Towards a Complete Formalization of PLN
(DRAFT)

Nil Geisweiller

June 13, 2025

1 Introduction

The goal is similar to Solomonoff Universal Induction [**?**], that is we want to approach a first order (unknown but computable) distribution μ given observations, using a second order (uncomputable but known) distribution ν, called the Universal Distribution. In Solomonoff Induction, observations are bit strings produced by a Turing machine[1]. In PLN however, observations are outcomes from an indexed boolean random variable, representing the outputs of evaluating a predicate on some inputs. Such predicate is called the *observable predicate*. In practice PLN allows multiple observable predicates however one can assume one predicate without loss of generality. Indeed, to emulate multiple predicates, one can introduce an extra component in the predicate's domain to "select" the predicate of interest. Also, since it is observed by a random variable, such predicate is not necessarily deterministic (though it could be). As such, one may think of the observable predicate as being a program drawn from a certain Probabilistic Programming Language. In the following section we formally define the above.

Because this reformulation of PLN departs somewhat from the definition of PLN in the PLN book [**?**], we give it a new name, νPLN.

2 Definitions

In its original form, PLN purposely avoids relying on an underlying global probability distribution. I am not against this in principle. I will simply admit that I cannot conceive a complete definition of PLN that does not rely on such global probability distribution. I would also point out that a publication released after the PLN book by the principal authors of the PLN book, Ben Goertzel and Matt Ikle, very much aligns with the idea of a global probability distribution [**?**], and was in fact a great source of inspiration for writing this very document. The

[1]Note that even though the sample space of ν is made of deterministic Turing machines, ν can approximate any computable distribution μ (thus non-deterministic) by maintaining an ensemble of such Turing machines.

next subsection is dedicated to define the global probability distribution which νPLN is intended to derive from.

2.1 Global Probability Distribution

Given an observable predicate μ with domain $\mathcal{D}$, let $(\Omega, \mathcal{F}, \nu)$ be a probability space such that

- $\mathcal{F}$ is the event space, a σ-algebra on Ω.
- $\nu : \mathcal{F} \to [0, 1]$ is a universal distribution, further defined below.
- Ω is the sample space associated to ν, such that each element $\omega \in \Omega$ contains a model μ[2], a probabilistic predicate over a certain domain $\mathcal{D}$, as well as the complete mapping of μ from $\mathcal{D}$ to its Boolean co-domain.

Since μ is a probabilistic predicate, its type signature cannot merely be

$$\mu : \mathcal{D} \to \text{Bool}$$

where $\text{Bool} = \{\bot, \top\}$. To capture its probabilistic nature we give it the following type signature

$$\mu : \mathcal{D} \to \Omega \to \text{Bool}$$

in a curried fashion. Meaning that given its argument, it produces a boolean random variable. Therefore the observable predicate can be viewed as an indexed boolean random variable. Of course, μ never gets to be evaluated on a different world than the one it belongs to, but we still need to keep the Ω argument in order to reason about possible worlds since the true underlying world is unknown. Also, in cases where the domain $\mathcal{D}$ can be decomposed into multiple components, a curried notation will be preferred. So instead of

$$\mu : (\mathcal{D}_1 \times \cdots \times \mathcal{D}_n) \to \Omega \to \text{Bool}$$

the following

$$\mu : \mathcal{D}_1 \to ... \to \mathcal{D}_n \to \Omega \to \text{Bool}$$

will be used. This will be convenient to express inheritance relationships between partially applied predicates. Applying μ to an input x will be denoted with the following functional programming notation

$$(\mu\ x)$$

Likewise, μ applied to more than two arguments will be denoted

$$(\mu\ x\ \omega)$$

[2]Those familiar with Solomonoff Universal Induction, please note that here μ represents a probabilistic predicate rather than a computable probability function calculating the probability of any event, although the latter can be derived from the former.

and so on. For μ such functional programming notation is used because it is currying-friendly. For the rest, we keep using the traditional mathematical function application notation, such as

$$\nu(E)$$

denoting the application of the probability distribution ν to the event E. In case μ is known to be deterministic, Ω could potentially be dropped, but that is not going to be our working assumption for now. With that, let us now define key random variables to access Ω:

- $M : \Omega \to \mathcal{L}$ with measurable space $(\mathcal{L}, \mathcal{F}_\mathcal{L})$, where $\mathcal{L}$ is a certain probabilistic programming language and $\mathcal{F}_\mathcal{L}$ is a σ-algebra on $\mathcal{L}$. Thus, M takes a particular world $\omega \in \Omega$ and outputs the probabilistic program $\mu \in \mathcal{L}$ generating that world. Note that this random variable is inaccessible from an observer within that world. An observer within that world only has access to a finite record of evaluations of μ. However, that random variable will be convenient to define aspects of the semantics of νPLN.

- $D_{x \in \mathcal{D}} : \Omega \to$ Bool, a Boolean random variable indexed by values in $\mathcal{D}$. Unlike M, $D_{x \in \mathcal{D}}$ is at least partially accessible from an observer within that world. Meaning, such observer can gather data for a finite subset $\mathcal{S}$ of $\mathcal{D}$. In that case $D_\mathcal{S}$ represents a finite family of Boolean random variables, corresponding the set of accessible observations. M and $D_{x \in \mathcal{D}}$ are related by the following equality

  $$(\mu\ x\ \omega) = (D_x\ \omega)$$

  where ω runs over all elements of Ω such that $(M\ \omega) = \mu$. Or simply, in curried fashion

  $$(\mu\ x) = D_x$$

Due to the equality above, the distribution of observations is entirely determined by a model μ. In other words, it suffices to define a distribution over $\mathcal{L}$, the prior distribution over possible models, to define ν (as far as M and $D_{x \in \mathcal{D}}$ are concerned anyway). Then, relating observations to models can be done using regular Bayesian inference. The prior is defined by

$$\nu(M \in L)$$

where $L \in \mathcal{F}_\mathcal{L}$. Note how it is expressed in terms of elements of $\mathcal{F}_\mathcal{L}$, instead of elements of $\mathcal{L}$. It is because, for the purpose of recovering PLN with the Bayesian approach put forward, $\mathcal{L}$ needs to be continuous. I will explain that aspect in detail, but for now let us use this notation to formulate Bayes' theorem

$$\nu(M \in L | D_\mathcal{S}) = \frac{\nu(M \in L) \times \nu(D_\mathcal{S} | M \in L)}{\nu(D_\mathcal{S})}$$

where $\mathcal{S}$ is a finite subset of $\mathcal{D}$. An example of prior will be given in **??**, but in general it can be viewed as a parameter of νPLN. That is, given a certain prior of ν over $\mathcal{L}$, one can derive a certain flavor of νPLN.

3 Concepts vs Predicates

The PLN book describes respectively the notions of *concepts* and *predicates*, and their respective associated relationships *inheritance* and *implication*. There is a perfect isomorphism between concepts and inheritance on one side and predicates and implication on the other side. Concerned with conciseness, we will pick a side, the predicate side, and essentially forget about the other side, the concept side, for the rest of this document. But before we do so let us recall what is a concept, the inheritance between two concepts, and the isomorphism between concepts and predicates.

3.1 Concepts and Inheritance

A concept is a fuzzy (or, as I prefer to say, probabilistic, for reasons I will explain in Section **??**) set, and the *extensional* inheritance between two concepts is a probabilitized subset relationship. Originally, in the PLN book, the inheritance relationship is defined as an explicit mixture of extensional and intensional inheritances. We will show however that they are in fact both the same thing, the extensional inheritance is a way to approach inheritance solely via *induction*, and intensional inheritance is a way to approach inheritance solely via *abduction*. Any one side, extensional or intensional, is good enough to define the other, so let us pick one, the extensional side, and define inheritance with it. The extensional inheritance between two concepts can be viewed as a probabilitized subset relationship. It allows to express things like "most members of a set are also members of another set". For instance one could express in PLN that 90% of birds fly, with

$$(\texttt{Inheritance Bird Fly}) \overset{\text{m}}{=} 0.9$$

Such knowledge might have been obtained by observing a finite sample of birds and whether or not they fly. Meaning there could be an uncertainty on the 90% itself. To represent such uncertainty PLN uses a second order distribution, in this case a Beta distribution as it is an ideal choice to represent the posterior of the parameter of a Bernouilli distribution given observations. Under this assumption only two numbers are required to determine the parameters, α and β, of the associated Beta distribution. A *truth value* called *simple truth value* was created for this purpose and is thus described by two numbers: a strength (a proxy for a probability estimate) and a confidence over this strength from which the α and β parameters of the Beta distribution can be recovered. For instance, given a simple truth value one may express that 90% of birds fly with a confidence of 0.99

$$(\texttt{Inheritance Bird Fly}) \overset{\text{m}}{=} <0.9, 0.99>$$

where 0.9 is the strength and 0.99 is the confidence. The confidence is a value between 0 and 1 that actually encodes the sample size that was used to obtained the strength. The higher the sample size, the higher the confidence. More

information about that will be provided in Section **??** but for now let us just leave it at that as it is enough for what we are concerned with in this Section which is the isomorphism between concepts and predicates.

3.2 Isomorphism between Concepts and Predicates

To every concept one can associate a predicate and vice versa. To go from concept to predicate one can use the *indicator function*, and to go from predicate to concept one can use the *satisfying set*. These notions are well known for crisp predicates and sets and thus will not be detailed any further here. The only difference in PLN is that concepts are probabilistic, meaning that a probability (or potentially a second order probability) can be attached to the membership of an element to a concept. Likewise, predicates are probabilistic in the sense that a (second order) probability can be attached to the evaluation of an argument to a predicate. As one would expect the isomorphism also applies between the inheritance relationship on the concept side and the implication relationship on the predicate side. So for instance, on the predicate side one can express the same inheritance relationship between birds and fly as follows

$$\texttt{(Implication IsBird DoesFly)} \stackrel{m}{=} <0.9, 0.99>$$

where `IsBird` and `DoesFly` have been obtained by taking the indicator functions of `Bird` and `Fly`. As mentioned earlier, we will drop the concept side and only focus on the predicate side for the remaining of the document. If the reader still wishes to understand more clearly this isomorphism the PLN book covers it well.

4 Deriving PLN Rules

Our goal here is to derive every PLN rules in the PLN book from the global distribution that has been defined in Section 2.1. By doing so we hope not only to provide a clear unambiguous definition for each rule, but also an ideal to approach as using a global distribution should give us the means to derive a convergence theorem a la Solomonoff.

4.1 Implication Direct Introduction

This rule is not explicitly stated as such in the PLN book but it can be derived from iteratively applying induction and revision, and directly reflects the formula of extensional inheritance/implication

5 conclusion

References