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

Day 2: Tableau Algorithms

Uli Sattler
Which of the following subsumptions hold?

- \( r \) some (A and B) is subsumed by \( r \) some A
  \[ \exists r. (A \cap B) \subseteq \exists r. A \]

- (r some A) and (r only B) is subsumed by \( r \) some B
  \[ \exists r. A \cap \forall r. B \subseteq \exists r. B \]

- r only (A and not A) is subsumed by r only B
  \[ \forall r. (A \cap \neg A) \subseteq \forall r. B \]

- \( r \) some (r only A) is subsumed by \( r \) some (r some (A or not A))
  \[ \exists r. (\forall r. A) \subseteq \exists r. (\exists r. (A \cup \neg A)) \]

- r only (A and B) is subsumed by (r only A) and (r only B)
  \[ \forall r. (A \cap B) \subseteq \forall r. A \cap \forall r. B \]

- \( r \) some B is subsumed by r only B
  \[ \exists r. B \subseteq \forall r. A \]
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? \( \mathcal{O} \models \top \subseteq \bot \)?
- is $\mathcal{O}$ coherent? is there concept name $A$ with $\mathcal{O} \models A \subseteq \bot$?
- compute class hierarchy! for all concept names $A, B$: $\mathcal{O} \models A \subseteq B$?
- classify individuals! for all concept names $A$, individual names $b$: $\mathcal{O} \models b : B$?

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 $\mathcal{O} \cup \{a : C\}$ is consistent
2. $\mathcal{O}$ is coherent iff, for each concept name $A$,
   $\mathcal{O} \cup \{a : A\}$ is consistent
3. $\mathcal{O} \models A \subseteq B$ iff $\mathcal{O} \cup \{a : (A \cap \neg B)\}$ is not consistent
4. $\mathcal{O} \models b : B$ iff $\mathcal{O} \cup \{b : \neg B\}$ is not consistent

\( \Rightarrow \) a decision procedure to solve consistency decides all standard DL reasoning problems
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:

- \( \mathcal{ALC} \): \( \land, \lor, \neg, \exists r.C, \forall r.C \)
- 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:

- initialised with a singleton set \( S = \{ \mathcal{A} \} \) when started with \( \mathcal{O} = (T, \mathcal{A}) \)
- 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 \( S \), \( (T, \mathcal{A}') \) is consistent
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 $\text{NNF}(C)$ by pushing negation inwards, using

\[
\neg(C \cap D) \equiv \neg C \cup \neg D \quad \neg(C \cup D) \equiv \neg C \cap \neg D \\
\neg(\exists R.C) \equiv (\forall R.\neg C) \\
\neg(\forall R.C) \equiv (\exists R.\neg C)
\]

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

From now on, all concepts in GCIs and concept assertions are assumed to be in NNF, and we use $\neg C$ to denote the $\text{NNF}(\neg C)$. 
A tableau algorithm for \textit{ALC} ontologies

The algorithm

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

Following Theorem 2, we can use the algorithm to test

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

...and interpreting the results according to Theorem 2.
Preliminary Tableau Expansion Rules for $\mathcal{ALC}$

$\Box$-rule: if $a : C_1 \cap C_2 \in A$ and $\{a : C_1, a : C_2\} \not\subseteq A$
then replace $A$ with $A \cup \{a : C_1, a : C_2\}$

$\square$-rule: if $a : C_1 \sqcup C_2 \in A$ and $\{a : C_1, a : C_2\} \cap A = \emptyset$
then replace $A$ with $A \cup \{a : C_1\}$ and $A \cup \{a : C_2\}$

$\exists$-rule: if $a : \exists s.C \in A$ and there is no $b$ with $\{(a, b) : s, b : C\} \subseteq A$
then create a new individual name $c$ and
replace $A$ with $A \cup \{(a, c) : s, c : C\}$

$\forall$-rule: if $\{a : \forall s.C, (a, b) : s\} \subseteq A$ and $b : C \not\in A$
then replace $A$ with $A \cup \{b : C\}$

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$ and $a : (\neg C \sqcup D) \not\in A$ for $a$ in $A$,
then replace $A$ with $A \cup \{a : (\neg C \sqcup D)\}$
Tableau Algorithm for $\mathcal{ALC}$: Observations

- We only apply rules if their application does "something new"
- The $\sqcap$-rule is the only one to replace an ABox with more than one other
- To understand the GCI-rule, convince yourself that
  \begin{align*}
  \mathcal{I} \text{ satisfies a GCI } C \sqsubseteq D \text{ iff, for each } e \in \Delta^\mathcal{I}, \text{ we have }& e \not\in C^\mathcal{I} \text{ or } e \in D^\mathcal{I} \\
  \text{and } e \not\in C^\mathcal{I} \text{ is the case iff }& e \in (\neg C)^\mathcal{I}
  \end{align*}
- 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, $a : F \not\in A$ for $a : B \in A$ and $B \sqsubseteq F \in \mathcal{T}$, then replace $A$ with $A \cup \{a : F\}$

- If $A$ is replaced with $A'$, then $A \subseteq A'$
Example: apply the tableau algorithm to $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ with

$$\mathcal{T} = \{ A \sqsubseteq B \cap \exists r.G \cap \forall r.C, \quad \mathcal{A} = \{ a: A, \quad b: E, \quad c: G \} \}
\quad E \sqsubseteq A \cap H \cap \forall r.F, \quad (a, c): r, \quad (b, c): r,\quad
g \sqsubseteq E \sqcap P, \quad c: G\}
\quad H \sqsubseteq E \cup \forall r.\neg C\}$$
Termination of our Tableau Algorithm for \( \text{ALC} \)

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 \( \mathcal{A} = \{ a : A \} \).

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

\[
\{ C \mid a : C \in \mathcal{A} \} \subseteq \{ C \mid b : C \in \mathcal{A} \}.
\]

In case we have

- a freshly introduced individual (i.e., not present in input ontology) \( a \),
- an individual \( b \) with
  - \( \{ C \mid a : C \in \mathcal{A} \} \subseteq \{ C \mid b : C \in \mathcal{A} \} \),
  - \( b \) 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 $a : C_1 \sqcap C_2 \in \mathcal{A}$, $a$ is not blocked, and $\{a : C_1, a : C_2\} \not\subseteq \mathcal{A}$
then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a : C_1, a : C_2\}$

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

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

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

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

...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 $\text{P}Space$.
3. Consistency of $\mathcal{ALC}$ ontologies is decidable in $\text{ExpSpace}$.
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 \( \text{sub}(\mathcal{O}) \) be the set of all subconcepts of concepts occurring in \( \mathcal{A} \) together with all subconcepts of \( \neg C \sqcup D \) for each \( C \sqsubseteq D \in \mathcal{T} \).

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 \( \text{sub}(\mathcal{O}) \) and
   \[
   \# \text{sub}(\mathcal{O}) \leq \Sigma_{C \sqsubseteq D \in \mathcal{O}} (2 + |C| + |D|) + \Sigma_{a: C \in \mathcal{O}} |C|
   \]
   because, at each position in a concept, at most one sub-concept starts
4. due to blocking, there can be at most \( 2\# \text{sub}(\mathcal{O}) \) individuals in each ABox: if \( \{C \mid a: C \in \mathcal{A}\} \subseteq \{C \mid b: C \in \mathcal{A}\} \), \( a \) 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 \( b : \exists R.C \in A \) 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.
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 $2^{\# \text{sub}(\mathcal{O})}$

This is not optimal: we will see tomorrow that consistency of $\mathcal{ALC}$ ontologies is decidable in exponential time, in fact ExpTime-complete.
(2) Let $A_f$ be a complete & clash-free ABox generated for $O = (T, A)$, and let $B_f$ be $A_f$ without assertions involving blocked individuals.

Define an interpretation $I$ as follows:

- $\Delta^I := \{x \mid x \text{ is an individual in } B_f\}$
- $A^I := \{x \in \Delta^I \mid x : A \in B_f\}$ for concept names $A$
- $r^I := \{(x, y) \in \Delta^I \times \Delta^I \mid (x, y) : r \in B_f$ or $(x, y') : r \in A_f$ and $y$ blocks $y'$ in $A_f\}$

and show, by induction on structure of concepts:

(C1) $x : D \in B_f$ implies $x \in D^I$

(C2) $C \subseteq D \in T$ implies $C^I \subseteq D^I$

$I$ is a model of $(T, B_f)$ ($I$ satisfies all role assertions by definition)

$I$ is a model of $(T, A)$ because $A \subseteq B_f$

$O = (T, A)$ is consistent
Proof of Lemma 3.2: Soundness II

\[\Delta^I := \{x \mid x \text{ is an individual in } \mathcal{B}_f\}\]
\[A^I := \{x \in \Delta^I \mid x: A \in \mathcal{B}_f\}\] for concept names \(A\)
\[r^I := \{(x, y) \in \Delta^I \times \Delta^I \mid (x, y): r \in \mathcal{B}_f \text{ or} (x, y'): r \in \mathcal{B}_f \text{ and } y \text{ blocks } y'\}\]

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

- for concept names \(D\): by definition of \(I\)
- 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

\[ \Delta^I := \{ x \mid x \text{ is an individual in } B_f \} \]

\[ A^I := \{ x \in \Delta^I \mid x : A \in B_f \} \quad \text{for concept names } A \]

\[ r^I := \{ (x, y) \in \Delta^I \times \Delta^I \mid (x, y) : r \in B_f \text{ or } (x, y') : r \in B_f \text{ and } y \text{ blocks } y' \} \]

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

This is an immediate consequence of

- \( \Delta^I \) 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 \( x \in C^I \), then \( x : C \in B_f \), and thus \( x : D \in B_f \)
  - else, \( x : (\dashv C \sqcup D) \in B_f \)
- (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 $\pi$:

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

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

\[
(\star) \text{ if } x : C \in \mathcal{A}, \text{ then } \pi(x) \in C^\mathcal{I}
\]

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

• easy for $\cap$-, $\forall$-, and the GCI-rule,

• for $\exists$-rule, we need to extend $\pi$ to the newly created $r$-successor

• for $\cup$-rule, if $C_1 \sqcup C_2 : x \in \mathcal{A}$, $(\star)$ implies that $\pi(x) \in (C_1 \sqcup C_2)^\mathcal{I}$

  $\implies$ we can choose $\mathcal{A}_i = \mathcal{A} \cup \{x : C_i\}$ with $\pi(x) \in C_i^\mathcal{I}$ and thus preserve $(\star)$$

$\implies$ easy to see: $(\star)$ 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.
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...
Naive implementation of 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

$\Rightarrow$ requires double exponential space or

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

$\Rightarrow$ requires exponential space
Implementing the \textit{ALC} Tableau Algorithm

Optimisations are crucial

- concern every aspect of the algorithm
- help in “many” cases (which?)
- are implemented in various \textit{DL} reasoners
  - e.g., FaCT++, Pellet, RacerPro

In the following: a selection of some vital optimisations
Reasoners provides service “classify all concept names $T$”, i.e., for all concept names $C, D$ in $T$, reasoner decides does $T \models C \subseteq D$?

$\leadsto$ test consistency of $T \cup \{a : (C \cap \neg D)\}$

$\leadsto n^2$ consistency tests!

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

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

$C \subseteq D_i$ w.r.t. $T$? $\quad \circ D_1 \quad \circ D_2$
Reasoners provides service "classify all concept names $\mathcal{T}$", i.e.,
for all concept names $C, D$ in $\mathcal{T}$, reasoner decides does $\mathcal{T} \models C \subseteq D$?

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

$\leadsto n^2$ consistency tests!

**Goal:** reduce number of consistency tests when classifying TBox

**Idea 1:** "trickle" new concept $C$ into hierarchy computed so far
Reasoners provides service “classify all concept names $T$”, i.e., for all concept names $C, D$ in $T$, reasoner decides does $T \models C \sqsubseteq D$?

\[
\implies \text{test consistency of } T \cup \{ a : (C \cap \neg D) \} \\
\implies n^2 \text{ consistency tests!}
\]

**Goal:** reduce number of consistency tests when classifying $T$Box

**Idea 1:** “trickle” new concept $C$ into hierarchy computed so far
Reasoners provides service “classify all concept names $\mathcal{T}$”, i.e., for all concept names $C, D$ in $\mathcal{T}$, reasoner decides does $\mathcal{T} \models C \sqsubseteq D$?

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

$\leadsto n^2$ 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
Remember: for \( T = \{ C_i \sqsubseteq D_i \mid 1 \leq i \leq n \} \), where no \( C_i \) is a concept name, each individual \( x \) will have \( n \) disjunctions \( x: (\neg C_i \sqcup D_i) \) due to

GCI-rule: if \( C \sqsubseteq D \in T \), \( a \) is not blocked, and

if \( C \) is a concept name, \( a: C \in A \) but \( a: D \notin A \),
then replace \( A \) with \( A \cup \{ a: D \} \)
else if \( a: (\neg C \sqcup D) \notin A \) for \( a \) in \( A \),
then replace \( A \) with \( A \cup \{ a: (\neg C \sqcup D) \} \)

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., Human \( \sqcap \ldots \sqsubseteq C \) or 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., $\text{Human} \sqcap \exists \text{owns Pet} \sqsubseteq C$ becomes $\text{Human} \sqsubseteq \neg \exists \text{owns Pet} \sqcup C$

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

**GCI-rule:** if $C \sqsubseteq D \in T$, $a$ is not blocked, and

if $C$ is a concept name, $a : C \in A$ but $a : D \notin A$,
then replace $A$ with $A \cup \{a : D\}$

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

**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!
Remember: If a clash is encountered, non-deterministic algorithm backtracks.
i.e., returns to last non-deterministic choice and
tries other possibility

Example: \( x: \exists R. (A \sqcap B) \sqcap ((C_1 \cup D_1) \sqcap \ldots \sqcap (C_n \cup D_n)) \sqcap \forall R. \neg A \)
Remember If a clash is encountered, non-deterministic algorithm backtracks
i.e., returns to last non-deterministic choice and tries other possibility

Example \( \exists R. (A \cap B) \cap ((C_1 \cup D_1) \cap \ldots \cap (C_n \cup D_n)) \cap \forall R. \neg A \)
Remember If a clash is encountered, non-deterministic algorithm backtracks

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

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

\begin{align*}
\mathcal{L}(x) &= \{C_1\} \\
\mathcal{L}(x) &= \{\neg C_1, D_1\} \\
\mathcal{L}(x) &= \{\neg C_2, D_2\} \\
\mathcal{L}(x) &= \{\neg C_n, D_n\} \\
\mathcal{L}(x) &= \{\neg C_n, D_n\} \\
\mathcal{L}(y) &= \{(A \land B), \neg A, A, B\} \\
\mathcal{L}(y) &= \{(A \land B), \neg A, A, B\} \\
\mathcal{L}(y) &= \{(A \land B), \neg A, A, B\} \\
\mathcal{L}(y) &= \{(A \land B), \neg A, A, B\} \\
\mathcal{L}(y) &= \{(A \land B), \neg A, A, B\}
\end{align*}

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

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

Example $x: \exists R. (A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)) \sqcap \forall R. \neg A$
Finally: $\mathcal{ALC}$ extends propositional logic

$\leadsto$ heuristics developed for SAT are relevant

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

can dramatically enhance performance

$\leadsto$ do they interact?

$\leadsto$ how?

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

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

Motivation: with inverse roles, one can use both

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

... and capture their interaction

$ALCI$ is the extension of $ALC$ with inverse roles $R^-$ in the place of role names:

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

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

Is $\exists r.\exists s.\neg A$ satisfiable w.r.t. $T = \{\top \sqsubseteq \forall s^-\forall r^-.\neg A\}$?
A tableau algorithm for \textit{ALCI} ontologies

**Modifications**

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

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

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

③ substitute “\( (x, y) : r \in A \)” in the $\forall$- and $\exists$-rule with “has an \( r \)-neighbour \( y \)”...
Tableau Expansion Rules for $\mathcal{ALCI}$

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

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

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

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

GCI-rule: if $C \sqsubseteq D \in \mathcal{T}$, $a$ is not blocked, and
if $C$ is a concept name, $a : C \in A$ but $a : D \not\in A$,
then replace $A$ with $A \cup \{a : D\}$
else if $a : (\neg C \sqcup D) \not\in A$ for $a$ in $A$,
then replace $A$ with $A \cup \{a : (\neg C \sqcup D)\}$
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. $\{A \sqsubseteq \exists R^-.A \cap (\forall R. (\neg A \sqcup \exists S.B))\}$?
Is $B$ satisfiable w.r.t. $\{B \sqsubseteq \exists R.B \cap \forall R^-.\forall R^-. (A \cap \neg A)\}$?

The algorithm is no longer sound!

“subset-blocking” $\left( \{C \mid a: C \in \mathcal{A}\} \subseteq \{C \mid b: C \in \mathcal{A}\}\right)$ no longer suffices:

In case we have

- a freshly introduced individual (i.e., not present in input ontology) $a$,
- an individual $b$ with
  - $\mathcal{L}(a) := \{C \mid a: C \in \mathcal{A}\} = \{C \mid b: C \in \mathcal{A}\} =: \mathcal{L}(b)$,
  - $b$ 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 \textit{ALCI} ontologies

Proof: 1. (Termination): identical to the \textit{ALC} case.

2. (Soundness): again, construct a finite (non-tree) model from a complete, clash-free ABox $A_f$ for $O$

\[
\Delta^I := \ldots \\
A^I := \ldots \\
r^I := \{ \langle x, y \rangle \in \Delta_{I}^2 \mid y \text{ is or blocks an } r\text{-neighbour of } x \text{ or } \}
\]

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

(C1) $x : D \in B_f$ implies $x \in D^I$

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

$\models \mathcal{I}$ is a model of $(\mathcal{T}, B_f)$ ($\mathcal{I}$ defines all role assertions by definition)

$\models \mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{A})$ because $\mathcal{A} \subseteq B_f$

$\models \mathcal{O} = (\mathcal{T}, \mathcal{A})$ is consistent
3. Completeness: again, use model $I$ of $O$ and a mapping $\pi$ to find a complete & clash-free ABox.

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

It can be shown that
- pure $\text{ALCI}$-concept satisfiability (without TBoxes) is $\text{PSpace-complete}$, just like $\text{ALC}$
- these algorithms can be extended to ABoxes and thus ontology consistency; rather straightforward
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., Bike $\sqsubseteq (\geq 2 \text{hasPart}.\text{Wheel})$

- **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., Bike $\sqsubseteq (\leq 2 \text{hasPart}.\text{Wheel})$
\textit{ALCQI} is the extension of \textit{ALCI} with cardinality restrictions, i.e., concepts are built like \textit{ALCI} concepts, plus $(\geq n\ r.C)$ and $(\geq n\ r.C)$, where $C$ is an \textit{ALCQI} concept.

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

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

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

TBoxes, ABoxes, and Ontologies are defined analogously.

**Observation:** \textit{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 \textit{ALCQI} still decidable?
A Tableau algorithm for $\text{ALCQI}$

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

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

- **blocking:**
  - $\text{ALC}$: subset blocking
  - $\text{ALCI}$: equality blocking
  - $\text{ALCQI}$: 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 $C$'s (and which are $\neg C$'s)
A Tableau algorithm for \texttt{ALCQI}

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

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

- tableau algorithm is no longer \textbf{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!