COMP61132: Modal and Description Logics

Week 5

Uli Sattler

Discussion of Coursework

1. Tableau:
   - straightforward but cumbersome application of tableau rules
   - should clarify/raise questions regardings these rules...any?

2. Proof:
   - not quite straightforward argumentation required – I saw some good reasoning!
   - some difficulties regardings all/some models:
     - in a coherent ontology $\mathcal{O}$, each concept name is satisfiable w.r.t. $\mathcal{O}$
     - hence for each A, there is a model $\mathcal{I}$ in which $A^{\mathcal{I}} \neq \emptyset$
     - ...it might be the case that you need **different** models for A and B (but not in $\mathcal{ALC}$)

3. Algorithm:

- advanced difficulty, required background reading & hard thinking for the PSpace argument
- crucial observations: for an acyclic TBox $\mathcal{T}$,
  - the **abs-GCI-rule** suffices!
  - we can define **role depth** of a concept C in $\mathcal{T}$: recursively replace in C all lhs with rhs of axioms in $\mathcal{T}$, then count max. role depth
  - for any **generated** individual a, the maximum role depth in concepts $a : C$ is strictly smaller than this depth for its ancestors.
  - hence the tree depth is bounded by maximum role depth, i.e., linearly in $\mathcal{T}$ and concept size in $\mathcal{A}$.

4. Ontology: any difficulties/insights/questions?

Leftovers from last session

Naive implementation of $\mathcal{ALC}$ tableau algorithm is **doomed to failure:**

It constructs a

- **set of ABoxes**,
- each ABox being of possibly **exponential size**, with possibly exponentially many individuals (see binary counting example)
- in the presence of a GCI such as $\top \sqsubseteq (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcap D_n)$ and exponentially many individuals, algorithm might generate double exponentially many ABoxes

$\rightsquigarrow$ requires double exponential space or

- use non-deterministic variant and backtracking to consider one ABox at a time

$\rightsquigarrow$ requires exponential space

Leftovers from last session

Soundness, completeness, and termination of $\mathcal{ALCI}$ tableau algorithm.

...see slides 38 – 41 from last session

Plan for today

1. Add one further constructor to $\mathcal{ALCI}$: number restrictions
2. Look at undecidable extensions: role chain inclusions and others
3. Justifications: what are they?
4. Finally: other interesting stuff that we can't cover properly

Number restrictions

- a standard constructor in DLs, rather rare in MLs
- given that
  - $\exists r.C$ is "at least 1 r-successor is a C" and
  - $\forall r.C$ is "at most 0 r-successor is **not** a C"
- why not also allow for
  - $\geq nr.C$ is "at least n r-successors are Cs" and
  - $\leq nr.C$ is "at most n r-successors are Cs"
- many examples from applications
  - cars have exactly 4 wheels (at least 4 and at most 4)
  - bicycles have exactly 2 wheels
  - ...

The DL $\mathcal{ALCQI}$

Further add **qualifying number restrictions** $(\geqslant n r.C)$ and $(\leqslant n r.C)$:

$$(\geqslant n r.C)^\mathcal{I} := \{x \in \Delta^\mathcal{I} \mid \#\{y \mid \langle x, y\rangle \in r^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \geq n\}$$
$$(\leqslant n r.C)^\mathcal{I} := \{x \in \Delta^\mathcal{I} \mid \#\{y \mid \langle x, y\rangle \in r^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \leq n\}$$

$\mathcal{ALCQI}$ is $\mathcal{ALCI}$ extended with qualifying number restrictions.

Observation: $\mathcal{ALCQI}$ ontologies do not enjoy the **finite model property:**

$$C_0 := \neg A, \quad \mathcal{T} := \{\top \sqsubseteq \exists R.A \sqcap (\leqslant 1 R^-.\top)\}$$

C_0 is satisfiable w.r.t. $\mathcal{T}$, but **only in infinite models**

Obvious: 2 new rules for tableau algorithm

$\geq$-rule: **If** $x\colon (\geqslant n r.C) \in \mathcal{A}$, x is not blocked, and x has less than n r-neighbours y_i with $y\colon C \in \mathcal{A}$
then create n new individuals $y_1, \ldots, y_n$ and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(x, y_i)\colon r,\ y_i\colon C \mid 1 \leq i \leq n\}$

$\leq$-rule: **If** $x\colon (\leqslant n r.C) \in \mathcal{A}$, x is not indirectly blocked, x has $n+1$ r-neighbours $y_0, \ldots, y_n$ with $y_i\colon C \in \mathcal{A}$, and for each i, j with y_j is **not** an ancestor of y_i, set $\mathcal{A}_{i,j} = \mathcal{A}[y_j/y_i]$
Then replace $\mathcal{A}$ with the set of all those $\mathcal{A}_{i,j}$

A tableau algorithm for $\mathcal{ALCQI}$ ontologies

Problem: Consider what the tableau algorithm does for

satisfiability of $C_0 = (\geqslant 3r.B) \sqcap (\leqslant 2r.A)$ w.r.t. the empty TBox

$\Rightarrow$ algorithm does not terminate due to **yoyo effect**

Solution: Use explicit inequality $\dot{\neq}$ to prevent yoyo effect

A tableau algorithm for $\mathcal{ALCQI}$ ontologies

Solution: Use explicit inequality $\dot{\neq}$ to prevent yoyo effect:

$\geq$-rule: **If** $x\colon (\geqslant n r.C) \in \mathcal{A}$, x is not blocked, and x has less than n r-neighbours y_i with $y_i\colon C \in \mathcal{A}$
then create n new individuals $y_1, \ldots, y_n$ and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(x, y_i)\colon r,\ y_i\colon C \mid 1 \leq i \leq n\} \cup \{y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n\}$

$\leq$-rule: **If** $x\colon (\leqslant n r.C) \in \mathcal{A}$, x is not indirectly blocked, x has $n+1$ r-neighbours $y_0, \ldots, y_n$ with $y_i\colon C \in \mathcal{A}$, and for each i, j with y_j is **not** an ancestor of y_i and $y_i \dot{\neq} y_j \notin \mathcal{A}$ set $\mathcal{A}_{i,j} = \mathcal{A}[y_j/y_i]$ and
Then replace $\mathcal{A}$ with the set of all those $\mathcal{A}_{i,j}$

Note:
- we assume that inequality $y_i \dot{\neq} y_j$ is implicitly symmetric
- an interpretation $\mathcal{I}$ satisfies $a \neq b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$

Extend definition of a clash to NRs: an ABox $\mathcal{A}$ contains a clash if

- $\{x\colon A,\ x\colon \neg A\} \subseteq \mathcal{A}$ for some x, A
- or if
  - $x\colon (\leqslant n r.C) \in \mathcal{A}$ and
  - x has more than n r-neighbours $y_0, \ldots, y_n$ with $y_i \dot{\neq} y_j$ for all $i \neq j$.

Does this suffice? **No:**

$$\{a\colon (\leqslant 1 r.A) \sqcap (\leqslant 1 r.\neg A) \sqcap (\geqslant 3 r.B)\}$$

is **inconsistent**, but the algorithm would answer **"consistent"**

Reason: if $(\leqslant n r.C) \in \mathfrak{L}(x)$ and x has an r-neighbour y, we need to know whether y is a C or not

Solution: add a third new rule:

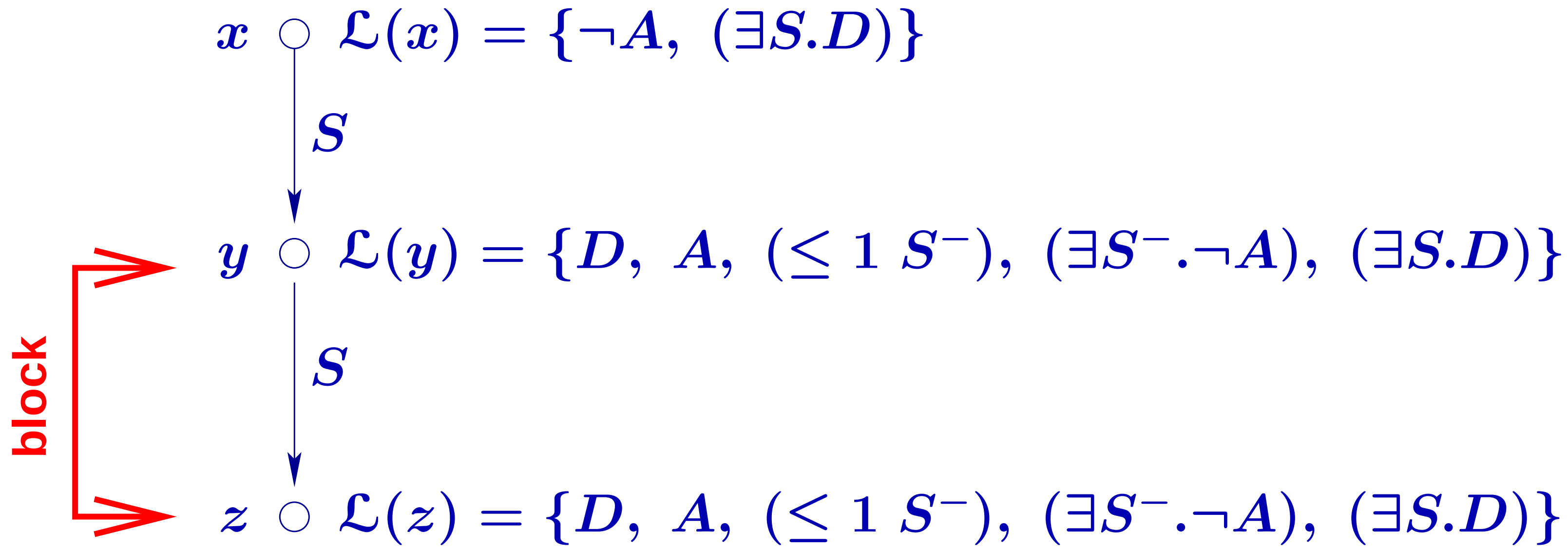
choose-rule: **If** $x\colon ({\leqslant}nr.C) \in \mathcal{A}$, x is not indirectly blocked,
x has an r-neighbour y with $\{y\colon C,\ y\colon \mathsf{NNF}(\neg C)\} \cap \mathcal{A} = \emptyset$
Then replace $\mathcal{A}$ with $\mathcal{A} \cup \{y\colon C\}$ and $\mathcal{A} \cup \{y\colon \mathsf{NNF}(\neg C)\}$

Does this suffice? **No . . .**

A tableau algorithm for $\mathcal{ALCQI}$ ontologies

Example: test satisfiability of $\neg A$ w.r.t. $\mathcal{T}$:

$$\mathcal{T} = \{\top \sqsubseteq \exists S.\underbrace{(A \sqcap ({\leqslant}1 S^{-}.\top) \sqcap (\exists S^{-}.\neg A))}_{D}\}$$



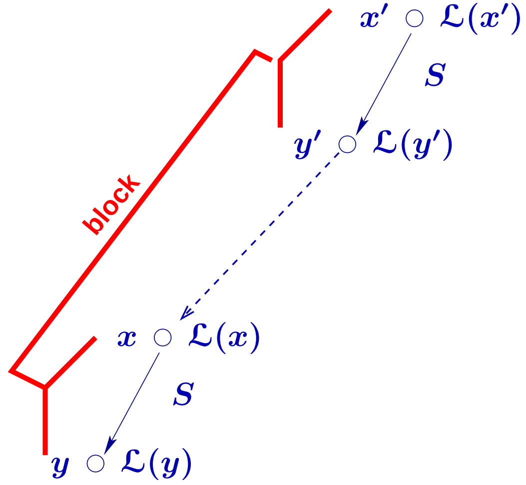
z would block y **but we cannot construct a model from this**

...and $\neg A$ is unsatisfiable w.r.t. $\mathcal{T}$, i.e., our algorithm is still **incorrect!**

Solution: use **double-blocking**:

y is **directly blocked** if there are ancestors x, x', and y' of y with

- x is predecessor of y,
- x' is predecessor of y',
- $\mathcal{E}(\langle x, y \rangle) = \mathcal{E}(\langle x', y' \rangle)$,
- $\mathfrak{L}(x) = \mathfrak{L}(x')$, and $\mathfrak{L}(y) = \mathfrak{L}(y')$.



...using "completion tree notation" rather than ABoxes for brevity

Lemma:

Let $\mathcal{O}$ be an $\mathcal{ALCQI}$ ontology. Then the

1. the algorithm terminates when applied to $\mathcal{O}$
2. if the rules generate a clash-free & complete ABox, then $\mathcal{O}$ is consistent
3. if $\mathcal{O}$ is consistent, then the rules generate a clash-free & complete ABox

Proof:

1. termination is mostly standard, but requires care because $\leq$-rule **removes** individuals
2. soundness is more complicated: we **unravel** a c. & c.-f. ABox into an **infinite tree model**, copying blocking individuals.
3. completeness is standard using a model and mapping π.

Pffffew!

That was hard!

- the tableau algorithm for $\mathcal{ALC}$ wasn't complicated...despite blocking and GCI rules
- extending it to inverse roles was (almost) straighforward:
  - r-neighbours instead of r-successors
  - equality-blocking instead of subset-blocking
- extending it to number restrictions was hard
  - 2 new **obvious** rules, one for $\leq n\ r.C$ and $\geq n\ r.C$
  - an **explicit inequality relation** to prevent yoyo-effect
  - **another** new rule to ensure that we count correctly for $\leq n\ r.C$
  - **double** (equality) blocking instead equality blocking

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Optimised Blocking

For $\mathcal{ALCQI}$, the blocking condition is:

y is blocked by y' if

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

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

$\rightsquigarrow$ blocking occurs **late**
$\rightsquigarrow$ search space is **huge**

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Optimised Blocking

For $\mathcal{ALCQI}$, the blocking condition is:

y is blocked by y' if

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

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

$\rightsquigarrow$ blocking occurs **late**
$\rightsquigarrow$ search space is **huge**

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

for "**relevant concepts** RC"

$\rightsquigarrow$ blocking occurs **earlier**
$\rightsquigarrow$ search space is **smaller**

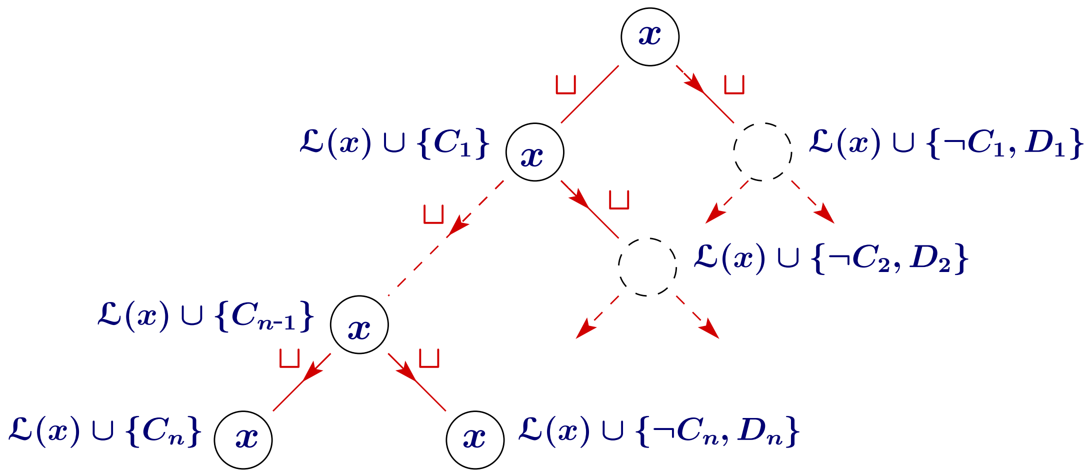
...details are beyond the scope of this course

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** is encountered, **non-deterministic** algorithm **backtracks**

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

Example $x\colon \exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A$

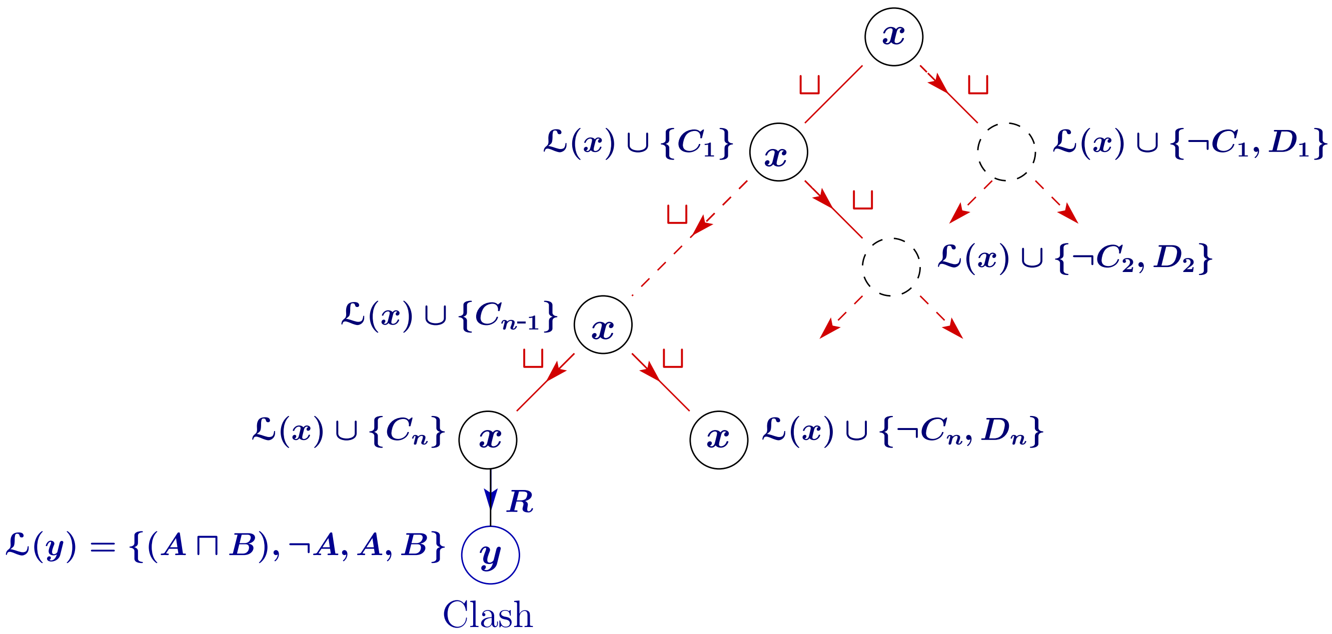


Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** is encountered, **non-deterministic** algorithm **backtracks**

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

Example $x : \exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A$

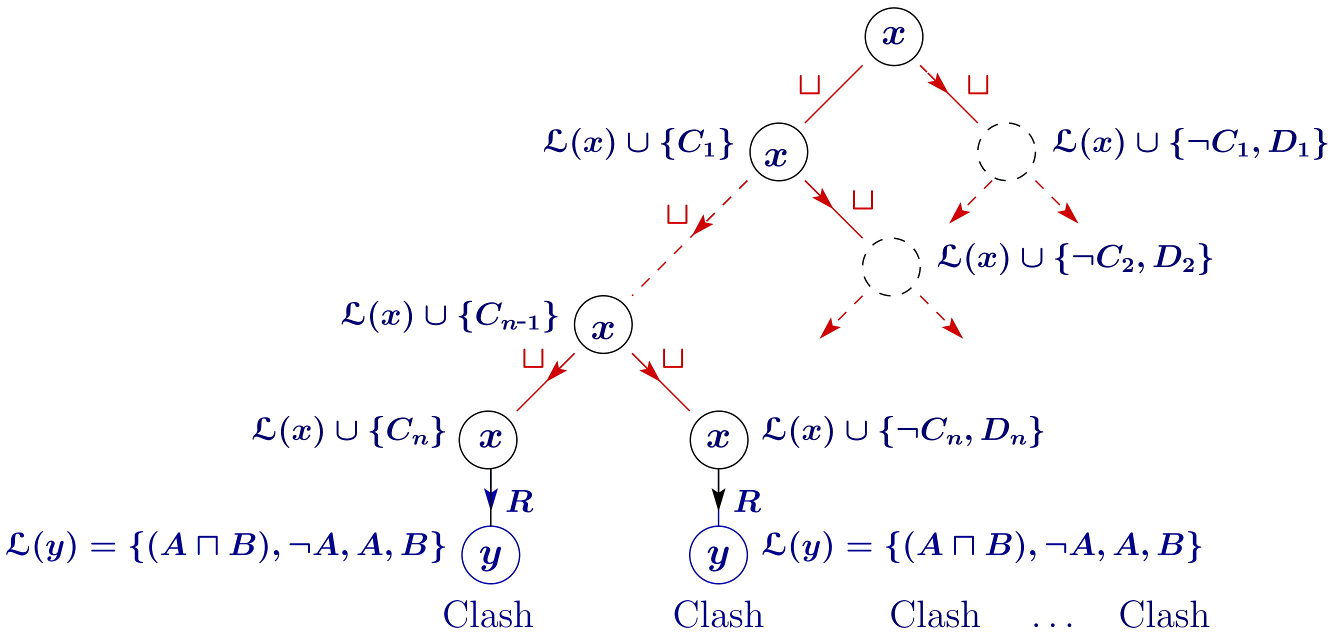


Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** is encountered, **non-deterministic** algorithm **backtracks**

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

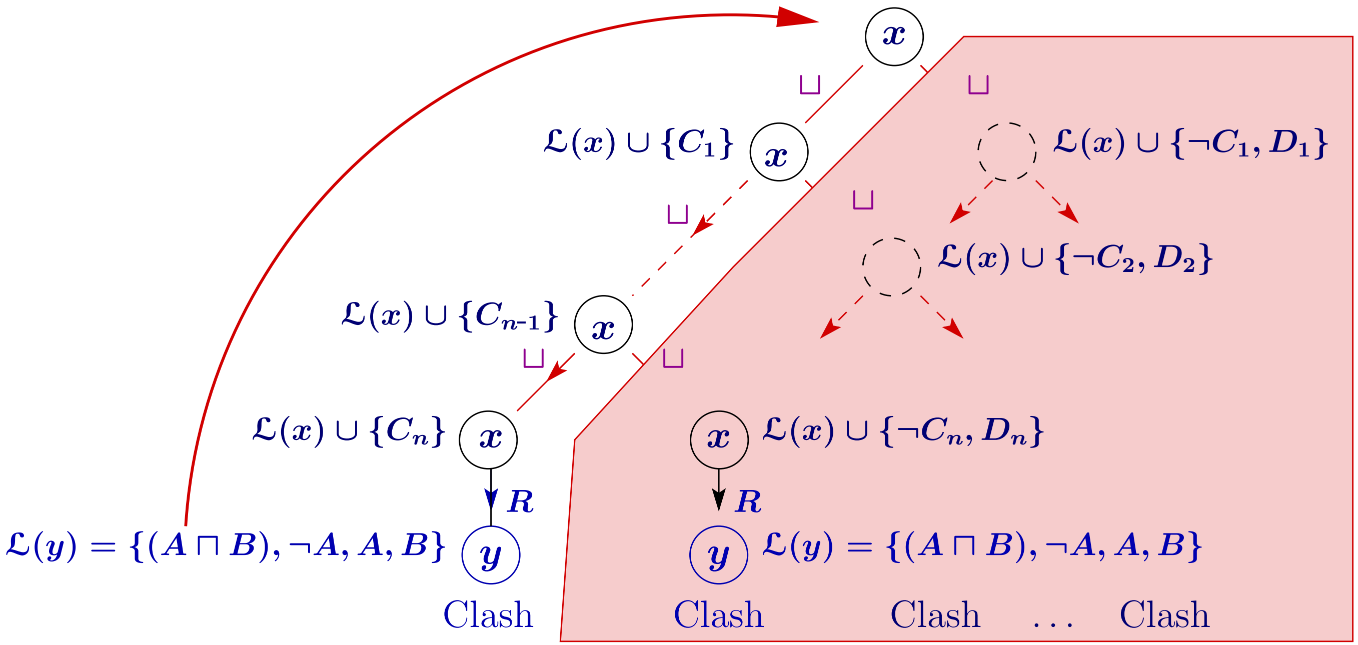
Example $x\colon \exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A$



Remember If a **clash** is encountered, **non-deterministic** algorithm **backtracks**

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

Example $x\colon \exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A$



Finally: $\mathcal{ALCQI}$ extends **propositional logic**

$\rightsquigarrow$ heuristics developed for **SAT** are relevant

Summing up: optimisations are possible at each aspect of tableau algorithm

can dramatically enhance performance

$\rightsquigarrow$ do they interact?

$\rightsquigarrow$ how?

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

$\rightsquigarrow$ is the optimised algorithm still correct?

...are tableau algorithms all there is?

Are all DLs decidable?

So far, we have extended $\mathcal{ALC}$ with

- inverse role and
- number restrictions
- ...which resulted in logics whose reasoning problems are **decidable**
- ...we even discussed **decision procedures** for these extensions

Next, we will discuss some undecidable extension

- $\mathcal{ALC}$ with role chain inclusions
- $\mathcal{ALC}$ with number restrictions on complex roles

An undecidable DL: $\mathcal{ALC}$ with role chain inclusions

OWL 2 supports axioms of the form

- $r \sqsubseteq s$: a model of $\mathcal{O}$ with $r \sqsubseteq s \in \mathcal{O}$ must satisfy $r^\mathcal{I} \subseteq s^\mathcal{I}$
- **trans**(r): a model of $\mathcal{O}$ with **trans**$(r) \in \mathcal{O}$ must satisfy $r^\mathcal{I} \circ r^\mathcal{I} \subseteq r^\mathcal{I}$,
  where $p \circ q = \{(x, z) \mid$ there is $y : (x, y) \in p$ and $(y, z) \in q\}$,
  i.e., a model $\mathcal{I}$ of $\mathcal{O}$ must interpret r as a transitive relation
- $r \circ s \sqsubseteq t$: a model of $\mathcal{O}$ with $r \circ s \sqsubseteq t \in \mathcal{O}$ must satisfy $r^\mathcal{I} \circ s^\mathcal{I} \subseteq t^\mathcal{I}$

subject to some complex restrictions

...**why do we need restrictions?**

...because axioms of this form lead to **loss of tree model property** and **undecidability**

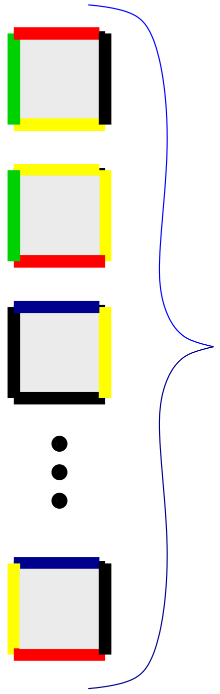
How to prove undecidability of a DL

Often, we prove undecidability of a DL as follows:

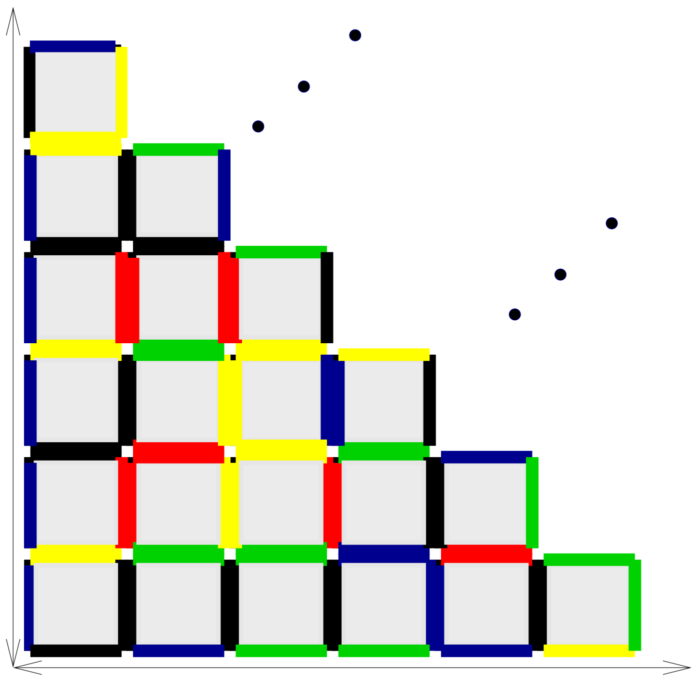
1. **fix reasoning problem**, e.g., satisfiability of a concept w.r.t. a TBox
   - remember Theorem 2?
   - if concept satisfiability w.r.t. TBox is undecidable,
   - then so is consistency of ontology
   - then so is subsumption w.r.t. an ontology
   - ...
2. **pick a decision problem** known to be undecidable, e.g., **the domino problem**
3. **provide a (computable) mapping** $\pi(\cdot)$ that
   - takes an instance D of the domino problem and
   - turns it into a concept A_D and a TBox $\mathcal{T}_D$ such that
   - D has a tiling **if and only if** A_D is satisfiable w.r.t. $\mathcal{T}_D$

   i.e., a decision procedure of concept satisfiability w.r.t. TBoxes could be used as a decision procedure for the domino problem

The Classical Domino Problem - a picture



***D*,**
a fixed
set
of
dominoe
types



can we tile the
first quadrant
using ***D*?**

The Classical Domino Problem

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** for $\mathcal{D}$ is a (total) function:

$$t : \mathbb{N} \times \mathbb{N} \longrightarrow D \text{ such that } \begin{array}{l} \langle t(m,n), t(m+1,n) \rangle \in H \text{ and} \\ \langle t(m,n), t(m,n+1) \rangle \in V \end{array}$$

Domino problem: given $\mathcal{D}$, has $\mathcal{D}$ a tiling?

It is well-known that this problem is undecidable [Berger66]

Almost Encoding the Classical Domino Problem in $\mathcal{ALC}$

We can express various **obligations** of the domino problem in $\mathcal{ALC}$ TBox axioms:

① each object carries **exactly one domino type** D_i

⇝ use unary predicate symbol D_i for each domino type and make sure that all elements carry at least 1 domino type, but not two domino types

$$\begin{array}{rcl} \top & \sqsubseteq & D_1 \sqcup \ldots \sqcup D_d \\ D_1 & \sqsubseteq & \neg D_2 \sqcap \ldots \sqcap \neg D_d \\ D_2 & \sqsubseteq & \neg D_3 \sqcap \ldots \sqcap \neg D_d \\ \vdots & & \vdots \\ D_{d-1} & \sqsubseteq & D_d \end{array}$$

Almost Encoding the Classical Domino Problem in $\mathcal{ALC}$

② every element has a horizontal (X-) successor and a vertical (Y-) successor

$$\top \sqsubseteq \exists X.\top \sqcap \exists Y.\top$$

③ every element satisfies D's **horizontal/vertical matching conditions**:

$$\begin{array}{lcl}
D_1 & \sqsubseteq & \bigsqcup_{(D_1,D)\in H} \forall X.D \sqcap \bigsqcup_{(D_1,D)\in V} \forall Y.D \\
D_2 & \sqsubseteq & \bigsqcup_{(D_2,D)\in H} \forall X.D \sqcap \bigsqcup_{(D_2,D)\in V} \forall Y.D \\
\vdots & \vdots & \\
D_d & \sqsubseteq & \bigsqcup_{(D_d,D)\in H} \forall X.D \sqcap \bigsqcup_{(D_d,D)\in V} \forall Y.D
\end{array}$$

Does this suffice?

I.e., does D have a tiling iff there is a D_i satisfiable w.r.t. the axioms from ① to ③?

- if yes, we have shown that satisfiability of $\mathcal{ALC}$ is undecidable
- so no…what is missing?

④ for each element, its horizontal-vertical-successors **coincide** with their vertical-horizontal-successors and vice versa

$$X \circ Y \sqsubseteq Y \circ X \text{ and } Y \circ X \sqsubseteq X \circ Y$$

Lemma: Let $\mathcal{T}_D$ be the axioms from ① to ④.
Then $\top$ is satisfiable w.r.t. $\mathcal{T}_D$ iff $\mathcal{D}$ has a tiling.

- since the domino problem is undecidable, this implies undecidability of concept satisfiability w.r.t. TBoxes of $\mathcal{ALC}$ with role chain inclusions
- due to Theorem 2, all other standard reasoning problems are undecidable, too
- Proof: 1. show that, from a tiling for D, you can construct a model of $\mathcal{T}_D$
  2. show that, from a model $\mathcal{I}$ of $\mathcal{T}_D$, you can construct a tiling for D
  (tricky because elements in $\mathcal{I}$ can have several X- or Y-successors but we can simply take **the right ones**, see picture)

Let's do this again!

What other constructors can us help to express ④?

- counting and complex roles (role chains and role intersection):

$$\top \sqsubseteq (\leq 1\boldsymbol{X}.\top) \sqcap (\leq 1\boldsymbol{Y}.\top) \sqcap (\exists(\boldsymbol{X} \circ \boldsymbol{Y}) \sqcap (\boldsymbol{Y} \circ \boldsymbol{X}).\top)$$

- restricted role chain inclusions (only 1 role on RHS), and counting (an **all** roles):

$$\begin{aligned} \top &\sqsubseteq (\leq 1\boldsymbol{X}.\top) \sqcap (\leq 1\boldsymbol{Y}.\top) \\ \boldsymbol{X} \circ \boldsymbol{Y} &\sqsubseteq \boldsymbol{r} \\ \boldsymbol{Y} \circ \boldsymbol{X} &\sqsubseteq \boldsymbol{r} \\ \top &\sqsubseteq (\leq 1\boldsymbol{r}.\top) \end{aligned}$$

- various others...see coursework

Are Standard Reasoning Problems/Services Everything?

So far, we have talked a lot about **standard reasoning problems**

- consistency
- satisfiability
- entailments
- ...is this all that is relevant?

Next, we will look at 1 reasoning problem that

- cannot be polynomially reduced to any of the above standard reasoning problems
- is relevant when working with a non-trivial ontology
- ...justifications!

Imagine you are building, possibly with your colleagues, an ontology $\mathcal{O}$, and

- $\mathcal{O}$ is non-trivial, say has 500 axioms, or 5,000, or even more

(S1) a class C is unsatisfiable w.r.t. $\mathcal{O}$

(S2) 27 classes C_i are unsatisfiable w.r.t. $\mathcal{O}$

- Claim: it is possible that $\mathcal{O} \setminus \{\alpha\}$ is coherent, but $\mathcal{O}$ contains 27 unsatisfiable classes
- ...even for a very sensible, small, harmless axiom α

(S3) $\mathcal{O}$ is inconsistent

- Claim: it is possible that $\mathcal{O} \setminus \{\alpha\}$ is consistent, but $\mathcal{O}$ is inconsistent
- ...even for a very sensible, small, harmless axiom α

? what do you do?

? how do you go about **repairing** $\mathcal{O}$?

? which tool support would help you to **repair** $\mathcal{O}$?

Building Ontologies for Real II

Imagine you are building, possibly with your colleagues, an ontology $\mathcal{O}$, and

- $\mathcal{O}$ is non-trivial, say has 500 axioms, or 5,000, or even more

(S4) $\mathcal{O} \models \alpha$, and you want to know **why**

- e.g., so that you can trust $\mathcal{O}$ and α
- e.g., so that you understand how $\mathcal{O}$ models its domain

? what do you do?

? how do you go about **understanding** this entailment?

? which tool support would help you to **understand** this entailment?

? would this tool support be the same/similar to the one to support repair?

Justifications

In all scenarios (Si), we clearly want to know at least the **reasons** for $\mathcal{O} \models \alpha$,

which axioms can I/should I

(S1) **change** so that $\mathcal{O}' \not\models C \sqsubseteq \bot$?

(S2) **change** so that $\mathcal{O}'$ becomes coherent?

(S3) **change** so that $\mathcal{O}'$ becomes consistent?

(S4) **look** at to understand $\mathcal{O} \models \alpha$?

Definition: Let $\mathcal{O}$ be an ontology with $\mathcal{O} \models \alpha$.
Then $\mathcal{J} \subseteq \mathcal{O}$ is a **justification for** α **in** $\mathcal{O}$ if

- $\mathcal{J} \models \alpha$ and
- $\mathcal{J}$ is minimal, i.e., for each $\mathcal{J}' \subsetneq \mathcal{J}$: $\mathcal{J}' \not\models \alpha$

An Example

Consider the following ontology $\mathcal{O}$ with $\mathcal{O} \models C \sqsubseteq \bot$:

$$
\begin{aligned}
\mathcal{O} := \{C &\sqsubseteq D \sqcap E & (1)\\
D &\sqsubseteq A \sqcap \exists r.B_1 & (2)\\
E &\sqsubseteq A \sqcap \forall r.B_2 & (3)\\
B_1 &\sqsubseteq \neg B_2 & (4)\\
D &\sqsubseteq \neg E & (5)\\
G &\sqsubseteq B \sqcap \exists s.C\} & (6)
\end{aligned}
$$

Find a justification for $C \sqsubseteq \bot$ in $\mathcal{O}$.
How many justifications are there?

More about Justifications

Claim: discuss the following claims:

1. for each entailment of $\mathcal{O}$, there exists at least one justification
2. one entailment can have several justifications in $\mathcal{O}$
3. justifications can overlap
4. let $\mathcal{O}'$ be obtained as follows from $\mathcal{O}$ with $\mathcal{O} \models \alpha$:
   - for each justification $\mathcal{J}_i$ of the n justifications for α in $\mathcal{O}$, pick some $\beta_i \in \mathcal{J}_i$
   - set $\mathcal{O}' := \mathcal{O} \setminus \{\beta_1, \ldots, \beta_n\}$

   then $\mathcal{O}' \not\models \alpha$, i.e., $\mathcal{O}'$ is a **repair** of $\mathcal{O}$.
5. due to monotonicity of DLs, if $\mathcal{J}$ is a justification for α and $\mathcal{O}' \supseteq \mathcal{J}$, then $\mathcal{O}' \models \alpha$.
   Hence any repair of α must touch **all** justifications.

A Naive Black-Box Algorithm to Compute Justifications

Let $\mathcal{O} = \{\beta_1, \ldots, \beta_m\}$ be an ontology with $\mathcal{O} \models \alpha$.

Get1Just($\mathcal{O}$, α)
Set $\mathcal{J} := \mathcal{O}$ and **Out** $:= \emptyset$
For each $\beta \in \mathcal{O}$
    If $\mathcal{J} \setminus \{\beta\} \models \alpha$ then
        Set $\mathcal{J} := \mathcal{J} \setminus \{\beta\}$ and **Out** $:=$ **Out** $\cup \{\beta\}$
Return $\mathcal{J}$

Claim:
- loop invariants: $\mathcal{J} \models \alpha$ and $\mathcal{O} = \mathcal{J} \cup$ **Out**
- Get1Just(,) returns 1 justification for α in $\mathcal{O}$
- it requires m entailment tests

Other approaches to computing justifications exists, more performant, glass-box and black-box.

Linking Justifications to our Scenarios

$(\mathbf{S4})$ 1 justification suffices, but which? A good, easy one...how to find?

$(\mathbf{S1\text{-}S3})$ require the computation of **all** justifications, possibly for several entailments

- even for one entailment, search space is exponential

$[(\mathbf{S2})\ ]$ requires even more:

- who wants to look at $x \times 27$ justifications? Where to start?
- A justification $\mathcal{J}$ (for α) is **root** if there is no justification $\mathcal{J}'$ (for β) with $\mathcal{J}' \subsetneq \mathcal{J}$
- start with root justifications, remove/change axioms in them and
- reclassify: you might have repaired several unsatisfiabilities at once!
- Check example on slide 38: both justifications for $C \sqsubseteq \bot$ are root, contained in 2 non-root justifications for $G \sqsubseteq \bot$
- repairing $C \sqsubseteq \bot$ repairs $G \sqsubseteq \bot$

More About Justifications

- recent, optimised implementations
  - behave well in practise
  - can compute all justifications for all atomic entailments of existing, complex ontologies
- recent surveys show that existing ontologies have entailments
  - with large justifications, e.g., over 35 axioms and
  - with numerous justifications, e.g., over 60 justifications for 1 entailment
  - for which justifications can be understood well by domain experts

Beyond Justifications

- there are **hard** justifications that need further explanation
  - e.g., consider $O = \{ \; P \sqsubseteq \neg M, \; RR \sqsubseteq CM, \; CM \sqsubseteq M, \; RR \equiv \exists h.TS \sqcup \forall v.H, \; \exists v.\top \sqsubseteq M \}$ with $\mathcal{O} \models P \sqsubseteq \bot$

$$
\begin{aligned}
P &\sqsubseteq \neg M \\
RR &\sqsubseteq CM \\
CM &\sqsubseteq M \\
RR &\equiv \exists h.TS \sqcup \forall v.H \\
\exists v.\top &\sqsubseteq M
\end{aligned}
$$

  - this has led to investigation of **lemmatised justifications**
- some justification contain **superfluous parts**
  - that distract the user
  - consider example and identify superfluous parts
  - identifying these can help user to focus on the **relevant** parts
  - this has led to investigation of **laconic and precise justifications**

What was left out...

That's it, mostly.

But there is loads more interesting stuff: there are

- other than tableau-based algorithms
- other than standard reasoning problems & services
- ...

Hypertableau in a Nutshell

Observation: in most tableau algorithms/systems, we normally use

- absorption to handle GCIs:
  - essential pre-processing step for reasoner's performance, but
  - can introduce un-necessary disjunctions, e.g., $A \sqcap \exists r.C \sqsubseteq B$
    is a Horn clause $B(x) :- A(x) \wedge \mathcal{R}(x,y) \wedge C(y)$,
    but its absorption $A \sqsubseteq \forall r.\neg C \sqcup B$ involves a disjunction
  - hence what is good in most of the cases is sometimes harmful

  $\rightsquigarrow$ binary/ternary absorption was introduced, but cumbersome
- traditionally, "ancestor" blocking: we only check ancestors for "blocking candidates"

Hypertableau [Motik et. al] avoids both

Hypertableau in a Nutshell

Hypertableau: works in several steps:

1. translate knowledge base (carefully) into a normal form using structural transformation
2. translate the result into FOL clauses of the form

$$\bigwedge R_i(x,y) \wedge \bigwedge A_i(x) \Rightarrow \bigvee S_i(x,y) \vee \bigvee B_i(x) \vee \bigvee y_i \simeq y_j \ldots$$

3. apply hypertableau rules to an ABox, most importantly

   if ABox matches body of a clause, then add head

   (other rules to deal with $\simeq$, $\geq$, and $\bot$)

- use "anywhere" blocking: consider **all** "older" individuals as blocking candidates

Absorption superfluous since built-in, handles Horn KBs in a deterministic way.

Another Approach: Automata-based Algorithms for DLs – Motivation

Observation: to obtain decision procedure, we need to ensure **termination**.

For tableau algorithms, we

- use blocking to ensure termination
- use unravelling to construct tree models

Also, they are often **non-deterministic** (e.g., $\sqcup$-rule). Hence

- ensuring and proving termination can be hard work
- proving soundness as well
- obtaining optimal algorithms can be difficult for deterministic complexity classes
- implementing requires backtracking/backjumping: implementer must work hard as well

Another Approach: Automata-based Algorithms for DLs – Sketch

A recipe for an automata-based algorithm:

1. Learn about (alternating) (two-way) (counting) (tree) automata and pick a **suitable** class X of automata, i.e., suitable for your logic & with decidable emptiness problem
2. Prove that your logic has **a tree model property**, i.e., the right one for X
3. Construct, for $KB = (C_0, \mathcal{T}, \ldots)$, an X automaton $\mathcal{A}_{KB}$ such that

$$L(\mathcal{A}_{KB}) = \{\tau \mid \tau \text{ is a tree model of } KB\}.$$

4. Check that $|\mathcal{A}_{KB}|$ is finite $\leadsto$ decidability of KB satisfiability
5. Check that $|\mathcal{A}_{KB}|$ is $\mathcal{O}(\ldots |KB|)$ and use known complexity of testing emptiness of X automata to obtain upper bound for KB satisfiability

What was Left out on tableau algorithms for expressive DLs

➔ query answering: in addition to "retrieve all ABox individuals a with $\mathcal{O} \models a : C$, more powerful query languages are considered

➔ here: $\mathcal{ALCQI}$,
in SOTA DL reasoners FaCT ++, Pellet, and Racer: $\mathcal{SROIQ}$, $\mathcal{ALCQI}$ plus

- **transitive roles:** if **Trans**(R), then $R^{\mathcal{I}}$ must be transitive,
- **role hierarchies:** if $R \sqsubseteq S$, then $\mathcal{I}$ must satisfy $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$,
- **complex role inclusions:** e.g., $\mathtt{owns} \circ \mathtt{hasPart} \sqsubseteq \mathtt{owns}$
- **nominals:** individual names can be used as (singleton) concepts
- etc.

$\rightsquigarrow$ the DL underlying **OWL2**

. . . extension of $\mathcal{ALCQI}$ tableau algorithm and proofs tedious and sometimes difficult (nominals)

Other related interesting things

- ➔ **concrete domains** to describe "concrete" properties such as age, height, weight, etc.
  ...extension of $\mathcal{ALCQI}$ tableau algorithm only possible for restricted cases
- ➔ combining DLs and rules
- ➔ combining DLs and description graphs for the representation of structured objects
- ➔ **fast** (sub-Boolean) DLs
  - different compromise for trade-off between expressive power and comp. complexity
  - $\mathcal{EL}$++ designed for **huge** TBoxes: SNOMED CT defines approx. 400,000 concepts
  - DL-LITE designed for huge ABoxes/data
- ➔ ontology editors such as **SWOOP** or **Protégé 4** that use a DL reasoner
- ➔ computational complexity of DLs
- ➔ modules of ontologies for re-use, etc.

That's it!

I hope you have enjoyed the class
and
learned a lot.

I will be available for further questions,
in person, via email or Blackboard.

Thanks for your attention!