Logical Foundations
for the
Semantic Web

3. Reasoning Services and Algorithms

Help knowledge engineer and users to build and use ontologies

Ian Horrocks and Ulrike Sattler
University of Manchester
Manchester, UK
{horrocks|sattler}@cs.man.ac.uk
Plan for today

1. “useful” reasoning services
2. relationship between DLs and other logics (briefly)
3. tableau algorithm for $\mathcal{ALC}$ and how to prove its correctness
4. how to extend this algorithm to DAML+OIL and OWL-DL
Remember ontology engineering tasks:

- design
- evolution and maintenance
- inter-operation and Integration
- deployment

Further complications are due to:

- sheer size of ontologies
- number of persons involved
- users not being knowledge experts
- natural laziness
- etc.
Reasoning Services: what we might want in the Design Phase

• be warned when making meaningless statements
  ➤ test satisfiability of defined concepts
  \[ SAT(C, \mathcal{T}) \text{ iff there is a model } \mathcal{I} \text{ of } \mathcal{T} \text{ with } C^{\mathcal{I}} \neq \emptyset \]
  defined concepts which are unsatisfiable \(\leadsto\) faulty modelling

• see consequences of statements made
  ➤ test defined concepts for subsumption
  \[ SUBS(C, D, \mathcal{T}) \text{ iff } C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \text{ for all model } \mathcal{I} \text{ of } \mathcal{T} \]
  unwanted subsumptions \(\leadsto\) faulty modelling
  missing subsumptions \(\leadsto\) imprecise modelling

• see redundancies
  ➤ test defined concepts for equivalence
  \[ EQUIV(C, D, \mathcal{T}) \text{ iff } C^{\mathcal{I}} = D^{\mathcal{I}} \text{ for all model } \mathcal{I} \text{ of } \mathcal{T} \]
  knowing about “redundant” classes helps avoid misunderstandings
Reasoning Services: what we might want when Modifying Ontologies

- the same system services as in the design phase, plus
- automatic generation of concept definitions from examples
  - given individuals $o_1, \ldots, o_n$ with assertions ("ABox") for them, create a (most specific) concept $C$ such that each $o_i \in C^I$ in each model $I$ of $\mathcal{T}$  
    "non-standard inferences"

- automatic generation of concept definitions for too many siblings
  - given concepts $C_1, \ldots, C_n$, create a (most specific) concept $C$ such that $\text{SUBS}(C_i, C, \mathcal{T})$  
    "non-standard inferences"

- etc.
Reasoning Services: what we might want when Integrating and Using Ontologies

For integration:

- the same system services as in the design phase, plus
- the possibility to abstract from concepts to **patterns** and compare patterns
  
  - e.g., compute those concepts $D$ defined in $T_2$ such that

\[
\text{SUBS}(\text{Human} \sqcap (\forall \text{child}. (X \sqcap \forall \text{child}. Y)), D, T_1 \cup T_2)
\]

"non-standard inferences"

When using ontologies:

- the same system services as in the design phase and the integration phase, plus
- automatic classification of individuals
  
  - given individual $o$ with assertions, return all defined concepts $D$ such that

\[
o \in D^\mathcal{I} \text{ for all models } \mathcal{I} \text{ of } T
\]
(many) reasoning problems are inter-reducible:

\[
\text{EQUIV}(C, D, T) \iff \text{sub}(C, D, T) \text{ and } \text{sub}(D, C, T)
\]

\[
\text{SUBS}(C, D, T) \iff \text{not } \text{SAT}(C \sqcap \neg D, T)
\]

\[
\text{SAT}(C, T) \iff \text{not } \text{SUBS}(C, A \sqcap \neg A, T)
\]

\[
\text{SAT}(C, T) \iff \text{cons}(\{o: C\}, T)
\]

➠ In the following, we concentrate on \(\text{SAT}(C, T)\)
Do Reasoning Services need to be Decidable?

We know \( \text{SAT} \) is reducible to \( \text{co-SUBS} \) and vice versa.

Hence \( \text{SAT} \) is undecidable iff \( \text{SUBS} \) is

\( \text{SAT} \) is semi-decidable iff \( \text{co-SUBS} \) is

\( \Rightarrow \) if \( \text{SAT} \) is undecidable but semi-decidable, then

there exists a complete \( \text{SAT} \) algorithm:

\( \text{SAT}(C, T) \Leftrightarrow \) “satisfiable”, but might not terminate if not \( \text{SAT}(C, T) \)

there exists a complete \( \text{co-SUBS} \) algorithm:

\( \text{SUBS}(C, D, T) \Leftrightarrow \) “subsumption”, but might not terminate if \( \text{SUBS}(C, D, T) \)

1. Do expressive ontology languages exist with decidable reasoning problems?

2. Is there a practical difference between ExpTime-hard and non-terminating?
Do Reasoning Services need to be Decidable?

We know \( \text{SAT} \) is reducible to co-\( \text{SUBS} \) and vice versa.

Hence \( \text{SAT} \) is undecidable iff \( \text{SUBS} \) is.

\( \text{SAT} \) is semi-decidable iff co-\( \text{SUBS} \) is.

\( \implies \) if \( \text{SAT} \) is undecidable but semi-decidable, then

there exists a complete \( \text{SAT} \) algorithm:

\[
\text{SAT}(C, T) \Leftrightarrow \text{"satisfiable"}, \text{ but might not terminate if not } \text{SAT}(C, T)
\]

there exists a complete co-\( \text{SUBS} \) algorithm:

\[
\text{SUBS}(C, D, T) \Leftrightarrow \text{"subsumption"}, \text{ but might not terminate if } \text{SUBS}(C, D, T)
\]

1. Do expressive ontology languages exist with \textbf{decidable} reasoning problems?
   \textbf{Yes}: DAML+OIL and OWL-DL

2. Is there a practical difference between \textbf{ExpTime}-hard and non-terminating?
   \textbf{let's see}
Interlude: \( Q \) versus \( N \)

Many DLs carry \( Q \) or \( N \) in their name to indicate that they provide number restrictions.

What is the difference between \( N \) and \( Q \)?

\[ N \]
- \((\geq nR), (\leq nR)\) for \( n \in \mathbb{N} \), role \( R \)
- e.g. \((\leq 4\text{hasChild})\)

\[ Q \]
- \((\geq nR.C), (\leq nR.C)\) for \( n \in \mathbb{N} \), role \( R \), (possibly complex) concept
- e.g. \((\leq 4\text{hasChild}.\text{Blond})\)
- \( \exists R.C \equiv (\geq 1R.C) \) and \( \forall R.C \equiv (\leq 0R.\neg C) \)

In the following, we often neglect these differences... e.g., talk about \textit{SHIQ} instead of \textit{SHIN}
(slide with translation)

- \textit{SHI} is a fragment of first order logic
- \textit{SHIQ} (and thus \textit{SHIQ}) is a fragment of FOL with counting quantifiers and equality
- \textit{SHI} without transitivity is a fragment of first order logic with two variables
- \textit{ALC} is a notational variant of the multi modal logic $K$
  - inverse roles are closely related to converse/past modalities
  - transitive roles are closely related to transitive frames/axiom 4
  - number restrictions are closely related to deterministic programs in PDL
Deciding Satisfiability of $\text{SHIQ}$

Remember: $\text{SHIQ}$ is OWL-DL without datatypes and nominals

Next: tableau-based decision procedure for $\text{SAT}(C, T)$
we start with $\mathcal{ALC}$ ($\sqcap, \sqcup, \neg, \exists, \forall$) instead of $\text{SHIQ}$ and $\text{SAT}(C, \emptyset)$

Technical: all concepts are assumed to be in Negation Normal Form
transform $C$ into equivalent $\text{NNF}(C)$ by pushing negation inwards, using

$$
\neg(C \sqcap D) \equiv \neg C \sqcup \neg D \quad \neg(C \sqcup D) \equiv \neg C \sqcap \neg D
$$

$$
\neg(\exists R.C) \equiv (\forall R.\neg C) \quad \neg(\forall R.C) \equiv (\exists R.\neg C)
$$

The algorithm decides $\text{SAT}(C, \emptyset)$ by trying to construct a model $\mathcal{I}$ for $C$
The algorithm works on a completion tree with

- nodes $x$ corresponding to elements $x$ in domain $\Delta^\mathcal{I}$ of $\mathcal{I}$ to be built
- concepts in node labels, i.e., $C \in \mathcal{L}(x)$ meaning $x \in C^\mathcal{I}$
- edge labels $(x, R, y)$ representing role successorships $(x, y) \in R^\mathcal{I}$

starts with root $x_0$ with $\mathcal{L}(x_0) = \{C_0\}$

applies rules that infer constraints on $\mathcal{I}$

answers “$C_0$ is satisfiable” if and only if rules

- can be applied (non-deterministic rules!)
- exhaustively (until no more rules apply)
- without generating a clash (node label with $\{A, \neg A\} \subseteq \mathcal{L}(x)$)

Rules: see slide

Example: $A \sqcap \exists R.A \sqcap \forall R.(\neg A \sqcup B)$ see blackboard
A Tableau Algorithm for \( ALC \)

**Theorem**  The tableau algorithm decides satisfiability of \( ALC \) concepts

**Lemma**  Let \( C_0 \) be an \( ALC \) concept in NNF.

(a) the t-algorithm terminates when started with \( C_0 \)

(b) \( SAT(C_0) \iff \) rules can be applied exhaustively without generating a clash

**Proof:** (a) the t-algorithm builds a completion tree

- in a monotonic way
- whose node labels are bounded: \( \mathcal{L}(x) \subseteq 2^{\text{sub}(C_0)} \) (and \( |\text{sub}(C_0)| \leq |C_0| \))
- whose depth is bounded by \( |C_0| \): if \( y \) is an \( R \)-successor of \( x \), then
  \[
  \max\{|D| \mid D \in \mathcal{L}(y)\} < \max\{|D| \mid D \in \mathcal{L}(x)\}
  \]
- whose breadth is bounded by \( |C_0| \): at most one successor per \( \exists R.D \in \text{sub}(C_0) \)
Lemma Let $C_0$ be an ALC concept in NNF.

(a) the t-algorithm terminates when started with $C_0$
(b) $\text{SAT}(C_0) \iff$ rules can be applied exhaustively without generating a clash

Proof: (b) $\iff$ the clash-free, complete tree built for $C_0$ corresponds to a model $\mathcal{I}$ of $C_0$:

- set $\Delta^\mathcal{I}$ to the nodes
- set $x \in A^\mathcal{I}$ iff $A \in \mathcal{L}(x)$
- set $(x, y) \in R^\mathcal{I}$ iff $(x, R, y)$ in completion tree
- prove that, if $D \in \mathcal{L}(x)$, then $x \in D^\mathcal{I}$, by induction on structure of $D$

Details: see blackboard

(this finishes the proof since $C_0 \in \mathcal{L}(x_0)$)
Lemma Let $C_0$ be an ALC concept in NNF.

(a) the t-algorithm terminates when started with $C_0$
(b) $\text{SAT}(C_0) \iff$ rules can be applied exhaustively without generating a clash

Proof: (b) $\Rightarrow$ use a model $\mathcal{I}$ of $C_0$ with $a \in C_0^\mathcal{I}$ to steer rule application via mapping

$$\pi : \text{nodes of completion tree into } \Delta^\mathcal{I}$$

built together with completion tree that satisfies

1. if $C \in \mathcal{L}(x)$, then $\pi(x) \in C^\mathcal{I}$
2. if $(x, R, y)$, then $(\pi(x), \pi(y)) \in R^\mathcal{I}$

Existence of $\pi$ implies clash-freeness of tree (1), termination is already proven

Construction of $\pi$: see blackboard with previous example
A Tableau Algorithm for $\mathcal{ALC}$ with TBoxes

Remember:

- A GCI is of the form $C \sqsubseteq D$ for $C$, $D$ (complex) concepts (replace $C \equiv D$ with $C \sqsubseteq D$, $D \sqsubseteq C$)
- A (general) TBox is a finite set of GCIs
- $\mathcal{I}$ satisfies $C \sqsubseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$
- $\mathcal{I}$ is a model of TBox $\mathcal{T}$ iff $\mathcal{I}$ satisfies each GCI in $\mathcal{T}$
- recall translation of GCIs into FOL

Extend $\mathcal{ALC}$ tableau algorithm to decide $\text{SAT}(C_0, T)$ for TBox $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$:

Add a new rule

$\rightarrow_{\text{GCI}}$: If $(\neg C_i \sqcup D_i) \not\in \mathcal{L}(x)$ for some $1 \leq i \leq n$
Then $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{(\neg C_i \sqcup D_i)\}$
Example: Consider TBox \( \{ C \models \exists R.C \} \). Is \( C \) satisfiable w.r.t. this TBox?
A tableau algorithm for $\cal ALC$ with TBoxes

Example: Consider TBox \( \{ C \sqsupset \exists R.C \} \). Is \( C \) satisfiable w.r.t. this TBox?
	ableau algorithm no longer terminates!

Reason: the size of concepts no longer decreases along paths in a completion tree

Observation: most nodes in example completion tree are similar,

algorithm is repeating the same nodes

Solution: Regain termination with cycle-detection

if \( \mathcal{L}(x) \) and \( \mathcal{L}(y) \) are “very similar”, only extend \( \mathcal{L}(x) \)
A tableau algorithm for $\mathcal{ALC}$ with TBoxes: Cycle-detection

Blocking:

- $x$ is **directly blocked** if it has an ancestor $y$ with $\mathcal{L}(x) \subseteq \mathcal{L}(y)$

- in this case (and if $y$ is the “closest” such node to $x$), $x$ is **blocked by** $y$

- A node is **blocked** if it is directly blocked or one of its ancestors is blocked

$\oplus$ restrict the application of all rules to nodes which are not blocked

$\Rightarrow$ Tableau algorithm for $\mathcal{ALC}$ w.r.t. TBoxes

**Example:** check previous example

**Theorem** The extended t-algorithm decides satisfiability of $\mathcal{ALC}$ concepts w.r.t. TBoxes
Lemma Let $C_0$ be an $\mathcal{ALC}$ concept and $T$ a TBox in in NNF.

(a) the $t$-algorithm terminates when started with $C_0$ and $T$
(b) $\text{SAT}(C_0, T) \iff$ rules can be applied exhaustively without generating a clash

Proof: (a) the $t$-algorithm builds a completion tree

- in a monotonic way
- whose depth is bounded by $2^{|C_0|}$: on any longer path, blocking would occur and paths with blocked nodes do not become longer
- whose breadth is bounded by $|C_0|$: at most one successor per $\exists R.D \in \text{sub}(C_0)$
Lemma Let $C_0$ be an $\mathcal{ALC}$ concept and $T$ a TBox in in NNF.

(a) the t-algorithm terminates when started with $C_0$ and $T$
(b) $\text{SAT}(C_0, T) \iff$ rules can be applied exhaustively without generating a clash

Proof: (b) $\Rightarrow$ similar to previous

$\iff$ the clash-free, complete tree built for $C_0$ corresponds to a model $I$ of $C_0$ and $T$:

- set $\Delta^I$ to the unblocked nodes
- set $x \in A^I$ iff $A \in \mathcal{L}(x)$
- set $(x, y) \in R^I$ iff $(x, R, y)$ or $(x, R, y')$ and $y$ blocks $y$
- prove that, if $D \in \mathcal{L}(x)$, then $x \in D^I$, by induction on structure of $D$

Details: see blackboard

(this finishes the proof since $C_0 \in \mathcal{L}(x_0)$ and $\neg C_i \sqcup D_i \in \mathcal{L}(x)$, for all $i, x$)
A tableau algorithm for \textit{SHIQ}: Transitive Roles

Remember: \textit{SHIQ} provides statements $\text{trans}(R)$ to enforce \textit{transitivity} of role $R$

Problem: if $\forall R.C \in \mathcal{L}(x)$ for $R$ transitive and $(x, R, y)$ and $(y, R, z)$ in completion tree, $C$ must go to $\mathcal{L}(z)$

Solution1: add edge $(x, R, z) \implies$ destroys handy tree structure

Solution2: new $\forall$ rule

\[
\rightarrow^+: \text{If } \forall R.C \in \mathcal{L}(x) \text{ and } (x, R, y) \text{ with } R \text{ transitive and } \forall R.C \not\in \mathcal{L}(y) \text{ Then } \mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{\forall R.C\}
\]

Proof of “the Lemma” is similar to previous case, but for model construction:

\begin{itemize}
  \item if $\text{trans}(R)$: $R^T = \{(x, y) \mid (x, R, y) \text{ or } (x, R, y') \text{ and } y' \text{ blocks } y\}^+$
\end{itemize}
A tableau algorithm for \( SHIQ \): Role Hierarchies

**Remember:** \( SHIQ \) allows to state role inclusions \( R \sqsubseteq S \)

**Problem:** if \((x, R, y)\) and \(R \sqsubseteq^+ S\), then \((x, y) \in S^I\)

**Solution:** define \( y \) being an \( S \)-successor of \( x \) if \((x, R, y)\) for some \( R \sqsubseteq^* S \) in rules, replace \“(x, R, y)“ with \“y is \( R \)-successor of \( x \)“

**Problem2:** if \( \forall S.C \in L(x) \) and \( R \) transitive and \( R \sqsubseteq S \) and \((x, R, y)\) and \((y, R, z)\) in completion tree, then \( C \) must go to \( L(z) \)

**Solution:** modify new \( \forall \) rule

\[
\rightarrow^+: \text{ If } \forall S.C \in L(x) \text{, } x \text{ has } R\text{-successor } y \text{ for } R \text{ transitive and } R \sqsubseteq^* S \text{ and } \forall R.C \notin L(y) \text{ Then } L(y) \rightarrow L(y) \cup \{ \forall R.C \} \]
A tableau algorithm for SHIQ: Inverse Roles

Remember: SHIQ allows to use role names and inverse roles $R^-$, e.g. $\forall R^- C$

Problem1: concepts need get pushed up the completion tree

Example: $\exists R. (A \sqcap \forall R^-. (B \sqcap \exists S^- (B \sqcap \forall S. \neg A)))$

Solution: treat role names and inverse roles symmetrically

define $R$-neighbours and replace “successor” with “neighbour” in rules

Problem2: algorithm not correct

Example: $\text{SAT}(A, \{ A \sqsubseteq \exists R. A \sqcap \forall R^- (A \sqcap \neg A) \})$

Reason: direct blocking condition ($x$ blocks $y$ if $L(x) \subseteq L(y)$) too weak

Solution: modify direct blocking condition: $x$ blocks $y$ if $L(x) = L(y)$
A tableau algorithm for \( \text{SHIQ} \): Number Restrictions

Remember: \( \text{SHIQ} \) allows to use number restrictions \((\geq nR.C), (\leq nR.C)\)

Obvious: new rule that generates \( R \)-successors \( y_i \) of \( x \) for \((\geq nR.C) \in \mathcal{L}(x)\)

new rule that identifies surplus \( R \)-successors of \( x \) with \((\leq nR.C) \in \mathcal{L}(x)\)

Example: \((\geq 2R.A) \sqcap (\geq 2R.(A \sqcap B)) \sqcap (\leq 3S.A)\) and \( R \sqsubseteq S \)

Less obvious: new choose rule required

Example: \((\geq 3R.B) \sqcap (\leq 1R.A) \sqcap (\leq 1R.\neg A)\)

Tricky: new blocking condition required

Proofs of Lemma become more demanding, i.e., model construction uses enhanced “unravelling” to construct possibly infinite models...
Models of $\textit{SHIQ}$

For $\textit{SHIQ}$ without number restriction, we built finite models

ok since $\textit{SHI}$ has finite model property, i.e.,
\[
\text{SAT}(C, \mathcal{T}) \Rightarrow C, \mathcal{T} \text{ have a finite model}
\]

For full $\textit{SHIQ}$, we build infinite tree models

ok since $\textit{SHIQ}$ has tree model property, i.e.,
\[
\text{SAT}(C, \mathcal{T}) \Rightarrow C, \mathcal{T} \text{ have a tree model}
\]

ok since $\textit{SHIQ}$ lacks finite model property, i.e.,
there are $C$ and $\mathcal{T}$ with $\text{SAT}(C, \mathcal{T})$,
but each of their models is infinite

Example: for roles $F \subseteq R$ and $R$ transitive,
\[
\neg A \sqcap \exists F.A \sqcap \forall R.(\exists F.A \sqcap (\leq 1 F^- \top))
\]
is satisfiable, but each model has an infinite $F$-chain (blackboard)
4. Reasoning Services, Complexity, and Decidability

Help knowledge engineer and users to build and use ontologies

Ian Horrocks and Ulrike Sattler
University of Manchester
Manchester, UK
{horrocks|sattler}@cs.man.ac.uk
Plan for today

1. a few interesting complexity results for DLs
2. why full DAML+OIL and OWL-DL are more complex
3. some interesting undecidability results
4. implementing and optimising tableau algorithm
Remember Yesterday

Yesterday, we have seen a **tableau-based algorithm** that decides

satisfiability of $\text{SHIQ}$ concepts w.r.t. $\text{SHIQ}$ TBoxes

Still missing from $\text{SHIQ}$ to OWL-DL:

- **data types** (integers, strings, with comparisons)
  
e.g., $\text{Human} \sqcap \exists \text{age} > 18$  
  extension of algorithm not too difficult

- **nominals**: special concepts; for $N$ nominal, only consider $\mathcal{I}$ with $|N^\mathcal{I}| = 1$
  
e.g., Human $\sqcap \exists \text{met.Pope}$

adding nominals to $\text{SHIQ}$ yields $\text{SHIQO}$:

- decidable — not yet proven (but there are good reasons)
- **no** tree model property: makes reasoning more “difficult”
- more complex than $\text{SHIQ}$
<table>
<thead>
<tr>
<th>concepts in</th>
<th>Definition</th>
<th>without a TBox is</th>
<th>w.r.t. a TBox is</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALC</td>
<td>$\cap, \cup, \neg, \exists R.C, \forall R.C,$</td>
<td>PSpace-c</td>
<td>ExpTime-c</td>
</tr>
<tr>
<td>S</td>
<td>ALC + transitive roles</td>
<td>PSPace-c</td>
<td>ExpTime-c</td>
</tr>
<tr>
<td>SI</td>
<td>S + inverse roles</td>
<td>PSPace-c</td>
<td>ExpTime-c</td>
</tr>
<tr>
<td>SH</td>
<td>S + role hierarchies</td>
<td>ExpTime-c</td>
<td>ExpTime-c</td>
</tr>
<tr>
<td>SHIQ</td>
<td>SHI + number restrictions</td>
<td>ExpTime-c</td>
<td>ExpTime-c</td>
</tr>
<tr>
<td>SHIQO</td>
<td>SHIQ + nominals</td>
<td>decidable</td>
<td>decidable</td>
</tr>
<tr>
<td>SHIQ +</td>
<td>SHIQ + “naive number restrictions”</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
<tr>
<td>SH +</td>
<td>SH + “naive role hierarchies”</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>
**ALC** is in PSpace

The **NExpTime** tableau algorithm for \( \text{SAT}(\text{ALC}, \emptyset) \) can be modified easily to run in \text{PSpace}:

For an **ALC**-concept \( C_0 \),

1. the c-tree can be built **depth-first**
2. branches are independent \( \leadsto \) keep only one branch in memory at any time
3. length of branch \( \leq |C_0| \)
4. for each node \( x \), \( \mathcal{L}(x) \subseteq \text{sub}(C_0) \) and \( \# \text{sub}(C_0) \) is linear in \( |C_0| \)

\( \leadsto \) **non-deterministic** \text{PSpace} decision procedure for \( \text{CSAT}(\text{ALC}) \) and Savitch: \text{PSpace} = \text{NPSpace}
Adding TBoxes to $\mathcal{ALC}$ yields ExpTime-hardness

Why is reasoning w.r.t. TBoxes more complex, i.e., ExpTime-hard?

Intuitively: we can enforce paths of exponential length (without loops) i.e.,

there are $A$, $T$ such that, in each model $I$ of $A$ and $T$, there is
a path $x_1, \ldots, x_n$ with $i \neq j$ implies $x_i \neq x_j$,

$$(x_i, x_{i+1}) \in R^I, \text{ and } n \geq 2(|A| + |T|)^2$$

$A$ and $T$ represent binary incrementation using $k$ bits
$i$-th bit is coded in concept name $X_i$ ($X_k$ is lowest bit, $C \Rightarrow D$ short for $\neg C \sqcup D$)

$A = \neg X_1 \sqcap \neg X_2 \sqcap \ldots \sqcap \neg X_k$ \hspace{1cm} (represents 000\ldots0)

$T = \{ A \sqsubseteq \exists R.A$ \hspace{1cm} (to enforce (possibly cyclic) chain)

$A \sqsubseteq (X_k \Rightarrow \forall R.\neg X_k) \sqcap (\neg X_k \Rightarrow \forall R.X_k)$ \hspace{1cm} (switch bits)

for $i < k$:

$\bigsqcap_{j < i} X_j \sqsubseteq (X_i \Rightarrow \forall R.\neg X_i) \sqcap (\neg X_i \Rightarrow \forall R.X_i)$ \hspace{1cm} (switch bits)

$\bigsqcup_{j < i} \neg X_j \sqsubseteq (X_i \Rightarrow \forall R.X_i) \sqcap (\neg X_i \Rightarrow \forall R.\neg X_i)$ \hspace{1cm} (switch bits)\}
Adding TBoxes to ALC yields ExpTime-hardness

Why is reasoning w.r.t. TBoxes more complex, i.e., ExpTime-hard?

**Lemma:** Satisfiability of ALC w.r.t. TBoxes can be reduced to the Halting Problem of polynomial-space-bounded alternating Turing machines

We know: the HP-f-PSB-A-TM is ExpTime-hard

**Proof of Lemma:** beyond the scope of this tutorial, but not difficult
Complexity of \( SHIQ \)

\( SHIQ \) is \text{ExpTime-hard} because \( ALC \) with TBoxes is and \( SHIQ \) can internalise TBoxes: polynomially reduce \( SAT(C, T) \) to \( SAT(C_T, \emptyset) \)

\[
C_T := C \cap \bigcap_{C_i \sqsubseteq D_i \in T} (C_i \Rightarrow D_i) \cap \forall U. \quad \bigcap_{C_i \sqsubseteq D_i \in T} (C_i \Rightarrow D_i)
\]

for \( U \) new role with \( \text{trans}(U) \), and

\[
R \sqsubseteq U, \quad R^- \sqsubseteq U \quad \text{for all roles } R \text{ in } T \text{ or } C
\]

**Lemma:** \( C \) is satisfiable w.r.t. \( T \) iff \( C_T \) is satisfiable

Why is \( SHIQ \) in \text{ExpTime}?

Tableau algorithms runs in worst-case \text{non-deterministic double exponential space using double exponential time}....
Translation of \textit{SHIQ} into Büchi Automata on infinite trees

\[ C, T \leadsto A_{C,T} \]

such that

1. \( \text{SAT}(C, T) \iff L(A_{C,T}) \neq \emptyset \)

2. \( |A_{C,T}| \) is exponential in \( |C| + |T| \)
   (states of \( C, T \) are sets of subconcepts of \( C \) and \( T \))

This yields ExpTime decision procedure for \( \text{SAT}(C, T) \) since emptiness of \( L(A) \) can be decided in time polynomial in \( |A| \)

\textbf{Problem} \( A_{C,T} \) needs (?) to be constructed before being tested: best-case ExpTime
**SHIQO** is NExpTime-hard

**Fact:** for **SHIQ** and **SHOQ**, SAT($C, T$) are ExpTime-complete

$I$ stands for “with inverse roles”, $O$ for “with nominals”

**Lemma:** their combination is NExpTime-hard

even for **ALCQIO** (**SHIQO** without trans. roles and role hierarchies),

SAT($C, T$) is NExpTime-hard
**SHIQO** is NExpTime-hard

**Fact:** for **SHIQ** and **SHOQ**, SAT\((C, T)\) are ExpTime-complete

I stands for “with inverse roles”, O for “with nominals”

**Lemma:** their combination is NExpTime-hard

even for **ALCQIO** (**SHIQO** without trans. roles and role hierarchies),
SAT\((C, T)\) is NExpTime-hard

**Proof:** by reduction of a NExpTime version of the domino problem:

\[\text{can we tile a } 2^n \times 2^n \text{ square using } D?\]
**NExpTime DLs:** $\texttt{ALCQIQO}$ is NExpTime-hard

**Definition:** A domino system $\mathcal{D} = (D, H, V)$

- set of domino types $D = \{D_1, \ldots, D_d\}$, and
- horizontal and vertical matching conditions $H \subseteq D \times D$ and $V \subseteq D \times D$

A tiling of the $\mathbb{N} \times \mathbb{N}$ grid using $\mathcal{D}$:

$$t : \mathbb{N} \times \mathbb{N} \rightarrow D \text{ such that }$$

$$\langle t(m, n), t(m + 1, n) \rangle \in H \text{ and }$$

$$\langle t(m, n), t(m, n + 1) \rangle \in V$$

**Domino problem**

*standard:* has $\mathcal{D}$ a tiling? **undecidable**

*exponential:* has $\mathcal{D}$ a tiling for a $2^n \times 2^n$ square? **NExpTime-c.**
Reducing the NExpTime domino problem to \( \text{CSAT}(\mathcal{ALCQI}) \) \( \leadsto \) four tasks:

1. each object carries exactly one domino type \( D_i \)
   \( \leadsto \) use concept name \( D_i \) for each domino type and

   \[
   \top \models \bigcup_{1 \leq i \leq d} (D_i \cap \bigcap_{j \neq i} \neg D_j)
   \]

2. each element \( x \) has exactly one \( H \)-successor
   exactly one \( V \)-successor

   whose domino types satisfy the horizontal/vertical matching conditions:

   \[
   \top \models \bigcap_{1 \leq i \leq n} \left( D_i \Rightarrow (\exists V. \bigcup_{(D_i,D_j) \in V} D_j) \cap (\leq 1 V. \top) \cap (\exists H. \bigcup_{(D_i,D_j) \in H} D_j) \cap (\leq 1 H. \top) \right)
   \]
the model must be large enough, i.e., have $2^n \times 2^n$ elements

encode the position $(x, y)$ of each point using binary coding in

the concept names $X_1, \ldots, X_n, Y_1, \ldots, Y_n$:

$$
\top \sqsubseteq \exists H. \top \land \exists V. \top
$$

$$
\top \sqsubseteq (X_k \Rightarrow \forall H. \neg X_k) \land (\neg X_k \Rightarrow \forall H. X_k) \land (\text{same for } Y_i, V)
$$

for $i < k$:

$$
\bigsqsubseteq (X_i \Rightarrow \forall H. \neg X_i) \land (\neg X_i \Rightarrow \forall H. X_i) \land (\text{same for } Y_i, V)
$$

$$
\bigsqsubseteq (X_i \Rightarrow \forall H. X_i) \land (\neg X_i \Rightarrow \forall H. \neg X_i) \land (\text{same for } Y_i, V)
$$

E.g., if $x \in (\neg X_1 \sqcap X_2 \sqcap X_3 \sqcap Y_1 \sqcap \neg Y_2 \sqcap Y_3) \uparrow$, then

$x$ represents $(011, 101)$, and thus the point $(3, 5)$
ensure that the $V \circ H$-successor of each node coincides with its $H \circ V$-successor

\[ \rightarrow \text{enforce that each object is the } H\text{-successor of at most one element} \]
(\text{and the same for } V):

\[ T \models (\leq 1V^- \cdot T) \cap (\leq 1H^- \cdot T) \]

\[ \rightarrow \text{enforce that there is } \leq 1 \text{ object in the upper right corner:} \]

\[ X_1 \cap \ldots \cap X_n \cap Y_1 \cap \ldots \cap Y_n \subseteq N \]

\text{for nominal } N

\textbf{Harvest:}

\[ \neg X_1 \cap \ldots \cap \neg X_n \cap \neg Y_1 \cap \ldots \cap \neg Y_n \]

is satisfiable w.r.t. to $\mathcal{T}_D$ defined above iff $D$ has a $2^n \times 2^n$-tiling
An Undecidable Extension for $SHIQ$

In $SHIQ$, each role $R$ in a number restriction ($\bowtie n R C$) must be simple, i.e., not $\text{Trans}(S)$ for any sub-role $S$ of $R$.

**Lemma:** Without this restriction, $SHIQ$ (even $SHQ$) becomes undecidable.
In $\text{SHIQ}$, each role $R$ in a number restriction ($\Join n R; C$) must be simple, i.e., not $\text{Trans}(S)$ for any sub-role $S$ of $R$.

**Lemma:** Without this restriction, $\text{SHIQ}$ (even $\text{SHQ}$) becomes undecidable.

**Proof**  by a reduction of the standard, unbounded domino problem.

$D$, a fixed set of domino types.

can we tile the first quadrant using $D$?
An Undecidable Extension for \( SHIQ \)

Remember and adapt 4 tasks in the previous domino reduction:

1. each object carries exactly one domino type \( D_i \)
   ~⇒ use concept name \( D_i \) for each domino type and

   \[
   \top \sqsubseteq \bigcup_{1 \leq i \leq d} (D_i \cap \bigcap_{j \neq i} \neg D_j)
   \]

2. each element \( x \) has exactly one \( H \)-successor
   exactly one \( V \)-successor
   whose domino types satisfy the horizontal/vertical matching conditions:

   \[
   \top \sqsubseteq \bigcap_{1 \leq i \leq n} \left( D_i \Rightarrow (\leq 1 V. \top) \cap (\exists V. \bigcup_{(D_i, D_j) \in V} D_j) \right) \cap \\
   (\leq 1 H. \top) \cap (\exists H. \bigcup_{(D_i, D_j) \in H} D_j))
   \]
An Undecidable Extension for \textit{SHIQ}

Remember and adapt 4 tasks in the previous domino reduction:

\begin{enumerate}
\item model must be large enough
\[ \top \sqsubseteq \exists \mathbf{V} . \top \cap \exists \mathbf{H} . \top \]
\item vertical-horizontal and horizontal-vertical successor coincide
\end{enumerate}

- use additional roles \( \mathbf{V}_1, \mathbf{V}_2 \sqsubseteq \mathbf{V}, \mathbf{H}_1, \mathbf{H}_2 \sqsubseteq \mathbf{H} \)
  with additional GCIs, e.g.,
  \[ \top \sqsubseteq (\exists \mathbf{V}_1 . \top \cap \forall \mathbf{V}_1 . \forall \mathbf{V}_1 . \bot) \sqcup \ldots \]
- transitive roles \( \mathbf{D}_{i,j} \) with \( \mathbf{H}_i \sqsubseteq \mathbf{D}_{i,j} \) and \( \mathbf{V}_j \sqsubseteq \mathbf{D}_{i,j} \)
- number restrictions
  \[ \top \sqsubseteq \bigcap_{i,j} (\leq 3 \mathbf{D}_{i,j} . \top) \]
Naive implementation of $SHIQ$ tableau algorithm is doomed to failure:

Construct a tree of exponential depth in a non-deterministic way
\[\Rightarrow\] requires backtracking in a deterministic implementation

Optimisations are crucial
concern every aspect of the help in “many” cases (which?)

In the following: a selection of some vital optimisations
FaCT provides service “classify all concepts defined $\mathcal{T}$”, i.e.,
for all concepts $C, D$ defined in $\mathcal{T}$, FaCT decides $\text{SUBS}(C,D,\mathcal{T})$ and $\text{SUBS}(C,D,\mathcal{T})$
$\Dasharrow \text{SAT}(C \sqcap \neg D, \mathcal{T})$ and $\text{SAT}(D \sqcap \neg C, \mathcal{T})$
$\Dasharrow n^2$ satisfiability tests!

Goal: reduce number of satisfiability tests when classifying $TBox$

Idea: “trickle” new concept into hierarchy
computed so far
FaCT provides service "classify all concepts defined $T$", i.e.,
for all concepts $C, D$ defined in $T$, FaCT decides $\text{SUBS}(C, D, T)$ and $\text{SUBS}(C, D, T)$
$\leadsto \text{SAT}(C \sqcap \neg D, T)$ and $\text{SAT}(D \sqcap \neg C, T)$
$\leadsto n^2$ satisfiability tests!

Goal: reduce number of satisfiability tests when classifying $TBox$

Idea: “trickle” new concept into hierarchy computed so far
FaCT provides service “classify all concepts defined $\mathcal{T}$”, i.e.,
for all concepts $C, D$ defined in $\mathcal{T}$, FaCT decides $\text{SUBS}(C, D, \mathcal{T})$ and $\text{SUBS}(C, D, \mathcal{T})$
\[ \leadsto \text{SAT}(C \sqcap \neg D, \mathcal{T}) \text{ and } \text{SAT}(D \sqcap \neg C, \mathcal{T}) \]
\[ \leadsto n^2 \text{ satisfiability tests!} \]

**Goal:** reduce number of satisfiability tests when classifying $TBox$

**Idea:** “trickle” new concept into hierarchy computed so far
Remember: \( \rightarrow_{\text{GCI}}: \) If \((\neg C_i \sqcup D_i) \not\in \mathcal{L}(x)\) for some \(1 \leq i \leq n\)

Then \(\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{(\neg C_i \sqcup D_i)\}\)

Problem: 1 disjunction per GCI \(\sim\) high degree of non-determinism

huge search space

Observation: many GCIs are of the form \(A \sqcap \ldots \sqsubseteq C\) for concept name \(A\)

e.g., Human \(\sqcap \ldots \sqsubseteq C\) versus Device \(\sqcap \ldots \sqsubseteq C\)

Idea: restrict applicability of \(\rightarrow_{\text{GCI}}\) by translating

\[A \sqcap X \sqsubseteq C\]

into equivalent \(A \sqsubseteq \neg X \sqcup C\)

e.g., Human \(\sqcap \exists \text{owns}.\text{Pet} \sqsubseteq C\) becomes Human \(\sqsubseteq \neg \exists \text{owns}.\text{Pet} \sqcup C\)

this yields localisation of GCIs to As
Optimising the $SHIQ$ Tableau Algorithm: Optimised Blocking

For $SHIQ$, the blocking condition is:

$y$ is blocked by $y'$ if

for $x$ the predecessor of $y$, $x'$ the predecessor of $y'$

1. $L(x) = L(x')$
2. $L(y) = L(y')$
3. $(x, R, y) \iff (x', R, y')$

$\leadsto$ blocking occurs late
$\leadsto$ search space is huge
For $\textit{SHIQ}$, the blocking condition is:

\[ y \text{ is blocked by } y' \text{ if} \]

for \( x \) the predecessor of \( y \), \( x' \) the predecessor of \( y' \)

1. \( L(x) = L(x') \)
2. \( L(y) = L(y') \)
3. \( (x, R, y) \text{ iff } (x', R, y') \)

\[ \Longrightarrow \text{ blocking occurs late} \]
\[ \Longrightarrow \text{ search space is huge} \]

1. \( L(x) \cap RC = L(x') \cap RC \)
2. \( L(y) \cap RC = L(y') \cap RC \)
3. \( (x, R, y) \text{ iff } (x', R, y') \)

\[ \Longrightarrow \text{ blocking occurs earlier} \]
\[ \Longrightarrow \text{ search space is smaller} \]
Remember If a clash \((A, \neg A \in \mathcal{L}(x))\) is encountered, algorithm backtracks

i.e., returns to last non-deterministic choice and
tries other possibility

Example \(\exists R. (A \cap B) \cap (C_1 \cup D_1) \cap \ldots \cap (C_n \cup D_n) \cap \forall R. \neg A \in \mathcal{L}(x)\)
Remember If a clash \((A, \neg A \in \mathcal{L}(x))\) is encountered, algorithm backtracks

i.e., returns to last non-deterministic choice and
tries other possibility

Example \(\exists R. (A \cap B) \cap (C_1 \cup D_1) \cap \ldots \cap (C_n \cup D_n) \cap \forall R. \neg A \in \mathcal{L}(x)\)
Optimising the SHIQ Tableau Algorithm: Backjumping

Remember If a clash \((A, \neg A \in \mathcal{L}(x))\) is encountered, algorithm backtracks

i.e., returns to last non-deterministic choice and tries other possibility

Example \(\exists R. (A \land B) \land (C_1 \lor D_1) \land \ldots \land (C_n \lor D_n) \land \forall R. \neg A \in \mathcal{L}(x)\)

\[
\begin{align*}
\mathcal{L}(x) &\cup \{C_1\} \\
\mathcal{L}(x) &\cup \{C_{n-1}\} \\
\mathcal{L}(x) &\cup \{C_n\} \\
\mathcal{L}(x) &\cup \{\neg C_n, D_n\} \\
\mathcal{L}(x) &\cup \{\neg C_2, D_2\} \\
\mathcal{L}(x) &\cup \{\neg C_1, D_1\}
\end{align*}
\]

\[
\begin{align*}
\mathcal{L}(y) &\cup \{(A \land B), \neg A, A, B\} \\
\mathcal{L}(y) &\cup \{(A \land B), \neg A, A, B\}
\end{align*}
\]

Clash \quad Clash \quad Clash \quad \ldots \quad Clash
Remember If a clash \((A, \neg A \in \mathcal{L}(x))\) is encountered, algorithm backtracks
i.e., returns to last non-deterministic choice and
tries other possibility

Example \(\exists R. (A \sqcap B) \sqcap (C_1 \cup D_1) \sqcap \ldots \sqcap (C_n \cup D_n) \sqcap \forall R. \neg A \in \mathcal{L}(x)\)
Finally: $\textit{SHIQ}$ extends propositional logic

$\Rightarrow$ heuristics developed for SAT are relevant

Summing up: optimisations at each aspect of tableau algorithm can dramatically enhance performance

$\Rightarrow$ do they interact?

$\Rightarrow$ how?

$\Rightarrow$ which combination works best for which “cases”?

$\Rightarrow$ is the optimised algorithm still correct?
5. Future Challenges, Outlook, and Leftovers

Ian Horrocks and Ulrike Sattler
University of Manchester
Manchester, UK
{horrocks|sattler}@cs.man.ac.uk
Plan for today

1. DLs and OWL-DL
2. System Demonstrations of OilEd and Instance Store
3. Concrete Domains and Data Types
4. ABoxes and instances
5. “Non-standard” reasoning services
So far: the Semantic Web and ontologies

importance of automated reasoning

ontology languages: RDF, RDFS, OWL-lite, OWL-full, and OWL-DL

relationship between OWL-DL and DLs such as $SHIQ$ and $SHIQO$

tableau algorithm for $ALC$ with TBoxes

its extension to $SHIQ$

complexity results for $ALC$, $SHIQ$, and $SHIQ$ with nominals

implementation and optimisation of tableau algorithms
OilEd System Demo II
(Remember: $\mathcal{I}$ stands for “with inverse roles”, $\mathcal{O}$ for “with nominals”)

So far, we discussed DLs that are fragments of OWL-DL

$\textbf{SHI}_\mathcal{Q} + \text{Nominals} = \textbf{SHI}_\mathcal{Q}\mathcal{O}$
- we have seen: $\textbf{SHI}_\mathcal{Q}\mathcal{O}$ is $\text{NExpTime}$-hard
- so far: no “goal-directed” reasoning algorithm known for $\textbf{SHI}_\mathcal{Q}\mathcal{O}$
- unclear: whether $\textbf{SHI}_\mathcal{Q}\mathcal{O}$ is “practicable”
- but: t-algorithm designed for $\textbf{SHO}_\mathcal{Q}$
  → live without nominals or inverses

$\textbf{SHI}_\mathcal{Q} + \text{Datatypes} = \textbf{SHI}_\mathcal{Q}(D_n)$
$\textbf{SHO}_\mathcal{Q} + \text{Datatypes} = \textbf{SHO}_\mathcal{Q}(D_n)$
- extend $\textbf{SH}_\mathcal{Q}$ with concrete data and built-in predicates
- extend $\textbf{SH}_\mathcal{Q}$ with, e.g.,
  $\exists \text{age. } > 18$ or
  $\exists \text{age, shoeSize. } =$
- relevant in many ontologies
- dangerous, but well understood extension
- currently being implemented and tested for $\textbf{SHI}_\mathcal{Q}(D_n)$
In Description Logics, datatypes are known as **concrete domains**

**Concrete domain** \( D = (\text{dom}(D), \text{pred}) \) consists of

- a set \( \text{dom}(D) \), e.g., integers, strings, lists of reals, etc.
- a set \( \text{pred} \) of predicates, each predicate \( P \in \text{pred} \) comes with
  - *arity* \( n \in \mathbb{N} \) and
  - a (fixed!) extension \( P^n \subseteq \text{dom}(D)^n \)
- e.g. predicates on \( \mathbb{Q} \): unary \( =_3 \), \( \leq_7 \), binary \( \leq, = \), ternary \( \{(x, y, z) \mid x + y = z\} \)
Datatypes and Concrete Domains

Extending a DL $\mathcal{L}$ with a concrete domain $D$ yields $\mathcal{L}(D)$:

- **new set of abstract features** $N_A$: for $g \in N_A$, $g^\mathcal{I} : \Delta^\mathcal{I} \rightarrow \Delta^\mathcal{I}$

- **new set of concrete features** $N_F$: for $f \in N_F$, $f^\mathcal{I} : \Delta^\mathcal{I} \rightarrow \text{dom}(D)$

- **path**: $u = g_1 \ldots g_k f$, for $g_i \in N_A$, $f \in N_F$
  semantics: $(g_1 \ldots g_k f)^\mathcal{I}(x) = f(g_k(\cdots(g_1^\mathcal{I}(x))\cdots))$

- **new concepts**: $\exists u_1, \ldots, u_n. P$ for $u_i$ paths and $P \in \text{pred}$ $n$-ary predicate
  semantics: $(\exists u_1, \ldots, u_n. P)^\mathcal{I} = \{ x \in \Delta^\mathcal{I} \mid (u_1^\mathcal{I}(x), \ldots, u_n^\mathcal{I}(x)) \in P \}$

- **Examples**
  Human $\sqcap \exists \text{age, marriedTo age.} =$
  Product $\sqcap \exists \text{price, costs.} <$
Known:

\( \exists \text{DL} \ \mathcal{L} : \) finite conjunctions in \( D \) decidable \( \Rightarrow \mathcal{L}(D) \) decidable

e.g., \( \mathcal{L} = ALC, \ D = (\mathbb{N}, \{\geq, \leq, =, \ldots, \geq_k, \leq_k, \ldots\}) \)

\( \exists \text{DL} \ \mathcal{L} : \) finite conjunctions in \( D \) decidable in PTime and \( \mathcal{L}(D) \) undecidable

\( \mathcal{L} = ALC \) with TBoxes, \( D = (\text{String}, \{\text{isPrefixOf}, \ldots\}) \)

\( \leadsto \) in OWL-DL, only paths of length 1 (concrete features) in \( \exists u_1, \ldots, u_n. P \)

and “concrete roles” rather than concrete features...
Summing up: SAT and SUBS in OWL-DL

We know

- how to reason in $\text{SHIQ}$ (proven to be ExpTime-complete) implementation and optimisations well understood
- how to reason in $\text{SHOQ}(D_n)$ (decidable, exact complexity unknown) implementation with nominals $\mathcal{O}$ need more investigations optimisations for $D_n$ are currently being investigated
- that their combination, OWL-DL$^1$, is more complex: NExpTime-hard so far, no “goal-directed” reasoning algorithm known for OWL-DL
t️ live with either nominals or inverses while waiting for/developing full OWL algorithm

1. $\text{SHIQO}(D)$ with number restrictions restricted to $(\geq nR.\top), (\leq nR.\top)$
ABoxes and Instances

Remember: when using ontologies, we would like to automatically classify individuals described in an ABox.

An ABox $\mathcal{A}$ is a finite set of assertions of the form

$$C(a) \text{ or } R(a, b)$$

$I$ is a model of $\mathcal{A}$ if $a^I \in C^I$ for each $C(a) \in \mathcal{A}$

$$(a^I, b^I) \in R^I$$ for each $R(a, b) \in \mathcal{A}$

Cons($\mathcal{A}, \mathcal{T}$) if there is a model $I$ of $\mathcal{A}$ and $\mathcal{T}$

Inst($a, C, \mathcal{A}, \mathcal{T}$) if $a^I \in C^I$ for each model $I$ of $\mathcal{A}$ and $\mathcal{T}$

Easy: \ Inst($a, C, \mathcal{A}, \mathcal{T}$) iff not Cons($\mathcal{A} \cup \{\neg C(a)\}, \mathcal{T}$)

Example: $\mathcal{A} = \{A(a), R(a, b), A(b), S(b, c), B(c)\}$

$\mathcal{T} = \{A \sqsubseteq (\leq 1R. \top)\}$

Inst($a, \forall R.A, \mathcal{A}, \mathcal{T}$) but not Inst($b, \forall S.B, \mathcal{A}, \mathcal{T}$)
ABoxes and Instances

How to decide whether \( \text{Cons}(\mathcal{A}, \mathcal{T}) \)?

\[ \leadsto \text{extend tableau algorithm to start with ABox } \quad C(a) \in \mathcal{A} \Rightarrow C \in \mathcal{L}(a) \]

\[ R(a, b) \in \mathcal{A} \Rightarrow (a, R, b) \]

this yields a graph—in general, not a tree
work on forest—rather than on a single tree
i.e., trees whose root nodes intertwine in a graph
theoretically not too complicated
many problems in implementation

Current Research: how to provide ABox reasoning for huge ABoxes
approach: restrict relational structure of ABox
Instance Store System Demonstration
For Ontology Engineering, useful reasoning services can be based on SAT and SUBS.

Are all useful reasoning services based on SAT and SUBS?

Remember: to support modifying ontologies, we wanted

- automatic generation of concept definitions from examples
  - given ABox $\mathcal{A}$ and individuals $a_i$ create
    - a (most specific) concept $C$ such that each $a_i \in C^I$ in each model $I$ of $\mathcal{T}$
      $$\text{msc}(a_1, \ldots, a_n, \mathcal{A}, \mathcal{T})$$

- automatic generation of concept definitions for too many siblings
  - given concepts $C_1, \ldots, C_n$, create
    - a (most specific) concept $C$ such that $\text{SUBS}(C_i, C, \mathcal{T})$
      $$\text{lcs}(C_1, \ldots, C_n, \mathcal{A}, \mathcal{T})$$
Non-Standard Reasoning Services: msc and lcs

Unlike SAT, SUBS, etc., msc and lcs are computation problems.

Fix a DL $\mathcal{L}$. Define

$$ C = \text{msc}(a_1, \ldots, a_n, A, T) \iff \forall 1 \leq i \leq n \text{ and } \forall I \text{ model of } A \text{ and } T$$

$$a^T_i \in C^I \text{ } \forall 1 \leq i \leq n$$

$C$ is the smallest such concept.

$$ C = \text{lcs}(C_1, \ldots, C_n, T) \iff \text{SUBS}(C_i, C, T) \forall 1 \leq i \leq n$$

$C$ is the smallest such concept.

Clear:

$$ \text{msc}(a_1, \ldots, a_n, A, T) = \text{lcs}(\text{msc}(a_1, A, T), \ldots, \text{msc}(a_n, A, T))$$

$$\text{lcs}(C_1, C_2, C_3, T) = \text{lcs}(\text{lcs}(C_1, C_2, T), C_3, T))$$
Non-Standard Reasoning Services: msc and lcs

Unlike SAT, SUBS, etc., msc and lcs are computation problems.

Fix a DL $\mathcal{L}$. Define

$$C = \text{msc}(a_1, \ldots, a_n, \mathcal{A}, \mathcal{T}) \iff a_i^\mathcal{T} \in C^\mathcal{I} \ \forall 1 \leq i \leq n \text{ and } \forall \mathcal{I} \text{ model of } \mathcal{A} \text{ and } \mathcal{T}$$

$C$ is the smallest such concept, i.e.,

if $a_i^\mathcal{T} \in C^\mathcal{I} \ \forall 1 \leq i \leq n \text{ and } \forall \mathcal{I} \text{ model of } \mathcal{A} \text{ and } \mathcal{T}$

then $\text{SUBS}(C, C', \mathcal{T})$

$$C = \text{lcs}(C_1, \ldots, C_n, T) \iff \text{SUBS}(C_i, C, \mathcal{T}) \ \forall 1 \leq i \leq n$$

$C$ is the smallest such concept, i.e.,

if $C_i \in C' \ \forall 1 \leq i \leq n$

then $\text{SUBS}(C, C', \mathcal{T})$

Clear:

$$\text{msc}(a_1, \ldots, a_n, \mathcal{A}, \mathcal{T}) = \text{lcs}(\text{msc}(a_1, \mathcal{A}, \mathcal{T}), \ldots, \text{msc}(a_n, \mathcal{A}, \mathcal{T}))$$

$$\text{lcs}(C_1, C_2, C_3, \mathcal{T}) = \text{lcs}(\text{lcs}(C_1, C_2, \mathcal{T}), C_3, \mathcal{T}))$$
Non-Standard Reasoning Services: msc and lcs

Known Results:

- **lcs** in DLs with $\sqcap$ is **useless**: \( \text{lcs}(C_1, C_2, T) = C_1 \sqcup C_2 \)

- **msc**\((a, A, T)\) might not exist: e.g., \( \mathcal{L} = \mathcal{ALC} \)
  \[ T = \emptyset \]
  \[ A = \{A(a), R(a, a)\} \]
  \[ \text{msc}(a, A, T) = A \sqcap \exists R.A? A \sqcap \exists R.(A \sqcap \exists R.A)? \]

- \( \exists \) **DLs**: (SUBS, SAT) **msc**, **lcs** are decidable/computable in **polynomial time** \( \mathcal{EL} \) with cyclic TBoxes (only \( \sqcap \) and \( \exists R.C \))

- \( \exists \) **DLs**: **lcs** can be computed, but might be of **exponential size** \( \mathcal{ALE} \) (only \( \sqcap \), primitive \( \neg \), \( \forall R.C \), \( \exists R.C \))
concept pattern: concept with variables in the place of concepts

The following non-standard reasoning services also come w.r.t. TBoxes

unification: $C \equiv ? D$ for $C, D$ concept patterns

solution to $C \equiv ? D$: a substitution $\sigma$ (replacing variables with concepts)

such that $\sigma(C) \equiv \sigma(D)$

Goal: decide unification problem and find a (most specific) such substitution

matching: $C \equiv ? D$ for $C$ concept patterns and $D$ a concept

solution to $C \equiv ? D$: a substitution $\sigma$ with $\sigma(C) \equiv D$
Non-Standard Reasoning Services: other

**approximation:** given DLs $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_1$-concept $C$, find $\mathcal{L}_2$-concept $\hat{C}$ with $\text{SUBS}(C, \hat{C})$ and $\text{SUBS}(C, D)$ implies $\text{SUBS}(\hat{C}, D)$ for all $\mathcal{L}_2$-concepts $D$

Interesting to show concepts to a non-expert user (in some inexpressive, “easy” DL)

**rewriting** given $C$, $\mathcal{T}$, find “shortest” $\hat{C}$ such that $\text{EQUIV}(C, \hat{C}, \mathcal{T})$

Interesting to show concepts computed by msc or lcs to user
(if it doesn’t fit on your screen, it’s no good)
...have a look at the course web page for slides of the tutorial

http://www.cs.man.ac.uk/~horrocks/ESSLI2003/

...write us/ask us if you have any questions