# Semantic Data

# Chapter 8 : Reasoning with description logics

Jean-Louis Binot

# Sources and recommended readings

- There are no additional required references for this chapter.
- Sources and useful additional readings :
  - A good description of the tableau algorithm for $\mathcal{ALC}$ is found in *Basic description logic* (*Baader and Nutt 2003)*, and also in *An introduction to description logic (Baader et al. 2017)*.
- University courses having partially inspired ideas and examples for this chapter :
  - *Ontology Engineering for The Semantic Web (COMP62342), Bechhofer and Sattler, University of Manchester.*
  - *Description Logic, Penaloza, Technische Universität Dresden.*
  - *Ontology languages (COMP321), Wolter, University of Liverpool.*
  - *Grundlagen von Ontologien und Datenbanken für Informationssysteme (CS5130), Ozcep, Universität Lübeck.*

# Agenda



Pages 39 to 41 are not in the material for the exam except for the basic fact that GCIs require a special rule with « blocking » to avoid infinite loops

# How large are ontologies ?

- Small ontologies
  - Foaf : 13 classes, ~ 60 properties.
  - Dublin Core : *DC elements :* 15 properties. *DC terms :* 22 classes and 55 properties.
  - Music Ontology : ~ 50 classes and 150 properties.
- Large ontologies
  - SNOMED CT : > 350000 classes, > 50 properties
  - GO (Gene Ontology) : > 50000 terms, few properties; very large number of annotations.
  - NCIT : > 160000 classes, 97 properties.
  - DBpedia : ~ 680 classes, 2,800 properties, > 5,000,000 instances.
- Very large ontologies
  - Yago : 350,000 classes, 10 million entities, 120 million facts about these entities.
  - (CIC) > 500000 classes, 17000 properties, 7 million assertions

# How good are the reasoners ?


- 14 reasoners evaluated.
- Benchmark containing :
  - A wide range of ontologies from the web, selected from a web crawler and from the BioPortal repository for biomedical ontologies.
  - Some user-designed ontologies with special difficult test cases.

Results

# How good are the reasoners ? ./.

Results for discipline **OWL DL Realisation**

Final Ranking:

1. **Konclude**
2. **FaCT++**
3. **HermiT**
4. HermiT-OA4
5. TrOWL
6. Pellet-OA4
7. JFact
8. PAGOdA
9. Chainsaw
10. Racer

Results for discipline **OWL DL Consistency**

Final Ranking:

1. **Konclude**
2. **HermiT**
3. **HermiT-OA4**
4. Chainsaw
5. Pellet-OA4
6. FaCT++
7. TrOWL
8. MOReHermiT
9. Racer
10. JFact

Results for discipline **OWL EL Consistency**

Final Ranking:

1. **ELK**
2. **Konclude**
3. **ELepHant**
4. MOReHermiT
5. Pellet-OA4
6. HermiT
7. HermiT-OA4
8. Chainsaw
9. TrOWL
10. FaCT++
11. jcel
12. Racer
13. JFact

Results for discipline **OWL DL Classification**

Final Ranking:

1. **Konclude**
2. **MOReHermiT**
3. **HermiT-OA4**
4. HermiT
5. TrOWL
6. FaCT++
7. Pellet-OA4
8. Racer
9. JFact
10. Chainsaw

Results for discipline **OWL EL Realisation**

Final Ranking:

1. **Konclude**
2. **ELK**
3. **TrOWL**
4. PAGOdA
5. ELepHant
6. FaCT++
7. JFact
8. Pellet-OA4
9. HermiT
10. HermiT-OA4
11. Chainsaw
12. Racer
13. jcel

The competition covered two types of ontologies (why ?) and several reasoning problems.

# How good are the reasoners ? ./.

Discipline: **OWL EL Realisation** (finished)

| Rank | Reasoner | Score | ! | Time |
|---|---|---|---|---|
| 1 | Konclude | 104 / 109 | 5 | 229.9 s |
| 2 | ELK | 102 / 109 | 7 | 277.8 s |
| 3 | TrOWL | 86 / 109 | 23 | 242.3 s |
| 4 | PAGOdA | 86 / 109 | 23 | 1,771.7 s |
| 5 | ELepHant | 84 / 109 | 25 | 424.8 s |
| 6 | FaCT++ | 79 / 109 | 30 | 354.2 s |
| 7 | JFact | 63 / 109 | 46 | 280.7 s |
| 8 | Pellet-OA4 | 60 / 109 | 49 | 1,154.3 s |
| 9 | HermiT | 57 / 109 | 52 | 905.1 s |
| 10 | HermiT-OA4 | 57 / 109 | 52 | 934.4 s |
| 11 | Chainsaw | 43 / 109 | 66 | 251.9 s |
| 12 | Racer | 32 / 109 | 77 | 518.8 s |
| 13 | jcel | 0 / 109 | 109 | 0.0 s |

Discipline: **OWL DL Realisation** (finished)

| Rank | Reasoner | Score | ! | Time |
|---|---|---|---|---|
| 1 | Konclude | 247 / 264 | 17 | 739.3 s |
| 2 | FaCT++ | 172 / 264 | 92 | 1,111.3 s |
| 3 | HermiT | 163 / 264 | 101 | 2,934.9 s |
| 4 | HermiT-OA4 | 162 / 264 | 102 | 3,022.5 s |
| 5 | TrOWL | 150 / 264 | 114 | 503.5 s |
| 6 | Pellet-OA4 | 136 / 264 | 128 | 1,434.2 s |
| 7 | JFact | 109 / 264 | 155 | 1,252.6 s |
| 8 | PAGOdA | 104 / 264 | 160 | 3,437.5 s |
| 9 | Chainsaw | 79 / 264 | 185 | 1,067.6 s |
| 10 | Racer | 46 / 264 | 218 | 294.8 s |

Discipline: **OWL EL Classification** (finished)

| Rank | Reasoner | Score | ! | Time |
|---|---|---|---|---|
| 1 | ELK | 298 / 298 | 0 | 674.1 s |
| 2 | Konclude | 294 / 298 | 4 | 622.3 s |
| 3 | MOReHermiT | 294 / 298 | 4 | 1,685.1 s |
| 4 | ELepHant | 291 / 298 | 7 | 957.0 s |
| 5 | TrOWL | 275 / 298 | 23 | 767.4 s |
| 6 | HermiT | 272 / 298 | 26 | 2,012.9 s |
| 7 | HermiT-OA4 | 272 / 298 | 26 | 2,068.6 s |
| 8 | Pellet-OA4 | 261 / 298 | 37 | 2,169.5 s |
| 9 | FaCT++ | 244 / 298 | 54 | 2,671.9 s |
| 10 | Racer | 237 / 298 | 61 | 1,322.2 s |
| 11 | Chainsaw | 191 / 298 | 107 | 1,587.4 s |
| 12 | JFact | 189 / 298 | 109 | 2,404.3 s |
| 13 | jcel | 133 / 298 | 165 | 98.4 s |

Discipline: **OWL DL Classification** (finished)

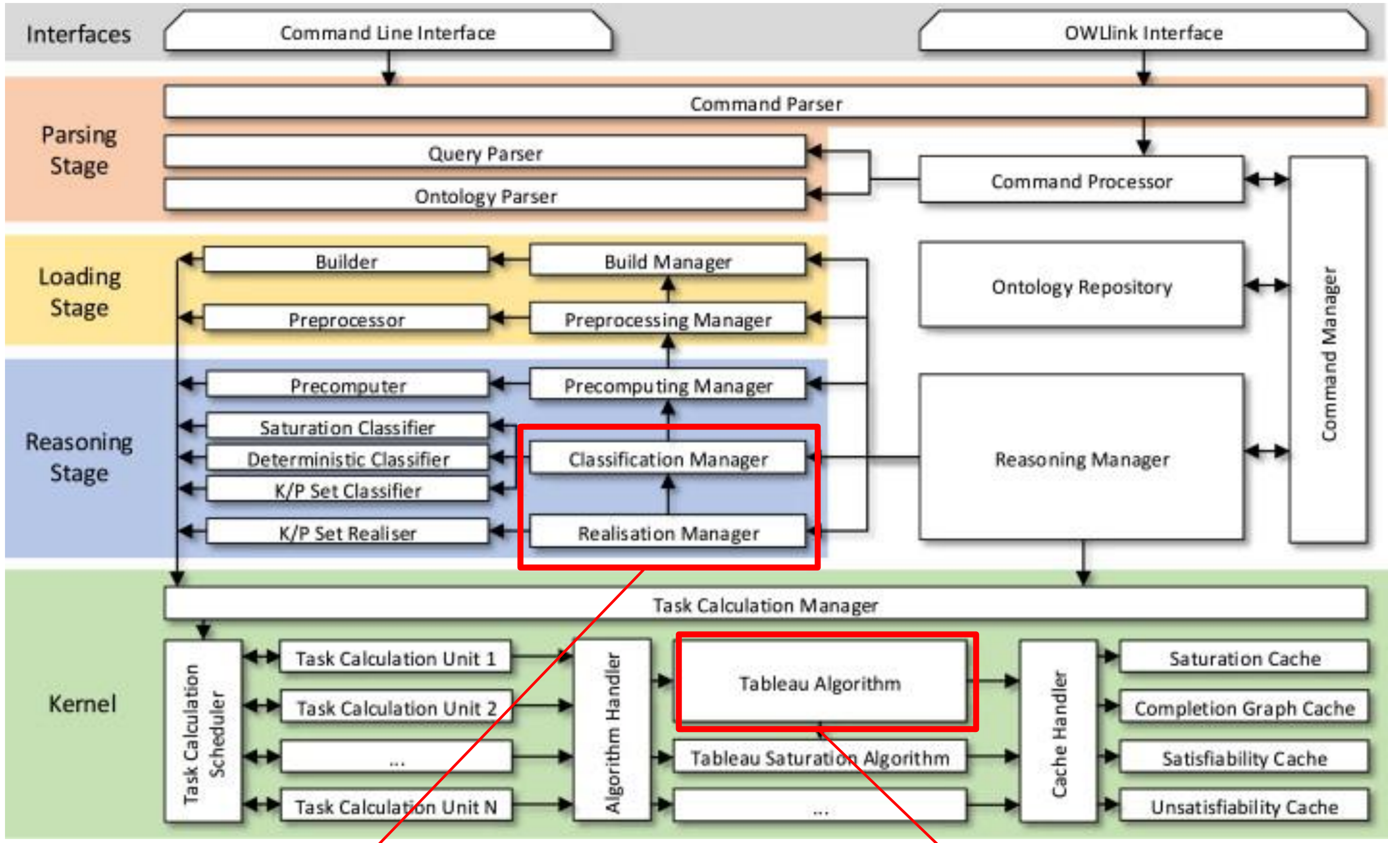
| Rank | Reasoner | Score | ! | Time |
|---|---|---|---|---|
| 1 | Konclude | 288 / 306 | 18 | 1,308.9 s |
| 2 | MOReHermiT | 247 / 306 | 59 | 2,143.0 s |
| 3 | HermiT-OA4 | 237 / 306 | 69 | 5,808.2 s |
| 4 | HermiT | 236 / 306 | 70 | 5,416.4 s |
| 5 | TrOWL | 201 / 306 | 105 | 971.1 s |
| 6 | FaCT++ | 200 / 306 | 106 | 1,361.3 s |
| 7 | Pellet-OA4 | 187 / 306 | 119 | 2,179.3 s |
| 8 | Racer | 164 / 306 | 142 | 1,103.8 s |
| 9 | JFact | 128 / 306 | 178 | 889.5 s |
| 10 | Chainsaw | 119 / 306 | 187 | 1,709.0 s |

**LEGEND:** Score: correctly solved / number of benchmarks !: errors + timeouts + unexpected Time: time for correctly solved benchmarks

Progress: correctly solved errors + timeouts + unexpected Running: Finished:

*(time in seconds)*

# Konclude




Complex reasoning services

Heart of the reasoner.


- Parallel high-performance reasoner for OLW 2 / DL $\mathcal{SROIQ}(\mathcal{D})$.
- Implemented in C++.
- Uses highly optimized tableau algorithm assisted by saturation procedures.
- Offers GNU free software license and interface to OWL API.
- http://derivo.de/en/products/konclude/

# Agenda

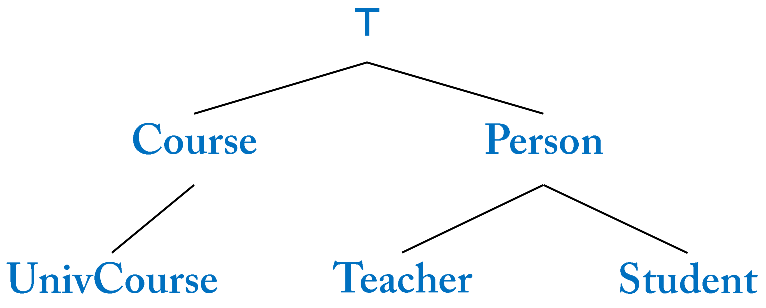
# Reminder

- A reasoning service is an algorithm providing a decision procedure to a logical problem.
- An algorithm is a decision procedure for a problem if it solves the problem with a yes/no answer.
  - It is sound : when the algorithm answers yes, this answer is always correct.
  - It terminates : it stops after a finite number of steps with some answer.
  - It is complete : for any problem which has a positive answer, the algorithm will find it.
- For a logic, the basic decision problem is to determine if a well-formed formula is valid. There are however other interesting questions (entailment, satisfiability … ).
- Consider a procedure p deciding entailment in a knowledge base KB :
  - p is sound if it does not make wrong inferences : if $KB_p \models \alpha$ then $KB \models \alpha$.
  - p is complete if it does make all the correct inferences : if $KB \models \alpha$ then $KB_p \models \alpha$.

# Useful reasoning services for a description logic ?

- ❑ Checking a description logic KB is not as easy as checking links in a semantic network. Reasoning services are useful for inferences, but also for knowledge validation.

TBox $\mathcal{T}$ = {Course $\sqsubseteq \neg$Person,
UnivCourse $\sqsubseteq$ Course,
Teacher $\equiv$ Person $\sqcap\ \exists$teaches.Course,
$\exists$teaches.$\top \sqsubseteq$ Person,
Student $\equiv$ Person $\sqcap\ \exists$attends.Course,
$\exists$attends.$\top \sqsubseteq$ Person}




ABOX $\mathcal{A}$ = {Mary : Person; Logic : Course; (Mary, Logic) : teaches}.

- ❑ Can we check the subsumption hierarchy ? So far, easily.
  - If we add Professor $\equiv \exists$teaches.UnivCourse ? Less easy but we can see that Professor $\sqsubseteq$ Person (why ?).
  - If we add LazyStudent $\sqsubseteq \forall$attends.$\neg$Course, is a Lazy Student a Student ?

No, $\mathcal{T}$ does not entail LazyStudent $\sqsubseteq$ Student !

- ❑ Can we check individual instances ? Is Mary a Teacher ? (Yes).

*(after Baader et al. 2017)*

# Useful reasoning services for a description logic ?

Given an ontology $\mathcal{O}$ (or a knowledge base $\mathcal{K}$) = $<\mathcal{T}, \mathcal{A}>$, the main reasoning services check :

- For a TBox
  - Concept Satisfiability : C is satisfiable w.r.t. $\mathcal{O}$ (check if knowledge is meaningful).
  - Subsumption : $\mathcal{O} \models C \sqsubseteq D$ (check if knowledge is correct; build classification).
  - Equivalence : $\mathcal{O} \models C \equiv D$ (check if knowledge is minimally redundant).
- For an ABox
  - Instance of a concept : $\mathcal{O} \models a : C$ (check instantiation).
  - Instance of a role : $\mathcal{O} \models (a, b) : r$ (check if a role holds between individuals).
- For an Ontology / KB
  - Ontology consistency : $\mathcal{O}$ has at least one model.
  - Ontology coherence : all named concepts in $\mathcal{O}$ are satisfiable.

# Semantics for reasoning services

Given an ontology $\mathcal{O}$ (or a knowledge base $\mathcal{K}$) = $<\mathcal{T}, \mathcal{A}>$ :

- For a TBox
  - Concept Satisfiability : C is satisfiable w.r.t. $\mathcal{O}$ iff there is a model $\mathcal{I}$ of $\mathcal{O}$ such as $C^{\mathcal{I}} \neq \emptyset$.
  - Subsumption : $\mathcal{O} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{O}$.
  - Equivalence : $\mathcal{O} \models C \equiv D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{O}$.
- For an ABox
  - Instance of a concept : $\mathcal{O} \models a : C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{O}$.
  - Instance of a role : $\mathcal{O} \models (a, b) : r$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{O}$.

# Complex reasoning services

- Classification : classifying an ontology $\mathcal{O}$ is a reasoning service consisting of :
  1. Testing whether $\mathcal{O}$ is consistent; if yes, then:
  2. Checking, for each pair A, B of class names in $\mathcal{O}$ plus Thing, Nothing, if $\mathcal{O} \models A \subseteq B$
  3. Checking, for each individual name b and class name A in $\mathcal{O}$, whether $\mathcal{O} \models b : A$

  and returning the result in a suitable form : $\mathcal{O}$'s inferred class hierarchy.

- Instance retrieval : given a concept C, finds in the ontology all individuals a such as:

  $$\mathcal{O} \models a : C.$$

- Realization : given an individual a, finds the most specific concept C such as

  $$\mathcal{O} \models a : C.$$

# Example of classification

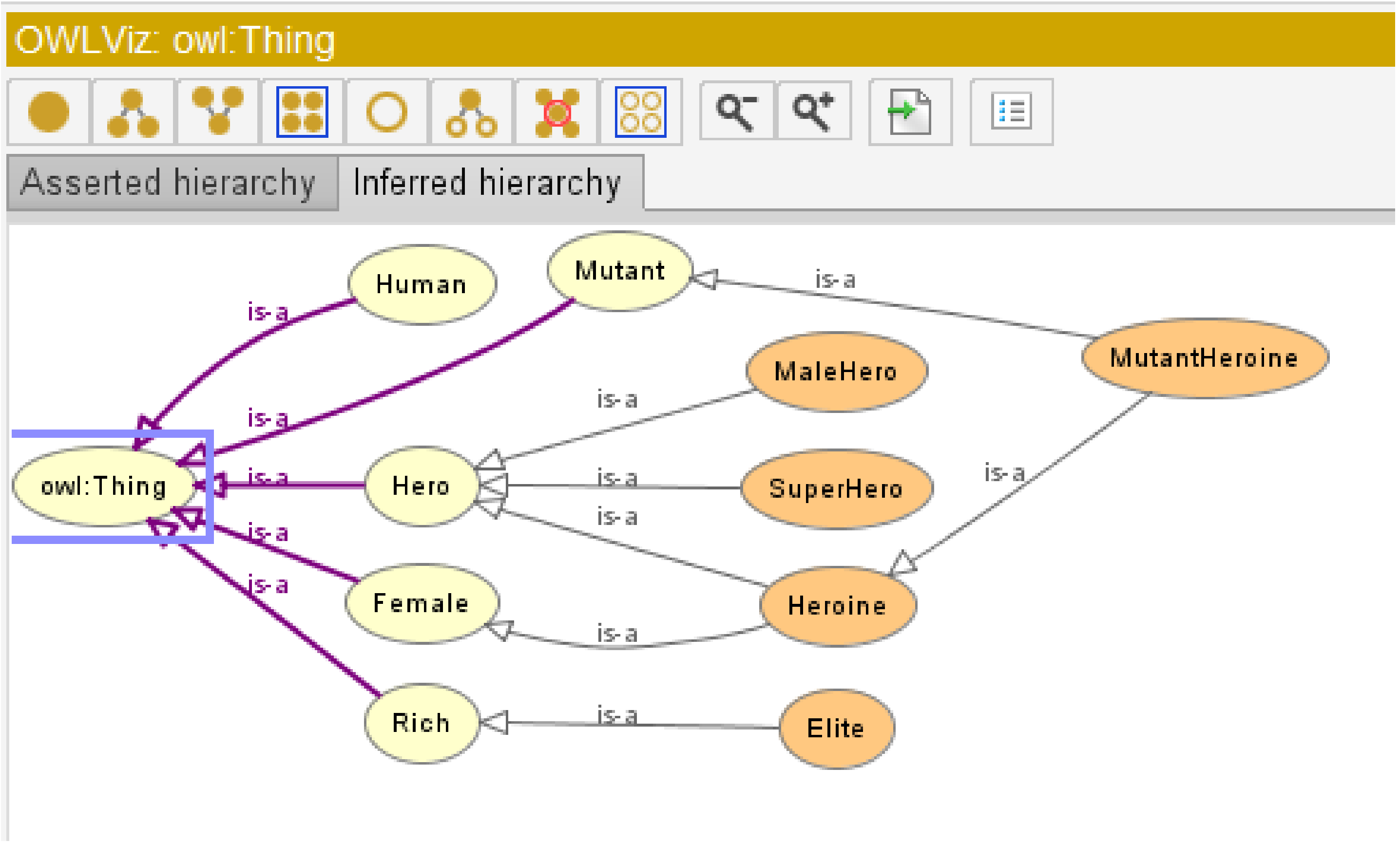
Heroine ≡ Hero ⊓ Female

MaleHero ≡ Hero ⊓ ¬ Female

MutantHeroine ≡ Heroine ⊓ Mutant

Elite ≡ Rich ⊓ ¬ Human

Superhero ≡ Hero ⊓ Elite



# Reduction of reasoning services to ontology consistency

For any $\mathcal{ALC}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, concepts C, D and individual name a, the following holds :

- The reasoning tasks for concepts can be answered by a decision procedure for concept satisfiability :

1. C is subsumed by D w.r.t. $\mathcal{O}$, or $\mathcal{O} \vDash C \sqsubseteq D$, iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{O}$;
2. C and D are equivalent w.r.t. $\mathcal{O}$, or $\mathcal{O} \vDash C \equiv D$, iff both $C \sqcap \neg D$ and $\neg C \sqcap D$ are unsatisfiable w.r.t. $\mathcal{O}$.

- Concept satisfiability, instance checking, ontology coherence can be reduced to ontology consistency :

3. C is satisfiable w.r.t. $\mathcal{O}$ iff $\{\mathcal{T}, (a : C)\}$ is consistent.
4. a is an instance of C w.r.t. $\mathcal{O}$, or $\mathcal{O} \vDash a : C$ iff $\mathcal{O} \cup \{\neg (a : C)\}$ is inconsistent(*).
5. $\mathcal{O}$ is coherent iff, for each concept name A, $\mathcal{O} \cup \{a : A\}$ is consistent(*).

A decision procedure for ontology consistency will decide all standard $\mathcal{ALC}$ inference problems.

* : $\mathcal{O} \cup \{(a : C)\}$ is a shorthand for $\{\mathcal{T}, \mathcal{A} \cup \{(a : C)\}$

# Agenda

| | |
|---|---|
| 1 | How good are reasoners ? |
| 2 | Reasoning services |
| 3 | Structural subsumption |
| 4 | Tableau algorithm for $\mathcal{ALC}$ |
| 5 | Observations on complexity |

# Structural subsumption algorithms

- Some description logics do not support negation.
- For such DLs, concept subsumption can usually be computed by structural subsumption algorithms.

  structural : comparing the syntactic structure of normalized concept descriptions.

- They are very efficient but only complete for simple languages with little expressivity.

# Structural subsumption algorithm

- We consider as example the small language $\mathcal{FL}_0$ :
  - Limited to A | C ⊓ D | ∀r.C .
  - For this presentation, considered without a TBox (cf. next section for TBox elimination).
- The structural subsumption algorithm has two phases :

1. Normalization of concept expressions.
   - A $\mathcal{FL}_0$ concept expression is in normal form iff it is in the form $A_1 \sqcap \ldots \sqcap A_m \sqcap \forall r_1.C_1 \sqcap \ldots \sqcap \forall r_n.C_n$

   Ai are distinct concept names, rj distinct role names and Ck concept expressions in normal form.

   - To reduce to normal form :

   a) Flatten all embedded conjunctions: A ⊓ (B ⊓ C) => A ⊓ B ⊓ C

   b) Factorize all conjunctions of universal quantifiers over the same role : ∀r.C ⊓ ∀r.D => ∀r.(C ⊓ D)

# Structural subsumption algorithm

2. Recursive comparison of expressions :

Let $C = C_1 \sqcap \ldots \sqcap C_n$ and $D = D_1 \sqcap \ldots \sqcap D_m$ be two concepts in normal form :

**Subsumes?**(C, D) tests $D \sqsubseteq C$ and returns true iff for all $C_i \in C$ :

If Ci is an atomic concept name, there exists a Dj such as Dj = Ci;

If Ci is of the form $\forall r.C'$, there exists a Dj of the form $\forall r.D'$ such as **Subsumes?**(C', D') is true.

- Time complexity : $\mathcal{O}(|C| \times |D|)$.
- Soundness : whenever the algorithm answers "yes", then $D \sqsubseteq C$ (shown by induction).
- Completeness : whenever $D \sqsubseteq C$, the algorithm answers "yes".

# Example

- Check if the following subsumption is valid :

  $\forall$child.Adult $\sqcap$ $\forall$child.Male $\sqsubseteq$ $\forall$child.Adult

1. Normalisation

   $\forall$child.(Adult $\sqcap$ Male) $\sqsubseteq$ $\forall$child.Adult

2. Subsumption check

   Second rule: check if Adult $\sqcap$ Male $\sqsubseteq$ Adult

   First rule: Adult is present on both sides of $\sqsubseteq$

   => success

# Limits of structural subsumption

- Algorithms based on syntactic analysis cannot handle more complex logics.

  In particular, DLs with negation and disjunction cannot be handled by structural subsumption algorithms.

  - For instance, $A \sqcup \neg A$ subsumes any concept $C$ even if $C$ is not mentioned in $A \sqcup \neg A$.

- Without negation, the relationship between the $\neq$ reasoning services is lost.

# Agenda

# Tableau algorithm for $\mathcal{ALC}$ : basic ideas

- The basic algorithm works on ABoxes only (without TBoxes).
  - It is easier to formulate algorithms of this kind by assuming that the TBox is empty.
  - We can eliminate the TBox for acyclic terminologies.
- All concept descriptions must be in negative normal form.
- The algorithm tries to decide ontology consistency by trying to construct a model :
  - If successful, a model exists => the ontology is consistent.
  - If not successful, the algorithm will terminate with failure.
- The algorithm can be initiated in different ways to deal with different decision problems.

Warning : the algorithm is not tractable (does not execute in polynomial time). We will discuss its complexity later.

# Eliminating the TBox

- We can eliminate the TBox for acyclic terminologies.
- To expand an acyclic terminology :
  - Replace each occurrence of a name on the right-hand side of a definition with its definition, until no named concept remains on the right-hand side of the terminology.
  - For a finite acyclic terminology that process will always terminate.
  - As the process uses equivalent substitutions, the TBox and its expansion have the same models.

**TBox**

Woman ≡ Person ⊓ Female
Man ≡ Person ⊓ ¬Woman
Mother ≡ Woman ⊓ ∃hasChild.Person
Father ≡ Man ⊓ ∃hasChild.Person

**Expanded TBox**

Woman ≡ Person ⊓ Female
Man ≡ Person ⊓ ¬ (Person ⊓ Female)
Mother ≡ (Person ⊓ Female) ⊓ ∃hasChild.Person
Father ≡ (Person ⊓ ¬ (Person ⊓ Female)) ⊓ ∃hasChild.Person

# Eliminating the TBox ./.

- The concept expansion process preserves all logical inferences, and thus allows to eliminate the TBox in reasoning problems for acyclic terminologies.
- Let us consider any concept C in an acyclic terminology and its expansion C' :
- *C is satisfiable w.r.t. the TBOX ($\mathcal{T}$) iff C' is satisfiable* :
  - By construction $C \equiv_{\mathcal{T}} C'$.
  - C' does not contain any defined names in its definition, so does no longer depends on $\mathcal{T}$.
  - Hence C' is satisfiable w.r.t. $\mathcal{T}$ iff it is satisfiable.
- With similar arguments we can show that
  - $\mathcal{T} \models C \sqsubseteq D$ *iff* $\models C' \sqsubseteq D'$;
  - $\mathcal{T} \models C \equiv D$ *iff* $\models C' \equiv D'$;
  - C and D are disjoint w.r.t. $\mathcal{T}$ iff C' and D' are disjoint.

# Reduction to negation normal form

- A concept is in Negation Normal Form (NNF) if all occurrences of negations are pushed inwards in front of the concept names.

  Example : $(\exists r.A) \sqcap (\exists r.B) \sqcap \neg(\exists r.(A \sqcap B))$ becomes $(\exists r.A) \sqcap (\exists r.B) \sqcap \forall r.(\neg A \sqcup \neg B)$

- Every $\mathcal{ALC}$ concept can be transformed in NNF using the following rules:
  - $\neg \top \equiv \bot$
  - $\neg \bot \equiv \top$
  - $\neg \neg C \equiv C$
  - $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$ (De Morgan's laws)
  - $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$ (De Morgan's laws)
  - $\neg \forall r.C \equiv \exists r.\neg C$ (Generalized De Morgan's laws)
  - $\neg \exists r.C \equiv \forall r.\neg C$ (Generalized De Morgan's laws)
  - $C \sqsubseteq D \equiv \neg C \sqcup D$

# Tableau decision – example 1

- Is the concept (∀hasChild.Male) ⊓ (∃hasChild.¬Male) satisfiable ?
- We create the ABox $\mathcal{A}_0$ = {x : ((∀hasChild.Male) ⊓ (∃hasChild.¬Male))} and check if it is consistent.

| | | |
|---|---|---|
| 1. | Initial ABox $\mathcal{A}_0$ | x : ((∀hasChild.Male) ⊓ (∃hasChild.¬Male)) |
| 2. | From (1), we add the axiom(s) | x : (∀hasChild.Male) |
| 3. | From (1), we add | x : (∃hasChild.¬Male) |
| 4. | From (3), we add | (x, y) : hasChild and y : ¬Male for a new y |
| 5. | From (2) and (4), we add | y : Male |
| 6. | From (4) and (5) | contradiction : y : Male and y : ¬Male |

- The ABox is not consistent. Hence the concept is not satisfiable.

# Tableau decision – example 2

- Is the concept $\forall r.(\neg C \sqcup D) \sqcap \exists r.(C \sqcap D)$ satisfiable ?

| | | | |
|---|---|---|---|
| 1. | Initial ABox $\mathcal{A}_0$ | $x : (\forall r.(\neg C \sqcup D) \sqcap \exists r.(C \sqcap D))$ | |
| 2. | From (1), we add the axiom(s) | $x : \forall r.(\neg C \sqcup D)$ | |
| 3. | From (1), we add | $x : \exists r.(C \sqcap D)$ | |
| 4. | From (3), we add | $(x, y) : r$ and $y : C \sqcap D$ | for a new $y$ |
| 5. | From (4), we add | $y : C$ | |
| 6. | From (4), we add | $y : D$ | |
| 7. | From (2), we add | $y : \neg C \sqcup D$ | |
| 8. | Two possibilities : we create a branching | | |
| a) | From (7), we add | $y : \neg C$ | contradiction: $y : \neg C$ (8.a) and $y : C$ (5) |
| b) | From (7), we add | $y : D$ | no new axiom can be generated. |

- 8.b yields a satisfying model : $\Delta^{\mathcal{I}} = \{x, y\}$ ; $C^{\mathcal{I}} = \{y\}$ ; $D^{\mathcal{I}} = \{y\}$; $r^{\mathcal{I}} = \{(x, y)\}$. The ontology is consistent and the concept satisfiable.

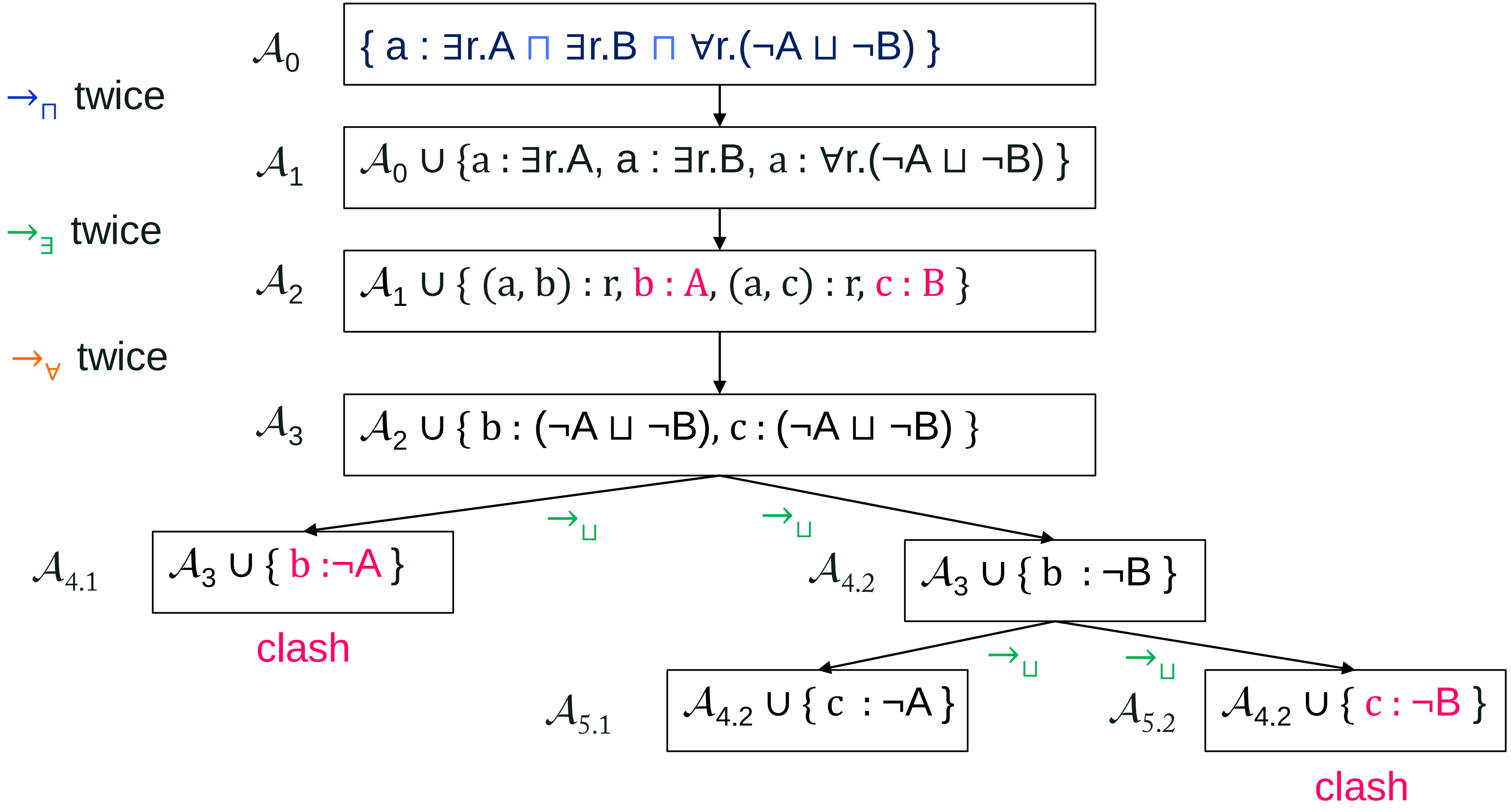
# The tableau decision algorithm

- Works on an ABox (set of assertion axioms) $\mathcal{A}$.
  - For checking the satisfiability of a concept C, it starts with { x : C }.
- Applies decision rules (in arbitrary order) to infer new constraints on the axioms :
  - These constraints are expressed as new assertion axioms of the form y : C or (x, y) : r.
  - The new axioms are added to $\mathcal{A}$, yielding one or two (in case of branching) derived ABox(es).
  - The algorithm is applied on each derived ABox.
  - Its stops when no rule can be further applied.
  - Optimization : the algorithm can stop applying rules to any branch containing a clash.
- Will answer that $\mathcal{A}$ is consistent iff rule application leads to one ABox that is :
  - Complete : no more rule can be applied to it, and
  - Clash-free : it does not contain any pairs of assertions of the form {a : C, a : ¬C}.
- Will answer that $\mathcal{A}$ is inconsistent if that is not possible.

# The tableau decision rules

- $\sqcap$-rule $\rightarrow_{\sqcap}$ :
  - if $x : C1 \sqcap C2 \in \mathcal{A}$ and $\{x : C1, x: C2\} \not\subseteq \mathcal{A}$ then replace $\mathcal{A}$ with $\mathcal{A} \cup \{x: C1, x: C2\}$
- $\sqcup$-rule $\rightarrow_{\sqcup}$ :
  - if $x : C1 \sqcup C2 \in \mathcal{A}$ and $\{x : C1, x : C2\} \cap \mathcal{A} = \emptyset$
    then create two branches replacing $\mathcal{A}$ with $\mathcal{A} \cup \{x : C1\}$ and with $\mathcal{A} \cup \{x : C2\}$ respectively
- $\exists$-rule $\rightarrow_{\exists}$ :
  - if $x : \exists r.C \in \mathcal{A}$ and there is no z such as $\{(x, z) : r, z : C\} \subseteq \mathcal{A}$
    then create a new individual name y and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(x, y) : r, y : C\}$
- $\forall$-rule $\rightarrow_{\forall}$ :
  - if $\{x : \forall r.C, (x, y) : r\} \subseteq \mathcal{A}$ and $y : C \notin \mathcal{A}$ then replace $\mathcal{A}$ with $\mathcal{A} \cup \{y : C\}$

# A tableau derivation




5.1 yields a satisfying model : $\Delta^{\mathcal{I}} = \{a, b, c\}$ ; $A^{\mathcal{I}} = \{b\}$ ; $B^{\mathcal{I}} = \{c\}$; $r^{\mathcal{I}} = \{(a, b), (a, c)\}$.

# The tableau decision algorithm – formal version

**function** consistent ($\mathcal{A}$) returns *True* or *False*

  **Input:** a normalized $\mathcal{ALC}$ ABox $\mathcal{A}$

  **if** expand($\mathcal{A}$) ≠ 0 **then return** *True*

  **else return** *False*

**function** expand ($\mathcal{A}$) returns an expanded ABox or 0

  **Input:** a normalized $\mathcal{ALC}$ ABox $\mathcal{A}$

  {R, α} = selectRule($\mathcal{A}$)

  // the function selectsRule selects a rule R applicable to a (pair of) assertion(s) α of $\mathcal{A}$ . If $\mathcal{A}$ is complete, R = 0.

  **if** R ≠ 0 **then**

    // the function applyRule applies R to α and the ABox $\mathcal{A}$ and returns the set of resulting ABoxes.

    **if** there is an ABox $\mathcal{A}'$ ∈ applyRule(R, α, $\mathcal{A}$) such that expand($\mathcal{A}'$) ≠ 0 then

      **return** expand($\mathcal{A}'$)

    **else return** 0

  **else if** $\mathcal{A}$ contains a clash **then return** 0

    **else return** $\mathcal{A}$

# Tableau algorithm properties : local correctness

- Local correctness property : any derivation of an ABox by applying the rules preserves consistency.

This is easy to verify :

- If a set of axioms $\mathcal{A}'$ has been produced from $\mathcal{A}$ by applying a rule $\rightarrow_{\sqcap}$, $\rightarrow_{\forall}$ or $\rightarrow_{\exists}$, $\mathcal{A}$ is obviously satisfiable iff $\mathcal{A}'$ is satisfiable;

- If a set of axioms $\mathcal{A}'$ has been produced from $\mathcal{A}$ by applying $\rightarrow_{\sqcup}$, $\mathcal{A}$ is satisfiable iff one of the two branches is satisfiable.

# Tableau algorithm: soundness

If the algorithm concludes that a set of axioms $\mathcal{A}$ is consistent, it is consistent.

High level proof :

- If the algorithm concludes positively there is a clash-free ABox $\mathcal{A}_n$ derived from $\mathcal{A}$ for which no rule is applicable.
- We can use $\mathcal{A}_n$ to construct an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ :
  - $\Delta^{\mathcal{I}}$ contains all individuals in $\mathcal{A}_n$;
  - for $x \in \Delta^{\mathcal{I}}$ and a concept name $C$, $x \in C^{\mathcal{I}}$ iff $x : C$ is in $\mathcal{A}_n$;
  - for $x, y \in \Delta^{\mathcal{I}}$ and a role name $r$, $(x, y) \in r^{\mathcal{I}}$ iff $(x, y) : r$ is in $\mathcal{A}_n$.
- By construction $\mathcal{I}$ satisfies all role assertions of $\mathcal{A}_n$. One can prove by induction on the structure of the concepts that $\mathcal{I}$ satisfies all concept assertions of $\mathcal{A}_n$ too.
- As all axioms of $\mathcal{A}$ are in $\mathcal{A}_n$, $\mathcal{I}$ also satisfies $\mathcal{A}$.

# Tableau algorithm: termination

For any initial ABox (set of axioms) $\mathcal{A}_0$, the algorithm will never generate an infinite sequence $\mathcal{A}_0, \mathcal{A}_1 \ldots \mathcal{A}_i, \mathcal{A}_{i+1}$ such as each $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by application of the rules :

- All rules but $\rightarrow_\forall$ are never applied twice to the same constraint.
- $\rightarrow_\forall$ is never applied to an individual x more times than the number of direct successors of x (i.e., y such that (x, y) : r) present in $\mathcal{A}_i$.

  This process is bounded by the size of $\mathcal{A}_i$.
- Each rule application to a constraint y : C adds axioms of type z : D such that D is a subconcept of C. This process is bounded by the size of C.

# Completeness

Any consistent ABox $\mathcal{A}$ will receive a positive answer through the algorithm.

This is easy to verify as :

- As the algorithm terminates, it will always give a positive or negative answer.
- If the input $\mathcal{A}$ is complete (cannot be expanded further), the answer is immediate.
- If A can be expanded further, the answer follows from the local correctness property : any application of the rules will preserve consistency.

Starting with a consistent ABox, the expansion process will terminate with a consistent and complete ABox.

# Using the algorithm for various inference problems

In order to check :

- Consistency of an ABox $\mathcal{A}$, check if $\{\mathcal{A}\}$ is consistent.
- Satisfiability of a concept C, check if $\{ a : C \}$ is consistent;
- Satisfiability of a concept C w.r.t. ontology $\mathcal{O}$, check if $\mathcal{O} \cup \{ a : C \}$ is consistent;
- Subsumption $C \sqsubseteq D$, check if $\{ a : C \sqcap \neg D \}$ is not consistent;
- Subsumption $C \sqsubseteq D$ w.r.t. ontology $\mathcal{O}$ check if $\mathcal{O} \cup \{ a : C \sqcap \neg D \}$ is not consistent;
- Whether b is an instance of C w.r.t. ontology $\mathcal{O}$, check if $\mathcal{O} \cup \{ b : \neg C \}$ is not consistent.

# Extending the algorithm to TBoxes

- We cannot fully eliminate the TBox if it contains general inclusion axioms of the form $C \sqsubseteq D$ where both $C$ and $D$ can be complex concept descriptions.

- To cover this case, we add a rule for general inclusions.

  - Basic approach : $\mathcal{I} \models C \sqsubseteq D$ iff $\mathcal{I} \models \top \sqsubseteq \neg C \sqcup D$, hence for any $x$ in $\mathcal{A}$, $x$ must belong to $\neg C \sqcup D$.

  - If $C$ is an atomic concept name, it is easier to check if $C$ is satisfied by checking $x : C \in \mathcal{A}$.
    In that case we only need to check that $D$ will be satisfied as well without introducing branches.

GCI-rule $\rightarrow_{GCI}$ : if $C \sqsubseteq D \in \mathcal{T}$

  - If $C$ is a concept name and $x : C \in \mathcal{A}$ but $x : D \notin \mathcal{A}$, then replace $\mathcal{A}$ with $\mathcal{A} \cup \{ x : D \}$
  - Else if $x$ occurs in $\mathcal{A}$ and $x : \neg C \sqcup D \notin \mathcal{A}$, replace $\mathcal{A}$ with $\mathcal{A} \cup \{ x : \neg C \sqcup D \}$

- However, the algorithm with this rule no longer terminates !

  - Example : apply the algorithm to $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ with $\mathcal{T} = \{A \sqsubseteq \exists r.A\}$ and $\mathcal{A} = \{a : A\}$.

  - $\{a : A\} \rightarrow_{GCI} \{a : \exists r.A\} \rightarrow_{\exists} \{b : A\} \rightarrow_{GCI} \{ b : \exists r.A\} \ldots$

# Blocking rules

- To avoid cycles, we will introduce blocking rules. Basic intuition:
  - If we have introduced before an individual $x$ belonging to a concept $C$, we should not introduce a new individual $y$ belonging to a concept description which is a subpart of the same concept $C$.
- Blocking rule : in the case when :
  - One of the rules introduces a new individual $y$;
  - And there is in the ABox $\mathcal{A}$ an older individual $x$ (introduced before $y$);
  - And $\{C \mid y : C \in \mathcal{A}\} \subseteq \{D \mid x : D \in \mathcal{A}\}$;

  We will say that $x$ blocks $y$ : no rule can be applied to $y$.
- It can be verified that with appropriate blocking the tableau algorithm will always terminate with general inclusions.

# The tableau decision rules with GCIs

- $\sqcap$-rule $\rightarrow_{\sqcap}$
  - if $x : C1 \sqcap C2 \in \mathcal{A}$, x not blocked and $\{x : C1, x : C2\} \not\subseteq \mathcal{A}$ then replace $\mathcal{A}$ with $\mathcal{A} \cup \{x : C1, x : C2\}$
- $\sqcup$-rule $\rightarrow_{\sqcup}$
  - if $x : C1 \sqcup C2 \in \mathcal{A}$, x not blocked and $\{x : C1, x : C2\} \cap \mathcal{A} = \emptyset$
  - then create two branches replacing $\mathcal{A}$ with $\mathcal{A} \cup \{x : C1\}$ and with $\mathcal{A} \cup \{x : C2\}$ respectively
- $\exists$-rule $\rightarrow_{\exists}$
  - if $x : \exists r.C \in \mathcal{A}$, x not blocked and there is no z such as $\{(x, z) : r, z : C\} \subseteq \mathcal{A}$
  - then create a new individual name y and replace $\mathcal{A}$ with $\mathcal{A} \cup \{(x, y) : r, y : C\}$
- $\forall$-rule $\rightarrow_{\forall}$
  - if $\{x : \forall r.C, (x, y) : r\} \subseteq \mathcal{A}$, x not blocked and $y : C \notin \mathcal{A}$ then replace $\mathcal{A}$ with $\mathcal{A} \cup \{y : C\}$
- GCI-rule $\rightarrow_{GCI}$ :if $C \sqsubseteq D \in \mathcal{T}$ and x is not blocked
  - If C is a concept name, $x : C \in \mathcal{A}$ but $x : D \notin \mathcal{A}$, then replace $\mathcal{A}$ with $\mathcal{A} \cup \{x : D\}$
  - Else if $x : \neg C \sqcup D \notin \mathcal{A}$ for x in $\mathcal{A}$, replace $\mathcal{A}$ with $\mathcal{A} \cup \{x : \neg C \sqcup D\}$

# Agenda

# Reminder

- **P** : the class of decision problems that are decided by a Turing machine in Polynomial time (whose time complexity function is bounded by a polynomial function).
- **NP** : the class of decision problems that are decided by a Nondeterministic Turing machine in Polynomial time.
- **EXPTIME** : the class of decision problems decided by a deterministic Turing machine whose time complexity function is bounded by an exponential function ( $\mathcal{O}(2^{p(n)})$ where $p(n)$ is a polynomial function of $n$).
- **PSPACE** : the class of decision problems decided by a deterministic Turing machine whose space complexity function is bounded by a polynomial function.
- **EXPSPACE** : the class of all decision problems solvable by a deterministic Turing machine in $\mathcal{O}(2^{p(n)})$ space, where $p(n)$ is a polynomial function of $n$.

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$$

# Complexity of the tableau algorithm for $\mathcal{ALC}$

- $\mathcal{ALC}$ with acyclic TBoxes :
  - The algorithm as explained needs exponential space and time.
  - By various optimizations it can be modified such as both concept satisfiability and ontology consistency are PSPACE-complete(*).
- $\mathcal{ALC}$ , TBoxes with general inclusions :
  - Both concept satisfiability and ontology consistency for $\mathcal{ALC}$ are EXPTIME-complete(*).
- The complexity varies with the syntactic constructs included in the DL :
  - The algorithms for OWL DL have NEXPTIME (nondeterministic exponential) complexity, although best reasoners offer acceptable performances for real life problems.
  - The profiles of OWL 2, such as OWL EL, offer tractable algorithms (executable in polynomial time) which can be used for large ontologies (this is indeed the main reason for their creation).

*: a decision problem is complete for a complexity class C if it is in that class and any other problem in C can be reduced to it by a (polynomial) transformation. It is an example of the hardest problems in C.

# The description logic complexity navigator

## Complexity of reasoning in Description Logics

**Note: the information here is (always) incomplete and updated often**

Base description logic: $\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplements

$\mathcal{ALC}$ ::= ⊥ | ⊤ | $A$ | ¬$C$ | $C \sqcap D$ | $C \sqcup D$ | ∃$R.C$ | ∀$R.C$

**Concept constructors:**

- ☐ $\mathcal{F}$ – functionality[2]: $(\leq 1\ R)$
- ☐ $\mathcal{N}$ – (unqualified) number restrictions: $(\geq n\ R)$, $(\leq n\ R)$
- ☐ $\mathcal{Q}$ – qualified number restrictions: $(\geq n\ R.C)$, $(\leq n\ R.C)$
- ☐ $\mathcal{O}$ – nominals: $\{a\}$ or $\{a_1, \ldots, a_n\}$ ("one-of")
- ☐ $\mu$ – least fixpoint operator: $\mu X.C$

Forbid complex roles[5] in number restrictions[6]

**Role constructors:** trans reg

- ☐ $\mathcal{I}$ – role inverse: $R^-$
- ☐ ∩ – role intersection[3]: $R \cap S$
- ☐ ∪ – role union: $R \cup S$
- ☐ ¬ – role complement: $\neg R$ full
- ☐ o – role chain (composition): $R \circ S$
- ☐ * – reflexive-transitive closure[4]: $R^*$
- ☐ *id* – concept identity: $id(C)$

**TBox (concept axioms):**

- ○ empty TBox
- ○ acyclic TBox ($A \equiv C$, $A$ is a concept name; no cycles)
- ◉ general TBox ($C \sqsubseteq D$, for arbitrary concepts $C$ and $D$)

**RBox (role axioms):** OWL-Lite OWL-DL OWL 1.1

- ☐ $\mathcal{S}$ – role transitivity: Tr($R$)
- ☐ $\mathcal{H}$ – role hierarchy: $R \sqsubseteq S$
- ☐ $\mathcal{R}$ – complex role inclusions: $R \circ S \sqsubseteq R$, $R \circ S \sqsubseteq S$
- ☐ $s$ – some additional features (click to see them)

Reset You have selected a Description Logic: $\mathcal{ALC}$

| **Complexity[7] of reasoning problems[8]** | | |
|---|---|---|
| Concept satisfiability | **ExpTime-complete** | • Hardness: originally proved in [77]; see also [2, Theorem 3.27].<br>• Upper bound: an ExpTime tableaux algorithm is given in [33]. |
| ABox consistency | **ExpTime-complete** | • Hardness follows from ExpTime-hardness of concept satisfiability w.r.t. general TBoxes.<br>• Upper bound even for $\mathcal{SHIQ}$ was proved in [12, Corollary 6.30]. |
| **Important properties of the Description Logic** | | |
| Finite model property | **Yes** | For all sublogics of $\mathcal{SHOQ}$. This is mentioned in [63], where a similar result is obtained in Corollary 4.3 for $\mathcal{SHOQ}$ extended with concrete domains and keys. (I did not find a "proper" reference for $\mathcal{SHOQ}$ or its sublogics.) |
| Tree model property | **Yes** | For all sublogics of $\mathcal{ALCFI}_{reg}$ with any TBoxes; see [2, p.189, Theorem 5.6]. |

Maintained by: Evgeny Zolin
Please see the list of updates

Any comments are welcome: EZolin@cs.man.ac.uk

*(http://www.cs.man.ac.uk/~ezolin/dl/)*

# Summary

- All important inference problems for decision logics can be reduced to a decision procedure for ontology consistency.
- Syntax-based decision procedures, such as structural subsumption, are performant but only applicable to very simple decision logics without negation.
- Acyclic terminologies for $\mathcal{ALC}$ can be handled by eliminating the TBox and using a tableau algorithm looking for a complete derivation without clash. This procedure is sound and complete.
- Extending the algorithm to TBoxes with general inclusions requires the addition of blocking rules to ensure termination.
- The complexity of the $\mathcal{ALC}$ decision algorithm with general TBoxes is EXPTIME-complete, hence not tractable. However existing reasoners are subject to constant improvements (and competition) and are usable in real-life situations.
- The specific sub profiles defined for OWL 2 (EL, QL, RL) have tractable decision algorithms for their most important tasks.

# References


- *[Baader and Nutt 2003]: Baader F. and Nutt W.*, Basic description logic, *in Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. (eds.):* The Description Logic Handbook: Theory, Implementation, and Applications, *chapter 2, Cambridge University Press, New York, NY, USA, 2003.*
- *[Baader et al. 2017]: Baader, F., Horrocks, I. Lutz C. and Sattler, U.*, An introduction to Description Logic, *Cambridge University Press, 2017.*

# THANK YOU