# Knowledge Reasoning

Pınar Yolum
Email: p.yolum@uu.nl

Department of Information and Computing Sciences
Utrecht University

# Reasoning techniques

- Model construction
  - Prove that KB does not entail a given axiom because a model cannot be constructed
  - Tableau algorithms
- Proof derivation
  - Prove that KB entails a given axiom because a proof can be found
  - Rule-based algorithms, completion algorithms

# A tableau algorithm for DLs

- To decide the consistency of an ontology, where the ontology contains TBox axioms (schema) and ABox axioms (data) (O = T, A)
    - T = $\{Parent \equiv Person \sqcap \exists hasChild.Person\}$
    - A = $\{john : Parent; mary : Person\}$
- By building a model for the ontology
- If the algorithm succeeds, then there is a description that satisfies the ontology

# How it works

- Start with the ABox (i.e., facts already known)
- Convert Tbox into Negation Normal Form
- Keep applying expansion rules to derive more precise facts
- See if contradictions (e.g., clashes happen $\{$a: B, a:$\neg$B$\}$)

# Negation Normal Form (NNF)

Transform all concepts in $O$ into NNF by pushing the negation inward

- $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$
- $\neg(\forall R.C) \equiv \exists R.\neg C$
- $\neg(\exists R.C) \equiv \forall R.\neg C$

# Transformation rules (1)

- $\sqcap$ rule: if $a : C_1 \sqcap C_2 \in \mathcal{A}$ and $\{a : C_1, a : C_2\} \not\subseteq \mathcal{A}$ then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a : C_1, a : C_2\}$
  - T = $\{Parent \equiv Person \sqcap \exists hasChild.Person\}$
  - A = $\{john : Parent, mary : Person\}$
  - A' = A $\cup$ $\{john : Person, john : \exists hasChild.Person\}$
- $\sqcup$ rule: if $a : C_1 \sqcup C_2 \in \mathcal{A}$ and $\{a : C_1, a : C_2\} \cap \mathcal{A} = \emptyset$ then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a : C_1\}$ and $\mathcal{A} \cup \{a : C_2\}$
  - T = $\{Person \equiv FemalePerson \sqcup MalePerson\}$
  - A = $\{john : Person\}$
  - A' = A $\cup$ $\{john : FemalePerson\}$
  - A" = A $\cup$ $\{john : MalePerson\}$

# Transformation rules (2)

- $\exists$ rule: if $a : \exists s.C \in \mathcal{A}$ and there is no $b$ with $\{(a,b) : s, b : C\} \subseteq \mathcal{A}$
  then create a new individual name $d$ and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(a,d) : s, d : C\}$
  - T = $\{Father \equiv \exists hasChild.MalePerson\}$
  - A = $\{john : Father\}$
  - A' = A $\cup$ $\{d : MalePerson, (john, d) : hasChild\}$
- $\forall$ rule: if $\{a : \forall s.C, (a,b) : s\} \subseteq \mathcal{A}$ and $b : C \notin \mathcal{A}$
  then replace $\mathcal{A}$ with $\mathcal{A} \cup \{b : C\}$
  - T = $\{Father \equiv \forall hasChild.MalePerson\}$
  - A = $\{john : Father, (john, ali) : hasChild\}$
  - A' = A $\cup$ $\{ali : MalePerson\}$
- GCI rule: if $C \sqsubseteq D \in T$ and $a : (\neg C \sqcup D) \notin \mathcal{A}$ for $a$ in $\mathcal{A}$
  then replace $\mathcal{A}$ with $\mathcal{A} \cup \{a : (\neg C \sqcup D)\}$

# Use of the algorithm

- Derive all possible branches and check if there exists a branch that is complete (nothing to add) and does not contain a clash $\rightarrow$ Consistent
- Given a query, add the negation in and check if for all the branches there is a clash $\rightarrow$ KB satisfies the query

# Consistency example

T = $\{$*Father* $\equiv$ ($\forall$ hasChild.MalePerson) $\sqcap$($\exists$ hasChild.$\neg$ MalePerson)$\}$
A = $\{$*john* : *Father*$\}$

# Consistency example

T = $\{$*Father* $\equiv$ ($\forall$ hasChild.MalePerson) $\sqcap$($\exists$ hasChild.$\neg$ MalePerson)$\}$
A = $\{$*john* : *Father*$\}$

- john:Father
- john:$\forall$ hasChild.MalePerson ($\sqcap$ rule)
- john:$\exists$ hasChild.$\neg$MalePerson ($\sqcap$ rule)

# Consistency example

T = $\{Father \equiv (\forall$ hasChild.MalePerson) $\sqcap(\exists$ hasChild.$\neg$ MalePerson)$\}$
A = $\{john : Father\}$

- john:Father
- john:$\forall$ hasChild.MalePerson ($\sqcap$ rule)
- john:$\exists$ hasChild.$\neg$MalePerson ($\sqcap$ rule)
- (john,d):hasChild ($\exists$ rule)
- d:$\neg$ MalePerson ($\exists$ rule)

# Consistency example

T = $\{Father \equiv (\forall$ hasChild.MalePerson) $\sqcap(\exists$ hasChild.$\neg$ MalePerson)$\}$
A = $\{john : Father\}$

- john:Father
- john:$\forall$ hasChild.MalePerson ($\sqcap$ rule)
- john:$\exists$ hasChild.$\neg$MalePerson ($\sqcap$ rule)
- (john,d):hasChild ($\exists$ rule)
- d:$\neg$ MalePerson ($\exists$ rule)
- d:MalePerson ($\forall$ rule)

# Consistency example

T = $\{Father \equiv (\forall$ hasChild.MalePerson) $\sqcap(\exists$ hasChild.$\neg$ MalePerson)$\}$

A = $\{john : Father\}$

- john:Father
- john:$\forall$ hasChild.MalePerson ($\sqcap$ rule)
- john:$\exists$ hasChild.$\neg$MalePerson ($\sqcap$ rule)
- (john,d):hasChild ($\exists$ rule)
- d:$\neg$ MalePerson ($\exists$ rule)
- d:MalePerson ($\forall$ rule)

CLASH! (The tableau is complete and there is a contradiction, so ontology not consistent)

# Query example (1)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.Doctor\}$

A = {john: HappyParent, (john, mary):hasChild}

Q = {mary: Doctor}?

# Query example (1)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.Doctor\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild
- mary: $\neg$ Doctor

# Query example (1)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.Doctor\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild
- mary: $\neg$ Doctor
- john: Parent ($\sqcap$ rule)
- john: $\forall$ hasChild.Doctor ($\sqcap$ rule)

# Query example (1)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.Doctor\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild
- mary: $\neg$ Doctor
- john: Parent ($\sqcap$ rule)
- john: $\forall$ hasChild.Doctor ($\sqcap$ rule)
- john: Person ($\sqcap$ rule)
- john: $\exists$ hasChild.Person ($\sqcap$ rule)

# Query example (1)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.Doctor\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild
- mary: $\neg$ Doctor
- john: Parent ($\sqcap$ rule)
- john: $\forall$ hasChild.Doctor ($\sqcap$ rule)
- john: Person ($\sqcap$ rule)
- john: $\exists$ hasChild.Person ($\sqcap$ rule)
- (john,d):hasChild, d: Person ($\exists$ rule)
- d: Doctor ($\forall$ rule)
- mary: Doctor ($\forall$ rule)

CLASH! (KB entails that Mary is a doctor)

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild ,mary: $\neg$Doctor

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild ,mary: $\neg$Doctor
- john: Parent, john: $\forall$hasChild.(Doctor $\sqcup\exists$hasChild.Doctor)
- john: Person, john: $\exists$hasChild.Person

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild ,mary: $\neg$Doctor
- john: Parent, john: $\forall$hasChild.(Doctor $\sqcup\exists$hasChild.Doctor)
- john: Person, john: $\exists$hasChild.Person
- mary:Doctor $\sqcup\exists$hasChild.Doctor (Two alternative branches)

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild ,mary: $\neg$Doctor
- john: Parent, john: $\forall$hasChild.(Doctor $\sqcup\exists$hasChild.Doctor)
- john: Person, john: $\exists$hasChild.Person
- mary:Doctor $\sqcup\exists$hasChild.Doctor (Two alternative branches)
- mary:Doctor (Branch 1. CLASH!)

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild ,mary: ¬Doctor
- john: Parent, john: ∀hasChild.(Doctor ⊔∃hasChild.Doctor)
- john: Person, john: ∃hasChild.Person
- mary:Doctor ⊔∃hasChild.Doctor (Two alternative branches)
- mary: ∃hasChild.Doctor
- (Mary,b):hasChild, b: Doctor, b:Person (Branch 2. Complete but NO CLASH!)

# Query example (2)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild}
Q = {mary: Doctor}?

- john: HappyParent, (john, mary):hasChild ,mary: $\neg$Doctor
- john: Parent, john: $\forall$hasChild.(Doctor $\sqcup \exists$hasChild.Doctor)
- john: Person, john: $\exists$hasChild.Person
- mary:Doctor $\sqcup \exists$hasChild.Doctor (Two alternative branches)

KB does not entail that Mary is a doctor

# Query example (3)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild, mary:$\forall$hasChild.$\perp$}
Q = {mary: Doctor}?

# Query example (3)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$

A = {john: HappyParent, (john, mary):hasChild, mary:$\forall$hasChild.$\perp$}

Q = {mary: Doctor}?

What will be different?

# Query example (3)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild, mary:$\forall$hasChild.$\perp$}
Q = {mary: Doctor}?
What will be different?

- (mary,b):hasChild, b: Doctor, b:Person
- b: $\perp$ (Clash!)

# Query example (3)

T = $\{Doctor \sqsubseteq Person, Parent \equiv Person \sqcap \exists hasChild.Person, HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)\}$
A = {john: HappyParent, (john, mary):hasChild, mary:$\forall$hasChild.$\perp$}
Q = {mary: Doctor}?
What will be different?

- (mary,b):hasChild, b: Doctor, b:Person
- b: $\perp$ (Clash!)

KB does entail that Mary is a doctor

# Optimization

- Termination:
  - Naive implementation of the tableau methods would not terminate as it keeps introducting new instances
  - Introduce the idea of *Blocked* so that an older instance is reused
- Dependency directed backtracking
- Reuse of previous computations

# Correctness

- Sound: If clash-free ABox is constructed, then KB is satisfiable
- Complete: If KB is satisfiable, then clash-free ABox is created
- Terminating: The algorithm will always give an answer

# General reasoning assumptions

- Closed world assumption (a la Prolog)
  - Negation as failure: If you cannot prove something, assume not true
  - A shortcut to get a result
  - Assumes the world has been described in sufficient detail
- Unique name assumption
  - An object can have a single name
  - Ex: John and Mary cannot be the same person
- Neither hold for DL reasoning