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)
- By building a model for the ontology
- If the algorithm succeeds, then there is a description that satisfies the ontology
- Works on the ABoxes

How it works

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

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

- $\sqcap$ 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 $\mathcal{A} \cup \{a : C_1, a : C_2\}$
- $\sqcup$ 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 $\mathcal{A} \cup \{a : C_2\}$
- $\exists$ 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 $\mathcal{A} \cup \{(a, d) : s, d : C\}$
- $\forall$ rule: if $\{a : \forall s.C, (a, b) : s\} \subseteq \mathcal{A}$ and $b : C \notin \mathcal{A}$
  then replace $\mathcal{A}$ with $\mathcal{A} \cup \{b : C\}$
- 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 = $\{(\forall$ hasChild.MalePerson) $\sqcap(\exists$ hasChild.$\neg$ MalePerson)$\}$

Consistency example

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

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

Consistency example

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

- a: $\forall$ hasChild.MalePerson ($\sqcap$ rule)
- a: $\exists$ hasChild.$\neg$MalePerson ($\sqcap$ rule)
- child(a, b) ($\exists$ rule)
- b: $\neg$ male ($\exists$ rule)

Consistency example

T = {(∀ hasChild.MalePerson) ⊓(∃ hasChild.¬ MalePerson)}

- a: ∀ hasChild.MalePerson (⊓ rule)
- a: ∃ hasChild.¬MalePerson (⊓ rule)
- child(a, b) (∃ rule)
- b: ¬ male (∃ rule)
- b:male (∀ rule)

Consistency example

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

- a: $\forall$ hasChild.MalePerson ($\sqcap$ rule)
- a: $\exists$ hasChild.$\neg$MalePerson ($\sqcap$ rule)
- child(a, b) ($\exists$ rule)
- b: $\neg$ male ($\exists$ rule)
- b:male ($\forall$ 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, hasChild(John, Mary)} 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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent
- hasChild(John, Mary)
- 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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent
- hasChild(John, Mary)
- Mary: $\neg$ Doctor
- John: Parent
- John: $\forall$ hasChild.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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent
- hasChild(John, Mary)
- Mary: $\neg$ Doctor
- John: Parent
- John: $\forall$ hasChild.Doctor
- John: Person
- John: $\exists$ hasChild.Person

Query example (1)

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

- John: HappyParent
- hasChild(John, Mary)
- Mary: $\neg$ Doctor
- John: Parent
- John: $\forall$ hasChild.Doctor
- John: Person
- John: $\exists$ hasChild.Person
- Mary: Doctor

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, hasChild(John, Mary)$\}$ 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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent, hasChild(John, Mary)
- 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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent, hasChild(John, Mary)
- Mary: $\neg$Doctor
- John: Parent, John: $\forall$hasChild.(Doctor $\sqcup \exists$hasChild.Person)
- John: Person, John: $\exists$hasChild.Person
- Mary:Doctor $\sqcup \exists$hasChild.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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent, hasChild(John, Mary)
- Mary: $\neg$Doctor
- John: Parent, John: $\forall$hasChild.(Doctor $\sqcup\exists$hasChild.Person)
- John: Person, John: $\exists$hasChild.Person
- Mary:Doctor $\sqcup\exists$hasChild.Doctor
- hasChild(John, a), a: Person
- a: Doctor $\sqcup\exists$hasChild.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, hasChild(John, Mary)$\}$ Q = $\{$Mary: Doctor$\}$?

- John: HappyParent, hasChild(John, Mary)
- Mary: $\neg$Doctor
- John: Parent, John: $\forall$hasChild.(Doctor $\sqcup \exists$hasChild.Person)
- John: Person, John: $\exists$hasChild.Person
- Mary:Doctor $\sqcup \exists$hasChild.Doctor
- hasChild(John, a), a: Person
- a: Doctor $\sqcup \exists$hasChild.Doctor
- Mary: Doctor (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, hasChild(John, Mary)$\}$ Q = $\{$Mary: Doctor$\}$?

- John: HappyParent, hasChild(John, Mary)
- Mary: $\neg$Doctor
- John: Parent, John: $\forall$hasChild.(Doctor $\sqcup\exists$hasChild.Person)
- John: Person, John: $\exists$hasChild.Person
- Mary:Doctor $\sqcup\exists$hasChild.Doctor
- hasChild(John, a), a: Person
- a: Doctor $\sqcup\exists$hasChild.Doctor
- Mary: $\exists$hasChild.Doctor
- hasChild(Mary,b), b: Doctor, b:Person
- a: Doctor (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, hasChild(John, Mary)} Q = {Mary: Doctor}?

- John: HappyParent, hasChild(John, Mary)
- Mary: $\neg$Doctor
- John: Parent, John: $\forall$hasChild.(Doctor $\sqcup\exists$hasChild.Person)
- John: Person, John: $\exists$hasChild.Person
- Mary:Doctor $\sqcup\exists$hasChild.Doctor
- hasChild(John, a), a: Person
- a: Doctor $\sqcup\exists$hasChild.Doctor

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, hasChild(John, Mary), Mary:$\forall$hasChild.$\bot$} 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, hasChild(John, Mary), Mary:$\forall$hasChild.$\bot$} 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, hasChild(John, Mary), Mary:$\forall$hasChild.$\bot$} Q = {Mary: Doctor}?
What will be different?

- hasChild(Mary,b), b: Doctor, b:Person
- b: $\bot$ (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, hasChild(John, Mary), Mary:$\forall$hasChild.$\perp$} Q = {Mary: Doctor}?

What will be different?

- hasChild(Mary,b), 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