Optimizing Tableau Reasoning in $\mathcal{ALC}$ Extended with Uncertainty

Volker Haarslev, Hsueh-Ieng Pai, and Nematollaah Shiri

Concordia University
Dept. of Computer Science & Software Engineering
Montreal, Quebec, Canada
{haarslev, hsueh_pa, shiri}@cse.concordia.ca

Abstract. There has been an increased interest in recent years to incorporate uncertainty in Description Logics (DLs), and a number of proposals have been put forward for modeling uncertainty in DL frameworks. While much progress has been made on syntax, semantics, and query processing issues, optimizing queries in this context has received little attention. In this paper, we study query processing for a tableau-based DL framework with uncertainty and focus on optimization of resolution of certainty inequality constraints, obtained from a translation in query processing phase. We develop a running prototype which evaluates DL knowledge bases with ABoxes and TBoxes annotated with uncertainty parameters and computes the corresponding semantics encoded as a set of constraints in the form of linear and/or nonlinear inequations. We also explore various existing and new opportunities for optimizing the reasoning procedure in this context. Our experimental evaluation indicates that the optimization techniques we considered result in improved efficiency significantly.

1 Introduction

Uncertainty is a form of imperfection commonly found in the real-world information, and refers to situations where the truth of such information is not established definitely. Despite of recent advances on extending Description Logics (DLs) with various forms of uncertainty (such as vagueness or probability), there is generally a lack of effort in studying optimization aspects of uncertainty reasoning. This paper is the first step in this direction.

This work is a continuation of our previous theoretical work on extending the DL fragment $\mathcal{ALC}$ with various forms of uncertainty [4–6] in which we abstract away the notion of uncertainty in the description language, the knowledge base, and the reasoning services, and we encode their corresponding semantics as a set of constraints in the form of linear and/or nonlinear inequations. In this paper, we explore various opportunities for optimizing the tableau-based reasoning procedure for the prototype of our generic framework called GURDL – a Generic Uncertainty Reasoner for the DL $\mathcal{ALC}$.

The rest of this paper is organized as follows. Section 2 provides a brief overview of our generic framework for DL with uncertainty. We also review the

existing tools that are available in this area. In Section 3, we present some optimization techniques that are implemented in GURDL, while Section 4 reports our performance evaluation results. Finally, we conclude in Section 5 with some directions for future work.

2 Related Work

In this section, we first give a brief overview of our generic framework for DL with uncertainty. We then survey the existing tools that are available for reasoning with DL and uncertainty.

2.1 Generic Framework for DL with Uncertainty

As mentioned in [6], existing extensions of DLs with uncertainty can be classified into one of the three approaches according to the underlying mathematical foundation and the type of uncertainty they model: (1) the fuzzy approach (such as [9, 10]), based on fuzzy set theory, essentially deals with the vagueness in the knowledge; (2) the probabilistic approach (such as [1, 3, 8]), based on the classical probability theory, deals with the uncertainty due to lack of knowledge; (3) the possibilistic approach [7], based on possibility theory, allows necessity and possibility measures to be handled in the same formalism.

In order to support the various forms of uncertainty within the same framework, we abstracted away the notion of uncertainty (fuzzy logic, probability, possibilistic logic) and proposed a generic framework for DL with uncertainty [5]. In particular, our generic framework consists of three components:

1. *Description Language with Uncertainty:* In our framework, we keep the syntax of the description language identical to that of the classical $\mathcal{ALC}$, while extending the corresponding semantics with uncertainty. In order to flexibly represent various forms of uncertainty, we assume that certainty values form a complete lattice $\mathcal{L} = \langle \mathcal{V}, \preceq \rangle$, where $\mathcal{V}$ is the certainty domain, and $\preceq$ is the partial order defined on $\mathcal{V}$. We also use b to denote the least element in $\mathcal{V}$, t for the greatest element in $\mathcal{V}$, $\oplus$ for the join operator in $\mathcal{L}$, $\otimes$ for its meet operator, and $\sim$ for the negation operator.
   The semantics of the description language is based on the notion of an interpretation, where an interpretation $\mathcal{I}$ is defined as a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain and $\cdot^{\mathcal{I}}$ is an interpretation function. For example, if individual $John \in \Delta^{\mathcal{I}}$, then $Obese^{\mathcal{I}}(John)$ gives the certainty that $John$ belongs to concept $Obese$. The syntax and the semantics of the description language supported in our framework are summarized in Table 1. Note that f_c and f_d in the table denote conjunction and disjunction functions. They are used to specify how one should interpret a given description language. For example, in the fuzzy approach, we would have the min function as f_c and max function as f_d, whereas in a probabilistic approach, we might have algebraic product ($prod(\alpha, \beta) = \alpha\beta$) as f_c, and the independent function ($ind(\alpha, \beta) = \alpha + \beta - \alpha\beta$) as f_d.

Name	**Syntax**	**Semantics** ($a \in \Delta^{\mathcal{I}}$)
Atomic Concept	A	$A^{\mathcal{I}}(a) = CF_C$, with $CF_C : \Delta^{\mathcal{I}} \rightarrow \mathcal{V}$
Atomic Role	R	$R^{\mathcal{I}}(a,b) = CF_R$, with $CF_R : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow \mathcal{V}$
Top Concept	$\top$	$\top^{\mathcal{I}}(a) = t$
Bottom Concept	$\bot$	$\bot^{\mathcal{I}}(a) = b$
Concept Negation	$\neg C$	$(\neg C)^{\mathcal{I}}(a) = \sim C^{\mathcal{I}}(a)$
Concept Conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}}(a) = f_c(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$
Concept Disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}}(a) = f_d(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$
Role Exists Restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}}(a) = \oplus_{b \in \Delta^{\mathcal{I}}} \{f_c(R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\}$
Role Value Restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}}(a) = \otimes_{b \in \Delta^{\mathcal{I}}} \{f_d(\sim R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\}$

Table 1. Syntax/Semantics of the Description Language Supported

2. *Knowledge Bases with Uncertainty:* As usual, the knowledge base (Σ) consists of both the TBox and the ABox. However, unlike the classical case, each axiom and assertion is associated with a certainty value, as well as the conjunction/disjunction functions used to interpret the concept descriptions. More specifically, the TBox includes a set of terminological axioms that could be concept subsumptions $\langle C \sqsubseteq D, \alpha \rangle \langle f_c, f_d \rangle$ and/or concept definitions $\langle C \equiv D, \alpha \rangle \langle f_c, f_d \rangle$, where C and D are concept descriptions, $\alpha \in \mathcal{V}$ is the certainty that the axiom holds, and f_c and f_d are the conjunction and disjunction functions. As an example, the certainty of the axiom $\langle \mathit{Rich} \sqsubseteq ((\exists \mathit{owns}.\mathit{ExpensiveCar} \sqcup \exists \mathit{owns}.\mathit{Airplane}) \sqcap \mathit{Golfer}), [0.8,\ 1] \rangle \langle \mathit{min}, \mathit{max} \rangle$ is at least 0.8, with all the concept conjunctions interpreted using min, and all the concept disjunctions interpreted using max. Note that, although our framework supports simple probabilities such as independent or mutually exclusive events, we are investigating ways to model knowledge base with more general probability theory such as conditional independence, since reasoning with it requires extra information about the events and facts in the world (Σ).
   The ABox in our framework consists of a set of concept assertions of the form $\langle a : C, \alpha \rangle \langle f_c, f_d \rangle$ or role assertions $\langle (a,b) : R, \alpha \rangle \langle -, - \rangle$, where a, b are individuals, C is a concept, R is a role, $\alpha \in \mathcal{V}$, f_c is the conjunction function, f_d is the disjunction function, and $-$ denotes that the corresponding function is not applicable. For instance, the assertion "Mary is tall and thin with a degree between 0.6 and 0.8" can be expressed as $\langle \mathit{Mary} : \mathit{Tall} \sqcap \mathit{Thin}, [0.6, 0.8] \rangle \langle \mathit{min}, - \rangle$. Here, min is used as the conjunction function, and the disjunction function is not necessary since there is no concept disjunction here.
3. *Reasoning with Uncertainty:* The inference problems supported by our framework include the satisfiability problem and the entailment problem, where the former checks if an admissible knowledge base is satisfiable and the later determines the degree with which an assertion is true given the knowledge base. Similar to the classical DL reasoning, pre-processing steps are first applied to abstract the TBox. Then, completion rules are applied to simplify the ABox, and blocking is introduced to ensure termination [4]. However, unlike the classical case, each rule application generates a set of derived assertions and a set of constraints in the form of linear/nonlinear inequations

which encode the semantics of the assertion. For example, given the assertion $\langle Mary : Tall \sqcap Thin, [0.6, 0.8]\rangle\langle min, -\rangle$, the conjunction rule can be applied, which yields $\langle Mary : Tall,\ x_{Mary:Tall}\rangle\langle -, -\rangle$ and $\langle Mary : Thin, x_{Mary:Thin}\rangle$ $\langle -, -\rangle$, and the constraint $(min(x_{Mary:Tall},\ x_{Mary:Thin}) = [0.6, 0.8])$, where $x_{Mary:Tall}$ (resp., $x_{Mary:Thin}$) is the variable representing the certainty that *Mary* is $Tall$ (resp., *Thin*). The completion rules we introduced in [4] are applied in arbitrary order until either the extended ABox contains a clash or no further rule could be applied. If a clash is encountered (such as an assertion has two conflicting certainty values), the knowledge base is unsatisfiable. Otherwise, a constraint solver is called to solve/optimize the system of inequations to check satisfiability of the knowledge base or the degree with which an assertion is true. Details and the proof for the correctness of the reasoning procedure can be found in [4].

2.2 Existing Tools for DL with Uncertainty

To the best of our knowledge, the only DL/uncertainty reasoner that is publicly available is fuzzyDL [2], which inspired our preliminary prototype. As the name suggests, fuzzyDL supports only fuzzy knowledge (i.e., it cannot handle other uncertainty formalisms such as probabilities). Although fuzzyDL supports two types of fuzzy knowledge – those with Zadeh semantics and those with Lukasiewicz logic, it uses two sets of completion rules instead of using a generic set of inference rules to deal with different semantics. Nevertheless, fuzzyDL has some interesting features. For example, it supports concept modifiers and a more expressive fragment of DL $\mathcal{SHIF}$.

3 Optimization Techniques Employed in GURDL

GURDL is the prototype of our generic framework for DL with uncertainty. A number of optimization techniques have been incorporated in GURDL. Due to the limited space, we discuss only some of them here. The idea is to investigate whether some existing optimization techniques used in classical DL systems could be applied to the uncertainty case (including lexical normalization, concept simplification, partition based on connectivity as Individual Groups, and caching), while exploring new optimization technique that is specific to deal with uncertainty (partition based on connectivity as Assertion Groups).

3.1 Lexical Normalization

Lexical normalization is a common optimization technique used in classical DL systems, where concepts are transformed into a canonical form. For example, concepts like $(C \sqcap (B \sqcap A))$, $(B \sqcap (C \sqcap A))$, and $((B \sqcap A) \sqcap C)$ can all be transformed into the canonical form $(A \sqcap (B \sqcap C))$. In GURDL, lexical normalization is realized by sorting the sub-concepts in the concept description. The major advantage of lexical normalization is that it allows obvious clashes be detected early. For example, given the assertions $\langle Mary : Tall \sqcap Thin, [0.8, 1]\rangle$ and

$\langle Mary : Thin \sqcap Tall, [0, 0.4]\rangle$, the second assertion becomes $\langle Mary : Tall \sqcap Thin$, $[0, 0.4]\rangle$ after the normalization. This allows us to easily notice the inconsistency between the two assertions, which would be hard to detect otherwise. Another advantage of lexical normalization is that it facilitates concept simplification.

3.2 Concept Simplification

Concept simplification is another optimization technique that is commonly employed in classical DL systems, done by removing redundant sub-concepts in a given concept. In GURDL, the following simplifications are applied:

$$\top \sqcap C \rightsquigarrow C \qquad \top \sqcup \cdots \rightsquigarrow \top \qquad \forall R.\top \rightsquigarrow \top$$
$$\bot \sqcap \cdots \rightsquigarrow \bot \qquad \bot \sqcup C \rightsquigarrow C \qquad \exists R.\bot \rightsquigarrow \bot$$

The above simplifications are valid due to the boundary-condition properties of the combination functions [5]. Note that simplification must be applied with care when uncertainty is introduced. For example, it is a common practice in classical DL systems to remove duplicated sub-concepts in a concept conjunction or disjunction, such as simplifying $(A \sqcap A)$ to A. However, such simplification is not valid once uncertainty is introduced. For example, assume that the interpretation of concept A is 0.4. If the conjunction function is min, then $(A \sqcap A)$ is A since $min(0.4, 0.4) = 0.4$. However, if the conjunction function is the algebraic product $(\times)$, then $(A \sqcap A)$ is not the same as A, since $\times(0.4, 0.4) = 0.16 \neq 0.4$. Therefore, such simplification is invalid, hence cannot be applied.

The major advantage of the simplification method is that it could potentially reduce the number of sub-concepts in a concept description, hence reducing the number of completion rule applications. In some extreme case, a complicated concept description can be simplified to only $\top$ or $\bot$, hence eliminating the need to apply the completion rule.

3.3 Partition Based on Connectivity

In GURDL, the ABox is partitioned into Individual Groups (IGs) and Assertions Groups (AGs) based on the notion of connectivity.

Individual Groups (IGs) Similar to the classical DL systems, the individuals in the ABox are divided into one or more partitions called Individual Groups. Each group consists of individuals that are "related" to each other through role assertions. By partitioning the ABox this way, inferences can be performed independently for each IG. Once no more completion rule can be applied to a given IG, we could pass the derived assertions and their corresponding constraints to the constraint solver to build the model, and we can be sure that the model built will not be changed even if we perform inference on other IGs in the ABox. This allows the consistency of the ABox be checked incrementally and hence reduces the reasoning complexity when the knowledge base includes many individuals which could be partitioned as described. This also allows us to check the consistency of the ABox related to one particular individual without checking the consistency of the complete ABox.

Assertion Groups (AGs) As mentioned in Section 2.1, the reasoning procedure for our uncertainty framework differs from the classical one because, in addition to derive assertions, a set of constraints in the form of linear/nonlinear inequations is also generated, which is later on feed into the constraint solver to check for its consistency. Since the number of constraints generated is usually large, it is important to optimize the constraint solving process.

In GURDL, each IG is partitioned into one or more independent subsets called Assertion Groups. In general, two assertions A_1 and A_2 are in the same AG if A_1 is directly or indirectly inferred from A_2 (through the application of completion rules), or A_1 and A_2 differ only in terms of their certainty values and/or conjunction and disjunction functions. The interesting property about this partition is that, when we union all the constraints (resp., variables associated with the constraints) in the AGs that belong to a particular IG, we obtain all the constraints (resp., variables associated with the constraints) in that IG. On the other hand, if we take the intersection, we obtain an empty set. This implies that constraints in each AG can be solved independently, while assuring that the model built will not be changed when we solve constraints in other AGs.

This has several advantages. First, the consistency of the IG can be checked incrementally. At any given time, the constraints in one single AG are fed into the constraint solver. If any AG is found to be inconsistent, this implies that the whole IG is inconsistent. A related advantage is that, in case an IG is inconsistent, the reasoner will be able to more precisely identify the assertions that cause the inconsistency. Another advantage is that we are now able to determine the degree to which a particular assertion (say, X) is true by simply solving the constraints in the AG that X belongs. Finally, since the number of constraints (and the variables used in the constraints) in one single AG is, in general, no more than those of the whole IG, the speed of solving a few small constraint sets would be faster than solving one large constraint set. The performance evaluation of AG-partitioning is studied in Section 4.

3.4 Caching

To save the reasoner from doing redundant/repeated work, each assertion and constraint is stored only once. A flag is set to indicate whether completion rules have been applied to a given assertion (resp., IG). In addition, after the constraints in an AG are solved, the result is cached for later use.

4 Performance Evaluation

In this section, we study the performance of GURDL. All the experiments were conducted under Windows XP on a Pentium 2.40 GHz computer with 3.25 GB of RAM. Due to the limited space, we present only highlights of our results here.

Table 2 lists a few test cases, the number of concept assertions (C) in each test case, the number of role assertions (R), the number of axioms with necessary condition (N), the number of axioms with concept definitions (D), the functions

Test Case	C	R	N	D	F	$\mathcal{V}$	IG	AG	W	H	L	I	S	O	Total
1. Classical	15	2	5	0	*min/max*	{0, 1}	3	84	25	6	0.014	0.10	2.19	0.22	2.52
2. Min-Max	15	2	5	0	*min/max*	[0, 1]	3	84	25	6	0.015	0.10	2.47	0.19	2.78
3. Mixed	15	2	5	0	*mixed*	[0, 1]	3	159	44	6	0.016	0.17	12.92	0.32	13.43
4. Min-Max/Def.	15	2	0	5	*min/max*	[0, 1]	3	13	58	6	0.016	0.54	21.36	0.29	22.20
5. University	1	0	47	6	*min/max*	{0, 1}	1	231	45	5	0.020	1.25	17.15	0.75	19.17

Table 2. Performance of test cases (in seconds)

used to interpret the concept description (F), the certainty domain ($\mathcal{V}$), the number of IGs (IG), the number of AGs (AG), the width of the ABox (W), the height of the ABox (H), the time to load the knowledge base (L), the time to apply the inference rules (I), the time to solve constraints (S), other time (mostly I/O) (O), and the total time for ABox consistency checking (L + I + S + O). All the time measures are in seconds.

As shown in the table, the time spent on solving constraints (S) dominates the overall reasoning time (Total) for all the test cases. Note also that test cases 1 and 2 differ by the certainty domain, but this has limited effect on the performance. Test cases 2 and 3 differ by the functions used to interpret the description language (F). We can see that it takes longer to solve constraints that include a mix of nonlinear functions (*prod*, *ind*) and simple ones (*min*, *max*). Test cases 4 illustrates that it takes longer when we have axioms with concept definitions (D) instead of those with necessary conditions (N). Test case 5 shows the case where an IG is partitioned into many AGs.

Note that our prototype runs slower than the classical reasoners for standard knowledge bases, where we use $\{0, 1\}$ for the certainty domain, and *min* and *max* for conjunction and disjunction functions (one or two seconds vs. many seconds). This was expected, partly because standard reasoners implement many more optimization techniques, some of which we could not use in our context. Also, unlike in our context, they do not need to rely on constraint solvers as part of their reasoning process. Note also that we have not compared the performance with fuzzyDL here, because fuzzyDL uses a different constraint solver than GURDL, and the effect of such factor is not negligible.

Table 3 compares the total time for solving constraints when we partition the ABox into AGs, IGs, or no partition at all (ALL). Note that when we partition the ABox into AGs, the performance is the best. Note also that for the test case University, when the ABox is not partitioned into AGs, the constraint set is simply too large for the constraint solver to handle (we get stack overflow error). This shows the importance of keeping the constraint set as small as possible by partitioning the ABox.

5 Conclusion and Future Work

In this paper, we have explored various existing and new optimizing techniques for the reasoning procedure of our generic framework for DL with uncertainty, for which we have incorporated in our prototype. Due to the space limit, we present

Test Case	AG	IG	ALL	Gain1	Gain2	Gain3
1. Classical	2.52	3.42	4.84	26.31%	29.37%	47.95%
2. Min-Max	2.78	4.15	6.17	32.99%	32.68%	54.88%
3. Mixed	13.43	25.54	28.99	47.43%	11.90%	53.69%
4. Min-Max/Def.	22.20	37.24	100.58	40.37%	62.98%	77.93%
5. University	19.17	N/A	N/A	N/A	N/A	N/A

Table 3. Performance evaluation for partition based on connectivity (in seconds) (Gain1: AG vs. IG, Gain2: IG vs. ALL, Gain3: AG vs. ALL)

the partial performance evaluation result, which shows that the optimization techniques we employed are effective. As future research, we plan to extend the generic framework to a more expressive portion of DL. We also plan to optimize the reasoning procedure further. For example, since constraint-solving is the phase that takes the longest time, in case we have multiple AGs, we could solve them concurrently by running multiple threads on different computers. Another optimization would be to reduce the number of constraints or the number of variables in the constraints generated during the reasoning procedure. These methods are expected to greatly enhance the performance.

Acknowledgement. This work was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada, and by ENCS Concordia University. We also thank anonymous reviewers for their helpful comments.

References

1. Ding, Z., Peng, Y., and Pan, R. A Bayesian approach to uncertainty modeling in OWL ontology. In *Proc. of AISTA*, Luxembourg, 2004.
2. fuzzyDL. http://gaia.isti.cnr.it/ straccia/software/fuzzyDL/fuzzyDL.html.
3. Giugno, R. and Lukasiewicz, T. P-$\mathcal{SHOQ}$(D): A probabilistic extension of $\mathcal{SHOQ}$(D) for probabilistic ontologies in the Semantic Web. In *Proc. of JELIA*, pages 86–97, London, 2002. Springer-Verlag.
4. Haarslev, V., Pai, H.I., and Shiri, N. A formal framework for description logics with uncertainty. *Submitted to International Journal of Automate Reasoning.*
5. Haarslev, V., Pai, H.I., and Shiri, N. Completion rules for uncertainty reasoning with the description logic $\mathcal{ALC}$. In *Proc. of CSWWS*, pages 205–225, Quebec City, 2006. Springer Verlag.
6. Haarslev, V., Pai, H.I., and Shiri, N. Uncertainty reasoning in description logics: A generic approach. In *Proc. of FLAIRS*, pages 818–823, Florida, 2006. AAAI Press.
7. Hollunder, B. An alternative proof method for possibilistic logic and its application to terminological logics. In *Proc. of UAI*, pages 327–335. Morgan Kaufmann, 1994.
8. Koller, D., Levy, A. Y., and Pfeffer, A. P-CLASSIC: A tractable probablistic description logic. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 390–397, Providence, Rhode Island, July 1997. AAAI Press.
9. Straccia, U. Description logics with fuzzy concrete domains. In *Proc. of UAI*, 2005.
10. Tresp, C. and Molitor, R. A description logic for vague knowledge. In *Proc. of ECAI*, pages 361–365, Brighton, UK, 1998. John Wiley and Sons.