

SONIC Manual – Version 0.1

Anni-Yasmin Turhan
turhan@tcs.inf.tu-dresden.de

Christian Kissig
kissig@tcs.inf.tu-dresden.de

TU Dresden
Theoretical Computer Science
Chair for Automata Theory
Hans-Grundig Strasse 25
01062 Dresden

November 2004

Contents

1	Introducing SONIC	1
2	How to set-up SONIC	2
2.1	How to obtain SONIC	2
2.2	Required system resources	3
2.3	Required components	3
2.4	How to install SONIC	4
3	How to start SONIC	4
4	How to use SONIC	4
4.1	DLs and ontologies handled by SONIC	5
4.2	How to compute an Approximation	6
4.3	How to process a result concept	8
4.4	How to compute a LCS	8
5	Future Work—What’s next?	9
	References	10

1 Introducing SONIC

SONIC¹ is a prototype system developed in a research project dedicated to so-called *non-standard inferences* for Description Logics —for an overview of these kind of inferences see [7]. SONIC offers non-standard inferences via the well-known ontology editor OILED. SONIC was and still is under development in a research project funded by the German Research Community (DFG) under grant BA 1122/4-3.

The inference problems for Description Logics (DLs) can be divided into so-called standard and non-standard ones. Well-known standard inference problems are satisfiability and subsumption of concepts. For a great range of DLs, sound and complete decision procedures for these problems could be devised, see for example [3] and most of them are put into practice in state of the art DL systems such as RACER [8].

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 the *bottom-up approach* where new concepts are derived in an example-driven way. The bottom-up approach is realized by non-standard inferences [2, 7] and (partially) implemented in SONIC. In the current version SONIC implements two non-standard inference services, namely *least common subsumer (LCS)* and *Approximation*.

LCS of a set of concepts \mathcal{M} is the concept that subsumes each of the concepts from the set \mathcal{M} and is the least one (w.r.t. subsumption) with this property. Using the LCS one can generalize a set of concepts into one single concept by extracting the commonalities. SONIC implements the LCS inference for the DL \mathcal{ALN} .

Approximation takes one concept from a DL \mathcal{L}_1 and translates it into another, typically less expressive DL \mathcal{L}_2 . Since the DL \mathcal{L}_2 is less expressive, the obtained concept might be more general. So, using the Approximation inference one can translate a concept from one DL to another. SONIC implements Approximation for \mathcal{ALN} concepts by \mathcal{ALN} concepts.

SONIC is a plug-in for the well-known ontology editor OILED (see [5]) and offers the non-standard reasoning services via OILED's graphical user interface. OILED can be linked to state of the art DL (standard) reasoning systems, such as RACER. SONIC uses RACER as an underlying reasoner when computing LCS or Approximation. Hence, the OILED editor is a good starting point to provide users from practical applications with non-standard inference reasoning services. The SONIC system is the first software system that provides these reasoning services via a graphical user interface. The SONIC system can be freely down-loaded for research or teaching purposes from the web page:

¹SONIC stands for "Simple OILED non-standard inference component".

<http://lat.inf.tu-dresden.de/systems/sonic.htm>. For more information on how the SONIC system is realized and on the implementations of the inferences please refer to [10, 11].

Disclaimer: SONIC comes with absolutely no warranty. Use at your own risk. Commercial use is prohibited; contact sonic@tcs.inf.tu-dresden.de for licensing and further information.

Acknowledgments: We would like to thank some persons who were involved in the development of SONIC (in no particular order): Ralf Möller, Volker Haarslev, Sean Bechhofer, Ekaterina (Katja) Timoshenko and Jörg Model.

If you have questions regarding SONIC, suggestions to make how to improve SONIC or if you have errors to report, please do not hesitate to contact us.

2 How to set-up SONIC

2.1 How to obtain SONIC

SONIC can be freely down-loaded for research purposes from the website

<http://lat.inf.tu-dresden.de/systems/sonic.htm>.

There are several distributions for different operation systems and different LISP systems. Depending on what LISP system you use down-load the appropriate files. After the down-load unpack the down-loaded file using GZip or BZip2 under Linux or Zip under Microsoft Windows. The SONIC distribution contains the following files:

- `Read-me.txt`,
- `Sonic-manual.pdf`,
- `install-sonic-cmucl.sh` for CMU Common LISP or `install-sonic-acl.sh` for Allegro Common LISP,
- `sonic.x86f` for CMU Common LISP or `sonic.dxl` for Allegro Common LISP,
- `start-sonic-cmucl.sh` for CMU Common LISP or `start-sonic-acl.sh` for Allegro Common LISP and
- `SonicPlugin.jar`

Before installing SONIC, make sure that the following resources and software components are already installed on your system.

2.2 Required system resources

We recommend an i686 processor and at least 128 RAM to run SONIC. So far SONIC can only run under Linux operating systems. SONIC requires that Java (for example Sun's J2SE 1.5 Version 2.1 or higher) is installed on your system. In addition one of the supported LISP systems must be installed. Currently SONIC can run under:

- CMU Common LISP (version 18e or higher),
- Allegro Common LISP (version 6.2).

It is planned to provide SONIC for more LISP systems and for other operating systems in the future.

2.3 Required components

Besides a LISP and a Java system you need two software components to be installed on your system to run SONIC.

OILED [5] is an ontology editor for Description Logics. It allows users to build and maintain their ontologies. Moreover, OILED allows to connect to different standard description logic reasoners and displays the results of their computations, e.g. the concept hierarchy.

SONIC extends OILED by a plug-in and thus OILED must be installed on your system before you can run SONIC. SONIC works with OILED version 3.5.3 or higher. OILED can be freely down-loaded for research purposes from the website: <http://oiled.man.ac.uk/>.

RACER [8] is the state of the art Description Logics reasoner for standard inferences. It can be used to check whether the concept definitions are satisfiable and it can classify the concepts defined in the ontology.

SONIC uses RACER as a background reasoner, thus RACER must be installed on your system before you can run SONIC. There are two RACER system components needed to run SONIC— the RACER server and LRACER. While the LRACER component comes with the SONIC distribution and is installed by the SONIC install script, the RACER server must be down-loaded and installed by you. SONIC uses RACER server version 1-7-14 or higher. We recommend to use the latest available version. The RACER server can be freely down-loaded for research purposes from the author's web pages: either from <http://www.sts.tu-harburg.de/~r.f.moeller/racer> or from <http://www.cs.concordia.ca/~haarslev/racer/>.

For the installation and use of these systems please refer to their manuals (OILED [4], RACER [9]).

2.4 How to install SONIC

SONIC comes with an installation script called `install-sonic-cmucl.sh` for CMU Common LISP or `install-sonic-acl.sh` for Allegro Common LISP. Under Linux operating system you can start the installation script by executing `install-sonic-cmucl.sh` or `install-sonic-acl.sh` in the shell. This script installs SONIC and LRACER. It edits OILED's preferences s.t. the SONIC panels are loaded when OILED is started. In case the installation procedure was not successful, please refer to the file `Read-me.txt` from the SONIC distribution for trouble shooting.

3 How to start SONIC

SONIC consists of two parts: one part realizes the graphical user interface as a plug-in for OILED—realized in `SonicPlugin.jar`—and the other part is the SONIC server and implements the non-standard inferences in LISP—realized in `sonic.x86f` or `sonic.dxl`, respectively. In the current implementation the SONIC server has to be started by the script `start-sonic-cmucl.sh` for CMU Common LISP or `start-sonic-acl.sh` for Allegro Common LISP. If SONIC is correctly installed as described in Section 2.4, the SONIC plug-in for OILED is loaded when OILED is started. Moreover, one has to start the RACER server before starting the SONIC server. In the current version all three components, the RACER server, OILED and the SONIC script have to run on the same host. To sum it up the complete start procedure for SONIC is:

1. Start RACER (with HTTP-port 8080), see [9].
2. Execute the script `start-sonic-cmucl.sh` for CMU Common LISP or `start-sonic-acl.sh` for Allegro Common LISP.
3. Start OILED as usual, see [4].
4. Connect OILED to the DIG reasoner RACER.

After these steps the OILED window should look as in Figure 1. The panels “LCS” and “Approximation” should be loaded and in the lower left corner the information should appear that RACER is alive.

4 How to use SONIC

After SONIC is set up and started as described in the last sections, an ontology must be loaded or be build in order to use the reasoning services of SONIC. In the following we assume that there is an ontology loaded into OILED.

SONIC uses RACER as an underlying reasoner and thus depends on the concept definitions as transferred from OILED to RACER. This necessitates that the ontology is classified each time after the ontology has been edited and before the reasoning services provided by SONIC can be invoked.

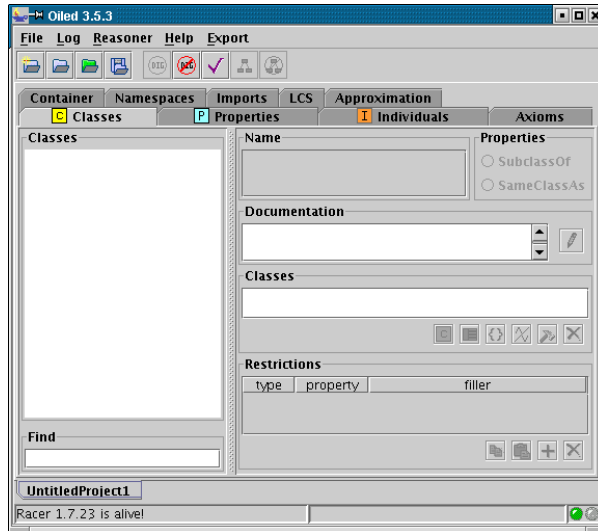


Figure 1: OILED with SONIC plug-in.

4.1 DLs and ontologies handled by SONIC

In the current version SONIC can provide reasoning services for the Description Logics \mathcal{ALCN} and \mathcal{ALEN} and their respective sublanguages. Ontologies to be processed by SONIC may only be build by concept constructors offered in these languages. The description logic \mathcal{ALEN} offers conjunction ($C \sqcap D$), existential restriction ($\exists r.C$), value restriction ($\forall r.C$), the negation of concept names —so-called primitive negation ($\neg C$) and unqualified number restrictions ($(\leq nr), (\geq nr)$). The DL \mathcal{ALCN} offers in addition to the concept constructors of \mathcal{ALEN} disjunction ($C \sqcup D$) and full negation ($\neg C$), i.e., negation of arbitrary concepts. All DLs and concept constructors used in SONIC are displayed in Table 1, for the semantics of the concept constructors please refer to [1].

SONIC can only handle ontologies that are *unfoldable*, i.e., only concept names may appear on the left-hand side of concept definitions, each concept name may only appear once on the left-hand side of a concept definition and concept definitions may not contain cycles.

We illustrate how to work with SONIC by a running example. The following ontology models a few concepts from the application domain of Airbuses and Airbus configurations. We can first state in our ontology that a cargo configuration is not a passenger configuration by using primitive negation:

$$\text{Cargo-Configuration} \equiv \neg \text{Passenger-Configuration}.$$

Next, we can specify some facts about Airbuses. For the Airbus 300 we can state that it is a plane and that it has a cargo configuration. Moreover, an Airbus 300 has a configuration that is a passenger configuration with at most

Construct name	$\mathcal{AL}\mathcal{E}$	$\mathcal{AL}\mathcal{E}\mathcal{N}$	$\mathcal{AL}\mathcal{C}$	$\mathcal{AL}\mathcal{C}\mathcal{N}$
Conjunction	X	X	X	X
Existential restrictions	X	X	X	X
Value restrictions	X	X	X	X
Number restrictions		X		X
Negation	prim.	prim.	full	full
Disjunction			X	X

Table 1: Overview of DLs supported in SONIC.

two classes:

$$\text{Airbus-300} \equiv \text{Plane} \sqcap \exists \text{has-configuration.Cargo-Config} \sqcap \\ \exists \text{has-configuration.}(\text{Passenger-Configuration} \sqcap (\leq 2 \text{ has-classes})).$$

Note, that this concept definition uses only concept constructors from $\mathcal{AL}\mathcal{E}\mathcal{N}$. Next, we give a description of an Airbus 340. It is a plane with all configurations being passenger configurations with at least 260 seats. Moreover, an Airbus 340 has a configuration, which has at most 419 seats and has at most 2 classes or it has a configuration, which has at most 380 seats and has at most 3 classes. This can be written in DL as:

$$\text{Airbus-340} \equiv \text{Plane} \sqcap (\geq 2 \text{ has-configuration}) \sqcap \\ \forall \text{has-configuration.}(\text{Passenger-Configuration} \sqcap (\geq 261 \text{ has-seats})) \sqcap \\ (\exists \text{has-configuration.}((\leq 419 \text{ has-seats}) \sqcap (\leq 2 \text{ has-classes})) \sqcup \\ \exists \text{has-configuration.}((\leq 380 \text{ has-seats}) \sqcap (\leq 3 \text{ has-classes}))).$$

Note, that the description of the concept Airbus 340 is not an $\mathcal{AL}\mathcal{E}\mathcal{N}$ concept, but an $\mathcal{AL}\mathcal{C}\mathcal{N}$ -concept, since it contains a disjunction.

To illustrate how to work with SONIC, assume we want to find out the commonalities between the two Airbus concepts defined above. In general the commonalities of a set of concepts can be computed by employing the LCS inference. In our case this would not yield something interesting, since the LCS in $\mathcal{AL}\mathcal{C}$ of the concepts is simply the disjunction of the two Airbus concepts. By employing the LCS directly we would not learn anything about the commonalities of the two concepts. So, we have to translate the description of the Airbus 340 concept into $\mathcal{AL}\mathcal{E}\mathcal{N}$ by computing the Approximation of it. Subsequently, we want to compute the LCS of the approximated concept and the concept of the Airbus 300. Before we can use the inferences in SONIC, we have to make sure that the current ontology is classified.

4.2 How to compute an Approximation

The idea underlying Approximation is to translate a concept from one DL into a typically less expressive DL. Like in the case of our Airbus example, Ap-

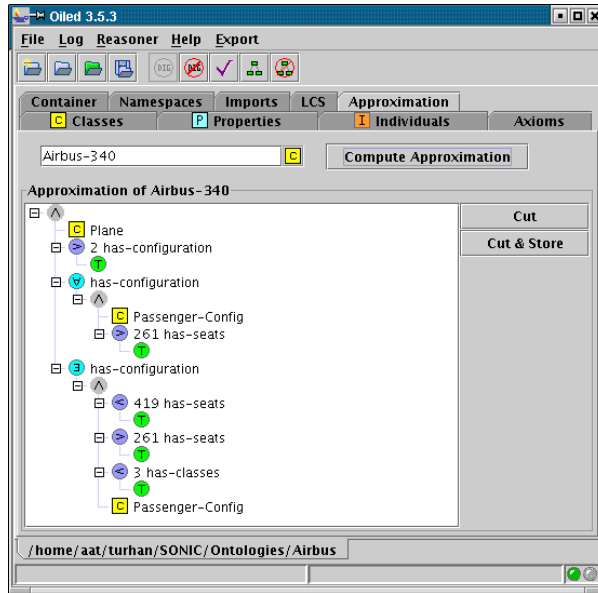


Figure 2: SONIC’s Approximation panel, with Approximation of the Airbus-340 concept.

proximation can be used to make non-standard inferences accessible to more expressive DLs so that at least an approximate solution can be computed, for more information on Approximation inference see [6].

If the ontology is classified, click on the panel called “Approximation” which is shown in Figure 2. In order to compute the approximation of the concept *Airbus-340* select a concept name from the ontology – in our case *Airbus-340* and click the button “Compute Approximation”. After a while the resulting concept is displayed as a syntax tree in the field below. Since the returned concepts can be quite long, it is convenient to close the syntax tree of subconcepts to inspect the rest of the concept. For our example we get the following Approximation of the concept *Airbus-340*:

$$\begin{aligned}
 & \text{Plane} \sqcap (\geq 2 \text{ has-configuration}) \sqcap \\
 & \forall \text{ has-configuration.}(\text{Passenger-Configuration} \sqcap (\geq 261 \text{ has-seats})) \sqcap \\
 & \exists \text{ has-configuration.}(\text{Passenger-Configuration} \sqcap (\geq 261 \text{ has-seats}) \sqcap \\
 & \quad (\leq 419 \text{ has-seats}) \sqcap (\leq 3 \text{ has-classes}))
 \end{aligned}$$

In this example the Approximation finds the commonalities of the two existential restrictions from the description of the concept *Airbus-340*. In Figure 2 you can see how SONIC displays the concept returned by the Approximation call. Next, we want to add this new concept to the ontology.

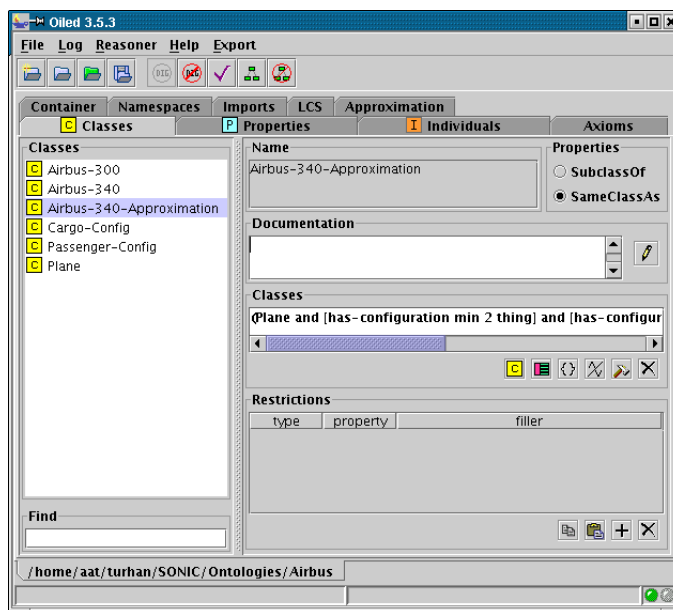


Figure 3: The new concept is added to the ontology.

4.3 How to process a result concept

SONIC offers the facility to delete parts of the obtained concept. You can mark a sub-concept and click the “Cut” button to remove a sub-concept. A concept or parts of a concept can be stored in the ontology by marking the desired (sub-)concept and click the “Cut & Store” button. Then a dialog comes up where you are asked to name the new concept. After the concept is added to the ontology, it can be processed on OILED’s “Classes” panel as usual.

In our example we want to store the obtained Approximation of the Airbus-340 in the ontology. After we have marked the concept, clicked the button and named the concept *Airbus-340-Approximation*, we turn to OILED’s “Classes” panel and declare that *Airbus-340-Approximation* is the *same class* as its description in the “Properties” field as it can be seen in Figure 3. Before we can compute the LCS of *Airbus-300* and *Airbus-340-Approximation*, we have to classify the ontology.

4.4 How to compute a LCS

Given a set of concepts \mathcal{M} the LCS is the concept that subsumes all the concepts from the set \mathcal{M} and is the least one (w.r.t. subsumption) with this property. Using the LCS one can generalize a set of concepts into one concept by extracting their commonalities. It has been argued in [2, 7] that the LCS facilitates a “bottom-up”-approach to the modeling task: a domain expert can select a number of intuitively related concepts already existing in an ontology and use

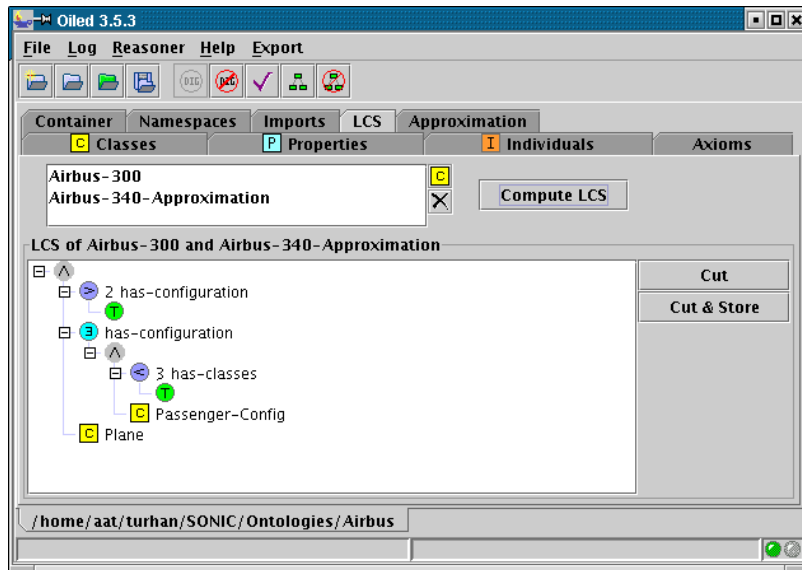


Figure 4: SONIC’s LCS panel – displaying the LCS from the example.

the LCS operation to automatically construct a new concept representing the closest generalization of them. SONIC implements the LCS for the DL \mathcal{ALCN} and its sub-languages.

In our example application we want to compute the LCS of *Airbus-300* and *Airbus-340-Approximation*. In order to do this we have to switch to the “LCS” panel, see Figure 4, and select the input concepts from the set of concept names mentioned in the ontology. It is possible to compute the LCS of a set of n concepts in SONIC, not just of two. After we have chosen the input concepts, we click the “Compute LCS” button and after a while the concept of the LCS is displayed in the same fashion as the results for Approximation (cf. Section 4.2). In our example we obtain for the LCS of *Airbus-300* and *Airbus-340-Approximation*:

$$\text{Plane} \sqcap (\geq 2 \text{ has-configuration}) \sqcap \\ \exists \text{has-configuration} . (\text{Passenger-Configuration} \sqcap (\leq 3 \text{ has-classes})).$$

We have made explicit the commonalities of the two concepts *Airbus-300* and the (Approximation of) *Airbus-340*: they are both planes with at least two configurations where one configuration is a passenger configuration with up to 3 classes. You can prune the obtained concept and add it to the ontology as a new concept in the same way as described in Section 4.3.

5 Future Work—What’s next?

SONIC is a prototype system that is currently under development. This system

will be improved and extended in several ways in the coming versions. One way to improve, for example, SONIC's user interface is to compute Approximation on demand when the LCS is invoked for DLs with disjunction. Furthermore, we want to improve the implementations of the inferences provided by SONIC in two ways. On the one hand we would like to speed up the Approximation by implementing known optimization techniques for Approximation, e.g. implement methods for so-called nice concepts. On the other hand we would like to reduce the size of the returned concepts of both inferences by implementing minimal rewriting. Moreover, the user interface of SONIC will be improved.

In the longer run SONIC will comprise more non-standard inferences, for example *matching* or the *good common subsumer* (GCS) inference. The latter allows to compute a common subsumer, but not necessarily the least one for more expressive DLs and ontologies which allow for GCIs. Furthermore, it is planned to extend SONIC to PROTEGÉ, which is another, more widely used ontology editor.

If you have suggestions to make how to improve SONIC or if you have errors to report, please do not hesitate to contact us.

References

- [1] F. Baader. Description logic terminology. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 485–495. Cambridge University Press, 2003.
- [2] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann, Los Altos.
- [3] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
- [4] S. Bechhofer. *OILED 3.4 Manual*. Information Management Group, Computer Science Department, University of Manchester, 2002. Available at <http://oiled.man.ac.uk/resources.shtml>.
- [5] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reasonable Ontology Editor for the Semantic Web. In *Proceedings of the 24th German Annual Conf. on Artificial Intelligence (KI'01)*, volume 2174 of *Lecture Notes in Artificial Intelligence*, pages 396–408, Vienna, Sep 2001. Springer-Verlag.
- [6] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on the Principles*

of Knowledge Representation and Reasoning (KR-02), San Francisco, CA, 2002. Morgan Kaufmann Publishers.

- [7] S. Brandt and A.-Y. Turhan. Using non-standard inferences in description logics — what does it buy me? In G. Görz, V. Haarslev, C. Lutz, and R. Möller, editors, *Proceedings of the 2001 Applications of Description Logic Workshop (ADL 2001)*, number 44 in CEUR Workshop, Vienna, Austria, September 2001. RWTH Aachen. See <http://CEUR-WS.org/Vol-44/>.
- [8] V. Haarslev and R. Möller. RACER system description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR-01)*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2001.
- [9] V. Haarslev, R. Möller, and M. Wessel. *RACER User's Guide and Manual*, Apr. 2004. Available from: <http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-19.pdf>.
- [10] A.-Y. Turhan and C. Kissig. SONIC — non-standard inferences go OILED. In D. Basin and M. Rusinowitch, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR-04)*, number 3097 in Lecture Notes in Computer Science, pages 321–325. Springer-Verlag, 2004.
- [11] A.-Y. Turhan and C. Kissig. SONIC—system description. In V. Haarslev and R. Möller, editors, *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, number 104 in CEUR Workshop, 2004. See <http://CEUR-WS.org/Vol-104/>.