Rule Chaining and Parsing

Linas Vepstas

July 2015

Abstract

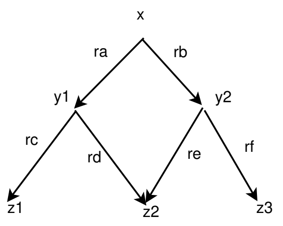
Some notes about reduction, states, rule engines, backward and forward chaining, parsing and Markov networks. These are all inter-related topics, but not obviously so, thus these notes. In particular, a functional probabilistic chainer is of central importance for OpenCog, so a clear exposition is needed if forward proress is to be made.

The notes here are informal and just a rough draft, based on points of confusion that have occured on the OpenCog mailing list.

Intro

The goal of this set of notes is to talk about the overall design of the OpenCog forward and backwards chainers, the URE (universal rule engine) and to highlight some unresolved issues.

Consider the problem of forward-chaining, as it is commonly envisioned. Its shown in the graph below:


$$\begin{tikzcd}
 & & x \arrow[dl, "ra"'] \arrow[dr, "rb"] & & \\
 & y1 \arrow[dl, "rc"'] \arrow[dr, "rd"] & & y2 \arrow[dl, "re"'] \arrow[dr, "rf"] & \\
z1 & & z2 & & z3
\end{tikzcd}$$

Here, the initial contents of the atomspace is represented by 'x'; after applying rule 'ra', the contents of the atomspace is represented by 'y1', and so on. Applying rule 'rb' to the initial state 'x' results in a *different* atomspace 'y2', *i.e.* an atomspace with different contents.

The differences can be mutually exclusive. Consider this example: The position 'x' could correspond to the statements "*he has the ball.*", together with the statements "*Natnael is a male.*" and "*Amen is a male.*" Based on this, the rule 'ra' could be an anaphora resolution rule that takes, as pre-requeists, the "*he*" refers to males, and suggests as output, that "*he*" is "*Natnael*". Similarly, 'rb' is the same rule, but applied to different data, and thus suggests that "*he*" is "*Amen*". Clearly, both statements cannot be true at the same time. Thus, in effect, the “universe” splits into two: in one universe, “*he*” is “*Natnael”*, and in the other “*he*” is “*Amen*”. It is also useful to conclude that in universe ’y1’, the association “*he*” is “*Natnael*” has a truth-value of 100% true and 90% confident. Thus, the truth of the statements in y1 and y2 are mutually exclusive. At this point, though, we are not sure which “universe” is correct. Thus, it is reasonable to assign each universe a probability of 50%.

There is a set of unresolved issues in the above example. These are:

- What is the right way of representing the state in ’y1’ and ’y2’ in the atomspace, simulataneously?
- What is the right way for assigning truth values to ’y1’ and ’y2’?

There has been the general suggestion that ContextLinks can be used to solve the first issue, but the details have not been specified, and there is no general agreed-upon format, yet.

There is a third issue shown in the diagram: a confluent diamond. That is, after applying rule ’ra’ and then ’rd’, one ends up with the same “facts” ’z2’ as one would by applying ’rb’ then ’re’. (For example, this could happen if ’ra’=’re’ aand ’rb’=’rd’, and ’ra’ and ’rb’ commute.) How can one “know” that both cases end with the same ’z2’? As a practical matter, how does one identify, in the code, that one ended up in the same spot? As another important related issue: what should the probability of “universe” ’z2’ be? Should it be the sum of the probabilities of taking paths (ra,rd) and (rb,re)? Should it be something else? (Does PLN have something to say about this?)

Markov Nets

In the above example, we start with the state ’x’ and finish with multiple mutually-exclusive “universes” ’z1’, ’z2’ and ’z3’. Ideally, we want to assign probabilities to each of these universes, in such a way that only one is likely, and the others are not. That is, in ’z1’, we have that “*he*” is “*Natnael*” holds true, but we also had additional evidence, via rule ’rc’ that perhaps “*he*” is actually “*Amen*”, for example, if a sentence arrived that said “*and then Amen threw the ball*”. In this cae, we want to lower the probability of universe ’z1’ to a very low value, while raising the probability of the universe ’z3’ to a very high value (since ’z3’ is the one where Amen has the ball).

This implies that the distribution of probabilites between these “aternate universes” must be Markovian. If the intial probability of the univers ’x’ is 100%, then the sume of the probabilities on ’y1’ and ’y2’ must also be 100%, and likewise, the sume of ’z1’, ’z2’ and ’z3’ must sum to 100%. (I am abusing the term ’Markovian’ somewhat; the precise definition is different, but for the current purposes, its OK.)

Parsing, Learning.

I don’t think we are ready, yet, with the current state of the URE code in OpenCog, to deal correctly with the learning of rules, but it is appropriate to start a conversation.

So, in the above example, we assumed some a-priori probabilities: for the first pass, of who has the ball, a 50-50 split was assumed. The later rules, 'rc', 'rd' etc. were also assumed to impart fixed a-priori probabilities.

But what if, for some reason, we don't have the a-priori probabilities for these 'state-transition' rules? If, instead, we know that we start in state 'x' and we know with 100% probability that the real world finishes in state 'z1', then there several algorithms for estimating the transition probabilites in moving down the graph. One of these is the forward-backward algorithm. This tends to be cpu-time-consuming, so there is another, simplified but sometimes less accurate one, the Viterbi algorithm.

One reason that the a-priori probabilities are not known is because we are attempting to learn the state-transition rules from scratch. Thus, for example, we could generate some random rule 'rx', stick it in the list of rules that the URE is applying, and watch it generate garbage as a result. Does this garbage actually correspond to observed reality, or is it indeed, junk lacking predictive power? This can, in principle, be answered by applying forward-backward, or Viterbi, to estimate the transition probability of rule 'rx'. If this probability is small or zero, then 'rx' is a useless or false rule. If it is high, then 'rx' is a valuable 'rule of thumb' tha can be applied during deduction; it should be kept and added to the rule-set.

Details

Much of what is talked about here should go onto wiki pages. In particular, details and notation for a lot of this needs to be worked out. Right now, it was eaaier to create a PDF.

References

See the following:

- Baader & Nipkow, "*Term Rewriting and All That*". Cambridge 1998 – Provides a good introduction to abstract rewriting systems, and the concepts of reduction, unification, and some very basic proof theory.