# Knowledge Representation

## Lecture 6: The Tableau Method and More Expressive DLs

**Patrick Koopmann**

November 13, 2023

# The Story so Far

- Reasoning is what makes DLs “intelligent”
- Semantics defines what is entailed from an ontology
- To compute entailments in practice, we need a reasoning algorithm
- $\mathcal{EL}$ is a DL that allows for very efficient reasoning
  - completion method
  - special model that captures all entailments (*canonical model*)
- What about $\mathcal{ALC}$ and more expressive DLs?

# Flashback: What is Knowledge Representation?

- KR as surrogate
- KR as expression of ontological commitment
- KR as theory of intelligent reasoning
- **KR as medium for efficient computation**
  - automated deduction is useless if it is not practical
  - trade-off between expressivity and reasoning performance
- KR as medium of human expression

# Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

Given an ontology $\mathcal{O}$ and a concept $C$, $C$ is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of $C$ and $\mathcal{O}$).

# Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

> Given an ontology $\mathcal{O}$ and a concept $C$, $C$ is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of $C$ and $\mathcal{O}$).

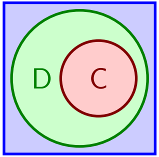
- For $\mathcal{EL}$, this wouldn't have made sense, since every concept is satisfiable

# Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

> Given an ontology $\mathcal{O}$ and a concept $C$, $C$ is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of $C$ and $\mathcal{O}$).

- For $\mathcal{EL}$, this wouldn't have made sense, since every concept is satisfiable
- In $\mathcal{ALC}$, we can reduce many problems to it:
  - To decide $\mathcal{O} \models C \sqsubseteq D$, we check whether $C \sqcap \neg D$ is unsatisfiable
  - To decide consistency of $\mathcal{O}$, we check whether $\top$ is satisfiable



# Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept

# Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept
- We then try, on different branches, to find a model for $\mathcal{O}$ and $C$

# Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept
- We then try, on different branches, to find a model for $\mathcal{O}$ and $C$
- Again rules to add concepts and individuals

# Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept
- We then try, on different branches, to find a model for $\mathcal{O}$ and $C$
- Again rules to add concepts and individuals
- A branch might get a clash in case it is inconsistent
  - We then go to the next branch

# Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept
- We then try, on different branches, to find a model for $\mathcal{O}$ and $C$
- Again rules to add concepts and individuals
- A branch might get a clash in case it is inconsistent
  - We then go to the next branch
- If we complete a branch without clash, the concept is satisfiable
  - The branch corresponds to a model

# Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept
- We then try, on different branches, to find a model for $\mathcal{O}$ and $C$
- Again rules to add concepts and individuals
- A branch might get a clash in case it is inconsistent
  - We then go to the next branch
- If we complete a branch without clash, the concept is satisfiable
  - The branch corresponds to a model
- If all branches clash, the concept is unsatisfiable

# Normalization

- A concept $C$ is in negation normal form (NNF) iff the negation symbol $\neg$ only occurs in front of concept names.
- A TBox is in NNF if every axiom is of the form $\top \sqsubseteq C$, where $C$ is in NNF
- An ABox is in NNF if every concept in it is in NNF
- An ontology is in NNF if TBox and ABox are in NNF

# The Tableaux procedure

- We assume everything is normalized.
- Branches are represented as ABoxes.
- We start with the assertion $a : C$.
- We then step-wise apply a set of $\mathcal{ALC}$ expansion rules to construct different ABoxes on different branches.

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

- $\sqcap$-rule: $a : \ C \sqcap D$

  $$\Longrightarrow \quad a : \ C, a : \ D$$

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

- $\sqcap$-rule: $a : \ C \sqcap D$
  $$\Longrightarrow \quad a : \ C, a : \ D$$
- $\sqcup$-rule: $a : \ C \sqcup D$ and we have neither $a : \ C$ nor $a : \ D$
  $$\Longrightarrow \quad a : \ C \text{ or } a : \ D$$

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

- $\sqcap$-rule: $a: \ C \sqcap D$
  $$\Longrightarrow \quad a: \ C, a: \ D$$
- $\sqcup$-rule: $a: \ C \sqcup D$ and we have neither $a: \ C$ nor $a: \ D$
  $$\Longrightarrow \quad a: \ C \text{ or } a: \ D$$

The $\sqcup$-rule introduces a new branch.

- One branch for every possibility
- We first continue on the branch for $C$. If it fails, we continue on the branch for $D$.

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

- $\sqcap$-rule: $a: \ C \sqcap D$
  $\Longrightarrow \quad a: \ C, a: \ D$
- $\sqcup$-rule: $a: \ C \sqcup D$ and we have neither $a: \ C$ nor $a: \ D$
  $\Longrightarrow \quad a: \ C$ or $a: \ D$
- $\exists$-rule: $a: \ \exists r.C$, no $\langle a, b \rangle: \ r$ s.t. $b: \ C$
  $\Longrightarrow \langle a, b \rangle: \ r, b: \ C$ for a new individual $b$

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

- $\sqcap$-rule: $a : \ C \sqcap D$
  $\Longrightarrow \quad a : \ C, a : \ D$
- $\sqcup$-rule: $a : \ C \sqcup D$ and we have neither $a : \ C$ nor $a : \ D$
  $\Longrightarrow \quad a : \ C$ or $a : \ D$
- $\exists$-rule: $a : \ \exists r.C$, no $\langle a, b \rangle : \ r$ s.t. $b : \ C$
  $\Longrightarrow \langle a, b \rangle : \ r, b : \ C$ for a new individual $b$
- $\forall$-rule: $a : \ \forall r.C, \langle a, b \rangle : \ r$
  $\Longrightarrow b : \ C$

# $\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If $X$ is on the current branch and not $Y$, add $Y$

- $\sqcap$-rule: $a : C \sqcap D$
  $\Longrightarrow \quad a : C, a : D$
- $\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$
  $\Longrightarrow \quad a : C$ or $a : D$
- $\exists$-rule: $a : \exists r.C$, no $\langle a, b \rangle : r$ s.t. $b : C$
  $\Longrightarrow \langle a, b \rangle : r, b : C$ for a new individual $b$
- $\forall$-rule: $a : \forall r.C, \langle a, b \rangle : r$
  $\Longrightarrow b : C$
- $\mathcal{T}$-rule: $\top \sqsubseteq C \in \mathcal{T}$
  $\Longrightarrow \quad a : C$ for any individual $a$ we already introduced

# Differences to the Precompletion Method for $\mathcal{EL}$

1. Rules only go in "one direction"
   - We only break concepts into simpler ones
   - We never introduce conjunction, disjunction or role restrictions

# Differences to the Precompletion Method for $\mathcal{EL}$

1. Rules only go in "one direction"
   - We only break concepts into simpler ones
   - We never introduce conjunction, disjunction or role restrictions
   - We want to decide satisfiability not subsumption

# Differences to the Precompletion Method for $\mathcal{EL}$

1. Rules only go in "one direction"
   - We only break concepts into simpler ones
   - We never introduce conjunction, disjunction or role restrictions
   - We want to decide satisfiability not subsumption
   - We don't have to deal with concepts on the left-hand side of an axiom (all axioms are of the form $\top \sqsubseteq C$)

# Differences to the Precompletion Method for $\mathcal{EL}$

1. Rules only go in "one direction"
   - We only break concepts into simpler ones
   - We never introduce conjunction, disjunction or role restrictions
   - We want to decide satisfiability not subsumption
   - We don't have to deal with concepts on the left-hand side of an axiom (all axioms are of the form $\top \sqsubseteq C$)
   - In the $\mathcal{EL}$ method, every concept has to already occur in the input
   - For $\mathcal{ALC}$, this would be to restrictive
   - Since concepts only get smaller, we only introduce finitely many concepts

# Differences to the Precompletion Method for $\mathcal{EL}$

1. Rules only go in "one direction"
   - We only break concepts into simpler ones
   - We never introduce conjunction, disjunction or role restrictions
   - We want to decide satisfiability not subsumption
   - We don't have to deal with concepts on the left-hand side of an axiom (all axioms are of the form $\top \sqsubseteq C$)
   - In the $\mathcal{EL}$ method, every concept has to already occur in the input
   - For $\mathcal{ALC}$, this would be to restrictive
   - Since concepts only get smaller, we only introduce finitely many concepts
2. No special treatment of initial concepts
   - In the $\mathcal{EL}$ method, we would reuse individuals based on their initial concepts
   - We already saw that this does not work together with the $\forall$-rule
   - We will need another mechanism to deal with cyclic TBoxes such as $\{A \sqsubseteq \exists r.A\}$

# Termination Criterion

A branch has a clash if for some individual $a$, we either have $a : \bot$ or for some concept name $A$, we have both $a : A$ and $a : \neg A$.

# Termination Criterion

A branch has a clash if for some individual $a$, we either have $a : \bot$ or for some concept name $A$, we have both $a : A$ and $a : \neg A$.

- Once we found a clash, we know that we cannot find a model anymore on the current branch.
- We then close the branch, and continue on the next.
- If all branches are closed with a clash, we know that the concept is unsatisfiable.

# Termination Criterion

A branch has a clash if for some individual $a$, we either have $a : \bot$ or for some concept name $A$, we have both $a : A$ and $a : \neg A$.

- Once we found a clash, we know that we cannot find a model anymore on the current branch.
- We then close the branch, and continue on the next.
- If all branches are closed with a clash, we know that the concept is unsatisfiable.

A branch is complete if it has no clash and no more rule can be applied.

# Termination Criterion

A branch has a clash if for some individual $a$, we either have $a : \perp$ or for some concept name $A$, we have both $a : A$ and $a : \neg A$.

- Once we found a clash, we know that we cannot find a model anymore on the current branch.
- We then close the branch, and continue on the next.
- If all branches are closed with a clash, we know that the concept is unsatisfiable.

A branch is complete if it has no clash and no more rule can be applied.

- Once we have a complete branch, we know that our concept is satisfiable.

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\mathcal{T}_1 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \; \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \; \top \sqsubseteq \neg(A \sqcap \exists r.\top) \sqcup \neg D \}$$

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\mathcal{T}_1 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq \neg(A \sqcap \exists r.\top) \sqcup \neg D \ \}$$
$$\mathcal{T}_2 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \neg \exists r.\top) \sqcup \neg D \}$$

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\mathcal{T}_1 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq \neg(A \sqcap \exists r.\top) \sqcup \neg D \ \}$$
$$\mathcal{T}_2 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \neg \exists r.\top) \sqcup \neg D \}$$
$$\mathcal{T}_3 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\neg \top) \sqcup \neg D \}$$

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\mathcal{T}_1 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq \neg(A \sqcap \exists r.\top) \sqcup \neg D \}$$
$$\mathcal{T}_2 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \neg \exists r.\top) \sqcup \neg D \}$$
$$\mathcal{T}_3 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\neg \top) \sqcup \neg D \}$$
$$\mathcal{T}_4 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\begin{aligned}
\mathcal{T}_1 =& \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq \neg(A \sqcap \exists r.\top) \sqcup \neg D \ \} \\
\mathcal{T}_2 =& \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \neg \exists r.\top) \sqcup \neg D \} \\
\mathcal{T}_3 =& \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\neg \top) \sqcup \neg D \} \\
\mathcal{T}_4 =& \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \ \} \\
=& \ \mathcal{T}'
\end{aligned}$$

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\begin{aligned} \mathcal{T}_4 =& \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \} \\ =& \mathcal{T}' \end{aligned}$$

   - The concept $A \sqcap \neg B$ is already in NNF.

# Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\mathcal{T}_4 = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \} \\ = \mathcal{T}'$$

   - The concept $A \sqcap \neg B$ is already in NNF.
3. We are now ready to apply the Tableaux procedure.

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

$a : A \sqcap \neg B$

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

$a : A \sqcap \neg B$

$\sqcap$-rule: $a : \ C \sqcap D \quad \Longrightarrow \quad a : \ C, a : \ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$\sqcap$-rule: $a :\ C \sqcap D \quad \Longrightarrow \quad a :\ C, a :\ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$\mathcal{T}$-rule: $\top \sqsubseteq C \in \mathcal{T} \quad \Longrightarrow \quad a :\ C$ for any individual $a$ we already introduced

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a: \neg A$

$\sqcup$-rule: $a : \ C \sqcup D$ and we have neither $a : \ C$ nor $a : \ D \quad \Longrightarrow a : \ C$ or $a : \ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : \neg A$

CLASH!

Clash: For some $a$, $A$, either $a : \bot$ or both $a : A$ and $a : \neg A$.

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$\sqcup$-rule: $a :\ C \sqcup D$ and we have neither $a :\ C$ nor $a :\ D$ $\Longrightarrow a :\ C$ or $a :\ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $B$

$\sqcup$-rule: $a :\ C \sqcup D$ and we have neither $a :\ C$ nor $a :\ D$ $\implies a :\ C$ or $a :\ D$

# Example: Tableaux Method

$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C), \ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F), \ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $B$

CLASH!

Clash: For some $a$, $A$, either $a : \bot$ or both $a : A$ and $a : \neg A$.

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $C$

$\sqcup$-rule: $a :\ C \sqcup D$ and we have neither $a :\ C$ nor $a :\ D$ $\implies a :\ C$ or $a :\ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $C$

$a : \neg C \sqcup (D \sqcap \exists r.F)$

$\mathcal{T}$-rule: $\top \sqsubseteq C \in \mathcal{T} \quad \Longrightarrow \quad a :\ C$ for any individual $a$ we already introduced

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a : C$

$a : \neg C \sqcup (D \sqcap \exists r.F)$

$a : \neg C$

$\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$ $\Longrightarrow a : C$ or $a : D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $C$

$a : \neg C \sqcup (D \sqcap \exists r.F)$

$a$: $\neg C$

CLASH!

**Clash:** For some $a$, $A$, either $a : \bot$ or both $a : A$ and $a : \neg A$.

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $C$

$a : \neg C \sqcup (D \sqcap \exists r.F)$

$a : D \sqcap \exists r.F$

$\sqcup$-rule: $a :\ C \sqcup D$ and we have neither $a :\ C$ nor $a :\ D$ $\implies a :\ C$ or $a :\ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

$a : A \sqcap \neg B$

$a : A$

$a : \neg B$

$a : \neg A \sqcup (B \sqcup C)$

$a : B \sqcup C$

$a$: $C$

$a : \neg C \sqcup (D \sqcap \exists r.F)$

$a : D \sqcap \exists r.F$

$a$: $D$

$a : \exists r.F$

$\sqcap$-rule: $a : \ C \sqcap D \quad \Longrightarrow \quad a : \ C, a : \ D$

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

| | |
|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ |
| $a : A$ | $a : D \sqcap \exists r.F$ |
| $a : \neg B$ | $a$: $D$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ |
| $a : B \sqcup C$ | $\langle a, b\rangle : r$ |
| $a$: $C$ | $b : F$ |

$\exists$-rule: $a :\ \exists r.C$, no $\langle a, b\rangle :\ r$ s.t. $b :\ C \quad \Longrightarrow \langle a, b\rangle :\ r, b :\ C$ for new $b$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | |
| $a : \neg B$ | $a : D$ | |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | |
| $a : C$ | $b : F$ | |

$\mathcal{T}$-rule: $\top \sqsubseteq C \in \mathcal{T} \quad \Longrightarrow \quad a :\ C$ for any individual $a$ we already introduced

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $a : \neg A \sqcup \forall r.\bot$ |
| $a : \neg B$ | $a : D$ | |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | |
| $a : C$ | $b : F$ | |

$\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$ $\Longrightarrow$ $a : C$ or $a : D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $a : \neg A \sqcup \forall r.\bot$ |
| $a : \neg B$ | $a : D$ | $a : \neg A$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | |
| $a : C$ | $b : F$ | |
| | | CLASH! |

$\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$ $\Longrightarrow a : C$ or $a : D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $a : \neg A \sqcup \forall r.\bot$ |
| $a : \neg B$ | $a : D$ | $a : \forall r.\bot$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | |
| $a : C$ | $b : F$ | |

$\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$ $\Longrightarrow a : C$ or $a : D$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $a : \neg A \sqcup \forall r.\bot$ |
| $a : \neg B$ | $a : D$ | $a : \forall r.\bot$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | $b : \bot$ |
| $a : B \sqcup C$ | $\langle a, b\rangle : r$ | |
| $a : C$ | $b : F$ | |

$\forall$-rule: $a : \forall r.C,\ \langle a, b\rangle : r \implies b : C$

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $a: \neg A \sqcup \forall r.\bot$ |
| $a : \neg B$ | $a: D$ | $a: \forall r.\bot$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | $b: \bot$ |
| $a : B \sqcup C$ | $\langle a, b\rangle : r$ | |
| $a: C$ | $b : F$ | |
| | | CLASH! |

Clash: For some $a$, $A$, either $a : \bot$ or both $a : A$ and $a : \neg A$.

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $\cancel{a : \neg A \sqcup \forall r.\bot}$ |
| $a : \neg B$ | $a : D$ | $\cancel{a : \forall r.\bot}$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | $\cancel{b : \bot}$ |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | $a : \neg D$ |
| $a : C$ | $b : F$ | |

$\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$ $\Longrightarrow a : C$ or $a : D$

# Example: Tableaux Method

$$\mathcal{T}' = \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $\cancel{a : \neg A \sqcup \forall r.\bot}$ |
| $a : \neg B$ | $a : D$ | $\cancel{a : \forall r.\bot}$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | $\cancel{b : \bot}$ |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | $a : \neg D$ |
| $a : C$ | $b : F$ | |
| | | CLASH! |

Clash: For some $a$, $A$, either $a : \bot$ or both $a : A$ and $a : \neg A$.

# Example: Tableaux Method

$$\mathcal{T}' = \{\ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D\ \}$$

| | | |
|---|---|---|
| $a : A \sqcap \neg B$ | $a : \neg C \sqcup (D \sqcap \exists r.F)$ | $a : (\neg A \sqcup \forall r.\bot) \sqcup \neg D$ |
| $a : A$ | $a : D \sqcap \exists r.F$ | $\cancel{a: \neg A \sqcup \forall r.\bot}$ |
| $a : \neg B$ | $a: D$ | $\cancel{a: \forall r.\bot}$ |
| $a : \neg A \sqcup (B \sqcup C)$ | $a : \exists r.F$ | $\cancel{b: \bot}$ |
| $a : B \sqcup C$ | $\langle a, b \rangle : r$ | $a: \neg D$ |
| $a: C$ | $b : F$ | |
| | | CLASH! |

- All branches clashed $\Longrightarrow$ $A \sqcap \neg B$ is unsatisfiable!

  $\Longrightarrow$ $\mathcal{O} \models A \sqsubseteq B$

# Termination

$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$

$a : A$

# Termination

$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$

$$a : A$$
$$a : \exists r.B$$

# Termination

$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$

$$
\begin{aligned}
a &: A \\
a &: \exists r.B \\
\langle a, b\rangle &: r \\
b &: B
\end{aligned}
$$

# Termination

$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$

$$
\begin{aligned}
a &: A \\
a &: \exists r.B \\
\langle a, b \rangle &: r \\
b &: B \\
b &: \exists r.B
\end{aligned}
$$

# Termination

$$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$$

$$\begin{aligned}
a &: A \\
a &: \exists r.B \\
\langle a, b\rangle &: r \\
b &: B \\
b &: \exists r.B \\
\langle b, c\rangle &: r \\
c &: B
\end{aligned}$$

# Termination

$$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$$

$$\begin{aligned} a &: A \\ a &: \exists r.B \\ \langle a, b\rangle &: r \\ b &: B \\ b &: \exists r.B \\ \langle b, c\rangle &: r \\ c &: B \\ c &: \exists r.B \end{aligned}$$

# Termination

$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$

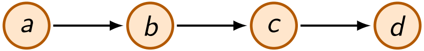
$$
\begin{aligned}
a &: A \\
a &: \exists r.B \\
\langle a, b \rangle &: r \\
b &: B \\
b &: \exists r.B \\
\langle b, c \rangle &: r \\
c &: B \\
c &: \exists r.B \\
\langle c, d \rangle &: r \\
d &: B
\end{aligned}
$$

# Termination

$\mathcal{T} = \{\top \sqsubseteq \exists r.B\}$

$$
\begin{aligned}
a &: A \\
a &: \exists r.B \\
\langle a, b \rangle &: r \\
b &: B \\
b &: \exists r.B \\
\langle b, c \rangle &: r \\
c &: B \\
c &: \exists r.B \\
\langle c, d \rangle &: r \\
d &: B \\
&\vdots
\end{aligned}
$$

# Ensuring Termination




- Every individual $d$ has a path from the individual $a$ from which we started
- The other individuals on this path (including the $a$) are called ancestors of $d$

# Ensuring Termination

$$a \longrightarrow b \longrightarrow c \longrightarrow d$$

- Every individual $d$ has a path from the individual $a$ from which we started
- The other individuals on this path (including the $a$) are called ancestors of $d$

> - An individual $a$ is blocked by an ancestor $b$ if for every $a : C$, we also have $b : C$.
> - An indivudal is blocked if it is blocked by some ancestor, or if some ancestor of it is blocked.

# Ensuring Termination

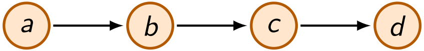
$$a \longrightarrow b \longrightarrow c \longrightarrow d$$

- Every individual $d$ has a path from the individual $a$ from which we started
- The other individuals on this path (including the $a$) are called ancestors of $d$

> - An individual $a$ is blocked by an ancestor $b$ if for every $a : C$, we also have $b : C$.
> - An indivudal is blocked if it is blocked by some ancestor, or if some ancestor of it is blocked.

- No rule can be applied on a blocked individual

# Ensuring Termination

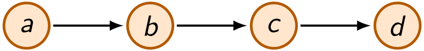



- Every individual $d$ has a path from the individual $a$ from which we started
- The other individuals on this path (including the $a$) are called ancestors of $d$

> - An individual $a$ is blocked by an ancestor $b$ if for every $a : C$, we also have $b : C$.
> - An indivudal is blocked if it is blocked by some ancestor, or if some ancestor of it is blocked.

- No rule can be applied on a blocked individual
- Idea: any expansion rule that applies to $a$ also applies to $b$, so there is no reason to apply it also on $a$

# Ensuring Termination




- Every individual $d$ has a path from the individual $a$ from which we started
- The other individuals on this path (including the $a$) are called ancestors of $d$

> - An individual $a$ is blocked by an ancestor $b$ if for every $a : C$, we also have $b : C$.
> - An indivudal is blocked if it is blocked by some ancestor, or if some ancestor of it is blocked.

- No rule can be applied on a blocked individual
- Idea: any expansion rule that applies to $a$ also applies to $b$, so there is no reason to apply it also on $a$
- Note: only ancestors can block

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$$a : A \sqcap \exists r.A$$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$a : A \sqcap \exists r.A$

$a : A$

$a : \exists r.A$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$a : A \sqcap \exists r.A$

$a : A$

$a : \exists r.A$

$a : \neg A \sqcup \forall r.B$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$a : A \sqcap \exists r.A$

$a : A$

$a : \exists r.A$

$a : \neg A \sqcup \forall r.B$

$a : \neg A$

CLASH!

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$a : A \sqcap \exists r.A$

$a : A$

$a : \exists r.A$

$a : \neg A \sqcup \forall r.B$

$a : \forall r.B$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$$\begin{aligned}
a &: A \sqcap \exists r.A \\
a &: A \\
a &: \exists r.A \\
a &: \neg A \sqcup \forall r.B \\
a &: \forall r.B \\
\langle a, b \rangle &: r \\
b &: A
\end{aligned}$$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$a : A \sqcap \exists r.A$ $\qquad$ $a : \neg B \sqcup \exists r.A$

$a : A$

$a : \exists r.A$

$a : \neg A \sqcup \forall r.B$

$a : \forall r.B$

$\langle a, b \rangle : r$

$b : A$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$$\begin{array}{rl} a : & A \sqcap \exists r.A \\ a : & A \\ a : & \exists r.A \\ a : & \neg A \sqcup \forall r.B \\ a : & \forall r.B \\ \langle a, b \rangle : & r \\ b : & A \end{array} \qquad \begin{array}{rl} a : & \neg B \sqcup \exists r.A \\ a : & \neg B \end{array}$$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$$\begin{array}{rlrl}
a : & A \sqcap \exists r.A & a : & \neg B \sqcup \exists r.A \\
a : & A & a : & \neg B \\
a : & \exists r.A & b : & B \\
a : & \neg A \sqcup \forall r.B & & \\
a : & \forall r.B & & \\
\langle a, b \rangle : & r & & \\
b : & A & &
\end{array}$$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | |
|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ |
| $a : A$ | $a : \neg B$ |
| $a : \exists r.A$ | $b : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ |
| $a : \forall r.B$ | |
| $\langle a, b \rangle : r$ | |
| $b : A$ | |

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | |
|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ |
| $a : A$ | $a : \neg B$ |
| $a : \exists r.A$ | $b : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ |
| $a : \forall r.B$ | $b : \neg A$ |
| $\langle a, b \rangle : r$ | |
| $b : A$ | |

CLASH!

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$$\begin{array}{rll}
a: & A \sqcap \exists r.A & \qquad a: \neg B \sqcup \exists r.A \\
a: & A & \qquad a: \neg B \\
a: & \exists r.A & \qquad b: B \\
a: & \neg A \sqcup \forall r.B & \qquad b: \neg A \sqcup \forall r.B \\
a: & \forall r.B & \qquad b: \forall r.B \\
\langle a, b \rangle : & r & \\
b: & A &
\end{array}$$

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | |
|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ |
| $a : A$ | $a : \neg B$ |
| $a : \exists r.A$ | $b : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ |
| $a : \forall r.B$ | $b : \forall r.B$ |
| $\langle a, b \rangle : r$ | $b : \neg B \sqcup \exists r.A$ |
| $b : A$ | |

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | |
|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ |
| $a : A$ | $a : \neg B$ |
| $a : \exists r.A$ | $b : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ |
| $a : \forall r.B$ | $b : \forall r.B$ |
| $\langle a, b \rangle : r$ | $b : \neg B \sqcup \exists r.A$ |
| $b : A$ | $b : \neg B$ |

CLASH!

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | |
|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ |
| $a : A$ | $a : \neg B$ |
| $a : \exists r.A$ | $b : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ |
| $a : \forall r.B$ | $b : \forall r.B$ |
| $\langle a, b \rangle : r$ | $b : \neg B \sqcup \exists r.A$ |
| $b : A$ | $b : \exists r.A$ |

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | | |
|---|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ | $\langle b, c \rangle : r$ |
| $a : A$ | $a : \neg B$ | $c : A$ |
| $a : \exists r.A$ | $b : B$ | |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ | |
| $a : \forall r.B$ | $b : \forall r.B$ | |
| $\langle a, b \rangle : r$ | $b : \neg B \sqcup \exists r.A$ | |
| $b : A$ | $b : \exists r.A$ | |

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | | |
|---|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ | $\langle b, c\rangle : r$ |
| $a : A$ | $a : \neg B$ | $c : A$ |
| $a : \exists r.A$ | $b : B$ | $c : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ | |
| $a : \forall r.B$ | $b : \forall r.B$ | |
| $\langle a, b\rangle : r$ | $b : \neg B \sqcup \exists r.A$ | |
| $b : A$ | $b : \exists r.A$ | |

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | | |
|---|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ | $\langle b, c \rangle : r$ |
| $a : A$ | $a : \neg B$ | $c : A$ |
| $a : \exists r.A$ | $b : B$ | $c : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ | |
| $a : \forall r.B$ | $b : \forall r.B$ | |
| $\langle a, b \rangle : r$ | $b : \neg B \sqcup \exists r.A$ | |
| $b : A$ | $b : \exists r.A$ | $c$ is blocked by $b$! |

# Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

| | | |
|---|---|---|
| $a : A \sqcap \exists r.A$ | $a : \neg B \sqcup \exists r.A$ | $\langle b, c \rangle : r$ |
| $a : A$ | $a : \neg B$ | $c : A$ |
| $a : \exists r.A$ | $b : B$ | $c : B$ |
| $a : \neg A \sqcup \forall r.B$ | $b : \neg A \sqcup \forall r.B$ | |
| $a : \forall r.B$ | $b : \forall r.B$ | |
| $\langle a, b \rangle : r$ | $b : \neg B \sqcup \exists r.A$ | |
| $b : A$ | $b : \exists r.A$ | $c$ is blocked by $b$! |

$\Rightarrow \quad A \sqcap \exists r.A$ is satisfiable!

# Decision Procedure

- One can show that, with the blocking condition, the tableaux method always terminates
  - every branch is closed or complete after finitely many steps
  - only finitely many branches are introduced

# Decision Procedure

- One can show that, with the blocking condition, the tableaux method always terminates
  - every branch is closed or complete after finitely many steps
  - only finitely many branches are introduced
- It is also easy to see that the method is sound
  - If it returns "satisfiable", then the concept is satisfiable.
  - Reason: we can easily transform the branch into a model
  - For the blocked nodes, we add a loop

# Decision Procedure

- One can show that, with the blocking condition, the tableaux method always terminates
  - every branch is closed or complete after finitely many steps
  - only finitely many branches are introduced
- It is also easy to see that the method is sound
  - If it returns "satisfiable", then the concept is satisfiable.
  - Reason: we can easily transform the branch into a model
  - For the blocked nodes, we add a loop
- In fact, the algorithm is also complete
  - For every satisfiable concept, it returns "satisfiable"
  - Idea: If $C$ is satisfiable, then there is a model $\mathcal{I}$ for it
  - We can use this model to "guide" the tableaux procedure $\rightarrow$ determine which rules to apply how

# Decision Procedure

**Theorem:** The tableaux algorithm is a decision procedure for $\mathcal{ALC}$ concept satisfiability.

- This means: it terminates, it is sound and it is complete

# Decision Procedure

**Theorem:** The tableaux algorithm is a decision procedure for $\mathcal{ALC}$ concept satisfiability.

- This means: it terminates, it is sound and it is complete
- However, the complexity is much worse as for the $\mathcal{EL}$ procedure
  - It needs much more inference steps in relation to the ontology size

# Decision Procedure

**Theorem:** The tableaux algorithm is a decision procedure for $\mathcal{ALC}$ concept satisfiability.

- This means: it terminates, it is sound and it is complete
- However, the complexity is much worse as for the $\mathcal{EL}$ procedure
  - It needs much more inference steps in relation to the ontology size
- In fact, reasoning with $\mathcal{ALC}$ is provably harder than for $\mathcal{EL}$
  - Exponential time instead of polynomial time complexity

# Decision Procedure

**Theorem:** The tableaux algorithm is a decision procedure for $\mathcal{ALC}$ concept satisfiability.

- This means: it terminates, it is sound and it is complete
- However, the complexity is much worse as for the $\mathcal{EL}$ procedure
  - It needs much more inference steps in relation to the ontology size
- In fact, reasoning with $\mathcal{ALC}$ is provably harder than for $\mathcal{EL}$
  - Exponential time instead of polynomial time complexity
- Modern DL reasoners try to exploit the "$\mathcal{EL}$-like" parts of the ontology as much as possible.

# More Expressive DLs

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:

  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.
  - hasIngredient is transitive

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.
  - hasIngredient is transitive
  - A Margherita has 1,120 kcal and costs 12.90€ .

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.
  - hasIngredient is transitive
  - A Margherita has 1,120 kcal and costs 12.90€ .
  - The enemies of my friends are also my enemies.

# Motivation

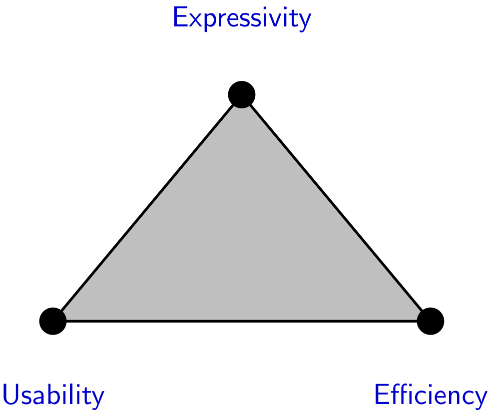
- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.
  - hasIngredient is transitive
  - A Margherita has 1,120 kcal and costs 12.90€ .
  - The enemies of my friends are also my enemies.

# Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.
  - hasIngredient is transitive
  - A Margherita has 1,120 kcal and costs 12.90€ .
  - The enemies of my friends are also my enemies.
- Modelling these with $\mathcal{ALC}$ would be awkward to impossible.

# The Limits of Expressivity

- A central feature of DLs is not only the syntax, but also *decidability*.
- A challenge is to stay *decidable*, while offering sufficient expressivity.



# Very Expressive Description Logics

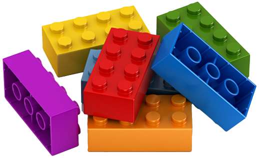
- $\mathcal{SROIQ}(D)$ is one of the most expressive logics of the description logic family that is still decidable.

# Very Expressive Description Logics

- $\mathcal{SROIQ}(D)$ is one of the most expressive logics of the description logic family that is still decidable. *The DL underlying OWL!*

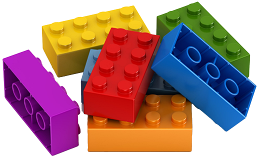
# Very Expressive Description Logics

- $\mathcal{SROIQ}(D)$ is one of the most expressive logics of the description logic family that is still decidable. *The DL underlying OWL!*
- The name describes its main additional features to $\mathcal{ALC}$:
  - transitive roles ($\mathcal{S}$),
  - complex **R**ole axioms ($\mathcal{R}$),
  - role **H**ierarchies ($\mathcal{H}$, contained in $\mathcal{R}$)
  - n**O**minals ($\mathcal{O}$),
  - **I**nverse roles ($\mathcal{I}$),
  - (**Q**ualified) number restrictions ($\mathcal{Q}$),
  - Concrete **D**omains ($\mathrm{D}$).



# Very Expressive Description Logics

- $\mathcal{SROIQ}(D)$ is one of the most expressive logics of the description logic family that is still decidable. *The DL underlying OWL!*
- The name describes its main additional features to $\mathcal{ALC}$:
  - transitive roles ($\mathcal{S}$),
  - complex **R**ole axioms ($\mathcal{R}$),
  - role **H**ierarchies ($\mathcal{H}$, contained in $\mathcal{R}$)
  - n**O**minals ($\mathcal{O}$),
  - **I**nverse roles ($\mathcal{I}$),
  - (**Q**ualified) number restrictions ($\mathcal{Q}$),
  - Concrete **D**omains ($\mathrm{D}$).
- DLs between $\mathcal{EL}$ and $\mathcal{SROIQ}(D)$ follow the same naming scheme: $\mathcal{ELHO}$, $\mathcal{ALCI}$, $\mathcal{SOQ}$, etc.



# Number Restrictions

(Qualified) number restrictions restrict the number of outgoing role connections.

For all role names $r$, concepts $C$, and $n \geq 0$, the following are concepts:

| | | |
|---|---|---|
| Name | at-least restriction | at-most restriction |
| Syntax | $\geq nr.C$ | $\leq nr.C$ |
| Semantics | $\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d, e) \in r^{\mathcal{I}}\} \geq n\}$ | $\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d, e) \in r^{\mathcal{I}}\} \leq n\}$ |

# Number Restrictions

(Qualified) number restrictions restrict the number of outgoing role connections.

For all role names $r$, concepts $C$, and $n \geq 0$, the following are concepts:

| | | |
|---|---|---|
| Name | at-least restriction | at-most restriction |
| Syntax | $\geq n r.C$ | $\leq n r.C$ |
| Semantics | $\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \geq n\}$ | $\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \leq n\}$ |

$Student \sqcap (\geq 2 attends.Lecture)$ $\quad$ $\leq 1 belongsTo.\top$

# Number Restrictions

(Qualified) number restrictions restrict the number of outgoing role connections.

For all role names $r$, concepts $C$, and $n \geq 0$, the following are concepts:

| Name | at-least restriction | at-most restriction |
|---|---|---|
| Syntax | $\geq nr.C$ | $\leq nr.C$ |
| Semantics | $\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \geq n\}$ | $\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \leq n\}$ |

$Student \sqcap (\geq 2\, attends.Lecture)$ $\quad\quad$ $\leq 1\, belongsTo.\top$

- We can also write $=nr.C := (\geq nr.C) \sqcap (\leq nr.C)$

$Cow \sqsubseteq =4\, hasBodyPart.Leg$

$Hand \sqsubseteq =5\, hasBodyPart.Finger$

# Choosing the Right Construct

$$
\mathcal{O} = \{ \quad
\begin{aligned}
\mathit{Margherita} &\sqsubseteq \mathit{Pizza} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{TomatoSauce} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{Mozarella} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{Basil} \\
\mathit{VegetarianPizza} &\equiv \mathit{Pizza} \sqcap \forall \mathit{hasTopping}.\mathit{Vegetarian} \\
\mathit{TomatoSauce} \sqcup \mathit{Mozarella} \sqcup \mathit{Basil} &\sqsubseteq \mathit{Vegetarian} \\
\mathit{TomatoSauce} &\sqsubseteq \neg \mathit{Mozarella} \sqcap \neg \mathit{Basil} \\
\mathit{Mozarella} &\sqsubseteq \neg \mathit{Basil} \quad \}
\end{aligned}
$$

# Choosing the Right Construct

$$
\mathcal{O} = \{ \quad \begin{aligned}
\mathit{Margherita} &\sqsubseteq \mathit{Pizza} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{TomatoSauce} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{Mozarella} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{Basil} \\
\mathit{VegetarianPizza} &\equiv \mathit{Pizza} \sqcap \forall \mathit{hasTopping}.\mathit{Vegetarian} \\
\mathit{TomatoSauce} \sqcup \mathit{Mozarella} \sqcup \mathit{Basil} &\sqsubseteq \mathit{Vegetarian} \\
\mathit{TomatoSauce} &\sqsubseteq \neg \mathit{Mozarella} \sqcap \neg \mathit{Basil} \\
\mathit{Mozarella} &\sqsubseteq \neg \mathit{Basil} \quad \}
\end{aligned}
$$

How to add closure:

1. $\mathit{Margherita} \sqsubseteq \forall \mathit{hasTopping}.(\mathit{TomatoSauce} \sqcup \mathit{Mozarella} \sqcup \mathit{Basil})$

# Choosing the Right Construct

$$
\begin{aligned}
\mathcal{O} = \{ \qquad \mathit{Margherita} &\sqsubseteq \mathit{Pizza} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{TomatoSauce} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{Mozarella} \\
&\quad \sqcap \exists \mathit{hasTopping}.\mathit{Basil} \\
\mathit{VegetarianPizza} &\equiv \mathit{Pizza} \sqcap \forall \mathit{hasTopping}.\mathit{Vegetarian} \\
\mathit{TomatoSauce} \sqcup \mathit{Mozarella} \sqcup \mathit{Basil} &\sqsubseteq \mathit{Vegetarian} \\
\mathit{TomatoSauce} &\sqsubseteq \neg \mathit{Mozarella} \sqcap \neg \mathit{Basil} \\
\mathit{Mozarella} &\sqsubseteq \neg \mathit{Basil} \qquad \}
\end{aligned}
$$

How to add closure:

1. $\mathit{Margherita} \sqsubseteq \forall \mathit{hasTopping}.(\mathit{TomatoSauce} \sqcup \mathit{Mozarella} \sqcup \mathit{Basil})$
2. or $\mathit{Margherita} \sqsubseteq \leq 3 \mathit{hasTopping}.\top$ ?

# Inverse Roles

For all role names $r$, the following is also a role, and can be used in all places where a role name can be used:

| | |
|---|---|
| Name: | inverse role |
| Syntax: | $r^-$ |
| Semantics: | $(r^-)^{\mathcal{I}} = \{(d, e) \mid (e, d) \in r^{\mathcal{I}}\}$ |

# Inverse Roles

For all role names $r$, the following is also a role, and can be used in all places where a role name can be used:

| | |
|---|---|
| Name: | inverse role |
| Syntax: | $r^-$ |
| Semantics: | $(r^-)^\mathcal{I} = \{(d, e) \mid (e, d) \in r^\mathcal{I}\}$ |

$belongsTo^-$ $\qquad \top \sqsubseteq \forall hasChild^-.Parent$

# Nominals and Self-Love

# Nominals and Self-Love

For all individual names $a$ and role names $r$, the following are concepts:

| | | |
|---|---|---|
| Name: | nominal | local reflexivity |
| Syntax: | $\{a\}$ | $\exists r.\mathsf{Self}$ |
| Semantics: | $\{a^{\mathcal{I}}\}$ | $\{x \mid (x,x) \in r^{\mathcal{I}}\}$ |

# Nominals and Self-Love

For all individual names $a$ and role names $r$, the following are concepts:

| Name: | nominal | local reflexivity |
|---|---|---|
| Syntax: | $\{a\}$ | $\exists r.\mathsf{Self}$ |
| Semantics: | $\{a^{\mathcal{I}}\}$ | $\{x \mid (x, x) \in r^{\mathcal{I}}\}$ |

$\exists \mathit{employedBy}.\{\mathit{VUAmsterdam}\}$ $\qquad$ $\exists \mathit{loves}.\mathsf{Self}$

# Extensional Definitions

Nominals allow to provide extensional definitions in addition to the intensional ones.

# Extensional Definitions

Nominals allow to provide extensional definitions in addition to the intensional ones.

Intensional definitions consist of the superclass(es) and any distinguishing characteristics.

> A cat is a mammal that has claws, 4 legs, and a tail.
>
> A carnivore is an animal that eats only meat.
>
> A pet is a domesticated animal that lives with humans.

# Extensional Definitions

Nominals allow to provide extensional definitions in addition to the intensional ones.

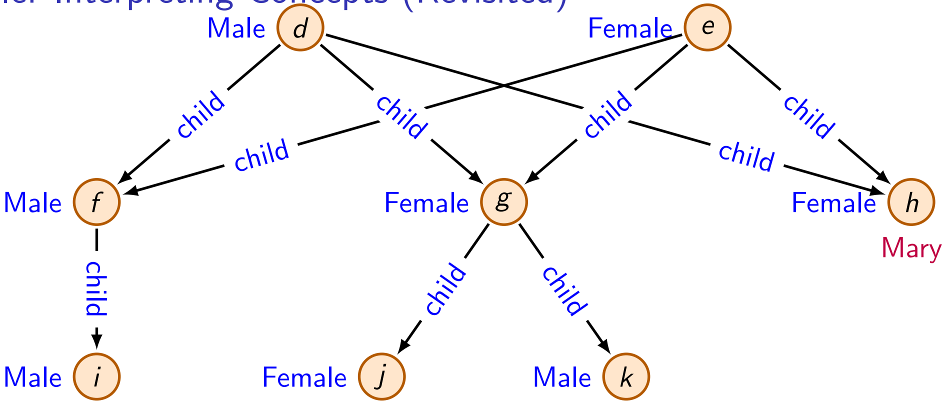
Intensional definitions consist of the superclass(es) and any distinguishing characteristics.

> A cat is a mammal that has claws, 4 legs, and a tail.
>
> A carnivore is an animal that eats only meat.
>
> A pet is a domesticated animal that lives with humans.

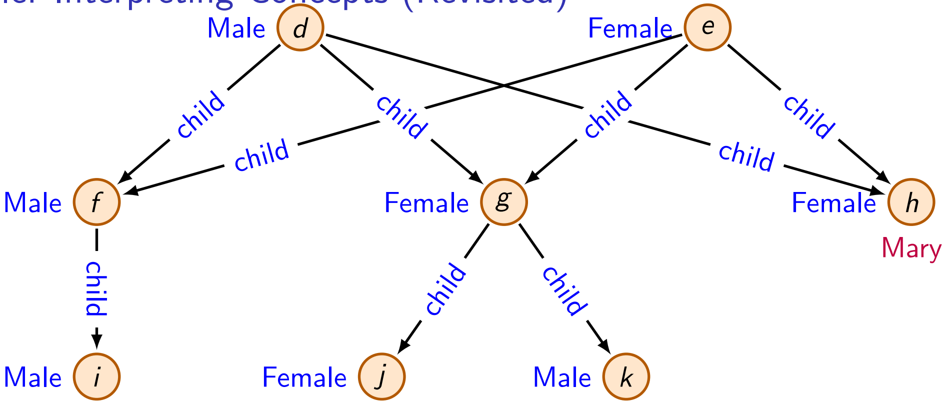
Extensional definitions instead list the elements of the class.

> $\mathit{EUMember} \equiv \{\mathit{France}\} \sqcup \{\mathit{Germany}\} \sqcup \{\mathit{Italy}\} \sqcup \ldots$

# Example: Interpreting Concepts (Revisited)



# Example: Interpreting Concepts (Revisited)




$(\geq 2 child.Female)^{\mathcal{I}} =$

$(\exists child.\mathrm{Self})^{\mathcal{I}} =$

$(Male \sqcap \exists child^{-}.\exists child.Female)^{\mathcal{I}} =$

$(\neg\exists child^{-}.\top)^{\mathcal{I}} =$

$(\exists child.\{Mary\})^{\mathcal{I}} =$

# Additional Assertions

For all individual names $a, b$ and role names $r$, the following are assertions:

| Name | equality | inequality | negated role assertion |
|---|---|---|---|
| Syntax | $a \approx b$ | $a \not\approx b$ | $(a, b) : \neg r$ |
| Semantics | $a^{\mathcal{I}} = b^{\mathcal{I}}$ | $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ | $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$ |

$\mathit{anna} \not\approx \mathit{tom}$ $\qquad$ $\mathit{morningStar} \approx \mathit{eveningStar}$ $\qquad$ $(\mathit{Ernie}, \mathit{Bert}) : \neg\mathit{hasBrother}$

# Additional Assertions

For all individual names $a, b$ and role names $r$, the following are assertions:

| Name | equality | inequality | negated role assertion |
|---|---|---|---|
| Syntax | $a \approx b$ | $a \not\approx b$ | $(a, b) : \neg r$ |
| Semantics | $a^{\mathcal{I}} = b^{\mathcal{I}}$ | $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ | $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$ |

$\mathit{anna} \not\approx \mathit{tom}$ $\quad$ $\mathit{morningStar} \approx \mathit{eveningStar}$ $\quad$ $(\mathit{Ernie}, \mathit{Bert}) : \neg \mathit{hasBrother}$

In fact, all assertions are now syntactic sugar:

$$
\begin{array}{lcl}
a : C & \iff & \{a\} \sqsubseteq C \\
a \approx b & \iff & \{a\} \sqsubseteq \{b\} \\
a \not\approx b & \iff & \{a\} \sqsubseteq \neg\{b\}
\end{array}
\qquad
\begin{array}{lcl}
(a, b) : r & \iff & \{a\} \sqsubseteq \exists r.\{b\} \\
(a, b) : \neg r & \iff & \{a\} \sqsubseteq \forall r.\neg\{b\}
\end{array}
$$

# Role Axioms

With the concept constructors so far, a range of role axioms can be expressed:

| Name | Syntax | Meaning |
|---|---|---|
| Domain | $\mathrm{dom}(r) \sqsubseteq C$ | $\exists r.\top \sqsubseteq C$ |
| Range | $\mathrm{ran}(r) \sqsubseteq C$ | $\exists r^{-}.\top \sqsubseteq C, \quad \top \sqsubseteq \forall r.C$ |
| Functionality | $\mathrm{fun}(r)$ | $\top \sqsubseteq {\leq}1 r.\top$ |
| Reflexivity | $\mathrm{Ref}(r)$ | $\top \sqsubseteq \exists r.\mathrm{Self}$ |

# Role Axioms

In $\mathcal{SROIQ}(D)$, an ontology consists of three parts $\mathcal{O} = \mathcal{A} \cup \mathcal{T} \cup \mathcal{R}$, where $\mathcal{R}$ is an RBox, i.e. a finite set of role axioms.

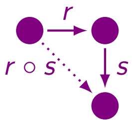
# Role Axioms

In $\mathcal{SROIQ}(D)$, an ontology consists of three parts $\mathcal{O} = \mathcal{A} \cup \mathcal{T} \cup \mathcal{R}$, where $\mathcal{R}$ is an RBox, i.e. a finite set of role axioms.

If $r, s, s_1, \dots, s_n$ are roles, then the following are role axioms:

| Name: | role inclusion | complex role inclusion | role disjointness |
|---|---|---|---|
| Syntax: | $r \sqsubseteq s$ | $s_1 \circ \dots \circ s_n \sqsubseteq r$ | $\mathrm{dis}(r, s)$ |
| Semantics: | $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ | $s_1^{\mathcal{I}} \circ \dots \circ s_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$ | $r^{\mathcal{I}} \cap s^{\mathcal{I}} = \emptyset$ |

$r \circ s$ represents the concatenation of $r$ and $s$:



# Role Axioms

Things we can express in an RBox:

- sub-roles and chains:

  $$\mathit{hasMother} \sqsubseteq \mathit{hasParent} \qquad \mathit{hasParent} \circ \mathit{hasMother} \sqsubseteq \mathit{hasGrandMother}$$

- "Inverse of":

  $$\mathit{hasParent} \sqsubseteq \mathit{hasChild}^{-} \qquad \mathit{hasChild}^{-} \sqsubseteq \mathit{hasParent}$$

- Transitivity:

  $$\mathit{partOf} \circ \mathit{partOf} \sqsubseteq \mathit{partOf}$$

- Other features:

  $$\mathrm{dis}(\mathit{hasDaughter}, \mathit{hasSon}) \qquad \mathrm{Ref}(\mathit{hasRelative})$$

# Additional Axioms: Syntactic Sugar

| Name | Syntax | Defined as |
|---:|---|---|
| disjointness | $\mathsf{dis}(C, D)$ | $C \sqsubseteq \neg D$ or $D \sqsubseteq \neg C$ or $C \sqcap D \sqsubseteq \bot$ |
| role equivalence | $r \equiv s$ | $r \sqsubseteq s$, $s \sqsubseteq r$ |
| domain restriction | $\mathsf{dom}(r) \sqsubseteq C$ | $\top \sqsubseteq \forall r^{-}.C$ or $\exists r.\top \sqsubseteq C$ |
| range restriction | $\mathsf{ran}(r) \sqsubseteq C$ | $\top \sqsubseteq \forall r.C$ or $\exists r^{-}.\top \sqsubseteq C$ |
| role irreflexivity | $\mathsf{irr}(r)$ | $\exists r.\mathsf{Self} \sqsubseteq \bot$ |
| role functionality | $\mathsf{fun}(r)$ | $\top \sqsubseteq {\leq}1 r.\top$ |
| role symmetry | $\mathsf{sym}(r)$ | $r \sqsubseteq r^{-}$ |
| role asymmetry | $\mathsf{asy}(r)$ | $\mathsf{dis}(r, r^{-})$ |
| role transitivity | $\mathsf{tra}(r)$ | $r \circ r \sqsubseteq r$ |

# Note: Domain and Range Restrictions

Be careful of declared domains and ranges. They affect all class expressions using the property:

$\mathrm{ran}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\top \sqsubseteq \forall \mathit{eats}.\mathit{Organism}$)

$\mathit{Bird} \sqsubseteq \exists \mathit{eats}.\mathit{Stone}$

# Note: Domain and Range Restrictions

Be careful of declared domains and ranges. They affect all class expressions using the property:

$$\mathrm{ran}(\mathit{eats}) \sqsubseteq \mathit{Organism} \qquad \text{(equivalent to } \top \sqsubseteq \forall \mathit{eats}.\mathit{Organism}\text{)}$$
$$\mathit{Bird} \sqsubseteq \exists \mathit{eats}.\mathit{Stone}$$

implies that some stones are organisms (those that are eaten by birds).

# Note: Domain and Range Restrictions

Be careful of declared domains and ranges. They affect all class expressions using the property:

$\mathrm{ran}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\top \sqsubseteq \forall \mathit{eats}.\mathit{Organism}$)

$\mathit{Bird} \sqsubseteq \exists \mathit{eats}.\mathit{Stone}$

implies that some stones are organisms (those that are eaten by birds).

If *Stone* and *Organism* are disjoint, then Bird is unsatisfiable.

# Note: Domain and Range Restrictions

Be careful of declared domains and ranges. They affect all class expressions using the property:

$\mathrm{ran}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\top \sqsubseteq \forall \mathit{eats}.\mathit{Organism}$)
$\mathit{Bird} \sqsubseteq \exists \mathit{eats}.\mathit{Stone}$

implies that some stones are organisms (those that are eaten by birds).

If *Stone* and *Organism* are disjoint, then Bird is unsatisfiable.

$\mathrm{dom}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\exists \mathit{eats}.\top \sqsubseteq \mathit{Organism}$)
$\mathit{Tornado} \sqsubseteq \exists \mathit{eats}.\mathit{House}$

# Note: Domain and Range Restrictions

Be careful of declared domains and ranges. They affect all class expressions using the property:

$\mathrm{ran}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\top \sqsubseteq \forall \mathit{eats}.\mathit{Organism}$)

$\mathit{Bird} \sqsubseteq \exists \mathit{eats}.\mathit{Stone}$

implies that some stones are organisms (those that are eaten by birds).

If *Stone* and *Organism* are disjoint, then Bird is unsatisfiable.

$\mathrm{dom}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\exists \mathit{eats}.\top \sqsubseteq \mathit{Organism}$)

$\mathit{Tornado} \sqsubseteq \exists \mathit{eats}.\mathit{House}$

entails $\mathit{Tornado} \sqsubseteq \mathit{Organism}$.

If *Tornado* and *Organism* are disjoint, then *Tornado* becomes unsatisfiable.

# Concrete Domains

Examples of concrete domains are strings, numbers, dates.

In DLs with concrete domains (including $\mathcal{SROIQ}(D)$), there are special role names (attributes, or data properties in OWL terminology) that refer to elements in the concrete domain.

In addition, we have special predicates, that can be used to refer to sets in the concrete domain.

- For example: numbers larger than 10, strings starting with "Mrs. ", etc.

$$\exists \mathit{hasAge}.{\leq_{18}} \qquad \exists \mathit{hasPrice}.{\geq_{1{,}000€}} \qquad \exists \mathit{hasSize}.\{30\}$$

The formal definition is a bit involved, which is why we skip it here.

# $\mathcal{SROIQ}(D)$ concepts in OWL

| DL syntax | Manchester syntax | Remark |
|---|---|---|
| $\top$ | owl:Thing | (a special named class) |
| $\bot$ | owl:Nothing | (a special named class) |
| $C \sqcap D$ | $C$ `and` $D$ | |
| $C \sqcup D$ | $C$ `or` $D$ | |
| $\neg C$ | `not` $C$ | |
| $\exists r.C$ | $r$ `some` $C$ | (similarly for data properties) |
| $\forall r.C$ | $r$ `only` $C$ | (similarly for data properties) |
| $\geq nr.C$ | $r$ `min` $n$ $C$ | (similarly for data properties) |
| $\leq nr.C$ | $r$ `max` $n$ $C$ | (similarly for data properties) |
| $r^-$ | `inverse` $r$ | |
| $\{a\}$ | $\{a\}$ | (similar for data values) |
| $\exists r.\mathsf{Self}$ | $r$ `Self` | |

# Decidability

Apart from adding constructors and axioms to $\mathcal{ALC}$, $\mathcal{SROIQ}(D)$ imposes several restrictions on the use of roles, to retain decidability.

- The RBox must be regular.
- Number restrictions, self restrictions, and disjoint role axioms can only contain simple roles.

# Regular RBoxes

Intuitively, an RBox is regular if there are no cyclic dependencies between role names.

> The RBox $\{\mathit{hasFather} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle},$
> $\mathit{hasChild} \circ \mathit{hasUncle} \sqsubseteq \mathit{hasBrother}\}$ is not regular.

# Regular RBoxes

Intuitively, an RBox is regular if there are no cyclic dependencies between role names.

The RBox $\{\mathit{hasFather} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle},$
$\mathit{hasChild} \circ \mathit{hasUncle} \sqsubseteq \mathit{hasBrother}\}$ is not regular.

Certain cycles are however allowed:

- $r \circ r \sqsubseteq r$ to express transitivity
- $r_1 \circ \ldots \circ r_n \circ r \sqsubseteq r$ (role directly at the end of the chain)
- $r \circ r_1 \circ \ldots \circ r_n \sqsubseteq r$ (role directly at the beginning of the chain)

# Regular RBoxes

Intuitively, an RBox is regular if there are no cyclic dependencies between role names.

The RBox $\{\mathit{hasFather} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle},$
$\mathit{hasChild} \circ \mathit{hasUncle} \sqsubseteq \mathit{hasBrother}\}$ is not regular.

Certain cycles are however allowed:

- $r \circ r \sqsubseteq r$ to express transitivity
- $r_1 \circ \ldots \circ r_n \circ r \sqsubseteq r$ (role directly at the end of the chain)
- $r \circ r_1 \circ \ldots \circ r_n \sqsubseteq r$ (role directly at the beginning of the chain)

Non-regular RBoxes make reasoning undecidable and are therefore forbidden in $\mathcal{SROIQ}(D)$!

# Simple Roles

Apart from adding constructors and axioms to $\mathcal{ALC}$, $\mathcal{SROIQ}(D)$ imposes several restrictions on the use of roles, to retain decidability.

- The RBox must be regular.
- Number restrictions, self restrictions, and disjoint role axioms can only contain simple roles.

# Simple Roles

Apart from adding constructors and axioms to $\mathcal{ALC}$, $\mathcal{SROIQ}(D)$ imposes several restrictions on the use of roles, to retain decidability.

- The RBox must be regular.
- Number restrictions, self restrictions, and disjoint role axioms can only contain simple roles.

The set of non-simple roles is inductively defined as follows:

- If $r_1 \circ \cdots \circ r_n \sqsubseteq r \in \mathcal{R}$ with $n \geq 2$, then $r$ and $r^-$ are non-simple.
- If $s \sqsubseteq r \in \mathcal{R}$ and $s$ is non-simple, then $r$ and $r^-$ are non-simple.

All other roles are simple.

# Simple Roles

Apart from adding constructors and axioms to $\mathcal{ALC}$, $\mathcal{SROIQ}(D)$ imposes several restrictions on the use of roles, to retain decidability.

- The RBox must be regular.
- Number restrictions, self restrictions, and disjoint role axioms can only contain simple roles.

The set of non-simple roles is inductively defined as follows:

- If $r_1 \circ \cdots \circ r_n \sqsubseteq r \in \mathcal{R}$ with $n \geq 2$, then $r$ and $r^-$ are non-simple.
- If $s \sqsubseteq r \in \mathcal{R}$ and $s$ is non-simple, then $r$ and $r^-$ are non-simple.

All other roles are simple.

Transitive roles and roles that have transitive subroles are not simple.

# Example: Partonomies

When defining partOf-relations, it is more useful to refer to the direct parts only, instead of all (indirect) sub-parts.

$$Piston \sqsubseteq \exists directPartOf.Engine \qquad Engine \sqsubseteq \exists directPartOf.Car$$

# Example: Partonomies

When defining partOf-relations, it is more useful to refer to the direct parts only, instead of all (indirect) sub-parts.

$$Piston \sqsubseteq \exists directPartOf.Engine \qquad Engine \sqsubseteq \exists directPartOf.Car$$

directPartOf is not transitive, but has a transitive super role.

$$directPartOf \sqsubseteq partOf \qquad \mathrm{tra}(partOf)$$

# Example: Partonomies

When defining partOf-relations, it is more useful to refer to the direct parts only, instead of all (indirect) sub-parts.

$$\mathit{Piston} \sqsubseteq \exists \mathit{directPartOf}.\mathit{Engine} \qquad \mathit{Engine} \sqsubseteq \exists \mathit{directPartOf}.\mathit{Car}$$

directPartOf is not transitive, but has a transitive super role.

$$\mathit{directPartOf} \sqsubseteq \mathit{partOf} \qquad \mathrm{tra}(\mathit{partOf})$$

This separation allows us to use $\mathit{directPartOf}$ in number restrictions.

$$\top \sqsubseteq {\leq}1\mathit{directPartOf}.\top \qquad \mathit{Car} \sqsubseteq {\leq}4\mathit{hasDirectPart}.\mathit{Wheel} \sqcap \ldots$$

This is not possible for $\mathit{partOf}$, since non-simple roles are not allowed in number restrictions!

# Caution with Using Too Much Expressivity

- In practice, decidability is not everything:
  - Reasoning in $\mathcal{SROIQ}(D)$ is harder than in $\mathcal{ALC}$
  - $\Rightarrow$ Reasoners may struggle with too much
- In general, one should be cautious with over-using constructs
- If reasoning becomes slow, ask yourself:
  - Do I really need a nominal, or would a concept name be sufficient?
  - Can I avoid number restrictions?
  - Can I avoid disjunction?

# Expressivity in OWL

OWL defines different subsets, called profiles, with different use cases:

- OWL DL — based on $\mathcal{SROIQ}(D)$, expressive while decidable
- OWL EL — based on $\mathcal{ELHO}$, optimized for classifying large TBoxes
- OWL RL — optimized for materializing large ABoxes
- OWL QL — optimized for querying large ABoxes

# End of Part I

DLs are a very important formalism for KR

- Many use cases of ontologies to work with data and knowledge
- High expressiveness within the boundaries of decidability
- Fast reasoning with modern DL reasoners
  - Important examples: ELK (for $\mathcal{ELH}$), HermiT and Konclude (the fastest reasoner)

# End of Part I

DLs are a very important formalism for KR

- Many use cases of ontologies to work with data and knowledge
- High expressiveness within the boundaries of decidability
- Fast reasoning with modern DL reasoners
  - Important examples: ELK (for $\mathcal{ELH}$), HermiT and Konclude (the fastest reasoner)

But DLs are not the formalism of choice for every use case

- Another important formalism are rule-based languages (datalog, ASP)

# End of Part I

DLs are a very important formalism for KR

- Many use cases of ontologies to work with data and knowledge
- High expressiveness within the boundaries of decidability
- Fast reasoning with modern DL reasoners
  - Important examples: ELK (for $\mathcal{ELH}$), HermiT and Konclude (the fastest reasoner)

But DLs are not the formalism of choice for every use case

- Another important formalism are rule-based languages (datalog, ASP)

Some limitations come through the foundation on first-order logic:

- do not deal well with contradictions
- cannot represent probabilities