, 13th-16th May 2007, Oslo, Norway, pp. 29-34.
24. Z. Lu, J. Liu, J. Augusto, and H. Wang. Multi-valued temporal reasoning framework for decision making, *Proceedings of the 8th International FLINS Conference on Computational Intelligence in Decision and Control*, Madrid, Spain, 21 - 24 September 2008, in "*World Scientific Proceedings Series on Computer Engineering and Information Science - Vol. 1: Computational Intelligence in Decision and Control*", pp. 301-306, D. Ruan, J. Montero *et al (Eds.)*, World Scientific, 2008.
25. Z. Lu, J. Liu, J. Augusto, and H. Wang. A many-valued temporal logic and reasoning framework for decision making. Book chapter in "*Computational Intelligence in Complex Decision Systems*" (Edited by Ruan, D.), Atlantis Press/World Scientific, Paris/Singapore, March, 2010, pp. 127-148. 2010.
26. Z. Lu, J. Augusto, J. Liu, and H. Wang. A linguistic truth-valued temporal reasoning system (LTR) and its application to the design of an intelligent environment. To appear in *International Journal of Computational Intelligence Systems*. Atlantis Press. 2010.
27. M. Bouzid, and A. Mouaddib. Uncertain temporal reasoning for the distributed transportation scheduling problem. *In Proceeding of the Fifth International Workshop on Temporal Representation and Reasoning*, 16th–17th May 1998, Sanibel Island, USA, pp. 21-28.
28. L. Godo and L.Vila. Possibilistic temporal reasoning based on fuzzy temporal constraints. *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, 20th-25th August 1995, Montreal, Canada, pp. 1916-1922.
29. S. Raha and K. Ray. Approximate reasoning with time. *Fuzzy Sets and Systems* **107**:59-79, 1999.
30. M. Campos, J. Palma, and R. Marin. A model for fuzzy temporal reasoning on a database. *In Proceeding of 10th Conference of the Spanish Association for Artificial Intelligence CAEPIA 2003 and 5th Conference on Technology Transfer TTIA 2003,* eds. Conejo, R., Urretavizcaya, M. and Perez-de-la-Cruz, J., 12th-14th November 2003, San Sebastian, Spain, pp. 57-65.
31. W. Long. Temporal reasoning for diagnosis in a causal probabilistic knowledge base. *Artificial Intelligence in Medicine* **8**:193-215. 1995.
32. S. Ribaric, B. Basic and L. Males. An approach to validation of fuzzy qualitative temporal reasoning. *Journal of Computing and Information Technology* **3**:163-170, 2002.
33. V. Ryabov, S. Puuronen, and T. Terziyan. Representation and reasoning with uncertain temporal relations. *In Proceedings of Twelfth International Florida AI Research Society Conference*, eds. Kumar, A. and Russell I., 1st-5th May 1999, Orlando, USA, pp. 449-453.
34. V. Ryabov and A. Trudel. Probabilistic temporal interval networks. *In Proceeding of the 11th International Symposium on Temporal Representation and Reasoning (TIME'04)*, 1st-3rd July 2004, Tatihou, France, pp. 64-67.
35. S. Schockaert. Reasoning about fuzzy temporal and spatial information from the web. *Ph.D. thesis*, Ghent University. 2008.

36. A. Tawfik and E. Neufeld. Temporal Bayesian networks. *In Proceedings of the TIME-94 International Workshop on Temporal Representation and Reasoning*, eds. Goodwin, S. and Hamilton, H., 4th May 1994, Pensacola USA, pp. 85-92.
37. M. Puterman. *Markov Decision Processes*. Wiley: USA. 1994.
38. M. Viedma, M. Morales and I. Sanchez. Fuzzy temporal constraint logic: a valid resolution principle. *Fuzzy Sets and Systems* **117**:231-250. 2001.
39. G. Palshikar. A fuzzy temporal notation and its application to specify fault patterns for diagnosis. *Pattern Recognition Letters* **22**:381-394, 2001.
40. T. Mora and E. Sanchez. Fuzzy logic-based real-time navigation controller for a mobile robot. *In Proceeding of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 13th-17th October 1998, Victoria, B.C., Canada, pp. 612-617.
41. M. Mucientes, R. Iglesias, C. Reguerio, A. Bugarin and S. Barro. A fuzzy temporal rule-based velocity controller for mobile robotics. *Fuzzy Sets and Systems* **134**:83-99, 2003.
42. M. Mucientes and A. Bugarin. People detection through quantified fuzzy temporal rules. *Pattern Recognition* **43**(4):1441-1453. 2010.
43. Z. Pei, D. Ruan, J. Liu and Y. Xu. *Linguistic Values Based Intelligent Information Processing: Theory, Methods, and Applications*, Atlantis Press/World Scientific: Paris/Singapore, 2009.
44. J. Augusto, and P. McCullagh. Ambient intelligence: concepts and applications. *Invited Paper by the International Journal on Computer Science and Information Systems* **4**(1):1-28, 2007.
45. J. Augusto, P. McCullagh, V. Croft and J. Walkden. Enhanced healthcare provision through assisted decision-making in a smart home environment. *In Proc. 2nd Workshop on Artificial Intelligence Techniques for Ambient Intelligence (AITAmI07)* (India, Hyderabad, 2007), pp. 27-32.
46. P. Rashidi, D.J. Cook, L.B. Holder, and M. Schmitter-Edgecombe. Discovering Activities to Recognize and Track in a Smart Environment. IEEE Transactions on Knowledge and Data Engineering **23**(4):527-539, April 2011.