Description Logics: 

\textit{ALC}
Outline

Topics:

1. Introduction
2. The description logic $ALC$
3. Extensions to $ALC$
4. A tableau algorithm for $ALC$
Introduction

Description logics

• A DL is a formalism for expressing concepts, their attributes (or associated roles), and the relationships between them.

• Can be regarded as providing a KR system based on a structured representation of knowledge.

Predecessors of DLs

• Semantic networks of the 70s

• Frame-based systems
Why Description Logics?

Prototypical AI case:

- Scientific (logical) vs engineering approach
- **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
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.

However, unlike in FOL, concepts need not be atomic and can have semantic relationships to each other and structure.
An KB in a DL contains two parts:

- Define terminology: \textit{TBox}
- Give assertions: \textit{ABox}
DL Knowledge Bases

An KB in a DL contains two parts:

- Define terminology: $TBox$
- Give assertions: $ABox$

Main components of the TBox:

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

ABox: Assertions that individuals satisfy certain concepts.
- E.g. $MWD(Mary)$
DL Knowledge Bases

An KB in a DL contains two parts:

- Define terminology: $TBox$
- Give assertions: $ABox$

Main components of the TBox:

- Concepts: classes of individuals
  - E.g. $Mother$
- Roles: binary relations between individuals
  - E.g. $\forall ParentOf. \neg Female$
DL Knowledge Bases

An KB in a DL contains two parts:

- Define terminology: *TBox*
- Give assertions: *ABox*

Main components of the TBox:

- Concepts: classes of individuals
  - E.g. *Mother*
- Roles: binary relations between individuals
  - E.g. ∀ParentOf.¬Female
- Complex concepts using constructors
  - E.g. *Mother* ⊓ ∀ParentOf.¬Female
DL Knowledge Bases

An KB in a DL contains two parts:

- Define terminology: \( TBox \)
- Give assertions: \( ABox \)

Main components of the TBox:

- Concepts: classes of individuals
  - E.g. \( Mother \)
- Roles: binary relations between individuals
  - E.g. \( \forall ParentOf \cdot \neg Female \)
- Complex concepts using constructors
  - E.g. \( Mother \sqcap \forall ParentOf \cdot \neg Female \)
- Assertions concerning complex concepts
  - E.g. \( MWD \models Mother \sqcap \forall ParentOf \cdot \neg Female \)
DL Knowledge Bases

An KB in a DL contains two parts:

- Define terminology: \( TBox \)
- Give assertions: \( ABox \)

Main components of the TBox:

- Concepts: classes of individuals
  - E.g. \( Mother \)
- Roles: binary relations between individuals
  - E.g. \( \forall ParentOf . \neg Female \)
- Complex concepts using constructors
  - E.g. \( Mother \sqcap \forall ParentOf . \neg Female \)
- Assertions concerning complex concepts
  - E.g. \( MWD \equiv Mother \sqcap \forall ParentOf . \neg Female \)

\( ABox \): Assertions that individuals satisfy certain concepts.

- E.g. \( MWD(Mary) \)
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.:

- Medical informatics
  - SnoMed, Galen
- Configuration of systems
- Semantic Web
  - Next generation web
  - OWL: W3C recommendation.
  - Comes in three flavours:
    - OWL Full, OWL DL, OWL Lite
- Enhanced database systems
  - DL-Lite
The Logic $\mathcal{ALC}$

An $\mathcal{ALC}$ KB contains two parts:

- Define terminology: TBox
- Give assertions: ABox

Main components of the TBox:

- Concepts: classes of individuals
- Roles: 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 $\mathcal{ALC}$: Concepts

Logical symbols:

- Propositional constructors: $\sqcap$, $\sqcup$, $\neg$
- Other restrictions: $\forall$, $\exists$
- $\top$, $\bot$
The Logic $\mathcal{ALC}$: Concepts

Logical symbols:
- Propositional constructors: $\sqcap, \sqcup, \neg$
- Other restrictions: $\forall, \exists$
- $\top, \bot$

Nonlogical symbols:
- Concept names
- Role names
The Logic $\mathcal{ALC}$: Concepts

Logical symbols:
- Propositional constructors: $\cap$, $\sqcup$, $\neg$
- Other restrictions: $\forall$, $\exists$
- $\top$, $\bot$

Nonlogical symbols:
- Concept names
- Role names

Concept construction
- Let $C$ and $D$ be concepts and $R$ a role.
- $\neg C$, $C \cap D$, $C \sqcup D$ are concepts.
- $\forall R.C$, $\exists R.C$ are concepts.
The Logic $\mathcal{ALC}$: Knowledge Bases

Axioms in the Tbox:

- Subsumption: $C \sqsubseteq D$ where $C$ and $D$ are concepts
- Equivalence axioms: $C \equiv D$ where $C$ and $D$ are concepts
The Logic $\mathcal{ALC}$: Knowledge Bases

Axioms in the Tbox:
- Subsumption: $C \sqsubseteq D$ where $C$ and $D$ are concepts
- Equivalence axioms: $C \equiv D$ 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 in the Tbox:

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

- $\text{Person} \sqsubseteq \text{Animal} \cap \text{Biped}$
- $\text{Woman} \models \text{Person} \cap \text{Female}$
- $\text{Mother} \models \text{Woman} \cap \exists \text{ParentOf} \cdot \text{Person}$
- $\text{Parent} \models \text{Mother} \sqcup \text{Father}$
- $\text{Man} \models \text{Person} \cap \neg \text{Woman}$
- $\text{MotherWithoutDaughter} \models \text{Mother} \cap \forall \text{ParentOf} \cdot \neg \text{Female}$
- $\text{GrandMother} \models \text{Woman} \cap \exists \text{ParentOf} \cdot \text{Parent}$

ABox:

- $\text{GrandMother}(\text{Sally})$
- $(\text{Person} \cap \text{Male})(\text{John})$
Formal Semantics for Concepts and Names

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

- An interpretation is a pair $I = \langle D, .I \rangle$
  - Domain $D$: non-empty set of objects
  - Interpretation function $I$: Maps structures into the domain.
  - $I$ maps every concept name $A$ to a subset $A_I \subseteq D_I$
  - $I$ maps every role name $R$ to a binary relation $R_I \subseteq D_I \times D_I$
  - $I$ maps individual names $a$ to elements of $D_I$: $a_I \in D_I$
  - $\top_I = D_I$ and $\bot_I = \emptyset$
Formal Semantics for Concepts and Names

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

An interpretation is a pair $I = \langle D, .^I \rangle$

- Domain $D$: non-empty set of objects
- Interpretation function $.^I$: Maps structures into the domain.
Semantically, a DL can be seen as a fragment of FOL.

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

- Domain $D$: non-empty set of objects
- Interpretation function $^I$: Maps structures into the domain.

Then:

- $^I$ maps every concept name $A$ to a subset $A^I \subseteq D^I$
- $^I$ maps every role name $R$ to a binary relation $R^I \subseteq D^I \times D^I$
- $^I$ maps individual names $a$ to elements of $D^I: a^I \in D^I$
- $\top^I = D^I$ and $\bot^I = \emptyset$. 
Semantics for Complex Concepts

Assume $C^I$, $D^I$, $R^I$ are given

- $(\neg C)^I = D^I \setminus C^I$
- $(C \cap D)^I = C^I \cap D^I$
- $(C \cup D)^I = C^I \cup D^I$
- $(\forall R.C)^I = \{x \mid y \in C^I \text{ for every } y \text{ s.t. } (x, y) \in R^I\}$
- $(\exists R.C)^I = \{x \mid y \in C^I \text{ for some } y \text{ s.t. } (x, y) \in R^I\}$
Semantics for Axioms and Assertions

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

• $C \sqsubseteq D$ is true in $I$ iff $C^I \subseteq D^I$
• $C \models D$ is true in $I$ iff $C^I = D^I$
• $C(a)$ is true in $I$ iff $a^I \in C^I$
• $R(a, b)$ is true in $I$ iff $(a^I, b^I) \in R^I$
Reasoning in \textit{ALC}

- Sentences: Axioms or assertions
- \( I \) is a \textit{model} for a sentence \( S \) iff \( S \) is true in \( I \)
- \( I \) 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 \textit{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 $\text{ALC}$

$K$ 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:** $K \models C \subseteq D$
- **Equivalence checking:** $K \models C \equiv D$
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: $K \models C \equiv D$
- Consistency checking: $K \not\models \top \sqsubseteq \bot$
- Concept satisfiability: $K \not\models C \sqsubseteq \bot$
- Disjoint concepts: $K \models C \sqcap D \sqsubseteq \bot$
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: $K \models C \equiv D$
- Consistency checking: $K \not\models \top \sqsubseteq \bot$
- Concept satisfiability: $K \not\models C \sqsubseteq \bot$
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: $K \models C \equiv D$
- Consistency checking: $K \not\models \top \sqsubseteq \bot$
- Concept satisfiability: $K \not\models C \sqsubseteq \bot$
- Disjoint concepts: $K \models C \cap D \sqsubseteq \bot$
Reduction to Consistency Checking

Let $b$ be a new individual

- Instance checking:
  \[ K \models C(a) \text{ iff } K \cup \{ \neg C(a) \} \models \top \sqsubseteq \bot \]
Reduction to Consistency Checking

Let $b$ be a new individual

- **Instance checking:**
  \[ K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \bot \]

- **Subsumption checking:**
  \[ K \models C \sqsubseteq D \text{ iff } K \cup \{(C \cap \neg D)(b)\} \models \top \sqsubseteq \bot \]

- **Equivalence checking:**
  \[ K \models C = D \text{ iff } K \cup \{(C \cap \neg D)(b), (\neg C \cap D)(b)\} \models \top \sqsubseteq \bot \]

- **Concept satisfiability:**
  \[ K \not\models C \sqcup \bot \text{ iff } K \cup \{C(b)\} \not\models \top \sqsubseteq \bot \]

- **Disjoint concepts:**
  \[ K \models C \sqcup D \sqsubseteq \bot \text{ iff } K \cup \{(C \cap D)(b)\} \models \top \sqsubseteq \bot \]
Reduction to Consistency Checking

Let $b$ be a new individual

- **Instance checking:**
  \[ K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \bot \]

- **Subsumption checking:**
  \[ K \models C \sqsubseteq D \text{ iff } K \cup \{(C \cap \neg D)(b)\} \models \top \sqsubseteq \bot \]

- **Equivalence checking:**
  \[ K \models C \equiv D \text{ iff } K \cup \{(C \cap \neg D)(b), (\neg C \cap D)(b)\} \models \top \sqsubseteq \bot \]
Reduction to Consistency Checking

Let $b$ be a new individual

- **Instance checking:**
  \[ K \models C(a) \text{ iff } K \cup \{\neg C(a)\} \models \top \sqsubseteq \bot \]

- **Subsumption checking:**
  \[ K \models C \sqsubseteq D \text{ iff } K \cup \{(C \sqcap \neg D)(b)\} \models \top \sqsubseteq \bot \]

- **Equivalence checking:**
  \[ K \models C \equiv D \text{ iff } K \cup \{(C \sqcap \neg D)(b), (\neg C \sqcap D)(b)\} \models \top \sqsubseteq \bot \]

- **Concept satisfiability:**
  \[ K \not\models C \sqsubseteq \bot \text{ iff } K \cup \{C(b)\} \not\models \top \sqsubseteq \bot \]
Reduction to Consistency Checking

Let $b$ be a new individual

- **Instance checking:**
  \[ K \models C(a) \iff K \cup \{\neg C(a)\} \models T \sqsubseteq \bot \]

- **Subsumption checking:**
  \[ K \models C \sqsubseteq D \iff K \cup \{(C \cap \neg D)(b)\} \models T \sqsubseteq \bot \]

- **Equivalence checking:**
  \[ K \models C \equiv D \iff K \cup \{(C \cap \neg D)(b), (\neg C \cap D)(b)\} \models T \sqsubseteq \bot \]

- **Concept satisfiability:**
  \[ K \not\models C \sqsubseteq \bot \iff K \cup \{C(b)\} \not\models T \sqsubseteq \bot \]

- **Disjoint concepts:**
  \[ K \models C \cap D \sqsubseteq \bot \iff K \cup \{(C \cap D)(b)\} \models T \sqsubseteq \bot \]
Extensions to $\mathit{ALC}$

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.
- Functional roles: $R \in \mathbf{F}$ where $R$ is a role
- Transitive roles: $R \in R^+$ where $R$ is a role
Extensions to $\mathcal{ALC}$: Examples

- $ParentWithManySons \doteq (\geq 3\text{ParentOf}.\text{Male})$
- $\exists \text{ParentOf}^-.\text{Citizen} \sqsubseteq \text{Citizen}$
- $\text{ParentOf} \sqsubseteq \text{AncestorOf}$
- $\text{AncestorOf} \in R^+$
- $\text{AgeOf} \in F$
Extensions to $\text{ALC}$: Semantics

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

Assume *unfoldable terminologies*:

- Only axioms of the form $A \sqsubseteq C$ and $A \sqsupseteq C$ where $A$ is a concept name.
- For each concept name $A$, at most one axiom of the form $A \sqsubseteq C$ or $A \sqsupseteq C$.
- Axioms are acyclic:
  - $A \sqsubseteq C$ or $A \sqsupseteq C$ *directly uses* a concept name $A_1$ iff $A_1$ occurs in $C$.
  - $A \sqsubseteq C$ or $A \sqsupseteq C$ *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 $A \sqsupseteq C$ 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 $P \models Q$
  - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \models B \sqcap C$ 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 $P \models Q$
  - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \models B \sqcap C$ 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 $C \models D$ is in the KB, replace $C$ by $D$ in the query.
  - Recall that for $C \models D$ 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 $C \sqsubseteq D$ is in the KB, replace $C$ by $D$ in the query.
  - Recall that for $C \sqsubseteq D$ 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 $x \in C^I$
- Each edge $(x, y)$ represents a pair occurring in the interpretation of a role, labelled with the role name
  - $R = L((x, y))$ implies $(x, y) \in R^I$
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 $D$, $\bot \in L(y)$ or $\{D, \neg D\} \subseteq L(y)$.
Expansion Rules

(∩-rule) If \((C_1 \cap C_2) \in L(x)\) and \(\{C_1, C_2\} \not\subseteq L(x)\) then:
Add \(C_1\) and \(C_2\) to \(L(x)\).
Expansion Rules

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

(\sqcup\text{-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

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

(\sqcup\text{-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\text{-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

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

(\cup\text{-rule}) If \((C_1 \cup C_2) \in L(x)\) 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\text{-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\text{-rule}) If \(\forall R.C \in L(x)\) and there is some \(y\) s.t.
\(L((x, y)) = R\) and \(C \not\in L(y)\) 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:
  - $D^I = \{x \mid x \text{ is a node in } T\}$.
  - $A^I = \{x \in D^I \mid A \in L(x)\}$ 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 $\cap$, $\cup$, 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:

- \textit{vegan} \equiv \textit{person} \sqcap \forall \text{eats.plant}
- \textit{vegetarian} \equiv \textit{person} \sqcap \forall \text{eats.}(\text{plants} \sqcup \text{dairy})

Query: \textit{vegan} \sqsubseteq \textit{vegetarian}

Convert to:

- \textit{vegan} \sqcap \neg \textit{vegetarian} \text{ is unsatisfiable ?}
Example 1

- Unfold and normalise \( \text{vegan} \sqcap \neg \text{vegetarian} \):
  \[
  \text{person} \sqcap \forall \text{eats. plant} \sqcap (\neg \text{person} \sqcup \exists \text{eats.} (\neg \text{plant} \sqcap \neg \text{dairy}))
  \]
Example 1

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

- Initialise \( T \) to \( L(x) \) to contain:
  \( \text{person} \sqcap \forall \text{eats.} \text{plant} \sqcap (\neg \text{person} \sqcup \exists \text{eats.}(\neg \text{plant} \sqcap \neg \text{dairy})) \)
Example 1

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

- Initialise $T$ to $L(x)$ to contain:
  
  $\text{person} \sqcap \forall \text{eats}. \text{plant} \sqcap (\neg \text{person} \sqcup \exists \text{eats}. (\neg \text{plant} \sqcap \neg \text{dairy}))$

- Apply $\sqcap$-rule and add to $L(x)$:
  
  $\{\text{person}, \forall \text{eats}. \text{plant}, \neg \text{person} \sqcup \exists \text{eats}. (\neg \text{plant} \sqcap \neg \text{dairy})\}$
Example 1

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

- Both applications of the $\sqcup$-rule lead to clashes
- So $\text{vegan} \sqcup \neg\text{vegetarian}$ is unsatisfiable
- So $\text{vegan} \sqsubseteq \text{vegetarian}$
Example 1

- Apply $\cap$-rule to $\neg plant \cap \neg dairy$ 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: \textit{vegetarian} \sqsubseteq \textit{vegan}
- Convert to: \textit{vegetarian} \sqcap \neg \textit{vegan} is satisfiable?
- Unfold and normalise \textit{vegetarian} \sqcap \neg \textit{vegan}:
  \begin{align*}
  \text{person} \sqcap \forall \text{eats} . (\text{plant} \sqcup \text{dairy}) \sqcap (\neg \text{person} \sqcup \exists \text{eats} . \neg \text{plant})
  \end{align*}
- Initialise $T$ to $L(x)$ to contain:
  \begin{align*}
  \{ \text{person} \sqcap \forall \text{eats} . (\text{plant} \sqcup \text{dairy}) \sqcap (\neg \text{person} \sqcup \exists \text{eats} . \neg \text{plant}) \}
  \end{align*}
Example 2

- Apply $\sqcap$-rule and add to $L(x)$:
  \[
  \{ \text{person}, \forall \text{eats.} (\text{plant} \sqcup \text{dairy}), \neg \text{person} \sqcup \exists \text{eats.} \neg \text{plant} \}\n  \]
Example 2

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

- Apply $\sqcup$-rule to $\neg \textit{person }\sqcup \exists \textit{eats.} \neg \textit{plant}$:
  Add $\neg \textit{person}$ to $L(x)$: Clash
  Go back and add $\exists \textit{eats.} \neg \textit{plant}$ to $L(x)$
Example 2

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

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

- Apply $\exists$-rule to $\exists \text{eats.}\neg \text{plant}$:
  Create new node $y$ and new edge $(x, y)$
  $L(y) = \{\neg \text{plant}\}; L((x, y)) = \text{eats}$
Example 2

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

- Conclusion

  - No rules are applicable, so $T$ is fully expanded

  - So $\text{vegetarian} \sqcup \neg\text{vegan}$ is satisfiable

  - So $\text{vegetarian} \not\sqsubseteq \text{vegan}$
Example 2

- Apply $\forall$-rule to $\forall e \text{ats.} (\text{plant} \sqcup \text{dairy})$ 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 $\text{plant}$ to $L(y)$: Clash
  Go back and add $\text{dairy}$ to $L(y)$

- Conclusion
  No rules are applicable, so $T$ is fully expanded
  So vegetarian $\sqcap \neg$ vegan is satisfiable
  So vegetarian $\not\sqsubseteq$ vegan
Example 2

- Apply $\forall$-rule to $\forall x. (\text{plant} \sqcup \text{dairy})$ 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 $\text{plant}$ to $L(y)$: Clash
  Go back and add $\text{dairy}$ to $L(y)$
- Conclusion
  - No rules are applicable, so $T$ is fully expanded
  - So $\text{vegetarian} \sqcap \neg \text{vegan}$ is satisfiable
  - So $\text{vegetarian} \nsubseteq \text{vegan}$
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