SONIC — System Description*

Anni-Yasmin Turhan and Christian Kissig
Institute for Theoretical Computer Science,
TU Dresden, Germany
lastname`@tcs.inf.tu-dresden.de`

Abstract

SONIC[1] is the first prototype implementation of non-standard inferences for Description Logics that can be used via a graphical user interface. In addition to that our implementation extends an earlier implementation of the least common subsumer and of the approximation inference service to more expressive Description Logics, more precisely to Description Logics with number restrictions. SONIC offers these reasoning services via an extension of the graphical ontology editor OILED [4].

1 Introduction and Motivation

Inference problems for Descriptions Logics (DLs) are 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 for very expressive DLs in state of the art DL reasoners as FACT [15] and RACER [13].

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. Moreover, the expressive power of the DL under consideration can make it difficult to come up with a faithful formal definition of the concept originally intended. This kind of difficulties can be alleviated by the use of non-standard inferences in the *bottom-up* construction of DL knowledge bases, as described in [1, 8]. Here instead of directly defining a new concept, the knowledge engineer introduces several typical examples as objects, which are then automatically generalized into a concept description by the DL system. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept. The task of computing

* 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".

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.

The lcs was first mentioned as an inference problem for DLs in [12]. 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, 8] that the lcs facilitates the "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 these concepts. For a variety of DLs there have been algorithms devised for computing the lcs, see [1, 16, 10] for details.

Approximation was first mentioned as a new inference problem in [1]. The *approximation* of a concept description C_1 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_1. The idea underlying approximation is to translate a concept description 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 concept disjunction, the lcs of C_1 and C_2 is just the concept disjunction $(C_1 \sqcup C_2)$. Thus, a user inspecting this concept description does not learn anything about the commonalities between 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 obtained approximations in $\mathcal{L}$. Approximation has so far been investigated for a few DLs, see [7, 6].

Another application of approximation lies in user-friendly DL systems, such as the editor OilEd [4], 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 reasoners, Racer [13] and FaCT [15]. Hence this editor is a good starting point to provide users from practical applications with non-standard inference reasoning services. The prototype system Sonic is the first system that provides some of the non-standard inference reasoning services via a graphical user interface and thus makes them accessible to a wider user group. Sonic was first introduced in [17] and it can be downloaded from `http://lat.inf.tu-dresden.de/systems/sonic.html`.

In the next section we give an application example which underlines that—although the supported DLs are much less expressive compared to the DLs supported by the current DL reasoners—the inferences implemented in Sonic can be useful in practice. In Section3 we turn to the implementation of Sonic and describe how the inferences are realized and and how Sonic is coupled to OilEd and the underlying DL reasoner Racer. Then we give an impression how users can work with Sonic and in the end we sketch how the Sonic prototype system can be extended in future versions.

2 An Application Example

Let us briefly recall the DLs covered by SONIC. Starting with a set N_C of *concept names* and a set N_R of *role names concept descriptions* are inductively defined with the help of a set of *concept constructors*. The DL $\mathcal{ALE}$ offers the top- ($\top$) and bottom-concept ($\bot$), concept conjunction ($C \sqcap D$), existential restrictions ($\exists r.C$), value restrictions ($\forall r.C$), and primitive negation ($\neg P, P \in N_C$). The DL $\mathcal{ALC}$ extends $\mathcal{ALE}$ by concept disjunction ($C \sqcup D$) and full negation ($\neg C$). Extending each of these DLs by number restrictions, i.e., *at most restrictions* ($\leq n\ r$) and *at least restrictions* ($\geq n\ r$) one obtains $\mathcal{ALEN}$ and $\mathcal{ALCN}$, respectively.

The semantics of these concept descriptions is defined in the usual model-theoretic way in terms of an *interpretation* $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$. The domain $\Delta_\mathcal{I}$ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^\mathcal{I}$ maps each concept name $P \in N_C$ to a set $P^\mathcal{I} \subseteq \Delta_\mathcal{I}$ and each role name $r \in N_R$ to a binary relation $r^\mathcal{I} \subseteq \Delta_\mathcal{I} \times \Delta_\mathcal{I}$. The semantics are extended to complex concept descriptions in the usual way, see for example [1, 6].

A *TBox* is a finite set of concept definitions of the form $A \equiv C$, where A is a concept name and C is a concept description. SONIC can only process TBoxes that are *unfoldable*, i.e., they are acyclic and do not contain multiple definitions. Concept names occurring on the left-hand side of a definition are called *defined concepts*. All other concept names are called *primitive concepts*.

Next, let us illustrate the procedure of computing the lcs for $\mathcal{ALEN}$-concept descriptions and the approximation of $\mathcal{ALCN}$-concept descriptions by $\mathcal{ALEN}$-concept descriptions with an application example. Consider the TBox $\mathcal{T}$ with $\mathcal{ALCN}$-concept descriptions modeling Airbuses and their configurations. $\mathcal{T}$ contains the following concept definitions:

$$
\begin{aligned}
\textsf{Cargo-Config} &\equiv \neg\textsf{Passenger-Config} \\
\textsf{Airbus-300} &\equiv \textsf{Plane} \\
&\quad \sqcap \exists\textsf{has-configuration.Cargo-Config} \\
&\quad \sqcap \exists\textsf{has-configuration.}(\textsf{Passenger-Config} \sqcap (\leq 2\ \textsf{has-classes})) \\
\textsf{Airbus-340} &\equiv \textsf{Plane} \\
&\quad \sqcap (\geq 2\ \textsf{has-configuration}) \\
&\quad \sqcap \forall\,\textsf{has-configuration.}(\textsf{Passenger-Config} \sqcap (\geq 261\ \textsf{has-seats})) \\
&\quad \sqcap \big(\exists\textsf{has-configuration.}((\leq 419\ \textsf{has-seats}) \sqcap (\leq 2\ \textsf{has-classes})) \sqcup \\
&\qquad \exists\textsf{has-configuration.}((\leq 380\ \textsf{has-seats}) \sqcap (\leq 3\ \textsf{has-classes}))\big)
\end{aligned}
$$

If we want to find the commonalities between the concepts Airbus-300 and Airbus-340 by using the lcs in $\mathcal{ALEN}$, we first have to compute the approximation of Airbus-340 since its concept definition contains a disjunction. After that can we compute the lcs of Airbus-300 and $approx_{\mathcal{ALEN}}(\textsf{Airbus-340})$.

In order to compute the approximation of the concept definition of Airbus-340 we first make implicit information explicit. In our example this is done by propagating the value restriction onto the two existential restrictions which yields the new disjunction:

$\exists$has-configuration.
$(\text{Passenger-Config} \sqcap (\geq 261 \text{ has-seats}) \sqcap (\leq 419 \text{ has-seats}) \sqcap (\leq 2 \text{ has-classes})) \sqcup$
$\exists$has-configuration.
$(\text{Passenger-Config} \sqcap (\geq 261 \text{ has-seats}) \sqcap (\leq 380 \text{ has-seats}) \sqcap (\leq 3 \text{ has-classes}))$.

After the propagation of the value restriction, we can obtain the approximation of our example concept description by simply computing the lcs of these two disjuncts. They differ only w.r.t. the number occurring in the at-most restrictions. Consequently we have to pick the at-most restriction with the greater number from the two disjuncts and conjoin them with (Passenger-Config $\sqcap$ ($\geq$ 261 has-seats) to obtain their lcs. We get the following approximation of Airbus-340:

$$\begin{aligned}
&approx_{\mathcal{ALEN}}(\text{Airbus-340}) = \\
&\quad \text{Plane} \ \sqcap \ (\geq 2 \text{ has-configuration}) \\
&\quad \sqcap \forall \text{has-configuration.}(\text{Passenger-Config} \sqcap (\geq 261 \text{ has-seats})) \\
&\quad \sqcap \exists \text{has-configuration.} \\
&\qquad (\text{Passenger-Config} \sqcap (\geq 261 \text{ has-seats}) \sqcap (\leq 419 \text{ has-seats}) \sqcap (\leq 3 \text{ has-classes}))
\end{aligned}$$

To compute the lcs of $approx_{\mathcal{ALEN}}$(Airbus-340) and the concept definition of Airbus-300, we need to unfold the concept Airbus-300 w.r.t. $\mathcal{T}$ by replacing Cargo-Config with its concept definition $\neg$Passenger-Config. It obvious now that the concept description implies two distinct configurations, since one of the existential restriction requires Passenger-Config and the other one $\neg$Passenger-Config. Thus the concept definition of Airbus-300 induces ($\geq$ 2 has-configuration) which occurs in $approx_{\mathcal{ALEN}}$(Airbus-340) directly. The primitive concept Plane appears in both concept descriptions and thus also in their lcs. Since the concept definition of Airbus-300 does not imply a value restriction the lcs does not contain any. Furthermore, we have to compute the lcs of each of the two existential restrictions from Airbus-300 and the one from $approx_{\mathcal{ALEN}}$(Airbus-340). We obtain for the overall lcs:

$$\begin{aligned}
&\text{lcs}(approx_{\mathcal{ALEN}}(\text{Airbus-340}), \text{Airbus-300}) = \\
&\quad \text{Plane} \ \sqcap \ (\geq 2 \text{ has-configuration}) \ \sqcap \ \exists \text{has-configuration.}\top \\
&\quad \sqcap \exists \text{has-configuration.}(\text{Passenger-Config} \sqcap (\leq 3 \text{ has-classes})).
\end{aligned}$$

The first existential restriction is redundant and therefore can be omitted—it would not be returned by our implementation. We have now extracted the commonalities of the Airbus-340 and the Airbus-300: they are both planes with at least two configurations where one configuration is a passenger configuration with up to 3 classes.

Although the DLs under consideration are not very expressible compared to the DLs handled by the state of the art DL reasoners this application example shows that an implementation of lcs and approximation for these DLs can be very useful to help users to extend their ontologies.

3 The SONIC Implementation

The SONIC system implements the algorithms for computing the lcs for $\mathcal{ALEN}$-concept descriptions and the approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions. Furthermore, SONIC implements a graphical user interface to offer these non-standard inferences and an interface to a DL reasoner needed for subsumption queries.

3.1 Implementing the Inferences

We briefly sketch the main idea of the inference algorithms here. The algorithm for computing the lcs in $\mathcal{ALEN}$ was devised and proven correct in [16], thus our implementation is well-founded. The algorithm for computing the lcs of $\mathcal{ALEN}$-concept descriptions consists of three main steps:

1. *Unfold* the input concept descriptions by recursively replacing defined concepts by their definitions from the TBox.
2. *Normalize* the unfolded concept descriptions to make implicit information (e.g. inconsistencies, induced existential restrictions, induced value restrictions or induced number restrictions) explicit.
3. Represent the normalized concepts as concept trees, build the cross-product of the trees and read out a concept description from it.

For the DL $\mathcal{ALEN}$ the normalization and the structural comparison are much more involved than in $\mathcal{ALE}$. Firstly, the number restrictions for roles, more precisely the at most restrictions, necessitates merging of role-successors mentioned in the existential restrictions. To obtain all valid mergings is a combinatorial problem. Second, the commonalities of all mergings for existential restrictions of a concept description have to be determined by computing their lcs. These in turn are then used to compute the cross-product. In our implementation we use lists as data structures to represent the concept descriptions and implement the algorithms without advanced optimizations in order to keep this first implementation of the lcs for $\mathcal{ALEN}$-concept descriptions simple and easy to test.

The lcs algorithm for $\mathcal{ALEN}$ can return concept descriptions double exponential in the size of the input concepts in the worst case. Nevertheless, so far the lcs in $\mathcal{ALEN}$ realized in SONIC is a plain implementation of this algorithm. Surprisingly, a first evaluation of our implementation shows that for concepts of an application TBox with only integers from 0 to 5 used in number restrictions the run-times remained under a second (with Allegro Common Lisp on a Pentium IV System, 2 GHz). Our implementation of the lcs for $\mathcal{ALE}$-concept descriptions as described in [3] uses lazy unfolding. Due to this technique shorter and thus more comprehensible concept descriptions can be obtained more quickly, see [3]. To implement lazy unfolding for the lcs for $\mathcal{ALEN}$-concept descriptions in SONIC is yet future work.

The algorithm for computing the approximation of $\mathcal{ALCN}$-concept descriptions by $\mathcal{ALEN}$-concept descriptions was introduced and proven correct in [6]. The idea underlying this algorithm is similar to the lcs algorithm. For approximation the normalization process additionally has to build a disjunctive normal form on each role-level by "pushing" the disjunctions outward. With concept descriptions in this normal form the commonalities of the disjuncts are computed by applying the lcs on each role-level.

The approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions is also implemented in Lisp and uses the above mentioned implementation of the lcs for $\mathcal{ALEN}$-concept descriptions as a subfunction. A first implementation of the approximation of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions is described in [7]. This implementation is now extended to number restrictions and provided by Sonic.

In the worst case the approximation in both pairs of DLs can yield concept descriptions that are double exponential in the size of the input concepts descriptions. Nevertheless, this is not a tight bound. A first evaluation of approximating randomly generated concept descriptions shows that, unfortunately, both implementations run out of memory already for concepts that contain several disjunctions with about 6 disjuncts. It is unknown whether this kind of concept descriptions appears in application TBoxes from practical applications. Nevertheless, effective optimization techniques are needed for computing approximation, before this service can be applied to large ontologies. Similar to the lcs one can apply lazy unfolding to avoid "unnecessary" unfolding and thereby obtain smaller concept descriptions even faster. Besides lazy unfolding there is also the approach of so called *nice concepts* described in [9] known as an optimization technique for approximation. Currently these techniques are implemented and evaluated for approximation of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions in a student's project in our group.

The implementation of the algorithms for both inferences are realized in a straightforward way without sophisticated data structures or advanced optimizations as, for example the caching of results. This facilitated the testing and debugging of the Sonic prototype.

3.2 Linking the System Components

In order to provide the inferences lcs and approximation to users of the ontology editor OilEd, we need not only to connect to the editor OilEd, but also to a DL reasoner since both inferences, lcs and approximation, use subsumption tests heavily during their computation. The connection from Sonic to the editor OilEd, is realized as 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 (or higher) and realizes mainly the graphical user interface of Sonic. A screen-shot of the lcs tab in Sonic is shown in Figure 3.2. Sonic's Java plug-in connects via a Telnet connection to the Lisp implementation of the non-standard inferences to pass concept descriptions or messages between the components.

The OilEd user can classify an ontology in the OilEd editor, by either connecting OilEd to the FaCT reasoner via a CORBA interface or to any DL reasoner supporting the DIG ("Description Logic Implementation Group") protocol. The DIG protocol is an XML-based interface for DL systems with a tell/ask syntax, see [5]. DL developers of most DL systems have committed to implement this standard in their system making it a standard for 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

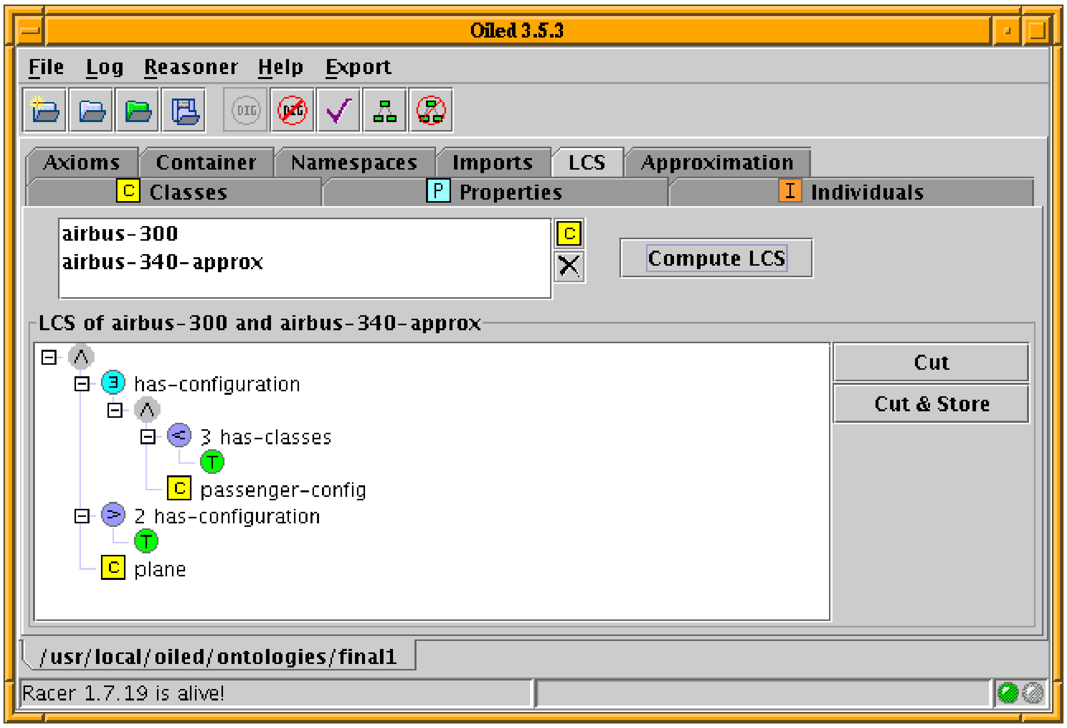

Figure 1: SONIC's lcs Tab in OILED.

during classification by the DL reasoner. If OILED and the DL reasoner do not have consistent versions of the ontology, the computed results for lcs and approximation might simply be incorrect due to this inconsistency.

SONIC needs the functionality of retrieving the concept definition of a concept defined in the TBox in order to perform unfolding. Since such a function is not included in the DIG protocol, we cannot use the DIG interface to connect to the DL reasoner. Since the CORBA interface to FACT is slow, we use RACER as underlying DL reasoner. SONIC connects to RACER version 1.7.7 (or higher) via the TCP socket interface described in [14]. Note, that in this setting the RACER reasoner need not run locally, but may even be accessed via the web by OILED and SONIC.

3.3 SONIC at Work

After the user has started the OILED editor with SONIC, the lcs and approximation inference are available on two extra tabs in OILED— as shown in Figure 3.2. After 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. In order to do so she can select some of the concept names from the ontology on the lcs tab. When the button 'compute lcs' is clicked, the selected concept names are transmitted to SONIC's Lisp component and the lcs is computed based on the current concept definitions stored in Racer.[2] The concept description obtained from the lcs implementation is send to the plug-in via Telnet and displayed on the lcs tab. The approximation inference is offered on a similar Sonic tab in OilEd.

Since the concept descriptions returned by the lcs and the approximation inference can be very large, it is not feasible to display them in a plain fashion. Sonic displays the returned concept descriptions in a tree representation, where uninteresting subconcepts can be folded away by the user and inspected later. In Figure 3.2 we see how the concept description for the lcs obtained from the application example in Section 2 is displayed on Sonic's tab in OilEd. Based on this representation Sonic also provides limited functionality on both of its tabs to edit concept descriptions. OilEd users can 'cut' subdescriptions from the displayed concept description and thereby reduce the displayed concept description to interesting aspects. OilEd users can also 'cut and store' (a part of) the obtained concept description under a new concept name in their ontology.

4 Conclusions and Future Work

The Sonic prototype is a graphical tool for supporting main steps of the bottom-up approach for augmenting ontologies. These steps are realized by implementations of the least common subsumer in $\mathcal{ALEN}$ and the approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions. These reasoning services can be used from within the OilEd ontology editor. Since Sonic is the first system that implements a graphical user interface for non-standard inferences, it is now possible to evaluate how useful these inference services are to users from practical applications.

Currently there is a big language gap between the DLs implemented in the state of the art DL reasoners and the DLs for which non-standard inferences are investigated or even implemented. To overcome this language gap to some extend we are currently studying a new approach to compute approximate solutions for the lcs and thus obtain a "good" common subsumer (instead of a least one) for input concept descriptions referring to concepts defined in a more expressive DL, see [2].

Developing Sonic is ongoing work. Our next step is to optimize the current implementation of approximation—on the one hand to speed-up the computation and on the other hand to obtain smaller concepts. This can be achieved by using lazy unfolding as our lcs implementation for $\mathcal{ALE}$ has shown, see [3]. Another step is to implement minimal rewriting w.r.t. TBoxes to obtain more concise and thus better comprehensible result concept descriptions from both reasoning services. In the longer run we want to comprise the implementations of the difference operator (see [7]) and of matching for $\mathcal{ALE}$ (see [11]) in Sonic and provide these inference services to users from practical applications. The Sonic system can be down loaded for research purposes from `http://lat.inf.tu-dresden.de/systems/sonic.html`.

Finally we would like to thank Ralf Möller, Volker Haarslev and Sean Bechhofer for their help on how to implement Sonic's linking to Racer and OilEd.

[2]This is why the TBox should be classified first.

References

[1] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, editor, *Proceedings of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann, Los Altos.

[2] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. In the Proceedings of 2004 International Workshop on Description Logics (DL 2004). To appear.

[3] Franz Baader and Anni-Yasmin Turhan. On the problem of computing small representations of least common subsumers. In *Proceedings of the 25th German Annual Conf. on Artificial Intelligence (KI'02)*, LNAI. Springer–Verlag, 2002.

[4] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proceedings of the 24th German Annual Conf. on Artificial Intelligence (KI'01)*, volume 2174 of *LNAI*, pages 396–408, Vienna, Sep 2001. Springer-Verlag.

[5] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG description logic interface. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, Rome, Italy, 2003.

[6] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In *Proceedings of the 2002 Description Logic Workshop (DL 2002)*, number 53 in CEUR-WS. RWTH Aachen, April 2002.

[7] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuinness, and M.-A. Williams, eds., *Proceedings of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-02)*, 2002. Morgan Kaufmann Publishers.

[8] Sebastian Brandt and Anni-Yasmin Turhan. Using non-standard inferences in description logics — what does it buy me? In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (KIDLWS'01)*, number 44 in CEUR-WS, Vienna, Austria, September 2001. RWTH Aachen.

[9] Sebastian Brandt and Anni-Yasmin Turhan. An approach for optimized approximation. In *Proceedings of the KI-2002 Workshop on Applications of Description Logics (KIDLWS'02)*, number 63 in CEUR-WS, Aachen, Germany, September 2002. RWTH Aachen.

[10] Sebastian Brandt, Anni-Yasmin Turhan, and Ralf Küsters. Extensions of non-standard inferences for description logics with transitive roles. In *Proceedings of the 10th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'03)*, LNCS. Springer, 2003.

[11] Sebastian Brandt. Implementing matching in $\mathcal{ALE}$—first results. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, CEUR-WS, 2003.

[12] Wiliam Cohen, Alex Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In W. Swartout, editor, *Proceedings of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, pages 754–760, San Jose, CA, 1992. AAAI Press/The MIT Press.

[13] Volker Haarslev and Ralf Möller. RACER system description. In *Proceedings of the International Joint Conference on Automated Reasoning IJCAR'01*, LNAI. Springer Verlag, 2001.

[14] Volker Haarslev and Ralf Möller. *RACER User's Guide and Manual*, Sept. 2003. available from: `http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-7.pdf`.

[15] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In A.G. Cohn, L.K. Schubert, and S.C.Shapiro, editors, *Proceedings of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 636–647, 1998.

[16] Ralf Küsters and Ralf Molitor. Computing Least Common Subsumers in $\mathcal{ALEN}$. In B. Nebel, editor, *Proceedings of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI-01)*, pages 219–224. Morgan Kaufman, 2001.

[17] Anni-Yasmin Turhan and Christian Kissig. Sonic—Non-standard inferences go OilEd. In *Proceedings of the International Joint Conference on Automated Reasoning IJCAR'04*, LNAI. Springer Verlag, 2004. To appear.