An Introduction to Description Logics: Techniques, Properties, and Applications

NASSLLI, Day 3, Part 1

Reasoning via Tableau Algorithms (ctd)

Uli Sattler

Today

- remember the tableau algorithm from yesterday
- prove its correctness
- with some model properties and some optimisations
- ...as a result, we should have
  - a deep understanding the semantics
  - a deep understanding of the reasoning problems
  - some understanding of the sources of complexity
- some selected computational complexity results

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

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

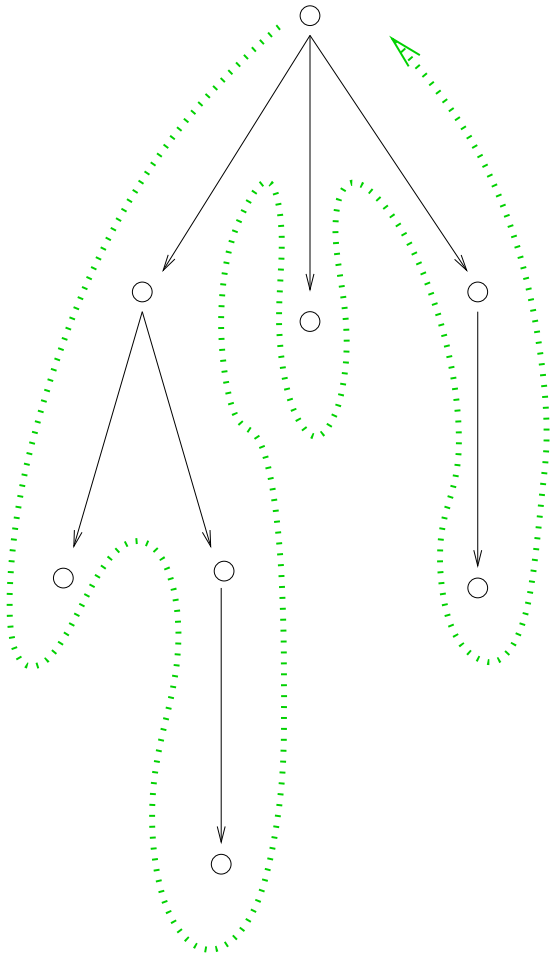
(1) **Termination** is a consequence of these observations:

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 $\mathbf{sub}(\mathcal{O})$ and
$$\#\,\mathbf{sub}(\mathcal{O}) \leq \Sigma_{C \sqsubseteq D \in O}(2 + |C| + |D|) + \Sigma_{a\colon C \in O}|C|$$
because, at each position in a concept, at most one sub-concept starts
4. due to blocking, there can be at most $2^{\#\,\mathbf{sub}(\mathcal{O})}$ 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.

Regarding Corollary 1.2

If we start the algorithm with $\{a : C\}$
to test satisfiability of C (w.r.t. empty TBox!), 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 $2^{\#\,\mathbf{sub}(\mathcal{O})}$

This is **not** optimal: we will see later 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:

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

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

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

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

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

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

This is an immediate consequence of

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

Proof of Lemma 3.3: Completeness

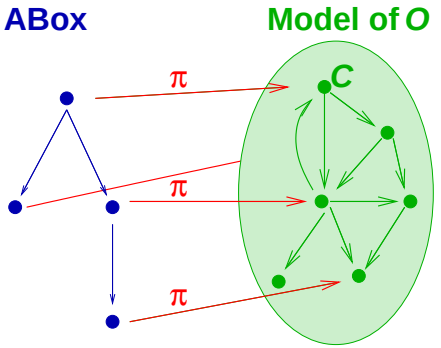
(3) Let $\mathcal{O}$ be consistent, and let $\mathcal{I}$ be a model of $\mathcal{O}$.

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



$(*)$ if $x\colon C \in \mathcal{A}$, then $\pi(x) \in C^{\mathcal{I}}$
if $(x, y)\colon r \in \mathcal{A}$, 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 $C_1 \sqcup C_2\colon x \in \mathcal{A}$, $(*)$ implies that $\pi(x) \in (C_1 \sqcup C_2)^{\mathcal{I}}$
  $\rightsquigarrow$ we can choose $\mathcal{A}_i = \mathcal{A} \cup \{x\colon C_i\}$ 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 $\top \sqsubseteq (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)$ 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, Konclude,...

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.,

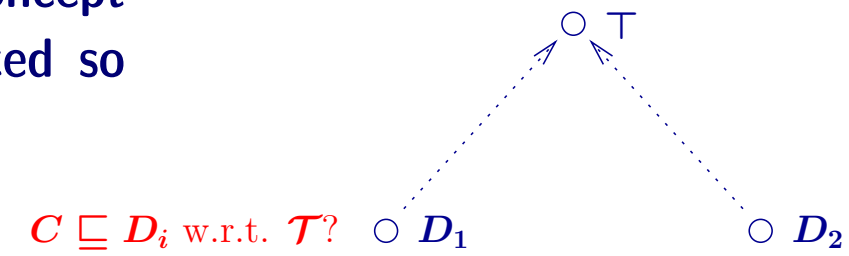
for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

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

$\rightsquigarrow$ 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.,

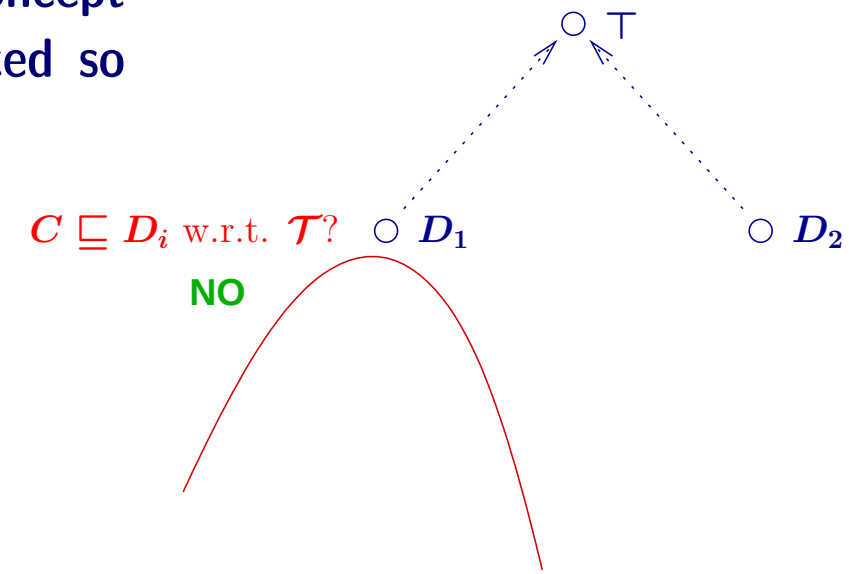
for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

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

$\rightsquigarrow$ 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.,

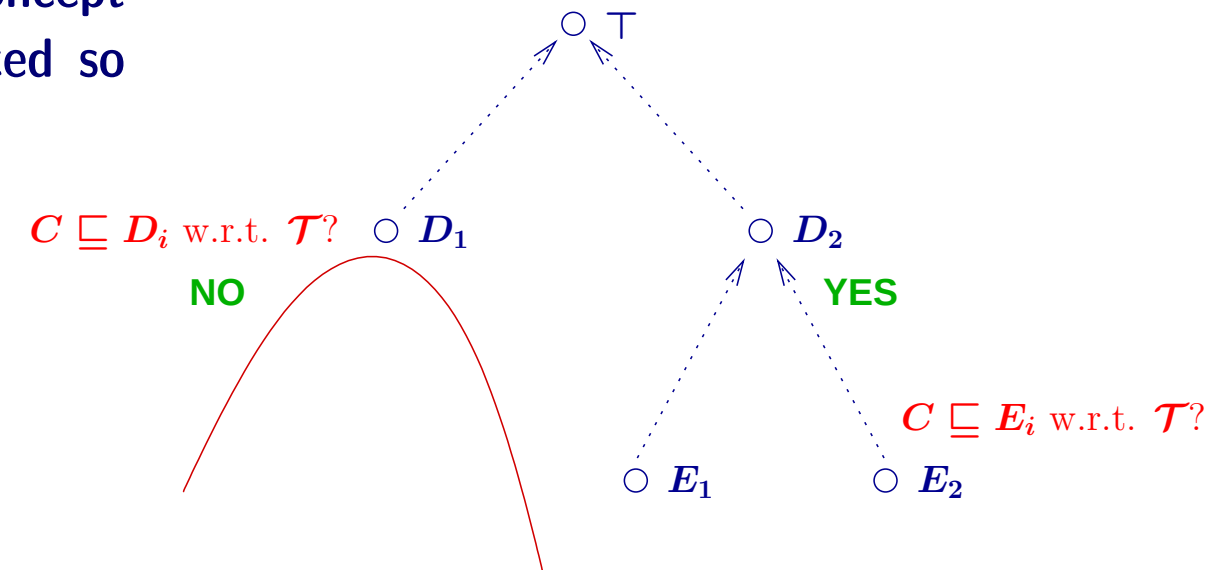
for all concept names C, D in $\mathcal{T}$, reasoner decides **does** $\mathcal{T} \models C \sqsubseteq D$?

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

$\rightsquigarrow$ 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., $\texttt{Human} \sqcap \ldots \sqsubseteq C$ or $\texttt{Device} \sqcap \ldots \sqsubseteq C$

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., $\texttt{Human} \sqcap \exists \texttt{owns.Pet} \sqsubseteq C$ becomes $\texttt{Human} \sqsubseteq \neg\exists \texttt{owns.Pet} \sqcup C$

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

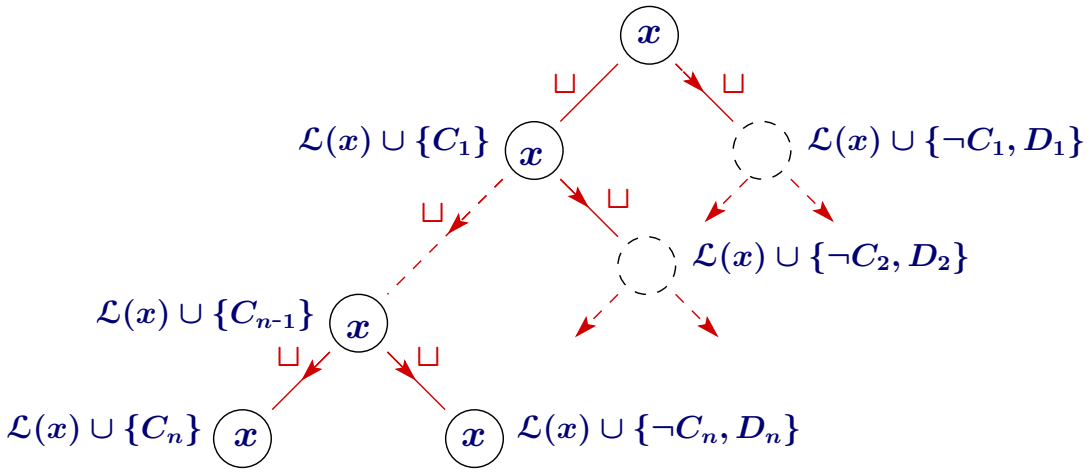
Observations: If no GCI is absorbable, nothing changes
Each absorption saves 1 disjunction **per individual outside** A_i,
in the best case, this avoids **almost all** disjunctions from TBox axioms!

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

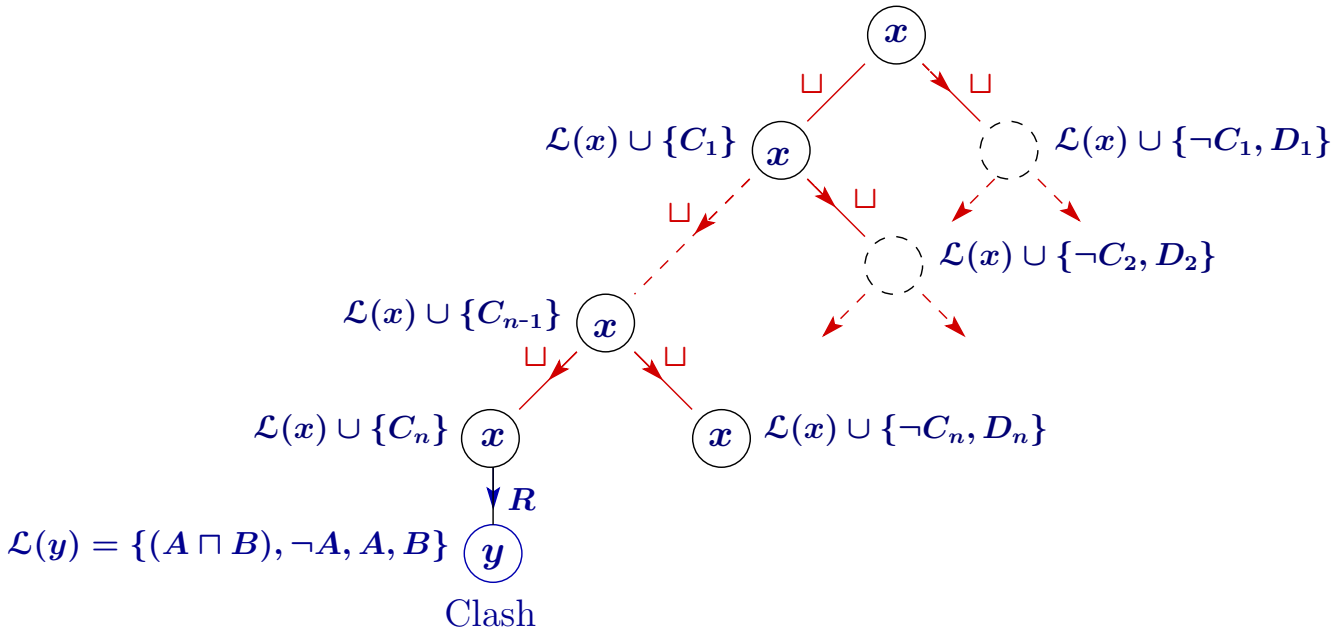


Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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

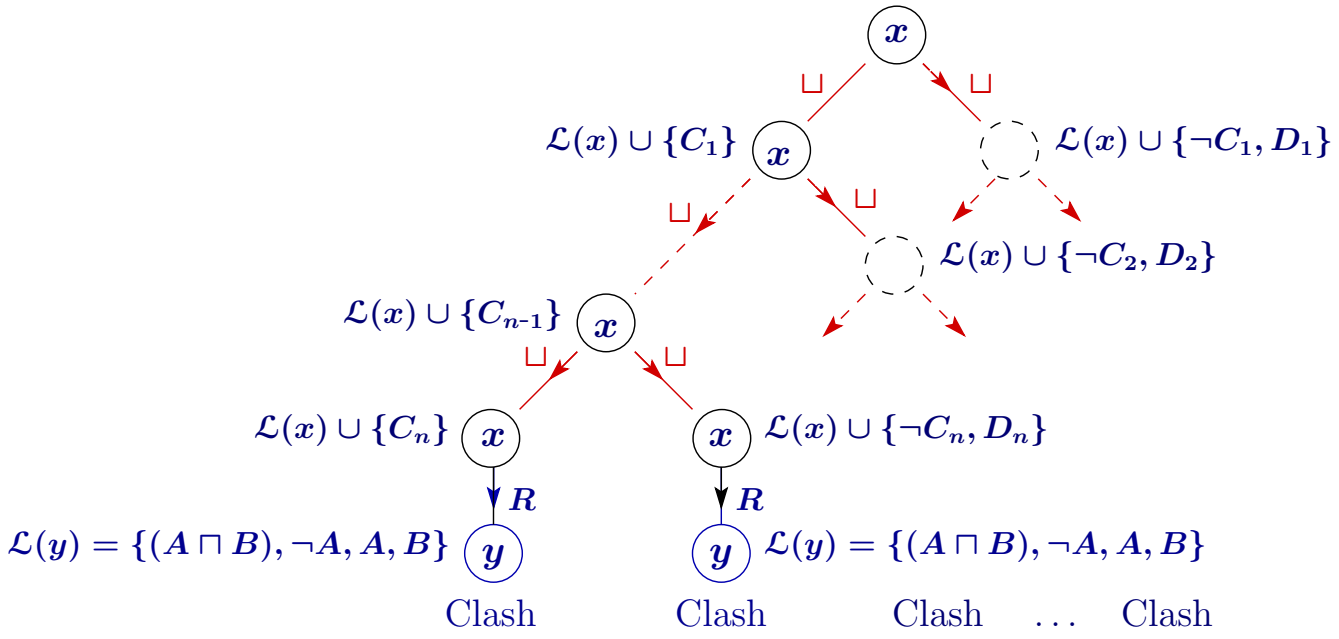


Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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

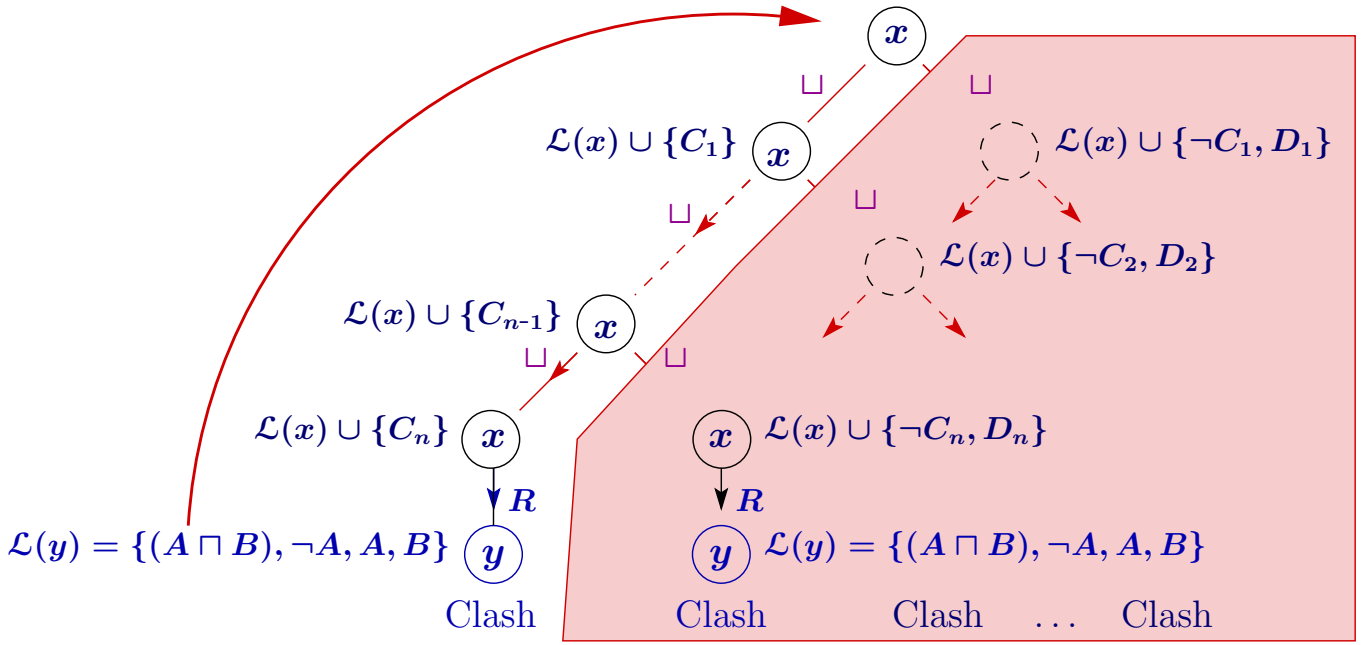


Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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



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

Let's look at complexity!