SONIC — Non-standard Inferences go OILED

Anni-Yasmin Turhan and Christian Kissig*

TU Dresden, Germany
email: *lastname*@tcs.inf.tu-dresden.de

Abstract. SONIC[1] is the first prototype implementation of non-standard inferences for Description Logics usable via a graphical user interface. The contribution of our implementation is twofold: it extends an earlier implementation of the least common subsumer and of the approximation inference to number restrictions, and it offers these reasoning services via an extension of the graphical ontology editor OILED [3].

1 Introduction and Motivation

Description Logics (DLs) are a family of formalisms used to represent 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. The inference problems for DLs can be divided into so-called standard and non-standard ones. Well known standard inference problems are satisfiability and subsumption of concept descriptions. For a great range of DLs, sound and complete decision procedures for these problems could be devised and some of them are put into practice in state of the art DL systems as FACT [11] and RACER [9].

Prominent non-standard inferences are the least common subsumer (lcs), and approximation. Non-standard inferences resulted from the experience with real-world DL ontologies, where standard inference algorithms sometimes did not suffice for building and maintaining purposes. For example, the problem of how to structure the application domain by means of concept definitions may not be clear at the beginning of the modeling task. This kind of difficulties can be alleviated by non-standard inferences [1, 7].

Given two concept descriptions A and B in a description logic $\mathcal{L}$, the *lcs* of A and B is defined as the least (w.r.t. subsumption) concept description in $\mathcal{L}$ subsuming A and B. The idea behind the lcs inference is to extract the commonalities of the input concepts. It has been argued in [1, 7] that the lcs facilitates a "bottom-up"-approach to the modeling task: a domain expert can select a number of intuitively related concept descriptions already existing in an ontology and use the lcs operation to automatically construct a new concept description representing the closest generalization of them.

Approximation was first mentioned as a new inference problem in [1]. The *approximation* of a concept description C from a DL $\mathcal{L}_1$ is defined as the least

* This work has been supported by the Deutsche Forschungsgemeinschaft, DFG Project BA 1122/4-3.

[1] SONIC stands for "Simple OILED Non-standard Inference Component".

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 from one DL into a typically less expressive DL. Approximation can be used to make non-standard inferences accessible to more expressive DLs so that at least an approximate solution can be computed. In case the DL $\mathcal{L}$ provides disjunction, the lcs of C_1 and C_2 is just the disjunction $(C_1 \sqcup C_2)$. Thus, a user inspecting this concept does not learn anything about the commonalities of C_1 and C_2. Using approximation, however, one can make the commonalities explicit to some extent by first approximating C_1 and C_2 in a sublanguage of $\mathcal{L}$ which does not provide disjunction, and then compute the lcs of the approximations in $\mathcal{L}$. Another application of approximation lies in user-friendly DL systems, such as OilEd [3], that offer a simplified frame-based view on ontologies defined in an expressive background DL. Here approximation can be used to compute simple frame-based representations of otherwise very complicated concept descriptions.

OilEd is a widely accepted ontology editor and it can be linked to both state of the art DL systems, Racer [9] and FaCT [11]. Hence this editor is a good starting point to provide users from practical applications with non-standard inference reasoning services. The system Sonic is the first system that provides some of these reasoning services via a graphical user interface. Sonic can be downloaded from `http://lat.inf.tu-dresden.de/systems/sonic.html`.

2 The Sonic Implementation

Let us briefly recall the DLs covered by Sonic. The DL $\mathcal{ALE}$ offers the top- and bottom-concept $(\top, \bot)$, conjunction $(C \sqcap D)$, existential $(\exists r.C)$, value restrictions $(\forall r.C)$, and primitive negation $(\neg C)$. The DL $\mathcal{ALC}$ extends $\mathcal{ALE}$ by disjunction $(C \sqcup D)$ and full negation. Extending each of these DLs by number restrictions $((\leq n\ r), (\geq n\ r))$ one obtains $\mathcal{ALEN}$ and $\mathcal{ALCN}$, respectively. For the definition of the syntax and semantics of these DLs, refer to [5, 6]. A *TBox* is a finite set of concept definitions of the form $A \doteq C$, where A is a concept name and C is a concept description. Concept names occurring on the left-hand side of a definition are called *defined concepts*. All other concept names are called *primitive concepts*. Sonic can only process TBoxes that are acyclic and do not contain multiple definitions.

2.1 Implementing the Inferences

Sonic implements the lcs for $\mathcal{ALEN}$-concept descriptions and the approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions in Lisp. The algorithm for computing the lcs in $\mathcal{ALEN}$ was devised and proven correct in [12]. This algorithm consists of three main steps: first recursively replace defined concepts by their definitions from the TBox, then normalize the descriptions to make implicit information explicit, and finally make a recursive structural comparison of each role-level of the descriptions. In $\mathcal{ALEN}$ the last two steps are much more involved than in $\mathcal{ALE}$ since the number restrictions for a role, more precisely the at-most restrictions,

necessitates merging of role-successors. The lcs algorithm for $\mathcal{ALEN}$ takes double exponential time in the worst case. Nevertheless, the lcs for $\mathcal{ALEN}$ realized in SONIC is a plain implementation of this algorithm. Surprisingly, a first evaluation shows that for concepts of an application ontology with only integers from 0 to 7 used in number restrictions the run-times remained under a second (on a Pentium IV System, 2 GHz). The implementation of the lcs for $\mathcal{ALE}$ as described in [2] uses unfolding only on demand—a technique known as lazy unfolding. Due to this technique shorter and thus more easily comprehensible concept descriptions can be obtained more quickly, see [2]. To implement lazy unfolding also for $\mathcal{ALEN}$ is yet future work.

The algorithm for computing the $\mathcal{ALCN}$ to $\mathcal{ALEN}$ approximation was devised and proven correct in [5]. The idea underlying it is similar to the lcs algorithm in $\mathcal{ALEN}$. For approximation the normalization process additionally has to "push" the disjunctions outward on each role-level before the commonalities of the disjuncts are computed by applying the lcs on each role-level. The $\mathcal{ALCN}$ to $\mathcal{ALEN}$ approximation was implemented in Lisp using our $\mathcal{ALEN}$ lcs implementation. An implementation of the $\mathcal{ALC}$ to $\mathcal{ALE}$ approximation is described in [6]. It was the basis for the implementation presented here. The worst case complexity of approximation in both pairs of DLs is double exponential time, nevertheless this is not a tight bound. A first evaluation of approximating randomly generated concept descriptions show that, unfortunately, both implementations run out of memory already for concepts that contain several disjunctions with about 6 disjuncts. The implementation of the algorithms for both inferences are done in a straightforward way without code optimizations or sophisticated data structures. This facilitated testing and debugging of SONIC.

Let us illustrate the procedure of lcs and approximation by an example. Consider a $\mathcal{ALCN}$-TBox with role r, primitive concepts A, B, and concept definitions: $C_1 \doteq \exists r.A \sqcap \forall r.B \sqcap (\geq 3\ r)$, $C_2 \doteq \forall r.(A \sqcap B) \sqcap (\geq 2\ r)$, $C \doteq C_1 \sqcup C_2$ and $D \doteq \exists r.(A \sqcap B) \sqcap \exists r.(\neg A \sqcap B)$. If we want to find the commonalities between C and D, we first compute the $\mathcal{ALEN}$-approximation of C and then the $\mathcal{ALEN}$-lcs of D and $approx(C)$. We compute $approx(C)$ by first unfolding C and then extracting the commonalities of C_1 and C_2. Both have a value restriction and the lcs of these restrictions is $\forall r.B$. Both C_i induce the number restriction $(\geq 2\ r)$, since $(\geq 2\ r)$ subsumes $(\geq 3\ r)$. C_1 has a value and an existential restriction inducing the existential restriction $\exists r.(A \sqcap B)$, whereas in C_2 the number restriction requires at least two distinct r-successors which in addition to the value restriction also induces the restriction $\exists r.(A \sqcap B)$. Thus we obtain $approx(C) = \exists r.(A \sqcap B) \sqcap \forall r.B \sqcap (\geq 2\ r)$. In the concept definition of D the occurrence of A and $\neg A$ induce that at least two r-successors exist. Thus the commonalities of C and D are $lcs(approx(C), D) = \exists r.(A \sqcap B) \sqcap (\geq 2\ r)$.

2.2 Linking the Components

In order to provide the lcs and approximation to OILED users, SONIC does not only have to connect to the editor OILED, but also to a DL system

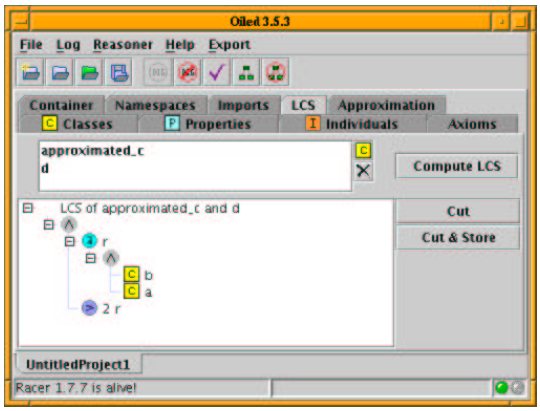

Fig. 1: Interface of SONIC

since both, lcs and approximation, use subsumption tests during their computation. A connection from SONIC to the editor OILED, is realized by a plug-in. Like OILED itself, this plug-in is implemented in Java. SONIC's plug-in is implemented for OILED Version 3.5.3 and realizes mainly the graphical user interface of SONIC—its lcs tab is shown in Figure 1. SONIC's Java plug-in connects via the JLinker interface by Franz Inc. to the Lisp implementation to pass concepts between the components.

To classify an ontology from within OILED, the user can either connect OILED to the reasoner FACT (via CORBA) or to any DL reasoner supporting the DIG ("Description Logic Implementation Group") protocol. The DIG protocol is an XML-based standard for DL systems with a tell/ask syntax, see [4]. DL developers of most systems have committed to implement it in their system making it a promising standard for future DL related software.

SONIC must have access to the same instance of the reasoner that OILED is connected to in order to have access to the information from the ontology, more precisely, to make use of stored concept definitions and of cached subsumption relations obtained during classification by the DL reasoner. Obtaining the concept definitions from OILED directly, would result in storing the ontology in all of the three components and, moreover, the results for lcs and approximation might be incorrect, if OILED and the DL reasoner do not have consistent data.

Since SONIC needs to retrieve the concept definition of a defined concept in order to perform unfolding—a functionality that RACER provides—we decided to use RACER in our implementation. SONIC connects to RACER Version 1.7.7 via the TCP socket interface described in [10]. Note, that in this setting the RACER system need not run locally, but may even be accessed via the web by OILED and SONIC.

2.3 SONIC at Work

Starting the OILED editor with SONIC, the lcs and approximation inferences are available on extra tabs—as shown in Figure 1. Once the OILED user has defined some concepts in the OILED ontology, has connected to the DIG reasoner RACER and classified the ontology, she can use, for example, the lcs reasoning service to add a new super-concept of a number of concepts to the ontology. On the lcs tab she can select some concept names from all concept names in ontology. When the lcs button is clicked, the selected names are transmitted to SONIC's Lisp component and the lcs is computed based on the current concept definitions stored in RACER. The obtained lcs concept description is send to

the plug-in and displayed on the lcs tab in OilEd. Since the returned concept descriptions can become very large, Sonic displays them in a tree representation, where uninteresting subconcepts can be folded away by the user and inspected later. In Figure 1 we see how the concept description obtained from the example in Section 2.1 is displayed in Sonic. Based on this representation Sonic also provides limited editing functionality. The OilEd user can cut subdescriptions from the displayed lcs concept description or cut and store (a part of) it under a new concept name in the ontology.

3 Outlook

Developing Sonic is ongoing work. Our next step is to optimize the current implementation of reasoning services and to implement minimal rewriting to obtain more concise result concept descriptions. Future versions of Sonic will comprise the already completed implementations of the difference operator [6] and of matching for $\mathcal{ALE}$ [8].

We would like to thank Ralf Möller and Sean Bechhofer for their help on how to implement Sonic's linking to Racer and to OilEd.

References

1. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In, *Proceedings of IJCAI-99*, Stockholm, Sweden. Morgan Kaufmann, 1999.
2. F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In *Proceedings of KI'02*, LNAI. Springer–Verlag, 2002.
3. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proceedings of KI'01*, *LNAI*, Springer-Verlag, 2001.
4. S. Bechhofer, R. Möller, and P. Crowther. The DIG description logic interface. In *Proceedings of DL 2003*, Rome, Italy, CEUR-WS, 2003.
5. S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In *Proceedings of DL 2002*, nr. 53 in CEUR-WS. RWTH Aachen, 2002.
6. S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In *Proceedings of KR-02*, Morgan Kaufmann, 2002.
7. S. Brandt and A.-Y. Turhan. Using non-standard inferences in description logics — what does it buy me? In *Proc. of KIDLWS'01*, CEUR-WS. RWTH Aachen, 2001.
8. S. Brandt. Implementing matching in $\mathcal{ALE}$—first results. In *Proceedings of DL2003*, Rome, Italy, CEUR-WS, 2003.
9. V. Haarslev and R. Möller. RACER system description. In *Proceedings of the Int. Joint Conference on Automated Reasoning IJCAR'01*, LNAI. Springer Verlag, 2001.
10. V. Haarslev and R. Möller. *RACER User's Guide and Manual, Version 1.7.7*, Sept, 2003. available from: `http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-7.pdf`.
11. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of KR-98*, Trento, Italy, 1998.
12. R. Küsters and R. Molitor. Computing Least Common Subsumers in $\mathcal{ALEN}$. In *Proceedings of IJCAI-01*, Morgan Kaufman, 2001.