Knowledge Representation for the Semantic Web
Lecture 5: Description Logics IV

Daria Stepanova

slides based on Reasoning Web 2011 tutorial “Foundations of Description Logics and OWL” by S. Rudolph

Max Planck Institute for Informatics
D5: Databases and Information Systems group

WS 2017/18
Unit Outline

Satisfaction and Entailment

Other Reasoning Problems

Algorithmic Approaches to DL Reasoning
Satisfaction and Satisfiability
Satisfaction and Satisfiability of Knowledge Bases

Satisfaction of a KB by an interpretation

An interpretation $\mathcal{I}$ satisfies (or is a model of) a knowledge base $\mathcal{K} = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{I}$ satisfies every axiom of $\mathcal{K}$, i.e., $\mathcal{I} \models \alpha$ for $\alpha \in \mathcal{K}$.
Satisfaction and Satisfiability of Knowledge Bases

**Satisfaction of a KB by an interpretation**

An interpretation $\mathcal{I}$ satisfies (or is a model of) a knowledge base $\mathcal{K} = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{I}$ satisfies every axiom of $\mathcal{K}$, i.e., $\mathcal{I} \models \alpha$ for $\alpha \in \mathcal{K}$.

**KB (un)satisfiability / (in)consistency**

A KB $\mathcal{K}$ is satisfiable (also: consistent), if it has some model; otherwise it is unsatisfiable (also: inconsistent or contradictory).
Satisfaction and Satisfiability of Knowledge Bases

Satisfaction of a KB by an interpretation
An interpretation $\mathcal{I}$ satisfies (or is a model of) a knowledge base $\mathcal{K} = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{I}$ satisfies every axiom of $\mathcal{K}$, i.e., $\mathcal{I} \models \alpha$ for $\alpha \in \mathcal{K}$.

KB (un)satisfiability / (in)consistency
A KB $\mathcal{K}$ is satisfiable (also: consistent), if it has some model; otherwise it is unsatisfiable (also: inconsistent or contradictory).

- unsatisfiability of a KB hints at a design bug
- unsatisfiable axioms carry no information:

  $\alpha$ is unsatisfiable $\iff$ $\neg \alpha$ is tautologic (if negation is applicable),
  i.e., $\mathcal{I} \models \neg \alpha$ for every interpretation $\mathcal{I}$
### Example: KB Satisfiability

**RBox** $\mathcal{R}$

<table>
<thead>
<tr>
<th>owns $\sqsubseteq$ caresFor</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;If somebody owns something, s/he cares for it.&quot;</td>
</tr>
</tbody>
</table>

**TBox** $\mathcal{T}$

<table>
<thead>
<tr>
<th>Healthy $\sqsubseteq$ $\neg$Dead</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Healthy beings are not dead.&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cat $\sqsubseteq$ Dead $\sqcap$ Alive</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Every cat is dead or alive.&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HappyCatOwner $\sqsubseteq$ $\exists$owns.Cat $\sqcap$$\forall$caresFor.Healthy</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A happy cat owner owns a cat and all beings he cares for are healthy.&quot;</td>
</tr>
</tbody>
</table>

**ABox** $\mathcal{A}$

<table>
<thead>
<tr>
<th>HappyCatOwner(schroedinger)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Schrödinger is a happy cat owner.&quot;</td>
</tr>
</tbody>
</table>

Is $\mathcal{K} = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ satisfiable?
Example: KB Satisfiability

**RBox** \( \mathcal{R} \)

\[
\text{owns} \sqsubseteq \text{caresFor}
\]

"If somebody owns something, s/he cares for it."

**TBox** \( \mathcal{T} \)

\[
\text{Healthy} \sqsubseteq \neg \text{Dead}
\]

"Healthy beings are not dead."

\[
\text{Cat} \sqsubseteq \text{Dead} \sqcap \text{Alive}
\]

"Every cat is dead or alive."

\[
\text{HappyCatOwner} \sqsubseteq \exists \text{owns}. \text{Cat} \sqcap \forall \text{caresFor}. \text{Healthy}
\]

"A happy cat owner owns a cat and all beings he cares for are healthy."

**ABox** \( \mathcal{A} \)

\[
\text{HappyCatOwner}(\text{schroedinger})
\]

"Schrödinger is a happy cat owner."

Is \( \mathcal{K} = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle \) satisfiable? Yes!
**Example: KB Satisfiability**

**TBox $\mathcal{T}$**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deer $\sqsubseteq$ Mammal</td>
<td>&quot;Deers are mammals.&quot;</td>
</tr>
<tr>
<td>Mammal $\sqcap$ Flies $\sqsubseteq$ Bat</td>
<td>&quot;Mammals, who fly are bats.&quot;</td>
</tr>
<tr>
<td>Bat $\sqsubseteq \forall$ worksFor.{batman}</td>
<td>&quot;Bats work only for Batman&quot;</td>
</tr>
</tbody>
</table>

**ABox $\mathcal{A}$**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deer $\sqcap \exists$ hasNose.Red(rudolph)</td>
<td>&quot;Rudolph is a deer with a red nose.&quot;</td>
</tr>
<tr>
<td>$\forall$ worksFor$^\bot$.($\neg$Deer $\sqcup$ Flies)(santa)</td>
<td>&quot;Only non-deers or fliers work for Santa.&quot;</td>
</tr>
<tr>
<td>worksFor(rudolph, santa)</td>
<td>&quot;Rudolph works for Santa.&quot;</td>
</tr>
<tr>
<td>santa $\neq$ batman</td>
<td>&quot;Santa is different from Batman.&quot;</td>
</tr>
</tbody>
</table>

Is $\mathcal{K}$ satisfiable?
Example: KB Satisfiability

**TBox** $\mathcal{T}$

- **Deer ⊑ Mammal**
  "Deers are mammals."

- **Mammal ∩ Flies ⊑ Bat**
  "Mammals, who fly are bats."

- **Bat ⊑ ∀worksFor.\{batman\}**
  "Bats work only for Batman"

**ABox** $\mathcal{A}$

- **Deer ∩ ∃hasNose.Red(rudolph)**
  "Rudolph is a deer with a red nose."

- **∀worksFor^-. (¬Deer ∩ Flies)(santa)**
  "Only non-deers or fliers work for Santa."

- **worksFor(rudolph, santa)**
  "Rudolph works for Santa."

- **santa ≠ batman**
  "Santa is different from Batman."

Is $\mathcal{K}$ satisfiable? **No!**
Entailment of Axioms

Entailment checking

A knowledge base $\mathcal{K}$ entails an axiom $\alpha$ (in symbols, $\mathcal{K} \models \alpha$), if every model $\mathcal{I}$ of $\mathcal{K}$ satisfies $\alpha$.

- Informally, $\mathcal{K} \models \alpha$ elicits implicit knowledge
- If $\alpha$ occurs in $\mathcal{K}$, then trivially $\mathcal{K} \models \alpha$
- If $\mathcal{K}$ is unsatisfiable, then $\mathcal{K} \models \alpha$ for every axiom $\alpha$
### Example: Entailment

<table>
<thead>
<tr>
<th>RBox $\mathcal{R}$</th>
<th>caresFor</th>
<th>owns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;If somebody owns something, s/he cares for it.&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TBox $\mathcal{T}$</th>
<th>Healthy ⊑ ¬Dead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Healthy beings are not dead.&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TBox $\mathcal{T}$</th>
<th>Dead ▍ Alive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Every cat is dead or alive.&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TBox $\mathcal{T}$</th>
<th>HappyCatOwner ⊑ ∃owns.Cat ⊓ ∀caresFor.Healthy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;A happy cat owner owns a cat and all beings he cares for are healthy.&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABox $\mathcal{A}$</th>
<th>HappyCatOwner(schroedinger)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Schrödinger is a happy cat owner.&quot;</td>
</tr>
</tbody>
</table>
### Example: Entailment

**RBox R**

- `owns` ⊑ `caresFor`

"If somebody owns something, s/he cares for it."

**TBox T**

- `Healthy` ⊑ `¬Dead`
  "Healthy beings are not dead."
- `Cat` ⊑ `Dead ⊔ Alive`
  "Every cat is dead or alive."
- `HappyCatOwner` ⊑ `∃owns.Cat ⊓ ∀caresFor.Healthy`
  "A happy cat owner owns a cat and all beings he cares for are healthy."

**ABox A**

- `HappyCatOwner(schroedinger)`
  "Schrödinger is a happy cat owner."

- $\mathcal{K} \models \exists caresFor.(Cat ⊓ Alive)(schroedinger)$?
### Example: Entailment

#### RBox $\mathcal{R}$

- `owns` $\sqsubseteq$ `caresFor`
- "If somebody owns something, s/he cares for it."

#### TBox $\mathcal{T}$

- `Healthy` $\sqsubseteq$ `¬Dead`
  - "Healthy beings are not dead."
- `Cat` $\sqsubseteq$ `Dead $\sqcup$ Alive`
  - "Every cat is dead or alive."
- `HappyCatOwner` $\sqsubseteq$ `∃owns.Cat` $\cap$ `∀caresFor.Healthy`
  - "A happy cat owner owns a cat and all beings he cares for are healthy."

#### ABox $\mathcal{A}$

- `HappyCatOwner(schroedinger)`
  - "Schrödinger is a happy cat owner."

- $\mathcal{K} \models \exists caresFor.(Cat \cap Alive)(schroedinger)$
- $\mathcal{K} \models \forall owns.\neg Cat \sqsubseteq \neg HappyCatOwner$?
## Example: Entailment

### RBox $\mathcal{R}$
- **owns** $\sqsubseteq$ **caresFor**
  
  "If somebody owns something, s/he cares for it."

### TBox $\mathcal{T}$
- **Healthy** $\sqsubseteq$ $\neg$**Dead**
  
  "Healthy beings are not dead."
- **Cat** $\sqsubseteq$ **Dead** $\sqcup$ **Alive**
  
  "Every cat is dead or alive."
- **HappyCatOwner** $\sqsubseteq$ $\exists$**owns**.**Cat** $\cap$$\forall$**caresFor**.**Healthy**
  
  "A happy cat owner owns a cat and all beings he cares for are healthy."

### ABox $\mathcal{A}$
- **HappyCatOwner**(schroedinger)
  
  "Schrödinger is a happy cat owner."

- $\mathcal{K} \models \exists$**caresFor**.(**Cat** $\cap$ **Alive**)**(schroedinger)**
- $\mathcal{K} \models \forall$**owns**.$\neg$**Cat** $\sqsubseteq$ $\neg$**HappyCatOwner**
- $\mathcal{K} \models$ **Cat** $\sqsubseteq$ **Healthy**?
**Example: Entailment**

**RBox** $\mathcal{R}$

- *owns* $\sqsubseteq$ *caresFor*

"If somebody owns something, s/he cares for it."

**TBox** $\mathcal{T}$

- *Healthy* $\sqsubseteq \neg$ *Dead*

"Healthy beings are not dead."

- *Cat* $\sqsubseteq$ *Dead $\sqcup$ Alive*

"Every cat is dead or alive."

- *HappyCatOwner* $\sqsubseteq$ $\exists$ *owns*. *Cat* $\sqcap \forall$ *caresFor*. *Healthy*

"A happy cat owner owns a cat and all beings he cares for are healthy."

**ABox** $\mathcal{A}$

- *HappyCatOwner*(*schroedinger*)

"Schrödinger is a happy cat owner."

- $\mathcal{K} \models \exists$ *caresFor*. (*Cat* $\sqcap$ *Alive*)(*schroedinger*)

- $\mathcal{K} \models \forall$ *owns*. $\neg$ *Cat* $\sqsubseteq \neg$ *HappyCatOwner*

- $\mathcal{K} \not\models$ *Cat* $\sqsubseteq$ *Healthy*
Decidability of DLs

DLs are decidable, i.e., there exists an algorithm that

Given: a KB and an axiom $\alpha$,

Output: “yes” iff $\text{KB} \models \alpha$ and no otherwise.

Likewise, there is a similar algorithm that decides whether an input KB is satisfiable.

Just ask $\text{KB} \models \top \sqsubseteq \bot$: if the answer is “yes”, then KB is unsatisfiable, otherwise it is satisfiable.
Standard Reasoning Problems
Standard DL Reasoning Problems

- **KB Satisfiability:** verify whether the KB is satisfiable

- **Entailment:** verify whether the KB entails a certain axiom
e.g., $\mathcal{K} \models \text{CatOwner}(\text{Schroedinger})$

- **Concept Satisfiability:** verify whether a given concept is (un)satisfiable, e.g., $\mathcal{K} \models \text{Dead} \sqcap \text{Alive} \sqsubseteq \bot$

- **Coherence:** verify whether none of the concepts in the KB is unsatisfiable

- **Classification:** compute the subsumption hierarchy of all atomic concepts, e.g. $\mathcal{K} \models \text{Healthy} \sqsubseteq \neg \text{Dead}$, etc.

- **Instance Retrieval:** retrieve all the individuals known to be instances of a certain concept, e.g., find all $a$, s.t. $\exists \text{caresFor}(a)$
Deciding KB Satisfiability

• deciding KB satisfiability is a basic inference task (the “mother” of all standard reasoning tasks)

• directly needed in the process of KB engineering
  • detect severe modeling errors

• other reasoning tasks can be reduced to checking KB (un)satisfiability (and vice versa)

Theorem 1: Reducing reasoning problems to KB satisfiability

Let $\mathcal{K}$ be a KB and $a$ an individual name not in $\mathcal{K}$. Then

2. $C$ is satisfiable w.r.t. $\mathcal{K}$ iff $\mathcal{K} \cup C(a)$ is satisfiable;

3. $\mathcal{K}$ is coherent iff, for each concept name $C$, $\mathcal{K} \cup C(a)$ is satisfiable;

4. $\mathcal{K} \models A \subseteq B$ iff $\mathcal{K} \cup (A \sqcap \neg B)(a)$ is unsatisfiable;

5. $\mathcal{K} \models B(b)$ iff $\mathcal{K} \cup \neg B(b)$ is unsatisfiable.
Entailment Checking

- used in the KB modeling process to check, whether the specified knowledge has the intended consequences
- used for querying the KB if certain propositions are necessarily true

Reduction of entailment problem $\mathcal{K} \models \alpha$ to checking KB inconsistency (see Th.1) follows the idea of proof by contradiction:

- negate the axiom $\alpha$
- add the negated axiom $\neg \alpha$ to $\mathcal{K}$
- check for inconsistency of the resulting KB $\mathcal{K} \cup \{\neg \alpha\}$

If an axiom cannot be negated directly, its negation can be emulated ($\{\neg \alpha\} \leadsto A_\alpha$).
### Entailment Checking, cont’d

Axiom sets $A_\alpha$ such that $\mathcal{K} \models \alpha$ exactly if $\mathcal{K} \cup A_\alpha$ is unsatisfiable:

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$A_\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \circ \ldots \circ r_n \sqsubseteq r$</td>
<td>${ \neg r(c_0, c_1), r_1(c_0, c_n), \ldots, r_n(c_{n-1}, c_n) }$</td>
</tr>
<tr>
<td>$\text{Dis}(r, r')$</td>
<td>${ r(c_1, c_2), r'(c_1, c_2) }$</td>
</tr>
<tr>
<td>$C \sqsubseteq D$</td>
<td>${ (C \sqcap \neg D)(c) }$ or ${ \top \sqsubseteq \exists u(C \sqcap \neg D) }$</td>
</tr>
<tr>
<td>$C(a)$</td>
<td>${ \neg C(a) }$</td>
</tr>
<tr>
<td>$\neg C(a)$</td>
<td>${ C(a) }$</td>
</tr>
<tr>
<td>$r(a, b)$</td>
<td>${ \neg r(a, b) }$</td>
</tr>
<tr>
<td>$\neg r(a, b)$</td>
<td>${ r(a, b) }$</td>
</tr>
<tr>
<td>$a \approx b$</td>
<td>${ a \not\approx b }$</td>
</tr>
<tr>
<td>$a \not\approx b$</td>
<td>${ a \approx b }$</td>
</tr>
</tbody>
</table>

- Individual names $c$ with possible subscripts are supposed to be fresh\(^1\).
- For GCIIs (third line), the first variant is normally employed; the second is logical equivalent instead of just emulating.

\(^1\)Fresh individuals are those not appearing in the given KB $\mathcal{K}$.  

A concept expression $C$ is called **satisfiable** with respect to a knowledge base $\mathcal{K}$, if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^\mathcal{I} \neq \emptyset$.

- Unsatisfiable atomic concepts normally indicate KB modeling errors.
- Concept satisfiability can be reduced to KB consistency (Th. 1) and non-entailment resp.:
  
  - $C$ is **satisfiable** wrt. $\mathcal{K} \iff \mathcal{K} \cup \{C(a)\}$ is consistent, where $a$ is a fresh individual name
  
  - $C$ is **satisfiable** wrt. $\mathcal{K} \iff \mathcal{K} \not\models C \sqsubseteq \bot$
Concept Satisfiability, cont’d

Entailment of general concept inclusions $C \sqsubseteq D$ and equivalences $C \equiv D$ can be reduced to both concept (un)satisfiability and KB (un)satisfiability.

- $C \sqsubseteq D \iff C \cap \neg D$ is unsatisfiable
  $\iff$ KB $\{C(a), \neg D(a)\}$ is unsatisfiable

- $C \equiv D \iff (C \cap \neg D) \sqcup (D \cap \neg C)$ is unsatisfiable
  $\iff$ both $C \cap \neg D$ and $D \cap \neg C$ are unsatisfiable
  $\iff$ both KBs $\{C(a), \neg D(a)\}$ and $\{D(a), \neg C(a)\}$ are unsatisfiable
Classification

**KB Classification**

Classification of a knowledge base $\mathcal{K}$ is to determine for any two concept names $A, B$, whether $\mathcal{K} \models A \sqsubseteq B$ holds.

- This is useful at KB design time for checking the inferred concept hierarchy. Also, computing this hierarchy once and storing it can speed up further queries.

- Classification can be reduced to checking entailment of GCIs.

- While this requires quadratically many checks, one can often do much better in practice by applying optimizations and exploiting that subsumption is a preorder.
Instance Retrieval

Instance retrieval task is to find all named individuals that are known to be in a certain concept (role).

- \( \text{retrieve}(C, \mathcal{K}) = \{ a \in N_I \mid a^\mathcal{I} \in C^\mathcal{I} \text{ for every model } \mathcal{I} \text{ of } \mathcal{K} \} \)
- \( \text{retrieve}(r, \mathcal{K}) = \{ (a, b) \in N_I^2 \mid (a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I} \text{ for every model } \mathcal{I} \text{ of } \mathcal{K} \} \)

- It can be reduced to checking entailment of as many individual assertions as there are named individuals in \( \mathcal{K} \) i.e., test \( \mathcal{K} \models C(a) \) for each \( a \) occurring in \( \mathcal{K} \) (excluding pathologic cases).

- Depending on the system used and the inference algorithm, this can be done in a much more efficient way (e.g. by a transformation into a database query, in SQL or Datalog).
Novel Reasoning Problems

In recent years, other reasoning tasks have been gaining attention.
Novel Reasoning Problems

In recent years, other reasoning tasks have been gaining attention

- **Conjunctive Query Answering**

  conjunctive queries allow to join pieces of information
  more expressive: union of CQAs (akin to SQL), rules
Novel Reasoning Problems

In recent years, other reasoning tasks have been gaining attention

- **Conjunctive Query Answering**
  
  conjunctive queries allow to join pieces of information
  more expressive: union of CQAs (akin to SQL), rules

- **Inconsistency Handling**
  
  repair inconsistent KB or avoid that $\mathcal{K} \models \alpha$ for every $\alpha$ ('knowledge explosion') if $\mathcal{K}$ is inconsistent by introducing, e.g., new semantics
Novel Reasoning Problems

In recent years, other reasoning tasks have been gaining attention

- **Conjunctive Query Answering**
  
  Conjunctive queries allow to join pieces of information more expressive: union of CQAs (akin to SQL), rules

- **Inconsistency Handling**
  
  Repair inconsistent KB or avoid that $\mathcal{K} \models \alpha$ for every $\alpha$ (‘knowledge explosion’) if $\mathcal{K}$ is inconsistent by introducing, e.g., new semantics

- **Entailment Explanation**
  
  Identify axioms in the knowledge base that support a conclusion $\mathcal{K} \models \alpha$, typically a smallest $\mathcal{K}' \subseteq \mathcal{K}$ such that $\mathcal{K}' \models \alpha$ (‘axiom pinpointing’)

Conjunctive Query Answering

Generalize Instance Retrieval by allowing joins and projections:

\[ q(Z) = \exists Y \exists X . \text{childOf}(Z, Y) \land \text{childOf}(Z, X) \land \text{marriedWith}(Y, X) \]

- In databases:
  - just one model (the DB itself) by *Closed World Assumption* (R. Reiter, 1978: if atom \( A \) is not provable from DB, \( \neg A \) is true).

- this is rather easy
Conjunctive Query Answering, cont’d

Generalize Instance Retrieval by allowing joins and projections:

\[ q(Z) = \exists Y \exists X. \text{childOf}(Z, Y) \land \text{childOf}(Z, X) \land \text{marriedWith}(Y, X) \]

- In Description Logics:
  - one knowledge base, many models (*Open World Assumption*)

- not so easy

- the \( \exists \)-variables must be suitably mapped in *every* model
Algorithmic Approaches to DL Reasoning
Types of Reasoning Procedures

- Roughly, DL inference algorithms can be separated into two groups:
  - **model-based algorithms**: show satisfiability by constructing a model (or a representation of it).
    - **Examples**: tableau, automata, type elimination algorithms
  - **proof-based algorithms**: apply deduction rules to the KB to infer new axioms.
    - **Examples**: resolution, consequence-based algorithms

**Note:**
- both strategies are known from first-order logic theorem proving
- additional care is needed to ensure decidability (in particular completeness and termination of algorithms).
Types of Reasoning Procedures

• Roughly, DL inference algorithms can be separated into two groups:

  model-based algorithms: show satisfiability by constructing a model (or a representation of it).

    Examples: tableau, automata, type elimination algorithms

  proof-based algorithms: apply deduction rules to the KB to infer new axioms.

    Examples: resolution, consequence-based algorithms

Note:

• both strategies are known from first-order logic theorem proving

• additional care is needed to ensure decidability (in particular completeness and termination of algorithms).
Tableau Algorithm for DLs

Tableau-based techniques

They try to decide the satisfiability of a formula (or theory) by using rules to construct (a representation of) a model.

- Tableau-based techniques have been used in FOL and modal logics for many years.

- For DLs, they have been extensively explored since the late 1990s [Smolka, 1990], [Baader and Sattler, 2001].

- They are considered well-suited for implementation.

- In fact, many of the most successful DL reasoners implement tableau techniques or variations of them, e.g.: RACER, FaCT++, Pellet, Hermit, etc.
Tableau Algorithm for Deciding KB Satisfiability

- Perform a "bottom-up" construction of a model:
Tableau Algorithm for Deciding KB Satisfiability

- Perform a "bottom-up" construction of a model:
  - Initialize an interpretation by all explicitly known (i.e., named) individuals and their known properties.
Tableau Algorithm for Deciding KB Satisfiability

- Perform a "bottom-up" construction of a model:
  - Initialize an interpretation by all explicitly known (i.e., named) individuals and their known properties.
  - Most probably, this "model draft" will violate some of the axioms.
  - Iteratively "repair" it by adding new information about concept or role memberships and/or introducing new (i.e., anonymous) individuals, this may require case distinction and backtracking.
  - If we arrive at an interpretation satisfying all axioms, satisfiability has been shown.
  - If every repairing attempt eventually results in an overt inconsistency, unsatisfiability has been shown.

Note: as the finite model property does not hold in general, not a full model is constructed but a finite representation of it (cf. "blocking").
Tableau Algorithm for Deciding KB Satisfiability

- Perform a "bottom-up" construction of a model:
  - Initialize an interpretation by all explicitly known (i.e., named) individuals and their known properties.
  - Most probably, this "model draft" will violate some of the axioms.
  - Iteratively "repair" it by adding new information about concept or role memberships and/or introducing new (i.e., anonymous) individuals, this may require case distinction and backtracking.
Tableau Algorithm for Deciding KB Satisfiability

- Perform a "bottom-up" construction of a model:
  - Initialize an interpretation by all explicitly known (i.e., named) individuals and their known properties.
  - Most probably, this "model draft" will violate some of the axioms.
  - Iteratively "repair" it by adding new information about concept or role memberships and/or introducing new (i.e., anonymous) individuals, this may require case distinction and backtracking.
  - If we arrive at an interpretation satisfying all axioms, satisfiability has been shown.

Note: as the finite model property does not hold in general, not a full model is constructed but a finite representation of it (cf. "blocking").
Tableau Algorithm for Deciding KB Satisfiability

- Perform a "bottom-up" construction of a model:
  - Initialize an interpretation by all explicitly known (i.e., named) individuals and their known properties.
  - Most probably, this "model draft" will violate some of the axioms.
  - Iteratively "repair" it by adding new information about concept or role memberships and/or introducing new (i.e., anonymous) individuals, this may require case distinction and backtracking.
  - If we arrive at an interpretation satisfying all axioms, satisfiability has been shown.
  - If every repairing attempt eventually results in an overt inconsistency, unsatisfiability has been shown.

**Note:** as the finite model property does not hold in general, not a full model is constructed but a finite representation of it (cf. “blocking”).
### Tableau Overview Example: Happy Cat Owner

<table>
<thead>
<tr>
<th><strong>RBox</strong> $R$</th>
<th><strong>TBox</strong> $T$</th>
<th><strong>ABox</strong> $A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{owns} \sqsubseteq \text{caresFor}$</td>
<td>$\text{Healthy} \sqsubseteq \neg \text{Dead}$</td>
<td>$\text{HappyCatOwner(schroedinger)}$</td>
</tr>
<tr>
<td>&quot;If somebody owns something, s/he cares for it.&quot;</td>
<td>&quot;Healthy beings are not dead.&quot;</td>
<td>&quot;Schrödinger is a happy cat owner.&quot;</td>
</tr>
<tr>
<td>$\text{Cat} \sqsubseteq \text{Dead} \sqcup \text{Alive}$</td>
<td>$\text{HappyCatOwner} \sqsubseteq \exists \text{owns}. \text{Cat} \sqcap \forall \text{caresFor}. \text{Healthy}$</td>
<td></td>
</tr>
<tr>
<td>&quot;Every cat is dead or alive.&quot;</td>
<td>&quot;A happy cat owner owns a cat and all beings he cares for are healthy.&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Is $\mathcal{K}$ satisfiable?
### Tableau Overview Example: Happy Cat Owner

<table>
<thead>
<tr>
<th><strong>RBox R</strong></th>
<th>owns ⊑ caresFor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TBox T</strong></td>
<td>Healthy ⊑ ¬Dead</td>
</tr>
<tr>
<td></td>
<td>Cat ⊑ Dead ⊔ Alive</td>
</tr>
<tr>
<td></td>
<td>HappyCatOwner ⊑ ∃owns.Cat ⊓ ∀caresFor.Healthy</td>
</tr>
<tr>
<td><strong>ABox A</strong></td>
<td>HappyCatOwner((s))</td>
</tr>
</tbody>
</table>

Is \(\mathcal{K}\) satisfiable?
**Tableau Overview Example: Happy Cat Owner**

<table>
<thead>
<tr>
<th>RBox $\mathcal{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>owns</code> $\sqsubseteq$ <code>caresFor</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TBox $\mathcal{T}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Healthy</code> $\sqsubseteq$ <code>¬Dead</code></td>
</tr>
<tr>
<td><code>Cat</code> $\sqsubseteq$ <code>Dead</code> $\sqcup$ <code>Alive</code></td>
</tr>
<tr>
<td><code>HappyCatOwner</code> $\sqsubseteq$ <code>∃owns.Cat</code> $\sqcap$ <code>∀caresFor.Healthy</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABox $\mathcal{A}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>HappyCatOwner(s)</code></td>
</tr>
</tbody>
</table>

Is $\mathcal{K}$ satisfiable?

| $s$ | $\mathcal{L}(s) = \{HappyCatOwner\}$ |

---

**Algorithmic Approaches to DL Reasoning**
# Tableau Overview Example: Happy Cat Owner

**RBox** $\mathcal{R}$

- $\text{owns} \sqsubseteq \text{caresFor}$

**TBox** $\mathcal{T}$

- $\text{Healthy} \sqsubseteq \neg \text{Dead}$
- $\text{Cat} \sqsubseteq \text{Dead} \sqcup \text{Alive}$
- $\text{HappyCatOwner} \sqsubseteq \exists \text{owns}.\text{Cat} \sqcap \forall \text{caresFor}.\text{Healthy}$

**ABox** $\mathcal{A}$

- $\text{HappyCatOwner}(s)$

**Is $\mathcal{K}$ satisfiable?**

\[ \mathcal{L}(s) = \{ \text{HappyCatOwner} \} \]

\[ \mathcal{L}(c) = \{ \text{Cat} \} \]
Tableau Overview Example: Happy Cat Owner

\[ RBox \mathcal{R} \]
- \( \text{owns} \sqsubseteq \text{caresFor} \)

\[ TBox \mathcal{T} \]
- \( \text{Healthy} \sqsubseteq \neg \text{Dead} \)
- \( \text{Cat} \sqsubseteq \text{Dead} \sqcup \text{Alive} \)
- \( \text{HappyCatOwner} \sqsubseteq \exists \text{owns}.\text{Cat} \sqcap \forall \text{caresFor}.\text{Healthy} \)

\[ ABox \mathcal{A} \]
- \( \text{HappyCatOwner}(s) \)

Is \( \mathcal{K} \) satisfiable?

\[ s \]
- \( \mathcal{L}(s) = \{ \text{HappyCatOwner} \} \)

\[ \text{owns} \quad \text{caresFor} \]
- \( \mathcal{L}(c) = \{ \text{Cat} \} \)
Tableau Overview Example: Happy Cat Owner

**RBox** \( \mathcal{R} \)

\[ \text{owns} \sqsubseteq \text{caresFor} \]

**TBox** \( \mathcal{T} \)

\[
\begin{align*}
\text{Healthy} & \sqsubseteq \neg \text{Dead} \\
\text{Cat} & \sqsubseteq \text{Dead} \sqcup \text{Alive} \\
\text{HappyCatOwner} & \sqsubseteq \exists \text{owns}.\text{Cat} \sqcap \forall \text{caresFor}.\text{Healthy}
\end{align*}
\]

**ABox** \( \mathcal{A} \)

\[
\text{HappyCatOwner}(s)
\]

Is \( \mathcal{K} \) satisfiable?

\[
\begin{align*}
s & \bullet \\
\text{owns} & \quad \text{caresFor} \\
\bullet c & \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{L}(s) & = \{ \text{HappyCatOwner} \} \\
\mathcal{L}(c) & = \{ \text{Cat, Healthy} \}
\end{align*}
\]
Tableau Overview Example: Happy Cat Owner

**RBox 𝒫**

\[
\text{owns} \sqsubseteq \text{caresFor}
\]

**TBox 𝒰**

\[
\begin{align*}
\text{Healthy} & \sqsubseteq \neg \text{Dead} \\
\text{Cat} & \sqsubseteq \text{Dead} \sqcup \text{Alive} \\
\text{HappyCatOwner} & \sqsubseteq \exists \text{owns}.\text{Cat} \sqcap \forall \text{caresFor}.\text{Healthy}
\end{align*}
\]

**ABox 𝒮**

\[
\text{HappyCatOwner}(s)
\]

Is 𝒮 satisfiable?

\[
\begin{align*}
\mathcal{L}(s) &= \{ \text{HappyCatOwner} \} \\
\mathcal{L}(c) &= \{ \text{Cat}, \text{Healthy}, \neg \text{Dead} \}
\end{align*}
\]
Tableau Overview Example: Happy Cat Owner

\begin{align*}
\text{RBox } \mathcal{R} & \\
\text{owns} & \sqsubseteq \text{caresFor}
\end{align*}

\begin{align*}
\text{TBox } \mathcal{T} & \\
\text{Healthy} & \sqsubseteq \neg \text{Dead} \\
\text{Cat} & \sqsubseteq \text{Dead} \sqcup \text{Alive} \\
\text{HappyCatOwner} & \sqsubseteq \exists \text{owns}. \text{Cat} \sqcap \forall \text{caresFor}. \text{Healthy}
\end{align*}

\begin{align*}
\text{ABox } \mathcal{A} & \\
\text{HappyCatOwner}(s)
\end{align*}

Is \( \mathcal{K} \) satisfiable?

\[ \mathcal{L}(s) = \{ \text{HappyCatOwner} \} \]

\[ \begin{align*}
\mathcal{L}(c) &= \{ \text{Cat}, \text{Healthy}, \neg \text{Dead}, \text{Alive} \}
\end{align*} \]
### Tableau Overview Example: Happy Cat Owner

**RBox** $\mathcal{R}$

- owns $\sqsubseteq$ caresFor

**TBox** $\mathcal{T}$

- Healthy $\sqsubseteq \neg$Dead
- Cat $\sqsubseteq$ Dead $\sqcup$ Alive
- HappyCatOwner $\sqsubseteq \exists$owns.Cat $\sqcap \forall$caresFor.Healthy

**ABox** $\mathcal{A}$

- HappyCatOwner(s)

Is $\mathcal{K}$ satisfiable? **Yes!**

- $s$
  - $\mathcal{L}(s) = \{\text{HappyCatOwner}\}$

- $c$
  - $\mathcal{L}(c) = \{\text{Cat, Healthy, } \neg\text{Dead, Alive}\}$
Naive Tableau Algorithm for $\mathcal{ALC}$

Given a KB in NNF we construct a tableau, which for $\mathcal{ALC}$ KBs consists of

- a set of nodes, labeled with individual names or variable names
- directed edges between some pairs of nodes
- for each node labeled $x$, a set $\mathcal{L}(x)$ of class expressions and
- for each pair of nodes $x$ and $y$, a set $\mathcal{L}(x, y)$ of role names.

Provisos:

- omit edges which are labeled with the empty set
- assume $\top$ is contained in $\mathcal{L}(x)$ for any $x$.
- concept expressions should be in negation normal form
## Recall $\mathcal{ALC}$ Syntax

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>$A$</td>
<td>$Doctor$</td>
<td>$\mathcal{A}^I \subseteq \Delta^I$</td>
</tr>
<tr>
<td>atomic role</td>
<td>$r$</td>
<td>$hasChild$</td>
<td>$\mathcal{r}^I \subseteq \Delta^I \times \Delta^I$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$C \sqcap D$</td>
<td>$Human \sqcap Male$</td>
<td>$\mathcal{C}^I \cap \mathcal{D}^I$</td>
</tr>
<tr>
<td>unqual. exist. res.$^2$</td>
<td>$\exists r$</td>
<td>$\exists hasChild$</td>
<td>${o \mid \exists o'.(o, o') \in \mathcal{r}^I}$</td>
</tr>
<tr>
<td>value res.</td>
<td>$\forall r.C$</td>
<td>$\forall hasChild.Male$</td>
<td>${o \mid \forall o'.(o, o') \in \mathcal{r}^I \rightarrow o' \in \mathcal{C}^I}$</td>
</tr>
<tr>
<td>full negation</td>
<td>$\neg C$</td>
<td>$\neg \forall hasChild.Male$</td>
<td>$\Delta^I \setminus \mathcal{C}^I$</td>
</tr>
</tbody>
</table>

$^2$Unqualified existential restriction
**Negation Normal Form**

**Negation normal form (NNF)**

A concept expression $C$ is in negation normal form, if negation occurs in $C$ only in front of atomic concepts, nominal concepts and self-restrictions.
Negation Normal Form

Negation normal form (NNF)

A concept expression $C$ is in negation normal form, if negation occurs in $C$ only in front of atomic concepts, nominal concepts and self-restrictions.

Given a KB $\mathcal{K}$ to construct $\text{nnf}(\mathcal{K})$ we need to

- replace every $C \equiv D$ by $C \sqsubseteq D$ and $D \sqsubseteq C$;  
- replace every $C \sqsubseteq D$ by $\neg C \sqcup D$;
Negation Normal Form (NNF)

A concept expression $C$ is in negation normal form, if negation occurs in $C$ only in front of atomic concepts, nominal concepts and self-restrictions.

Given a KB $K$ to construct $\text{nnf}(K)$ we need to

- replace every $C \equiv D$ by $C \subseteq D$ and $D \subseteq C$;
- replace every $C \subseteq D$ by $\neg C \sqcup D$;
- recursively translate every $C$ into $\text{nnf}(C)$:

\[
\begin{align*}
\text{nnf}(C) & \leadsto C \text{ if } C \in \{A, \neg A, \{a_1 \ldots a_n\}, \neg\{a_1 \ldots a_n\}, \exists r. \text{Self}, \neg \exists r. \text{Self}, \top, \bot\} \\
\text{nnf}(C \cap D) & \leadsto \text{nnf}(C) \cap \text{nnf}(D) \\
\text{nnf}(C \sqcup D) & \leadsto \text{nnf}(C) \sqcup \text{nnf}(D) \\
\text{nnf}(\neg(C \sqcup D)) & \leadsto \text{nnf}(\neg C \sqcup \neg D) \\
\text{nnf}(\neg(C \cap D)) & \leadsto \text{nnf}(\neg C \cap \neg D) \\
\text{nnf}(\neg \neg C) & \leadsto \text{nnf}(C) \\
\text{nnf}(\neg \top) & \leadsto \bot \\
\text{nnf}(\forall r. C) & \leadsto \exists r. \text{nnf}(C) \\
\text{nnf}(\exists r. C) & \leadsto \forall r. \text{nnf}(C) \\
\text{nnf}(\leq kr. C) & \leadsto \leq kr. \text{nnf}(C) \\
\text{nnf}(\geq kr. C) & \leadsto \geq kr. \text{nnf}(C)
\end{align*} \]

$\text{nnf}(C)$ and $\text{nnf}(C)$ are logically equivalent, i.e., $C_I = \text{nnf}(C)_I$, for every interpretation $I$, and the translation process terminates in linear time.
Negation Normal Form

Negation normal form (NNF)

A concept expression $C$ is in negation normal form, if negation occurs in $C$ only in front of atomic concepts, nominal concepts and self-restrictions.

Given a KB $\mathcal{K}$ to construct $\text{nff} (\mathcal{K})$ we need to

- replace every $C \equiv D$ by $C \subseteq D$ and $D \subseteq C$;
- replace every $C \subseteq D$ by $\neg C \sqcup D$;
- recursively translate every $C$ into $\text{nff} (C)$:

$$\text{nff}(C) \leadsto C \quad \text{if} \quad C \in \{A, \neg A, \{a_1 \ldots a_n\}, \neg\{a_1 \ldots a_n\}, \exists r.\text{Self}, \neg\exists r.\text{Self}, \top, \bot\}$$

$$\text{nff}(C \sqcap D) \leadsto \text{nff}(C) \sqcap \text{nff}(D)$$

$$\text{nff}(C \sqcup D) \leadsto \text{nff}(C) \sqcup \text{nff}(D)$$

$$\text{nff}(\neg(C \sqcup D)) \leadsto \text{nff}(\neg C \sqcup \neg D)$$

$$\text{nff}(\neg(C \sqcap D)) \leadsto \text{nff}(\neg C \sqcap \neg D)$$

$$\text{nff}(\neg\neg C) \leadsto \text{nff}(C)$$

$$\text{nff}(\exists r.C) \leadsto \exists r.\text{nff}(C)$$

$$\text{nff}(\forall r.C) \leadsto \forall r.\text{nff}(C)$$

$$\text{nff}(\leq kr.C) \leadsto \leq kr.\text{nff}(C)$$

$$\text{nff}(\geq kr.C) \leadsto \geq kr.\text{nff}(C)$$

$$\text{nff}(\neg \top) \leadsto \bot$$

$C$ and $\text{nff} (C)$ are logically equivalent, i.e., $C^\mathcal{I} = \text{nff} (C)^\mathcal{I}$, for every interpretation $\mathcal{I}$, and the translation process terminates in linear time.
Example: Negation Normal Form

**Negation Normal Form**

\[ \text{FilmActor} \subseteq (\exists \text{actedIn} \cap \text{Artist}) \sqcup \neg(\neg \exists \text{actedIn} \sqcup \exists \text{playsIn. Theater}) \]
Example: Negation Normal Form

Negation Normal Form

\[ \neg \text{FilmActor} \sqsubseteq (\exists \text{actedIn} \cap \text{Artist}) \sqcup \neg (\neg \exists \text{actedIn} \sqcup \exists \text{playsIn}.\text{Theater}) \]

1. \[ \neg \text{FilmActor} \sqcup (\exists \text{actedIn} \cap \text{Artist}) \sqcup \neg (\neg \exists \text{actedIn} \sqcup \exists \text{playsIn}.\text{Theater}) \]
Example: Negation Normal Form

Negation Normal Form

FilmActor ⊆ (∃actedIn ∩ Artist) ∪ ¬(¬∃actedIn ∪ ∃playsIn. Theater)

1. ¬FilmActor ∪ (∃actedIn ∩ Artist) ∪ ¬(¬∃actedIn ∪ ∃playsIn. Theater)

2. ¬FilmActor ∪ (∃actedIn ∩ Artist) ∪ (∃actedIn ∪ ¬∃playsIn. Theater)
Example: Negation Normal Form

Negation Normal Form

*FilmActor* $\subseteq (\exists \text{actedIn} \cap \text{Artist}) \cup \neg(\neg\exists \text{actedIn} \cup \exists \text{playsIn}.\text{Theater})$

1. $\neg \text{FilmActor} \cup (\exists \text{actedIn} \cap \text{Artist}) \cup \neg(\neg\exists \text{actedIn} \cup \exists \text{playsIn}.\text{Theater})$

2. $\neg \text{FilmActor} \cup (\exists \text{actedIn} \cap \text{Artist}) \cup (\exists \text{actedIn} \cap \neg \exists \text{playsIn}.\text{Theater})$

3. $\neg \text{FilmActor} \cup (\exists \text{actedIn} \cap \text{Artist}) \cup (\exists \text{actedIn} \cap \forall \text{playsIn}.\neg \text{Theater})$
Initial Tableau

For an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ in negation normal form, the initial tableau is defined as follows:

1. For each individual $a$ occurring in $\mathcal{K}$, create a node labeled $a$ and set $\mathcal{L}(a) = \emptyset$.

2. For all pairs $a, b$ of individuals, set $\mathcal{L}(a, b) = \emptyset$.

3. For each ABox statement $C(a)$ in $\mathcal{K}$, set $\mathcal{L}(a) \leftarrow C$.

4. For each ABox statement $r(a, b)$ in $\mathcal{K}$ set $\mathcal{L}(a, b) \leftarrow r$.

Note: “$\leftarrow$” means “update with”, “add to”
Example: Initial Tableau

\[ \mathcal{K} = \{ A(a), \ (\exists r. B)(a), \ r(a, b), \ r(a, c), \ s(b, b), \ (A \sqcup B)(c), \ \neg A \sqcup (\forall s. B) \} \]
Example: Initial Tableau

\[ \mathcal{K} = \{ A(a), (\exists r.B)(a), r(a, b), r(a, c), s(b, b), (A \sqcup B)(c), \neg A \sqcup (\forall s.B) \} \]

\[ b \bullet \]

\[ a \bullet \quad \mathcal{L}(a) = \{ A, \} \]

\[ c \bullet \]
Example: Initial Tableau

\[ \mathcal{K} = \{ A(a), (\exists r.B)(a), r(a,b), r(a,c), s(b,b), (A \sqcup B)(c), \neg A \sqcup (\forall s.B) \} \]

\[ \mathcal{L}(a) = \{ A, \exists r.B \} \]
Example: Initial Tableau

\[ \mathcal{K} = \{ A(a), (\exists r.B)(a), r(a, b), r(a, c), s(b, b), (A \sqcup B)(c), \neg A \sqcup (\forall s.B) \} \]

\[ \mathcal{L}(a) = \{ A, \exists r.B \} \]
Example: Initial Tableau

\[ \mathcal{K} = \{A(a), (\exists r.B)(a), r(a, b), r(a, c), s(b, b), (A \sqcup B)(c), \neg A \sqcup (\forall s.B)\} \]

\[ \mathcal{L}(a) = \{A, \exists r.B\} \]
Example: Initial Tableau

\[ \mathcal{K} = \{ A(a), (\exists r. B)(a), r(a, b), r(a, c), s(b, b), (A \sqcup B)(c), \neg A \sqcup (\forall s. B) \} \]

\[ \mathcal{L}(a) = \{ A, \exists r. B \} \]

\[ \mathcal{L}(c) = \{ A \sqcup B \} \]
Example: Initial Tableau

\[ \mathcal{K} = \{ A(a), (\exists r.B)(a), r(a, b), r(a, c), s(b, b), (A \sqcup B)(c), \neg A \sqcup (%s.B) \} \]

\[
\begin{align*}
L(b) &= \emptyset \\
L(a) &= \{ A, \exists r.B \} \\
L(c) &= \{ A \sqcup B \}
\end{align*}
\]
Expansion Rules for the Naive Tableau

- $\cap$-rule: If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.
Expansion Rules for the Naive Tableau

- □-rule: If \( C \sqcap D \in \mathcal{L}(x) \) and \( \{C, D\} \not\subseteq \mathcal{L}(x) \), then set 
  \( \mathcal{L}(x) \leftarrow \{C, D\} \).

- ⊃-rule: If \( C \sqcup D \in \mathcal{L}(x) \) and \( \{C, D\} \cap \mathcal{L}(x) = \emptyset \), then set 
  \( \mathcal{L}(x) \leftarrow C \) or \( \mathcal{L}(x) \leftarrow D \).
Expansion Rules for the Naive Tableau

- **¬-rule**: If \( C \cap D \in \mathcal{L}(x) \) and \( \{C, D\} \not\subseteq \mathcal{L}(x) \), then set \( \mathcal{L}(x) \leftarrow \{C, D\} \).

- **∪-rule**: If \( C \cup D \in \mathcal{L}(x) \) and \( \{C, D\} \cap \mathcal{L}(x) = \emptyset \), then set \( \mathcal{L}(x) \leftarrow C \) or \( \mathcal{L}(x) \leftarrow D \).

- **∃-rule**: If \( \exists r.C \in \mathcal{L}(x) \) and there is no \( y \) with \( r \in \mathcal{L}(x, y) \) and \( C \in \mathcal{L}(y) \), then
  1. add a new node with label \( y \) (where \( y \) is a new node label),
  2. set \( \mathcal{L}(x, y) = \{r\} \), and
  3. set \( \mathcal{L}(y) = \{C\} \).

- **TBox-rule**: If \( C \) is a (rewritten and normalized) TBox statement and \( C \not\in \mathcal{L}(x) \), then set \( \mathcal{L}(x) \leftarrow C \).
Expansion Rules for the Naive Tableau

- **⊓-rule:** If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

- **⊔-rule:** If $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

- **∃-rule:** If $\exists r.C \in \mathcal{L}(x)$ and there is no $y$ with $r \in \mathcal{L}(x, y)$ and $C \in \mathcal{L}(y)$, then
  1. add a new node with label $y$ (where $y$ is a new node label),
  2. set $\mathcal{L}(x, y) = \{r\}$, and
  3. set $\mathcal{L}(y) = \{C\}$.

- **∀-rule:** If $\forall r.C \in \mathcal{L}(x)$ and there is a node $y$ with $r \in \mathcal{L}(x, y)$ and $C \not\in \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$. 

- **TBox-rule:** If $C$ is a (rewritten and normalized) TBox statement and $C \not\in \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$. 

Expansion Rules for the Naive Tableau

- **⊓-rule:** If $C \cap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

- **⊔-rule:** If $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

- **∃-rule:** If $\exists r.C \in \mathcal{L}(x)$ and there is no $y$ with $r \in L(x, y)$ and $C \in \mathcal{L}(y)$, then
  1. add a new node with label $y$ (where $y$ is a new node label),
  2. set $\mathcal{L}(x, y) = \{r\}$, and
  3. set $\mathcal{L}(y) = \{C\}$.

- **∀-rule:** If $\forall r.C \in \mathcal{L}(x)$ and there is a node $y$ with $r \in L(x, y)$ and $C \not\in \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

- **TBox-rule:** If $C$ is a (rewritten and normalized) TBox statement and $C \not\in \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$. 
Naive Tableau Algorithm

**Input:** $\mathcal{ALC}$ knowledge base $\mathcal{K}$ in negation normal form

**Output:** “yes” if $\mathcal{K}$ is satisfiable

1. Initialize the tableau;
2. While some expansion rule is applicable:
   2.1. nondeterministically apply an applicable rule;
   2.2. if for some node $x$, there exists some $C \in \mathcal{L}(x)$ such that $\neg C \in \mathcal{L}(x)$,
       output “no” and terminate;
3. Output “yes”.

A nondeterministic run of the algorithm terminates, if either

- for some node $x$, $\mathcal{L}(x)$ contains a contradiction (“no”; attempt to find a model for $\mathcal{K}$ was unsuccessful), or
- no expansion rule is applicable (“yes”; attempt was successful)
- $\mathcal{K}$ is satisfiable if some run outputs “yes”, and unsatisfiable if every run outputs “no”
- only the $\sqcap$-rule creates true branching regarding yes/no output
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r.D, \neg D \sqcup E, \forall r.\neg E(a) \} \]

\[ a \bullet \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r. D, \neg D \sqcup E, \forall r. \neg E(a) \} \]

\[ a \bullet \quad \mathcal{L}(a) = \{ C, \} \]
**Example: Expansion Rules**

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r. D, \neg D \sqcup E, \forall r. \neg E(a) \} \]

\[ a \bullet \quad \mathcal{L}(a) = \{ C, \forall r. \neg E \}, \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r. D, \neg D \sqcup E, \forall r. \neg E(a) \} \]

\[ a \bullet \quad \mathcal{L}(a) = \{ C, \forall r. \neg E, \neg C \sqcup \exists r. D, \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r.D, \neg D \sqcup E, \forall r.\neg E(a)\} \]

\[ a \quad \mathcal{L}(a) = \{ C, \forall r.\neg E, \neg C \sqcup \exists r.D, \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r. D, \neg D \sqcup E, \forall r. \neg E(a) \} \]

\[ \mathcal{L}(a) = \{ C, \forall r. \neg E, \neg C \sqcup \exists r. D, \exists r. D \} \]

\[ \mathcal{L}(x) = \{ D, \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r. D, \neg D \sqcup E, \forall r. \neg E(a) \} \]

\[ \mathcal{L}(a) = \{ C, \forall r. \neg E, \neg C \sqcup \exists r. D, \exists r. D \} \]

\[ \mathcal{L}(x) = \{ D, \neg D \sqcup E, \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{C(a), \neg C \sqcup \exists r . D, \neg D \sqcup E, \forall r . \neg E(a)\} \]

\[ \mathcal{L}(a) = \{C, \forall r . \neg E, \neg C \sqcup \exists r . D, \exists r . D, \} \]

\[ \mathcal{L}(x) = \{D, \neg D \sqcup E, \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r.D, \neg D \sqcup E, \forall r.\neg E(a) \} \]

\[ a \]
\[ r \]
\[ x \]

\[ \mathcal{L}(a) = \{ C, \forall r.\neg E, \neg C \sqcup \exists r.D, \exists r.D \} \]

\[ \mathcal{L}(x) = \{ D, \neg D \sqcup E, E, \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \lnot C \sqcup \exists r.D, \lnot D \sqcup E, \forall r.\lnot E(a) \} \]

\[ \mathcal{L}(a) = \{ C, \forall r.\lnot E, \lnot C \sqcup \exists r.D, \exists r.D \} \]

\[ \mathcal{L}(x) = \{ D, \lnot D \sqcup E, E, \lnot E \} \]
Example: Expansion Rules

\[ \mathcal{K} = \{ C(a), \neg C \sqcup \exists r.D, \neg D \sqcup E, \forall r.\neg E(a) \} \]

\[ \mathcal{L}(a) = \{ C, \forall r.\neg E, \neg C \sqcup \exists r.D, \exists r.D \} \]

\[ \mathcal{L}(x) = \{ D, \neg D \sqcup E, E, \neg E \} \]

Clash is obtained, KB is unsatisfiable!
Tableau Algorithm with Blocking for $\mathcal{ALC}$

- Naive tableau algorithm does not always terminate
  Example: $\mathcal{K} = \{\neg\text{Person} \sqcup \exists \text{hasParent}, \text{Person}(a_1)\}$.

- Modify the naive tableau algorithm to ensure termination

- Use blocking

- A node with label $x$ is **directly blocked** by a node with label $y$ if
  - $x$ is a variable (i.e., not an individual),
  - $y$ is an ancestor of $x$, and
  - $\mathcal{L}(x) \subseteq \mathcal{L}(y)$

- A node with label $x$ is **blocked**, if it is directly blocked or one of its ancestors is blocked

- Expansion rules may only be applied if $x$ is not blocked
Example: Blocking

\[ \mathcal{K} = \{ B(t), \neg H \sqcup \exists P.H, H(t) \} \]
Example: Blocking

\[ \mathcal{K} = \{ B(t), \neg H \sqcup \exists P.H, H(t) \} \]

\[ t \bullet \quad \mathcal{L}(t) = \{ B, H, \} \]
Example: Blocking

\( \mathcal{K} = \{ B(t), \neg H \sqcup \exists P.H, H(t) \} \)

\[ t \bullet \quad \mathcal{L}(t) = \{ B, H, \neg H \sqcup \exists P.H, \} \]
Example: Blocking

\[ K = \{ B(t), \neg H \sqcup \exists P.H, H(t) \} \]

\[ t \bullet \quad \mathcal{L}(t) = \{ B, H, \neg H \sqcup \exists P.H, \} \]
Example: Blocking

\[ \mathcal{K} = \{ B(t), \neg H \sqcup \exists P.H, H(t) \} \]

\[ \mathcal{L}(t) = \{ B, H, \neg H \sqcup \exists P.H, \exists P.H \} \]

\[ \mathcal{L}(x) = \{ H \} \]
Example: Blocking

\[ \mathcal{K} = \{ B(t), \neg H \sqcup \exists P.H, H(t) \} \]

\[ \mathcal{L}(t) = \{ B, H, \neg H \sqcup \exists P.H, \exists P.H \} \]

\[ \mathcal{L}(x) = \{ H \} \subseteq \mathcal{L}(t) \]
Computational Properties

- The simple tableau algorithm is expensive in general (worst case):
  - the worst case complexity is double exponential
  - testing KB consistency in $\mathcal{ALC}$ is EXPTIME-complete
  - testing concept satisfiability in $\mathcal{ALC}$ is PSPACE-complete
- Still in practice, (optimized) tableaux algorithms work well
  - other notions of blocking might be used, e.g. “cross-path blocking” (blocking node need not be an ancestor)
  - single exponential time tableaux algorithms are available
- For many other description logics, also tableaux algorithms exist (e.g. $\mathcal{SHIQ}$, $\mathcal{SHOIQ}$, ...)
  Some algorithms are quite involved!
References I

Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors.


Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.

Foundations of Semantic Web Technologies.

Sebastian Rudolph.

Foundations of description logics.