

DL $\mathcal{A}\mathcal{L}\mathcal{C}$

Outline

Topics:

- 1 Introduction to description logics
- 2 The description logic \mathcal{ALC}
- 3 Extensions to \mathcal{ALC}
- 4 A tableau algorithm for \mathcal{ALC}

Bevezetés

Leíró logika (Description Logics - DL)

- A DL egy logikai nyelv **fogalmak**, és ezek attribútumainak (ebben a környezetben: **szerepek**) és az ezek közti **relációk** leírására
 - Pl. **Személy** egy fogalom és a **Szülője** lehet egy hozzátartozó szerep.
- Can be regarded as a KR system based on a *structured representation of knowledge*.
- Most DLs are fragments of FOL, written in a distinct syntax.

Predecessors of DLs

- Semantic networks of the 70s
- Frame-based systems

Why Description Logics?

Ideal AI case:

- Approaches have scientific (logical) and engineering aspects
- *Scientific*: Analyse the problem formally and in detail
- *Engineering*: Get something working quickly and efficiently
- Success:
When these two approaches coincide – efficient implementations of (formally) well-understood systems.
- Description Logic research has (arguably) reached this point

Background: Concepts, Roles, Constants

- In a description logic, there are sentences that will be true or false (as in FOL).
 - These are restricted to *subsumption* and *instance* assertions.
- In addition, there are three sorts of expressions that act like nouns and noun phrases in English:
 - *Concepts* are like category nouns: Person, Female, GraduateStudent
 - *Roles* are like relational nouns: AgeOf, ParentOf, AreaOfStudy
 - Specify attributes of concepts and their types
 - *Constants* are like proper nouns: John, Mary
- These correspond to unary predicates, binary predicates and constants (respectively) in FOL.
- Unlike in FOL, concepts need not be atomic and can have structure.

DL Knowledge Bases

An KB in a DL contains two parts:

- Define terminology: *TBox*
 - E.g. $MWD \doteq Mother \sqcap \forall ParentOf. \neg Female$
- Give assertions: *ABox*
 - E.g. $MWD(sue)$.

DL Knowledge Bases: TBox

Main components of the TBox:

- *Concepts*: classes of individuals
 - E.g. *Mother*

DL Knowledge Bases: TBox

Main components of the TBox:

- *Concepts*: classes of individuals
 - E.g. *Mother*
- *Roles*: binary relations between individuals
 - E.g. $\forall \text{ParentOf}.\neg \text{Female}$

DL Knowledge Bases: TBox

Main components of the TBox:

- *Concepts*: classes of individuals
 - E.g. *Mother*
- *Roles*: binary relations between individuals
 - E.g. $\forall \text{ParentOf} . \neg \text{Female}$
- *Complex concepts* using constructors
 - E.g. $\text{Mother} \sqcap \forall \text{ParentOf} . \neg \text{Female}$

DL Knowledge Bases: TBox

Main components of the TBox:

- *Concepts*: classes of individuals
 - E.g. *Mother*
- *Roles*: binary relations between individuals
 - E.g. $\forall \text{ParentOf} . \neg \text{Female}$
- *Complex concepts* using constructors
 - E.g. $\text{Mother} \sqcap \forall \text{ParentOf} . \neg \text{Female}$
- *Assertions* concerning complex concepts
 - E.g. $MWD \doteq \text{Mother} \sqcap \forall \text{ParentOf} . \neg \text{Female}$
 $\text{Mother} \sqsubseteq \text{Female}$

DL Knowledge Bases: ABox

ABox: Assertions that individuals satisfy certain concepts and roles.

- Think of as a (very) simple relational database.
- E.g. $MWD(Mary)$, $ParentOf(Mary, John)$.

DL: Advantages

- Well-defined formal semantics.
- Known (and often good) complexity characteristics or implementations.
- Relatively easy to specify DL knowledge bases, in a structured hierarchical fashion.
- DLs constitute a *large* family of approaches.
 - Can tailor a language to a specific application.

Applications

Useful whenever a common vocabulary is important.

E.g.:

- Enhanced database systems
 - *DL-Lite*
- Medical informatics: SnoMed, Galen
 - *εL*
- Semantic Web
 - Next generation web
 - *OWL*: W3C recommendation.

👉 We'll look at perhaps the most central DL, *ALC*.

The Logic \mathcal{ALC}

An \mathcal{ALC} KB contains two parts:

- Define terminology: TBox
- Give assertions: ABox

The Logic *ALC*

An *ALC* KB contains two parts:

- Define terminology: TBox
- Give assertions: ABox

Main components of the TBox:

- Concepts: Represent classes of individuals
- Roles: Represent binary relations between individuals
- Complex concepts using constructors

Examples:

- Concept names: Person, Female
- Role names: ParentOf, HasHusband
- Individual names (in the ABox): John, Mary

The Logic \mathcal{ALC} : Language

Logical symbols:

- Propositional constructors: \sqcap, \sqcup, \neg
- Other restrictions: \forall, \exists
- \top, \perp

The Logic \mathcal{ALC} : Language

Logical symbols:

- Propositional constructors: \sqcap, \sqcup, \neg
- Other restrictions: \forall, \exists
- \top, \perp

Nonlogical symbols:

- Concept names
- Role names

The Logic \mathcal{ALC} : Language

Logical symbols:

- Propositional constructors: \sqcap, \sqcup, \neg
- Other restrictions: \forall, \exists
- \top, \perp

Nonlogical symbols:

- Concept names
- Role names

Concept construction

- Let C and D be concepts and R a role.
- $\neg C, C \sqcap D, C \sqcup D$ are concepts.
- $\forall R.C, \exists R.C$ are concepts.

The Logic \mathcal{ALC} : Language

Let C and D be concepts and R a role.

- C stands for a concept or set of individuals.

The Logic \mathcal{ALC} : Language

Let C and D be concepts and R a role.

- C stands for a concept or set of individuals.
- $\neg C$ stands for the concept of things that are not a C .

The Logic \mathcal{ALC} : Language

Let C and D be concepts and R a role.

- C stands for a concept or set of individuals.
- $\neg C$ stands for the concept of things that are not a C .
- $C \sqcap D$ is the concept of things that are both C and D .
 - E.g. *Female* \sqcap *Human*

The Logic \mathcal{ALC} : Language

Let C and D be concepts and R a role.

- C stands for a concept or set of individuals.
- $\neg C$ stands for the concept of things that are not a C .
- $C \sqcap D$ is the concept of things that are both C and D .
 - E.g. *Female* \sqcap *Human*
- $C \sqcup D$ is the concept of things that are either C or D or both.
 - E.g. *Male* \sqcup *Female*

The Logic \mathcal{ALC} : Language

Let C and D be concepts and R a role.

- C stands for a concept or set of individuals.
- $\neg C$ stands for the concept of things that are not a C .
- $C \sqcap D$ is the concept of things that are both C and D .
 - E.g. $Female \sqcap Human$
- $C \sqcup D$ is the concept of things that are either C or D or both.
 - E.g. $Male \sqcup Female$
- $\forall R.C$ is the concept of things such that all things that are R related to it are C 's.
 - E.g. $\forall ParentOf.Female$: things all of whose children are female

The Logic \mathcal{ALC} : Language

Let C and D be concepts and R a role.

- C stands for a concept or set of individuals.
- $\neg C$ stands for the concept of things that are not a C .
- $C \sqcap D$ is the concept of things that are both C and D .
 - E.g. $Female \sqcap Human$
- $C \sqcup D$ is the concept of things that are either C or D or both.
 - E.g. $Male \sqcup Female$
- $\forall R.C$ is the concept of things such that all things that are R related to it are C 's.
 - E.g. $\forall ParentOf.Female$: things all of whose children are female
- $\exists R.C$ is the concept of things such that some thing R related to it is a C .
 - $\exists ParentOf.Female$: things with a female child

The Logic \mathcal{ALC} : Knowledge Bases

Axioms (assertions) in the TBox:

- Subsumption: $C \sqsubseteq D$ where C and D are concepts
- Equivalence axioms: $C \doteq D$ where C and D are concepts

The Logic \mathcal{ALC} : Knowledge Bases

Axioms (assertions) in the TBox:

- Subsumption: $C \sqsubseteq D$ where C and D are concepts
- Equivalence axioms: $C \doteq D$ where C and D are concepts

Assertions in the ABox:

- $C(a)$ where C is a concept and a is an individual name.
- $R(a, b)$ where R is a role name, a and b are individual names.

The Logic \mathcal{ALC} : Knowledge Bases

Axioms (assertions) in the TBox:

- Subsumption: $C \sqsubseteq D$ where C and D are concepts
- Equivalence axioms: $C \doteq D$ where C and D are concepts

Assertions in the ABox:

- $C(a)$ where C is a concept and a is an individual name.
- $R(a, b)$ where R is a role name, a and b are individual names.

DL knowledge base:

- Set of TBox statements
- Set of ABox statements

Examples

TBox:

- $Person \sqsubseteq Animal \sqcap Biped$
- $Woman \doteq Person \sqcap Female$
- $Mother \doteq Woman \sqcap \exists ParentOf . Person$
- $Parent \doteq Mother \sqcup Father$
- $Man \doteq Person \sqcap \neg Woman$
- $MotherWithoutDaughter \doteq Mother \sqcap \forall ParentOf . \neg Female$
- $GrandMother \doteq Woman \sqcap \exists ParentOf . Parent$

ABox:

- $GrandMother(Sally)$
- $(Person \sqcap Male)(John)$

Formal Semantics for Concepts and Names

- ☞ Semantically, a DL can be seen as a fragment of FOL

Formal Semantics for Concepts and Names

☞ Semantically, a DL can be seen as a fragment of FOL

An interpretation is a pair $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$

- Domain Δ : non-empty set of objects
- Interpretation function $\cdot^{\mathcal{I}}$: Maps structures into the domain.
- Recall, Brachman and Levesque write this as $\mathcal{I} = \langle D, I \rangle$.

Formal Semantics for Concepts and Names

☞ Semantically, a DL can be seen as a fragment of FOL

An interpretation is a pair $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$

- Domain Δ : non-empty set of objects
- Interpretation function $\cdot^{\mathcal{I}}$: Maps structures into the domain.
- Recall, Brachman and Levesque write this as $\mathcal{I} = \langle D, I \rangle$.

Then:

- $\cdot^{\mathcal{I}}$ maps every concept name A to a subset $A^{\mathcal{I}} \subseteq \Delta$
- $\cdot^{\mathcal{I}}$ maps every role name R to a binary relation $R^{\mathcal{I}} \subseteq \Delta \times \Delta$
- $\cdot^{\mathcal{I}}$ maps individual names a to elements of Δ : $a^{\mathcal{I}} \in \Delta$
- $\top^{\mathcal{I}} = \Delta$ and $\perp^{\mathcal{I}} = \emptyset$.

Semantics for Complex Concepts

Assume C , D are concepts, and R is a role.

- $(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\forall R.C)^{\mathcal{I}} = \{x \mid y \in C^{\mathcal{I}} \text{ for every } y \text{ s.t. } (x, y) \in R^{\mathcal{I}}\}$
- $(\exists R.C)^{\mathcal{I}} = \{x \mid y \in C^{\mathcal{I}} \text{ for some } y \text{ s.t. } (x, y) \in R^{\mathcal{I}}\}$

Semantics for Axioms and Assertions

Assume C , D are concepts, R is a role, a and b are individual names.

Let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be an interpretation.

- $C \sqsubseteq D$ is true in \mathcal{I} iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- $C \doteq D$ is true in \mathcal{I} iff $C^{\mathcal{I}} = D^{\mathcal{I}}$
- $C(a)$ is true in \mathcal{I} iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$
- $R(a, b)$ is true in \mathcal{I} iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

Reasoning in \mathcal{ALC}

- Sentences: Axioms or assertions
- \mathcal{I} is a *model* for a sentence S iff S is true in \mathcal{I}
- \mathcal{I} is a model for a DL knowledge base K iff it is a model for every sentence in K
- Models of K are denoted by $[K]$
- S is *entailed* by K , written $K \models S$ iff $[K] \subseteq [S]$
(I.e. every model of K is a model of S .)

Types of Reasoning in \mathcal{ALC}

K a DL knowledge base;

C and D are concepts;

R is a role;

a and b are individual names

- Instance checking: $K \models C(a)$ or $K \models R(a, b)$

Types of Reasoning in \mathcal{ALC}

K a DL knowledge base;

C and D are concepts;

R is a role;

a and b are individual names

- Instance checking: $K \models C(a)$ or $K \models R(a, b)$
- Subsumption checking: $K \models C \sqsubseteq D$

Types of Reasoning in \mathcal{ALC}

K a DL knowledge base;

C and D are concepts;

R is a role;

a and b are individual names

- Instance checking: $K \models C(a)$ or $K \models R(a, b)$
- Subsumption checking: $K \models C \sqsubseteq D$
- Equivalence checking: $K \models C \doteq D$

Types of Reasoning in \mathcal{ALC}

K a DL knowledge base;

C and D are concepts;

R is a role;

a and b are individual names

- Instance checking: $K \models C(a)$ or $K \models R(a, b)$
- Subsumption checking: $K \models C \sqsubseteq D$
- Equivalence checking: $K \models C \doteq D$
- Consistency checking: $K \not\models \top \sqsubseteq \perp$

Types of Reasoning in \mathcal{ALC}

K a DL knowledge base;

C and D are concepts;

R is a role;

a and b are individual names

- Instance checking: $K \models C(a)$ or $K \models R(a, b)$
- Subsumption checking: $K \models C \sqsubseteq D$
- Equivalence checking: $K \models C \doteq D$
- Consistency checking: $K \not\models \top \sqsubseteq \perp$
- Concept satisfiability: $K \not\models C \sqsubseteq \perp$

Types of Reasoning in \mathcal{ALC}

K a DL knowledge base;

C and D are concepts;

R is a role;

a and b are individual names

- Instance checking: $K \models C(a)$ or $K \models R(a, b)$
- Subsumption checking: $K \models C \sqsubseteq D$
- Equivalence checking: $K \models C \doteq D$
- Consistency checking: $K \not\models \top \sqsubseteq \perp$
- Concept satisfiability: $K \not\models C \sqsubseteq \perp$
- Disjoint concepts: $K \models C \sqcap D \sqsubseteq \perp$

Reduction to Consistency Checking

Let b be a new individual

- Instance checking:

$$K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \perp$$

Reduction to Consistency Checking

Let b be a new individual

- Instance checking:

$$K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \perp$$

- Subsumption checking:

$$K \models C \sqsubseteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b)\} \models \top \sqsubseteq \perp$$

Reduction to Consistency Checking

Let b be a new individual

- Instance checking:

$$K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \perp$$

- Subsumption checking:

$$K \models C \sqsubseteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b)\} \models \top \sqsubseteq \perp$$

- Equivalence checking:

$$K \models C \doteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b), (\neg C \sqcap D)(b)\} \models \top \sqsubseteq \perp$$

Reduction to Consistency Checking

Let b be a new individual

- Instance checking:

$$K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \perp$$

- Subsumption checking:

$$K \models C \sqsubseteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b)\} \models \top \sqsubseteq \perp$$

- Equivalence checking:

$$K \models C \doteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b), (\neg C \sqcap D)(b)\} \models \top \sqsubseteq \perp$$

- Concept satisfiability:

$$K \not\models C \sqsubseteq \perp \text{ iff } K \cup \{C(b)\} \not\models \top \sqsubseteq \perp$$

Reduction to Consistency Checking

Let b be a new individual

- Instance checking:

$$K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \perp$$

- Subsumption checking:

$$K \models C \sqsubseteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b)\} \models \top \sqsubseteq \perp$$

- Equivalence checking:

$$K \models C \doteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b), (\neg C \sqcap D)(b)\} \models \top \sqsubseteq \perp$$

- Concept satisfiability:

$$K \not\models C \sqsubseteq \perp \text{ iff } K \cup \{C(b)\} \not\models \top \sqsubseteq \perp$$

- Disjoint concepts:

$$K \models C \sqcap D \sqsubseteq \perp \text{ iff } K \cup \{(C \sqcap D)(b)\} \models \top \sqsubseteq \perp$$

Aside: Extensions to \mathcal{ALC}

Extended concepts

- Number restrictions: $(\leq nR.C)$ and $(\geq nR.C)$

Role operators

- Inverse roles: R^- where R is a role

Role axioms

- Role hierarchy: $R \sqsubseteq S$ where R and S are roles
👉 So far have just used \sqsubseteq for concepts.
- Transitive roles: $R \in R^+$ where R is a role

Extensions to \mathcal{ALC} : Examples

- $ParentWithManySons \doteq (\geq 3ParentOf.Male)$
- $\exists ParentOf^-.Citizen \sqsubseteq Citizen$
- $ParentOf \sqsubseteq AncestorOf$
- $AncestorOf \in R^+$

Extensions to \mathcal{ALC} : Semantics

- $(\leq nR.C)^{\mathcal{I}} = \{x \mid |\{y \in C^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\}| \leq n\}$
- $(\geq nR.C)^{\mathcal{I}} = \{x \mid |\{y \in C^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\}| \geq n\}$
- Inverse roles: $(R^{-})^{\mathcal{I}} = \{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$
- $R \sqsubseteq S$ is true in I iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for roles R and S .
- $R \in R^{+}$ is true in I iff
 $(x,z) \in R^{\mathcal{I}}$ whenever $(x,y) \in R^{\mathcal{I}}$ and $(y,z) \in R^{\mathcal{I}}$

A Tableau Algorithm for \mathcal{ALC}

Goal: Show $KB \models A \sqsubseteq B$ by showing $KB \cup \{A \sqcap \neg B\}$ unsatisfiable.

Assume an *unfoldable terminology*:

- Axioms are of the form $A \sqsubseteq C$ and $A \doteq C$ where A is a concept name.
- For each concept name A , at most one axiom of the form $A \sqsubseteq C$ or $A \doteq C$.
- Axioms are acyclic:
 - $A \sqsubseteq C$ or $A \doteq C$ *directly uses* a concept name A_1 iff A_1 occurs in C .
 - $A \sqsubseteq C$ or $A \doteq C$ *uses* a concept name A_1 iff it directly uses A_1 or it directly uses a concept name A_2 and A_2 uses A_1 .
 - $A \sqsubseteq C$ or $A \doteq C$ is *acyclic* iff it does not use A .

General Method

Show $KB \models A \sqsubseteq B$ by showing $KB \cup \{A \sqcap \neg B\}$ is unsatisfiable.

Try to prove concept (un)satisfiability by constructing a model.

- A *tableau* is a graph representing such a model.
- A set of tableau *expansion rules* is used to construct the tableau.
- Either a model is constructed or a contradiction is found.

General Method

At the start:

- Assume an unfoldable terminology.
- Assume that all axioms are of the form $P \doteq Q$
 - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \doteq B \sqcap C$ where C is a new concept name.

General Method

At the start:

- Assume an unfoldable terminology.
- Assume that all axioms are of the form $P \doteq Q$
 - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \doteq B \sqcap C$ where C is a new concept name.

If the query is $A \sqsubseteq B$:

- *negate* the query to get $A \sqcap \neg B$ (to show unsatisfiable);
- *unfold* the negated query;
- *convert* to *negation normal form*.


General Method

At the start:

- Assume an unfoldable terminology.
- Assume that all axioms are of the form $P \doteq Q$
 - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \doteq B \sqcap C$ where C is a new concept name.

If the query is $A \sqsubseteq B$:

- *negate* the query to get $A \sqcap \neg B$ (to show unsatisfiable);
- *unfold* the negated query;
- *convert* to *negation normal form*.

 Once the negated query has been unfolded, the rest of the KB can be ignored.

To Start

Unfold:

Expand every concept name occurring in the (negated) query.

- I.e. if concept C appears in the query and $C \doteq D$ is in the KB, replace C by D in the query.
 - Recall that for $C \doteq D$ in the KB, C is a concept name and D is an arbitrary \mathcal{ALC} concept expression.

To Start

Unfold:

Expand every concept name occurring in the (negated) query.

- I.e. if concept C appears in the query and $C \doteq D$ is in the KB, replace C by D in the query.
 - Recall that for $C \doteq D$ in the KB, C is a concept name and D is an arbitrary \mathcal{ALC} concept expression.

Negation normal form:

Negation occurs only in front of concept names

- $\neg(C \sqcap D)$ gives $\neg C \sqcup \neg D$, and
 $\neg(C \sqcup D)$ gives $\neg C \sqcap \neg D$
- $\neg\exists R.C$ gives $\forall R.\neg C$, and
 $\neg\forall R.C$ gives $\exists R.\neg C$
- $\neg\neg C$ gives C

Algorithm

- Use a tree to represent the model being constructed
- Each node x represents an individual, labelled with a set $L(x)$ of concepts it has to satisfy
 - $C \in L(x)$ implies $x \in C^{\mathcal{I}}$
- Each edge (x, y) represents a pair occurring in the interpretation of a role, labelled with the role name
 - $R = L((x, y))$ implies $(x, y) \in R^{\mathcal{I}}$

To Determine the Satisfiability of a Concept C

- Initialise the tree T with a single node x with $L(x) = \{C\}$.
- Expand by repeatedly applying a set of *expansion rules*.
- T is *fully expanded* when none of the rules can be applied.
- T contains a *clash* when, for a node y and a concept D ,
 $\perp \in L(y)$ or $\{D, \neg D\} \subseteq L(y)$.
- If T can't be expanded without producing a clash, the concept is unsatisfiable.

Expansion Rules

(\sqcap -rule) If $(C_1 \sqcap C_2) \in L(x)$ and $\{C_1, C_2\} \not\subseteq L(x)$ then:
Add C_1 and C_2 to $L(x)$.

Expansion Rules

- (\sqcap -rule) If $(C_1 \sqcap C_2) \in L(x)$ and $\{C_1, C_2\} \not\subseteq L(x)$ then:
Add C_1 and C_2 to $L(x)$.
- (\sqcup -rule) If $(C_1 \sqcup C_2) \in L(x)$ and $\{C_1, C_2\} \cap L(x) = \emptyset$ then:
Add C_1 to $L(x)$.
If this leads to a clash, go back and add C_2 to $L(x)$.

Expansion Rules

- (\sqcap -rule) If $(C_1 \sqcap C_2) \in L(x)$ and $\{C_1, C_2\} \not\subseteq L(x)$ then:
Add C_1 and C_2 to $L(x)$.
- (\sqcup -rule) If $(C_1 \sqcup C_2) \in L(x)$ and $\{C_1, C_2\} \cap L(x) = \emptyset$ then:
Add C_1 to $L(x)$.
If this leads to a clash, go back and add C_2 to $L(x)$.
- (\exists -rule) If $\exists R.C \in L(x)$ and there is no y s.t. $L((x, y)) = R$
and $C \in L(y)$ then:
Create a new node y and edge (x, y) with $L(y) = C$
and $L((x, y)) = R$.

Expansion Rules

- (\sqcap -rule) If $(C_1 \sqcap C_2) \in L(x)$ and $\{C_1, C_2\} \not\subseteq L(x)$ then:
Add C_1 and C_2 to $L(x)$.
- (\sqcup -rule) If $(C_1 \sqcup C_2) \in L(x)$ and $\{C_1, C_2\} \cap L(x) = \emptyset$ then:
Add C_1 to $L(x)$.
If this leads to a clash, go back and add C_2 to $L(x)$.
- (\exists -rule) If $\exists R.C \in L(x)$ and there is no y s.t. $L((x, y)) = R$
and $C \in L(y)$ then:
Create a new node y and edge (x, y) with $L(y) = C$
and $L((x, y)) = R$.
- (\forall -rule) If $\forall R.C \in L(x)$ and there is some y s.t.
 $L((x, y)) = R$ and $C \notin L(y)$ then:
Add C to $L(y)$.

Interpreting a tree T

- If T contains a clash the concept C is unsatisfiable.
- If T is fully expanded and clash-free, then C is satisfiable.
- In the second case, construct a model I as follows:
 - $\Delta = \{x \mid x \text{ is a node in } T\}$.
 - $A^I = \{x \in \Delta \mid A \in L(x)\}$ for all concept names A in C .
 - $R^I = \{(x, y) \mid (x, y) \text{ is an edge in } T \text{ and } L((x, y)) = R\}$.

Termination of the Algorithm

- The \sqcap -, \sqcup -and \exists -rules can only be applied once to a concept in $L(x)$.
- The \forall -rule can be applied many times to a given $\forall R.C$ expression in $L(x)$, but only once to a given edge (x, y) .
- Applying any rule to a concept C extends the labelling with a concept strictly smaller than C .



Therefore the algorithm must terminate.

Tableau Algorithm: Example 1

DL knowledge base:

- $vegan \doteq person \sqcap \forall eats.plant$
- $vegetarian \doteq person \sqcap \forall eats.(plants \sqcup dairy)$

Query: $vegan \sqsubseteq vegetarian$

Convert to:

- $vegan \sqcap \neg vegetarian$ is unsatisfiable ?

Example 1

- Unfold and normalise $vegan \sqcap \neg vegetarian$:
 $person \sqcap \forall eats.plant \sqcap (\neg person \sqcup \exists eats.(\neg plant \sqcap \neg dairy))$

Example 1

- Unfold and normalise $vegan \sqcap \neg vegetarian$:
 $person \sqcap \forall eats.plant \sqcap (\neg person \sqcup \exists eats.(\neg plant \sqcap \neg dairy))$
- Initialise T to $L(x)$ to contain:
 $person \sqcap \forall eats.plant \sqcap (\neg person \sqcup \exists eats.(\neg plant \sqcap \neg dairy))$

Example 1

- Unfold and normalise $\text{vegan} \sqcap \neg\text{vegetarian}$:
 $\text{person} \sqcap \forall \text{eats.plant} \sqcap (\neg\text{person} \sqcup \exists \text{eats} . (\neg\text{plant} \sqcap \neg\text{dairy}))$
- Initialise T to $L(x)$ to contain:
 $\text{person} \sqcap \forall \text{eats.plant} \sqcap (\neg\text{person} \sqcup \exists \text{eats} . (\neg\text{plant} \sqcap \neg\text{dairy}))$
- Apply \sqcap -rule and add to $L(x)$:
 $\{\text{person}, \forall \text{eats.plant}, \neg\text{person} \sqcup \exists \text{eats} . (\neg\text{plant} \sqcap \neg\text{dairy})\}$

Example 1

- Apply \sqcup -rule to $\neg person \sqcup \exists eats.(\neg plant \sqcap \neg dairy)$:
Add $\neg person$ to $L(x)$: Clash
Go back and add $\exists eats.(\neg plant \sqcap \neg dairy)$ to $L(x)$

Example 1

- Apply \sqcup -rule to $\neg person \sqcup \exists eats.(\neg plant \sqcap \neg dairy)$:
Add $\neg person$ to $L(x)$: Clash
Go back and add $\exists eats.(\neg plant \sqcap \neg dairy)$ to $L(x)$
- Apply \exists -rule to $\exists eats.(\neg plant \sqcap \neg dairy)$:
Create new node y and new edge (x, y) :
 $L(y) = \{\neg plant \sqcap \neg dairy\}$; $L((x, y)) = eats$

Example 1

- Apply \sqcup -rule to $\neg person \sqcup \exists eats.(\neg plant \sqcap \neg dairy)$:
Add $\neg person$ to $L(x)$: Clash
Go back and add $\exists eats.(\neg plant \sqcap \neg dairy)$ to $L(x)$
- Apply \exists -rule to $\exists eats.(\neg plant \sqcap \neg dairy)$:
Create new node y and new edge (x, y) :
 $L(y) = \{\neg plant \sqcap \neg dairy\}$; $L((x, y)) = eats$
- Apply \forall -rule to $\forall eats.plant$ in $L(x)$ and $L((x, y)) = eats$:
Add $plant$ to $L(y)$

Example 1

- Apply \Box -rule to $\neg plant \Box \neg dairy$ in $L(y)$:
Add $\{\neg plant, \neg dairy\}$ to $L(y)$: Clash

Example 1

- Apply \sqcap -rule to $\neg plant \sqcap \neg dairy$ in $L(y)$:
Add $\{\neg plant, \neg dairy\}$ to $L(y)$: Clash
- Conclusion
 - Both applications of the \sqcup -rule lead to clashes
 - So $vegan \sqcap \neg vegetarian$ is unsatisfiable
 - So $vegan \sqsubseteq vegetarian$

Example 2

- Query: $vegetarian \sqsubseteq vegan$
- Convert to: $vegetarian \sqcap \neg vegan$ is satisfiable ?
- Unfold and normalise $vegetarian \sqcap \neg vegan$:
 $person \sqcap \forall eats.(plant \sqcup dairy) \sqcap (\neg person \sqcup \exists eats.\neg plant)$
- Initialise T to $L(x)$ to contain:
 $\{person \sqcap \forall eats.(plant \sqcup dairy) \sqcap (\neg person \sqcup \exists eats.\neg plant)\}$

Example 2

- Apply \sqsupset -rule and add to $L(x)$:
 $\{person, \forall eats.(plant \sqcup dairy), \neg person \sqcup \exists eats.\neg plant\}$

Example 2

- Apply \sqcap -rule and add to $L(x)$:
 $\{person, \forall eats.(plant \sqcup dairy), \neg person \sqcup \exists eats.\neg plant\}$
- Apply \sqcup -rule to $\neg person \sqcup \exists eats.\neg plant$:
Add $\neg person$ to $L(x)$: Clash
Go back and add $\exists eats.\neg plant$ to $L(x)$

Example 2

- Apply \sqcap -rule and add to $L(x)$:
 $\{person, \forall eats.(plant \sqcup dairy), \neg person \sqcup \exists eats.\neg plant\}$
- Apply \sqcup -rule to $\neg person \sqcup \exists eats.\neg plant$:
Add $\neg person$ to $L(x)$: Clash
Go back and add $\exists eats.\neg plant$ to $L(x)$
- Apply \exists -rule to $\exists eats.\neg plant$:
Create new node y and new edge (x, y)
 $L(y) = \{\neg plant\}$; $L((x, y)) = eats$

Example 2

- Apply \forall -rule to $\forall \text{eats.}(plant \sqcup \text{dairy})$ in $L(x)$ and $L((x, y)) = \text{eats}$:
Add $plant \sqcup \text{dairy}$ to $L(y)$

Example 2

- Apply \forall -rule to $\forall \text{eats} . (\text{plant} \sqcup \text{dairy})$ in $L(x)$ and $L((x, y)) = \text{eats}$:
Add $\text{plant} \sqcup \text{dairy}$ to $L(y)$
- Apply \sqcup -rule to $\text{plant} \sqcup \text{dairy}$ in $L(y)$:
Add plant to $L(y)$: Clash
Go back and add dairy to $L(y)$

Example 2

- Apply \forall -rule to $\forall \text{eats} . (\text{plant} \sqcup \text{dairy})$ in $L(x)$ and $L((x, y)) = \text{eats}$:
Add $\text{plant} \sqcup \text{dairy}$ to $L(y)$
- Apply \sqcup -rule to $\text{plant} \sqcup \text{dairy}$ in $L(y)$:
Add plant to $L(y)$: Clash
Go back and add dairy to $L(y)$
- Conclusion
 - No rules are applicable, so T is fully expanded
 - So $\text{vegetarian} \sqcap \neg \text{vegan}$ is satisfiable
 - So $\text{vegetarian} \not\sqsubseteq \text{vegan}$

The Brachman&Levesque DL and \mathcal{ALC}

Constructor	B&L	\mathcal{ALC}
Conj.	(AND $A B$)	$A \sqcap B$
Univ. quant.	(ALL $R C$)	$\forall R.C$
Exist. quant.		$\exists R.C$
Unqual. exist. quant.	(EXISTS 1 R)	$\exists R.\top$
Number restriction	(EXISTS $n R$)	
Role filler	(FILLS $R a$)	
Assertion	$a \rightarrow C$	$C(a)$

- \mathcal{FL}^- consists of Conj., Univ. quant., and Unqual. exist. quant.
- The B&L DL is slightly more general than \mathcal{FL}^- .
- \mathcal{ALC} is \mathcal{FL}^- plus \top , \perp , and general negation.
- The extension to \mathcal{ALC} for a role filler would use $\forall R.\{a\}$.

References

- Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, Peter Patel-Schneider (ed.): The Description Logic Handbook
- <http://www.inf.unibz.it/~franconi/dl/course/>
- <http://www.dl.kr.org>