Towards a Nonmonotonic Extension to OWL

Yarden Katz and Bijan Parsia

MIND Lab, University of Maryland
College Park, MD 20740

Abstract. We discuss $\mathcal{ALCK}$, a nonmontonic logic that augments $\mathcal{ALC}$ with the epistemic operator $\mathbf{K}$, and argue that a similar extension to OWL would be desirable. We show, however, that at its current state the OWL syntax is too inflexible to capture even this syntactically simple-minded extension. Finally, we introduce our implementation of $\mathcal{ALCK}$ as an extension to the tableau-based OWL-DL reasoner Pellet.

1 Nonmonotonic logics, open and closed worlds

Nonmonotonic logics were proven generally successful in capturing, among other things, several forms of common sense and database reasoning. A bold divide between the field of nonmonotonic formalisms and first-order reasoning (where description logics and OWL lie) rests in the *closed* v. *open world* assumption. In database systems, it is reasonable to assume that the data at hand is complete. Since no information outside the database is relevant, we say that our world is *closed.* Formally, if $\Sigma \not\models \varphi$ (where Σ is the knowledge in our database and φ is some formula), we assume $\neg\varphi$ is the case. This constitutes the *closed-world assumption* (CWA). Under the *open world assumption* (OWA), on the other hand, it is accepted that the knowledge in our database is perhaps incomplete. When asked whether $\Sigma \models \varphi$, we simply come to *no conclusion* in case $\Sigma \not\models \varphi$ and $\Sigma \not\models \neg\varphi$.

We will not delve into the question of which semantics is "better" overall or more appropriate for OWL. Rather, we argue that it would be desirable to be able to "turn on" the closed-world assumption when needed in OWL, in order the reap the benefits of nonmonotonicity, but without giving up OWL's open-world semantics in general. The logic $\mathcal{ALCK}$ allows for this interaction. Many useful nonmonotonic features such as integrity constraints and procedural rules (among others−we point to [1] for full treament) are formalizable in this logic.

Consider as a motivational example for CWA, an *integrity constraint.* Suppose that for a certain application, we require all the employees in our knowledge base to be explictly declared to be either a manager or a programmer, since employees will be treated differently according to their positions. The subsumption axiom $\mathit{Employee} \sqsubseteq \mathit{Manager} \sqcup \mathit{Programmer}$ will not suffice, since an individual only declared to be an instance of *Employee* will trivially satisfy the disjunction even if it is not explcitly a member of either disjunct (so long as it is not in $\neg \mathit{Manager} \sqcap \neg \mathit{Programmer}$ of course.) Thus the knowledge base $\Sigma = \{\mathit{Employee}(\mathit{bob})\}$ is a legitimate model of this concept, yet does not settle the question of whether *bob* is actually a manager or a programmer. An integrity constraint in this case will *check* that every employee is *known* to be one or the other.

Integrity constraints are common in database and frame systems, where the CWA is usually made. Without completely abandoning OWL's open world, it would be useful to capture such constraints. The epistemic formulation of such integrity constraints−i.e. asking the database only for *known* facts−was noted by Reiter. These constraints, phrased epistemically, can be captured in $\mathcal{ALCK}$.

2 $\mathcal{ALCK}$

In [1], the epistemic operator $\mathbf{K}$[1] is added to the description logic $\mathcal{ALC}$. The $\mathbf{K}$ operator allows queries that assume the CWA, making $\mathcal{ALCK}$ a nonmonotonic formalism. The $\mathbf{K}$ operator (which is a kind of necessity operator) can be applied to a concept or role.

We focus on the use of $\mathbf{K}$ in queries and assume that queries are posed to an $\mathcal{ALC}$ knowledge base Σ. Intuitively, the query $\langle a : \mathbf{K}C\rangle$ (resp. $\langle a\mathbf{K}Rb\rangle$ for a role) is read as "Is the individual *a known to be C*?" More formally, we let $\mathcal{I}$ be the usual interpretation, and $\mathcal{W}$ a set of interpretations that satisfy C. We say $\Sigma \models \langle a : \mathbf{K}C\rangle$ if and only if for every interpretation $\mathcal{I} \in \mathcal{W}$, $a \in \mathcal{C}^{\mathcal{I}}$. The set $\mathcal{W}$ must be maximal, which means that in our case it is simply the set of all first-order models of C. Returning to our motivational example, the closed-world query desired can be phrased as: $\Sigma \models \langle bob : \mathbf{K}Manager \sqcup \mathbf{K}Programmer\rangle$.

3 An OWL syntax for K?

The **K** operator is quite simple from a syntactic standpoint: it can be applied to classes and properties. We outline several failed approaches to encoding **K** into the RDF/XML syntax of OWL. It is clear that the operator cannot be encoded as a class, for the simple reason that it applies to other classes and properties. This left us with two options:

1. **As a property?** We could coin a new reserved OWL object property, `owl:K`. To represent $\langle a : \mathbf{K}C\rangle$, we must make an anonymous class related via `owl:K` to C. Suppose that we want to encode $\langle bob : Manager \sqcap \neg\mathbf{K}Hacker\rangle$:

```
<rdf:Description rdf:about="#bob">
  <rdf:type>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Manager"/>
        <owl:Class>
          <owl:complementOf>
            <owl:Class>
```

[1] Named after Kripke originally

```
              <owl:K>
                <owl:Class rdf:about="#Hacker"/>
              </owl:K>
            </owl:Class>
          </owl:complementOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </rdf:type>
</rdf:Description>
```

There are two immediate problems with this: (1) we are forced to generate superfluous anonymous classes to maintain the "striped" structure of the syntax (see [4]), and (2) we are able to arbitrarily name (and refer to using `nodeID`) any of our anonymous classes, causing the structure of the above encoding to become even less clear. However, setting aside these problems, this method will simply not work for **K** on properties (e.g. $a\mathbf{K}Rb$) as there is no way to "apply" a property to a property.

2. **As an annotation property?** The second alternative would be to represent **K** as an annotation property that is used to "label" classes. The advantage of this approach is that annotation properties can be used to label both classes and properties. To avoid having an empty annotation, we call the property `modality` and assume that the annotation will always be the string "K". Let us try to represent the assertion $\langle bob : \mathbf{K}Manager \sqcap \mathbf{K}Person \rangle$ using this method:

```
<rdf:Description rdf:about="bob">
  <rdf:type>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Manager">
          <owl:modality>K</owl:modality>
        </owl:Class>
        <owl:Class rdf:about="#Person">
          <owl:modality>K</owl:modality>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </rdf:type>
</rdf:Description>
```

The problem here is that we could easily reserialize this description such that the syntax becomes ambiguous. For example, it is perfectly correct to pull the following out of the conjunction:

```
<owl:Class rdf:about="#Manager">
  <owl:modality>K</owl:modality>
</owl:Class>
```

And place it in the top-level. In its place in the conjunction, we can substitute `<owl:Class rdf:about="#Manager">`. While this is a valid serialization, our assertion now lost its proper structure. With this change, our assertion simply becomes $\langle bob : Manager \sqcap \mathbf{K}Person\rangle$. This is not the behavior we wanted; we only meant to apply **K** to the class $Manager$ in the first conjunction, and not in general.

With all these complications, remember that we haven't mentioned more sophisticated use of **K**. We have not considered the use of the operator in existential and universal role restrictions, such as $\exists\mathbf{K}R.C$ or $\forall R.\mathbf{K}C$, for example. Even if we adopt the first approach and ignore its severe drawbacks, it is clear that such a syntax is not going to be human-readable.

4 Implementation

We have implemented the $\mathcal{ALCK}$ language as an extension to the tableau-based OWL-DL reasoner Pellet. In addition to the **K** queries outlined above, we also admit a restricted use of **K** in the terminology, in the form of an *epistemic rule*. An epistemic rule is of the form of $\mathbf{K}C \sqsubseteq D$ where C and D are $\mathcal{ALC}$ concepts. Frustrated with the failure to come up with an OWL syntax extension, we have extended the KRSS format to allow for **K** and epistemic rules. The extension is trivial and is described in [3].

5 Future work and conclusion

Our examples focused on the use of **K** in queries posed to a first-order ($\mathcal{ALC}$) knowledge base. Extending more expressive description logics with this functionality (including epistemic rules) is easy. Other useful features of nonmonotonic logics, like default rules, can be formalized by admitting **K** freely in the terminology, and by adding another epistemic operator, **A**, to the language (see [2].) However, it must be noted that modelling with both operators allowed arbitrarily in the knowledge base can be far from intuitive, compared with many other formalisms. We conclude that aside from the superficial, though problematic barrier posed by the OWL syntax, extending the OWL semantics (or as large a subset of it as possible) to accomodate these operators would be useful and doable, and a task which we are actively pursuing.

References

1. Donini, F (et al). An epistemic operator for description logics. J. Art. Intel. **100** (1998) 225-274.
2. Donini, F (et al). Description logics of minimal knowledge and negation as failure. ACM Trans. Comp. Log. (2001) 1529-3785.
3. Katz, Y. $\mathcal{ALCK}$ implementation in Pellet. http://www.mindswap.org/2005/alck
4. Beckett, D (et al). RDF/XML syntax specification (revised). http://www.w3.org/TR/rdf-syntax-grammar/