Pushing the Sonic Border — Sonic 1.0*

Anni-Yasmin Turhan
Theoretical Computer Science, TU Dresden, Germany
turhan@tcs.inf.tu-dresden.de

Abstract. This paper reports on extensions of the Description Logics non-standard inference system Sonic. The recent contributions to the system are two-fold. Firstly, Sonic is extended by two new of non-standard inferences, namely, implementations of the *good common subsumer* w.r.t. a background terminology and a heuristics for computing a *minimal rewriting.* Secondly, Sonic is available as a plugin for the well-known ontology editor Protégé [11].

1 Introduction and Motivation

Sonic[1] is an inference component for so-called non-standard inferences for Description Logics (DLs). DLs are a class of formalisms for representing terminological knowledge of a given application domain in a structured and well-defined way. The basic notions of DLs are *concept descriptions* and *roles*, representing unary predicates and binary relations, respectively. DLs are a decidable fragment of first-order logics. Besides satisfiability, DL systems usually provide their users with standard inference services like subsumption of concepts.

More recently, non-standard inferences were introduced for DLs. They resulted from the experience with large real-world DL ontologies, where standard inference algorithms did not suffice for building and maintaining purposes. Non-standard inferences realize the so-called *bottom-up approach* [1] —instead of directly defining a new concept, the knowledge engineer introduces several intuitively related, typical examples as objects, which are then automatically generalized into a concept description by the system. This concept description for the closest generalization is then offered to the knowledge engineer as a candidate for the definition of the intended concept. The task of computing such a concept description can be split into two subtasks: computing the most specific concepts of the given objects, and then computing the least common subsumer of these concepts.

Given concept descriptions C_1, C_2 to C_n in a description logic $\mathcal{L}$, the *least common subsumer (lcs)* of all C_i (where $1 \le i \le n$) is defined as the least (w.r.t. subsumption) concept description in $\mathcal{L}$ subsuming all C_i. The idea behind the lcs inference is to extract the commonalities of the input concepts. It has been argued in [1] that the lcs facilitates one step of the 'bottom-up"-approach to

* This work was supported by DFG under the grant BA 1122/4-4.

[1] Sonic stands for "simple ontology non-standard inference component".Sonic is available from http://wwwtcs.inf.tu-dresden.de/~sonic/

the modeling task. For a variety of DLs there have been algorithms devised for computing the lcs, see [1, 15] for details.

The *approximation* in $\mathcal{L}_2$ of a concept description C from a DL $\mathcal{L}_1$ is defined as the least concept description (w.r.t. subsumption) in a DL $\mathcal{L}_2$ that subsumes C. The idea underlying approximation is to translate a concept description into a typically less expressive DL with as little loss of information as possible. Approximation can be used to make non-standard inferences accessible to more expressive DLs so that at least an approximate solution can be computed. Approximation has so far been investigated for a few pairs of DLs, see [10, 9]. The first version of SONIC implemented both of these non-standard inferences, see [16]. SONIC realizes the lcs for the DL $\mathcal{ALEN}$. This DL provides conjunction, existential, value and number restrictions and primitive negation (negation of primitive concept names) as concept constructors. The DL $\mathcal{ALCN}$ extends $\mathcal{ALEN}$ by the boolean operators on concepts. Approximation in SONIC can translate $\mathcal{ALCN}$- to $\mathcal{ALEN}$-concept descriptions.

It has been shown that for this DL the size of the concept descriptions returned by the lcs can grow exponential [15] in the size of the input concept descriptions. Although these worst case complexities are not likely to appear in practice, it is desirable in our application to obtain short concepts descriptions more comprehensible to a human reader. To this end we extend SONIC with an implementation of *minimal rewriting*, that compresses the result obtained from the lcs in a subsequent step. Given a terminology $\mathcal{T}$ and a concept description C (w.r.t. the signature $\mathcal{S}$), then, intuitively, the *minimal rewriting* is a concept description D (w.r.t. signature $\mathcal{S}$) that is equivalent to C w.r.t. $\mathcal{T}$ and is the concept with the smallest concept size with this property. The computation problem for finding a minimal rewriting for the DL $\mathcal{ALE}$ was investigated in [2].

However, the reasoning services provided by SONIC's first version are available for DLs with rather limited expressiveness. This refers as well to the set of concept constructors as to the fact that only *unfoldable* terminologies are supported. Unfoldable terminologies neither allow for cyclic definitions nor multiple concept definitions per concept name. Many applications nowadays use ontologies written in expressive DLs and that make use of cyclic definitions or general concept inclusion axioms. Standard DL reasoners like RACER [12] can handle this kind of ontologies. In order to bridge this gap, at least to some extent, reasoning w.r.t. a background terminology has been introduced recently [4, 5].

Here the idea is to have a fixed *background terminology* defined in an expressive DL, e.g., a large terminology written by experts. The user then wants to extend this terminology and employs a less expressive DL and needs support through the bottom-up approach when building this user-specific extension of the background terminology. In our case the DL $\mathcal{ALC}$ is used for the background terminology and $\mathcal{ALE}$ is the less expressive user DL. For this setting so far no feasible lcs algorithm could be devised. Thus the notion of *good common sub-*

sumers (gcs) was introduced. A gcs is a common subsumer which may, however, not be the least one. Obviously there exist different algorithms for computing the gcs, since the result of such an algorithm can be more or less close (w.r.t. subsumption) to the lcs. In [5] several algorithms to compute a gcs were introduced. One of them computes the gcs based on the subsumption closure and is called the *scs*. This algorithm is implemented in SONIC.

In order to assess the usefulness of the inferences and the bottom-up approach as a whole it is expedient to put them into practice and to offer an implementation to a wider user group. To this end SONIC provides these inferences as a plugin for the ontology editor OILED [6] already in its first version and provides them now as a plugin for ontology editor PROTÉGÉ [11] together with the OWL plugin [14]. However, it should be noted that SONIC is a prototype system under development and not a fully-fledged end-user tool yet.

2 A Heuristic for Minimal Rewriting in SONIC

Usually, algorithms for non-standard inferences, such as computing the lcs, start by unfolding the input concept descriptions (i.e., replace defined names by their definition). These algorithms produce concept descriptions not containing names defined in the terminology, which are rather large and thus hard to comprehend. The idea to produce more succinct result concept description is now to "re-introduce" concept names for (parts of) concept descriptions.

It turned out that the complexity for computing a minimal rewriting for $\mathcal{ALE}$ and unfoldable terminologies is in PSPACE [2], thus a heuristic was devised in [3] that computes a small, but not necessarily minimal rewriting in deterministic polynomial time. The algorithm for this heuristic is implemented in SONIC.

The algorithm $\mathsf{rewrite}(C)$ has an extension phase and a reduction phase in every recursive step. In the extension phase all concept names subsuming C w.r.t. the terminology are conjoined to the concept C. In the subsequent reduction phase all complex sub-concept descriptions of C that are redundant to the names added in the extension phase are removed. The algorithm is then applied to the remaining sub-concept descriptions of C.

Our implementation of the heuristic for a minimal Rewriting is done in Lisp in a straightforward way. In the reduction phase the more selective condition is applied first to reduce the number of subsumption tests. We tested our implementation on a terminology from a chemical process engineering application [8]. For the 510 lcs calls in the test suite it turned out that the concept descriptions were decreased by one third on the average (and up to a half) by computing the minimal rewriting for the result returned by the lcs. The run-times were increased by 3.1 seconds on the average by computing the minimal rewriting, which is a feasible run-time.

3 SCS in Sonic

The methods for computing the lcs are restricted to rather inexpressive descriptions logics not allowing for disjunction (and thus not allowing for full negation). In fact, for languages with disjunction, the lcs of a collection of concepts is just their disjunction, and nothing new can be learned from building it. In order to provide the bottom-up approach also for more expressive DLs, the *extended bottom-up approach* w.r.t. background terminologies was introduced in [4, 5]. In the latter paper several algorithms for computing the gcs are proposed. The concept descriptions returned by these gcs algorithms use only the concept constructors of $\mathcal{ALE}$, but can, however, use concept names that have an $\mathcal{ALC}$ definition in the background terminology.

Sonic implements the scs, i.e., a gcs that is based on the subsumption closure, see [5]. It uses the concept hierarchy of the background terminology and user terminology, which is computed by the DL reasoner, to find the commonalities of the (negated) concept names mentioned in the concept descriptions. Our implementation of the scs is done in Lisp and shares large parts of the code for the lcs implementation for $\mathcal{ALE}$. However, it uses a different unfolding function that unfolds only the $\mathcal{ALE}$-part of a concept description (and leaves names with $\mathcal{ALC}$-definitions in the concept description). It turned out in our experiments presented in [5] that this variant of the gcs performs well w.r.t. the specificity of the results compared to other gcs variants. For example using $\mathcal{ALC}$ to $\mathcal{ALE}$-approximation first and then applying the lcs yields a more general concept description in about 30% of the cases.

4 Sonic plugin for Protégé

Protégé [11] is a well-known ontology editor with a big user community. Moreover, it comes with the OWL plugin [14], which enables user to build OWL ontologies and to connect to Racer or any other DIG [7] compliant standard DL reasoner to classify the ontology. OWL is a standard closely related to DLs and the OWL plugin uses a syntax similar to what is used in DL systems. This and the use of Racer facilitated the task of building the Sonic plugin for Protégé, although Sonic can, of course, handle only a small fragment of OWL. The Sonic plugin provides two new panels for Protégé. One for computing approximations and the other one for computing commonalities of concepts. on the latter one the inferences lcs and scs are both offered. The Sonic user can choose in preferences which of the two inferences is to be applied, if the commonalities of concept descriptions is computed. Moreover rewriting can be applied, if an $\mathcal{ALE}$ terminology is used.

Sonic uses Racer [12] as an underlying DL reasoner for satisfiability and subsumption tests. Moreover it uses Racer to access the definitions of the concepts from the terminology. The two DL reasoners are connected via LRacer [13] a TCP-based client for RACER.

5 Future Work

We have implemented two new inferences to overcome barriers for the usability of SONIC in practical applications. Moreover we offer SONIC as a plugin for PROTÉGÉ users. Future work for SONIC is to optimize the implementation of $\mathcal{ALC}$-$\mathcal{ALE}$ approximation. This can be done already by fairly simple methods, e.g. by introducing caching of intermediate results. To enhance the expressiveness of DLs and terminologies for which the computation of commonalities is available, we would like to investigate the scs for terminologies with cycles and GCIs – here it mainly depends on the unfolding function to come up with a gcs that does not lose too much of the information and is thus still useful in practice.

Finally, we would like to thank Ida Siahaan who implemented SONIC's connection to PROTÉGÉ and to the OWL plugin.

References

1. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, ed., *Proc. of IJCAI-99*, p. 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann.
2. F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, eds., *Proc. of KR-00*, p. 297–308, CA, 2000. Morgan Kaufmann.
3. F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies – revisited. LTCS-Report 00-04, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2000. See `http://lat.inf.tu-dresden.de/research/reports.html`.
4. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In J.J. Alferes and J.A. Leite, eds., *Proc. of JELIA'04*, vol. 3229 of *LNCS*, p. 400–412, Lisbon, Portugal, 2004. Springer.
5. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Applied Logics*, 2005. To appear.
6. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proc. of KI'01*, vol. 2174 of *LNAI*, p. 396–408, Vienna, Sep 2001. Springer.
7. S. Bechhofer, R. Möller, and P. Crowther. The DIG description logic interface. In *Proc. of DL'03*, Rome, Italy, 2003.
8. R. Bogusch, B. Lohmann, and W. Marquardt:. Computer-aided process modeling with MODKIT. *Computers and Chemical Engineering*, p. 963–995, 2001.
9. S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In I. Horrocks and S. Tessaris, eds., *Proc. of DL'02*, nr. 53 of CEUR-WS, 2002.
10. S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuinness, and M. Williams, eds., *Proc. of KR-02*, CA, 2002. Morgan Kaufmann.
11. J. Gennari, M. Musen, R. Fergerson, W. Grosso, M Crubézy, H. Eriksson, N. Noy, and S. Tu. The evolution of PROTÉGÉ-2000: An environment for knowledge-based system development. *Int. Journal of Human-Computer Studies*, 58:89–123, 2003.
12. V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkov, eds., *Proc. of IJCAR '01*, LNCS. Springer, 2001.
13. V. Haarslev, R. Möller, and M. Wessel. *RACER User's Guide and Manual*, 2004. Available from: `http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-19.pdf%`.
14. H. Knublauch, M. Musen, and A. Rector. Editing description logic ontologies with the PROTÉGÉ owl plugin. In V. Haarslev and R. Möller, eds., *Proc. of DL'04*, nr. 104 in CEUR-WS, 2004.
15. R. Küsters and R. Molitor. Computing Least Common Subsumers in $\mathcal{ALEN}$. In B. Nebel, ed., *Proc. of IJCAI-01*, p. 219–224. Morgan Kaufman, 2001.
16. A.-Y. Turhan and C. Kissig. SONIC — Non-standard inferences go OILED. In D. Basin and M. Rusinowitch, eds., *Proc. of IJCAR '04*, vol. 3097 of *LNCS*, p. 321–325. Springer, 2004. Download: `http://wwwtcs.inf.tu-dresden.de/~sonic/`.