Reasoner Prototype

Implementing new reasoner with datatypes support

Dmitry Tsarkov & Ian Horrocks
University of Manchester
Kilburn Building
Oxford Road
Manchester M13 9PL
email: tsarkov@cs.man.ac.uk

Identifier	Del 13
Class	Deliverable
Version	1.0
Date	25-09-2003
Status	Final
Distribution	Internal
Lead Partner	VUM

WonderWeb Project

This document forms part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2001-33052.

For further information about WonderWeb, please contact the project co-ordinator:

Ian Horrocks
The Victoria University of Manchester
Department of Computer Science
Kilburn Building
Oxford Road
Manchester M13 9PL
Tel: +44 161 275 6154
Fax: +44 161 275 6236
Email: wonderweb-info@lists.man.ac.uk

Contents

Executive Summary

This document provides an overview of the FaCT++ reasoner. FaCT++ is a Description Logic reasoner, which is a re-implementation of the FaCT system, extended with new features and optimisations. The current version supports the OWL Lite language; the full one will support the OWL DL language. The latest version of FaCT++ can be downloaded from the project website:

http://wonderweb.semanticweb.org/deliverables/D13.shtml

1 Introduction

The OWL language,[1] which is designed for representing Semantic Web ontologies, allows the user to introduce datatypes into ontologies (in addition to the usual mechanisms for describing "abstract" classes, such as boolean conjunctive, quantifiers and cardinality constraints). In order to use these ontologies one should have an efficient, robust reasoner, which supports all the features of the language. We have set out to satisfy these requirement by creating the FaCT++ DL reasoner.

FaCT++ was started as re-implementation of the well-known highly optimised DL reasoner FaCT [2]. It used the established FaCT algorithms, but with a different internal architecture. Additionally, the implementation language C++ was chosen in order to create a more efficient software tool, and to maximise portability. During the implementation process, new optimisations were also introduced, and some new features were added.

2 Overview of FaCT++

FaCT++ can be used for the following tasks:

- Checking the consistency of an ontology;
- Checking the satisfiability of a single concept / group of concepts;
- Checking the subsumption relationship between two concepts;
- Classifying the whole ontology (creating a taxonomy);
- Creating First-Order logic problems for tasks 1-3.

FaCT++ provides full reasoning support for the $\mathcal{SHIF}(\mathcal{D})$ Description Logic.[2] More precisely, it supports:

- Intersection, Union and Complementation of concepts;
- Universal and Existential restriction on roles;
- Transitive, Functional and Inverse roles;
- Integer and String datatypes.

In addition, FaCT++ can create FOL problems for subsumption and satisfiability checking in the $\mathcal{SHOIQ}$ logic using a standard syntax (TPTP) that can be read by most first order theorem provers.[3] See [4] for a comparison of the performance of the FaCT++ reasoner with a state of the art first order theorem prover running on TPTP problems created using FaCT++.

[1] http://www.w3.org/TR/owl-features/

[2] for a detailed syntax description see Appendix A

[3] see http://www.cs.miami.edu/ tptp/

Ontology	FaCT v2.33 time, sec	FaCT++ v0.8 time, sec
GALEN	49	790
GO	227	12
wine	—	11

Table 1: Comparison between FaCT and FaCT++

FaCT++ currently has limited support for individuals. All individuals are treated as concepts, and reasoning is performed on the modified ontology. This approach results in no loss of inferences if the ontology contains no relations between individuals, a restriction that holds for many of useful ontologies.

In addition to standard "definition" axioms, FaCT++ efficiently supports some other kinds of axiom which appear frequently in real ontologies, and which usually lead to a significant slowdown of the reasoning process:

- **General axioms** are introduced into ontologies quite frequently. FaCT++, like the original FaCT, uses the absorption technique to deal with general axioms efficiently.
- **Role domain and range axioms** are common in real-life ontologies. If the reasoning tool has no support of these constructions (as in FaCT), it must treat them as general axioms of a kind which are not absorbable. This slows the reasoning process significantly. In FaCT++, native support of these constructions was implemented, and this results in speedups of as much as two orders of magnitude for some ontologies.

3 Performance evaluation

3.1 Working Scenario

FaCT++ is currently available as a standalone software component, and the only mode available for version 0.8 is batch mode. In order to either perform classification/consistency checking of an ontology, or to test satisfiability/subsumption of given concept(s), users must create a configuration file. This file contains all necessary information about the input ontology, reasoning task(s) and configuration options for the reasoner.

The ontology is given in lisp-like language which is very similar to (but slightly differ from) the one used by FaCT. The underlying logic is $\mathcal{SHIF}(\mathcal{D})$. The user can either create the ontology manually, or can use the OilEd [1] ontology editor, exporting results in the appropriate format. Existing OWL ontologies can be translated into the FaCT++ format using the OWL Servlet[4] implemented by Sean Bechhofer and Raphael Volz.

3.2 Comparison with FaCT

Some results from a comparison between FaCT++ and FaCT can be found in Table 1.

[4] http://phoebus.cs.man.ac.uk:9999/OWL/Convertor

The GALEN ontology is a quite large (2800 classes) and complex ontology used in medicine. FaCT was optimised and tuned specifically to perform well on this particular ontology, and is still able to outperform FaCT++ in this test. This is mainly due to various caching optimisations, not all of which are currently implemented in FaCT++. These optimisations are currently being added to FaCT++, and when completed should allow FaCT++ to perform at least as well as FaCT on this ontology.

The Gene Ontology (GO) is large (>13500 classes and individuals), but has a very simple structure. There are no relations between individuals, so it is possible to use our individual-to-class translation without loss of correctness. This approach, together with the range of optimisations employed, makes FaCT++ at least 10 times faster than FaCT on this ontology (and on other large but simply structured ontologies).

The wine ontology is part of OWL test suite. It contains 9 roles with range and domain restrictions together with individuals and oneOf constructions (nominals). This makes it unsolvable by FaCT (it exceeds resource limits before completing the task), while FaCT++ can solve it in only a few seconds. This is mainly due to FaCT++'s native support for role range and domain restrictions, and improved absorption optimisations [3].

4 Obtaining FaCT++

FaCT++ is freely available under the GNU General Public License. The latest version of FaCT++ can be downloaded from the project website:

`http://wonderweb.semanticweb.org/deliverables/D13.shtml`

5 Discussion

FaCT provides highly optimised reasoning support for the OWL Lite language, and will eventually be extended to support OWL DL. It already outperforms FaCT by 1–2 orders of magnitude on several ontologies, and new features and optimisations are still being added.

References

[1] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able ontology editor for the semantic web. In *Proc. of the Joint German Austrian Conference on AI*, number 2174 in Lecture Notes In Artificial Intelligence, pages 396–408. Springer-Verlag, 2001.

[2] I. Horrocks. The FaCT system. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 307–312. Springer-Verlag, Berlin, May 1998.

[3] I. Horrocks. Implementation and optimisation techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. 2003.

[4] Dmitry Tsarkov and Ian Horrocks. DL reasoner vs. first-order prover. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, 2003. To appear.

A FaCT++ Input Language

The syntax of FaCT++ input language is presented in the Table 2. Here $C[N]$ if for class [name], $R[N]$ is for role [name], $I[N]$ is for individual [name], TN is for the datatype name, DN is for datarole name, DV is for data value.

B FaCT++ Optimisation Flags

All flags that affect FaCT++ behaviour are listed in the Table 3. There are name of option, it's type, quite informal description (which sometime contains possible values) and the default value.

Construction	syntax	comment
Maximal class	*TOP*	equal to owl:Thing
Minimal class	*BOTTOM*	equal to owl:Nothing
define primitive concept	(defprimconcept *CN* [*C*])	define $CN[\sqsubseteq C]$
define non-primitive concept	(defconcept *CN C*)	define $CN = C$
define concept subsumption	(implies_c C_1 C_2)	define $C_1 \sqsubseteq C_2$
define concept equality	(equal_c C_1 C_2)	define $C_1 = C_2$
define concept disjointness	(disjoint $C_1 \ldots C_n$)	classes $C_1, \ldots, C_n$ are disjoint
define role	(defprimrole *RN*)	
define role's transitivity	(transitive *RN*)	
define role's functionality	(functional *RN*)	
define role hierarchy	(implies_r R_1 R_2)	R_1 is a subrole of R_2
define role equality	(equal_r R_1 R_2)	R_1 is the same role as R_2
define role's domain	(domain *R C*)	add *C* to *R*'s domain
define role's range	(range *R C*)	add *C* to *R*'s range
define individual	(instance *IN C*)	*IN* is an instance of *C*
define individual equality	(same $IN_1 \ldots IN_n$)	individuals $IN_1 \ldots IN_n$ are the same
define individual inequality	(different $IN_1 \ldots IN_n$)	individuals $IN_1 \ldots IN_n$ are different
define individual's relation	(related IN_1 *R* IN_2)	$R(IN_1, IN_2)$ holds
define user's data type	(datatype *TN*)	*TN* is a name of user defined type
define data role	(datarole *DN*)	*DN* is ranged on a datatype
define data role's range	(range *DN TN*)	*DN* is ranged on a *TN*
intersection of classes	(and $C_1 \ldots C_n$)	the following returns *C*
union of classes	(or $C_1 \ldots C_n$)	
negation of class	(not *C*)	
existential restriction	(some *R* [*C*])	
universal restriction	(all *R C*)	
[qualified] number restriction	(at-least *n R* [*C*])	
[qualified] number restriction	(at-most *n R* [*C*])	
set of individuals	(one-of $IN_1 \ldots IN_n$)	
existential data restriction	(some *DR* [*DN*])	data type name
existential data restriction	(some *DR DV*)	data value
universal data restriction	(all *DR* [*DN*])	data type name
universal data restriction	(all *DR DV*)	data value
inversion of role	(inv *R*)	returns *R*

Table 2: FaCT++ Syntax

Name	Type	Description	Default
useTG	bool	Option 'useTG' is of internal use only. Default value is true. User should not change it	true
useAllNames	bool	Option 'useAllNames' determine whether non-primitive named concepts should exist in Completion Tree labels or not. If true, all concept names add into labels	true
useRelevantOnly	bool	Option 'useRelevantOnly' is used when creating internal DAG representation for externally given TBox. If true, DAG contains only concepts, relevant to query. It is safe to leave this option false	false
useUndefinedConcept	bool	Option 'useUndefinedConcept', set to 'true', allows forward references to named concepts in TBox. Set this option to false if you want to check additional correctness of your hand-made KB	true
useAbsorption	bool	Option 'useAbsorption' switch on and off absorption process for general axioms. This option is of internal use only. It is crucial for reasoning performance to leave this option true	true
orSortBy	text	Option 'orSortBy' define the sorting mode of OR vertices in the DAG. Possible values are 'depth', 'size' and 'freq'. All other strings leave OR unsorted. See also 'orSortOrder' option	depth
orSortOrder	text	Option 'orSortOrder' define the sorting order of OR vertices in the DAG. Possible values are 'up' and 'down'. See also 'orSortBy' option	up
IAOEFLG	text	Option 'IAOEFLG' define the priorities of different operations in TODO list. Possible values are 7-digit strings where the only possible digits are 0-6. The digits in places 1, 2, ..., 7 are for priority of Id, And, Or, Exists, Forall, LE and GE operations respectively. The smaller number means the higher priority. All other constructions (TOP, BOTTOM, etc) has priority 0	1236045
useSemanticBranching	bool	Option 'useSemanticBranching' switch semantic branching on and off. The usage of semantic branching usually leads to faster reasoning, but sometime could give small overhead	true
useBackjumping	bool	Option 'useBackjumping' switch backjumping on and off. The usage of backjumping usually leads to much faster reasoning	true
useCompletelyDefined	bool	Option 'useCompletelyDefined' leads to simpler Taxonomy creation if TBox contains no non-primitive concepts. Unfortunately, it is quite a rare case	false

Table 3: FaCT++ Option Set