

---



---



---

## Reasoning in Description Logic

INFS4206/INFS7206

School of Information Technology and Electrical Engineering

Semester 2, 2006




---



---



---

## Reasoning Services in Description Logic

- ▶ **Concept Satisfiability:**  $\mathcal{T} \not\models C \equiv \perp$  (e.g.,  $Lecturer \equiv \perp$ )  
The problem of checking whether  $C$  is satisfiable w.r.t.  $\mathcal{T}$ , i.e., whether there exists a model  $\mathcal{I}$  of  $\mathcal{T}$  such that  $C^{\mathcal{I}} \neq \emptyset$ .
- ▶ **Subsumption:**  $\mathcal{T} \models C \sqsubseteq D$  (e.g.,  $ITEELecturer \sqsubseteq Lecturer$ )  
The problem of checking whether  $C$  is subsumed by  $D$  w.r.t.  $\mathcal{T}$ , i.e., whether  $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$  in every model  $\mathcal{I}$  of  $\mathcal{T}$ .
- ▶ **Satisfiability:**  $\mathcal{T} \not\models$  (e.g.,  $Student \equiv Lecturer$ )  
the problem of checking whether  $\mathcal{T}$  is satisfiable, i.e., whether it has a model
- ▶ **Instance Checking:**  $\mathcal{T} \models C(a)$  (e.g.,  $Lecturer(GUIDO)$ )  
the problem of checking whether the assertion  $C(a)$  is satisfied in every model of  $\mathcal{T}$ .

## Reduction to satisfiability

- ▶ Concept Satisfiability

$$\mathcal{T} \not\models C \equiv \perp \text{ iff}$$

exists  $x$  such that  $\mathcal{T} \cup \{C(x)\}$  has a model

- ▶ Subsumption

$$\mathcal{T} \models C \sqsubseteq D \text{ iff}$$

$\mathcal{T} \cup \{(C \sqcap \neg D)(x)\}$  has no models

- ▶ Instance Checking

$$\mathcal{T} \models C(a) \text{ iff}$$

$\mathcal{T} \cup \{\neg C(a)\}$  has no models

## Reasoning Procedures

Terminating, efficient and complete algorithms for deciding satisfiability –and all the other reasoning services– are available.

- ▶ Structural subsumption (for Description Logics with only  $\sqcap$ , atomic negation,  $\forall R.C$  and  $\perp$ )
- ▶ Tableaux algorithm (for all Description Logics)

## Structural subsumption

1. Transform the concepts in normal form
2. Check for subsumption
  - ▶  $\perp$  is subsumed by every concept
  - ▶  $D$  subsumes  $C$  (or  $C$  is subsumed by  $D$ ) if

$$D = A_1 \sqcap \dots \sqcap A_n \sqcap \forall R_1.D_1 \sqcap \dots \sqcap \forall R_k.D_k$$

and

$$C = B_1 \sqcap \dots \sqcap B_m \sqcap \forall S_1.C_1 \sqcap \dots \sqcap \forall S_l.C_l$$

and

- (a) for every  $i$ ,  $1 \leq i \leq n$  there is  $j$ ,  $1 \leq j \leq m$  such that  $A_i = B_j$  and
- (b) for every  $i$ ,  $1 \leq i \leq k$  there is  $j$ ,  $1 \leq j \leq l$  such that  $R_i = S_j$  and  $C_j \sqsubseteq D_i$ .

## Example

Given the concepts

$$\forall S.(F \sqcap \forall R.D) \sqcap A \sqcap \forall R.C \sqcap \forall S.\forall R.(C \sqcap \neg D)$$

and

$$\forall S.(A \sqcap \forall R.(\neg C \sqcap C) \sqcap \neg A) \sqcap \forall R.(C \sqcap B) \sqcap A \sqcap \forall R.(\neg C \sqcap \forall S.F)$$

Prove that the first concept subsumes the second

---

---

---

## Tableaux

- ▶ The Tableaux Algorithm is a decision procedure solving the problem of satisfiability.
- ▶ If a formula is satisfiable, the procedure will constructively exhibit a model of the formula.
- ▶ The basic idea is to incrementally build the model by looking at the formula, by decomposing it in a top/down fashion. The procedure exhaustively looks at all the possibilities, so that it can eventually prove that no model could be found for unsatisfiable formulas.

---

---

---

## Tableaux Algorithm

1. List all the assertions in an ABox  $\mathcal{A}$ .
2. Add assertions to  $\mathcal{A}$ , applying specific **completion rules**. Completion rules are either deterministic – they yield a uniquely determined set of assertions system – or nondeterministic – yielding several possible alternative sets of assertions systems (**branches**).
3. Apply the completion rules until either a contradiction (a **clash**) is generated in every branch, or there is a **completed** branch where no more rule is applicable.
4. The completed constraint system gives a model of  $\mathcal{K}$ ; it corresponds to a particular branch of the tableaux.

## Negation Normal Form

we can transform any  $\mathcal{ALC}$  formula into an equivalent one in Negation Normal Form, so that negation appears only in front of atomic concepts:

- ▶  $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$
- ▶  $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$
- ▶  $\neg\forall R.C \equiv \exists R.\neg C$
- ▶  $\neg\exists R.C \equiv \forall R.\neg C$

## Completion Rules: the AND rule

- ▶ The propagation rules come straightforwardly from the semantics of constructors.
- ▶ If in a given interpretation  $\mathcal{I}$ , whose domain contains the element  $a$ , we have that  $a \in (C \sqcap D)^{\mathcal{I}}$ , then from the semantics we know that such element  $a$  should be in the intersection of  $C^{\mathcal{I}}$  and  $D^{\mathcal{I}}$ , i.e., it should be in both  $C^{\mathcal{I}}$  and  $D^{\mathcal{I}}$ .
- ▶ Since this must be true for any interpretation, we can abstract from interpretations and their elements, and say that if in a generic interpretation we have a generic element  $x$  that is in the interpretation of the concept  $C \sqcap D$  (denote this by  $(C \sqcap D)(x)$ ) then the element  $x$  should belong both to the interpretation of  $C$  and to the interpretation of  $D$ .

## The AND rule

- ▶ Suppose now we want to construct a generic interpretation  $\mathcal{I}$  for an ABox  $\mathcal{A}$  containing  $C \sqcap D$ , such that the set corresponding to the concept  $C \sqcap D$  contains at least one element. We can state this initial requirement as the assertion  $(C \sqcap D)(x)$ .
- ▶ Following the semantics, we know that  $\mathcal{A}$  must be such that the assertions  $C(x)$  and  $D(x)$  must hold, hence we can add these new assertions to  $\mathcal{A}$ , knowing that if  $\mathcal{A}$  will ever satisfy them then it will also satisfy the first assertion.
- ▶ hence we have the following propagation rule:
  - ▶ if  $(C \sqcap D)(x) \in \mathcal{A}$ , but  $\mathcal{A}$  does not contain both  $C(x)$  and  $D(x)$ ; then  $\mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$

## The OR rule

- ▶ If in a given interpretation  $\mathcal{I}$ , whose domain contains the element  $a$ , we have that  $a \in (C \sqcup D)^{\mathcal{I}}$ , then from the semantics we know that there either  $a \in C^{\mathcal{I}}$  or  $a \in D^{\mathcal{I}}$ ;
- ▶ Since this must be true for any interpretation, we can abstract from interpretations and their elements, and say that if in a generic interpretation we have a generic element  $x$  that is in the interpretation of the concept  $C \sqcup D$  ( $(C \sqcup D)(x)$ ) then we can add either the assertion  $C(x)$  or the assertion  $D(x)$ ; thus we have a non-deterministic choice and we have to explore both branches.
- ▶ hence we have the following propagation rule:
  - ▶ if  $(C \sqcup D)(x) \in \mathcal{A}$ , but neither  $C(x) \in \mathcal{A}$  nor  $D(x) \in \mathcal{A}$ ; then  $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$ ,  $\mathcal{A}'' = \mathcal{A} \cup \{D(x)\}$

## The SOME rule

- ▶ If in a given interpretation  $\mathcal{I}$ , whose domain contains the element  $a$ , we have that  $a \in (\exists R : C)^{\mathcal{I}}$ , then from the semantics we know that there must be an element  $b$  (not necessarily distinct from  $a$ ) such that  $(a, b) \in R^{\mathcal{I}}$ , and  $b \in C^{\mathcal{I}}$ .
- ▶ Since this must be true for any interpretation, we can abstract from interpretations and their elements, and say that if in a generic interpretation we have a generic element  $x$  that is in the interpretation of the concept  $\exists R.C$  (denote this by  $\exists R.C(x)$ ) then there must be a generic element  $y$  such that  $x$  and  $y$  are in relation through  $R$  (denote it  $R(x, y)$ ) and  $y$  belongs to the interpretation of  $C$  (denoted as  $C(y)$ ).
- ▶ hence we have the following propagation rule:
  - ▶ if  $(\exists R.C)(x) \in \mathcal{A}$ , but there is no individual  $z$  such that  $C(z) \in \mathcal{A}$  and  $R(x, z) \in \mathcal{A}$ ; then  $\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x, y)\}$  where  $y$  is an individual that does not occur in  $\mathcal{A}$ .

## The FORALL rule

- ▶ If in a given interpretation  $\mathcal{I}$ , whose domain contains the element  $a$ , we have that  $a \in (\forall R : C)^{\mathcal{I}}$  then for every element  $b$  (not necessarily distinct from  $a$ ) if  $(a, b) \in R^{\mathcal{I}}$ , then from the semantics we know that  $b \in C^{\mathcal{I}}$ .
- ▶ Since this must be true for any interpretation, we can abstract from interpretations and their elements, and say that if in a generic interpretation we have a generic element  $x$  that is in the interpretation of the concept  $\forall R.C$  (denote this by  $\forall R.C(x)$ ), then for every element  $y$  such that  $x, y$  is in the interpretation of  $R$  (i.e.,  $R(x, y)$ ), then  $y$  is in the interpretation of  $C$  (denoted as  $C(y)$ ).
- ▶ hence we have the following propagation rule:
  - ▶ if  $(\forall R.C)(x) \in \mathcal{A}$  and  $R(x, y) \in \mathcal{A}$ , but  $C(y)$  is not in  $\mathcal{A}$ , then  $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$

## Clash

- ▶ While building an interpretation system, we can look for evident contradictions to see if the set of assertions is not satisfiable. We call these contradictions **clashes**.
- ▶ A **clash** is a constraint system having the form:

$$\{C(a), \neg C(a)\}$$

- ▶ A clash is evidently an unsatisfiable set of assertions, hence any set of assertions containing a clash is obviously unsatisfiable.

## Examples of tableaux algorithm

- ▶ Determine whether the concept

$$\forall R.C \sqcap \forall R.D$$

subsumes the concept

$$\forall R.(C \sqcap D)$$

- ▶ Determine whether the concept

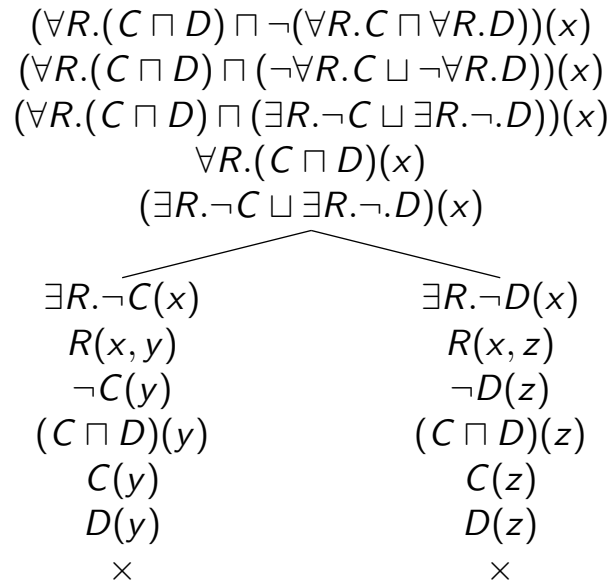
$$\exists R.(C \sqcap D)$$

subsumes the concept

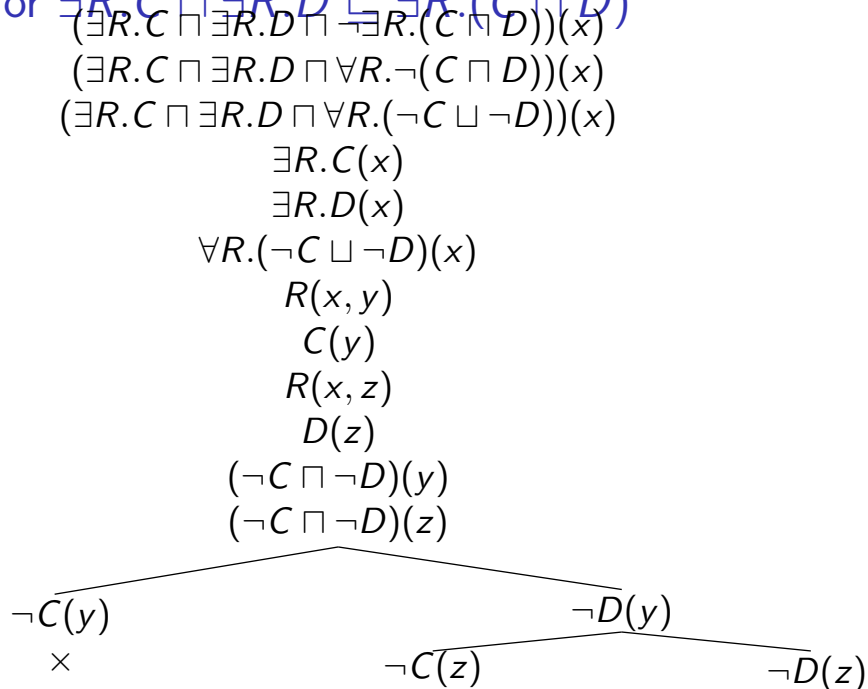
$$\exists R.C \sqcap \exists R.D$$

In case of negative answer give a counter-model to the subsumption problem.

### Tableaux for $\forall R.(C \sqcap D) \sqsubseteq \forall R.C \sqcap \forall R.D$



### Tableaux for $\exists R.C \sqcap \exists R.D \sqsubseteq \exists R.(C \sqcap D)$



---

---

---

## Reasoning with TBoxes

### Statements in a TBox

1.  $A \sqsubseteq C$  (Primitive concept definition)
2.  $A \equiv C$  (Concept definition)
3.  $C \sqsubseteq D$  (Concept inclusion)
4.  $C \equiv D$  (Concept equation)

## Simple (Acyclic) TBox

1.  $A \sqsubseteq C$  (Primitive concept definition)
2.  $A \equiv C$  (Concept definition)
3. well-founded definition (Simple Acyclic TBox)

A concept name  $A$  directly uses a concept name  $B$  in a TBox  $\mathcal{T}$  iff the definition of  $A$  mentions  $B$ . A concept name  $A$  uses a concept name  $B_n$  iff there is a chain of concept names  $A, B_1, \dots, B_n$  such that  $B_i$  directly uses  $B_{i+1}$ . A TBox is acyclic iff no concept name uses itself.

## Subsumption in simple acyclic TBoxes

### Theorem

Subsumption in acyclic simple TBoxes ( $\mathcal{T} \models C \sqsubseteq C$ ) can be reduced in subsumption in an empty TBox ( $\models \hat{C} \sqsubseteq \hat{D}$ ).

In order to get  $\hat{C}$  (and  $\hat{D}$ )

1. Transform the TBox  $\mathcal{T}$  into a new TBox  $\mathcal{T}'$ , by replacing every primitive concept definition in  $\mathcal{T}$  of the form  $A \sqsubseteq C$  with a concept definition  $A \equiv C \sqcap A^*$ , where  $A^*$  is a freshly new generated concept name (called *primitive component* of  $A$ ). Now  $\mathcal{T}'$  contains only (acyclic) concept definitions.
2. Iteratively substitute every occurrence of any defined concept name in  $C$  (and  $D$ ) by the corresponding definition in  $\mathcal{T}'$ . Since  $\mathcal{T}'$  is still acyclic, the process terminates in a finite number of iterations. This process is called **unfolding** or **expansion**.

## Reasoning with acyclic TBoxes

$$A \sqsubseteq C \rightsquigarrow A \equiv C \sqcap A^*$$

$A^*$  denotes the *unexpressed* part of meaning implicitly contained in the primitive concept definition.

### Theorem

1. For each interpretation  $\mathcal{I}$  of  $\mathcal{K}$  there exists an interpretation  $\mathcal{I}'$  of  $\mathcal{K}'$  (and vice-versa) such that  $C^{\mathcal{I}} = C^{\mathcal{I}'}$  for each concept name  $C \in \mathcal{K}$ .
2.  $\mathcal{K} \models C \sqsubseteq D$  iff  $\mathcal{K}' \models C \sqsubseteq D$
3.  $\mathcal{K}' \models C \sqsubseteq D$  iff  $\models \hat{C} \sqsubseteq \hat{D}$ .

## Example

Prove that the knowledge base

$$\begin{aligned} Student &\sqsubseteq \neg Lecturer \\ Lecturer &\sqsubseteq \exists teaches.Course \end{aligned}$$

entails

$$\neg(Student \sqcap \exists teaches.Course)$$

Create the new TBox

$$\begin{aligned} Student &\equiv \neg Lecturer \sqcap Student^* \\ Lecturer &\equiv \exists teaches.Course \sqcap Lecturer^* \end{aligned}$$

Unfolding the concept

$$\begin{aligned} &\neg((\neg Lecturer \sqcap Student^*) \sqcap \exists teaches.Course) \\ &\neg((\neg(\exists teaches.Course \sqcap Lecturer^*) \sqcap Student^*) \sqcap \exists teaches.Course) \end{aligned}$$

You can use the tableaux algorithm to check whether the concept is satisfiable.

## Necessary and sufficient conditions

- ▶ A primitive concept definition  $A \sqsubseteq C$  states a necessary but not sufficient condition for membership in the class  $A$ . Having the property  $C$  is necessary for an object in order to be in the class  $A$ ; however, this condition alone is not sufficient in order to conclude that the object is in the class  $A$ .
- ▶ A concept definition  $A \equiv C$  states necessary and sufficient condition for membership in the class  $A$ . Having the property  $C$  is necessary for an object in order to be in the class  $A$ ; moreover, this condition alone is sufficient in order to conclude that the object is in the class  $A$ .
- ▶ When transforming primitive concept definitions into concept definitions we get necessary and sufficient conditions for membership in the primitive class  $A$ . However, the condition of being in the primitive component  $A^*$  can never be satisfied, since the concept name  $A^*$  can never be referred to by any other concept.

## Definitions

- ▶ Definitions are intended to provide an exact account for the concept name being defined.
- ▶ Given an initial interpretation of the primitive concept names there exists a unique way determine the interpretation of defined concept names; indeed, that's why they are called definitions.
- ▶ This justifies the correctness of unfolding: we can always replace a concept name with its definition, since it doesn't add anything to the theory.
- ▶ However, if the (simple) TBox is cyclic, this is no more true.