A Description Logic System for Learning in Complex Domains*

Jordi Alvarez
Universitat Politècnica de Catalunya
Departament de Sistemes i Llenguatges Informàtics
jalvarez@lsi.upc.es

Abstract

This paper introduces YAYA[1], a Description Logic system focused towards learning complex interrelations among objects. YAYA Concept Language (YCL) is quite restricted; although it introduces a limited use of variables.

Language limited expressiveness in combination with a special inference mechanism allow YAYA to achieve its main goal: to acquire a model (TBox) from a set of examples that allows it to complete the information of new sets of (incomplete) examples from the same domain.

1 Introduction

Knowledge structuring and reasoning services provided by Description Logics can be very useful for Machine Learning tasks. Nevertheless, little attention has been paid to defining Concept Languages suitable for acquiring the model we will use for reasoning purposes. C-CLASSIC [Ventos, 1996] is an example. [Cohen and Hirsh, 1994] study learnability properties of CoreClassic, proposing an efficiently learnable subset of it.

In this paper we introduce a new Description Logic system called YAYA that has been designed to solve complex inference problems by learning from examples. Using Description Logics, the problem gets reduced to the acquisition or completion of a TBox from a set of examples (the training ABox). The final TBox should make YAYA able to infer as much information as possible in order to complete other incomplete ABoxes from the same domain.

For example, if the problem to be solved is to obtain the syntactic analysis for a set of sentences, the training ABox would contain a set of sentences (sets of words related by the *next* and *previous* relations) with their corresponding analysis (sets of grammatical objects related through syntactic roles). On the other hand, incomplete ABoxes to be completed will contain only the text part.

YAYA Concept Language (YCL) is quite restricted. Its syntax tries to represent complex interactions among objects in an easy way. YCL has a special construct that introduces variables. Although a general use of variables could increase the complexity of the reasoning algorithms (see [Borgida, 1996]), YCL syntax restricts the use of variables to situations in which the complexity is never increased. Regardless language restrictions, YAYA is able to capture complex object interrelations and makes possible to solve problems in complex domains.

YAYA provides a learning algorithm that performs a local search over the space of possible YCL terms and is the responsible for acquiring/completing the TBox. Information completion is performed by YAYA Inference Engine (YIE), that uses the acquired TBox to incrementally perform inferences and represent them *explicitly* on the ABox.

2 YAYA Concept Language

Several decisions have influenced the design of YAYA Concept Language. First, we take the epistemological position that *important knowledge is always and only at concept level.* No instances will appear in a YCL definition. Secondly, we can learn only about knowledge that we can represent at the instance level (nodes and relations among them).

And finally, we want our language to remain tractable in reasoning and learning tasks. Learnability has not been studied yet; but, as [Cohen and Hirsh, 1994] states, having deductive reasoning tractability raises the possibility that learning can also be done efficiently. Regarding deductive reasoning tractability, subsumption for a DL language with disjunction or with the combination of universal and existential quantification is intractable

[1] YAYA means Yet Another Yet Another. The name is after YACC (Yet Another Compiler Compiler) and other less famous YAxx.

* This work is partially supported by Catalan Government (CIRIT-1997SGR-51).

[Donini *et al.*, 1996].

Intuitively, we choose to represent directly what we see in a semantic network: *chains of nodes* and conjunctions of them (corresponds to the following DL constructors: *concept name*, *conjunction* and *existential quantification*). We do not perform any abstraction over them. This would result in the introduction of new DL constructors (negation, quantified universal ...) that could lead YCL to lose tractability properties). Cyclic definitions are not allowed. On the other hand, disjunction is not necessary for our purposes in the definition of concepts; since a disjunctive definition can be *simulated* (for our inference purposes) with a set of concepts corresponding to the disjunction subterms.

YCL without variables is a bit less expressive than $\mathcal{FLE}^-$. So, $\mathcal{FLE}^-$ has been taken as reference when explaining YCL constructs in the following paragraphs. The constructors are as follows:

:strc Allows us to define basic structural properties; that is, an ordered set of nodes linked each one to the next. It is equivalent to a chain of *quantified existentials* linked with conjunctions in $\mathcal{FLE}^-$. For example: $(:\ strc\ \ human\ \xrightarrow{married-to}\ human\ \xrightarrow{has-parent}\ human)$ corresponds to the $\mathcal{FLE}^-$ term $human\ \sqcap\ \exists(married-to.(human\ \sqcap\ \exists(has-parent.human)))$.

:and New structural properties can be defined as the conjunction of simpler structural properties. This constructor corresponds to the conjunction $\mathcal{FLE}^-$ constructor ($\sqcap$).

:unify The previous constructs permit us to express a subset of what [Woods, 1991] has called *type I descriptions*. But this does not allow to express coordination among nodes.

The *:unify* particle permits us to state that a node appearing in a structural property should be the same one that appears in another place of the structural property. This corresponds to a generalization of the *path reasoning* idea introduced by SNePS [Shapiro, 1991] and of the SAME-AS CLASSIC construct [Brachman *et al.*, 1991]. Moreover, the introduction of variables has been pointed out in [Borgida, 1996]. The *:unify* particle can be abbreviated by using *:* between the concept name and the variable name.

Using the *:unify* particle we could try (see next section) to express facts like: *the father of my father is my grandfather*, through the following term[2]:

$$(:\ and\quad (:\ strc\ \ human\ \xrightarrow{father}\ human\ \xrightarrow{father}\ human{:}x)\ \ (:\ strc\ \ human\ \xrightarrow{grandfa.}\ human{:}x)\ )$$

[2] In fact, the term is only an approximation to the real statement we want to represent (as explained later).

YCL TERM	INTERPRETATION
C	$C^{\mathcal{I}}$
$(:\ strc\ \ C_1\ R_1\ C_2\ \ R_2\ C_3\ \ldots)$	$\{\delta \mid C_1^{\mathcal{I}} \wedge \exists x.R_1^{\mathcal{I}}(\delta, x) \wedge C_2^{\mathcal{I}}(x) \wedge \exists y.R_3^{\mathcal{I}}(x, y) \wedge C_3^{\mathcal{I}}(y) \ldots\}$
$(:\ and\ \ C_1\ C_2)$	$C_1^{\mathcal{I}} \sqcap C_2^{\mathcal{I}}$

Table 1: YCL semantics without variables.

SITUATION	YCL TERM	INTERPRETATION
no var	$(:\ strc\ \ldots R_{i-1}\ C_i \ldots)$	$\{\delta \mid \ldots \exists x.R_{i-1}^{\mathcal{I}}(y, x) \wedge C_i^{\mathcal{I}}(x) \ldots\}$
first v	$(:\ strc\ \ldots\ R_{i-1}\ (:\ unify\ \ C_i\ v)\ \ldots)$	$\{\delta \mid \ldots \exists z_v.R_{i-1}^{\mathcal{I}}(y, z_v) \wedge C_i^{\mathcal{I}}(z_v) \ldots\}$
other v's	$(:\ strc\ \ldots\ R_{i-1}\ (:\ unify\ \ C_i\ v)\ \ldots)$	$\{\delta \mid \ldots \exists x.R_{i-1}^{\mathcal{I}}(y, x) \wedge C_i^{\mathcal{I}}(x) \wedge z_v = x \ldots\}$

Table 2: YCL term interpretation with variables.

YCL semantics is given denotationally, using the notion of *interpretation* $\mathcal{I} = <\Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}}>$. $\mathcal{I}$ has a domain of values $\Delta^{\mathcal{I}}$ and a mapping $(\cdot)^{\mathcal{I}}$. This approach has been largely used in Description Logics literature (for a wider explanation see [Donini *et al.*, 1996]).

In table 2 we can see semantic interpretation of YCL constructors without variables. The interpretation of YCL terms with variables implies technical difficulties because a variable can appear in any subterm and the interpretation of $(:\ unify\ \ C\ x)$ depends on whether it is the first time that the variable x appears or not. Fortunately, variables will appear always inside a *:strc* construction. This allows us to provide a general way to deal with variables in term interpretations. Table 2 shows this. For both tables, variables x and y can be used alternatively without loss of generality [Borgida, 1996].

Is YCL expressive enough?

In fact, as we know that the *:strc* construct is equivalent to a *chain* of quantified existentials, the YCL term in the *:unify* construct does not state that *the father of my father is my grandfather* (supposing "I" has been inferred to be an instance of the concept that corresponds to that definition). Instead, we are stating: *I have a father that has a father that is one of my grandfathers*. This is not a precise statement of the concept we wish to represent.

The point is that inferring that an ABox object holds a YCL term is not enough to infer links from that object to other objects in the ABox, as we would like. For example, in the syntactical analysis problem, we would like to infer the creation of the nodes and links corresponding to the analysis. The problem is that there are several ways an ABox object can be modified in order to hold a YCL term (so, we do not know which one is the correct one).

In order to solve this problem, YAYA provides an explicit inference engine (YIE) that chooses one of the different ways in which an object description can be modified to hold a YCL term. The criteria used by YIE tries to minimize the amount of information added to the TBox. We intuit that this criteria works quite well for most domains (we have tested it empirically for the domains used in the experiments in section 6).

So, YIE allows us to maintain YCL this simple. This is quite useful to maintain the complexity of the learning process as low as possible.

3 Incidental Properties

In addition to YCL capabilities, YAYA stores a *summary* of information for every concept called *concept description*. It is made up of a set of YCL terms each one of them has an associated probability. The probability corresponds to the percentage of known concept instances that hold the YCL term. The set of YCL terms stored in the summary corresponds to what [Brachman *et al.*, 1991] calls incidental properties.

In this way, the combination of concept definitions and concept descriptions provides a set of conditional probabilities. For any defined concept, we will have a pair constituted by its definition and its description $< def, \{< p_1, \alpha_1 >, < p_2, \alpha_2 >, \ldots, < p_n, \alpha_n >\}$. This provides n probabilistic rules to YAYA that can be summarized by the following equation:

$$Prob(p_i(x)|def(x)) \;=\; \alpha_i \tag{1}$$

Probabilistic rules can be converted to non-probabilistic rules if a threshold θ_r is introduced; and then, only rules with probability above it are taken into account[3].

YCL terms taking part in concept descriptions constitute only a small subset of YCL terms considered by YAYA. The learning process decides which YCL terms are set as important properties (see next section).

The combination of concept definitions and descriptions is useful to:

- Define a set of heuristics to guide the learning process based on statistical information about concept instances. Concept descriptions can be seen as a generalization of COBWEB probabilistic concept representation [Fisher, 1987].
- Serve as a set of probabilistic (or non-probabilitic) rules that will form the basic knowledge used by YIE (this is similar to PDL inclusion coefficients [Bonifati *et al.*, 1997]).

[3]Rules with probability under $1 - \theta_r$ can be seen as $Prob(\neg p_i(x)|def(x)) \;=\; 1 - \alpha_i$. So, they can be used to remove inconsistencies just as the "positive" rules are used to remove incompleteness. This point has not been studied yet.

4 Learning

The learning algorithm performs a local search over the space of possible YCL terms. The goal of the search is to find as many interesting correlations among YCL terms as possible. These correlations are stored in the TBox as the conditional probabilities stored in concept descriptions. The information used to establish them is the Abox constituting the training set (the stored probabilities are an estimation of the real ones). So, only positive examples are used by YAYA.

The local search is guided by a set of operators and heuristics. Operators are triggered when an event occurs in the learning environment. For example when a new concept is introduced in the TBox, an important property is taken into account, or equally when a primitive TBox concept is set as a valid node to take part in YCL terms[4]. These operators generate new YCL terms that are heuristically evaluated. Depending on the evaluation, the terms are defined as TBox concepts and/or introduced as important properties (or none of them).

The YAYA learning process is closely related to ILP techniques such as FOIL [Quinlan, 1990]. Similar to FOIL, the local search is a generate and test iterative method. On the contrary, no negative examples are used by YAYA. In addition, YAYA builds (or completes) the overall model (TBox) incrementally; instead of building one rule at a time (as FOIL does).

The local search performed by YAYA is neither a generalization nor a specialization. There are around 15 different operators. Some of them modify syntactically YCL terms in different ways in order to build new ones. Others modify the learning environment (adding new valid nodes, for example). Most of them use information in the training ABox to decide which syntactic modifications can be useful.

Heuristic Measures

YAYA uses two kind of heuristics. The first one tries to restrict the generated YCL terms in a syntactical way (restricting the set of nodes taking part in a YCL term, for example). The second type tries to value the inference power added to the learned model by performing an action (defining a concept, adding an important property or making a concept a valid node). These heuristics use statistical information taken from the description associated to the corresponding YCL term[5].

[4]YAYA restricts the set of nodes that can appear in a valid YCL term. An initial set of valid nodes can be set manually. From that point on, the learning process itself introduces new valid concepts in certain situations when it discovers useful YCL terms that contain a non-valid node.

[5]Descriptions can be computed for any YCL term; as all we need is a set of objects (in this case, those holding the YCL term)

The Extension Hierarchy

Concepts and important properties should be discovered in order to provide a good knowledge model. Sometimes the gap between existing knowledge and knowledge to be discovered is too big to be covered by the local search.

In order to solve this problem, another hierarchy of concepts is built by YAYA. This new hierarchy is based on the conditional probabilities introduced in section 3. It is called the extension hierarchy (EH) because it is computed from information taken from concept instances (concept descriptions); opposed to the subsumption hierarchy, which is an intensional one.

In order to build the EH, a threshold θ_{EH} is defined; and when a concept is created, and its description contains a stored conditional probability such that: $Prob(p(x)|def(x)) \geq \theta_{EH}$; then:

1. Two concepts are created in the EH, one corresponding to p and the other corresponding to def (if they have been already created, nothing happens).
2. The concept corresponding to def is set as a subconcept of the one corresponding to p (as $Prob(p(x)|def(x)) \geq \theta_{EH}$).

   Both EH concepts can be directly or indirectly related. Even so, both YCL terms can be represented by the same EH concept. All this depends on other conditional probabilities relevant to them.

We have observed empirically for artificially generated domains that EH is very similar to the original hierarchy used to generate the training set. For the NL experiments performed, the EH seems to provide an accurate hierarchy of different behaviour showed by objects in the training set. In addition, the EH is totally understandable.

So, the EH can be considered a higher level source of information than that provided by the subsumption hierarchy. It is quite similar to the initial basic taxonomy induced in [Kietz and Morik, 1994]; although YAYA's EH contains representations for non-primitive concepts, and is updated throughout the learning process.

There are a subset of operators specially designed to benefit from the EH.

5 The Inference Engine (YIE)

As it has been said before, YIE is an inference engine that explicitly modifies the ABox. That is, YIE creates nodes and links between nodes. When one of the rules introduced in section 3 (either probabilistic or not) tells YAYA that an ABox object a should hold a YCL term p, YIE starts a process that adds information to the ABox: a) it adds new links between objects; and b) it creates new objects when necessary. When the process ends, the ABox contains enough information to know a holds p.

As we have said before, YIE tries to add the minimum amount of information to the ABox. Intuitively, when there is some rule saying that an object a should hold a YCL term p; it looks for the most specific YCL terms held by a that subsumes p. Then, it tries to reach p from those subsumers. Figure 1 shows an example of a simple inference performed by YIE. See [Alvarez, 1998] for a formal definition of all this.

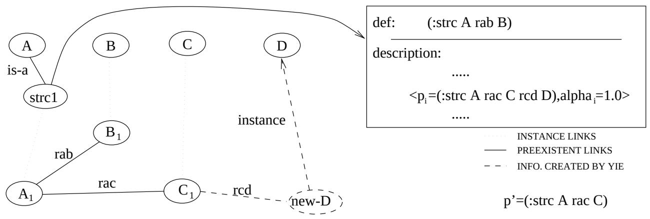

Figure 1: From *strc1* description we know that A instances should hold p_i. As p_i is not held by a_1, YIE looks for the most specific p_i subsumer held by a_1. That subsumer is p'. p_i is reached from p' by generating a new node ($new - D$) and relating c_1 to it by rcd.

Information added by YIE to the ABox can be broken down into five basic types of *atomic inference*: the addition of a link, the removal of a link (for inconsistencies), the creation of a new node, being the instance of a concept, and the unification of two nodes (YIE can realize that two nodes represent the same object).

It can happen that YIE creates a node and later it realizes that the created node corresponds to a node that already was in the ABox. This forces YIE to introduce the *node unification* basic inference, and to relax *unique name assumption* while YIE is working (see figure 2).

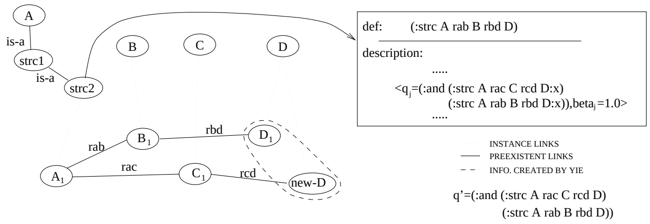

Figure 2: Suppose that after the process in figure 1, the *rbd* link between b_1 and d_1 has been added by YIE. Then, a_1 is classified as an $strc2$ instance. There is a rule telling YIE that a_1 should hold q_j. YIE looks for the most specific q_j subsumer held by a_1. This corresponds to the YCL term q'. So, YIE infers that d_1 and $new - D$ represent the same object.

6 Experiments

Some basic experiments using several hundreds artificially generated objects from several models have been performed. Results have been promising and have served to fine tune the learning algorithm.

A natural language experiment using the training set defined in [McClelland and Kawamoto, 1986] and reused later by [Zelle and Mooney, 1993] has been performed. The training set contains the text for 1500 simple sentences and their corresponding semantic interpretation. Only 126 sentences have been used in the experiment (those corresponding to the verb eat). YAYA tried to learn restrictions among word senses and semantic roles. This kind of information can later be very useful in Word Sense Disambiguation tasks.

The initial TBox contained a simple language model (only primitive concepts), and the hierarchy of word senses taken from WordNet [Miller *et al.*, 1991] (a linguistic ontology with over 90000 word senses). Words in the training set have been manually disambiguated.

The TBox resulting from the learning process contained all the restrictions in the 126 sentences, and no useless information. The experiment has demonstrated that the YAYA learning process is able to use quite a large ontology as background knowledge.

7 Conclusion

A Description Logic System called YAYA has been presented. It has been designed to acquire models from sets of examples in complex domains. The goal of the acquired model is to allow YIE to complete the information of new sets of examples from the same domain. Relying on all this, YCL, the heuristics used to guide the learning process and YIE have been designed.

YCL has been designed to easily capture structures appearing in a semantic network. No abstraction other than that provided by the concept taxonomy is allowed. YCL syntax allows a limited use of variables.

YAYA provides also a learning algorithm and an explicit inference mechanism that works by introducing basic inferences into the ABox. The combination between *unique name assumption* relaxation and *node unification* basic inference simplifies the work to be done by YIE and, makes it more robust.

Some successful but "small" experiments have been performed. The next step is to scale up the learning process and make it able to deal with real noisy domains.

References

[Alvarez, 1998] J. Alvarez. YAYA technical manual. Technical report, Universitat Politècnica de Catalunya, 1998.

[Bonifati *et al.*, 1997] A Bonifati, L Palopoli, D. Saccà, and D. Ursino. Discovering description logic assertions from database schemes. In *International Workshop on Description Logics (DL'97)*, 1997.

[Borgida, 1996] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82:353–367, 1996.

[Brachman *et al.*, 1991] R. Brachman, D. McGuiness, F. Patel-Schneider, L. Resnik, and A. Borgida. Living with Classic: When and how to use a KL-ONE-like language. In Sowa [1991], pages 401–456.

[Cohen and Hirsh, 1994] W. Cohen and H. Hirsh. The learnability of description logics with equality constraints. *Machine Learning*, 17(2–3):169–199, 1994.

[Donini *et al.*, 1996] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Studies in Logic, Language and Information*, pages 193–238. CLSI Publications, 1996.

[Fisher, 1987] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[Kietz and Morik, 1994] J. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–217, 1994.

[McClelland and Kawamoto, 1986] J. L. McClelland and A. H. Kawamoto. *Parallel Distributed Processing. Vol II*, chapter Mechanisms of Sentence Processing: Assigning Roles to Constituents of Sentences, pages 318–362. MIT Press, 1986.

[Miller *et al.*, 1991] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Five papers on WordNet. *International Journal of Lexicography*, 1991.

[Quinlan, 1990] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Shapiro, 1991] Stuart C. Shapiro. Cables, paths, and "subconscious reasoning in propositional semantic networks. In Sowa [1991], chapter 4, pages 137–156.

[Sowa, 1991] John F. Sowa, editor. *Principles of Semantic Networks. Explorations in the Representation of Knowledge*. Morgan Kaufman Publishers, 1991.

[Ventos, 1996] V. Ventos. A deductive study of the C-CLASSIC description logic. In *International Workshop on Description Logics (DL'96)*, 1996.

[Woods, 1991] W. A. Woods. Understanding subsumption and taxonomy: A framework for progress. In Sowa [1991], chapter 1, pages 45–94.

[Zelle and Mooney, 1993] J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *817–822*, page Proceedings of 11th AAAI, 1993.