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 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:

α 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 $\mathcal{R}$

$owns \sqsubseteq caresFor$

"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."

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

Example: KB Satisfiability

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."

Is $\mathcal{K} = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ satisfiable? Yes!

Example: KB Satisfiability

TBox $\mathcal{T}$

$$Deer \sqsubseteq Mammal$$

"Deers are mammals."

$$Mammal \sqcap Flies \sqsubseteq Bat$$

"Mammals, who fly are bats."

$$Bat \sqsubseteq \forall worksFor.\{batman\}$$

"Bats work only for Batman"

ABox $\mathcal{A}$

$$Deer \sqcap \exists hasNose.Red(rudolph)$$

"Rudolph is a deer with a red nose."

$$\forall worksFor^{-}.(\neg Deer \sqcup Flies)(santa)$$

"Only non-deers or fliers work for Santa."

$$worksFor(rudolph, santa)$$

"Rudolph works for Santa."

$$santa \not\approx batman$$

"Santa is different from Batman."

Is $\mathcal{K}$ satisfiable?

Example: KB Satisfiability

TBox $\mathcal{T}$

$$Deer \sqsubseteq Mammal$$

"Deers are mammals."

$$Mammal \sqcap Flies \sqsubseteq Bat$$

"Mammals, who fly are bats."

$$Bat \sqsubseteq \forall worksFor.\{batman\}$$

"Bats work only for Batman"

ABox $\mathcal{A}$

$Deer \sqcap \exists hasNose.Red(rudolph)$

"Rudolph is a deer with a red nose."

$\forall worksFor^{-}.(\neg Deer \sqcup Flies)(santa)$

"Only non-deers or fliers work for Santa."

$worksFor(rudolph, santa)$

"Rudolph works for Santa."

$santa \not\approx batman$

"Santa is different from Batman."

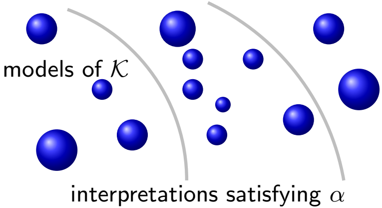


Is $\mathcal{K}$ satisfiable? No!

Entailment of Axioms

Entailment checking

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



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

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."

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."

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

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."

- $\mathcal{K} \models \exists caresFor.(Cat \sqcap Alive)(schroedinger)$
- $\mathcal{K} \models \forall owns.\neg Cat \sqsubseteq \neg HappyCatOwner$?

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."

- $\mathcal{K} \models \exists caresFor.(Cat \sqcap Alive)(schroedinger)$
- $\mathcal{K} \models \forall owns.\neg Cat \sqsubseteq \neg HappyCatOwner$
- $\mathcal{K} \models Cat \sqsubseteq Healthy$?

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."

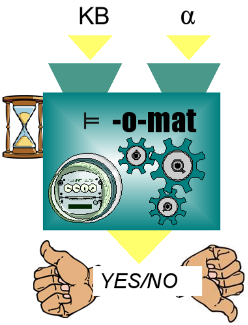
- $\mathcal{K} \models \exists caresFor.(Cat \sqcap Alive)(schroedinger)$
- $\mathcal{K} \models \forall owns.\neg Cat \sqsubseteq \neg HappyCatOwner$
- $\mathcal{K} \not\models Cat \sqsubseteq Healthy$

Decidability of DLs

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

Given: a KB and an axiom α,

Output: "yes" iff KB $\models \alpha$ and no otherwise.



- Likewise, there is a similar algorithm that decides whether an input KB is satisfiable
- Just ask 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., $\mathcal{K} \models CatOwner(Schroedinger)$
- **Concept Satisfiability:** verify whether a given concept is (un)satisfiable, e.g., $\mathcal{K} \models Dead \sqcap Alive \sqsubseteq \bot$
- **Coherence:** verify whether none of the concepts in the KB is unsatisfiable
- **Classification:** compute the subsumption hierarchy of all atomic concepts, e.g. $\mathcal{K} \models Healthy \sqsubseteq \neg Dead$, etc.
- **Instance Retrieval:** retrieve all the individuals known to be instances of a certain concept, e.g., find all a, s.t. $\exists caresFor(a)$

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. $\mathcal{K} \models A \sqsubseteq B$ 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 α
- 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 ($\{\neg\alpha\} \rightsquigarrow A_\alpha$).

Entailment Checking, cont'd

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

α	$\mathcal{A}_\alpha$
$r_1 \circ \ldots \circ r_n \sqsubseteq r$	$\{\neg r(c_0, c_n), r_1(c_0, c_1), \ldots, r_n(c_{n-1}, c_n)\}$
$\mathsf{Dis}(r, r')$	$\{r(c_1, c_2), r'(c_1, c_2)\}$
$C \sqsubseteq D$	$\{(C \sqcap \neg D)(c)\}$ or $\{\top \sqsubseteq \exists u(C \sqcap \neg D)\}$
$C(a)$	$\{\neg C(a)\}$
$\neg C(a)$	$\{C(a)\}$
$r(a, b)$	$\{\neg r(a, b)\}$
$\neg r(a, b)$	$\{r(a, b)\}$
$a \approx b$	$\{a \not\approx b\}$
$a \not\approx b$	$\{a \approx b\}$

- Individual names c with possible subscripts are supposed to be fresh[1].
- For GCIs (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 $\mathcal{K}$.

Concept Satisfiability

Concept Satisfiability

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. $\mathcal{K} \Longleftrightarrow \mathcal{K} \cup \{C(a)\}$ is consistent, where a is a fresh individual name
- C is satisfiable wrt. $\mathcal{K} \Longleftrightarrow \mathcal{K} \not\models C \sqsubseteq \bot$

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.

- $$\begin{aligned} C \sqsubseteq D &\iff C \sqcap \neg D \text{ is unsatisfiable} \\ &\iff \text{KB } \{C(a),\ \neg D(a)\} \text{ is unsatisfiable} \end{aligned}$$

- $$\begin{aligned} C \equiv D &\iff (C \sqcap \neg D) \sqcup (D \sqcap \neg C) \text{ is unsatisfiable} \\ &\iff \text{both } C \sqcap \neg D \text{ and } D \sqcap \neg C \text{ are unsatisfiable} \\ &\iff \text{both KBs } \{C(a),\ \neg D(a)\} \text{ and} \\ &\qquad \{D(a),\ \neg C(a)\} \text{ are unsatisfiable} \end{aligned}$$

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

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

- retrieve(C,$\mathcal{K}$) $= \{a \in N_I \mid a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$ $\}$
- retrieve(r,$\mathcal{K}$) $= \{(a, b) \in {N_I}^2 \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for every model $\mathcal{I}$ 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 α ('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 α ('knowledge explosion') if $\mathcal{K}$ is inconsistent by introducing, e.g., new semantics

- **Entailment Explanation**

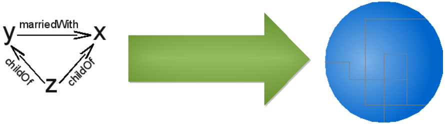
  identify axioms in the knowledge base that support a conclusion $\mathcal{K} \models \alpha$, typically a smallest $\mathcal{K}' \subseteq \mathcal{K}$ such that $\mathcal{K}' \models \alpha$
  (*'axiom pinpointing'*)

Conjunctive Query Answering

Generalize Instance Retrieval by allowing joins and projections:

$$q(Z) = \exists Y \exists X.childOf(Z, Y) \wedge childOf(Z, X) \wedge marriedWith(Y, X)$$

- 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).

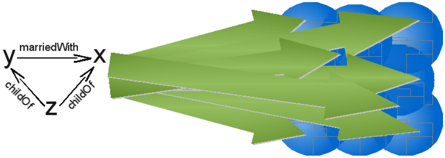


  - this is rather easy

Conjunctive Query Answering, cont'd

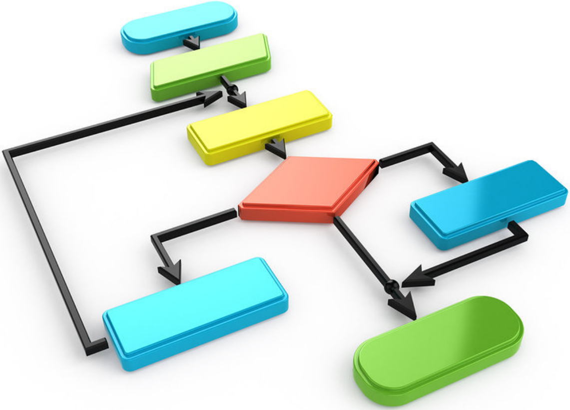
Generalize Instance Retrieval by allowing joins and projections:
$q(Z) = \exists Y \exists X. childOf(Z,Y) \land childOf(Z,X) \land marriedWith(Y,X)$

- 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.

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.

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

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."

Is $\mathcal{K}$ satisfiable?

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$owns \sqsubseteq caresFor$$

TBox $\mathcal{T}$

$$\begin{aligned} Healthy &\sqsubseteq \neg Dead \\ Cat &\sqsubseteq Dead \sqcup Alive \\ HappyCatOwner &\sqsubseteq \exists owns.Cat \sqcap \forall caresFor.Healthy \end{aligned}$$

ABox $\mathcal{A}$

$$HappyCatOwner(s)$$

Is $\mathcal{K}$ satisfiable?

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$\mathit{owns} \sqsubseteq \mathit{caresFor}$$

TBox $\mathcal{T}$

$$\begin{aligned} \mathit{Healthy} &\sqsubseteq \neg \mathit{Dead} \\ \mathit{Cat} &\sqsubseteq \mathit{Dead} \sqcup \mathit{Alive} \\ \mathit{HappyCatOwner} &\sqsubseteq \exists \mathit{owns}.\mathit{Cat} \sqcap \forall \mathit{caresFor}.\mathit{Healthy} \end{aligned}$$

ABox $\mathcal{A}$

$$\mathit{HappyCatOwner}(s)$$

Is $\mathcal{K}$ satisfiable?

s ●

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

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$owns \sqsubseteq caresFor$$

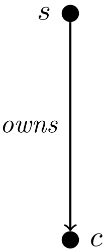
TBox $\mathcal{T}$

$$\begin{aligned} Healthy &\sqsubseteq \neg Dead \\ Cat &\sqsubseteq Dead \sqcup Alive \\ HappyCatOwner &\sqsubseteq \exists owns.Cat \sqcap \forall caresFor.Healthy \end{aligned}$$

ABox $\mathcal{A}$

$$HappyCatOwner(s)$$

Is $\mathcal{K}$ satisfiable?



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

$\mathcal{L}(c) = \{Cat\}$

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$owns \sqsubseteq caresFor$$

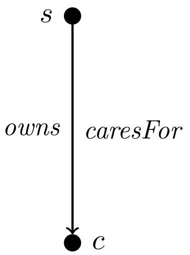
TBox $\mathcal{T}$

$$\begin{aligned} Healthy &\sqsubseteq \neg Dead \\ Cat &\sqsubseteq Dead \sqcup Alive \\ HappyCatOwner &\sqsubseteq \exists owns.Cat \sqcap \forall caresFor.Healthy \end{aligned}$$

ABox $\mathcal{A}$

$$HappyCatOwner(s)$$

Is $\mathcal{K}$ satisfiable?



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

$\mathcal{L}(c) = \{Cat\}$

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$owns \sqsubseteq caresFor$$

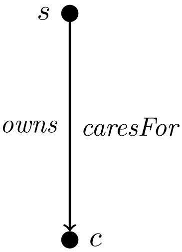
TBox $\mathcal{T}$

$$\begin{aligned} Healthy &\sqsubseteq \neg Dead \\ Cat &\sqsubseteq Dead \sqcup Alive \\ HappyCatOwner &\sqsubseteq \exists owns.Cat \sqcap \forall caresFor.Healthy \end{aligned}$$

ABox $\mathcal{A}$

$$HappyCatOwner(s)$$

Is $\mathcal{K}$ satisfiable?



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

$$\mathcal{L}(c) = \{Cat, Healthy\}$$

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$owns \sqsubseteq caresFor$$

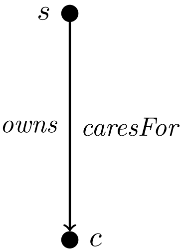
TBox $\mathcal{T}$

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

ABox $\mathcal{A}$

$$HappyCatOwner(s)$$

Is $\mathcal{K}$ satisfiable?



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

$$\mathcal{L}(c) = \{Cat, Healthy, \neg Dead\}$$

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$owns \sqsubseteq caresFor$$

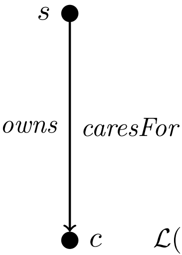
TBox $\mathcal{T}$

$$\begin{aligned} Healthy &\sqsubseteq \neg Dead \\ Cat &\sqsubseteq Dead \sqcup Alive \\ HappyCatOwner &\sqsubseteq \exists owns.Cat \sqcap \forall caresFor.Healthy \end{aligned}$$

ABox $\mathcal{A}$

$$HappyCatOwner(s)$$

Is $\mathcal{K}$ satisfiable?



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

$$\mathcal{L}(c) = \{Cat, Healthy, \neg Dead, Alive\}$$

Tableau Overview Example: Happy Cat Owner

RBox $\mathcal{R}$

$$\mathit{owns} \sqsubseteq \mathit{caresFor}$$

TBox $\mathcal{T}$

$$\begin{aligned} \mathit{Healthy} &\sqsubseteq \neg \mathit{Dead} \\ \mathit{Cat} &\sqsubseteq \mathit{Dead} \sqcup \mathit{Alive} \\ \mathit{HappyCatOwner} &\sqsubseteq \exists \mathit{owns}.\mathit{Cat} \sqcap \forall \mathit{caresFor}.\mathit{Healthy} \end{aligned}$$

ABox $\mathcal{A}$

$$\mathit{HappyCatOwner}(s)$$

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

$$\begin{tikzcd} s \arrow[d, "owns"', "caresFor"] \\ c \end{tikzcd}$$

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

$\mathcal{L}(c) = \{\mathit{Cat}, \mathit{Healthy}, \neg\mathit{Dead}, \mathit{Alive}\}$

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

Construct	Syntax	Example	Semantics
atomic concept	A	$Doctor$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	r	$hasChild$	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$Human \sqcap Male$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
unqual. exist. res.[2]	$\exists r$	$\exists hasChild$	$\{o \mid \exists o'.(o,o') \in r^{\mathcal{I}}\}$
value res.	$\forall r.C$	$\forall hasChild.Male$	$\{o \mid \forall o'.(o,o') \in r^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
full negation	$\neg C$	$\neg\forall hasChild.Male$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$

[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 $nnf(\mathcal{K})$ 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

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 $nnf(\mathcal{K})$ we need to

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

$nnf(C) \rightsquigarrow C$ if $C \in \{A, \neg A, \{a_1 \dots a_n\}, \neg\{a_1 \dots a_n\}, \exists r.\mathsf{Self}, \neg\exists r.\mathsf{Self}, \top, \bot\}$

$$
\begin{aligned}
nnf(\neg\neg C) &\rightsquigarrow nnf(C) \\
nnf(C \sqcap D) &\rightsquigarrow nnf(C) \sqcap nnf(D) \\
nnf(C \sqcup D) &\rightsquigarrow nnf(C) \sqcup nnf(D) \\
nnf(\neg(C \sqcup D)) &\rightsquigarrow nnf(\neg C \sqcap \neg D) \\
nnf(\neg(C \sqcap D)) &\rightsquigarrow nnf(\neg C \sqcup \neg D) \\
nnf(\forall r.C) &\rightsquigarrow \forall r.nnf(C) \\
nnf(\exists r.C) &\rightsquigarrow \exists r.nnf(C)
\end{aligned}
$$

$$
\begin{aligned}
nnf(\neg\forall r.C) &\rightsquigarrow \exists r.nnf(\neg C) \\
nnf(\neg\exists r.C) &\rightsquigarrow \forall r.nnf(\neg C) \\
nnf(\leq k\, r.C) &\rightsquigarrow \leq k\, r.nnf(C) \\
nnf(\geq k\, r.C) &\rightsquigarrow \geq k\, r.nnf(C) \\
nnf(\neg \leq k\, r.C) &\rightsquigarrow \geq (k+1)\, r.nnf(C) \\
nnf(\neg \geq k\, r.C) &\rightsquigarrow \leq (k-1)\, r.nnf(C) \\
nnf(\neg\top) &\rightsquigarrow \bot
\end{aligned}
$$

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 $nnf(\mathcal{K})$ we need to

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

$nnf(C) \rightsquigarrow C$ if $C \in \{A, \neg A, \{a_1 \dots a_n\}, \neg\{a_1 \dots a_n\}, \exists r.\mathsf{Self}, \neg\exists r.\mathsf{Self}, \top, \bot\}$

$$
\begin{aligned}
nnf(\neg\neg C) &\rightsquigarrow nnf(C) \\
nnf(C \sqcap D) &\rightsquigarrow nnf(C) \sqcap nnf(D) \\
nnf(C \sqcup D) &\rightsquigarrow nnf(C) \sqcup nnf(D) \\
nnf(\neg(C \sqcup D)) &\rightsquigarrow nnf(\neg C \sqcap \neg D) \\
nnf(\neg(C \sqcap D)) &\rightsquigarrow nnf(\neg C \sqcup \neg D) \\
nnf(\forall r.C) &\rightsquigarrow \forall r.nnf(C) \\
nnf(\exists r.C) &\rightsquigarrow \exists r.nnf(C)
\end{aligned}
$$

$$
\begin{aligned}
nnf(\neg\forall r.C) &\rightsquigarrow \exists r.nnf(\neg C) \\
nnf(\neg\exists r.C) &\rightsquigarrow \forall r.nnf(\neg C) \\
nnf(\leq k\, r.C) &\rightsquigarrow \leq k\, r.nnf(C) \\
nnf(\geq k\, r.C) &\rightsquigarrow \geq k\, r.nnf(C) \\
nnf(\neg \leq k\, r.C) &\rightsquigarrow \geq (k+1)\, r.nnf(C) \\
nnf(\neg \geq k\, r.C) &\rightsquigarrow \leq (k-1)\, r.nnf(C) \\
nnf(\neg\top) &\rightsquigarrow \bot
\end{aligned}
$$

C and $nnf(C)$ are logically equivalent, i.e., $C^{\mathcal{I}} = nnf(C)^{\mathcal{I}}$, for every interpretation $\mathcal{I}$, and the translation process terminates in linear time.

Example: Negation Normal Form

Negation Normal Form

$FilmActor \sqsubseteq (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$

Example: Negation Normal Form

Negation Normal Form

$FilmActor \sqsubseteq (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$

1. $\neg FilmActor \sqcup (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$

Example: Negation Normal Form

Negation Normal Form

$FilmActor \sqsubseteq (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$

1. $\neg FilmActor \sqcup (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$
2. $\neg FilmActor \sqcup (\exists actedIn \sqcap Artist) \sqcup (\exists actedIn \sqcap \neg\exists playsIn.Theater)$

Example: Negation Normal Form

Negation Normal Form

$FilmActor \sqsubseteq (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$

1. $\neg FilmActor \sqcup (\exists actedIn \sqcap Artist) \sqcup \neg(\neg\exists actedIn \sqcup \exists playsIn.Theater)$
2. $\neg FilmActor \sqcup (\exists actedIn \sqcap Artist) \sqcup (\exists actedIn \sqcap \neg\exists playsIn.Theater)$
3. $\neg FilmActor \sqcup (\exists actedIn \sqcap Artist) \sqcup (\exists actedIn \sqcap \forall playsIn.\neg Theater)$

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

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

Example: Initial Tableau

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

$\mathcal{L}(a) = \{A,$

Example: Initial Tableau

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

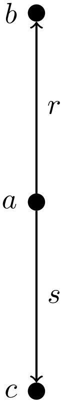
b ●

a ● $\quad \mathcal{L}(a) = \{A, \exists r.B\}$

c ●

Example: Initial Tableau

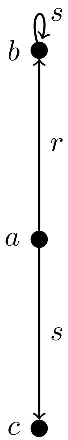
$$\mathcal{K} = \{A(a), \quad (\exists r.B)(a), \quad r(a,b), \quad r(a,c), \quad s(b,b), \quad (A \sqcup B)(c), \quad \neg A \sqcup (\forall s.B)\}$$



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

Example: Initial Tableau

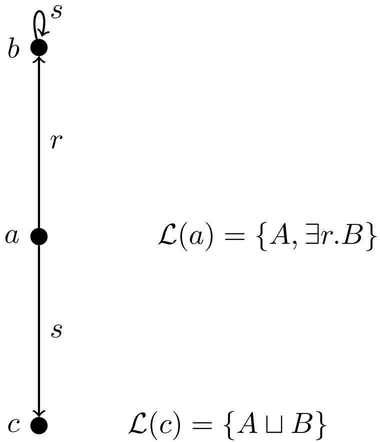
$$\mathcal{K} = \{A(a), \quad (\exists r.B)(a), \quad r(a,b), \quad r(a,c), \quad s(b,b), \quad (A \sqcup B)(c), \quad \neg A \sqcup (\forall s.B)\}$$



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

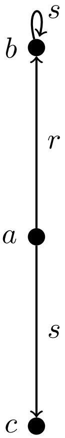
Example: Initial Tableau

$$\mathcal{K} = \{A(a), \quad (\exists r.B)(a), \quad r(a,b), \quad r(a,c), \quad s(b,b), \quad {\color{red}(A \sqcup B)(c)}, \quad \neg A \sqcup (\forall s.B)\}$$



Example: Initial Tableau

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



$\mathcal{L}(b) = \emptyset$

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

$\mathcal{L}(c) = \{A \sqcup B\}$

Expansion Rules for the Naive Tableau

- $\sqcap$-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

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

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

Expansion Rules for the Naive Tableau

- $\sqcap$-rule: If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.
- $\sqcup$-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 $\mathcal{L}(x) \leftarrow D$.
- $\exists$-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 $\mathcal{L}(y) = \{C\}$.
- $\forall$-rule: If $\forall r.C \in \mathcal{L}(x)$ and there is a node y with $r \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

Expansion Rules for the Naive Tableau

- $\sqcap$-rule: If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.
- $\sqcup$-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 $\mathcal{L}(x) \leftarrow D$.
- $\exists$-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 $\mathcal{L}(y) = \{C\}$.
- $\forall$-rule: If $\forall r.C \in \mathcal{L}(x)$ and there is a node y with $r \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.
- TBox-rule: If C is a (rewritten and normalized) TBox statement and $C \notin \mathcal{L}(x)$, 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 $\sqcup$-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 ●

$\mathcal{L}(a) = \{C,$

Example: Expansion Rules

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

a ● $\mathcal{L}(a) = \{C, \forall r.\neg E,$

Example: Expansion Rules

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

a ●

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

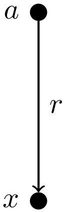
Example: Expansion Rules

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

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

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

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

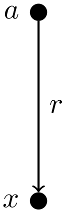
$$\begin{tikzcd} a \arrow[d, "r"] \\ x \end{tikzcd}$$

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

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

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

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

$$\begin{tikzcd} a \arrow[d, "r"] \\ x \end{tikzcd}$$

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

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

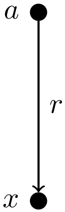
Example: Expansion Rules

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

$$\begin{tikzcd}
a \arrow[d, "r"] & \mathcal{L}(a) = \{C, \forall r.\neg E, \neg C \sqcup \exists r.D,\ \exists r.D\} \\
x & \mathcal{L}(x) = \{D, \neg D \sqcup E,\ E,\ \neg E\}
\end{tikzcd}$$

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: $\mathcal{K} = \{\neg Person \sqcup \exists hasParent,\ Person(a_1)\}$.
- 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 ● $\mathcal{L}(t) = \{B, H,$

Example: Blocking

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

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

Example: Blocking

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

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

Example: Blocking

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

$$\begin{tikzcd} t \arrow[d, "P"] \\ x \end{tikzcd}$$

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

$\mathcal{L}(x) = \{H\}$

Example: Blocking

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

$$\begin{tikzcd} t \arrow[d, "P"] \\ x \end{tikzcd}$$

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

$\mathcal{L}(x) = \{H\} \subseteq \mathcal{L}(t)$

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!

Summary

1. Satisfaction and Entailment
    - Notions
    - Decidability

2. Reasoning Problems
    - Knowledge base consistency
    - Entailment checking
    - Concept satisfiability
    - Classification
    - Instance retrieval

3. Algorithmic Approaches to DL Reasoning
    - Types of reasoning procedures
    - Tableaux

4. Novel reasoning problems
    - Conjunctive query answering

References I

Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2007.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.
Foundations of Semantic Web Technologies.
Chapman and Hall, 2010.

Sebastian Rudolph.
Foundations of description logics.
In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data*, volume 6848 of *Lecture Notes in Computer Science*, pages 76–136. Springer Berlin / Heidelberg, 2011.