Expressive DLs

The term expressive DLs is used informally to refer to $\mathcal{ALC}$ and its extensions.

As we know, deciding KB satisfiability in all these DLs is at least ExpTime-hard.

- For the DL $\mathcal{ALC}$, we have studied:
  - reasoning algorithms, in particular tableau
  - complexity of reasoning
- We are also familiar with the DLs $\mathcal{ALCI}$, $\mathcal{ALCQ}$ and $\mathcal{ALCIQ}$
- Today we introduce other important expressive DLs
- Similarly as we did for $\mathcal{ALC}$ (but in less detail), we discuss:
  - reasoning algorithms, in particular using tableau, and
  - complexity of reasoning

Recommended Reading and Links

- The DL Complexity Navigator contains links to dozens of complexity results
  http://www.cs.man.ac.uk/~ezolin/dl/
Extending \( \mathcal{ALC} \)

The most common ways to obtain an extension of \( \mathcal{ALC} \) are:

- Adding other concept constructors
  - For example, number restrictions
- Adding role constructors
  - For example, inverses
- Allowing, apart from GCIs, other kind of axioms in the TBox
  - For example, inclusions between roles

Concept Constructors

We have seen number restrictions as examples of additional constructors.

- **Qualified number restrictions**
  \[ Q \geq 2 \text{hasChild.Male} \]
  \[ Q \leq 2 \text{hasChild.Male} \]
- **(Unqualified) number restrictions**
  \[ N \geq 3 \text{hasChild} \]
  \[ N \leq 3 \text{hasChild} \]
- **Functional restrictions**
  \[ F \leq 1 \text{hasFather} \]

Other common concept constructors are:

- **Nominals**, aka \( \mathcal{O} \)
- **Self concepts**, a more recent construct
  (no standard symbol on its own)

Nominals \( \mathcal{O} \)

- allow us to build a concept from a set of individuals \( \{a_1, \ldots, a_n\} \)
- stands for the set of objects that interpret one of the individuals \( a_i \)

For example, we can define:

\[
\{\text{Gaia, Chaos, Chronos, Ananke}\} \quad \text{the primordial gods}
\]
\[
\{\text{Austria, Belgium, \ldots, UK}\} \quad \text{the countries of the EU}
\]

Nominals

**Syntax**

If \( a_1, \ldots, a_n \) are individuals, then \( \{a_1, \ldots, a_n\} \) is a concept

**Semantics**

\[
\{a_1, \ldots, a_n\}^I = \{a_1^I, \ldots, a_n^I\}
\]

Self concepts

- allow us to build a concept \( \exists R.\text{Self} \) from a role \( R \)
- stands for the set of objects that are related via \( R \) to itself

For example, we can define:

\[
\exists \text{loves.\text{Self}} \quad \text{narcissists}
\]
\[
\exists \text{hasAncestor.\text{Self}} \quad \text{individuals that are their own ancestors}
\]

Self

**Syntax**

If \( R \) is a role, then \( \exists R.\text{Self} \) is a concept

**Semantics**

\[
\exists R.\text{Self}^I = \{d \mid (d, d) \in R^I\}
\]
### Some Role Constructors

- The **inverse** is (by far) the most popular role constructor
  
  \[
  \text{Inverse} \quad \neg \text{hasParent}^{-}
  \]

- Sometimes, **Boolean** role constructors are considered
  
  - Union (role disjunction) \( \text{hasFriend} \cup \text{hasRelative} \)
  - Intersection (role conjunction) \( \text{hasFriend} \cap \text{hasRelative} \)
  - Negation \( \neg \text{hasFriend} \)
  - Difference \( \text{hasRelative} \setminus \text{hasFriend} \)

- \( B \) stands for the Booleans intersection, union, and negation
- \( b \) stands for the safe Booleans intersection, union, and difference

- **Other constructors**
  
  - Role composition \( \text{hasParent} \circ \text{hasSibling} \)
  - Regular expressions \( \text{reg} \)

### Summary of Main constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept constructors</td>
<td>( (R \text{ is a role, } C \text{ a concept, } a_i \text{ individuals, } n \in \mathbb{N}) )</td>
<td></td>
</tr>
<tr>
<td>( \mathcal{N} )</td>
<td>NR ( \geq n R )</td>
<td>( {d_1</td>
</tr>
<tr>
<td></td>
<td>( \leq n R )</td>
<td>( {d_1</td>
</tr>
<tr>
<td>( \mathcal{Q} )</td>
<td>QNR ( \geq n \mathcal{R}, C )</td>
<td>( {d_1</td>
</tr>
<tr>
<td></td>
<td>( \leq n \mathcal{R}, C )</td>
<td>( {d_1</td>
</tr>
<tr>
<td>( \mathcal{O} )</td>
<td>nominals ( {a_1, \ldots, a_n} )</td>
<td>( {a_1^2, \ldots, a_2^2} )</td>
</tr>
<tr>
<td></td>
<td>self ( \exists R.\text{Self} )</td>
<td>( {d</td>
</tr>
<tr>
<td>( \mathcal{T} )</td>
<td>inverse ( R^{-} )</td>
<td>( {(d_2, d_1)</td>
</tr>
<tr>
<td>( \text{reg} )</td>
<td>( R_1 \circ R_2 )</td>
<td>( {(d_1, d_3)</td>
</tr>
<tr>
<td>( B )</td>
<td>( R_1 \cap R_2 )</td>
<td>( R_1^2 \cap R_2^2 )</td>
</tr>
<tr>
<td>( b )</td>
<td>( R_1 \cup R_2 )</td>
<td>( R_1^2 \cup R_2^2 )</td>
</tr>
<tr>
<td>( b )</td>
<td>( \neg R )</td>
<td>( \Delta^2 \times \Delta^2 \setminus R^2 )</td>
</tr>
<tr>
<td>( B )</td>
<td>( R_1 \setminus R_2 )</td>
<td>( R_1^2 \setminus R_2^2 )</td>
</tr>
</tbody>
</table>

### Role Axioms

We can also extend these DLs by allowing terminological axioms that refer to roles and their relations

- **Role inclusions** \( \mathcal{H} \) are expressions of the form

  \[ R \subseteq S \]

  for roles \( R \) and \( S \)

  A set of role inclusions is called a role hierarchy or RBox

- **Transitivity axioms** are expressions of the form

  \[ \text{trans}(R) \]

  for a role \( R \), asserting that \( R \) is transitive.

  The extension of \( \mathcal{ALC} \) with transitivity axioms is denoted \( S \)

### Role Axioms (cont’d)

Semantically, in a model of a KB

- For every role inclusion \( R \subseteq S, R^2 \subseteq S^2 \) must hold

- For each \( \text{trans}(R) \), \( R^2 \) must be transitively closed: if \( (d_1, d_2) \in R^2 \) and \( (d_2, d_3) \in R^2 \), then \( (d_1, d_3) \in R^2 \).

In \( \mathcal{SH} \) we can express, for example:

\[
\text{hasParent} \sqsubseteq \text{hasAncestor} \quad \text{trans} \left( \text{hasAncestor} \right)
\]
Some examples of expressive DLs are:

\[
\begin{align*}
ALCHIQb & \quad SHOINb & \quad SHOIQ \\
SHIQ & \quad SHOQ & \quad SHIO \\
ALC_{reg} & \quad ALCI_{reg} & \quad ALCI_{reg}
\end{align*}
\]

- The \(SH\) family of DLs, and in particular \(SHIQ\) and \(SHOIQ\), are closely related to the OWL languages (more later).
- Probably the most widely studied DLs, and most existing reasoners support them.

Important note:

In all extensions of \(SQ\), only simple roles that are not transitive and have no transitive subroles are allowed in the number restrictions.

The \(SR\) family

The new OWL 2 standard is based on the \(SR\) family of DLs, which is an extension of the \(SH\) family.

- It supports Self concepts
- It has complex role inclusion axioms:
  
  \[
  \begin{align*}
  \text{hasParent} & \subseteq \text{hasAncestor} \\
  \text{hasAncestor} \circ \text{hasAncestor} & \subseteq \text{hasAncestor} \\
  \text{hasParent} \circ \text{hasSibling} & \subseteq \text{hasUncle}
  \end{align*}
  \]

The implications between roles must satisfy certain syntactic restrictions

- Strong restrictions on cyclic dependencies
- Witnessed by an order on the roles
- Ensures that the role inclusions form a regular grammar
The DLs underlying OWL

The OWL standards proposed by W3C are based on these logics:

<table>
<thead>
<tr>
<th>OWL Variant</th>
<th>DL counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWL 1 - Lite</td>
<td>SHIF</td>
</tr>
<tr>
<td>OWL 1 - DL</td>
<td>SHOIQ</td>
</tr>
<tr>
<td>OWL 2</td>
<td>SROIQ</td>
</tr>
</tbody>
</table>

Additionally, the OWL standards support datatypes, which are captured by concrete domains in DLs. We do not consider them in this course.

Finite Models

Consider a KB $\mathcal{K}$ with ABox $\{C_0(a)\}$ and TBox $T$:

$$T = \{C_0 \sqsubseteq \lnot A, \top \sqsubseteq \exists R.A \sqcap \leq 1 R^- . \top\}$$

- Is $\mathcal{K}$ satisfiable?
- Does $\mathcal{K}$ have a finite model?

All logics that contain $\mathcal{ALCIF}$ (like $\mathcal{SHIQ}$) do not have the finite model property

Extending the $\mathcal{ALC}$ Tableau to more Expressive DLs

- In general, tableau algorithms for expressive DLs can be obtained by adapting the $\mathcal{ALC}$ tableau with
  - new expansion rules for the new constructs
  - possibly more involved blocking conditions (e.g., pairwise blocking)
  - possibly enriched completion graphs (e.g., more complex labels, store inequalities between nodes)

- We illustrate this by adapting our algorithm to $\mathcal{SHIQ}$, i.e., adding
  - transitivity
  - role hierarchies
  - inverse roles
  - qualified number restrictions

Towards a Tableau Algorithm for $\mathcal{SHIQ}$

Recall that our tableau:

- Takes a KB $\mathcal{K}$ as an input
- Tries to build a labeled graph representing a model of $\mathcal{K}$
- Starts from a simple graph representing the ABox
- Uses tableau rules to non-deterministically expand the graph
- If no clash occurs, the expansion continues until the graph is complete
- To ensure that a complete graph is eventually reached, we need suitable blocking conditions.
Adding inverses and role hierarchies

- Both inverse roles and role hierarchies can be easily accommodated.
- We only need to modify slightly the $\exists$-rule and the $\forall$-rule.
- Instead of looking only for a child $w$ of $v$ such that $R \in \mathcal{L}(v, w)$, we must now consider $R$-neighbors.

**Definition**

We say that $w$ is an $R$-neighbor of $v$ if there is an $S \sqsubseteq^* R$ such that

- $S \in \mathcal{L}(v, w)$
- $S^* \in \mathcal{L}(v, w)$

where $\sqsubseteq^*$ denotes the reflexive transitive closure of the $\sqsubseteq$ relation (given by the role hierarchy).

<table>
<thead>
<tr>
<th>Expansion Rules for $\mathcal{ALC}$ KB satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$\neg$-rule:</strong> if $C_1 \cap C_2 \in \mathcal{L}(v)$, and ${C_1, C_2} \notin \mathcal{L}(v)$ then $\mathcal{L}(v) := \mathcal{L}(v) \cup {C_1, C_2}$</td>
</tr>
<tr>
<td><strong>$\sqcup$-rule:</strong> if $C_1 \sqcup C_2 \in \mathcal{L}(v)$, and ${C_1, C_2} \cap \mathcal{L}(v) = \emptyset$ then $\mathcal{L}(v) := \mathcal{L}(v) \cup {D}$ for some $D \in {C_1, C_2}$</td>
</tr>
<tr>
<td><strong>$\exists$-rule:</strong> if $\exists R.C \in \mathcal{L}(v)$, $v$ is not blocked, and there is no $w$ such that $R \in \mathcal{L}(v, w)$ and $C \in \mathcal{L}(w)$ then create new node $w$ and an arc $(v, w)$, and set $\mathcal{L}(v, w) := {R}$ and $\mathcal{L}(w) := {C}$</td>
</tr>
<tr>
<td><strong>$\forall$-rule:</strong> if $\forall R.C \in \mathcal{L}(v)$, there is some $w$ such that $R \in \mathcal{L}(v, w)$ and $C \notin \mathcal{L}(w)$ then $\mathcal{L}(w) := \mathcal{L}(w) \cup {C}$</td>
</tr>
<tr>
<td><strong>$T$-rule:</strong> if $C_T \notin \mathcal{L}(v)$, then $\mathcal{L}(v) := \mathcal{L}(v) \cup {C_T}$</td>
</tr>
</tbody>
</table>

Handling transitive roles

It is quite easy to simulate the effect of transitive roles in $\mathcal{SHIQ}$:

- We can simply ignore the transitivity axioms, and add GCIs of the form
  $$\forall S.C \sqsubseteq \forall R.\forall R.C$$
  for every $R$ with $\text{trans}(R)$ and $R \sqsubseteq^* S$.
- It is well known that this simple trick results in an equisatisfiable KB.

The tableau algorithm simulates the trick with an special expansion rule:

$$\forall^+\text{-rule: if } \forall S.C \in \mathcal{L}(v), \text{ there is some } R \text{ with } \text{trans}(R) \text{ and } R \sqsubseteq^* S \text{ and there is some } R\text{-neighbor } w \text{ of } v \text{ such that } \forall R.C \notin \mathcal{L}(w) \text{ then } \mathcal{L}(w) := \mathcal{L}(w) \cup \{\forall R.C\}$$
Adding Quantified Number restrictions

The straightforward way of handling QNRs is to add two new rules:

\[\geq\text{-rule: if } \geq n \cdot R.C \in L(v), \text{ and there are less than } n \text{ } R\text{-neighbors } w_i \text{ of } v \text{ with } C \in L(w)\text{, then create } n \text{ new nodes } w_i \text{ with } L(v, w_i) := \{R\} \text{ and } L(w_i) := \{C\}\]

\[\leq\text{-rule: if } \leq n \cdot R.C \in L(v), \text{ and there are } n+1 \text{ } R\text{-neighbors } w_0, \ldots, w_n \text{ of } v \text{ with } C \in L(w_i)\text{, then merge some pair } w_i, w_j \text{ into one}\]

By merging \(w'\) into \(w\), we mean
- \(L(w)\) becomes \(L(w) \cup L(w')\), and
- the children of \(w'\) in the graph become children of \(w\).

Note: if one is ancestor of the other, then we merge the descendant into the ancestor

Adding Quantified Number restrictions (cont'd)

However, the approach above has a problem:

- We can fall into the following situation:
  1. generate successors due to some \(\geq n \cdot R.C\) concept,
  2. merge them to satisfy some \(\leq n \cdot R.C'\),
  3. generate new successors due to \(\geq n \cdot R.C\),
  4. merge them due to \(\leq n \cdot R.C'\), etc.

- The so called ‘yo-yo’ effect can result in non-termination
- To avoid this, we store an inequality relation \(\neq\) between nodes, indicating which nodes can not be merged.

The Choice Rule

Another important thing to observe:

- Consider a KB \(\mathcal{K}\) with ABox \(\{C(a)\}\) and TBox \(\mathcal{T}\):
  \[\mathcal{T} = \{C \subseteq (\leq 1 \cdot R.A) \cap (\leq 1 \cdot R.\neg A) \cap (\geq 3 \cdot R.B)\}\]
  - Is \(\mathcal{K}\) satisfiable?
  - Would the algorithm answer correctly?

- If \(\leq 1 \cdot R.C \in L(v)\) for some \(v\), then we need to know whether \(C\) or \(\neg C\) holds for each \(R\)-neighbor of \(v\).
- For this we add an additional choice rule.

\[\text{choice-rule: if } \leq n \cdot R.C \in L(v) \text{ and there is some } R\text{-neighbor } w \text{ of } v \text{ such that } \{C, \neg \text{NNF}(-C)\} \cap L(w) = \emptyset, \text{ then } L(w) := L(w) \cup \{E\} \text{ for some } E \in \{C, \neg C\}\]

Blocking

As before, we need suitable blocking conditions to ensure termination

For \(\mathcal{SHIQ}\) they are a bit trickier
- blocking should only occur when the structure represents a model
- we must make sure that ‘reusing’ the blocker and its successors does not violate any number restrictions
- it is not enough to look at one node, but at pairs of nodes

A blocked finite completion graph will now induce an infinite model
- in the induced model, rather than creating a cycle for each blocked node, we ‘copy’ the full subtree rooted at its blocker
Reasoning in Expressive DLs  2. Extending the tableau algorithms  2.1 SHIQ Tableau

Blocking (cont’d)

Definition ((Pairwise) blocking)

- **w** is directly blocked (by **w’**) if there are ancestors **v**, **v’** and **w’** of **w** such that
  - **w** is a child of **v** and **w’** a child of **v’**
  - \( L(v) = L(v’), L(w) = L(w’), \) and
  - \( L((v, w)) = L((v’, w’)) \)

- A node is indirectly blocked if one of its ancestors is blocked

- A node is blocked if it is directly or indirectly blocked

Clashes

A violation of the number restrictions may now generate a new type of clashes.

Definition (clash)

A completion graph \( G \) contains a clash if one of the following holds:

- there is some **v** with \( \{C, \neg C\} \subseteq L(v) \) for some concept \( C \), or

- there is some **v** with \( \leq n \ R.C \subseteq L(v) \) and **v** has \( n + 1 \) \( R \)-neighbors \( w_0, \ldots w_n \) with \( w_i \neq w_j \) for each \( i \neq j \), and \( C \in L(w_i) \) for each \( i \).

Reasoning in Expressive DLs  2. Extending the tableau algorithms  2.1 SHIQ Tableau

Expansion Rules for SHIQ, Part 2

\( \forall \_+ \)-rule: if \( \forall S.C \in L(v), \) there is some \( R \) with \( \text{trans}(R) \) and \( R \sqsubseteq^* S \) and there is some \( R \)-neighbor \( w \) of \( v \) such that \( \forall R.C \notin L(w) \)

then \( L(w) := L(w) \cup \{\forall R.C\} \)

\( \geq \)-rule: if \( \geq n \ R.C \in L(v), \) \( v \) is not blocked, and there are less than \( n \) \( R \)-neighbors \( w_i \) \( w \) of \( C \in L(w) \)

then create \( n \) new nodes \( w_i \) with \( L(v, w_i) := \{R\} \)

and \( L(w_i) := \{C\} \) for each \( i \),

and set \( w_i \neq w_j \) for each \( i \neq j \)

Reasoning in Expressive DLs  2. Extending the tableau algorithms  2.1 SHIQ Tableau

Expansion Rules for SHIQ, Part 3

\( \leq \)-rule: if \( \leq n \ R.C \in L(v), \) there are \( n + 1 \) \( R \)-neighbors \( w_0, \ldots w_n \) of \( v \) such that \( C \in L(w_i) \) for each \( i \), and for some \( i \neq j, i, j \leq n, w_i \neq w_j \) does not hold and \( w_j \) is not an ancestor of \( w_i \),

then set \( L(w_j) := L(w_j) \cup L(w_i) \), make each successor of \( w_i \) be a successor of \( w_j \), add \( w_j \neq w \) for every \( w \) with \( w_i \neq w \), and delete \( w_i \)

choice-rule: if \( \leq n \ R.C \in L(v) \) and there is some \( R \)-neighbor \( w \) of \( v \) such that \( \{C, \neg C\} \cap L(w) = \emptyset \),

then \( L(w) := L(w) \cup \{E\} \) for some \( E \in \{C, \neg C\} \)
Correctness of the Tableau Procedure for $\mathcal{K}$

This extended tableau algorithm is a decision procedure for KB satisfiability in $\mathcal{SHIQ}$.

**Theorem**

For an $\mathcal{SHIQ}$ KB $\mathcal{K}$, the procedure terminates, and a complete and clash-free graph can be obtained iff $\mathcal{K}$ is satisfiable.

---

**Termination**

**Lemma**

The tableau algorithm terminates

It is proved as usual:

- As usual, all concepts added to the labels occur in $\mathcal{K}$ and the graph $G$ is a finite forest:
  - it has one root node for each individual in $\mathcal{A}$
  - blocking ensures that the depth of each branch is finite (bounded by an exponential in $|\mathcal{K}|$)
  - its breadth is polynomially bounded by $|\mathcal{K}|$ (more precisely, it is bounded by the number of $\geq$ and $\exists$ concepts, times the largest $n$ in some $\geq n R.C$).
- The construction is not monotonic, since we may merge nodes, but we ensure that what removed nodes are never regenerated.

---

**Soundness**

**Lemma**

If the algorithm builds a complete and clash-free $G$ for $\mathcal{K}$, then $\mathcal{K}$ is satisfiable.

Similarly as before, the complete and clash-free $G$ induces a model $\mathcal{I}$ of $\mathcal{K}$:

- The domain $\Delta^\mathcal{I}$ are now the paths in $G$
- Instead of going to a blocked node, a path goes to its blocker
- Individuals $a$ are interpreted as paths $\hat{a}$ of length 1
- For each concept name $A$,
  $$A^\mathcal{I} = \{ p | \text{the path } p \text{ ends at } v \text{ and } A \in \mathcal{L}(v) \}$$
- The pairs in extension of the roles are determined by the labels of the last arc in each path, taking into account blocking and inverses
- To get a model of $\mathcal{K}$, we need to add some transitive arcs

---

**Completeness, Forest Model Property**

Completeness is proves similarly as for $\mathcal{ALC}$.

**Lemma**

If $\mathcal{K}$ is satisfiable, then the algorithm builds a complete and clash-free $G$ for $\mathcal{K}$.

A form of the forest model property also holds for $\mathcal{ALCHIQ}$

- We call an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot)$ forest-shaped if the (undirected) graph $(V, E)$ with $V = \Delta^\mathcal{I}$ and $E = \{(d, d') | (d, d') \in R^\mathcal{I} \text{ or } (d', d) \in R^\mathcal{I} \text{ for some role } R \text{ and } d, d' \notin \{a^\mathcal{I} | a \in N_I \} \}$ is a set of (disconnected) trees.

Every satisfiable $\mathcal{SHIQ}$ KB has a model that can be obtained by adding some ‘missing’ transitive arcs to a forest-shaped model of an equisatisfiable $\mathcal{ALCHIQ}$ KB.
What about other DLs?

- Similar adjustments allow us to accommodate other constructors
- For $SHOQ$ and $SHIO$, accommodating nominals is not difficult
  - A tableau rule merges any node labeled $\{a\}$ into $\hat{a}$
- In $SHOIQ$, the interaction between nominals, inverses and QNRs makes the adaptation more involved
- For all these DLs, the tableau algorithm has the same worst-case complexity as for $ALC$ (2NExpTime, can be often modified to NExpTime)
- The most expressive DL for which there is a well known tableau procedure is $SROIQ$
- These tableau procedures have been implemented in reasoners

The complexity of reasoning in expressive DLs

In $ALC$:
- Reasoning w.r.t. acyclic (or empty) TBoxes is PSpace-complete
- Reasoning w.r.t. general TBoxes is ExpTime-complete

PSpace-hardness and ExpTime-hardness apply to all expressive DLs but the only upper bounds we have are the ones resulting from the (usually non-optimal) tableau algorithms
TBox internalization

In $\mathcal{SH}$ and its extensions, we can reduce reasoning w.r.t. a TBox $T$ to reasoning about a concept $C^*_T$ (w.r.t. an empty TBox)

- Recall our concept $C_T = \bigcap_{D \in T} C \sqsubseteq D \sqcup \neg C \sqcup D$
- We know that $I$ is a model of $T$ iff $C^*_T = \Delta^I$
- Consider a transitive role $U$ such that $R \sqsubseteq U$ for every role $R$ (incl. inverses) occurring in $T$, and let $C^*_T = \forall R.C_T$
- Then, in every connected interpretation $I$ we have that $C^*_T = \Delta^I$ iff $(C^*_T)^I \neq \emptyset$. Note: Every satisfiable KB has a connected model.

As a consequence of this:

Reasoning about concepts (w.r.t. empty/acyclic TBoxes) in $\mathcal{SH}$ is as hard as reasoning w.r.t. arbitrary TBoxes.

ExpTime-complete DLs

- All the reasoning tasks we have discussed are ExpTime-hard for $\mathcal{SH}$.
- The ExpTime upper bound we showed for $\mathcal{ALC}$ can be (sort of easily) extended to $\mathcal{ALCI}, \mathcal{ALCQ}$.
- For $\mathcal{SHIQ}, \mathcal{SHOQ},$ and $\mathcal{SHIO}$ it is a bit trickier, but the upper bound still holds.

Theorem

Reasoning in any extension of $\mathcal{SH}$ that is contained in $\mathcal{SHIQ}, \mathcal{SHOQ}$ or $\mathcal{SHIO}$ is ExpTime complete.

The same holds for every extension of $\mathcal{ALC}$ if we allow general TBoxes.

Even Harder DLs

- $\mathcal{SHIQ}, \mathcal{SHOQ}$ and $\mathcal{SHIO}$ are in ExpTime, but having nominals, inverses, and counting simultaneously makes reasoning even harder!
- Reasoning in $\mathcal{ALCFIO}$ is NExpTime-hard
  - Roughly, because forest model property is lost. We can build a KB enforcing a grid of exponential size, and then it is easy to show ExpTime-hardness.
- $\mathcal{SHOIQ}$ is NExpTime-complete
  - The upper bound is inherited from the two variable fragment of FOL with counting quantifiers.
- The (DLs underlying) the new OWL standards are even harder
  - $\mathcal{SRIQ}$ is 2ExpTime-complete
  - $\mathcal{SROIQ}$ is 2NExpTime-complete
- Many ‘simple’ extensions make reasoning undecidable, e.g. allowing transitive roles in the number restrictions.

Outline

1. Other expressive DLs
   1.1 The DLs underlying OWL
2. Extending the tableau algorithms
   2.1 A tableau algorithm for $\mathcal{SHIQ}$
3. Complexity of reasoning
4. Reasoners for expressive DLs
   4.1 Tableau based reasoners
   4.2 Reasoners using other techniques
5. Summary
Tableau based reasoners

- **FaCT++** is an open-source tableau-based reasoner for OWL 2 (i.e., \(SROIQ\) with datatypes), implemented in C++
- **Pellet** is an open-source tableau-based reasoner for OWL 2 (i.e., \(SROIQ\) with datatypes), implemented in Java
- **RacerPro** is a commercial tableau-based reasoner for \(SHIQ\) with datatypes (which extends OWL-Lite), implemented in Lisp
- **Cerebra Engine** is a commercial tableau-based reasoner implemented in C++

Beyond tableau

- **Hermit** is a free reasoner for OWL 2 (i.e., \(SROIQ\) with datatypes), based on Java. It implements a hypertableau algorithm, which can be seen as an optimized version of tableau
- **SHER** is a commercial reasoner for \(SHIN\) with datatypes (which extends OWL-Lite), implemented in Java. It is based on Pellet, but also incorporates some database technology for ABox reasoning (instance retrieval, querying)

Other reasoners

- **KAON2** is a free reasoner for \(SHIQ\) implemented in Java. It is based on a resolution algorithm.
- **MSPASS** is an open-source resolution-based reasoner for fragments of FOL, that supports several DLs and other modal logics. It is an extension of the FOL theorem prover SPASS.

For more information on these and other reasoners (e.g., reasoners for lightweight DLs, fuzzy DLs, for other reasoning problems, etc), see http://www.cs.manchester.ac.uk/~sattler/reasoners.html

Summary

- We are now familiar with a wide range of expressive description logics
  - their syntax and semantics
  - main reasoning problems
  - reasoning algorithms, in particular tableau
  - the complexity of reasoning in them existing free and commercial reasoners
- In the next lectures, we will study some lightweight DLs and some new reasoning problems