Description Logics:
an Introductory Course on a Nice Family of Logics

Day 2: Tableau Algorithms

Uli Sattler

Warm up

Which of the following subsumptions hold?

r some (A and B)	is subsumed by	r some A
$\exists r.(A \sqcap B)$	$\sqsubseteq$	$\exists r.A$
(r some A) and (r only B)	is subsumed by	r some B
$\exists r.A \sqcap \forall r.B$	$\sqsubseteq$	$\exists r.B$
r only (A and not A)	is subsumed by	r only B
$\forall r.(A \sqcap \neg A)$	$\sqsubseteq$	$\forall r.B$
r some (r only A)	is subsumed by	r some (r some (A or not A))
$\exists r.(\forall r.A)$	$\sqsubseteq$	$\exists r.(\exists r.(A \sqcup \neg A)$
r only (A and B)	is subsumed by	(r only A) and (r only B)
$\forall r.(A \sqcap B)$	$\sqsubseteq$	$\forall r.A \sqcap \forall r.B$
r some B	is subsumed by	r only B
$\exists r.B$	$\sqsubseteq$	$\forall r.A$

Today

- relationship between standard DL reasoning problems
- a tableau algorithm to decide consistency of $\mathcal{ALC}$ ontologies and all other standard DL reasoning problems
- a proof of its correctness
- with some model properties
- some optimisations
- some extensions
  - inverse roles
  - (sketch) number restrictions
- some discussions
- ...loads of stuff: **ask** if you have a question!

Standard DL Reasoning Problems

Given an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$,

- is $\mathcal{O}$ consistent? $\mathcal{O} \models \top \sqsubseteq \bot$?
- is $\mathcal{O}$ coherent? is there concept name A with $\mathcal{O} \models A \sqsubseteq \bot$?
- compute class hierarchy! for all concept names A, B: $\mathcal{O} \models A \sqsubseteq B$?
- classify individuals! for all concept names A, individual names b: $\mathcal{O} \models b \colon B$?

Theorem 2 Let $\mathcal{O}$ be an ontology and a an individual name **not** in $\mathcal{O}$. Then

1. C is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O} \cup \{a \colon C\}$ is consistent
2. $\mathcal{O}$ is coherent iff, for each concept name A,
   $\mathcal{O} \cup \{a \colon A\}$ is consistent
3. $\mathcal{O} \models A \sqsubseteq B$ iff $\mathcal{O} \cup \{a \colon (A \sqcap \neg B)\}$ is **not** consistent
4. $\mathcal{O} \models b \colon B$ iff $\mathcal{O} \cup \{b \colon \neg B\}$ is **not** consistent

➥ a decision procedure to solve consistency decides **all** standard DL reasoning problems

Decision Procedure

- A **problem** is a set $P \subseteq M$
  - e.g., M is the set of all $\mathcal{ALC}$ ontologies,
  - $P \subseteq M$ is the set of all **consistent** $\mathcal{ALC}$ ontologies
  - ...and **the problem** P is to decide whether, for a given $m \in M$, we have $m \in P$

- An algorithm is a **decision procedure** for a problem $P \subseteq M$ if it is
  - **sound** for P: if it answers "$m \in P$", then $m \in P$
  - **complete** for P: if $m \in P$, then it answers "$m \in P$"
  - **terminating**: it stops after finitely many steps on any input $m \in M$

Why does "sound and complete" not suffice for being a decision procedure?

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

For now:

- $\mathcal{ALC}$: $\sqcap, \sqcup, \neg, \exists r.C, \forall r.C$
- an algorithm to decide consistency of an **ontology**

The algorithm decides "Is $\mathcal{O}$ consistent" by trying to construct a **model** $\mathcal{I}$ for $\mathcal{O}$:

- if successful, $\mathcal{O}$ is consistent: "look, here is a (description of a) model"
- otherwise, no model exists – provably (we were not simply too lazy to find it)

Algorithm works on a **set of ABoxes**:

- intialised with a singleton set $\mathcal{S} = \{\mathcal{A}\}$ when started with $\mathcal{O} = (\mathcal{T}, \mathcal{A})$
- ABoxes are extended by rules to make constraints on models of $\mathcal{O}$ explicit
- $\mathcal{O}$ is consistent if, for (at least) one of the ABoxes $\mathcal{A}'$ in $\mathcal{S}$, $(\mathcal{T}, \mathcal{A}')$ is consistent

Negation Normal Form

Technical: we say C and D are **equivalent**, written $C \equiv D$, if they mutually subsume each other.

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

$$\begin{array}{rclcrcl}
\neg(C \sqcap D) & \equiv & \neg C \sqcup \neg D & \quad & \neg(C \sqcup D) & \equiv & \neg C \sqcap \neg D \\
\neg(\exists R.C) & \equiv & (\forall R.\neg C) & \quad & \neg(\forall R.C) & \equiv & (\exists R.\neg C)
\end{array}$$

Lemma: Let C be an $\mathcal{ALC}$ concept. Then $C \equiv$ **NNF**(C).

From now on, all concepts in GCIs and concept assertions are assumed to be in NNF, and we use $\dot\neg C$ to denote the **NNF**$(\neg C)$.

The algorithm

- works on **sets of ABoxes** $\mathcal{S}$
- starts with a singleton set $\mathcal{S} = \{\mathcal{A}\}$ when started with $\mathcal{O} = (\mathcal{T}, \mathcal{A})$
- applies **rules** that infer constraints on models of $\mathcal{O}$
- a rule is applied to **some** $\mathcal{A} \in \mathcal{S}$; its application replaces $\mathcal{A}$ with one or two ABoxes
- answers "$\mathcal{O}$ is consistent" if rule application leads to an ABox $\mathcal{A}$ that is
  - **complete**, i.e., to which no more rules apply and
  - **clash-free**, i.e., $\{a\colon A,\ a\colon \neg A\} \not\subseteq \mathcal{A}$, for any a, A
- for optimisation, we can avoid applying rules to ABoxes containing a clash

Using the tableau algorithm for $\mathcal{ALC}$ ontologies

Following Theorem 2, we can use the algorithm to test

- satisfiability of a concept C by starting it with $\{a\colon C\}$
- satisfiability of a concept C wr.t. $\mathcal{O}$ by starting it with $\mathcal{O} \cup \{a\colon C\}$ (a not in $\mathcal{O}$)
- subsumption $C \sqsubseteq D$ by starting it with $\{a\colon (C \sqcap \neg D)\}$
- subsumption $C \sqsubseteq D$ wr.t. $\mathcal{O}$ by starting it with $\mathcal{O} \cup \{a\colon (C \sqcap \neg D)\}$ (a not in $\mathcal{O}$)
- whether b is an instance of C w.r.t. $\mathcal{O}$ by starting it with $\mathcal{O} \cup \{b\colon \neg C\}$

- ...and interpreting the results according to Theorem 2.

Preliminary Tableau Expansion Rules for $\mathcal{ALC}$

$\sqcap$-rule: if $a\colon C_1 \sqcap C_2 \in \mathcal{A}$ and $\{a\colon C_1, a\colon C_2\} \not\subseteq \mathcal{A}$

then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon C_1, a\colon C_2\}$

$\sqcup$-rule: if $a\colon C_1 \sqcup C_2 \in \mathcal{A}$ and $\{a\colon C_1, a\colon C_2\} \cap \mathcal{A} = \emptyset$

then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon C_1\}$ and $\mathcal{A} \cup \{a\colon C_2\}$

$\exists$-rule: if $a\colon \exists s.C \in \mathcal{A}$ and there is no b with $\{(a,b)\colon s,\ b\colon C\} \subseteq \mathcal{A}$

then create a new individual name c and
replace $\mathcal{A}$ with $\mathcal{A} \cup \{(a,c)\colon s,\ c\colon C\}$

$\forall$-rule: if $\{a\colon \forall s.C,\ (a,b)\colon s\} \subseteq \mathcal{A}$ and $b\colon C \notin \mathcal{A}$

then replace $\mathcal{A}$ with $\mathcal{A} \cup \{b\colon C\}$

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$ and $a\colon (\dot{\neg} C \sqcup D) \notin \mathcal{A}$ for a in $\mathcal{A}$,

then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon (\dot{\neg} C \sqcup D)\}$

Tableau Algorithm for $\mathcal{ALC}$: Observations

- We only apply rules if their application does **“something new”**
- The $\sqcup$-rule is the only one to replace an ABox with more than one other
- To understand the GCI-rule, convince yourself that

$$\mathcal{I} \text{ satisfies a GCI } C \sqsubseteq D \text{ iff, for each } e \in \Delta^{\mathcal{I}}, \text{ we have } e \notin C^{\mathcal{I}} \text{ or } e \in D^{\mathcal{I}}$$

  – and $e \notin C^{\mathcal{I}}$ is the case iff $e \in (\neg C)^{\mathcal{I}}$
- The GCI-rule adds a disjunction per individual and GCI $\Rightarrow$ this is
  – **bad**, and
  – **stupid** for GCIs with a concept name on its left hand side (why?)

$\Rightarrow$ we add an **abbreviated** GCI rule:

GCI-2-rule: if B is a concept name, $a\colon F \notin \mathcal{A}$ for $a\colon B \in \mathcal{A}$ and $B \sqsubseteq F \in \mathcal{T}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon F\}$

- If $\mathcal{A}$ is replaced with $\mathcal{A}'$, then $\mathcal{A} \subseteq \mathcal{A}'$

Example: apply the tableau algorithm to $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ with

$$\begin{aligned} \mathcal{T} = \{A &\sqsubseteq B \sqcap \exists r.G \sqcap \forall r.C, \\ E &\sqsubseteq A \sqcap H \sqcap \forall r.F, \\ G &\sqsubseteq E \sqcap P, \\ H &\sqsubseteq E \sqcup \forall r.\neg C\} \end{aligned} \qquad \begin{aligned} A = \{\ & a\colon A,\ b\colon E, \\ & (a,c)\colon r,\ (b,c)\colon r, \\ & c\colon G\} \end{aligned}$$

Termination of our Tableau Algorithm for $\mathcal{ALC}$

As is, the tableau algorithm does not terminate:

Example: apply the tableau algorithm to $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ with $\mathcal{T} = \{A \sqsubseteq \exists r.A\}$ and $A = \{a\colon A\}$.

To ensure termination, use **blocking**: each rule is only applicable to an individual a in an ABox $\mathcal{A}$ if there is no other individual b with

$$\{C \mid a\colon C \in \mathcal{A}\} \subseteq \{C \mid b\colon C \in \mathcal{A}\}.$$

In case we have

- a freshly introduced individual (i.e., not present in input ontology) a,
- an individual b with
  - $\{C \mid a\colon C \in \mathcal{A}\} \subseteq \{C \mid b\colon C \in \mathcal{A}\}$,
  - b is older than a (i.e., was created earlier than a)

we say b **blocks** a and we say a is **blocked**.

Tableau Expansion Rules for $\mathcal{ALC}$

$\sqcap$-rule: if $a\colon C_1 \sqcap C_2 \in \mathcal{A}$, a **is not blocked**, and $\{a\colon C_1, a\colon C_2\} \not\subseteq \mathcal{A}$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon C_1, a\colon C_2\}$

$\sqcup$-rule: if $a\colon C_1 \sqcup C_2 \in \mathcal{A}$, a **is not blocked**, and $\{a\colon C_1, a\colon C_2\} \cap \mathcal{A} = \emptyset$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon C_1\}$ and $\mathcal{A} \cup \{a\colon C_2\}$

$\exists$-rule: if $a\colon \exists s.C \in \mathcal{A}$, a **is not blocked**, and there is no b with $\{(a,b)\colon s,\ b\colon C\} \subseteq \mathcal{A}$
then create a new individual c and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(a,c)\colon s,\ c\colon C\}$

$\forall$-rule: if $\{a\colon \forall s.C,\ (a,b)\colon s\} \subseteq \mathcal{A}$, a **is not blocked**, and $b\colon C \notin \mathcal{A}$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{b\colon C\}$

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$, a **is not blocked**, and
if C is a concept name, $a\colon C \in \mathcal{A}$ but $a\colon D \notin \mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon D\}$

else if $a\colon (\dot{\neg} C \sqcup D) \notin \mathcal{A}$ for a in $\mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon (\dot{\neg} C \sqcup D)\}$

Termination of our Tableau Algorithm for $\mathcal{ALC}$

Convince yourself that, for the given example, the tableau algorithm terminates:

Example: apply the tableau algorithm to $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ with $\mathcal{T} = \{A \sqsubseteq \exists r.A\}$ and $A = \{a\colon A\}$.

...now for the general case!

Properties of our tableau algorithm

Lemma 3: Let $\mathcal{O}$ an $\mathcal{ALC}$ ontology in NNF. Then

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

Corollary 1:

1. Our tableau algorithm **decides consistency** of $\mathcal{ALC}$ ontologies.
2. Satisfiability (and subsumption) of $\mathcal{ALC}$ concepts is decidable in **PSpace**.
3. Consistency of $\mathcal{ALC}$ ontologies is decidable in **ExpSpace**.
4. $\mathcal{ALC}$ ontologies have the **finite model property**
   i.e., every consistent ontology has a **finite** model.
5. $\mathcal{ALC}$ ontologies have the **tree model property**
   i.e., every consistent ontology has a **tree** model.

Proof of Lemma 3.1: Termination

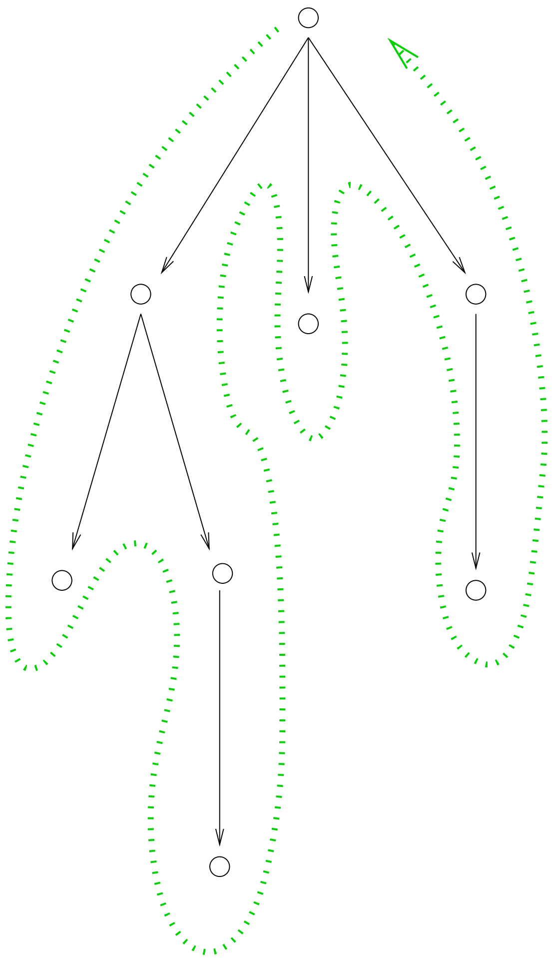
Let **sub**$(\mathcal{O})$ be the set of all subconcepts of concepts occurring in $\mathcal{A}$ together with all subconcepts of $\dot{\neg}C \sqcup D$ for each $C \sqsubseteq D \in \mathcal{T}$.

(1) **Termination** is a consequence of these observations:

1. a rule replaces one ABox with at most two ABoxes
2. the ABoxes are constructed in a **monotonic way**, i.e., each rule adds assertions, nothing is removed
3. concept assertions added are restricted to **sub**$(\mathcal{O})$ and
$$\#\,\mathbf{sub}(\mathcal{O}) \leq \Sigma_{C \sqsubseteq D \in O}(2 + |C| + |D|) + \Sigma_{a:\,C \in O}|C|$$
because, at each position in a concept, at most one sub-concept starts
4. due to blocking, there can be at most $2^{\#\,\mathsf{sub}(\mathcal{O})}$ individuals in each ABox: if $\{C \mid a\colon C \in \mathcal{A}\} \subseteq \{C \mid b\colon C \in \mathcal{A}\}$, a is blocked and no rules are applied to a.

Eventually, all ABoxes will be complete (and possibly have a clash), and the algorithm terminates.

If we start the algorithm with $\{a : C\}$
to test satisfiability of C, and
construct ABox in non-deterministic depth-first manner
rather than constructing set of ABoxes
so that we only consider a single ABox and
re-use space for branches already visited,
mark $b : \exists R.C \in \mathcal{A}$ with "todo" or "done"



we can run tableau algorithm (even without blocking) in **polynomial space**:

- ABox is of depth bounded by $|C|$, and
- we keep only a single branch in memory at any time.

Regarding Corollary 1.3

If we start the algorithm with $\mathcal{O}$ to test its consistency, and
construct ABox in non-deterministic depth-first manner
rather than constructing set of ABoxes
so that we only consider a single ABox

we can run tableau algorithm in **exponential space**:

- number of individuals in ABox is bounded by $2^{\#\,\mathsf{sub}(\mathcal{O})}$

This is **not** optimal: we will see tomorrow that consistency of $\mathcal{ALC}$ ontologies is decidable in **exponential time**, in fact **ExpTime**-complete.

Proof of Lemma 3.2: Soundness

(2) Let $\mathcal{A}_f$ be a complete & clash-free ABox generated for $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, and let $\mathcal{B}_f$ be $\mathcal{A}_f$ without assertions involving blocked individuals. Define an interpretation $\mathcal{I}$ as follows:

$$
\begin{aligned}
\Delta^\mathcal{I} &:= \{x \mid x \text{ is an individual in } \mathcal{B}_f\} \\
A^\mathcal{I} &:= \{x \in \Delta^\mathcal{I} \mid x\colon A \in \mathcal{B}_f\} \quad \text{for concept names } A \\
r^\mathcal{I} &:= \{(x,y) \in \Delta^\mathcal{I} \times \in \Delta^\mathcal{I} \mid \; (x,y)\colon r \in \mathcal{B}_f \text{ or} \\
&\qquad\qquad\qquad\qquad\quad (x,y')\colon r \in \mathcal{A}_f \text{ and } y \text{ blocks } y' \text{ in } \mathcal{A}_f\}
\end{aligned}
$$

and show, by induction on structure of concepts:

(C1) $x\colon D \in \mathcal{B}_f$ implies $x \in D^\mathcal{I}$

(C2) $C \sqsubseteq D \in \mathcal{T}$ implies $C^\mathcal{I} \subseteq D^\mathcal{I}$

- ➜ $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{B}_f)$ ($\mathcal{I}$ satisfies all role assertions by definition)
- ➜ $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{A})$ because $\mathcal{A} \subseteq \mathcal{B}_f$
- ➜ $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is consistent

Proof of Lemma 3.2: Soundness II

$$
\begin{aligned}
\Delta^{\mathcal{I}} &:= \{x \mid x \text{ is an individual in } \mathcal{B}_f\} \\
A^{\mathcal{I}} &:= \{x \in \Delta^{\mathcal{I}} \mid x\colon A \in \mathcal{B}_f\} \quad \text{for concept names } A \\
r^{\mathcal{I}} &:= \{(x,y) \in \Delta^{\mathcal{I}} \times \in \Delta^{\mathcal{I}} \mid \ (x,y)\colon r \in \mathcal{B}_f \text{ or} \\
& \qquad\qquad\qquad\qquad\qquad\quad (x,y')\colon r \in \mathcal{B}_f \text{ and } y \text{ blocks } y'\}
\end{aligned}
$$

Show, by induction on structure of concepts: (C1) $x\colon D \in \mathcal{B}_f$ implies $x \in D^{\mathcal{I}}$

- for concept names D: by definition of $\mathcal{I}$
- for negated concept names D: due to clash-freeness and induction
- for conjunctions/disjunctions/existential restrictions/universal restrictions D: due to completeness and by induction

Proof of Lemma 3.2: Soundness III

$$
\begin{array}{rcl}
\Delta^\mathcal{I} & := & \{x \mid x \text{ is an individual in } \mathcal{B}_f\} \\
A^\mathcal{I} & := & \{x \in \Delta^\mathcal{I} \mid x\colon A \in \mathcal{B}_f\} \quad \text{for concept names } A \\
r^\mathcal{I} & := & \{(x,y) \in \Delta^\mathcal{I} \times \in \Delta^\mathcal{I} \mid \ (x,y)\colon r \in \mathcal{B}_f \text{ or} \\
& & \qquad (x,y')\colon r \in \mathcal{B}_f \text{ and } y \text{ blocks } y'\}
\end{array}
$$

(C2): $C \sqsubseteq D \in \mathcal{T}$ implies $C^\mathcal{I} \subseteq D^\mathcal{I}$

This is an immediate consequence of

- $\Delta^\mathcal{I}$ being a set of individual names in $\mathcal{A}_f$,
- $\mathcal{A}_f$ being complete $\Rightarrow$ the GCI-rule is not applicable $\Rightarrow$ if $C \sqsubseteq D \in \mathcal{T}$:
  - if C is a concept name $x \in C^\mathcal{I}$, then $x\colon C \in \mathcal{B}_f$, and thus $x\colon D \in \mathcal{B}_f$
  - else, $x\colon (\dot\neg C \sqcup D) \in \mathcal{B}_f$
- (C1)

Proof of Lemma 3.3: Completeness

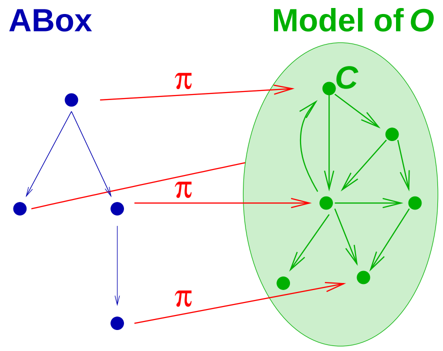
(3) Let $\mathcal{O}$ be consistent, and let $\mathcal{I}$ be a model of $\mathcal{O}$.

Use $\mathcal{I}$ to identify a clash-free & complete ABox:

Inductively define a total mapping π :

start with $\pi(a) = a^{\mathcal{I}}$, and show that

each rule can be applied such that $(*)$ is preserved



$$(*) \text{ if } x : C \in \mathcal{A}, \text{ then } \pi(x) \in C^{\mathcal{I}}$$
$$\text{if } (x, y) : r \in \mathcal{A}, \text{ then } \langle \pi(x), \pi(y) \rangle \in r^{\mathcal{I}}$$

- easy for $\sqcap$-, $\forall$-, and the GCI-rule,
- for $\exists$-rule, we need to extend π to the newly created r-successor
- for $\sqcup$-rule, if $C_1 \sqcup C_2 : x \in \mathcal{A}$, $(*)$ implies that $\pi(x) \in (C_1 \sqcup C_2)^{\mathcal{I}}$
  $\rightsquigarrow$ we can choose $\mathcal{A}_i = \mathcal{A} \cup \{x : C_i\}$ with $\pi(x) \in C_i^{\mathcal{I}}$ and thus preserve $(*)$

$\rightsquigarrow$ easy to see: $(*)$ implies that ABox is **clash-free**

Proof of Lemma 3: Harvest

Consider the model $\mathcal{I}$ constructed for a clash-free, complete ABox in soundness proof:

$\mathcal{I}$ is

- **finite** because ABox has finitely many individuals
- a **tree** if blocking has **not** occurred
- not a **tree** if blocking has occurred:
  but it can be **unravelled** into an (infinite) tree model

Hence we get Corollary 1.4 and 1.5 for (almost) free from our proof:

Corollary 1:

4. $\mathcal{ALC}$ ontologies have the **finite model property**
   i.e., every consistent ontology has a **finite** model.
5. $\mathcal{ALC}$ ontologies have the **tree model property**
   i.e., every consistent ontology has a **tree** model.

A tableau algorithm for $\mathcal{ALC}$ ontologies: Summary

The tableau algorithm presented here

- **decides** consistency of $\mathcal{ALC}$ ontologies, and thus also
- all other standard reasoning problems
- uses **blocking** to ensure termination, and
- can be implemented as such or
  using a **non-deterministic** alternative for the $\sqcup$-rule and backtracking.
- in the worst case, it builds ABoxes that are exponential in the size of the input.
  Hence it runs in (worst case) ExpSpace,
- can be implemented in various ways,
  - order/priorities of rules
  - data structure
  - etc.
- is amenable to optimisations...

Implementing the $\mathcal{ALC}$ Tableau Algorithm

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

Implementing the $\mathcal{ALC}$ Tableau Algorithm

Optimisations are crucial

concern every aspect of the algorithm

help in “many” cases (which?)

are implemented in various **DL reasoners**

e.g., FaCT++, Pellet, RacerPro

In the following: a selection of some vital optimisations

Optimising the $\mathcal{ALC}$ Tableau Algorithm: Classification

Reasoners provides service **“classify all concept names $\mathcal{T}$”**, i.e.,

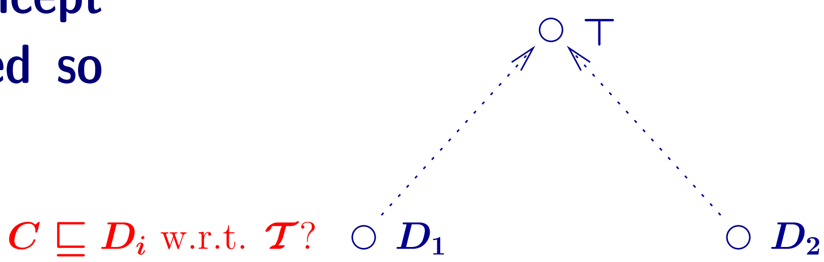
for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

$\rightsquigarrow$ test consistency of $\mathcal{T} \cup \{a\colon (C \sqcap \neg D)\}$

$\rightsquigarrow$ n^2 consistency tests!

Goal: reduce number of consistency tests when classifying TBox

Idea 1: “trickle” new concept C into hierarchy computed so far



Optimising the $\mathcal{ALC}$ Tableau Algorithm: Classification II

Reasoners provides service **"classify all concept names $\mathcal{T}$"**, i.e.,

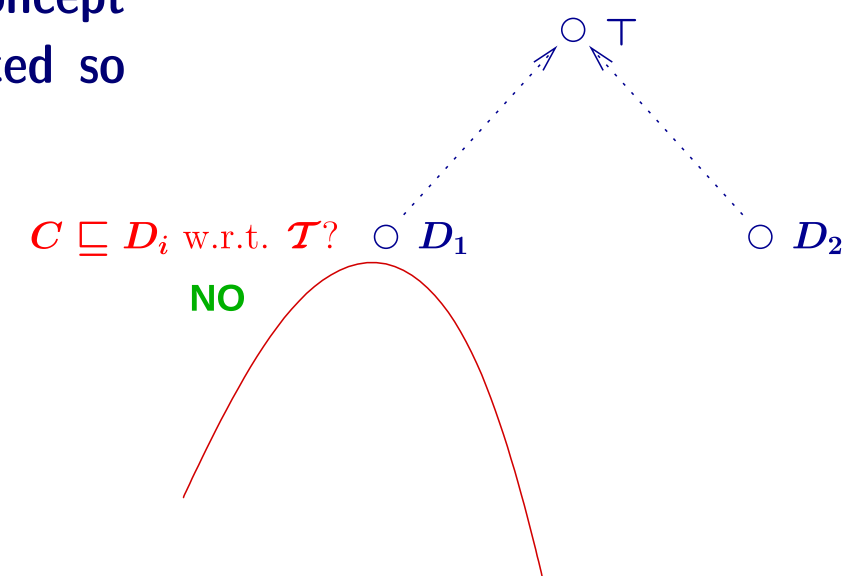
for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

$\rightsquigarrow$ test consistency of $\mathcal{T} \cup \{a\colon (C \sqcap \neg D)\}$

$\rightsquigarrow$ n^2 consistency tests!

Goal: reduce number of consistency tests when classifying TBox

Idea 1: "trickle" new concept C into hierarchy computed so far



Optimising the $\mathcal{ALC}$ Tableau Algorithm: Classification III

Reasoners provides service **"classify all concept names $\mathcal{T}$"**, i.e.,

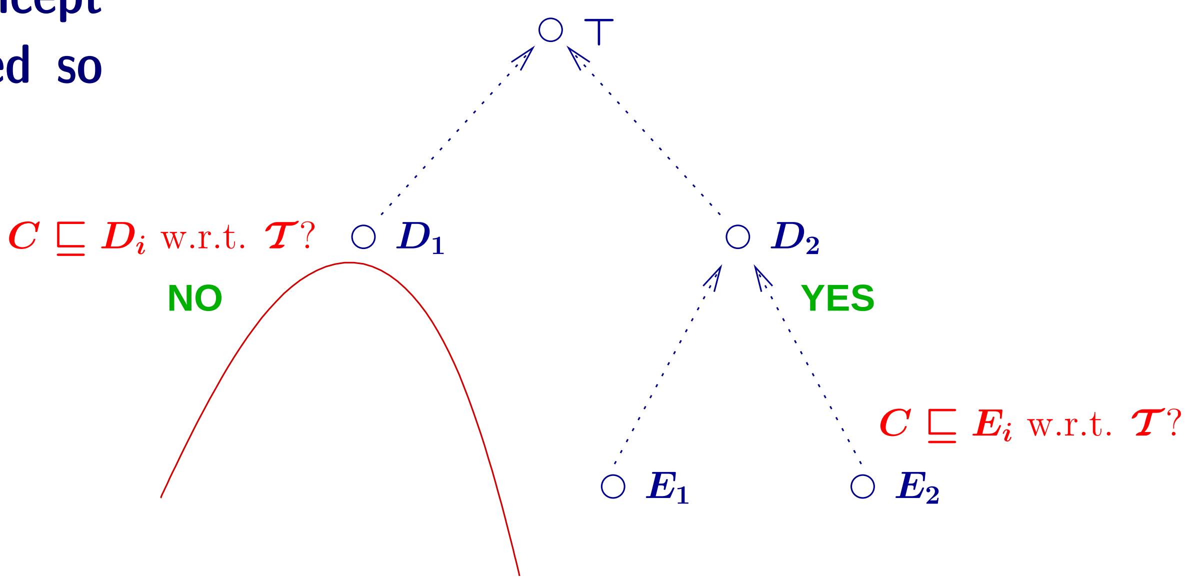
for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

$\rightsquigarrow$ test consistency of $\mathcal{T} \cup \{a : (C \sqcap \neg D)\}$

$\rightsquigarrow$ n^2 consistency tests!

Goal: reduce number of consistency tests when classifying TBox

Idea 1: "trickle" new concept C into hierarchy computed so far



Optimising the $\mathcal{ALC}$ Tableau Algorithm: Classification IV

Reasoners provides service **"classify all concept names $\mathcal{T}$"**, i.e.,

for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

$\rightsquigarrow$ test consistency of $\mathcal{T} \cup \{a\colon (C \sqcap \neg D)\}$

$\rightsquigarrow$ n^2 consistency tests!

Goal: reduce number of consistency tests when classifying TBox

Idea 2:

- maintain graph with a node for each concept name
- edges representing subsumption, disjointness ($\mathcal{T} \models A \sqsubseteq \neg B$), and non-subsumption
- initialise graph with all "obvious" information in $\mathcal{T}$
- to avoid testing subsumption, exploit
  - all info in ABox during tableau algorithm to update graph
  - transitivity of subsumption and its interaction with disjointness

Optimising the $\mathcal{ALC}$ Tableau Algorithm: Absorption

Remember: for $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, where no C_i is a concept name, each individual x will have n disjunctions $x\colon (\dot{\neg} C_i \sqcup D_i)$ due to

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$, a is not blocked, and
if C is a concept name, $a\colon C \in \mathcal{A}$ but $a\colon D \notin \mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon D\}$
else if $a\colon (\dot{\neg} C \sqcup D) \notin \mathcal{A}$ for a in $\mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon (\dot{\neg} C \sqcup D)\}$

Problem: high degree of **choice** and huge **search space**
blows up set of ABoxes

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

e.g., $\mathtt{Human} \sqcap \ldots \sqsubseteq C$ or $\mathtt{Device} \sqcap \ldots \sqsubseteq C$

Optimising the $\mathcal{ALC}$ Tableau Algorithm: Absorption

Idea: **localise** GCIs to concept **names** by transforming
$A \sqcap X \sqsubseteq C$ into equivalent $A \sqsubseteq \neg X \sqcup C$
e.g., $\mathtt{Human} \sqcap \exists\mathtt{owns.Pet} \sqsubseteq C$ becomes $\mathtt{Human} \sqsubseteq \neg\exists\mathtt{owns.Pet} \sqcup C$

For "absorbed" $\mathcal{T} = \{A_i \sqsubseteq D_i \mid 1 \leq i \leq n_1\} \cup \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n_2\}$
the second, non-deterministic choice in GCI-rule is taken only n_2 times.

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$, a is not blocked, and
if C is a concept name, $a\colon C \in \mathcal{A}$ but $a\colon D \notin \mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon D\}$
else if $a\colon (\dot{\neg} C \sqcup D) \notin \mathcal{A}$ for a in $\mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon (\dot{\neg} C \sqcup D)\}$

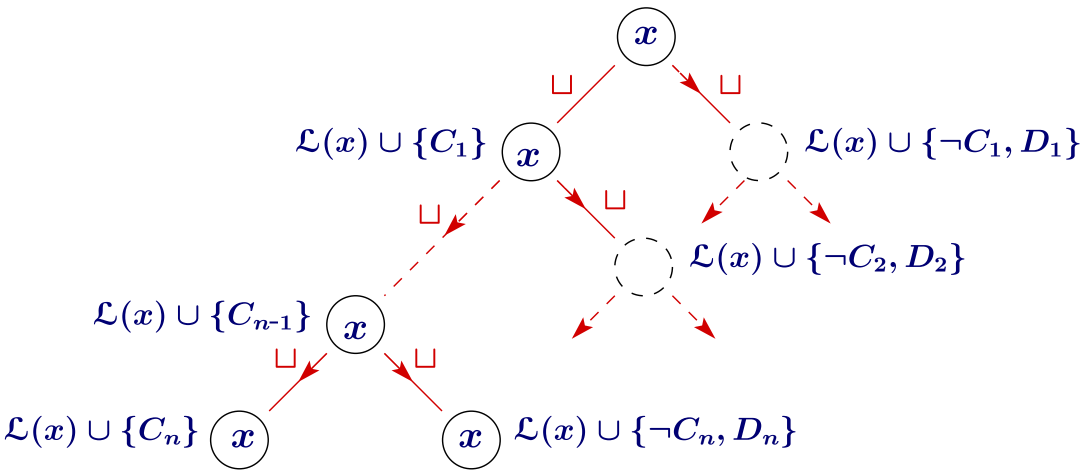
Observations: If no GCI is absorbable, nothing changes
Each absorption saves 1 disjunction **per individual outside** A_i,
in the best case, this avoids **almost all** disjunctions from TBox axioms!

Optimising the $\mathcal{ALC}$ 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

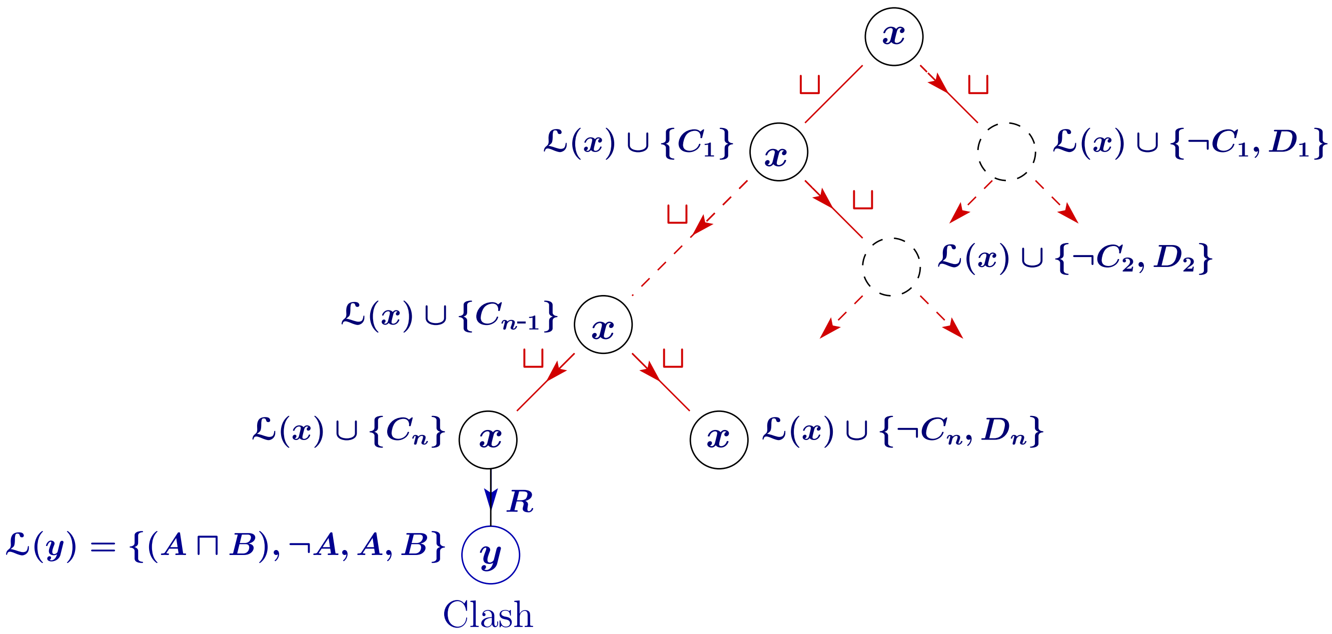
Example $x : \exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A$



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

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

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

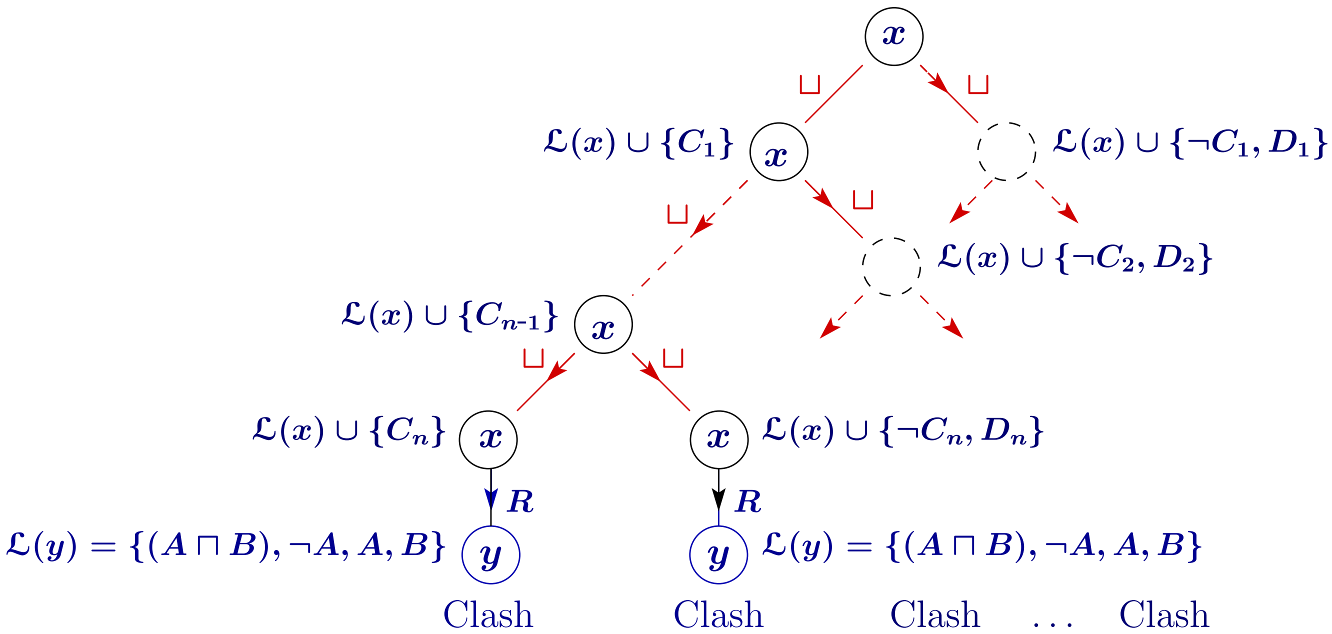


Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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

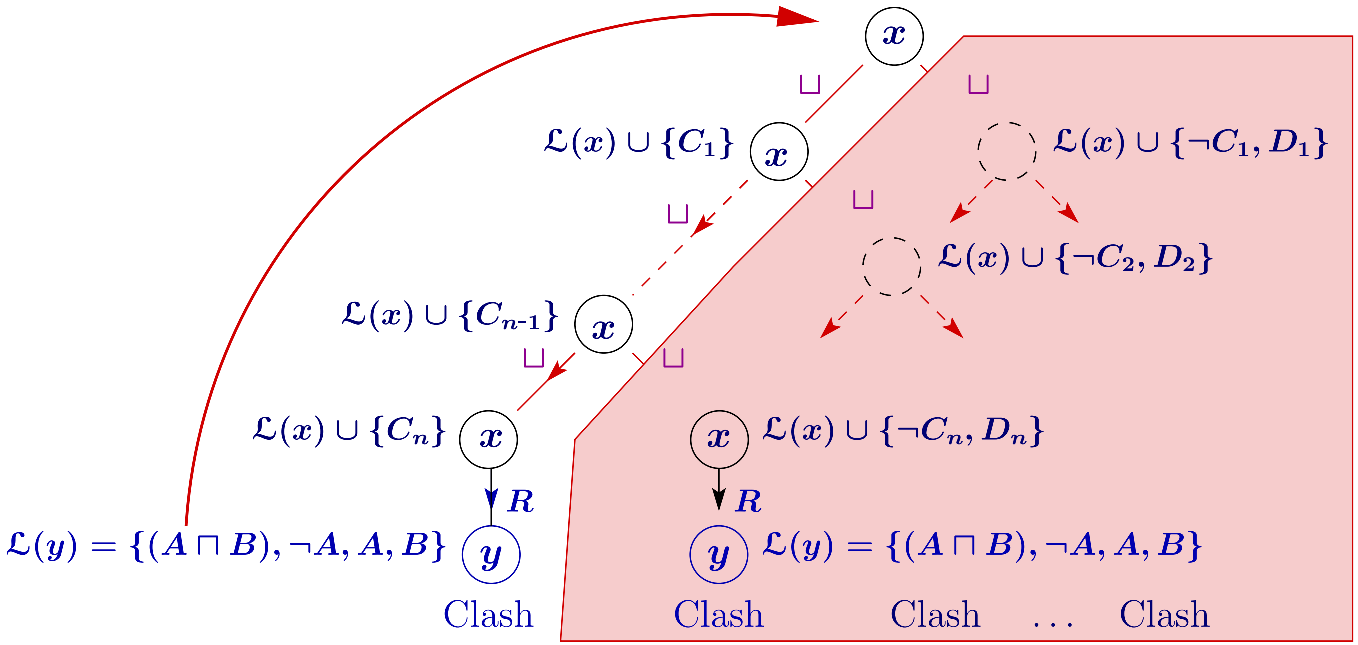


Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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 $\mathcal{ALC}$ Tableau Algorithm: SAT Optimisations

Finally: $\mathcal{ALC}$ 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?

Stuff seen today so far

- standard reasoning problems for $\mathcal{ALC}$ ontologies
- and their relationship & reducibility
- tableau algorithm for $\mathcal{ALC}$ ontologies that
  - requires blocking for termination
  - is a decision procedure for all standard $\mathcal{ALC}$ reasoning problems
  - works on a set of ABoxes or in a non-deterministic way with backtracking
  - is implemented in state-of-the-art reasoners
- proof of soundness, completeness, and termination of tableau algorithm
- some optimisations

Next: extension to more expressive DLs

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

Example: Does $\forall$parent.$\forall$child.Blond $\sqsubseteq$ Blond w.r.t. $\mathcal{T} = \{\top \sqsubseteq \exists$parent.$\top\}$?

Motivation: with inverse roles, one can use both

has-child	and is-child-of
has-part	and is-part-of
...	...

and capture their interaction

$\mathcal{ALCI}$ is the extension of $\mathcal{ALC}$ with **inverse roles** R^- in the place of role names:

$$(r^-)^\mathcal{I} := \{\langle y, x\rangle \mid \langle x, y\rangle \in r^\mathcal{I}\}.$$

Example: Does $\forall$parent.$\forall$parent$^-$.Blond $\sqsubseteq$ Blond w.r.t. $\mathcal{T} = \{\top \sqsubseteq \exists$parent.$\top\}$?

Is $\exists r.\exists s.A$ satisfiable w.r.t. $\mathcal{T} = \{\top \sqsubseteq \forall s^-.\forall r^-.\neg A\}$?

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

Modifications necessary to handle inverse roles: consider role assertions in both directions

① introduce $\mathbf{Inv}(r) = \begin{cases} r^- & \text{if } r \text{ is a role name} \\ s & \text{if } r = s^- \end{cases}$

② call y an r-**neighbour** of x if either $(x, y) : r \in \mathcal{A}$ or $(y, x) : \mathbf{Inv}(r) \in \mathcal{A}$

③ substitute "$(x, y) : r \in \mathcal{A}$" in the $\forall$- and $\exists$-rule with "has an r-neighbour y"...

Tableau Expansion Rules for $\mathcal{ALCI}$

$\sqcap$-rule: if $a\colon C_1 \sqcap C_2 \in \mathcal{A}$, a is not blocked, and $\{a\colon C_1, a\colon C_2\} \not\subseteq \mathcal{A}$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon C_1, a\colon C_2\}$

$\sqcup$-rule: if $a\colon C_1 \sqcup C_2 \in \mathcal{A}$, a is not blocked, and $\{a\colon C_1, a\colon C_2\} \cap \mathcal{A} = \emptyset$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon C_1\}$ and $\mathcal{A} \cup \{a\colon C_2\}$

$\exists$-rule: if $a\colon \exists s.C \in \mathcal{A}$, a is not blocked, and there is no s-**neighbour** b of a with $b\colon C \in \mathcal{A}$
then create a new individual c and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(a,c)\colon s,\ c\colon C\}$

$\forall$-rule: if $a\colon \forall s.C \in \mathcal{A}$, and a has an s-**neighbour** b in $\mathcal{A}$ that is not blocked with $b\colon C \notin \mathcal{A}$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{b\colon C\}$

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$, a is not blocked, and
if C is a concept name, $a\colon C \in \mathcal{A}$ but $a\colon D \notin \mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon D\}$
else if $a\colon (\dot{\neg} C \sqcup D) \notin \mathcal{A}$ for a in $\mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a\colon (\dot{\neg} C \sqcup D)\}$

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

Example: Is A satisfiable w.r.t. $\{A \sqsubseteq \exists R^{-}.A \sqcap (\forall R.(\neg A \sqcup \exists S.B))\}$?
Is B satisfiable w.r.t. $\{B \sqsubseteq \exists R.B \sqcap \forall R^{-}.\forall R^{-}.(A \sqcap \neg A)\}$?

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

Example: Is A satisfiable w.r.t. $\{A \sqsubseteq \exists R^{-}.A \sqcap (\forall R.(\neg A \sqcup \exists S.B))\}$?
Is B satisfiable w.r.t. $\{B \sqsubseteq \exists R.B \sqcap \forall R^{-}.\forall R^{-}.(A \sqcap \neg A)\}$?

The algorithm is no longer sound!

"subset-blocking" $(\{C \mid a\colon C \in \mathcal{A}\} \subseteq \{C \mid b\colon C \in \mathcal{A}\})$ **no longer suffices:**

In case we have

- a freshly introduced individual (i.e., not present in input ontology) a,
- an individual b with
  - $\mathfrak{L}(a) := \{C \mid a\colon C \in \mathcal{A}\} = \{C \mid b\colon C \in \mathcal{A}\} =: \mathfrak{L}(b)$,
  - b is older than a (i.e., b was introduced earlier than a)

we say b **blocks** a and we say a is **blocked**.

Lemma 4: Let $\mathcal{O}$ be an $\mathcal{ALCI}$ ontology in NNF. Then

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

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

Proof: **1.** (Termination): identical to the $\mathcal{ALC}$ case.

2. (Soundness): again, construct a finite (non-tree) model from
a complete, clash-free ABox $\mathcal{A}_f$ for $\mathcal{O}$

$$
\begin{aligned}
\Delta^{\mathcal{I}} &:= \ldots \\
A^{\mathcal{I}} &:= \ldots \\
r^{\mathcal{I}} &:= \{\langle x, y\rangle \in \Delta^{\mathcal{I}^2} \mid y \text{ is or blocks an } r\text{-neighbour of } x \text{ or } \}
\end{aligned}
$$

Again, prove that, for all $x \in \Delta^{\mathcal{I}}$:

(C1) $x\colon D \in \mathcal{B}_f$ implies $x \in D^{\mathcal{I}}$

(C2) $C \sqsubseteq D \in \mathcal{O}$ implies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

➜ $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{B}_f)$ ($\mathcal{I}$ defines all role assertions by definition)
➜ $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{A})$ because $\mathcal{A} \subseteq \mathcal{B}_f$
➜ $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is consistent

3. Completeness: again, use model $\mathcal{I}$ of $\mathcal{O}$ and a mapping π to find a complete & clash-free ABox.

Corollary:
- Consistency of $\mathcal{ALCI}$ ontologies is decidable
- $\mathcal{ALCI}$ has the finite model property

It can be shown that

- pure $\mathcal{ALCI}$-concept satisfiability (without TBoxes) is **PSpace-complete**, just like $\mathcal{ALC}$
- these algorithms can be extended to ABoxes and thus ontology consistency; rather straighforward

Even more expressive DLs

Most reasoners support more expressive DLs, in particular with **number restrictions** (aka cardinality restrictions or counting quantifiers).

They generalize

- existential restrictions $\exists r.C$
  "*there is at least one r-successor that is an instance of C*"

  to **at-least restrictions** $(\geq n\ r.C)$
  "*there are $\geq n$ r-successors that are instances of C*", for a non-neg. integer n,

  e.g., $\mathtt{Bike} \sqsubseteq (\geq 2\mathtt{hasPart.Wheel})$
- universal restrictions $\forall r.C$
  "*there are zero r-successor that are instances of $\neg C$*"

  to **at-most restrictions** $(\leq n\ r.D)$
  "*there are at most n r-successors that are instances of D*" for a non-neg. integer n,

  e.g., $\mathtt{Bike} \sqsubseteq (\leq 2\mathtt{hasPart.Wheel})$

The DL $\mathcal{ALCQI}$

$\mathcal{ALCQI}$ is the extension of $\mathcal{ALCI}$ with cardinality restrictions, i.e., concepts are built like $\mathcal{ALCI}$ concepts, plus $(\geq n\ r.C)$ and $(\geq n\ r.C)$,
where C is an $\mathcal{ALCQI}$ concept.

An **interpretation** $\mathcal{I}$ has to satisfy, in addition:

$$
\begin{aligned}
(\geq n\ r.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid |\{y \mid (x,y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}| \geq n\} \\
(\leq n\ r.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid |\{y \mid (x,y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}| \leq n\}
\end{aligned}
$$

TBoxes, ABoxes, and Ontologies are defined analogously.

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

Example: for $\mathcal{T} = \{A \sqsubseteq \exists r.A \sqcap (\leq 1\ r^{-}.\top)\}$, the concept $(\neg A \sqcap \exists r.A)$ is satisfiable w.r.t. $\mathcal{T}$, but only in infinite models.

Question: Is $\mathcal{ALCQI}$ still decidable?

A Tableau algorithm for $\mathcal{ALCQI}$

$\mathcal{ALCQI}$ is decidable (in ExpTime), but tableau algorithm goes beyond scope of this course.

Main changes to $\mathcal{ALCI}$ tableau required for handling cardinality restrictions:

- blocking:
  - $\mathcal{ALC}$: subset blocking
  - $\mathcal{ALCI}$: equality blocking
  - $\mathcal{ALCQI}$: double equality blocking (between 2 pairs of individuals)
- new rules:
  - (obvious) $\geq$-rule that generates n r-neighbours in C for $(\geq n\ r.C)$
  - (obvious) $\leq$-rule that merges r-neighbours in C for $(\leq n\ r.C)$ in case there are more than n
  - ?-rule to determine/guess, for $x : (\leq n\ r.C)$, which of x's r-successors are Cs (and which are $\neg C$s)

A Tableau algorithm for $\mathcal{ALCQI}$

$\mathcal{ALCQI}$ is decidable (in ExpTime), but tableau algorithm goes beyond scope of this course.

Main changes to $\mathcal{ALCI}$ tableau required for handling cardinality restrictions:

- tableau algorithm is no longer **monotonic** (because $\leq$-rule merges individuals)
  $\Rightarrow$ *yo-yo* effect might lead to non-termination
  $\Rightarrow$ use explicit inequality relation on individuals, to avoid *yo-yo-ing*, e.g., when
  - $x: (\geq 3\ r.\top)$ leads to generation of r-successors of x via $\geq$-rule in case there are less than 3 of them in r
  - $x: (\leq 2\ r.\top)$ leads to merging of r-successors of x via $\leq$-rule if there are more than 2 of them

Thank you for your attention!