Outline

1. Properties of $\mathcal{ALC}$
   - $\mathcal{ALC}$ and First-Order Logic
   - Bissimulation
   - Properties of $\mathcal{ALC}$
2. Reasoning over $\mathcal{ALC}$ concept expressions
   - Tableaux for concept satisfiability
3. Reasoning over $\mathcal{ALC}$ ontologies
   - Reasoning w.r.t. acyclic TBoxes
   - Reasoning w.r.t. arbitrary TBoxes

Outline

1. Properties of $\mathcal{ALC}$
   - $\mathcal{ALC}$ and First-Order Logic
   - Bissimulation
   - Properties of $\mathcal{ALC}$
2. Reasoning over $\mathcal{ALC}$ concept expressions
   - Tableaux for concept satisfiability
3. Reasoning over $\mathcal{ALC}$ ontologies
   - Reasoning w.r.t. acyclic TBoxes
   - Reasoning w.r.t. arbitrary TBoxes

Outline

Recall the definition of $\mathcal{ALC}$ - Concept language

Construct	Syntax	Example	Semantics
atomic concept	A	$Doctor$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	$hasChild$	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$Hum \sqcap Male$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\forall HasChild.Male$	$\{o \mid \forall o'.(o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
negation	$\neg C$	$\neg\forall hasChild.Male$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$

(C_1, C_2 denote arbitrary concepts and R an arbitrary role)

We make also use of the following abbreviations:

Construct	Stands for	
$\bot$	$A \sqcap \neg A$	(for some atomic concept A)
$\top$	$\neg \bot$	
$C_1 \sqcup C_2$	$\neg(\neg C_1 \sqcap \neg C_2)$	
$\exists R.C$	$\neg\forall R.\neg C$	

$\mathcal{ALC}$ ontology (or knowledge base)

Def.: $\mathcal{ALC}$ ontology

Is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox:

- The TBox is a set of inclusion assertions on $\mathcal{ALC}$ concepts: $C_1 \sqsubseteq C_2$
- The ABox is a set of membership assertions on individuals:
  - Membership assertions for concepts: $A(c)$
  - Membership assertions for roles: $P(c_1, c_2)$

Note: We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \sqsubseteq C_2, C_2 \sqsubseteq C_1$.

Example

$$
\begin{aligned}
TBox: \quad & Father \equiv Human \sqcap Male \sqcap \exists hasChild \\
& HappyFather \sqsubseteq Father \sqcap \forall hasChild.(Doctor \sqcup Lawyer \sqcup HappyPerson) \\
& HappyAnc \sqsubseteq \forall descendant.HappyFather \\
& Teacher \sqsubseteq \neg Doctor \sqcap \neg Lawyer
\end{aligned}
$$

$ABox: Teacher(mary), hasFather(mary, john), HappyAnc(john)$

From $\mathcal{ALC}$ to First-Order Logic

We have seen that $\mathcal{ALC}$ is a well-behaved fragment of function-free First-Order Logic with unary and binary predicates only (FOL$_{bin}$).

To translate an $\mathcal{ALC}$ TBox to FOL$_{bin}$ we proceed as follows:

1. Introduce: a unary predicate A(x) for each atomic concept A
   a binary predicate $P(x, y)$ for each atomic role P
2. Translate complex concepts as follows, using two translation functions t_x and t_y:

$$
\begin{aligned}
t_x(A) &= A(x) & \qquad t_y(A) &= A(y) \\
t_x(\neg C) &= \neg t_x(C) & t_y(\neg C) &= \neg t_y(C) \\
t_x(C \sqcap D) &= t_x(C) \wedge t_x(D) & t_y(C \sqcap D) &= t_y(C) \wedge t_y(D) \\
t_x(C \sqcup D) &= t_x(C) \vee t_x(D) & t_y(C \sqcup D) &= t_y(C) \vee t_y(D) \\
t_x(\exists P.C) &= \exists y.P(x, y) \wedge t_y(C) & t_y(\exists P.C) &= \exists x.P(y, x) \wedge t_x(C) \\
t_x(\forall P.C) &= \forall y.P(x, y) \rightarrow t_y(C) & t_y(\forall P.C) &= \forall x.P(y, x) \rightarrow t_x(C)
\end{aligned}
$$

3. Translate a TBox $\mathcal{T} = \bigcup_i \{C_i \sqsubseteq D_i\}$ as the FOL theory:
$$\Gamma_{\mathcal{T}} = \bigcup_i \{\forall x. t_x(C_i) \rightarrow t_x(D_i)\}$$
4. Translate an ABox $\mathcal{A} = \bigcup_i \{A_i(c_i)\} \cup \bigcup_j \{P_j(c'_j, c''_j)\}$ as the FOL theory:
$$\Gamma_{\mathcal{A}} = \bigcup_i \{A_i(c_i)\} \cup \bigcup_j \{P_j(c'_j, c''_j)\}$$

Via the translation to FOL_{bin}, there is a direct correspondence between DL reasoning services and FOL reasoning services:

C is satisfiable	iff	its translation $t_x(C)$ is satisfiable
C is satisfiable w.r.t. $\mathcal{T}$	iff	$\Gamma_{\mathcal{T}} \cup \{\exists x.t_x(C)\}$ is satisfiable
$\mathcal{T} \models_{\mathcal{ALC}} C \sqsubseteq D$	iff	$\Gamma_{\mathcal{T}} \models_{FOL} \forall x.(t_x(C) \rightarrow t_x(D))$
$C \sqsubseteq D$	iff	$\models_{FOL} t_x(C) \rightarrow t_x(D)$
$\top \sqsubseteq D$	iff	$\models_{FOL} t_x(D)$

(We use $\models_{FOL} \varphi$ to denote that φ is a valid FOL formula.)

From First-Order Logic to $\mathcal{ALC}$?

Question

Is it possible to define a transformation $\tau(\cdot)$ from FOL$_{bin}$ formulas to $\mathcal{ALC}$ concepts and roles such that the following is true?

$$\models_{FOL} \varphi \quad \text{implies} \quad \top \sqsubseteq \tau(\varphi)$$

- If yes, we should specify the transformation $\tau(\cdot)$.
- If not, we should provide a formal proof that $\tau(\cdot)$ does not exist.

Outline

Distinguishability of interpretations

Def.: Distinguishing between models

If $\mathcal{I}$ and $\mathcal{J}$ are two interpretations of a logic $\mathcal{L}$, then we say that $\mathcal{I}$ and $\mathcal{J}$ are distinguishable in $\mathcal{L}$ if there is a formula φ of the language of $\mathcal{L}$ such that

$$\mathcal{I} \models_{\mathcal{L}} \varphi \quad \text{and} \quad \mathcal{J} \not\models_{\mathcal{L}} \varphi$$

Example

- $\mathcal{I} = (\{a\}, \cdot^{\mathcal{I}})$ with $p^{\mathcal{I}} = \{a\}$ and $\mathcal{J} = (\{b\}, \cdot^{\mathcal{J}})$ with $p^{\mathcal{J}} = \{b\}$ are not distinguishable in first-order logic (all other predicates are interpreted as $\{\}$ in both)
- $\mathcal{I} = (\{a, b\}, \cdot^{\mathcal{I}})$ with $p^{\mathcal{I}} = \{a, b\}$ and $\mathcal{J} = (\{a, b\}, \cdot^{\mathcal{J}})$ with $p^{\mathcal{J}} = \{a\}$ are distinguishable in first-order logic

Proving non-equivalence:

To show that two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ with the same class of interpretations are not equivalent, it is enough to show that there are two interpretations $\mathcal{I}$ and $\mathcal{J}$ that are distinguishable in $\mathcal{L}_1$ and not distinguishable in $\mathcal{L}_2$.

The notion of bisimulation in description logics is intended to capture equivalence of objects and their properties.

Def.: Bisimulation

A bisimulation $\sim_{\mathcal{B}}$ between two $\mathcal{ALC}$ interpretations $\mathcal{I}$ and $\mathcal{J}$ is a relation in $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{J}}$ such that, for every pair of objects $o_1 \in \Delta^{\mathcal{I}}$ and $o_2 \in \Delta^{\mathcal{J}}$, if $o_1 \sim_{\mathcal{B}} o_2$, then the following hold:

- for every atomic concept $A : o_1 \in A^{\mathcal{I}}$ if and only if $o_2 \in A^{\mathcal{J}}$ (local condition);
- for every atomic role P:
  - for each o_1' with $(o_1, o_1') \in P^{\mathcal{I}}$, there is an o_2' with $(o_2, o_2') \in P^{\mathcal{J}}$ such that $o_1' \sim_{\mathcal{B}} o_2'$ (forth property);
  - for each o_2' with $(o_2, o_2') \in P^{\mathcal{J}}$, there is an o_1' with $(o_1, o_1') \in P^{\mathcal{I}}$ such that $o_1' \sim_{\mathcal{B}} o_2'$ (back property).

$(\mathcal{I}, o_1) \sim (\mathcal{J}, o_2)$ means that there is a bisimulation $\sim_{\mathcal{B}}$ between $\mathcal{I}$ and $\mathcal{J}$ such that $o_1 \sim_{\mathcal{B}} o_2$.

Lemma

$\mathcal{ALC}$ cannot distinguish o_1 in interpretation $\mathcal{I}$ and o_2 in interpretation $\mathcal{J}$ when $(\mathcal{I}, o_1) \sim (\mathcal{J}, o_2)$.
In other words, if $(\mathcal{I}, o_1) \sim (\mathcal{J}, o_2)$, then for every $\mathcal{ALC}$ concept C we have that

$$o_1 \in C^{\mathcal{I}} \quad \text{if and only if} \quad o_2 \in C^{\mathcal{J}}$$

Proof.

By induction on the structure of concepts. [Exercise]

Outline

1. Properties of $\mathcal{ALC}$
   - $\mathcal{ALC}$ and First-Order Logic
   - Bissimulation
   - Properties of $\mathcal{ALC}$
2. Reasoning over $\mathcal{ALC}$ concept expressions
   - Tableaux for concept satisfiability
3. Reasoning over $\mathcal{ALC}$ ontologies
   - Reasoning w.r.t. acyclic TBoxes
   - Reasoning w.r.t. arbitrary TBoxes

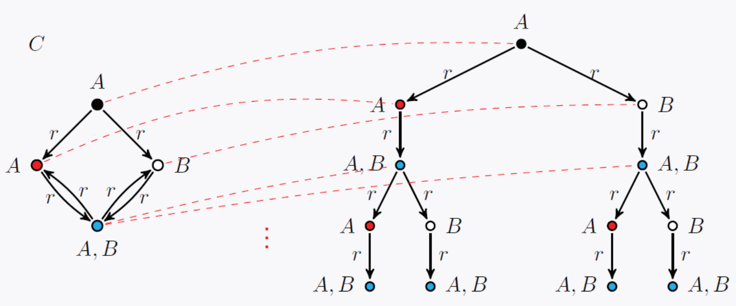
Tree model property of DLs

Theorem

An $\mathcal{ALC}$ concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ if and only if there is a tree-shaped model $\mathcal{I}$ of $\mathcal{T}$ and an object o such that $o \in C^{\mathcal{I}}$.

Proof.

The "if" direction is obvious. For the "only-if" direction, we exploit the fact that an interpretation and its unraveling into a tree are bisimilar.



Exercise

Prove, using the tree model property, that the FOL$_{bin}$ formula $\forall x.P(x,x)$ cannot be translated into $\mathcal{ALC}$. In other words, prove that there is no $\mathcal{ALC}$ TBox $\mathcal{T}$ such that

$$\mathcal{I} \models_{\mathcal{ALC}} \mathcal{T} \text{ if and only if } \mathcal{I} \models_{FOL} \forall x.P(x,x)$$

A consequence of the above fact, and of the fact that $\mathcal{ALC}$ can be expressed in FOL$_{bin}$ is that:

Expressive power of $\mathcal{ALC}$

$\mathcal{ALC}$ is strictly less expressive than FOL$_{bin}$.

Outline

Outline

1. Properties of $\mathcal{ALC}$
   - $\mathcal{ALC}$ and First-Order Logic
   - Bissimulation
   - Properties of $\mathcal{ALC}$
2. Reasoning over $\mathcal{ALC}$ concept expressions
   - Tableaux for concept satisfiability
3. Reasoning over $\mathcal{ALC}$ ontologies
   - Reasoning w.r.t. acyclic TBoxes
   - Reasoning w.r.t. arbitrary TBoxes

Tableau algorithms for DLs

Tableau-based techniques

Determine the satisfiability of a formula (or theory) by using rules to construct (a representation of) a model

- Used in FOL and modal logics for many years;
- For DLs, extensively explored since the late 1990s;
- Well-suited for implementation;
- Many of the most sucessful DL reasoners implement tableau techniques or variants thereof; e.g. , RACER, FaCT++, Pellet, Hermit, etc.

Tableau algorithm for $\mathcal{ALC}$ concepts - Overview

We describe an algorithm that decides concept satisfiability in $\mathcal{ALC}$.

For a given $\mathcal{ALC}$ concept C_0, it tries to build a graph representation of a model $\mathcal{I}$ of C_0:

- It works with labeled tree-shaped graphs:
  - the nodes are labeled with concepts, and
  - the edges are labeled with roles.
- At each moment, the algorithm stores a set $\mathcal{G}$ of labeled graphs.
- It starts with the set $\mathcal{G}_0$ containing one graph with just one node labeled C_0.
- It uses tableau rules corresponding to the constructors, to infer a new set $\mathcal{G}'$ of graphs from the previous set $\mathcal{G}$.
- Intuitively, each new graph makes explicit some constraint (on the model) resulting from C_0 that was still implicit in the previous step.

Tableau algorithm for $\mathcal{ALC}$ concepts - Overview (continued)

- Each rule, when applied to a graph G in the current set $\mathcal{G}$ may:
  - add new nodes to G, or
  - add new labels to the existing nodes of G.
- The rules are non-deterministic in general, i.e., they may be applied in more than one way, resulting in different possible graphs.
- If a graph contains a clash, i.e., an explicit contradiction, it is dropped and not expanded further.
- When no rule can be applied anymore to a graph, the graph is called complete. The algorithm continues
  - until some graph G in the current set is complete and clash-free, or
  - until all graphs contain a clash.
- A complete and clash-free graph G represents a model $\mathcal{I}$ of C_0.

Note: Such graphs can be viewed as ABoxes, and to ease notation, we will adopt this convention.

Negation Normal Form

Definition

A concept C is in negation normal form (NNF) if the $'\neg'$ operator is applied only to atomic concepts

Lemma

Every concept C can be transformed in linear time into an equivalent concept in NNF.

Proof.

A concept C can be transformed in NNF by the following rewriting rules that push inside the $\neg$ operator:

$$\begin{array}{rcl} \neg(C \sqcap D) & \equiv & \neg C \sqcup \neg D \\ \neg(C \sqcup D) & \equiv & \neg C \sqcap \neg D \\ \neg(\neg C) & \equiv & C \\ \neg\forall P.C & \equiv & \exists P.\neg C \\ \neg\exists P.C & \equiv & \forall P.\neg C \end{array}$$

Tableaux rules for checking concept satisfiability

Let C_0 be an $\mathcal{ALC}$ concept in NNF.
To test satisfiability of C_0, a tableaux algorithm:

1. starts with $A_0 := \{C_0(x_0)\}$, and
2. constructs new ABoxes, by applying the following tableaux rules:

Rule	Condition	$\rightarrow$	Effect
$\rightarrow_\sqcap$	$(C_1 \sqcap C_2)(x) \in A$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{C_1(x), C_2(x)\}$
$\rightarrow_\sqcup$	$(C_1 \sqcup C_2)(x) \in A$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{C_1(x)\}$ or $\mathcal{A} := \mathcal{A} \cup \{C_2(x)\}$
$\rightarrow_\exists$	$(\exists P.C)(x) \in A$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{P(x,y), C(y)\}$ where y is fresh
$\rightarrow_\forall$	$(\forall P.C)(x), P(x,y) \in A$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{C(y)\}$

Note:

- A rule is applicable to an ABox $\mathcal{A}$ only if it has an effect on $\mathcal{A}$, i.e., if it adds some new assertion; otherwise it is not applicable to $\mathcal{A}$.
- Since the $\rightarrow_\sqcup$ rule is non-deterministic, starting from $\mathcal{A}_0$, we obtain after each rule application a set S of ABoxes.

Complete and clash-free ABoxes

Definition

An ABox $\mathcal{A}$

- is complete if none of the tableaux rules applies to it.
- has a clash if $\{C(x), \neg C(x)\} \subseteq \mathcal{A}$, and is clash-free otherwise.

A clash represents an obvious contradiction. Hence, it is immediate so see that an ABox containing a clash is unsatisfiable.

Tableaux for concept satisfiability - Example

Consider concept $C_0 = \underbrace{(A_1 \sqcap \overbrace{\exists P.(A_2 \sqcup A_3))}^{C_3}}_{C_1} \sqcap \underbrace{\forall P.\neg A_2}_{C_2}$

$$
\begin{array}{c}
\mathcal{A}_0 = \{C_0(x_0)\} \\
| \\
\mathcal{A}_1 = \mathcal{A}_0 \cup \{C_1(x_0), C_2(x_0)\} \\
| \\
\mathcal{A}_2 = \mathcal{A}_1 \cup \{A_1(x_0), C_3(x_0)\} \\
| \\
\mathcal{A}_3 = \mathcal{A}_2 \cup \{P(x_0, x_1), (A_2 \sqcup A_3(x_1)\} \\
| \\
\mathcal{A}_4 = \mathcal{A}_3 \cup \{\neg A_2(x_1)\} \\
\swarrow \qquad \searrow \\
\mathcal{A}_5 = \mathcal{A}_4 \cup \{A_2(x_1)\} X \qquad \mathcal{A}_6 = \mathcal{A}_4 \cup \{A_3(x_1)\} \surd
\end{array}
$$

For a finite set $\mathcal{S}$ of ABoxes, we say that $\mathcal{S}$ is consistent if it contains at least one satisfiable ABox.

Lemma

1. Termination: There cannot be an infinite sequence of rule applications

$$\mathcal{S} = \{\{C_0(x_0)\}\} \rightarrow \mathcal{S}_1 \rightarrow \mathcal{S}_2 \rightarrow \ldots$$

2. Soundness:
   - If by applying a tableaux rule to the set $\mathcal{S}$ of ABoxes we obtain the set $\mathcal{S}'$, then $\mathcal{S}$ is consistent iff $\mathcal{S}'$ is consistent.
   - If the tableau algorithm builds a complete and clash-free ABox for an $\mathcal{ALC}$ concept C_0, then C_0 is satisfiable.
3. Completeness: If C_0 is satisfiable, then the tableau algorithm builds a complete and clash-free ABox for C_0.

Canonical interpretation and decidability of satisfiability

To show that every complete and clash-free ABox $\mathcal{A}$ is satisfiable, we describe how to generate from such an $\mathcal{A}$ an interpretation $\mathcal{I}_\mathcal{A}$ that is a model of $\mathcal{A}$.
This interpretation is called...

Def.: Canonical interpretation $\mathcal{I}_\mathcal{A}$ of a complete and clash-free ABox $\mathcal{A}$

- $\Delta^{\mathcal{I}_\mathcal{A}} = \{x \mid C(x), P(x,y),$ or $P(y,x) \in \mathcal{A}\}$.
- $A^{\mathcal{I}_\mathcal{A}} = \{x \mid A(x) \in \mathcal{A}\}$, for every atomic concept $\mathcal{A}$.
- $P^{\mathcal{I}_\mathcal{A}} = \{(x,y) \mid P(x,y) \in \mathcal{A}\}$, for every atomic role $\mathcal{P}$.

Theorem

Satisfiability of $\mathcal{ALC}$ concepts is decidable.

Proof.

Is based on showing that the canonical interpretation of an ABox $\mathcal{A}$ obtained starting from a concept C is indeed a model of C.

Exercise

Check the satisfiability of the following concepts:

1. $\neg(\forall R.A \sqcup \exists R.(\neg A \sqcap \neg B))$
2. $\exists R.(\forall S.C) \sqcap \forall R.(\exists S.\neg C)$
3. $\exists S.C \sqcap \exists S.D \sqcap \forall S.(\neg C \sqcup \neg D)$
4. $\exists S.(C \sqcap D) \sqcap (\forall S.\neg C \sqcup \exists S.\neg D)$
5. $C \sqcap \exists R.A \sqcap \exists R.B \sqcap \neg\exists R.(A \sqcap B)$

Exercise

Check if the following subsumption is valid:

$$\neg\forall R.A \sqcap \forall R.((\forall R.B) \sqcup A) \sqsubseteq \forall R.\neg(\exists R.A) \sqcap \exists R.(\exists R.B)$$

Outline

1. Properties of $\mathcal{ALC}$
   - $\mathcal{ALC}$ and First-Order Logic
   - Bissimulation
   - Properties of $\mathcal{ALC}$

2. Reasoning over $\mathcal{ALC}$ concept expressions
   - Tableaux for concept satisfiability

3. Reasoning over $\mathcal{ALC}$ ontologies
   - Reasoning w.r.t. acyclic TBoxes
   - Reasoning w.r.t. arbitrary TBoxes

Outline

TBox reasoning

- TBox Satisfiability: $\mathcal{T}$ is satisfiable, if it admits at least one model.
- Concept Satisfiability w.r.t. a TBox: C is satisfiable w.r.t. $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty, i.e., $\mathcal{T} \not\models C \equiv \bot$.
- Subsumption: C_1 is subsumed by C_2 w.r.t. $\mathcal{T}$ if, for every model $\mathcal{I}$ of $\mathcal{T}$, we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \sqsubseteq C_2$.
- Equivalence: C_1 and C_2 are equivalent w.r.t. $\mathcal{T}$ if, for every model $\mathcal{I}$ of $\mathcal{T}$, we have $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$.

We can reduce all reasoning tasks to concept satisfiability w.r.t. a TBox.
[Exercise]

Acyclic TBox

Def.: Concept definition

A definition of an atomic concept A is an assertion of the form $A \equiv C$, where C is an arbitrary concept expression in which A does not occur.

Def.: Cyclic concept definitions

A set of concept definitions is cyclic if it is of the form

$$A_1 \equiv C_1[A_2], \quad A_2 \equiv C_2[A_3], \ldots, \quad A_n \equiv C_n[A_1]$$

where $C[A]$ means that A occurs in the concept expression C.

Def.: Acyclic TBox

A TBox is acyclic if it is a set of concept definitions that neither contains multiple definitions of the same concept, nor a set of cyclic definitions.

Unfolding w.r.t. an acyclic TBox

Satisfiability of a concept C w.r.t. an acyclic TBox $\mathcal{T}$ can be reduced to pure concept satisfiability by unfolding C w.r.t. $\mathcal{T}$:

1. We start from the concept C to check for satisfiability.
2. Whenever $\mathcal{T}$ contains a definition $A \equiv C'$, and A occurs in C, then in C we substitute A with C'.
3. We continue until no more substitutions are possible.

Theorem

Let $Unfold_{\mathcal{T}}(C)$ be the result of unfolding C w.r.t $\mathcal{T}$.
Then C is satisfiable w.r.t. $\mathcal{T}$ iff $Unfold_{\mathcal{T}}(C)$ is satisfiable.

Proof.

By induction on the number of unfolding steps. [Exercise]

Unfolding a concept w.r.t. an acyclic TBox might lead to an exponential blow up.
For each n, let $\mathcal{T}_n$ be the acyclic TBox:

$$
\begin{array}{rcl}
A_0 & \equiv & \forall P.A_1 \sqcap \forall R.A_1 \\
A_1 & \equiv & \forall P.A_2 \sqcap \forall R.A_2 \\
 & \cdots & \\
A_{n-1} & \equiv & \forall P.A_n \sqcap \forall R.A_n
\end{array}
$$

It is easy to see that $Unfold_{\mathcal{T}}(A_0)$ grows exponentially with n.

Concept satisfiability w.r.t. an acyclic TBox

We adopt a smarter strategy: unfolding on demand

Rule	Condition	$\to$	Effect
$\to_\sqcap$	$(C_1 \sqcap C_2)(x) \in \mathcal{A}$	$\to$	$\mathcal{A} := \mathcal{A} \cup \{C_1(x), C_2(x)\}$
$\to_\sqcup$	$(C_1 \sqcup C_2)(x) \in \mathcal{A}$	$\to$	$\mathcal{A} := \mathcal{A} \cup \{C_1(x)\}$ or $\mathcal{A} := \mathcal{A} \cup \{C_2(x)\}$
$\to_\exists$	$(\exists P.C)(x) \in \mathcal{A}$	$\to$	$\mathcal{A} := \mathcal{A} \cup \{P(x,y), C(y)\}$where y is fresh
$\to_\forall$	$(\forall P.C)(x), P(x,y) \in \mathcal{A}$	$\to$	$\mathcal{A} := \mathcal{A} \cup \{C(y)\}$
$\to_\mathcal{T}$	$A(x) \in \mathcal{A}$ and $A \equiv C \in \mathcal{T}$	$\to$	$\mathcal{A} := \mathcal{A} \cup \{NNF(C)(x)\}$
$\to_\mathcal{T}$	$\neg A(x) \in \mathcal{A}$ and $A \equiv C \in \mathcal{T}$	$\to$	$\mathcal{A} := \mathcal{A} \cup \{NNF(\neg C)(x)\}$

Theorem

In $\mathcal{ALC}$, concept satisfiability w.r.t. acyclic TBoxes is PSpace-complete.

Outline

1. Properties of $\mathcal{ALC}$
   - $\mathcal{ALC}$ and First-Order Logic
   - Bissimulation
   - Properties of $\mathcal{ALC}$

2. Reasoning over $\mathcal{ALC}$ concept expressions
   - Tableaux for concept satisfiability

3. Reasoning over $\mathcal{ALC}$ ontologies
   - Reasoning w.r.t. acyclic TBoxes
   - Reasoning w.r.t. arbitrary TBoxes

Tableaux rule for arbitrary TBox axioms

When the TBox may contain cycles, unfolding cannot be used, since in general it would not terminate.
Instead, we modify the tableaux by relying on the following observations:

- $C \sqsubseteq D$ is equivalent to $\top \sqsubseteq \neg C \sqcup D$.
  Hence, $\bigcup_i \{C_i \sqsubseteq D_i\}$ is equivalent to a single inclusion $\top \sqsubseteq \prod_i (\neg C_i \sqcup D_i)$.
- If $\top \sqsubseteq C$ is in $\mathcal{T}$ then for every ABox $\mathcal{A}$ generated by the tableaux and for every occurrence of some x in $\mathcal{A}$, we have to add also the fact $C(x)$.
- We can obtain this effect by adding a suitable rule to the tableaux rules:

Rule	Condition	$\rightarrow$	Effect
$\rightarrow_\sqcap$	$(C_1 \sqcap C_2)(x) \in \mathcal{A}$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{C_1(x), C_2(x)\}$
$\rightarrow_\sqcup$	$(C_1 \sqcup C_2)(x) \in \mathcal{A}$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{C_1(x)\}$ or $\mathcal{A} := \mathcal{A} \cup \{C_2(x)\}$
$\rightarrow_\exists$	$(\exists P.C)(x) \in \mathcal{A}$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{P(x,y), C(y)\}$ where y is fresh
$\rightarrow_\forall$	$(\forall P.C)(x), P(x,y) \in \mathcal{A}$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{C(y)\}$
$\rightarrow_\mathcal{T}$	x occurs in $\mathcal{A}$	$\rightarrow$	$\mathcal{A} := \mathcal{A} \cup \{\prod_{C \sqsubseteq D \in \mathcal{T}} NNF(\neg C \sqcup D)(x)\}$

Tableaux rule for arbitrary TBox axioms - Example

Exercise

Check if C is satisfiable w.r.t. the TBox $\{C \sqsubseteq \exists R.C\}$

Solution

$$
\begin{array}{lll}
\{C(x_0)\} & \rightarrow_{\mathcal{T}} & \{C(x_0), (\neg C \sqcup \exists R.C)(x_0)\} \\
 & \rightarrow_{\sqcup} & \{C(x_0), \ldots, (\exists R.C)(x_0)\} \\
 & \rightarrow_{\exists} & \{C(x_0), \ldots, R(x_0, x_1), C(x_1)\} \\
 & \rightarrow_{\mathcal{T}} & \{C(x_0), \ldots, R(x_0, x_1), C(x_1), (\neg C \sqcup \exists R.C)(x_1)\} \\
 & \rightarrow_{\sqcup} & \{C(x_0), \ldots, R(x_0, x_1), C(x_1), \ldots, \exists R.C(x_1)\} \\
 & \rightarrow_{\exists} & \{C(x_0), \ldots, R(x_0, x_1), C(x_1), \ldots, R(x_1, x_2), C(x_2)\} \\
 & \rightarrow_{\mathcal{T}} & \ldots
\end{array}
$$

Termination is no longer guaranteed!

Due to the application of the $\rightarrow_{\mathcal{T}}$ -rule, the nesting of the concepts does not decrease with each rule-application step.

Blocking

To guarantee termination, we need to understand when it is not necessary anymore to create new objects.

Def.: Blocking

- y is an ancestor of x in an ABox $\mathcal{A}$, if $\mathcal{A}$ contains

$$R_0(y, x_1), R_1(x_1, x_2), \ldots, R_n(x_n, x)$$

- We label objects with sets of concepts: $\mathcal{L}(x) = \{C \mid C(x) \in \mathcal{A}\}$
- x is directly blocked in $\mathcal{A}$ if it has an ancestor y with $\mathcal{L}(x) \subseteq \mathcal{L}(y)$
- If y is the closest such node to x, we say that x is blocked by y.
- A node is blocked if it is directly blocked or one of its ancestors is blocked.

The application of all rules is restricted to nodes that are not blocked.
With this blocking strategy, one can show that the algorithm is guaranteed to terminate.

Blocking - Exercise

Exercise

Check if C is satisfiable w.r.t. the TBox $\{C \sqsubseteq \exists R.C\}$.

Solution

$$
\begin{array}{lll}
\{C(x_0)\} & \rightarrow_{\mathcal{T}} & \{C(x_0), (\neg C \sqcup \exists R.C)(x_0)\} \\
 & \rightarrow_{\sqcup} & \{C(x_0), (\neg C \sqcup \exists R.C)(x_0), (\exists R.C)(x_0)\} \\
 & \rightarrow_{\exists} & \{C(x_0), (\neg C \sqcup \exists R.C)(x_0), (\exists R.C)(x_0), R(x_0, x_1), C(x_1)\}
\end{array}
$$

x_1 is blocked by x_0 since $\mathcal{L}(x_1) = \{C\}$ and $\mathcal{L}(x_0) = \{C, \neg C \sqcup \exists R.C, \exists R.C\}$, hence $\mathcal{L}(x_1) \subseteq \mathcal{L}(x_0)$.

Cyclic interpretations

The interpretation $\mathcal{I}_\mathcal{A}$ generated from an ABox $\mathcal{A}$ obtained by the tableaux algorithm with blocking strategy is defined as follows:

- $\Delta^{\mathcal{I}_\mathcal{A}} = \{x \mid C(x) \in \mathcal{A}$ and x is not blocked $\}$
- $A^{\mathcal{I}_\mathcal{A}} = \{x \mid x \in \Delta^{\mathcal{I}_\mathcal{A}}$ and $A(x) \in \mathcal{A}\}$
- $P^{\mathcal{I}_\mathcal{A}} = \{(x,y) \mid \{x,y\} \subseteq \Delta^{\mathcal{I}_\mathcal{A}}$ and $P(x,y) \in \mathcal{A}\} \cup$
  $\{(x,y) \mid x \subseteq \Delta^{\mathcal{I}_\mathcal{A}}, P(x,y') \in \mathcal{A}$, and y' is blocked by $y\}$

Complexity

The algorithm runs no longer in PSpace since it may generate role paths of exponential length before blocking occurs.

Finite model property

Theorem

A satisfiable $\mathcal{ALC}$ TBox has a finite model.

Proof.

The model constructed via tableaux is finite.
Completeness of the tableaux procedure implies that if a TBox is satisfiable, then the algorithm will find a model, which is indeed finite.

Complexity of reasoning over DL ontologies

Reasoning over DL ontologies is much more complex than reasoning over concept expressions:

Bad news:

- without restrictions on the form of TBox assertions, reasoning over DL ontologies is already ExpTime-**hard**, even for very simple DLs (see, e.g., [Donini, 2003]).

Good news:

- We can add a lot of expressivity (i.e., essentially all DL constructs seen so far), while still staying within the ExpTime upper bound [Pratt, 1979; Schild, 1991; Calvanese and De Giacomo, 2003].
- There are DL reasoners that perform reasonably well in practice for such DLs (e.g, Racer, Pellet, Fact++, HermiT, ...)