Temporal and Procedural Reasoning with Probabilistic Logic Networks for Agent Control

Nil Geisweiller and Hedra Yusuf

SingularityNET Foundation, The Netherlands
{nil,hedra}@singularitynet.io

Abstract. TODO

Keywords: Temporal · Procedural · Reasoning · Probabilistic Logic Networks · OpenCog

1 Introduction

The goal of this project is to make an agent as rational as possible, not necessarily as efficient as possible. This stems from the concern that in order to autonomously gain efficiency the agent must first be able to make the best possible decisions, starting first in the outer world, and then in the inner world.

The paper presents

The agent starts in a completely unknown environment

The idea is that reasoning is used at all levels, discovering patterns from raw observations, building plans and making decisions.

It is a work in progress.

Neural networks are excellent at interpolation, but are rather poor at extrapolation, what we need for true intelligence is a system that thinks critically.

Rarely do causes and effects take place over arbitrary temporal scales. For instance it is unlikely to find a cause that may produce the same effect, or an effect at all, after 1ms, 1 century or any time in between. For that reason we focus on a real time temporal logic.

2 Related Work

Event Calculus. Temporal Logic (Next operator). Subjective logic and evidence-based subjective logic.

3 Contributions

The contributions of that paper are:

1. Build upon existing temporal reasoning framework defined in Chap.14 [TODO: cite PLN book].
2. Design an architecture for controlling an agent based on that temporal reasoning extension.

4 Outline

1. Temporal reasoning
2. ROCCA
3. Minecraft experiment

5 Recall: Probabilistic Logic Networks

PLN, which stands for Probabilistic Logic Networks, is a mixture of predicate and term logic that has been probabilitized to properly handle uncertainty. It has two types of rules

1. one type for introducing relationships from direct observations,
2. the other for introducing relationships from existing relationships.

As such it is especially suited for building an ongoing understanding of an unknown environment (using direct introduction rules), and then planning in that environment (using indirect introduction rules).

5.1 Elementary Notions

Graphically speaking, PLN statements are sub-hypergraphs[1] made of links and nodes, called *Atoms*, decorated with *Truth Values* that can be understood as uncertain probabilities [**?**]. Syntactically speaking however, PLN statements are not very different from statements expressed in another logic, except that they are usually formatted in prefixed-operator indented-argument style to emphasize their graphical nature and leave room for truth values. There is a large variety of constructs for PLN, here we will focus primarily on constructs for manipulating predicates. Let us recall that predicates are functions that take tuples of Atoms and output boolean values

$$P, Q, R, ... : Atom^n \mapsto \{True, False\}$$

Within predicate constructs there are two classes of operators

1. one for defining predicate from instances, such as *Evaluation* and *Lambda*,
2. and another for combining existing predicates, such as *And*, *Not* and *Implication*.

Let us present these operators below, corresponding to the minimum subset we will need in the rest of the paper.

– Evaluation:

$$\begin{array}{c} Evaluation\ \langle TV \rangle \\ P \\ E \end{array}$$

states that $P(E)$ outputs *True* to a degree set by the truth value TV.

[1] because links can point to links, not just nodes

– Lambda:

$$\begin{array}{c} \mathit{Lambda}\ \langle TV \rangle \\ x \\ P(x) \end{array}$$

is a predicate constructor with variable x and predicate body $P(x)$, where the true value TV corresponds to the probability $\mathbf{Pr}(P)$ of $P(x)$ to output *True* for a random input.

– Conjunction:

$$\begin{array}{c} \mathit{And}\ \langle TV \rangle \\ P \\ Q \end{array}$$

represents the predicate obtained by taking the conjunction of P and Q, or equivalently the indicator function corresponding to the intersection of the *satisfying sets* of P and Q. The truth value TV then represents an estimate of the probability $\mathbf{Pr}(P, Q)$ of the conjunction of P and Q.

– Negation:

$$\begin{array}{c} \mathit{Not}\ \langle TV \rangle \\ P \end{array}$$

represents the negation of P, or equivalently the indicator function corresponding to the complement of the satisfying set of P. The truth value TV then represents an estimate of the probability $\mathbf{Pr}(\neg P)$ of the negation of P.

– Implication:

$$\begin{array}{c} \mathit{Implication}\ \langle TV \rangle \\ P \\ Q \end{array}$$

represents the predicate Q conditioned on P, that is only defined for instances x for which $P(x)$ is *True*. The truth value TV then represents an estimate of the conditional probability $\mathbf{Pr}(Q|P)$. There is some subtleties to take into account due to the fact $P(x)$ can actually be partially true (stated by the truth values of *Evaluation* links as explained above), but this resolves nicely by assuming degrees of truth are probabilistic. More is explained about that below.

Truth values are fundamentally second order probability distributions. However in practice they are usually represented by two numbers, a strength and a confidence, both ranging from 0 to 1. The strength represents a probability while the confidence represents a precision over that probability. Underneath, strength and confidence can be mapped into a second order distribution such as a Beta distribution [TODO: add figure]. TODO: cite Subjective Logic and Chapt 4 of the PLN book.

5.2 Inference Rules

Beside operators, inferences rules are used to construct PLN statements and calculate their truth values. They mainly fall into two categories, direct and indirect. Direct rules infer abstract knowledge from direct evidence, while indirect

rules infer knowledge by combining existing abstractions, themselves inferred directly or indirectly. There are dozens of inference rules but for now we will only recall two which are needed for the paper:

1. *Implication Direct Introduction Rule*
2. *Deduction Rule*

The Implication Direct Introduction Rule (IDI) takes *Evaluation* links as premises and produces an *Implication* link as conclusion, formally depicted by the following proof tree

$$\frac{\begin{array}{ccc} \mathit{Evaluation}\ \langle TV_i^P \rangle & & \mathit{Evaluation}\ \langle TV_i^Q \rangle \\ P & \cdots & Q \\ E_i & & E_i \end{array}}{\begin{array}{c} \mathit{Implication}\ \langle TV \rangle \\ P \\ Q \end{array}}\ \text{(IDI)}$$

Assuming perfectly reliable direct evidence[2] then the resulting truth value is calculated as follows

$$TV.s = \frac{\sum_{i=1}^{n} f_\wedge(TV_i^P.s, TV_i^Q.s)}{\sum_{i=1}^{n} TV_i^P.s}$$

$$TV.c = \frac{n}{n+k}$$

where $TV.s$ and $TV.c$ respectively represent the strength and the confidence of TV, k is a system parameter, and $f_\wedge$ is a function embodying a probabilistic assumption about the intersection of the events corresponding to the *probabilitized degrees of truth* of $P(E_i)$ and $Q(E_i)$. Such function typically ranges from the product (perfect independence) to the min (perfect overlap).

The Deduction Rule (D) takes two *Implication* links as premises and produces a third one. Depending on what assumption is made there exists different variations of that rule. The simplest one, based on the Markov property

$$\mathbf{Pr}(R|Q,P) = \mathbf{Pr}(R|Q)$$

can be formally depicted by the following proof tree

$$\frac{\begin{array}{cccccc} \mathit{Implication}\ \langle TV^{PQ} \rangle & \mathit{Implication}\ \langle TV^{QR} \rangle & & & & \\ P & Q & P\langle TV^P \rangle & Q\langle TV^Q \rangle & R\langle TV^R \rangle & \\ Q & R & & & & \end{array}}{\begin{array}{c} \mathit{Implication}\ \langle TV \rangle \\ P \\ R \end{array}}\ \text{(D)}$$

[2] Dealing with unreliable direct evidence involves expensive convolution products and is outside of the scope of this paper.

The reader may notice that three additional premises have been added, corresponding to the probabilities $\mathbf{Pr}(P)$, $\mathbf{Pr}(Q)$ and $\mathbf{Pr}(R)$. This is a consequence of the Markov property. The exact formula for that variation will not be recalled[3] here but it merely derives from

$$\mathbf{Pr}(R|P) = \mathbf{Pr}(R|Q,P) \times \mathbf{Pr}(Q|P) + \mathbf{Pr}(R|\neg Q,P) \times \mathbf{Pr}(\neg Q|P)$$

6 Temporal Logic

The temporal logic define in [**?**] is somewhat partial and ambiguous. In that section we provide a possible completion. Let us begin by defining *Temporal Predicates*, also called *Fluents* as predicates tends to denote atemporal relationships, as regular predicates with a temporal dimension

$$P, Q, R, ... : Atom^n \times Time \mapsto \{True, False\}$$

Time here is considered discrete, formally defined as a natural number.

6.1 Temporal Operators

Given temporal predicates we can now define a small set of temporal operators.

Lag* and *Lead are temporal operators to shift the time dimension of a temporal predicate. *Lag*, respectively *Lead*, is similar to the metric variation of the *Past* operator denoted P_n, respectively the *Future* operator denoted F_n, of Temporal Logic [**?**, **?**], with the distinction that it applies over an expression that evaluates into a temporal predicate, as opposed to an expression that evaluates into boolean.

The *Lag* operator is formally defined as follows

$$\begin{array}{l} Lag \\ \quad P \\ \quad T \end{array}$$

:=

$$\begin{array}{l} Lambda \\ \quad x_1, ..., x_n, t \\ \quad P(x_1, ..., x_n, t - T) \end{array}$$

where the first line of the *Lambda* link is the variable declaration of the temporal predicate and the second line is the body representing a temporally shifted reconstruction of P. Informally speaking, the *Lag* operator gives a peek into the past, or equivalently, brings the past into the present. This is a major difference with [**?**] that defined sequential and using notions of the Event Calculus [**?**]. Here we do not do that (TODO: maybe such operator should be called PrecedeAnd,

[3] More information can be found in [**?**]

or such). That is because using this simpler notion of sequential and allows more flexibility to construct various forms of descriptions of how events initialization, termination, etc, are sequenced.

The *Lead* operator is the inverse of the *Lag* operator and is formally defined as follows

$$\begin{array}{l} \mathit{Lead} \\ \quad P \\ \quad T \end{array} :=$$

$$\begin{array}{l} \mathit{Lambda} \\ \quad x_1, ..., x_n, t \\ \quad P(x_1, ..., x_n, t + T) \end{array}$$

As the inverse of the *Lag* operator, the *Lead* operator gives a peek into the future, or equivalently, brings the future into the present. Finally, as being the inverse of one another, the following equivalence holds

$$\begin{array}{l} \mathit{Lag} \\ \quad \mathit{Lead} \\ \quad\quad P \\ \quad\quad T \\ \quad T \end{array} \equiv P$$

SequentialAnd is a temporal conjunction where one of the temporal predicate arguments have been temporally shifted. There are two variations one can define. A *BackSequentialAnd* variation where the past of one of the temporal predicates is brought into the present, using the *Lag* operator, formally defined as

$$\begin{array}{l} \mathit{BackSequentialAnd} \\ \quad T \\ \quad P \\ \quad Q \end{array} := \begin{array}{l} \mathit{And} \\ \quad \mathit{Lag} \\ \quad\quad P \\ \quad\quad T \\ \quad Q \end{array}$$

resulting into a temporal predicate such that in order to be true at time t requires that P be true at time t and Q be true at time $t + T$.

Inversely, there is a *ForeSequentialAnd* variation where the future of the other temporal predicate is brought into the present, formally defined as

$$\begin{array}{l} \mathit{ForeSequentialAnd} \\ \quad T \\ \quad P \\ \quad Q \end{array} := \begin{array}{l} \mathit{And} \\ \quad P \\ \quad \mathit{Lead} \\ \quad\quad Q \\ \quad\quad T \end{array}$$

resulting into a temporal predicate such that in order to be true at time t requires that P be true at time $t - T$ and Q be true at time t.

Finally one may notice that *BackSequentialAnd* and *ForeSequentialAnd* equivalent up to temporal shifting, as follows

NEXT

$$\begin{array}{l} \mathit{BackSequentialAnd} \\ \quad T \\ \quad P \\ \quad Q \end{array} \equiv \begin{array}{l} \mathit{BackSequentialAnd} \\ \quad T \\ \quad P \\ \quad Q \end{array}$$

PredictiveImplication likewise defines a temporal implication, formally

$$\begin{array}{l} \mathit{PredictiveImplication} \\ \quad T \\ \quad P \\ \quad Q \end{array}$$

:=

$$\begin{array}{l} \mathit{Implication} \\ \quad P \\ \quad \mathit{Lead} \\ \quad\quad Q \\ \quad\quad T \end{array}$$

resulting into a conditional predicate, that in order to be defined at time t requires that P be true at time t, and in order to be true at t requires that Q be true at $t + T$.

We now have everything we need to define temporal inference rules, but before that let us first introduce some notations in order to be easier to lay out.

6.2 Notations

The following notations can afford to ignore truth values, that is because no new formula is required for temporal reasoning. All that is required are the definitions above mapping temporal expressions into equivalent atemporal ones. The notations are summarized in the table below, ranked by syntactic precedence

to minimize the number of required parenthesis.

Atomese	Notation	Precedence
$Evaluation(P, List(X_1, \dots, X_n))$	$P(X_1, \dots, X_n)$	1
$Lambda(t, AtTime(Execution(A), t))$	$\widehat{A}$	1
$Lag(P, T)$	$\vec{P}^T$	1
$Lead(P, T)$	$\overleftarrow{P}^T$	1
$And(P, Q)$	$P \wedge Q$	2
$Or(P, Q)$	$P \vee Q$	2
$SequentialAnd(T, P, Q)$	$P \barwedge^T Q$	3
$SequentialOr(T, P, Q)$	$P \barvee^T Q$	3
$Implication(P, Q)$	$P \to Q$	4
$PredictiveImplication(T, P, Q)$	$P \rightsquigarrow^T Q$	4

The precedence of everything else (predicates nodes, etc) is 0.

For instance

TODO: show examples of notational format with the effect of precedence

6.3 Temporal Rules

TODO: *PredictiveImplication* direct introduction.

Given that we can now introduce our temporal rules, (PI), (IP), (S)

TODO: detail (PI), (IP), (S) rules

and the most important one Temporal Deduction (TD)

$$\frac{P \rightsquigarrow^{T_1} Q \qquad Q \rightsquigarrow^{T_2} R \qquad P \qquad Q \qquad R}{P \rightsquigarrow^{T_1+T_2} R} \text{ (TD)}$$

To determine the formula to calculate the resulting truth value of such rule, we only need to map such temporal deduction into a regular deduction as follows

$$\dfrac{\dfrac{\dfrac{P \rightsquigarrow^{T_1} Q}{P \to \tilde{Q}^{T_1}}\text{ (PI)} \qquad \dfrac{\dfrac{Q \rightsquigarrow^{T_2} R}{Q \to \tilde{R}^{T_2}}\text{ (PI)}}{\tilde{Q}^{T_1} \to \tilde{R}^{T_1+T_2}}\text{ (S)} \qquad P \qquad \dfrac{Q}{\tilde{Q}^{T_1}}\text{ (S)} \qquad \dfrac{R}{\tilde{R}^{T_1+T_2}}\text{ (S)}}{P \to \tilde{R}^{T_1+T_2}}\text{ (D)}}{P \rightsquigarrow^{T_1+T_2} R}\text{ (IP)}$$

6.4 Procedural Reasoning

Likewise, we can use the same temporal to regular deduction mapping to build inference rules for procedural reasoning. Given two cognitive schematics

$$P \wedge \widehat{A} \rightsquigarrow^{T_1} Q$$

expressing that executing A in context P likely leads to context Q after T_1 time units, and

$$Q \wedge \widehat{B} \rightsquigarrow^{T_2} R$$

expressing that executing B in context Q likely leads to context R after T_2 time units, the question becomes how to infer the likelihood that executing A and B in sequence starting from context P leads to context R after $T_1 + T_2$ time units, corresponding to the cognitive schematic

$$P \wedge \widehat{A} \nearrow^{T_1} \widehat{B} \rightsquigarrow^{T_1+T_2} R$$

Using the same rules to map *PredictiveImplication* to regular *Implication* and vice versa, as well as rules about the *Lead* operator, we can construct the following inference tree, in fact the PLN engine can construct it for us

$$\dfrac{\dfrac{\dfrac{\dfrac{P \wedge \hat{A} \rightsquigarrow^{T_1} Q}{P \wedge \hat{A} \rightarrow^{T_1} \bar{Q}^{T_1}}\,(\mathrm{PI})}{P \wedge \hat{A} \wedge \bar{\hat{B}}^{T_1} \rightarrow \bar{Q}^{T_1} \wedge \bar{\hat{B}}^{T_1}}\,(\mathrm{C}) \qquad \dfrac{\dfrac{Q \wedge \hat{B} \rightsquigarrow^{T_2} R}{Q \wedge \hat{B} \rightarrow \bar{R}^{T_2}}\,(\mathrm{PI})}{\bar{Q}^{T_1} \wedge \bar{\hat{B}}^{T_1} \rightarrow \bar{R}^{T_1+T_2}}\,(\mathrm{S}) \qquad P \wedge \hat{A} \wedge \bar{\hat{B}}^{T_1} \qquad \dfrac{Q \wedge \hat{B}}{\bar{Q}^{T_1} \wedge \bar{\hat{B}}^{T_1}}\,(\mathrm{S}) \qquad \dfrac{R}{\bar{R}^{T_1+T_2}}\,(\mathrm{S})}{P \wedge \hat{A} \wedge \bar{\hat{B}}^{T_1} \rightarrow \bar{R}^{T_1+T_2}}\,(\mathrm{D})}{P \wedge \hat{A} \nearrow^{T_1} \hat{B} \rightsquigarrow^{T_1+T_2} R}\,(\mathrm{IP})$$

which reduces to the following inference tree after retaining only the premises and the conclusion

$$\dfrac{P \wedge \hat{A} \rightsquigarrow^{T_1} Q \qquad Q \wedge \hat{B} \rightsquigarrow^{T_2} R \qquad P \wedge \hat{A} \wedge \bar{\hat{B}}^{T_1} \qquad Q \wedge \hat{B} \qquad R}{P \wedge \hat{A} \nearrow^{T_1} \hat{B} \rightsquigarrow^{T_1+T_2} R}\,(\mathrm{ASD})$$

providing an Action Sequence Deduction (ASD) rule ready to be used for procedural reasoning. Three additional premises have been added $P \wedge \widehat{A} \wedge \overleftarrow{\widehat{B}}^{T_1}$, $Q \wedge \widehat{B}$ and R.

7 Rational OpenCog Controlled Agent

To experiment with temporal and procedural reasoning in the context of embodied virtual agents in unknown environments we have implemented a project called ROCCA, which stands for *Rational OpenCog Controlled Agent*. ROCCA essentially acts as an interface between virtual environments such as Malmo [**?**] or OpenAI Gym [**?**] and OpenCog. It provides an Observation-Planning-Action control loop as well as various launchers to run OpenCog processes such as PLN reasoning, pattern mining, etc. Provided a top goal, such as maximizing a reward, ROCCA orchestrates the necessary learning and the planning to fulfill that goal. One may possibly see ROCCA as a reinforcement learning agent with the particularity that learning and planning are, at least in principle, entire done via reasoning. In that respect it is similar in spirit to OpenNARS for Applications (ONA) [**?**] but uses PLN as its core reasoning logic rather than NAL [**?**].

ROCCA is composed of two main processes, one for real-time agent control and another one for non-reactive background learning. In principle these two processes could happen in parallel, though as of right now they occur as distinct alternating phases.

7.1 Control Phase

The control phase is composed of control cycles, each decomposed into Observation, Planning and Acting steps, more precisely

1. Observation step:
   (a) receives and timestamps observations from the environment,
   (b) stores the timestamped observations in the atomspace.
2. Planning step:
   (a) selects the goal for that iteration,
   (b) finds plans fulfilling that goal,
   (c) given these plans, deduces a probabilistic distribution of actions,
   (d) selects the next action according to the deduced probabilistic distribution.
3. Acting step:
   (a) timestamps and stores in the atomspace the selected action,
   (b) runs the selected action and by that updates the environment,
   (c) receives the reward from the environment,
   (d) timestamps and stores the reward in the atomspace.

None of these steps are difficult to carry with the exception of deducing a probabilistic distribution of actions. For that we use a variation of Solomonoff induction described in [**?**] which is especially suited for plans described by conditional second order distributions, in other words *PredictiveImplication* links. More specifically plans are *PredictiveImplication* links of the form

$$C \wedge A \rightsquigarrow^{T} G$$

called *Cognitive Schematics*. Which can be read as *"in some context C, if some action (elementary or composite) A is executed, then after T time units, the goal G is likely to be fulfilled"*. The degree of expected fulfillment is specified by the truth value of the *PredictiveImplication* link, not indicated in that notational format but present in the extended Atomese format. The difficulty then comes down to discovering cognitive schematics that are as informative and applicable as possible.

7.2 Learning Phase

As hinted above, the ultimate goal of the learning phase is to discover maximally useful cognitive schematics, and by useful it is specifically meant that they are as predictive and cover as many cases as possible.

TODO: pattern mining and reasoning.

8 Experiment with Simple Minecraft Environment

In this experiment we built a minecraft environment using Malmo, which is a platform for Artificial Intelligence experimentation and research built on top of Minecraft. The demo environment consists of a small house with a locked door, diamonds inside and a key to get into the house. The agent, initially located outside of the house, can perform different actions like getting a key, opening a door of the house and collecting the diamonds in order to achieve a reward.

The aim of this experiment is to make the ROCCA agent learn from the actions and perceptions in the minecraft environment and do efficient planning so as to be able to collect as many diamonds as possible and accumulate reward. The ROCCA agent will be able to perform a series of possible actions with a goal of achieving a reward and learns from them by applying PLN (Probabilistic Logic Networks) and Pattern Miner, which can be seen as a specialized form of PLN reasoning. The Planning, the discovery of cognitive schematics, is also handled by PLN and its temporal reasoning rule base.

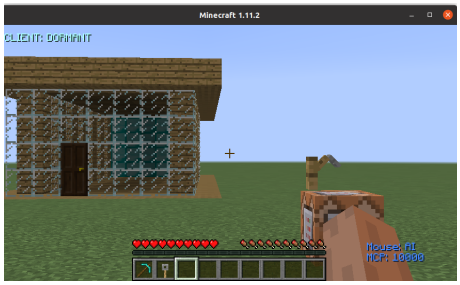

Fig. 1. Simple Minecraft demo with a house and a key.

There are lists of allowed actions provided by minecraft that an agent can perform like moving, turning, picking etc.. but due to the limited processing capacity we have to handle the observations from each action and to reduce complexity, we proposed to have an abstract action and perception where unnecessary details have been omitted. With that we generate three abstract actions namely go-to-key, go-to-house, go-to-diamonds where each of them contains a series of actions and returns an abstract perception about where the agent is (inside house, outside house, next to closed door etc..), about its inventory (has key, diamond pickaxe etc..) and the reward of completing a given action.

We perform various experiments tuning different parameters. A typical experiment has two iterations of the learning-training process with a duration of fifty iterations for each training. In the first cycle the agent will not have prior knowledge hence no learning will take place. The agent pursues the environment and builds its knowledge base by trying a combination of fifty randomly weighted actions. At the end of the first cycle the agent will have enough background knowledge to apply Pattern miner and PLN temporal reasoning. Hence,

during the second cycle, the agent will be able to learn and plan the desired cognitive schematics which leads to a positive goal of getting a reward.
The ROCCA agent is able to learn the following cognitive schematics with higher strength.

$$outside(self, house) \wedge \widehat{go\_to(key)} \rightsquigarrow^{1} hold(self, key)$$

$$hold(self, key) \wedge \widehat{go\_to(house)} \rightsquigarrow^{1} inside(self, house)$$

$$inside(self, house) \wedge \widehat{go\_to(diamond)} \rightsquigarrow^{1} reward(1)$$

In this experiment, we measure the agent's performance by the cognitive schematics learned and accumulated rewards achieved. The ROCAA agent is successful in learning the required cognitive schematics which leads the agent to collect more rewards in the second cycle. However, these findings with a simple minecraft environment with only few actions might not tell the overall performance of ROCCA. As a future work, further extensive experiments are needed to conclude the performance achieved.

9 Conclusion

TODO: implement more temporal and procedural rules, support temporal intervals, behavior trees, introduce Temporal Truth Value. Integrate Event Calculus.

References