Description Logics: $\mathcal{ALC}$
Outline

Topics:

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

- A DL is a formalism for expressing concepts, their attributes (or associated roles), and the relationships between them.
  - E.g. Person could be a concept and a role could be ParentOf.
- 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

A KB in a DL contains two parts:

• Define terminology: \textit{TBox}
  
  • Like definitions, or partial definitions
  
  • E.g. \textit{MWD} \models \textit{Mother} \sqcap \forall \textit{ParentOf}.\neg \textit{Female}
  
  \textit{Mother} \sqsubseteq \textit{Female}

• Give assertions: \textit{ABox}
  
  • E.g. \textit{MWD}(sue).
DL Knowledge Bases: TBox

Main components of the TBox:

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

- **Concepts**: classes of individuals
  - E.g. *Mother*
- **Roles**: binary relations between individuals
  - E.g. *ParentOf*
Main components of the TBox:

- **Concepts**: classes of individuals
  - E.g. *Mother*
- **Roles**: binary relations between individuals
  - E.g. *ParentOf*
- **Complex concepts** using constructors
  - E.g. \( \forall \text{ParentOf} . \neg \text{Female} \)
    
    \begin{align*}
    \text{Mother} \sqcap \forall \text{ParentOf} . \neg \text{Female} 
    \end{align*}
DL Knowledge Bases: TBox

Main components of the TBox:

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

- **Assertions** concerning complex concepts
  - E.g. $MWD \models Mother \sqcap \forall ParentOf. \neg Female$
    
    $Mother \sqsubseteq Female$
ABox: Assertions that individuals satisfy certain concepts and roles.

- Think of as a 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 CT, GALEN
  - \( \mathcal{EL} \)
- Semantic Web
  - OWL: W3C recommendation.
  - Comes in lots of flavours

⚠️ We’ll look at perhaps the most central DL, \( \mathcal{ALC} \).
The Logic $\mathcal{ALC}$

An $\mathcal{ALC}$ 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 $\mathcal{ALC}$

An $\mathcal{ALC}$ 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: $\forall$, $\exists$
  - Note: These are different from quantifiers as seen in FOL
- $\top$, $\bot$
The Logic $\mathcal{ALC}$: Language

Logical symbols:

• Propositional constructors: $\sqcap$, $\sqcup$, $\neg$
• Other restrictions: $\forall$, $\exists$
  • Note: These are different from quantifiers as seen in FOL
• $\top$, $\bot$

Nonlogical symbols:

• Concept names
• Role names
The Logic $\mathcal{ALC}$: Language

Logical symbols:
- Propositional constructors: $\sqcap, \sqcup, \neg$
- Other restrictions: $\forall, \exists$
  - Note: These are different from quantifiers as seen in FOL
- $\top, \bot$

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.

- $\neg C$ stands for the concept of things that are not $C$.

- $C \sqcap D$ is the concept of things that are both $C$ and $D$.

  - E.g. $\text{Female} \sqcap \text{Human}$

- $C \sqcup D$ is the concept of things that are either $C$ or $D$ or both.

  - E.g. $\text{Male} \sqcup \text{Female}$

- $\forall R. C$ is the concept of things such that all things that are $R$ related to it are $C$'s.

  - E.g. $\forall \text{ParentOf}. \text{Female}$: 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 $C$.

  - $\exists \text{ParentOf}. \text{Female}$: things with a female child
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. $\forall \text{ParentOf} . \text{Female}$: 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 $C$.
  - $\exists \text{ParentOf} . \text{Female}$: things with a female child
The Logic $ALC$: 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. $\text{Female} \sqcap \text{Human}$
- $C \sqcup D$ is the concept of things that are either $C$ or $D$ or both.
  - E.g. $\text{Male} \sqcup \text{Female}$
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 $\text{ALC}$: 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. $\text{Female} \sqcap \text{Human}$
- $C \sqcup D$ is the concept of things that are either $C$ or $D$ or both.
  - E.g. $\text{Male} \sqcup \text{Female}$
- $\forall R.C$ is the concept of things such that all things that are $R$ related to it are $C$’s.
  - E.g. $\forall \text{ParentOf}.\text{Female}$: 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 $C$.
  - $\exists \text{ParentOf}.\text{Female}$: 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: $C \equiv D$ 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: $C \equiv D$ 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 (assertions) 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:

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

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 $I = ⟨\Delta, .⟩$:
  - Domain $\Delta$: non-empty set of objects
  - Interpretation function $I$: Maps structures into the domain.
- Recall, Brachman and Levesque write this as $I = ⟨D, I⟩$.
- Then:
  - $I$ maps every concept name $A$ to a subset $A \subseteq \Delta$
  - $I$ maps every role name $R$ to a binary relation $R \subseteq \Delta \times \Delta$
  - $I$ maps individual names $a$ to elements of $\Delta$: $a \in \Delta$
- $\top_I = \Delta$ 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 $\mathcal{I} = \langle \Delta, .^\mathcal{I} \rangle$

- Domain $\Delta$: non-empty set of objects
- Interpretation function $^\mathcal{I}$: 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 $\mathcal{I} = \langle \Delta, .^\mathcal{I} \rangle$

- Domain $\Delta$: 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 $A^{\cdot^\mathcal{I}} \subseteq \Delta$
- $\cdot^\mathcal{I}$ maps every role name $R$ to a binary relation $R^{\cdot^\mathcal{I}} \subseteq \Delta \times \Delta$
- $\cdot^\mathcal{I}$ maps individual names $a$ to elements of $\Delta$ : $a^{\cdot^\mathcal{I}} \in \Delta$
- $\top^\mathcal{I} = \Delta$ and $\bot^\mathcal{I} = \emptyset$. 
Semantics for Complex Concepts

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

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

Assume $C$, $D$ are concepts, $R$ is a role, $a$ and $b$ are individual names.
Let $\mathcal{I} = (\Delta, .^\mathcal{I})$ be an interpretation.

- $C \sqsubseteq D$ is true in $\mathcal{I}$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$
- $C \models D$ 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 $ALC$

$K$ 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: $K \models C \equiv D$
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 (satisfiability) checking: $K \not\models \top \sqsubseteq \bot$
- Concept satisfiability: $K \not\models C \sqsubseteq \bot$
- Disjoint concepts: $K \models C \cap D \sqsubseteq \bot$
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 (satisfiability) 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 (satisfiability) 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:
  \[ K \models C(a) \text{ if and only if } K \cup \{\neg C(a)\} \models \top \sqsubseteq \bot \]

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

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

- Concept satisfiability:
  \[ K \not\models C \sqsubseteq \bot \text{ if and only if } K \cup \{C(b)\} \not\models \top \sqsubseteq \bot \]

- Disjoint concepts:
  \[ K \models C \sqcap D \sqsubseteq \bot \text{ if and only if } K \cup \{C \cap D(b)\} \models \top \sqsubseteq \bot \]
Reduction to Consistency Checking

Let $b$ be a new individual

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

- **Subsumption checking:**
  $K \models C \sqsubseteq D$ iff $K \cup \{(C \sqcap \neg D)(b)\} \models \top \sqsubseteq \bot$
Reduction to Consistency Checking

Let $b$ be a new individual

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

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

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

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

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

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

- **Disjoint concepts:**
  \[ K \models C \sqcap D \sqsubseteq \bot \iff K \cup \{(C \sqcap D)(b)\} \models \top \sqsubseteq \bot \]
Aside: Extensions to $\mathcal{ALC}$

- There are many other possible constructors that can be added

  - Extended concepts
    - Number restrictions: $(\leq nR.C)$ and $(\geq nR.C)$
      - E.g. `ParentWithManySons` = $(\geq 3 \text{ParentOf}.\text{Male})$
    - Blended `Wine` ⊑ $(\geq 2 \text{GrapeTypeOf}.\text{Grape})$
  
  - Nominals: Allow individuals in the TBox
    - E.g. `IndianCitizen` = `Person` ⊓ ∃ `CitizenOf`.{`India`
Aside: Extensions to $ALC$

There are many other possible constructors that can be added. For example:

Extended concepts

- Number restrictions: $(\leq nR.C)$ and $(\geq nR.C)$
Aside: Extensions to $\mathcal{ALC}$

There are many other possible constructors that can be added. For example:

Extended concepts

- Number restrictions: $(\leq nR.C)$ and $(\geq nR.C)$
  
  E.g. $\text{ParentWithManySons} \equiv (\geq 3\text{ParentOf}.\text{Male})$

  $\text{BlendedWine} \sqsubseteq (\geq 2\text{GrapeTypeOf}.\text{Grape})$
Aside: Extensions to $\mathcal{ALC}$

There are many other possible constructors that can be added. For example:

Extended concepts

- Number restrictions: $(\leq nR.C)$ and $(\geq nR.C)$
  
  E.g. $ParentWithManySons \equiv (\geq 3ParentOf\cdot Male)$

  $BlendedWine \sqsubseteq (\geq 2GrapeTypeOf\cdot Grape)$

- Nominals: Allow individuals in the TBox
Aside: Extensions to $\mathcal{ALC}$

There are many other possible constructors that can be added for example:

Extended concepts

- Number restrictions: $(\leq nR.C)$ and $(\geq nR.C)$
  E.g. $\text{ParentWithManySons} \equiv (\geq 3\text{ParentOf.Male})$
    $\text{BlendedWine} \sqsubseteq (\geq 2\text{GrapeTypeOf.Grape})$

- Nominals: Allow individuals in the TBox
  E.g. $\text{IndianCitizen} \equiv \text{Person} \sqcap \exists \text{CitizenOf.}\{\text{India}\}$
Extensions to $\mathcal{ALC}$

Role operators

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

E.g. $\exists$ ParentOf $\subseteq$ Citizen. GradCourse $\subseteq ∀$ teaches $\subseteq$ Professor.
Extensions to $\mathcal{ALC}$

Role operators

- Inverse roles: $R^-$ where $R$ is a role
  
  E.g. $\exists ParentOf^- . Citizen \sqsubseteq Citizen$
  
  $GradCourse \sqsubseteq \forall teaches^- . Professor$
Extensions to $\mathcal{ALC}$

Role operators

- Inverse roles: $R^-$ where $R$ is a role
  
  E.g. $\exists ParentOf^- . Citizen \sqsubseteq Citizen$
  
  $GradCourse \sqsubseteq \forall teaches^- . Professor$

Role axioms

- Role hierarchy: $R \sqsubseteq S$ where $R$ and $S$ are roles
  
  So far have just used $\sqsubseteq$ for concepts.
Extensions to \textit{ALC}

Role operators

- Inverse roles: $R^-$ where $R$ is a role
  
  E.g. $\exists ParentOf^- . Citizen \sqsubseteq Citizen$
  
  $GradCourse \sqsubseteq \forall teaches^- . Professor$

Role axioms

- Role hierarchy: $R \sqsubseteq S$ where $R$ and $S$ are roles
  
  $\therefore$ So far have just used $\sqsubseteq$ for concepts.
  
  E.g. $ParentOf \sqsubseteq AncestorOf$
Extensions to $\mathcal{ALC}$

Role operators

- Inverse roles: $R^-$ where $R$ is a role
  
  E.g. $\exists ParentOf^- . Citizen \sqsubseteq Citizen$
  
  $GradCourse \sqsubseteq \forall teaches^- . Professor$

Role axioms

- Role hierarchy: $R \sqsubseteq S$ where $R$ and $S$ are roles
  
  So far have just used $\sqsubseteq$ for concepts.
  
  E.g. $ParentOf \sqsubseteq AncestorOf$

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

Role operators

- **Inverse roles:** $R^{-}$ where $R$ is a role
  
  E.g. $\exists ParentOf^{-}.Citizen \sqsubseteq Citizen$
  
  $GradCourse \sqsubseteq \forall teaches^{-}.Professor$

Role axioms

- **Role hierarchy:** $R \sqsubseteq S$ where $R$ and $S$ are roles
  
  So far have just used $\sqsubseteq$ for concepts.
  
  E.g. $ParentOf \sqsubseteq AncestorOf$

- **Transitive roles:** $R \in R^{+}$ where $R$ is a role
  
  E.g. $AncestorOf \in R^{+}$

And lots of others ...
Extensions to \(\mathcal{ALC}\): 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 \(S\).
- \(R \in R^+\) 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 $\mathcal{ALC}$

Goal: Show $KB \models A \sqsubseteq B$ by showing $KB \cup \{A \sqcap \neg B\}$ unsatisfiable.
A Tableau Algorithm for $\mathcal{ALC}$

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

Assume an *unfoldable terminology*:

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

Show $KB \models A \subseteq B$ by showing $KB \cup \{A \cap \neg B\}$ 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 $P \vdash Q$
  - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \sqsubseteq B \sqcap C$ where $C$ is a new concept name.

If the query is $A \sqsubseteq B$, first convert to a normal form:

- **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 $P \sqsupseteq Q$
  - This can be done by replacing any axiom of the form $A \sqsubseteq B$ by $A \sqsupseteq B \sqcap C$ where $C$ is a new concept name.

If the query is $A \sqsubseteq B$, first convert to a normal form:

- *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.
**Unfolding**

**To Unfold:**
Expand every concept name occurring in the (negated) query.

- I.e. if concept $C$ appears in the query and $C \equiv D$ is in the KB, replace $C$ by $D$ in the query.

- Recall that for $C \equiv D$ in the KB, $C$ is a concept name and $D$ is an arbitrary $\mathcal{ALC}$ concept expression.

- As well, $C$ is guaranteed to not appear in $D$ or in any later substitutions.
Negation normal form

Negation normal form:
Move negation in so that it 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 $D$, $ot \in L(y)$ or $\{D, \neg D\} \subseteq L(y)$.
- If $T$ can’t be expanded without producing a clash, the concept is unsatisfiable.
Expansion Rules

(⊓-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

(⊓-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 \(L(x)\).

(⊔-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

(\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 \(L((x, y)) = R\).

(\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:
  - $\Delta = \{x \mid x \text{ is a node in } T\}$.
  - $A^I = \{x \in \Delta \mid A \in L(x)\}$ 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:

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

Query: $\text{vegan} \subseteq \text{vegetarian}$

Convert to:

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

- Unfold and normalise $\text{vegan} \sqcap \neg \text{vegetarian}$:
  $\text{person} \sqcap \forall \text{eats}. \text{plant} \sqcap \neg \text{person} \sqcup \exists \text{eats.} \left( \neg \text{plant} \sqcap \neg \text{dairy} \right)$
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 $\square$-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 $\sqcap$-rule to $\neg plant \sqcap \neg dairy$ in $L(y)$:
  Add $\{\neg plant, \neg dairy\}$ to $L(y)$: Clash
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 $\cup$-rule lead to clashes
  • So $vegan \sqcup \neg vegetarian$ 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}:
  \textit{person} \sqcap \forall \text{eats.}(\textit{plant} \sqcup \textit{dairy}) \sqcap (\neg \textit{person} \sqcup \exists \text{eats.}\neg \textit{plant})

• Initialise \( T \) to \( L(x) \) to contain:
  \{ \textit{person} \sqcap \forall \text{eats.}(\textit{plant} \sqcup \textit{dairy}) \sqcap (\neg \textit{person} \sqcup \exists \text{eats.}\neg \textit{plant}) \}
Example 2

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

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

• 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 \( L(x) \)
Example 2

• Apply $\cap$-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 $\forall e x (plan t \sqcup d a i r y)$ in $L(x)$ and $L((x, y)) = eats$:
  Add $plant \sqcup dairy$ to $L(y)$
Example 2

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

- Apply $\sqcup$-rule to $plant \sqcup dairy$ in $L(y)$:
  Add $plant$ to $L(y)$: Clash
  Go back and add $dairy$ to $L(y)$
Example 2

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

- Apply $\sqcup$-rule to $plant \sqcup dairy$ 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 $vegetarian \sqcap \neg \text{vegan}$ is satisfiable
  - So $vegetarian \not\sqsubseteq \text{vegan}$
### The Brachman&Levesque DL and $\mathcal{ALC}$

<table>
<thead>
<tr>
<th>Constructor</th>
<th>B&amp;L</th>
<th>$\mathcal{ALC}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conj.</td>
<td>(AND $A$ $B$)</td>
<td>$A \sqcap B$</td>
</tr>
<tr>
<td>Univ. quant.</td>
<td>(ALL $R$ $C$)</td>
<td>$\forall R.C$</td>
</tr>
<tr>
<td>Exist. quant.</td>
<td></td>
<td>$\exists R.C$</td>
</tr>
<tr>
<td>Unqual. exist. quant.</td>
<td>(EXISTS 1 $R$)</td>
<td>$\exists R.\top$</td>
</tr>
<tr>
<td>Number restriction</td>
<td>(EXISTS $n$ $R$)</td>
<td></td>
</tr>
<tr>
<td>Role filler</td>
<td>(FILLS $R$ $a$)</td>
<td></td>
</tr>
<tr>
<td>Assertion</td>
<td>$a \rightarrow C$</td>
<td>$C(a)$</td>
</tr>
</tbody>
</table>

- $\mathcal{FL^-}$ 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 $\top$, $\bot$, and general negation.
- The extension to $\mathcal{ALC}$ for a role filler would use $\forall R.\{a\}$. 

References

• Franz Baader, Ian Horrocks, Carsten Lutz, Uli Sattler: An Introduction to Description Logic
• 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