Declarative Knowledge Processing
Lecture 5: Complexity of reasoning in $\mathcal{ALC}$

Magdalena Ortiz
Knowledge Base Systems Group
Institute of Information Systems
ortiz@kr.tuwien.ac.at

3 November 2010

Complexity of reasoning in $\mathcal{ALC}$

Recommended Resources

- The DL Complexity Navigator contains links to dozens of complexity results
  http://www.cs.man.ac.uk/~ezolin/dl/

  Emphasis on different DLs, only fragments are really related to today’s lecture

Complexity of Reasoning

As we have mentioned, the tableau algorithm is not worst-case optimal

Apart from devising ‘practicable’ algorithms, we are also interested in understanding the expressiveness of $\mathcal{ALC}$ and the real complexity of the relevant reasoning problems

We now look briefly at the computational complexity of reasoning in $\mathcal{ALC}$

Outline

1. The complexity of concept satisfiability

2. The complexity of Knowledge Base satisfiability
   2.1 An ExpTime algorithm for reasoning in $\mathcal{ALC}$
   2.2 ExpTime-hardness (intuition only)

3. Summary
Complexity of reasoning in ALC 1. Concept satisfiability

Concept satisfiability in ALC

Recall:

Theorem

Deciding satisfiability of ALC concepts is PSpace-complete.

Proof. [Membership] For every ALC concept \( C \), it is possible to obtain in linear time a formula \( \varphi_C \) in multi-modal K such that \( \varphi_C \) is satisfiable iff \( C \) is satisfiable. Since formula satisfiability in multi-modal K can be decided in polynomial space, then ALC concept satisfiability can also be decided in polynomial space.

[Hardness] For every formula \( \varphi \) in multi-modal K it is possible to obtain in linear time an ALC concept \( C_\varphi \) such that \( C_\varphi \) is satisfiable iff \( \varphi \) is satisfiable. Since formula satisfiability in multi-modal K is PSpace-hard, then so is satisfiability of ALC concepts.

\( \square \)

PSpace completeness extends to ALC concept subsumption.

Reasoning w.r.t. Acyclic TBoxes

Lemma

Given an ALC KB \( K = (T,A) \) where \( T \) is acyclic, deciding satisfiability of \( K \) is PSpace-complete.

- PSpace hardness follows directly.
- It is only necessary to show that satisfiability of \( K \) can be decided in polynomial space. Intuitively:
  - Recall our (informal) PSpace argument for the tableau for concepts
  - If the TBox is acyclic, a tableau algorithm only needs to ‘unfold’ the definitions of the concepts in the current labels
  - This will always lead to concepts of smaller ‘implicit’ quantifier depth, and thus ensure that branches have polynomial depth
- Note: satisfiability w.r.t. acyclic TBoxes can be reduced to plain concept satisfiability, but the resulting concept may be exponentially larger

Reasoning w.r.t. Acyclic TBoxes (cont’d)

- In the presence of acyclic TBoxes, PSpace completeness also holds for concept subsumption and instance checking
  - the reductions preserve cyclicity
- However, reasoning gets harder if we consider arbitrary TBoxes!
ExpTime-completeness of ALC

**Theorem**

Deciding satisfiability of ALC knowledge bases is ExpTime-complete.

This means that:

1. Deciding satisfiability of ALC knowledge bases is ExpTime-hard.
   - Any correct algorithm will need exponential time to terminate, at least in some cases,
   + but we can solve some problems that are so hard that they can not be solved in polynomial time.
2. Deciding satisfiability of ALC knowledge bases is in ExpTime.
   + There is an algorithm that only needs exponential time,
   - but ALC can not express problems that are, for example, 2-ExpTime-hard.

ExpTime-completeness extends to the other mentioned reasoning tasks w.r.t. TBoxes and KBs.

**ExpTime-completeness of ALC (cont’d)**

How do we show this theorem?

One needs to show:

- **hardness (lower bound)** Deciding satisfiability of ALC knowledge bases is ExpTime-hard.
- **membership (upper bound)** Deciding satisfiability of ALC knowledge bases is in ExpTime.

- We will show membership, for a simplified case
  - No ABoxes, we consider the satisfiability of a TBox only
  - and briefly discuss the hardness, without giving a formal proof.

Memberhp

**Lemma**

Deciding satisfiability of an ALC TBox is in ExpTime.

To show the lemma, we show that there exists an algorithm such that:

- It takes an ALC TBox $T$ as an input.
- It always terminates, and answers ‘satisfiable’ or ‘unsatisfiable’
- If it answers ‘satisfiable’, then there exists a model of $T$ (i.e. the algorithm is sound)
- If there exists a model of $T$, then it answers ‘satisfiable’ (i.e. the algorithm is complete)
- It terminates in time $O(2^{p(T)})$ for some polynomial function $p(T)$.

A Type Elimination Algorithm

Our algorithm is based on type elimination

- Let $C_T = \prod_{C \subseteq T} \text{NFF}(\neg C \sqcup D)$ be defined as usual
- sub($T$) contains $C_T$ and is closed under subconcepts and their negations in NNF

**Definition (Type)**

A $T$-type is a set $\tau \subseteq \text{sub}(T)$ that satisfies:

- $C \in \tau$ iff $\text{NFF}(\neg C) \notin \tau$, for all $C \in \text{sub}(K)$
- if $C \sqcap D \in \tau$, then $C \in \tau$ and $D \in \tau$,
- if $C \sqcup D \in \tau$, then $C \in \tau$ or $D \in \tau$, and
- $C_T \in \tau$. 
A Type Elimination Algorithm (cont’d)

Roughly, models of $T$ are composed of types.

To decide the existence of a model, we:

- Generate all types
  - we can do it because there are only exponentially many
- Eliminate the ones that cannot occur in the models of $T$
- At the end, check whether the set of remaining types is empty
- If it is not empty, then there is a model of $T$, and the algorithm answers ‘satisfiable’
- If it is empty, then the algorithm answers ‘unsatisfiable’

Good and bad types

Which types can not occur in the models of $T$?

- Let $T$ be a set of types.
- We say that a type $\tau$ is good in $T$, if for every $\exists R. C \in \tau$, there is some $\tau' \in T$ such that
  - $C \in \tau'$ and
  - $\{D \mid \forall R. D \in \tau\} \subseteq \tau'$
- Intuitively, we can find suitable ‘successors’ for $\tau$ in $T$
- A type is bad in $T$ if it is not good in $T$
- A bad type contains some existential restriction that we cannot satisfy using the types in $T$

Type Elimination Algorithm

- First, we compute the set $T_0$ of all $T$-types
- We repeatedly compute $T_{i+1}$ from $T_i$, until $T_{i+1} = T_i$
  \[
  T_{i+1} = \{\tau \in T_i \mid \tau \text{ is good in } T_i\}
  \]
- If the final $T_\omega$ is not empty, then return ‘satisfiable’
- Otherwise, return ‘unsatisfiable’

To decide whether a concept $C$ is satisfiable w.r.t. to a TBox $T$, simply run the algorithm as above, but return ‘satisfiable’ if the final $T_\omega$ contains some type $\tau$ with $C \in \tau$, and ‘unsatisfiable’ otherwise

Correctness and Complexity

- The algorithm terminates in $O(2^{2^{|\text{sub}(T)|}})$ steps, and $\text{sub}(T)$ is linear in $\mathcal{K}$
- The algorithm returns ‘satisfiable’ iff $\mathcal{K}$ is satisfiable
  - If all type in a set are good, and one contains $C_0$, then we can build a tree model of $T$ whose root satisfies $C_0$
  - If we take a model of $\mathcal{K}$ and break it up into small pieces, we obtain a set of good types where one contains $C_0$
Comparing with Tableaux

Some similarities are clear:
- the $\cap$, $\cup$, and $T$ rules are captured by the definition of type
- the $\exists$ and $\forall$ rules are reflected in the notion of good in $T$
- absence of clashes is also reflected in definition of type

In the worst-case, the tableau algorithm may need more than double exponential time, while type elimination needs only single exponential!

But what about the best case? and average cases?

ExpTime-hardness

- To formally prove this, one needs to give a (polynomial) reduction from an ExpTime-hard problem to KB satisfiability in $\mathcal{ALC}$.
- For example, one could reduce some problem like:
  - The word problem for a deterministic Turing machine that runs in single exponential time
  - The word problem for an alternating Turing machine that runs in polynomial space (easier to encode)
  - The succinct version of the Graph-accessibility problem (see proof by Donini in the DL handbook)
- We do not do any such reduction here, and only discuss some informal intuitions

ExpTime-hardness (cont’d)

- Similarly as for $\mathcal{ALC}$ concepts, the graph representation of the model of an $\mathcal{ALC}$ KB is a forest with exponentially many branches
- But in the presence of GCIs, the depth of the relevant concepts need not decrease along the branches
- Branches may be infinite, and some kind of cycle detection is required
- In general, a cycle is only enforced after using exponential space
- This makes reasoning ExpTime hard
- It is not hard to write an $\mathcal{ALC}$ KB whose models simulate an exponential counter (this is important for the mentioned encodings)
- ExpTime-hardness holds already for deciding the existence of a model of a TBox (or concept satisfiability w.r.t. a TBox)

Some of the Main Ideas of Today

- Reasoning about concepts in $\mathcal{ALC}$ is PSpace-complete
- The same holds for KB reasoning if TBoxes are acyclic
- Without this restriction, reasoning in $\mathcal{ALC}$ is ExpTime-complete
- Basic intuition:
  - Branches of polynomial depth $\sim$ PSpace
  - Branches of exponential/unbounded depth $\sim$ ExpTime
- Type elimination is a simple and elegant way to show an ExpTime upper bound for reasoning in $\mathcal{ALC}$ and in other logics

In the next lecture, we will discuss reasoning techniques and complexity results for extensions of $\mathcal{ALC}$