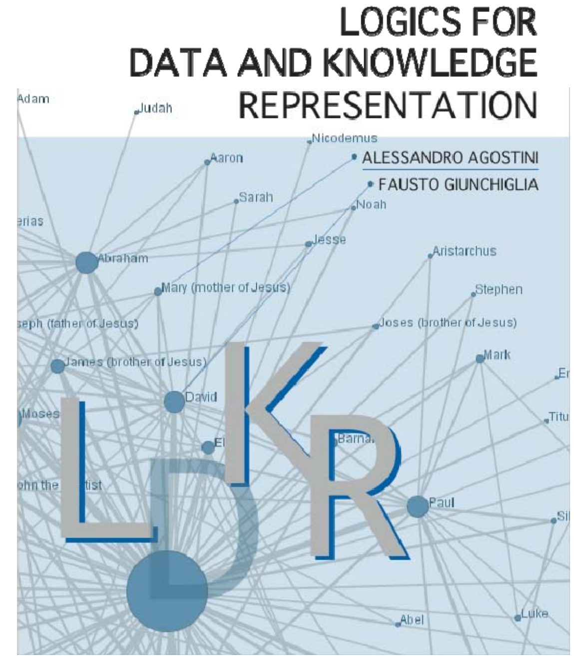


Logics for Data and Knowledge Representation

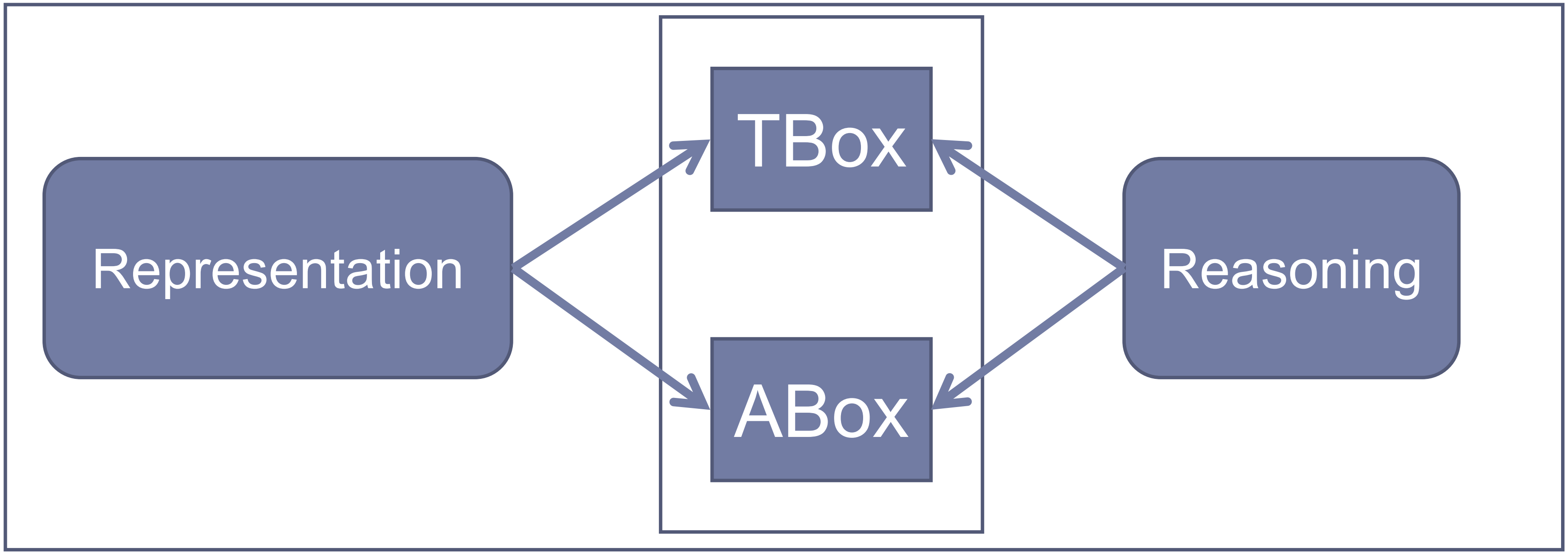
Description Logics

Outline

- Overview
- Syntax: the DL family of languages
- Semantics
- TBox
- ABox
- Tableau Algorithm

Overview

- Description Logics (DLs) is a family of KR formalisms



- Alphabet of symbols with two new symbols w.r.t. ClassL:
  - ∀R (value restriction)
  - ∃R (existential quantification)

R are atomic role names

AL (Attributive language) Logical Symbols

- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ... | ⊥ | ⊤

  <wff> ::= <Atomic> | ¬<Atomic> | <wff> ⊓ <wff> | ∀R.C | ∃R.⊤

  NOTE: no ⊔, ∃R.⊤ = limited existential quantifier, ¬ on atomic only

- Person ⊓ Female

  "persons that are female"
- Person ⊓ ∃hasChild. ⊤

  "(all those) persons that have a child"
- Person ⊓ ∀hasChild. ⊥

  "(all those) persons without a child"
- Person ⊓ ∀hasChild.Female

  "persons all of whose children are female"

ALU (AL with disjunction)

- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ... | ⊥ | ⊤

  <wff> ::= <Atomic> | ¬<Atomic> | <wff> ⊓ <wff> | ∀R.C | ∃R.⊤ |

  <wff> ⊔ <wff>

- Mother ⊔ Father

  “the notion of parent”

ALE (AL with extended existential)

- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ... | ⊥ | ⊤

  <wff> ::= <Atomic> | ¬<Atomic> | <wff> ⊓ <wff> | ∀R.C | ∃R.⊤ |
  ∃R | ∃R.C

- ∃R (there exists an arbitrary role)
- ∃R.C (full existential quantification)

- Parent ⊓ ∃hasChild.Female

  "parents having at least a daughter"

ALN (AL with number restriction)

- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ... | ⊥ | ⊤

  <wff> ::= <Atomic> | ¬<Atomic> | <wff> ⊓ <wff> | ∀R.C | ∃R.⊤ |

  ≥nR | ≤nR

- ≥nR (at-least number restriction)
- ≤nR (at-most number restriction)

- Parent ⊓ ≥2 hasChild

  "parents having at least two children"

ALC (AL with full concept negation)

- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ... | ⊥ | ⊤

  <wff> ::= <Atomic> | ¬ <wff> | <wff> ⊓ <wff> | ∀R.C | ∃R.⊤

- ¬ (Mother ⊓ Father)

  "it cannot be both a mother and father"

AL's extensions and sub-languages

- By extending *AL* with any subsets of the above constructors yields a particular DL language.
- Each language is denoted by a string of the form *AL*[*U*][*E*][*N*][*C*], where a letter in the name stands for the presence of the corresponding constructor.
- *ALC* is considered the **most important** for many reasons.

  NOTE: *ALU* ⊆ *ALC* and *ALE* ⊆ *ALC*

- By eliminating some of the syntactical symbols and rules, we get some sub-languages of *AL*
- The most important sub-language obtained by elimination in the AL family is ClassL
- We also have FL- and FL0 (where FL = frame language)

From *AL* to ClassL

- *ALUC* with the elimination of roles ∀R.C and ∃R.⊤

- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ... | ⊥ | ⊤

  <wff> ::= <Atomic> | ¬ <wff> | <wff> ⊓ <wff> | <wff> ⊔ <wff>

- The new language is a description language without roles which is ClassL (also called propositional DL)

  NOTE: So far, we are considering DL without TBOX and ABox.

AL's Contractions: *FL-* and *FL0*

- *FL- is AL* with the elimination of $\top$, $\bot$ and $\neg$
- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ...

  <wff> ::= <Atomic> | <wff> $\sqcap$ <wff> | $\forall$R.C | $\exists$R.$\top$

- *FL0 is FL-* with the elimination of $\exists$R.$\top$
- Formation rules:

  <Atomic> ::= A | B | ... | P | Q | ...

  <wff> ::= <Atomic> | <wff> $\sqcap$ <wff> | $\forall$R.C

*AL** Interpretation (Δ,I)

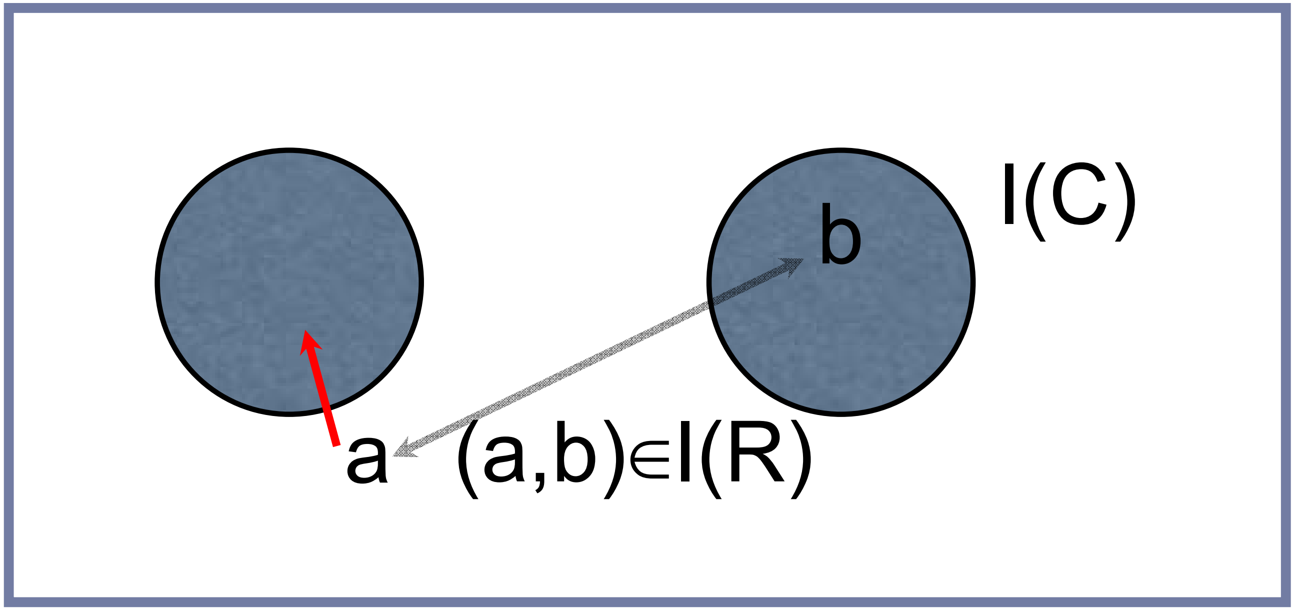
- $I(\bot) = \emptyset$ and $I(\top) = \Delta$ (full domain, "Universe")
- For every concept name A of L, $I(A) \subseteq \Delta$
- $I(\neg C) = \Delta \setminus I(C)$
- $I(C \sqcap D) = I(C) \cap I(D)$
- $I(C \sqcup D) = I(C) \cup I(D)$

The **SAME** as in **ClassL**

- For every role name R of L, $I(R) \subseteq \Delta \times \Delta$
- $I(\forall R.C) = \{a \in \Delta \mid \text{for all } b, \text{ if } (a,b) \in I(R) \text{ then } b \in I(C)\}$
- $I(\exists R.\top) = \{a \in \Delta \mid \text{exists } b \text{ s.t. } (a,b) \in I(R)\}$
- $I(\exists R.C) = \{a \in \Delta \mid \text{exists } b \text{ s.t. } (a,b) \in I(R), b \in I(C)\}$
- $I(\geq nR) = \{a \in \Delta \mid |\{b \mid (a, b) \in I(R)\}| \geq n\}$
- $I(\leq nR) = \{a \in \Delta \mid |\{b \mid (a, b) \in I(R)\}| \leq n\}$

Interpretation of Existential Quantifier

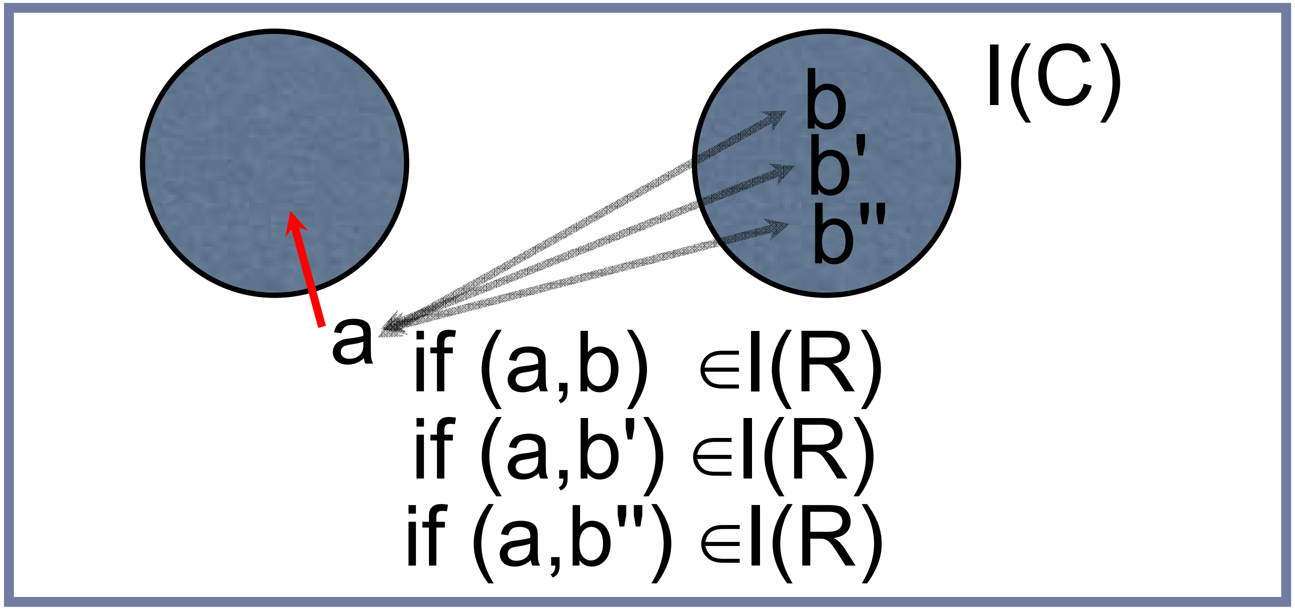
- $I(\exists R.C) = \{a \in \Delta \mid \text{exists } b \text{ s.t. } (a,b) \in I(R),\ b \in I(C)\}$



- Those a that have **some** value b in C with role R.

Interpretation of Value Restriction

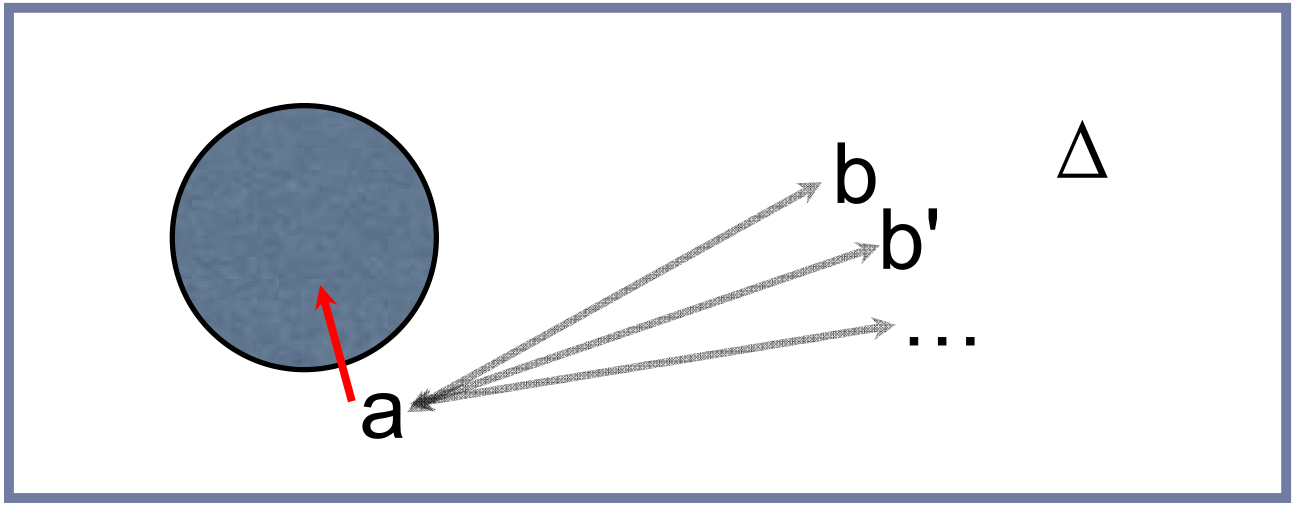
- $I(\forall R.C) = \{a \in \Delta \mid \text{for all } b, \text{ if } (a,b) \in I(R) \text{ then } b \in I(C)\}$



- Those a that have **only** values b in C with role R.

Interpretation of Number Restriction

- $I(\geq nR) = \{a \in \Delta \mid |\{b \mid (a, b) \in I(R)\}| \geq n\}$

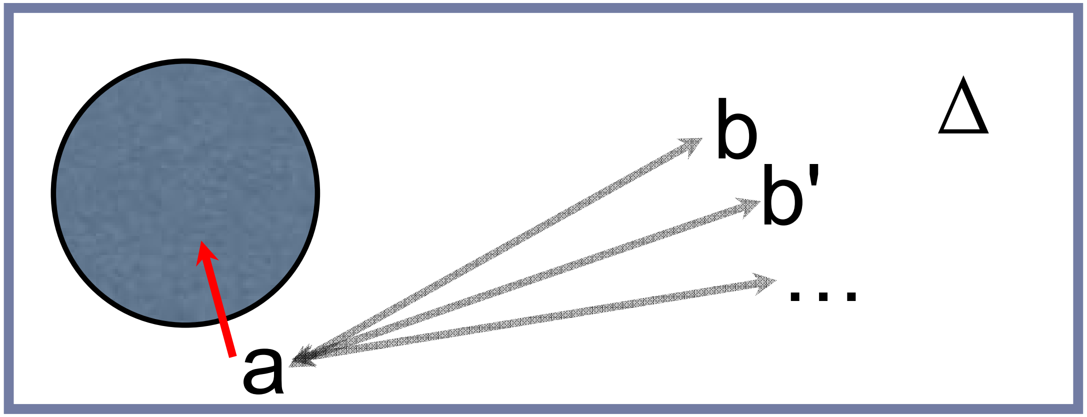


$$|\{b \mid (a, b) \in I(R)\}| \geq n$$

- Those a that have relation R to **at least n** individuals.

Interpretation of Number Restriction Cont.

- $I(\leq nR) = \{a \in \Delta \mid |\{b \mid (a, b) \in I(R)\}| \leq n\}$



$|\{b \mid (a,b) \in I(R)\}| \leq n$

- Those a that have relation R to **at most n** individuals.

Terminology (TBox), same as in ClassL

- A terminology (or TBox) is a set of definitions and specializations
- Terminological axioms express constraints on the concepts of the language, i.e. they limit the possible models
- The TBox is the set of all the constraints on the possible models

Equivalence

TBOX

Equality axiom Definition

Inclusion axiom Specialization

PhD ≡ Postgraduate ⊓ ≥3Publish.Paper

Parent ≡ Person ⊓ □hasChild.Person

hasGrandChild ⊑ hasChild

Subsumption

Reasoning with a TBox T, same as ClassL

- Given two class-propositions P and Q, we want to reason about:
  - Satisfiability w.r.t. T $\quad T \models P$?

    A concept P is satisfiable w.r.t. a terminology T, if there exists an interpretation I with $I \models \theta$ for all $\theta \in T$, and such that $I \models P$, $I(P) \neq \varnothing$
  - Subsumption $\quad T \models P \sqsubseteq Q$? $\quad T \models Q \sqsubseteq P$?

    A concept P is subsumed by a concept Q w.r.t. T if $I(P) \subseteq I(Q)$ for every model I of T
  - Equivalence $\quad T \models P \sqsubseteq Q$ and $T \models Q \sqsubseteq P$?

    Two concepts P and Q are equivalent w.r.t. T if I(P) = I(Q) for every model I of T
  - Disjointness $\quad T \models P \sqcap Q \sqsubseteq \bot$?

    Two concepts P and Q are disjoint with respect to T if their intersection is empty, $I(P) \cap I(Q) = \varnothing$, for every model I of T

ABox, syntax

- In an ABox one introduces individuals, by giving them names, and one *asserts* properties about them.
- We denote individual names as a, b, c,…
- An assertion with concept C is called concept assertion (or simply assertion) in the form:

  C(a), C(b), C(c), …
- An assertion with Role R is called role assertion in the form:

  R(a, b), R(b, c), …

Student(paul)
Professor(fausto)

Teaches(Fausto, LDKR)

ABox, semantics

- An interpretation I: L $\rightarrow$ pow(Δ^I) not only maps atomic concepts to sets, but in addition it maps each individual name a to an element $a^I \in \Delta^I$, namely

$$I(a) = a^I \in \Delta^I$$

$$I(C(a)) = a^I \in C^I,$$

$$I(R(a, b)) = (a^I, b^I) \in R^I$$

- Unique name assumption (UNA). We assume that distinct individual names denote distinct objects in the domain

NOTE: Δ^I denotes the domain of interpretation, a denotes the symbol used for the individual (the name), while a^I is the actual individual of the domain.

Reasoning Services, same as ClassL

- Given an ABox A, we can reason (w.r.t. a TBox T) about the following:
  - Satisfiability/Consistency: An ABox A is consistent with respect to T if there is an interpretation I which is a model of both A and T.
  - Instance checking: checking whether an assertion C(*a*) or R(a,b) is entailed by an ABox, i.e. checking whether *a* belongs to C.

    A ⊨ C(*a*) if every I that satisfies A also satisfies C(*a*).

    A ⊨ R(*a*,*b*) if every I that satisfies A also satisfies R(*a*,*b*).
  - Instance retrieval: given a concept C, retrieve all the instances *a* which satisfy C.
  - Concept realization: given *a* set of concepts and an individual *a* find the most specific concept(s) C (w.r.t. subsumption ordering) such that A ⊨ C(*a*).

Tableaux Calculus

- The Tableaux calculus is a decision procedure to check satisfiability of a DL formula.

- The procedure looks for a model satisfying the formula in input

- The basic idea is to incrementally build the model by looking at the formula and by decomposing it into pieces in a top-down fashion.

- The procedure exhaustively tries all possibilities so that it can eventually prove that no model could be found and therefore the formula is unsatisfiable.

Preview example

$C = (\exists R.A) \sqcap (\exists R.B) \sqcap (\exists R.\neg(A \sqcap B))$

$C = (\exists R.A) \sqcap (\exists R.B) \sqcap (\exists R.(\neg A \sqcup \neg B))$ De Morgan

In Negation Normal Form

C is safisfiable iff $I(C) \neq \varnothing$ for some I

$C1 = \exists R.A$ $C2 = \exists R.B$ $C3 = \exists R.(\neg A \sqcup \neg B)$ Decomposition

$C1 \Rightarrow \exists\ (b,c) \in I(R)$ and $c \in I(A)$

$C2 \Rightarrow \exists\ (b,d) \in I(R)$ and $d \in I(B)$

$C3 \Rightarrow \exists\ (b,e) \in I(R)$ and $e \in I(\neg A \sqcup \neg B) \Rightarrow e \in I(\neg A)$ or $I(\neg B)$

If we take e=c, must be $e \in I(\neg B)$ otherwise it reaches a contradiction

If we take e=d, must be $e \in I(\neg A)$ otherwise it reaches a contradiction

The Tableau Algorithm

- The formula C in input is translated into Negation Normal Form.

- An ABox A is incrementally constructed by adding assertions according to the constraints in C (identified by decomposition) following precise transformation rules
- Each time we have more than one option we split the space of the solutions as in a decision tree (i.e. in presence of ⊔)
- When a contradiction is found (i.e. A is inconsistent) we need to try another path in the space of the solutions (backtracking)

- The algorithm stops when either we find a consistent A satisfying all the constraints in C (the formula is satisfiable) or there is no consistent A (the formula is unsatisfiable)

Transformation rules

- ⊓-rule

  Condition: A contains (C1 ⊓ C2)(x), but not both C1(x) and C2(x)

  Action: A' = A ∪ {C1(x), C2(x)}

T={Mother ≡ Female ⊓ □hasChild.Person}   A={Mother(Anna)}
Is ¬□hasChild.Person ⊓ ¬□hasParent. Person) satisfiable?

Expand A w.r.t. T
Mother(Anna) ⇨ (Female ⊓ □hasChild.Person)(Anna) ⇨
A' = A □ {Female(Anna), (□hasChild.Person)(Anna)}

(¬□hasChild.Person ⊓ ¬□hasParent.Person)(Anna) ⇨
(¬□hasChild.(Person))(Anna) ⊓ (¬□hasParent.(Person))(Anna)
Both of them must be true, but the first constraint is clearly
in contradiction with A'

Transformation rules

- ⊔-rule

  Condition: A contains (C1 ⊔ C2)(x), but neither C1(x) or C2(x)

  Action: A’ = A ∪ {C1(x)} and A’’ = A ∪ {C2(x)}

T={Parent≡□hasChild.Female⊔∃hasChild.Male,
Person≡Male⊔Female, Mother≡Parent ⊓Female}
A={Mother(Anna)}
Is ¬(□hasChild.Person) satisfiable?

Expand A w.r.t. T
A = {Mother(Anna)} ⇨ A’ = A □ {Parent(Anna), Female(Anna)}
Parent(Anna) ⇨ (□hasChild.Female⊔∃hasChild.Male)(Anna) ⇨
(□hasChild.Female)(Anna) or (□hasChild.Male)(Anna)

Both are in contradiction with ¬(□hasChild.Person)

Transformation rules

- ∃-rule

  Condition: A contains (∃R.C)(x), but there is no z such that both C(z) and R(x,z) are in A

  Action: A’ = A ∪ {C(z), R(x,z)}

T={Parent≡□hasChild.Female⊔∃hasChild.Male,
  Person≡Male⊔Female, Mother≡Parent⊓Female}
A={Mother(Anna), hasChild(Anna,Bob), ¬Female(Bob)}
Is ¬(□hasChild.Person) satisfiable?

Expand A w.r.t. T
Mother(Anna) ⇨ Parent(Anna) ⇨
(□hasChild.Female⊔∃hasChild.Male)(Anna)

take (□hasChild.Male)(Anna) ⇨ hasChild(Anna,Bob), Male(Bob) …

Transformation rules

- ∀-rule

  Condition: A contains (∀R.C)(x) and R(x,z), but it does not C(z)

  Action: A' = A ∪ {C(z)}

T={DaughterParent≡□hasChild.Female, Male⊓Female⊑⊥}
A={hasChild(Anna,Bob), ¬Female(Bob)}
Is DaughterParent satisfiable?

Expand A w.r.t. T
DaughterParent(x) ⇨ □hasChild.Female(x) ⇨
A' = A □ {Female(Bob)}

but this in contradiction with ¬Female(Bob)

Example of Tableau Reasoning

- Is $\forall hasChild.Male \sqcap \exists hasChild.\neg Male$ satisfiable?

  NOTE: we do not have an initial T or A

  $(\forall hasChild.Male \sqcap \exists hasChild.\neg Male)(x)$ ⇨
  $A = \{(\forall hasChild.Male)(x), (\exists hasChild.\neg Male)(x)\}$ ⊓-rule

  $(\exists hasChild.\neg Male)(x)$ ⇨ $A' = A \cup \{hasChild(x,y), \neg Male(y)\}$ ∃-rule

  $(\forall hasChild.Male)(x), hasChild(x,y)$ ⇨ $A'' = A' \cup Male(y)$ ∀-rule

  A'' is clearly inconsistent

Additional Rules

The $\rightarrow_{\geq}$-rule

Condition: $\mathcal{A}$ contains $(\geqslant n\, R)(x)$, and there are no individual names $z_1, \ldots, z_n$ such that $R(x, z_i)$ $(1 \leq i \leq n)$ and $z_i \neq z_j$ $(1 \leq i < j \leq n)$ are contained in $\mathcal{A}$.

Action: $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid 1 \leq i < j \leq n\}$, where $y_1, \ldots, y_n$ are distinct individual names not occurring in $\mathcal{A}$.

The $\rightarrow_{\leq}$-rule

Condition: $\mathcal{A}$ contains distinct individual names $y_1, \ldots, y_{n+1}$ such that $(\leqslant n\, R)(x)$ and $R(x, y_1), \ldots, R(x, y_{n+1})$ are in $\mathcal{A}$, and $y_i \neq y_j$ is not in $\mathcal{A}$ for some $i \neq j$.

Action: For each pair y_i, y_j such that $i > j$ and $y_i \neq y_j$ is not in $\mathcal{A}$, the ABox $\mathcal{A}_{i,j} = [y_i/y_j]\mathcal{A}$ is obtained from $\mathcal{A}$ by replacing each occurrence of y_i by y_j.

Complexity of Tableau Algorithms

- The satisfiability algorithm of ALCN may need exponential time and space. It is PSPACE-complete.
- An optimized algorithm needs only polynomial space as it assumes a depth-first search and stores only the 'correct' path.
- The consistency and instance checking problem for ALCN are also PSPACE-complete.
- The complexity results for other Description Logics varies according to corresponding constructors.