

**Description Logics:  
a Nice Family of Logics**

**Day 3: More on Tableau Algorithms**

**ESSLLI 2016**

**Uli Sattler**

## Today

- brief revisit of a tableau algorithm to decide consistency of *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!**

Can we say

**All Sneetches like all Icecream**

in a Description Logic? In *ALC*?

Can we say

**All Sneetches like all Icecream**

in a Description Logic? In *ALC*?

No:  $S \sqsubseteq \forall l.C$

means **All Sneetches like only icecream** (and nothing else)

Yes::  $S \sqsubseteq \forall \neg l. \neg C$

works, but requires negation of roles,

not available in *ALC*

but in *ALC* with Boolean roles, see *ALBO* R. Schmidt

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

## Properties of our tableau algorithm

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

- ✓ 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

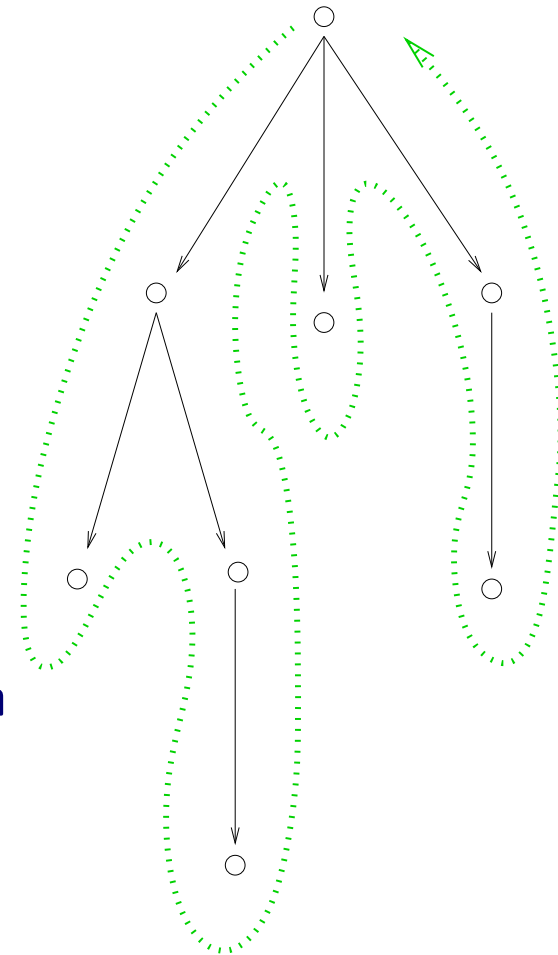
- Corollary 1:**
1. Our tableau algorithm **decides consistency** of  $\mathcal{ALC}$  ontologies.
  2. (with Theorem 2) all other  $\mathcal{ALC}$  reasoning problems are decidable.
  3. Satisfiability (and subsumption) of  $\mathcal{ALC}$  concepts is decidable in **PSpace**.
  4. Consistency of  $\mathcal{ALC}$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

## Regarding Corollary 1.2

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  $2^{\#\text{sub}(\mathcal{O})}$

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

$$\Delta^{\mathcal{I}} := \{x \mid x \text{ is an individual in } \mathcal{B}_f\}$$

$$A^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid x : A \in \mathcal{B}_f\} \quad \text{for concept names } A$$

$$r^{\mathcal{I}} := \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x, y) : r \in \mathcal{B}_f \text{ or} \\ (x, y') : r \in \mathcal{A}_f \text{ and } y \text{ blocks } y' \text{ in } \mathcal{A}_f\}$$

and show, by induction on structure of concepts:

$$(C0) \ (x, y) : r \in \mathcal{B}_f \text{ implies } (x, y) \in r^{\mathcal{I}}$$

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

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

→  $\mathcal{I}$  is a model of  $(\mathcal{T}, \mathcal{B}_f)$

→  $\mathcal{I}$  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

$$\Delta^{\mathcal{I}} := \{x \mid x \text{ is an individual in } \mathcal{B}_f\}$$

$$A^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid x : A \in \mathcal{B}_f\} \quad \text{for concept names } A$$

$$r^{\mathcal{I}} := \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x, y) : r \in \mathcal{B}_f \text{ or} \\ (x, y') : r \in \mathcal{B}_f \text{ and } y \text{ blocks } y'\}$$

(C0)  $(x, y) : r \in \mathcal{B}_f$  implies  $(x, y) \in r^{\mathcal{I}}$  holds by definition of  $\mathcal{I}$ .

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

- 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

$$\Delta^{\mathcal{I}} := \{x \mid x \text{ is an individual in } \mathcal{B}_f\}$$

$$A^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid x : A \in \mathcal{B}_f\} \quad \text{for concept names } A$$

$$r^{\mathcal{I}} := \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x, y) : r \in \mathcal{B}_f \text{ or} \\ (x, y') : r \in \mathcal{B}_f \text{ and } y \text{ blocks } y'\}$$

(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  $x : C \in \mathcal{B}_f$ , and thus  $x : D \in \mathcal{B}_f$
  - else,  $x : (\neg C \sqcup D) \in \mathcal{B}_f$
- (C1)

## Proof of Lemma 3.3: Completeness

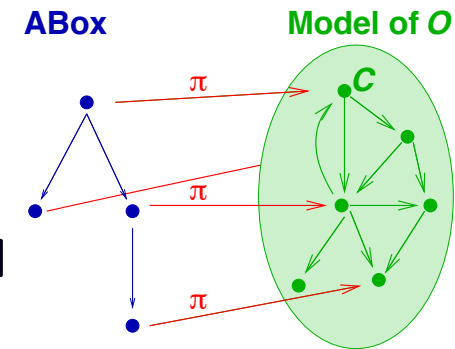
(3) Let  $\mathcal{O}$  be consistent, and let  $\mathcal{I}$  be a model of  $\mathcal{O}$ .

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

Inductively define a total mapping  $\pi$  :

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

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



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

- easy for  $\Box$ -,  $\forall$ -, and the GCI-rule,
- for  $\exists$ -rule, we need to extend  $\pi$  to the newly created  $r$ -successor
- for  $\sqcup$ -rule, if  $x : C_1 \sqcup C_2 \in \mathcal{A}$ , (\*) 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. *ALC* ontologies have the **finite model property**  
i.e., every consistent ontology has a **finite model**.
  5. *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  $\top \sqsubseteq (C_1 \sqcup D_1) \sqcap \dots \sqcap (C_n \sqcap D_n)$  and exponentially many individuals, algorithm might generate double exponentially many ABoxes
- ↪ requires double exponential space or
- use non-deterministic variant and backtracking to consider one ABox at a time
- ↪ 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.,

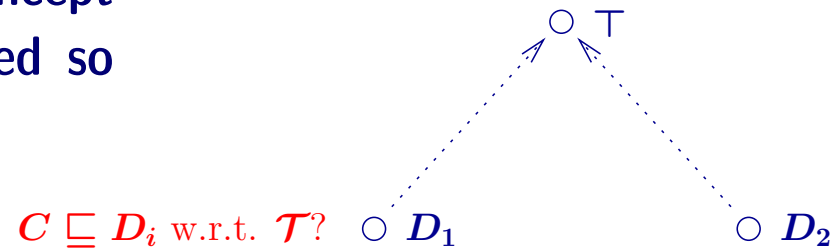
for all concept names  $C, D$  in  $\mathcal{T}$ , reasoner decides does  $\mathcal{T} \models C \sqsubseteq D$ ?

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

$\rightsquigarrow n^2$  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.,

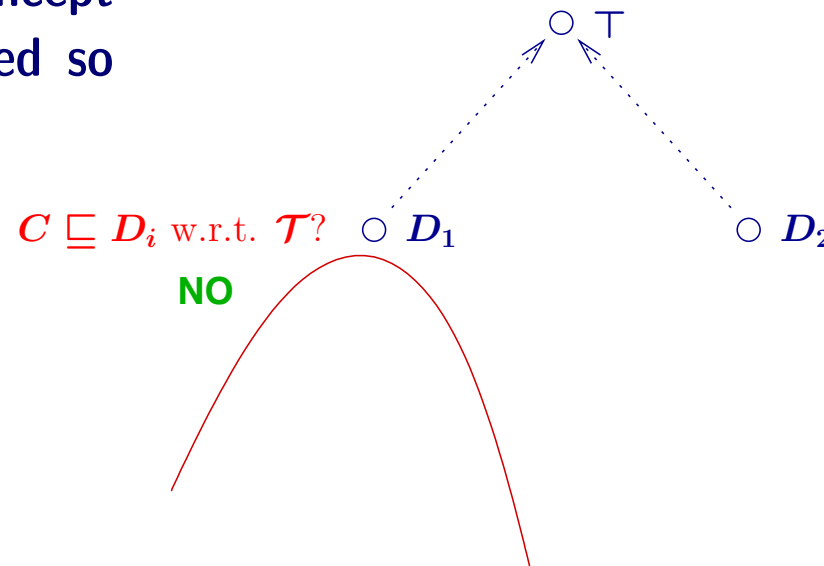
for all concept names  $C, D$  in  $\mathcal{T}$ , reasoner decides does  $\mathcal{T} \models C \sqsubseteq D$ ?

$\rightsquigarrow$  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.,

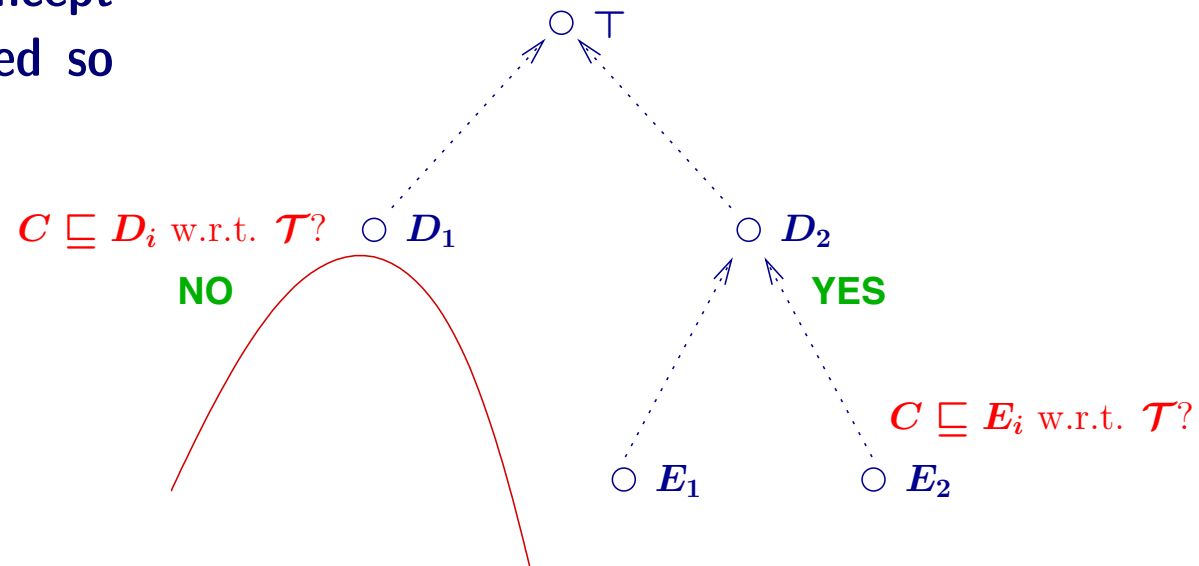
for all concept names  $C, D$  in  $\mathcal{T}$ , reasoner decides does  $\mathcal{T} \models C \sqsubseteq D$ ?

$\rightsquigarrow$  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  $x: (\neg C_i \sqcup D_i)$  due to

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

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

**Observation:** many GCIs are of the form  $A \sqcap \dots \sqsubseteq C$  for concept name  $A$   
e.g., Human  $\sqcap \dots \sqsubseteq C$  or Device  $\sqcap \dots \sqsubseteq C$

## 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”  $\mathcal{T} = \{A_i \sqsubseteq D_i \mid 1 \leq i \leq n_1\} \cup \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n_2\}$   
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,  $a : C \in \mathcal{A}$  but  $a : D \notin \mathcal{A}$ ,  
then replace  $\mathcal{A}$  with  $\mathcal{A} \cup \{a : D\}$   
else if  $a : (\neg C \sqcup D) \notin \mathcal{A}$  for  $a$  in  $\mathcal{A}$ ,  
then replace  $\mathcal{A}$  with  $\mathcal{A} \cup \{a : (\neg C \sqcup D)\}$

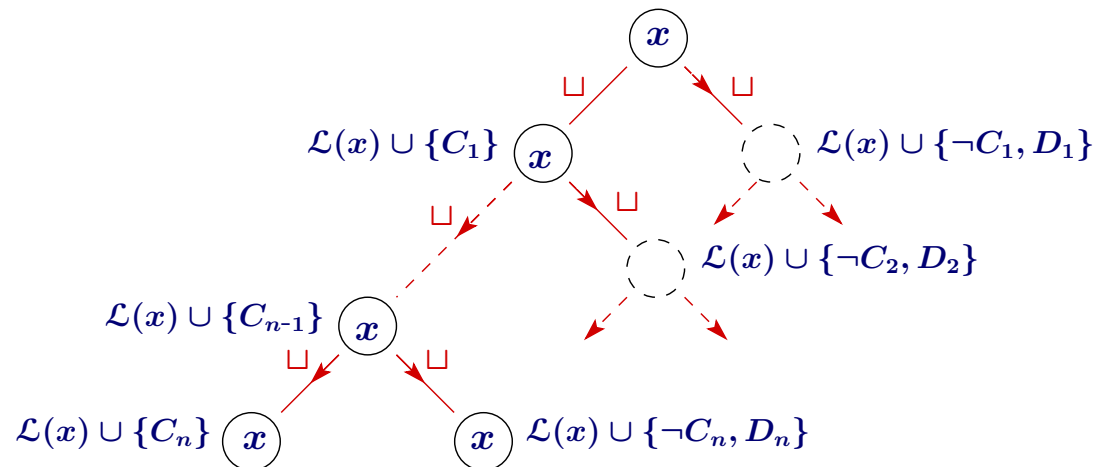
**Observations:** If no GCI is absorbable, nothing changes  
Each absorption saves 1 disjunction per individual outside  $A_i$ ,  
in the best case, this avoids almost all disjunctions from TBox axioms!

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

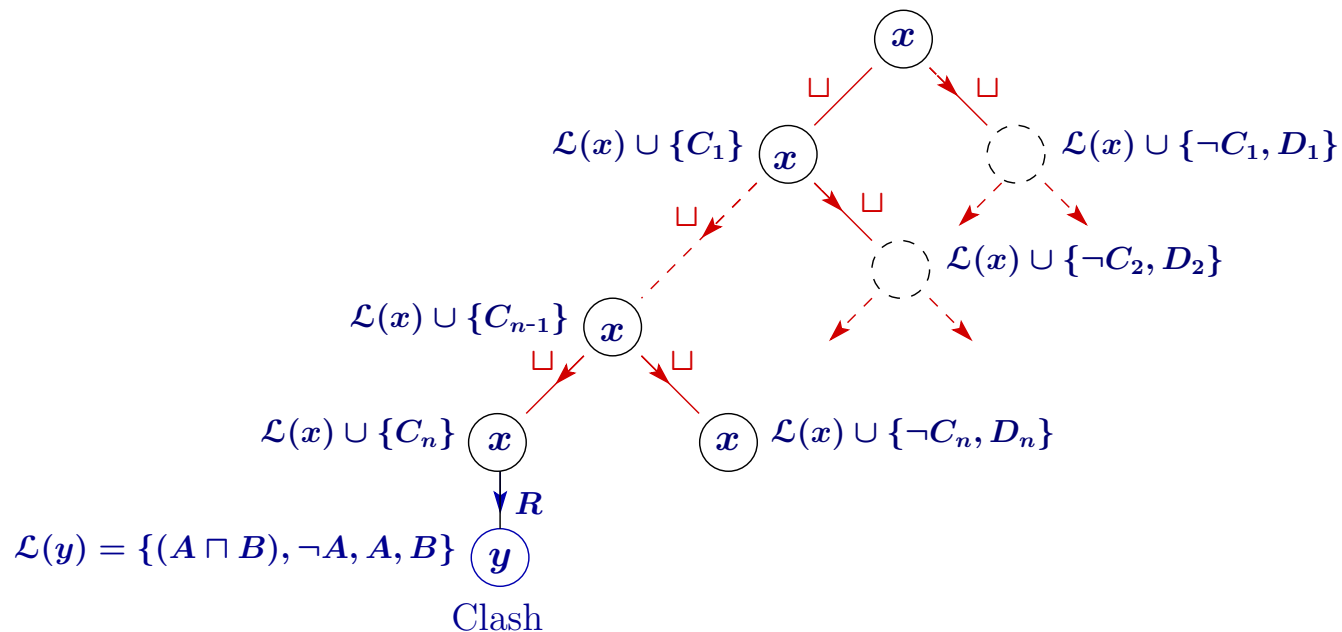


# Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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



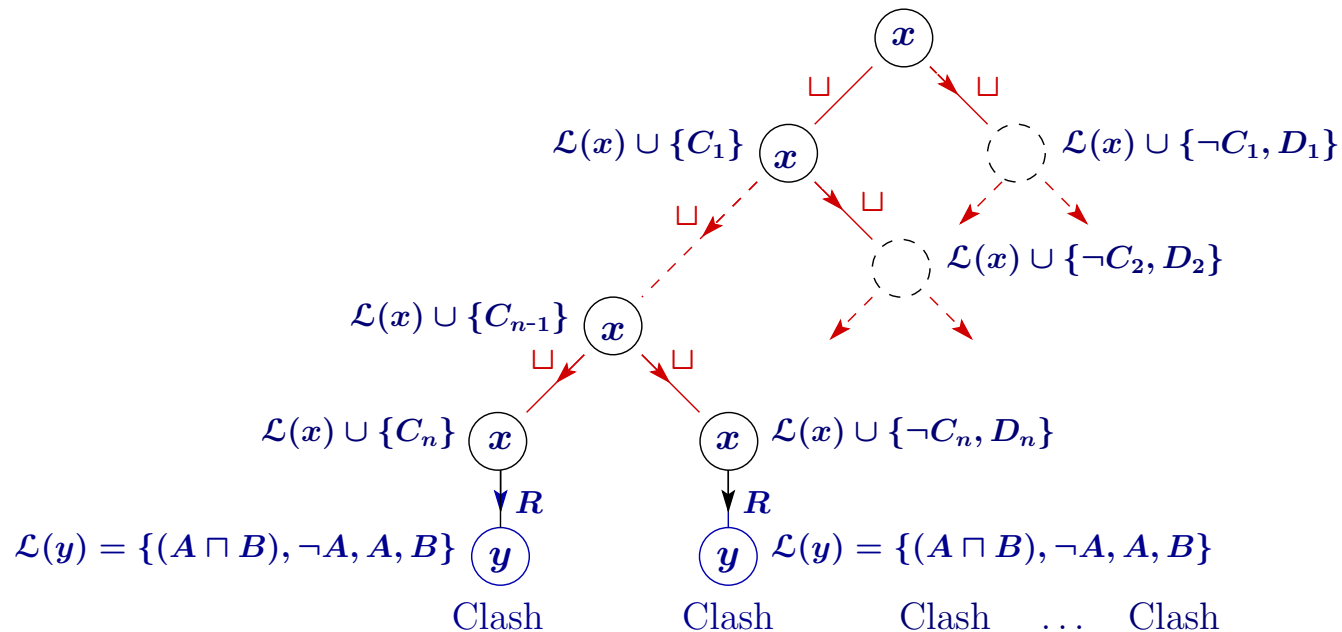


# Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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

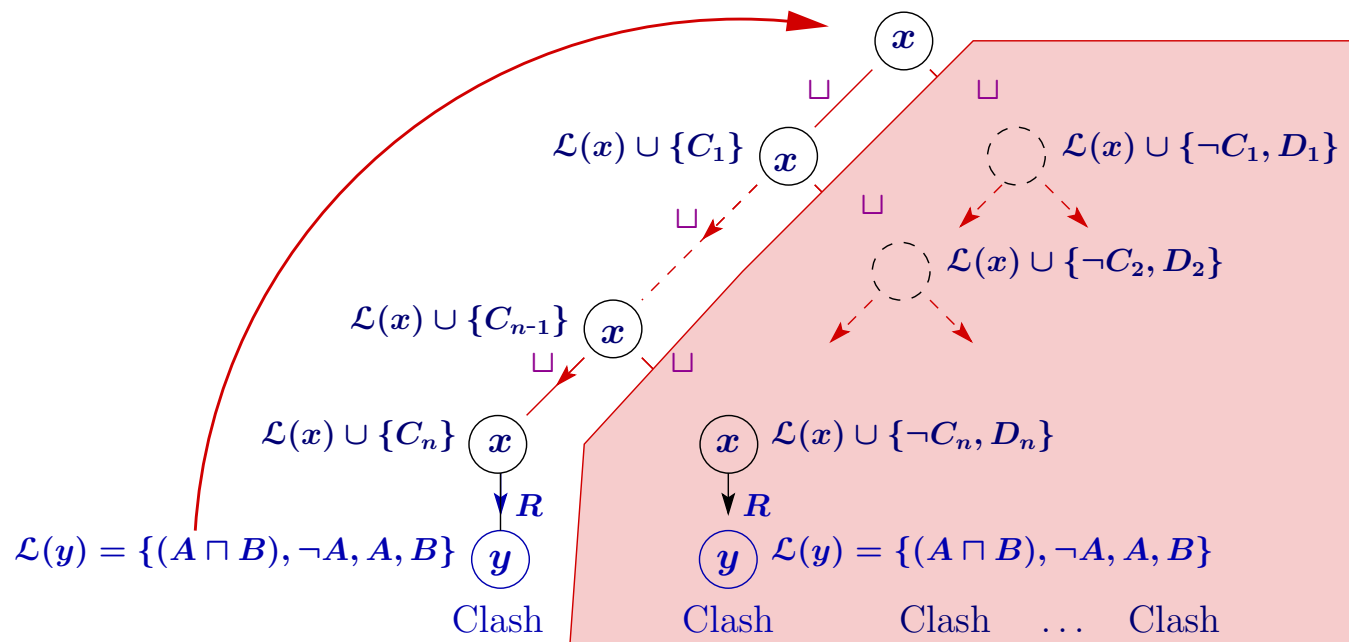


# Optimising the $\mathcal{ALC}$ Tableau Algorithm: Backjumping

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

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

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



## Optimising the *ALC* Tableau Algorithm: SAT Optimisations

Finally: *ALC* extends propositional logic

⇒ heuristics developed for **SAT** are relevant

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

can dramatically enhance performance

⇒ do they interact?

⇒ how?

⇒ which combination works best for which “cases”?

⇒ is the optimised algorithm still correct?

## Stuff seen today so far

- standard reasoning problems for *ALC* ontologies
- and their relationship & reducibility
- tableau algorithm for *ALC* ontologies that
  - requires blocking for termination
  - is a decision procedure for all standard *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  $\forall\text{parent}.\forall\text{child}.\text{Blond} \sqsubseteq \text{Blond}$  w.r.t.  $\mathcal{T} = \{\top \sqsubseteq \exists\text{parent}.\top\}$ ?

**Motivation:** with inverse roles, one can use both  
has-child and is-child-of  
has-part and is-part-of  
... ..  
and capture their interaction

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

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

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

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  $a: C_1 \sqcap C_2 \in \mathcal{A}$ ,  $a$  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  
 **$s$ -neighbour**  $b$  of  $a$  with  $b: C \in \mathcal{A}$   
then create a new individual  $c$  and replace  $\mathcal{A}$  with  $\mathcal{A} \cup \{(a, c): s, c: C\}$
- $\forall$ -rule: if  $a: \forall s.C \in \mathcal{A}$ , and  $a$  has an  
 **$s$ -neighbour**  $b$  in  $\mathcal{A}$  that is not blocked with  $b: C \notin \mathcal{A}$   
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  $a: D \notin \mathcal{A}$ ,  
then replace  $\mathcal{A}$  with  $\mathcal{A} \cup \{a: D\}$   
else if  $a: (\neg C \sqcup D) \notin \mathcal{A}$  for  $a$  in  $\mathcal{A}$ ,  
then replace  $\mathcal{A}$  with  $\mathcal{A} \cup \{a: (\neg C \sqcup D)\}$

## 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” ( $\{C \mid a: C \in \mathcal{A}\} \subseteq \{C \mid b: C \in \mathcal{A}\}$ ) no longer suffices:

In case we have

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

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

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

**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  $\mathcal{O}$

$$\Delta^{\mathcal{I}} := \dots$$

$$A^{\mathcal{I}} := \dots$$

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

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

(C0)  $(x, y) : r \in \mathcal{B}_f$  implies  $(x, y) \in r^{\mathcal{I}}$

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

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

→  $\mathcal{I}$  is a model of  $(\mathcal{T}, \mathcal{B}_f)$

→  $\mathcal{I}$  is a model of  $(\mathcal{T}, \mathcal{A})$  because  $\mathcal{A} \subseteq \mathcal{B}_f$

→  $\mathcal{O} = (\mathcal{T}, \mathcal{A})$  is consistent

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

3. Completeness: again, use model  $\mathcal{I}$  of  $\mathcal{O}$  and a mapping  $\pi$  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 straightforward

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.,  $\text{Bike} \sqsubseteq (\geq 2 \text{hasPart.Wheel})$

- **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.,  $\text{Bike} \sqsubseteq (\leq 2 \text{hasPart.Wheel})$

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