Knowledge representation and semantic technologies - Description Logic

Sveva Pepe

Thursday 17th December, 2020

1 Description Logics

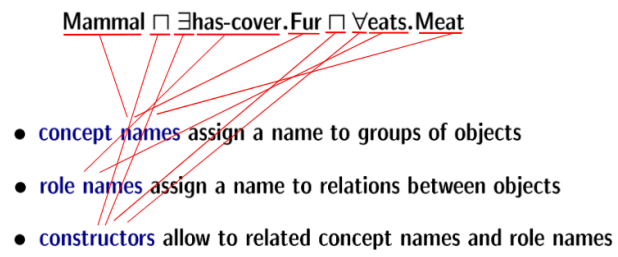
Knowledge representation: the goal is to develop a formalism for providing *high-level* descritpion of the world that can be effectively used to build applications.
We need to do some *choices* in the knowledge representation area:

- **formalism**: it determine *formal syntax* (we express them by mathematical rules) and *formal unambiguous semantic* (the meaning that we associate to these rules). We need to use these otherwise it is impossible to build automatic processing;
- **high-level description**: it is important because we work and describe different ways. High-level representation are typically symbolic representation, so we use symbols to represent knowledge, **not** numeric representation;
- **intelligent representation**: if you knowledge representation allow to derive new knowledge from a given knowledge, it will we consider intelligent;
- **effectively used**: a resoning techinque that should allow usable information. You can really build an implementation of this approach.

Knowledge is represented by *knowledge base* that it is composed by:

- **TBox** - Terminological background knowledge
- **ABox** - Knowledge about individuals

The distinction between TBox and ABox is similar to the distinction between databases schema and instance. The database schema is **structure** (how many tables) and then the *knowledge about the single instance is inside the table*. However, the TBox (schema part of the knowledge) can be more expressive and Abox is knowledge about the instance.
Concept language means that we ahve a formal language to write the TBox and ABox. Thelanguage is very important because you can see that *description logics* is a **family** of languages.
Reasoner because once you represent a KB then you have to use algorithm to be able to process such knowledge base. Most of these algorithm are called reasoner because they do very complex tasks. Since the KB is written in language that is very expressive it is very complicated to access and extract the knowledge from this KB, so you need a very sofisticated techinque to do this. This is why you need **Reasoner** because you *simulate deductive reasoning* that it is standard human being.



Example of the language of building a **concept expression** in desciption logics KB. We see the **syntactic** aspect of it.
The words by themselves does **not** mean anything, they are just symbols. The terms **don't** have any predefined meaning.
There are *classification* of the terms in three categories: **concept names**, **role names** and **individual names**. The last one is not present in the example. **Constructors** are used to create combination of *concept and role names*.

$$\text{Cat} \equiv \text{Mammal} \sqcap \exists \text{has-cover.Fur} \sqcap \forall \text{eats.Meat}$$

Primitive concept can build a complex concept. In example we can see that primitive concept as *Mammal*, etc that create complex concept ,**Cat**. This example is called **concept deinition**.

1.1 Syntax of one specific description logic - ALC

Atomic types: concept names $A, B, \ldots$ (unary predicates)
role names $r, s, \ldots$ (binary predicates)

$\mathcal{ALC}$ concept constructors:

$\neg C$	(negation)
$C \sqcap D$	(conjunction)
$C \sqcup D$	(disjunction)
$\exists r.C$	(existential restriction)
$\forall r.C$	(value restriction)

Special concepts:

$\top$	(top concept)
$\bot$	(bottom concept)

For example: $\neg(\ A\ \sqcup\ \exists r.(\forall s.B \sqcap \neg A))$
$\text{Mammal} \sqcap \exists \text{has-cover.Fur} \sqcap \forall \text{eats.Meat}$

We need to define a set of terms,*concept names* and *role names*. The we have the **concept constructor** to combine names to create some more complex class. And here we have 5 constructors. There are two **special concepts**: top concept andbottom concept, same rule of true and false in logic.

1.2 Semantic of named concepts

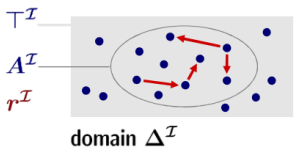
Semantics based on interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

Concepts: Subsets of domain $\Delta^{\mathcal{I}}$

Roles: binary relations on domain $\Delta^{\mathcal{I}}$

Primitive concepts

$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
$\bot^{\mathcal{I}} = \emptyset$
$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$



We use mathematical notion to define *semantic* of description logic.
We start from notion of interpretation I, it is composed by **domain of interpretation**, Δ^I, and **interpretation function**, $\cdot^I$.
The *domain of interpretation* is just a **non emplty set** of elements, so it should contain at 1**least one** element. It can be *infinity* byt **only** countably infinite (infinity of natural numbers). We use Δ^I to **interpret** the concept names and role names. *Concept names* are interpret by choosing **subset of domain** Δ^I for every concept names. We can also use empty set to interpret concept names. *Role names* are intepret as subset of **binary relation** on domain Δ^I.
Intepretation function takes all the names and provide and interpretation represt to delta of the alphabet of symbols.

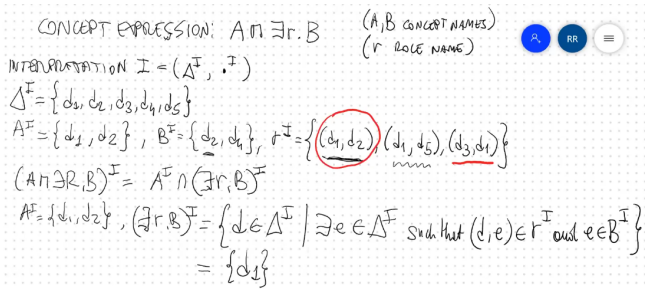
Interpretation is also called *possible world* because it is possible intepretation of my names, terms.

Semantics of complex concepts:

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\exists r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists e : e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$$
$$(\forall r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \forall e : e \in \Delta^{\mathcal{I}}, (d,e) \in r^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$$

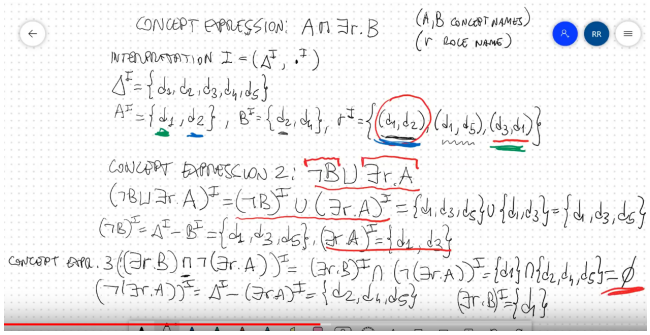
Now we have rules to give the meaning/semantics of the complex concept expression. Once we have interpretation of single concept names these recusrive rules define intepretation of complex concept expression.

1.3 Example 1 concept expression and interpretation



We can try to intepret concept expression, so we are going to provide an interpretation of $A \cap \exists r.B$.
First, we are goint to compute two internals interpretation, so A^I, that we have already, and $(\exists r.B)^I$.
$(\exists r.B)^I = \{d_1\}$, we remember that the **ordered matters**, so the order of pairs is *important*, we **cannot change it**.
Now we have to compute $(A \cap \exists r.B)^I$ that it is equal to $\{d_1\}$

1.4 Example 2 concept expression and interpretation



Now we try with another concept expression.
$(\neg B \cup \exists r.A)^I$

1.5 Usage of semantics - reasoning tasks for the concept

model of C: interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$

1. Concept satisfiability

   C is satisfiable if there exists a model of C.

   If unsatisfiable, the concept contains a contradiction.

2. Concept subsumption written $C \sqsubseteq D$

   Does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold for all $\mathcal{I}$?

   If $C \sqsubseteq D$, then D is more general than C

3. Concept equivalence written $C \equiv D$

   Does $C^{\mathcal{I}} = D^{\mathcal{I}}$ hold for all $\mathcal{I}$?

   If $C \equiv D$, then D and C 'say the same'.

Model of concept expression: interpretation I such that in this interpretation the *concept expression* C^I is **not empty**.
Satisfiability: exists a model means that it is possible that some "gods" build an interpretation where such that such concept interpreted in this interpretation is non empty. If concept expression C is **unsatisfiable** means that the concept contains contraddiction.
Concept subsumption: D is superclass of C
Concept equivalence: this property holds if for every interpretation I the interpretation od C is equal to the interpretation of D. Equality from *semantic* viewpoint.

1.6 TBox - syntax and semantic

Kinds of concept axioms:

- Primitive concept definition: $A \sqsubseteq D \qquad A \in N_C$
- Concept definition: $A \equiv D \qquad A \in N_C$
- General concept inclusion (GCI): $C \sqsubseteq D$

$C \sqsubseteq D$ holds in an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

- General concept equivalence: $C \equiv D$

$C \equiv D$ holds in an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$

TBox $\mathcal{T}$: Finite set of concept axioms.

$\mathcal{I}$ is a model of a TBox $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all $C \sqsubseteq D \in \mathcal{T}$.

TBox consist of formula called **axioms**. First of all there are different kids of **concept axioms**, axioms that talked about concepts. In TBox we say that, for example, concept name A is subsumed by concept expression D. It is a statement and **not** a question, things that happen in *concept assumption*.
In *Primitive concept definition* $A \subseteq D$ where A is **concept names**, instead, in *GCI* we have $C \subseteq D$ and C could be whatever we want **not only** concept name, it could be also **concept expression**.

Now we are focus on the **Reasoner**, algorithm techniques that are to process the information in a Knowledge base.
We have study an interpretation of the *concept expression* in a possible world, in any interpretations. So we have introduced a notion of interpretation and we have see **how to evaluate** interpretation and concept expression, in particular ALC concept expression.
We have introduce 3 kinds of formulas that we can write as a **statement of TBox**: *primitive concept definition*, *concept definition*, *general concept inclusion (GCI)*, *general concept equivalence*.
The *TBox* T is **finite set of concept axioms** and I is a **model** of T if $C^I \subseteq D^I$ for all $C \subseteq D \in$ T, i.e I is a model if for every concept inclusion $C \subseteq D$ that belong to the TBox T it is true axioms also in the interpretation, so $C^I \subseteq D^I$.
All the 4 cases that we have see can be *reduce to GCI*.

Why?
Because, first of all, if we consider the last case, *general concept equivalence*, $C \equiv D$ corresponds to a **pair of GCI** $C \subseteq D$ and also $D \subseteq C$. This means interpretation of C is contained in interpretation of D and viceversa.
The first two cases, are special case of GCI and GCE. Every kind of TBox statement that you can write can be encoded just by a set GCI.
We can simplify and say that, generally in TBox, respect to concept axiom, we can just consider GCI because it able to simulate the other cases.

1.7 Classification of TBoxes, ALC TBoxes

1. TBox $\mathcal{T}$ is a general TBox, if
   - it is a finite set of concept axioms
   - cyclic definitions and GCIs are allowed

   {WildAnimal ≡ Animal ⊓ ¬∃owner.⊤,
   Mammal ⊓ ∃bodypart.Hunch ≡
   Camel ⊔ Dromedary}
2. TBox $\mathcal{T}$ is an unfoldable TBox, if it has
   - only (primitive) concept definitions
   - concept names at most once on the left-hand side of definitions
   - no cyclic definitions, no GCIs

   ~~{Elephant ≡ Mammal ⊓ ∃bodypart.Trunk~~
   ~~Mammal ≡ Elephant ⊔ Lion ⊔ Zebra}~~

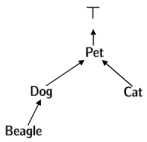
➳ Unfoldable TBoxes can be conceived as macro definitions.

We distinguish for TBoxes: **general TBox** and **unfoldable TBox**.
On top of the figure: we have **arbitrarly** TBox that it is compose of general concept inclusion (GCI). There are no special restrictions. In example, we can see equivalence instead of concept inclusion because it is just like a shortcut that we use instead of writing two GCIs. We can conclude that **general TBox** is a *set of arbitrarly concept axioms*.
On bottom of the figure: TBox can be classfies as **unfoldable** if it have **only** *primitive concept definition* and *concept names* **must** apperar **at most one** in left-side of definition and then there are **no cycle**.
The cycle is the fact that when we are talking about concept definition, on the left side of axiom we can **only** see a *concept name*, like Elephant, and this concept name is defined by other concept names in the right-side. **If one of thes concepts**, like Mammal, name has a definition the previous concept name (Elephant), we can see that we create a cycle.
In example, there is a cycle between Elephant and Mammal.
TBox is called **unfoldable** because essentially we are **not able** to unfold the definition if there are cycle in definition.

Reasoning Task to respect to the TBox

Reasoning tasks for TBoxes:

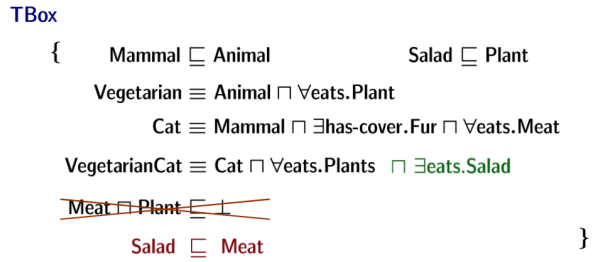
1. Concept satisfiability w.r.t. TBoxes
   Given C and $\mathcal{T}$. Does there exist a common model of C and $\mathcal{T}$?
2. Concept subsumption w.r.t. TBoxes ($C \sqsubseteq_{\mathcal{T}} D$)
   Given C,D and $\mathcal{T}$. Does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold in all models of $\mathcal{T}$?
3. Classification of the TBoxes
   Computation of all subsumption relationships between all named concepts in $\mathcal{T}$.
   ⟹ Subsumption can be used to compute a concept hierarchy:



Concept satisfiability w.r.t TBox: It there a *model* of T where concept C is satisfiable?
Remember a model for concept expression is a model such that the interpretation of concept of C is **non empty**
Concept subsumption w.r.t TBox: It is true that in every model of the TBox the subsumption between C and D is true?

When we have a KB we are stating knowledge about the world and knowledge base acts, in this case TBox, as a *filter of all possible* worlds. In fact, definition of these two reasoning tasks **only** focus on model of the TBox. The more axioms we write **less model** we obtain.
Why?
Because every formula becomes a **constraint** on the interpretation.
The third reasoning task is **classification of the TBox**, when we compute concept classification of a TBox we get a tree of concept names, and we can see subsumption relation between two concept names with an arrow.

We do the computation because it is **not** written explicity, the subsumption is **not** explicit in the TBox, we **must** do some reasoning to extract inclusion between concept names. You **cannot** just read the TBox and extract all subsumptions between concept names because thing produce **incomplete classification**, you have to check subsumption between all pair of concept names that apper in TBox to get classification. The classification task is **not trivial**. It corresponds to set of concept subsumption task.



We start with **black statements**. fist, ignore green and red modifications.

1. TBox is satisfiable - **satisfiability of TBox**
2. VegetarianCat is satisfiable is TBox in green - **satisfiability of concept expression w.r.t TBox**
3. VegetarianCat $\subseteq$ Vegetarian w.r.t to all TBox - **concept subsumptionproblem w.r.t. all TBox**

1) We need to prove that TBox is **satisfiable**. So exist a model for the TBox. If we want to find a model for Tbox we **must find** an interpretation when every axiom is true. We ca see that we have a kind of contradiction between *Cat, VegetarianCat* and $Meat \cap Plant \subseteq \bot$. VegetarianCat is a Cat that can only eat Plant but a Cat only eat Meat, and Meat and Plant are disjoint.
What can VegetarianCat eat?
Nothing, because it cannot eat Meat by definition that it can eat only Plant and Plant is disjoint from Meat. But VegetarianCat cannot eat Plant because a Cat can only eat Meat. So it cannot eat anything because otherwise he violate constraint.
How can we define a model for this TBox?
Model in which all the VegetarianCat cannot eat anything because it is not mandatory to eat something. If we make Cat to partecipate zero times to the role eats, we satisfies the restriction. It is **not** require to participate in a role, but if you partecipate you must satisfies these restriction.
Anther model we can have zero VegetarianCat.
We are taking on **model of TBox** and **not** model of concept, because otherwise we should have **non-empty interpretation of that concept**.

2) Now we are considering a specific *concept expression*, simple concept expression: concept name.
Considering black TBox VegetarianCat as we said **cannot** have instances, so VegetarianCat is satisfiable because we can create interpretation where there are VegetarianCat that don't eat anything . But now for the green TBox, VegetarianCat is forced to eat some Salad. Now must eat some object of Salad. Since now wgat is eaten by VegetarianCat is some element of type Salad but this **does not** satisfies conditions because Salad is subsumption of Plant. But Meat and Plant are disjoint so it is impossible to have a Cat that eat both Meat and Plant. We can conclude that we **don't** have model for the green TBox where we have VegetarianCat. So VegetarianCat is **unsatisfiable** w.r.t. green TBox.

3) VegetarianCat is subconcept of Vegetarian.

Why?
Because, considering first the black TBox, the point is that VegetarianCat is defined as a Cat that for all eat has a type Plants. Now a Vegetarian is an Animal and for all eat has Plant. Cat is subconcept of Mammal by definition and Mammal is subconcept of Animal. By **transity property** Cat is subconcept of Animal. Thus, Vegetarian is subconcept of Animal. Finally, VegetarianCat is subconcept of Vegetarian in every model of the TBox.
In green TBox VegetarianCat get smaller, it is a subset of Vegetarian but smaller, more restricted but it does **not** matter. The same is for the red TBox.
In knowledge representation and reasoning we need algorithm that answer these questions automatically. Now we have tried to do manual explanation. This is the important aspect of *Reasoner*, in which there are techniques that are able to answer reasoning problem of descriptio logic knowledge base.

1.8 ABox - syntax and semantic

ABox assertions in DL systems are:

- Concept assertions: $C(a)$
- Role assertions: $r(a, b)$

Extend interpretations to individuals:
$a \in N_I$, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$

Semantics of assertions:

- Concept Assertions: $\mathcal{I}$ satisfies $C(a) \iff a^{\mathcal{I}} \in C^{\mathcal{I}}$
- Role Assertions: $\mathcal{I}$ satisfies $r(a, b) \iff (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

An ABox $\mathcal{A}$ is a finite set of assertions.

$\mathcal{I}$ is a model for an ABox $\mathcal{A}$ if $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$.

Now we can see what are the **statement** that we can write in a ABox. ABox assertion are of two types: **Concept assertion**, $C(a)$, where C is a concept expression and a is an *individual name* and the element with name a belong to the interpretation of concept expression C, and **Role assertion**, $r(a, b)$, pairs of individual belong to the role.
We need to **extend** our possible world because we have to consider also the individual names.

How do we intepret individual names on interpretation?
We have to define an element **for all** individual name, $a \in N_I, a^I \in \Delta^I$. The function that intepret individual return, for individual a, an element of interpretation domain. The difference between concept and individual is that the interpretation *concept* is **subset** of the domain, while the interpretation of the *individual* is **single element** of the domain.

A concept assertion, $C(a)$ satisfies by interpretation I, *if and only if* the interpretation of individual a belong to the interpretation of the concept C.
A role assertion, $r(a, b)$ is satisfies by interpretation I, *if and only if* the interpretation of a and interpretation of b belong to the interpretationof the role r.

Example ABox

An *Abox* is s **finite set of assetions**.
I is **model** for ABox A if A satisfies *all assertions* in A.

ABox is a partial description of the world.

(unlike models!)

ABox $\mathcal{A}$

Mammal(garfield)	Fur(f17)
Lasagna(l23)	has-cover(garfield, f17)
eats(garfield, l23)	likes-most(garfield, garfield)
∀eats.Beef(garfield)	

Example of ABox. For instance, element associate to *garfield* belong to Mammal. We can also write in a ABox, **not only** concept name but also *complex expression* as $\forall eats.Beef(garfield)$, garfield only eat Beef if it participate.

Exercise
TBox T

$$\begin{cases} A \sqsubseteq B \\ B \sqsubseteq \exists r.D \\ D \sqcap A \sqsubseteq \bot \end{cases}$$

We have 3 axioms.
Let's build an interpretation I:

$$\Delta^I = \{d_1, d_2, d_3, d_4\}$$
$$A^I = \{d_1\}$$
$$B^I = \{d_2\}$$
$$D^I = \{d_2, d_3, d_4\}$$
$$r^I = \{(d_2, d_1), (d_2, d_4)\}$$

Does I satisfy T? Is I a model of T? i.e all axioms in the TBox are satisfied
It is true that interpretation of A is subset of interpretation of B? *Yes* $\{d_1\}$ is subset of $\{d_1, d_2\}$.
It is true that interpretation of B is subset of interpretation of $\exists r.D$? *No* because $(\exists r.D)^I = \{d_2\}$ because we

have to take every element that it is in the first position of r such that inj the second position there is an element that belongs to D.
We can conclude that I is **not a model** for T becuase one of the axiom is **not satisfied**.
Modify the interpretation and consider I'

$$\Delta^{I'} = \{d_1, d_2, d_3, d_4\}$$

$$A^{I'} = \{d_1\}$$

$$B^{I'} = \{d_2\}$$

$$D^{I'} = \{d_2, d_3, d_4\}$$

$$r^{I'} = \{(d_2, d_1), (d_2, d_4), (d_1, d_3)\}$$

It is true that interpretation of A is subset of interpretation of B? *Yes* $\{d_1\}$ is subset of $\{d_1, d_2\}$.
It is true that interpretation of B is subset of interpretation of $\exists r.D$? *Yes* because $(\exists r.D)^{I'} = \{d_2, d_1\}$
It is true that $(D \sqcap A)^{I'} \sqsubset \bot^{I'}$? *Yes* because $(D \sqcap A)^{I'} = \emptyset$ and $\bot^{I'} = \emptyset$
We can conclude that I' is a model of T because **all axioms are true** in this interpretation I'.

It is $\exists r.D$ is satisfiable w.r.t. T?
it is a **concept satisfiability problem**. The concept satisfiability problem w.r.t to TBox means that we can answer YES if we find a model of T where the interpretation of the concept expression is **not empty**. Actually, we have that interpretation $(\exists r.D)^{I'} \neq \emptyset$

It is $A \sqcap B \sqcap \exists r.\neg A$ satisfiable w.r.t. T?
Let's consider I' that we have already define:
$(A \sqcap B \sqcap \exists r.\neg A)^{I'} = A^{I'} \cap B^{I'} \cap (\exists r.\neg A)^{I'} = \{d_1\} \cap \{d_1, d_2\} \cap \{d_1, d_2\} = \{d_1\}$
The formula is **satisfiable** because it is non empty

It is $\forall r.A$ satisfiable w.r.t. to T?
$(\forall r.A)^{I'}$ no we have to write element such that for every participation in the first position in the role r, the second element **must** belong to A.
In our, example we have that for pair (d_2, d_1) we have that d_1 is in A but it it **not enough** to put d_2 because we need to check that all partecipation of d_2 in the role must satisfies the fact that the other element belongs to A. In the case of (d_2, d_4) we have that d_4 is **not belong** to A and so this pair makes d_2 **not** in the interpretation of $\forall r.A$.
$\forall r.A = \{d_3, d_4\}$ we have them because there is no pair in r that start with d_3 and the same for d_4 so the elements trivially belong.

Is $\forall r.A \sqcap \exists r.A$ satisfiable w.r.t T?
$(\forall r.A \sqcap \exists r.A)^{I'} = (\forall r.A)^{I'} \cap (\exists r.A)^{I'} = \{d_3, d_4\} \cap \{d_2\} = \emptyset$
The interpretation of these concept expression in I' is empty, but we **cannot** conclude yet that concept is **unsatisfiable w.r.t T** because we have to prove that exists **no model for T** where the *interpretation* of this concept is **non empty** and we **only** verify that in this specific model of T the interpretation is empty.
Modify the interpretation and consider I"

$$\Delta^{I'} = \{d_1, d_2, d_3, d_4\}$$

$$A^{I''} = \{d_1\}$$

$$B^{I''} = \{d_2\}$$

$$D^{I''} = \{d_2, d_3, d_4\}$$

$$r^{I''} = \{(d_2, d_1), (d_2, d_4), (d_1, d_3), (d_3, d_1)\}$$

Now $(\forall r.A \sqcap \exists r.A)^{I'} = (\forall r.A)^{I'} \cap (\exists r.A)^{I'} = \{d_3, d_4\} \cap \{d_2, d_3\} = d_3$ and I" is still model for T and also for concept expression

Concept subsumption w.r.t TBox
Does $C \sqsubseteq D$ hold in every model for T? There is a **theorem**: $C \sqsubseteq D$ holds in every model for T **iff** $C \sqcap \neg D$is **unsatisfiable** w.r.t T.
We can see that a problem of *concept subsumption* corresponds to a problem of *concept unsatisfiability*. Thanks to this theorem we can focus just on **concept of satisfiability** problem. We answer YES to subsumption if we can answer NO to satisfiability problem.

Is $A \sqcap \neg B$ satisfiable w.r.t T? Can we find a model for T in which this concept is non empty
No because interpretation is a model if satisfies all axiom of the TBox, so we know that $A \sqsubseteq B$ becomes $A \sqcap \neg B$ and this means for the theorem that the satisfiability problem return NO and concept subsumption is true.

1.9 ABox Reasoning

In ABox we write information about **single individual**, we write a membership relation names of individuals and names of concepts and roles. Instead, in TBox we states knowledge about **concept and roles**. Axioms that states some correspondence among names of concept and names of roles. We define a relation of subconcept between two names.
We will consider all possible worlds that satisfies axioms in ABox, to make assetions formula true. Also in the case of ABox, it play a role of filter of in the interpretation. We are going to consider **only** models of ABox as *relevant possible worlds* for our **semantics** and so **discard** all interpretation that **do no satisfies** axioms in ABox.

Reasoning tasks for ABoxes:

1. ABox consistency
   Given: $\mathcal{A}$ and $\mathcal{T}$. Do they have a common model?

2. Instance checking
   Given: $\mathcal{A}$, $\mathcal{T}$, individual a, and concept C
   Does $a^{\mathcal{I}} \in C^{\mathcal{I}}$ hold in all models of $\mathcal{A}$ and $\mathcal{T}$?

3. ABox realization
   Given $\mathcal{A}$ and $\mathcal{T}$.
   Compute for each individual a in $\mathcal{A}$:
   the named concepts in $\mathcal{T}$ of which a is an instance of.

ABox consistency: Do have ABox and TBox common model? Here we have still considering Abox and TBox different *indipendent* components. The union of ABox and TBox is consider **knowledge base**, so we can also call this task as a **knowledge base consistency**. We need to find an interpretation that it is a model for both TBox and ABox.
Instance checking: Check *if* interpretation of individual a in the interpretation of concept C in any interpretation of both ABox and TBox. It is a **reasoning** problem that is very close to the *query answering* in a database, where the query is concept assertion $C(a)$.
In fact, what we actually check with this property is that $C(a)$ is true in *every model of ABox and TBox*.
ABox realization: we want to compute **all the concepts** to which every individual belongs. Map all individuals over every named concept. Make explicit some implicit knowledge contained in knowledge base.

ABox is a partial description of the world.

ABox	Mammal(garfield)	Fur(f17)
	Lasagna(l23)	has-cover(garfield, f17)
	~~eats(garfield, l23)~~	likes-most(garfield, garfield)
	∀eats.Beef(garfield)	
TBox	Cat ≡ Mammal ⊓ ∃has-cover.Fur ⊓ ∀eats.Meat	
	Meat ≡ Beef ⊔ Chicken	
	Lasagna ⊓ Beef ⊑ ⊥	

1. ABox is **inconsistent** w.r.t. T
2. garfield is an **instance** of Cat

1) If you look at the black version of these two ABox and TBox. We can prove that ABox it is **incosistent** w.r.t T or equivalently the KB of union of formula it is incosistent.
How can we convince ourselves that ABox it is incosistent w.r.t. T is true?
We need to answer question: can find a model for all statements appear in the picture? The answer is NO.
Why?

Because we can see in the ABox that *garfield* is Mamma and it eat **only** *Beef*. So since we add also *eats(garfield,123)* we must to impose that in every model of this ABox 123 belongs to Beef because every time garfield participate in role eats, the other element must belong to Beef. So from these assertions ($\forall eats.Beef(garfield)$ and eats(garfield,123)) there is **implicit consequence** that is 123 must belong to the concep beef in every model of the ABox. But we have also the concept Lasagna(123). Now there is a **problem** because the *last statement of TBox* says that $Lasagna \sqcap Beef \sqsubseteq \bot$. We have 123 that must belong to Beef and Lasagna **but** there is a *disjunction* between Lasagna and Beef so it is **impossible to satisfy** all axioms, these formula are *contradictory*.
We would conclude that ABox is **incosistent**.
The red cross to *eats(garfield,123)* it is a kind of modification to **make it consistent**.

2) In every model of the KB the *individual garfield* is **always** a member of concept Cat.
Notice that this is **not written** in the ABox. We do not see the assertion "Cat(garfield)" in the ABox, **but** assertions *has-cover(garfield, f17)*, *Fur(f17)*,*Mammal(garfield)*, $\forall eats.Beef(garfield)$ make the concept expression $Mammal \sqcap \exists has-cover.Fur \sqcap \forall eats.Meat$ true for the element that interpret individual garfield.
Furthermore, $Meat \equiv Beef \sqcup Chicken$, which implies that Beef is a **subconcept** of Meat. So everything that garfield eats is a subconcept of Meat because of $\forall eats.Beef(garfield)$. So garfield is **instance** of Cat.

We have just seen that **instance checking problem** it is very difficult to prove, we have to do some reasoning step. In particular, we have to *combine* different formulas, statement to prove a single statement.

Why ABox is consider a partial reprentation of the world?
We could add other concept assertion in this ABox **without changing** the meaning of knowledge base.

Why?
Because it is true in every model of knowledge base, it derive by **logical consequence**
The ABox together with TBox imply more assertion than assertion that we see in ABox because of logical consequence. We can understand that ABox is a *partial description of the world* because there are things that are true that are **not explicity written** in syntattic expression of KB.
One of the *main problem* of **reasoning** in description logic is to find the **hidden information**, the implicit consequence of our knowledge.
This is **one aspect** that it is *different* from databases. We can do a comparison on what happen in *desciption logic* in **knowledge base** and in **database**. In *database* we have a **complete desciption of the world**. In *description logic knowledge base* we have **partial description of the world**, this means that there are formula that are so complicated that they derive things that **do not appear** *syntactically* in KB **but** *semantically* are there, are consequence and we need to find them.
There is a *second interpretation* to partial desciption of the world, it is related to hidden information, and this characteristic of DL (the fact that there is implicit information) is due to a **different semantic assumption** that it is made in DL and it is inherit from classical logic with repect to database. This difference is **open world assumption**, it mean that we admitt *models* for our knowledge base that *contain more things tha we see* in KB itself. The world is contrained by ABox and TBox but then it can contain more element that we see. In *database* it is **not allowed**, because in database that a table contains that particular assertions, then also the interpretation contain that assertions, we **cannot** add other things. This is called **closed world assumption**, what you see syntactically in database is exacty the model.

1.10 Extending description logic language beyond ALC

Number restrictions $(\leq n\, r),\ (\geq n\, r)$

$$(\leq n\, r)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$$
$$(\geq n\, r)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$$

Qualified number restrictions $(\leq\ n\ r\ C),\ (\geq\ n\ r\ C)$

$$(\leq n\ r\ C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$$
$$(\geq n\ r\ C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$$

Example:

Car $\sqcap$ ($\geq$ 5 has-seat) $\sqcap$ ($\leq$ 5 has-seat)
$\sqcap$ ($\geq$ 1 has-seat Drivers-seat) $\sqcap$ ($\leq$ 1 has-seat Drivers-seat)

We can add some generalization of *existential restriction* and *forall restriction* that it is called **number restrictions**. **Number restrictions** are *concept expressions* of special syntax that can also be generalized to *qualified number restriction*. $(\leq nr)$ where n is **natural number** and r is **role name**, this expression is consider as *at most* We count how many times x appear in first element of pairs the interpretation of r. Essentially, $(\leq nr)^I$ returns the element of Δ^I such that the participation of x in r^I in first position is **not greater** than n. Instead $(\geq nr)$ is consider as *at least* and in this case we see if x participate at list n times. For *qualified number restriction* we have the **concept**. The *semantic* is similar to previous one but in this case we need to check if the second element of pairs in r belong to C^I.

The *qualified number restriction* is proper generalizaion of $\forall r.C$ restriction and $\exists r.C$ restriction.
In example of Car, we can see that a Car **should have** *exactly* 5 seats because we have ≥ 5 has-seat and also ≤ 5 has-seat.

Usage of nominals

Sometimes it is useful to refer to individuals in the TBox.

Recall: If they have same description

- Concepts are equivalent.
- Individuals are distinct.

C $\equiv$ ($\forall$ has-child.$\bot$)
D $\equiv$ ($\leq$ 0 has-child)
$\Longrightarrow$ C $\equiv$ D

(Carla, Luisa): parent, Person(Carla),
(Markus, Luisa): parent, Person(Markus)
$\Longrightarrow$ Carla $\neq$ Markus

Concept constructors using individuals:

- Nominals $\{a\}$ $\quad \{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$
- One-of $\{a_1, \ldots, a_n\}$ $\quad \{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$

E.g.: RomanCatholic $\sqsubseteq$ $\exists$ knows.{Pope}

We have said that individual can **only** appear in ABox because in TBox we can only write axioms, general concept inclusion etc.
But sometimes we can have some extension in DL in which we can write *individuals* in the TBox.
How write this?
In *two ways*:

- an extension is called **nominal concept**, $\{a\}$
  What is the interpretation of nominal concept?
  The interpretation is a *singleton* interpretation, that it interprets as a *set that contains* the single element and this element is interpretation of the individual a, $\{a\}^I = \{a^I\}$. We create an **artifical concept** that *only* contains one individual. This is why is called **nominal concept**.
- An extension called **One-of** in which we **do not** consider a single individual but a *list of individuals*, $\{a_1, ..., a_n\}$. You can turn nominal concept in "one-of" concept. The **semantic** of this list is the **set of interpretations of all individuals** appearing in One-of, $\{a_1, ..., a_n\}^I = \{a_1^I, ..., a_n^I\}$.

Role declarations

r	atomic role	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ e.g. has-child
f	feature or attribute	$f^{\mathcal{I}} = \{(x,y) \mid (x,y) \in f^{\mathcal{I}} \wedge (x,z) \in f^{\mathcal{I}} \Rightarrow y = z\}$ e.g. has-mother
$r \sqsubseteq s$	role inclusion role hierarchy	$r \sqsubseteq s$ holds in $\mathcal{I} \Leftrightarrow r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ e.g. has-mother $\sqsubseteq$ has-parent $\sqsubseteq$ has-family-member has-sibling $\sqsubseteq$ has-family-member

Roles are used in a very **minimal way in ALC** because we can write role in TBox **only** in *existential* and *universal* description. Even in ABox you can write **atomic role assertion**.
In DL the usage of role is much more sofisticated that this. For instance, we can define roles with special property, **attribute/feature**. We can *classify* property to be functional, it means that if there is a pair, (x,y), in the interpretation of f, every other pairs that have initial first element x, (x,z), is actually such that second element corresponds to y. This means that the participation of x in f^I is **functional/feature**, there is *at most* one participation on every element in first position of f. f is a *functional role*, for example matricola.
We can also have an extension in which we have **role inclusion axioms** that we can state that some role is **subrole** of another role, $r \sqsubseteq s$.
What happen if we add this role inclusion in our TBox?
The interpretation of r **must be a subset** of interpretation of s.

Role operators

r^+	transitive role	$(r^+)^{\mathcal{I}} = \{(x,z) \mid (x,y) \in r^{\mathcal{I}}, (y,z) \in r^{\mathcal{I}} \Rightarrow (x,z) \in r^{\mathcal{I}}\}$ e.g. has-ancestor
r^-	inverse role	$(r^-)^{\mathcal{I}} = \{(y,x) \mid (x,y) \in r^{\mathcal{I}}\}$ e.g. (has-parent)$^-$ = has-child

We can also have real **role operator**: *trainsitive role* and *inverse role*.
In **transitive role**, r^+, instead of just consider the interpretation of r we can consider the *transitive closure* of the interpretation of r.
In the **inverse role**, r^-, return the inverse relation in the role interpretation. We are inverting all the pairs.

Basis-DL: $\mathcal{ALC}$

- $\mathcal{E}$: Existential restrictions
- $\mathcal{N}$: Number restrictions
- $\mathcal{Q}$: Qualified number restrictions
- $\mathcal{O}$: nominals, Objects
- $\mathcal{F}$: Features, functional roles
- $^+$: Transitive roles
- $\mathcal{I}$: Inverse roles
- $\mathcal{H}$: role Hierarchies
- $\mathcal{R}$: complex Role inclusions

$\mathcal{S}$: Abbreviation for $\mathcal{ALC}^+$

The name of the *description logics* **change** in according of usage difference *concept* and *role construction*, and *concept* and *role axioms*. We can add one of this letter to extend ALC, for instance if we want to allowed *nominal, Object* we obtain ALCO.

Reasoning: automatic way to process information represented in a knowledge base. These automatic ways are all based on **semantic** of *description logics*. We exploit the *semantic* of knowledge base. We need **reasoning** because of the semantic complexity of KB. All the reasoning tasks, also very basic questions like if an individual is an *instance of* concept, are **not very easy to answer** because the sematic implies the possible derivation in all models of KB of information that it is **not explicity** written in knowledge base.

TBox and the ABox capture implicit information.
We want to access this information by making it explicit!

Does my knowledge base ...

- contain a concept that cannot have instances? (since its definition is contradictory.) — Check for satisfiabiliy w.r.t. TBox.
- contain an unwanted synonym for a concept? (unwanted / unintended redundancy in my TBox) — Check for equivalent concepts.
- yield the concept hierarchy I wanted? — Classify.
- contain individuals not compliant with the specification of the concepts they belong to? — Check ABox consistency.

Here we have a summary of our problems:

- **Satisfiability** of *concept w.r.t TBox*
- **Eequivalence** or **subsymption** between *concept expression*
- **Classification** of all concepts, building concept hierarchy
- **ABox consistency/KB consistency**
- **Instace checking**: checking whether in every model a given individual is a *instance* of a given concept expression/name

1.11 Fondamental requirements in reasoning

Requirements for good reasoning algorithms:

They should be decision procedures, i.e. they should be:

- terminating, — You get always an answer.
- sound, — Every positive answer is correct.
- complete. — Every negative answer is correct.

➳ Prerequisit for safe and reliable applications!

We are considering **decision procedure**.
What is a decision procedure?
It is an algorithm that answers YES or NO, a boolean answer is provided by this algorithm.
To make a *decision procedure* an algorithm **should**:

- **Terminating** for every legal instance of the input
- **Sound**
- **Complete**

Terminating means that for every KB that are syntactically correct for ALC, the algorithm **should always terminate** its computation. **Never get stuck** or loop without returning any answer.
Soundness means that if the algorithm answer YES, the answer is really YES according with the *semantic*. Every correct answer provided by the algorithm is a correct answer w.r.t the semantic of KB or DL.
Completeness is a complementary property of *soundness* and says that every negative answer provided by the algorithm is a negative answer according to the semantic of DL. When the algorithm answer NO it is really NO.
In combination *soundness* and *completeness* are **total correctness** because every time the algorithm answers, the algorithm answer correctly. The three properties together implies that algorithm **always answer and always answer correctly**. However, the three properties are **not easy** to obtain in the case of reasoning task for description logic.

Many standard reasoning services can be reduced to satisfiability.
(If negation is present in the DL!)

Use the reduction and implement one reasoning method!

- Equivalence $\Longleftrightarrow$ Satisfiability
  $C \equiv_{\mathcal{T}} D$ iff $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$
- Subsumption $\Longleftrightarrow$ Satisfiability
  $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ unsatisfiable w.r.t. $\mathcal{T}$
  $C \not\sqsubseteq_{\mathcal{T}} \bot$ if C is satisfiable w.r.t. $\mathcal{T}$ unsatisfiable w.r.t. $\mathcal{T}$

Reduction a problem to another problem.
We have just seen how to reduce **subsumption** problem to a **unsatisfiability** problem. We can do also reduction from *equivalence* problem to *satisfiability* problem. Our **goal** will be to find one reasoning algorithm/method and use it to solve problem of other tasks beside one for which we have written the algorithm.

- Instance checking $\Longleftrightarrow$ ABox consistency
  a is instance of C w.r.t. $(\mathcal{T}, \mathcal{A})$ iff $(\mathcal{T},\ \mathcal{A} \cup \{\neg C(a)\})$ is inconsistent

- Satisfiability $\Longleftrightarrow$ ABox consistency
  C is satisfiable w.r.t. $\mathcal{T}$ iff $(\mathcal{T}, \{C(a)\})$ is consistent

Correspondence between **instance checking** problem of KB and **ABox/KB consistency** problem. The idea is that individual a is a necessarly a instance of C, so in every model a mustbe and instance of C, if when we try to say the contrary $\neg C(a)$ then we **do no** find any model. It means that in every model it is impossible to impose additional knowledge, $\neg C(a)$. If $\neg C(a)$ causes a contraddiction means that a is always an instance of C. Also **concept satisfiability** can be reduced in **ABox/KB consistency**. We create an artificial knowledge base that we assert that there is an individual a in C. This means that in every model of this TBox and ABox will be **non empty**, then if we find a model for this TBox and ABox it means that the ABox has *at least* a model in which C is non empty.

1.12 Reasoning algorithm for ALC-KB with unfoldable TBox

We are going to check whether there exist a model for this knowledge base. We are consider *satisfiability* of concept w.r.t TBox.

1. Use **reduction to reformulate the reasoning problem:** because it may be the case that we are going to work with method that is able to answer slightly different questions;
2. **Expand concepts w.r.t TBox:** peculiarity of having to deal with *unfoldable TBox only.* This step in which we get rid of the TBox. We encode TBox inside the concept expressions;
3. **Normalize concept descriptions:** transform through normalization step;
4. **Apply tableau rules:** apply subalgorithms called tableau algorithms.

The reason why this algorithm is **only** for unfoldable TBoxes is that because step 2) can **only** be apply for unfoldable TBox.

1.12.1 Expansion of concept descriptions

Idea: get rid of the unfoldable TBox in a preprocessing step.

Naive approach for expansion:

Let C be concept, $\mathcal{T}$ unfoldable TBox

1. replace every concept name of a defined concept with the right-hand side of its definitions $A \equiv C$
2. repeat until no more replacements can be made.

Remember that we are going to consider the satisfiability problem of concept w.r.t TBox.
The **idea** is very simple. Since the Tbox is **only made** of concept definition what we do is to replace every concept names of a defined concept in the TBox with the right.hand side of its definition.
We **do not** change the meaning, it meant that if concept expression C were satisfiable w.r.t T, also the the expanded concept expression, obtained after replacement steps, will be satisfiable and viceversa. The original concept expression and final concept expression have the same satisfiability property. We apply as much as possible to expansion to transform our concept expression. After this step we get rid of the TBox, it is a kind of encoding the meaning of the TBox in concept expression C. We can look for concept expression alone *without* the TBox.

Expansion process terminates due to acyclicity of the concept definitions!

But: exponential blow-up in the worst case!

$$\mathcal{T} = \{ \quad A_0 \equiv \forall r.A_1 \sqcap \forall s.A_1$$
$$A_1 \equiv \forall r.A_2 \sqcap \forall s.A_2$$
$$\vdots$$
$$A_{k-1} \equiv \forall r.A_k \sqcap \forall s.A_k \quad \}$$

This step 2) can **only** be done in *unfoldable TBox*.
But firste we can have a *computational problem* because the expansion is **always** a finite set but it could be **exponential**. We might get concept expression after expansion that it is exponentially larger than the original concept expression.
TBox contains k axioms, so we have order of k TBox and input is also in the order of k and output is exponential.

A concept C is in negation normal form (NNF) if negation occurs only in front of concept names.

Transformation rules:

$$\neg\neg C \rightsquigarrow C$$
$$\neg(C \sqcap D) \rightsquigarrow \neg C \sqcup \neg D$$
$$\neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D$$
$$\neg(\exists r.C) \rightsquigarrow \forall r.\neg C$$
$$\neg(\forall r.C) \rightsquigarrow \exists r.\neg C$$

Normalization **only** involve the *negation operator*. The usage of **negation operator** is *normalize*. Normalize means that we are going to use rewriting rules to **eliminate non atomic** usage of the negation operator.
A concept expression C is in **negation normal form (NNF)** if the negation operator occurs **only** in front of concept names. We **cannot** have negation of complex expression.
It is **always** possible to reduce or transform concept expressions to equivalent one which is in NNF. We use *five rules* in the picture.

Try to construct a model for the input concept C_0 as follows:
(C_0: expanded and in NNF)

- Represent potential models by proof ABoxes
- To decide satisfiability of C_0,
  start with one initial proof ABox $\mathcal{A}_0$
- Repeatedly apply tableau rules
  and check for obvious contradictions
- Return 'satisfiable' iff a complete and contradiction-free proof ABox was found
  (I.e. if all proof ABoxes contain a contradiction,
  return 'not satisfiable')

Apply tableau rule is the application of tablau algorithm.
The algorithm is an *algorithm for ABox satisfiability*. In fact, the first step is to **represent potential models** by *proof ABoxes*, it means that this algorithm try to build a model for an ABox, without a TBox. It is like "Give me an ABox and I would try to check if there is a model for this ABox". Notice that we are starting with concept expression C_0 (expanded and apply NNF).
To **decide satisfiability** of C_0 we start with create an ABox which contains one concept assertion that use this concept expression.
Then the algorithm is the *iteratively one*, in which we apply **iteratively tableau rules** and check if there are contradictions and in a positive way the algorithm stop. The algorithm look for atomic contradictions.
At the end, if the algorithm has *succeds* in expanding in a **non contradictory way** the initial ABox it says that a model for this ABox *exists* and return YES. Otherwise, it return NO.

Tableau rules play with formulas, concept assertions. Algorithm plays with formula and **not** with models, **not with semantic structure** because if we use interpretation we have an *uncontable infinite* number of possible interpretations. In this algorithm I just do process of formula.

Example
We have a TBox T:

$$\begin{cases} A \equiv A \sqcup \exists r.C \\ B \equiv \neg C \sqcap D \\ C \equiv E \sqcap \forall s.D \end{cases}$$

It is T unfoldable?
Yes, because it is compose compose **only** of concept definition and it has **no** cycle definition, so **no mutual dependencies**.
Consider to have a *concept expression*:
A

How do we expand this concept expression w.r.t T?
For every concept name appearing in this concept expression we have to look if there a definition. There is a definition of A so we can replace A with its definition. *Expansion of A w.r.t T*:
$B \sqcup \exists r.C$
Is there some concept name that it is defined and we need to expand again?
Yes, for example B. We replace B with $\neg C \sqcap D$. The formula becomes: $\neg C \sqcap D \sqcap \exists r.C$
We can also expand C replacing with: $E \sqcap \forall s.D$
The final formula is $(\neg(E \sqcap \forall s.D) \sqcap D) \sqcap (\exists r.(E \sqcap \forall s.D))$

What have we done?
We have produced an expression that it is equivalent to the initial one, in all the model of this TBox. The interpretation of A and the interpretation of the final formula in all models of the TBox T are the **same**. The *semantic* of A and the *semantic of final formula* in the TBox T are the same.

Why do we have done something like this?
Because now we **do not** need TBox. **Only** look at concept expression satisfiability. This is good because the problem of checking satisfiability of concept respect to the TBox is more more complicated than problem of just checking satisfiability of the concept expression alone because we need to check only if there is a model for that concept expression.

What happen if TBox is not unfoldable?
TBox T':

$$\begin{cases} A \equiv A \sqcup \exists r.C \\ B \equiv \neg C \sqcap D \\ C \equiv E \sqcap \forall s.A \end{cases}$$

It is **not unfoldable** because there is a cyclic dependency. The cycle **does not mean** that the transformation done by unfolding concept expression is *wrong*. It is **still correct**. If we do the expansion we do correct step but we have a problem of **infinite expansion**, so never stop, never get rid of A, never terminate. In unfoldable TBox the expansion of concept expression may be inifinite, so we are loosing the termination property.

Check negation normal forn of TBox T. It is the final formula in NNF?
No because there is a negation symbol in front of complex expression, $\neg$. Let's apply rules for NNF.

$$(\neg(E \sqcap \forall s.D) \sqcap D) \sqcap (\exists r.E \sqcap \forall s.D)$$

$$(\neg E \sqcup \neg(\forall s.D) \sqcap D) \sqcap (\exists r.(E \sqcap \forall s.D))$$

$$(\neg E \sqcup \exists s.\neg D \sqcap D) \sqcap (\exists r.(E \sqcap \forall s.D))$$

This is formula in NNF because we have **only** negation in atomic element. We are done also with step 3 of the algorithm. We want to have a **syntactic** usage of the negation such that the negation appear in front of concept names, what we want is atomic usage of the negation.

Why?
Because in this way we **do not** need to process negation in a computated way, by the tableau algorithm. Now we check negation only for simple contradiction formula because we have negation in atomic element and not for entire formula.

Tableau algorithm works on sets of ABoxes: $\mathcal{S}$

Initially, $\mathcal{S}$ contains proof ABox for concept C_0:
$\mathcal{S} := \{\mathcal{A}_0\}$, with $\mathcal{A}_0 := \{C_0(x_0)\}$

Apply tableau rules to set of proof ABoxes $\mathcal{S}$ until

- a proof ABox is complete (no more rules applicable)

or

- there exists an individual x in $\mathcal{A}$ such that $\{B(x), \neg B(x)\} \subseteq \mathcal{A}$ for some concept name B (Clash)
or $\bot(x) \in \mathcal{A}$.

	Precondition	Replace $\mathcal{A}$ by:
$\longrightarrow_{\sqcap}$	$(C_1 \sqcap C_2)(x) \in \mathcal{A}$ $C_1(x) \notin \mathcal{A}$ or $C_2(x) \notin \mathcal{A}$	$\mathcal{A}' := \mathcal{A} \cup \{C_1(x), C_2(x)\}$
$\longrightarrow_{\sqcup}$	$(C_1 \sqcup C_2)(x) \in \mathcal{A}$ $C_1(x) \notin \mathcal{A}$ and $C_2(x) \notin \mathcal{A}$	$\mathcal{A}' := \mathcal{A} \cup \{(C_1)(x)\}$ $\mathcal{A}'' := \mathcal{A} \cup \{(C_2)(x)\}$
$\longrightarrow_{\exists}$	$(\exists r.C)(x) \in \mathcal{A}$, but no z in $\mathcal{A}$ s.t. $\{r(x,z), C(z)\} \subseteq \mathcal{A}$	$\mathcal{A}' := \mathcal{A} \cup \{r(x,z), C(z)\}$
$\longrightarrow_{\forall}$	$\{(\forall r.C)(x), r(x,y)\} \subseteq \mathcal{A}$, but $C(y) \notin \mathcal{A}$	$\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$

The initial step of this tableau algorithm is the **construction** of an ABox, A_0, that it is made by **only one** concept assertion. $A_0 = \{C_0(x_0)\}$ where C_0 is the resulting expression of *expansion of the TBox* and then *negation normal form transformation*. This concept expression appear in ABox where we assert for individual name x_0 belong to the concept expression C.
If we find a model for this ABox A_0 we also find a model in which C_0 is **satisfiable** because in this model x_0 belong to C_0, so C_0 is *non empty*. Viceversa if there is **no model** for ABox then C_0 is unsatisfiable because if C_0 were satisfiable in that model there is *domain element* that belong to C_0.
Then we apply some **tableau rules** to build a *set of ABoxed* and in the end we terminate this "expansion" after a finite number of application of tableau rules. And after this we have *two possible cases*:

- An ABox produced by the algorithm is **complete**, so we **cannot** apply any other tableau rules. Complete application of tableau rules to this ABox and this ABox **does not** contain any *atomic contradictions*. This means that we have found a model for this ABox.
- In *every ABox* produced by tablau algorithm havs an atomic contradiction then we **cannot** find any model for this ABox. ABox is **incosistent**. When we have this contradiction, we say that it is a **clash**.

For **conjunction rule**, $\sqcap$, we add at the same time, *if* $C_1(x)$ **or** $C_2(x)$ *do not* belong to ABox, the two assertion $C_1(x)$ and $C_2(x)$.
For **disjunction rule**, $\sqcup$, we *do not* have already neither $C_1(x)$ nor $C_2(x)$ in the ABox, then we produce *two different* ABoxes. The are **indipendent** to each other. We need to study two different cases.
For **existential rule**, $\exists$, if in one Abox that we have there is no z element in that ABox such that $\{r(x,z), C(z)\} \subseteq A$ then we add to A $\{r(x,z), C(z)\}$. z is a new individual name. In general adding a element, so give more information than original one is an operation that it is consider *unsound*, violate property of sound. But for the purpose for which tableau algorithm uses this assertion, this is **not** compromising the property of sound.
For **universal rule**, $\forall$, if $C(y)$ not in A, we add $C(y)$ to ABox. Here the application is *completely correct* for the semantic viewpoint.

1.13 Exercise

TBox T:

$$\begin{cases} A \equiv E \sqcap \exists r.D \\ B \equiv \neg C \sqcap E \end{cases}$$

Concept expression

$$A \sqcap \neg B$$

We want to **check** if this concept expression is *satisfied* in the TBox. If there a model, so interpretation that satisfies the two axioms such that this concept expression has non empty interpretation.
We apply our techinque, 4 steps of the algorithm. The first step is *useless* because we have already start with right problem.

2) Unfolding w.r.t T of concept expression The unfolding w.r.t. means we replace concept that are defined in T with thir definition so with the right side of the corresponding equivalence.

$$(E \sqcap \exists r.D) \sqcap \neg(\neg C \sqcap E)$$

3) Negation normal form - NNF
It is this formula in NNF? *No*, because the $\neg$ operator is **not** applied to an *atomic formula* but it is applied to complex subformula. We have to put concept expression in NNF using transformation rules of NNF.

$$(E \sqcap \exists r.D) \sqcap (\neg(\neg C) \sqcup \neg E)$$

$$(E \sqcap \exists r.D) \sqcap (C \sqcup \neg E)$$

Now we have to do the last reasoning step.

4) Tableau algorithm
This concept expression turning a concept expression into **concept assertions** of over one individual that we can choose and then we apply the tableau rules to see what happen in the end. Now we shoud apply tableau to this concept expression to check satisfiability.
We start from an ABox $\boxed{A_0 = \{(E \sqcap \exists r.D) \sqcap (C \sqcup \neg E)(x_0)}\}$ so we have turned concept expression into a concept assertions, it means a statement for the ABox. Statement for the ABox **alway** said an argument concept assertion, in particular, the argument is the individual. We assert in this concept that x_0 belong to this concept expression. This is the assertion.
We are *forcing* the model to contain **one individual**, who is name is x_0, in the concept. This actually implies **satisfiability** because if we find a model for x_0 we can answer that concept expression is satisfiable because we can find a model when the concept is **non empty**.
Now we **must run** the algorithm, it means that starting from this ABox we build a *set of ABoxes* applying the *4 expansion rules*.

$$\textit{(and-rule)} \quad A_1 = A_0 \cup \{(E \sqcap \exists r.D)(x_0), (C \sqcup \neg E)(x_0)\}$$

Now we have 3 assertion in A_1 but the first assertion, $(E \sqcap \exists r.D) \sqcap (C \sqcup \neg E)(x_0)$, is now expanded so I **cannot** apply the (and-rule) of that formula. This concept assertion has been processes by the tableau so we **cannot** consider it again. But now we have two new formulas. So I need to decide which one I want to expand.

Which one of the two can be applied first?
Unfortunately there is **no predefined** order of application of the formulas. The tableau method is **not** really a **deterministic algorithm** because there is choice point. Let's consider the first formula.

$$\textit{(and-rule)} \quad A_2 = A_1 \cup \{E(x_0), (\exists r.D)(x_0)\}$$

Now $(E \sqcap \exists r.D)(x_0)$ **cannot** be expanded anymore. $E(x_0)$ is an *atomic formula*, formula that cannot be expanded. Let'n now choose to expand the second formula of A_2.

$$\textit{(}\exists\textit{-rule)} \quad A_3 = A_2 \cup \{r(x_0, x_1), D(x_1)\}$$

Tableau has **any expansion** rule for role assertion. They are consider *atomic formula*. Also $D(x_1)$ is atomic so we cannot be expand, so now we have only one choice to expand.

$$\textit{(or-rule)} \qquad \begin{array}{l} A_4 = A_3 \cup \{C(x_0)\} \\ A_5 = A_3 \cup \{\neg E(x_0)\} - \textbf{clash} \end{array}$$

Or-rule is rule that produce two ABxes instead of just one ABox. Tow different ABoxes that are **indipendent**, they should no be consider together.
In A_4 there are **no** formula to need to be expanded, because $C(x_0)$ is atomic formula and we **do not have** any formula to expand. We have *completed* the expansion of this ABox respect to tableau formulas.
In A_5 we are in the same situation, in which we **cannot** apply any expansion rule because the negation of atomic formula is consider atomic because for tableau rules there is no expansion rule for negation of atomic formula. Also A_5 is *complete*.
When we finish to expand the tableau rule the algorithm says that we should decided whether every ABox generated by the tableau is **open** or **close**. **Open** means **no conflicts** in the ABox, **close** means that we see conflicts or contradictions in the ABox. *Conflicts* means the presence of two atomic assertions of the form "C(a) and ¬C(a)". For the *same individual* we see in the same ABox the assertion "C(a)" and "¬C(a)". Of course, try to satisfies **both** formulas is impossible because we are saying something and the opposite at the same time. This means that you **cannot** find a model for this ABox, the ABox is consider **unsatisfiable**, **close**. We have to check **only** the atomic concept assertions and the negation of concept assertions.

Are there conflicts on A_4?
No, A_4 is *open*, **no clashes**, we do not find any conflicts.

Are there conflicts on A_5?
Yes, because we have $E(x_0)$ and $\neg E(x_0)$, so this is **close** because it contain a **clash**, conflict.

What we conclude from the tableau method?
If the method find at least one ABox that it is **not** close, then the algorithm returns **YES**, *satisfiable*. There is a a model for A_0. By the contrary, if all the ABoxed generated by the tableau are **closed**, they contain conflix, then the algorithm returns **NO**, *unsatisfiable*, so A_0 **does not** have any model.
Propagating the result of the method w.r.t original problem that it is *concept satisfiability*, of course the **overall** algorithm return YES, so the concept expression $A \sqcap \neg B$ is **satisfiable** w.r.t to the TBox T.
We need a mathematical prove that this techinque **always** produced a correct result w.r.t semantic and this is complicated because the semantic is complicated in the description logics, the interpretations are *infinite*, also models of the TBox are infinite. So, checking the property means that for every inputs we need to check that it is **always** true that we have a model. It means that there are **no wrong** answer provided by this algorithm. First, for the first step of expansion of this procedure is correct because of general property of classical logic that it is *equivalence substitution*, so I can replace one formula with the other without changing the satisfiability of the formula itself. Also for NNF, we just play with syntactic form of negation but we are not changing in any way the meaning of the formula. For tableau method there is an *inductive* proof that says that this method is correct.

How can we build a model for an open and complete ABox of the tableu?
We must *discard* all the complex formulas and look **only** for atomic formula of A_4.

How many individual names are there in the atomic formula of A_4?
I have 2 individuals
Model for A_0 :

$$\begin{gathered}\Delta^I = \{d_0, d_1\} \\ \underbrace{x_0^I = \{d_0\}}_{\text{interpretation of individual name}} \\ x_1^I = \{d_1\} \\ \underbrace{C^I = \{d_0\}}_{\text{interpretation of concept name}} \\ D^I = \{d_1\} \\ E^I = \{d_0\} \\ \underbrace{r^I = \{(d_0, d_1)\}}_{\text{intepretation of the role name}}\end{gathered}$$

I have build automatically for thw atomic formula of the ABox A_4. The tableau method **does not** give only the boolean answer but we the tableau says YES we can examine the ABox and extract one model for the initial ABox. Isomorphically generate a model, looking on the atomic assertions.
Everytime you find and *open and complete ABox* generated by the tableau you can create corresponding interpretation and check is **always a model** for the initial ABox. The interpretation that we have written is **not yet** a model for the original question because we have just interpreting the concept names C,D,E. The

concept names A and B are not intepreted in this interpretation, it is kind of *definition problem*. Because if I do not have interpretation of A and B, **how can I check that $A \sqcap \neg B$ is non empty?**
But there is a **way** of extending this intepretation interpreting also A and B. We can easily extend I to also interpret the concept names A and B to satisfy both the TBox T and the concept expression $A \sqcap \neg B$.

Lemma

1. If the algorithm returns "satisfiable", then the input concept has a model.
2. If the algorithm returns "not satisfiable", then the input concept has no model.
3. The algorithm terminates on any input

Corollary

$\mathcal{ALC}$-concept satisfiability and subsumption are decidable

Now we eill focus on the formal property of the we have defined for *ABox satisfiability*. The overall method is a **decision procedure** if it return the correct answer with respect to the semantic of the knowledge base. This means that if the ABox is satisfiable it *should* returns **YES** otherwise returns **NO**. But there is also a *termination* property, so the algorithm should **always** terminate for every ALC ABox in input. No **get stuck** or loop. As a *consequence* of the Lemma there is a **decidability property** for problem of *concept satisfiability and concept subsumption* w.r.t unfoldable TBox. *Decidability* means that it is possible to find an algorithm that **terminates** and it provides **correct answer always**. Decidability means that there exists decision procedure for this problem.

Soundness of the procedure:
is shown by local correctness of each tableau rule.

Local correctness:
Let $\mathcal{S}'$ be obtained from $\mathcal{S}$ by the application of a tableau rule.
Then $\mathcal{S}$ is consistent iff $\mathcal{S}'$ is consistent.

Completeness of the procedure:
Directly follows from the definition of a clash.

Soundness means that we have to prove that **true answers** of the algorithm are actually true. Everytime the algorithm says that the concept is *satisfiable* there exists actually a model of the TBox in which the concept expression is *non empty*.
Why correctness property is true for our algorithm? How can we prove this?
Essentially the real crucial part of the proof is in the *tableau method*. Because in the tableau method there is one rule, the $\exists$-rule, which is the *potentially dangerous rule* Because give a formula $(\exists r.C)(x)$, already appearing in our ABox, we are going to add something that it is a little more informative than what was written before. Because with $(\exists r.C)(x)$, it means that x participare in r with some other element in the role r and this other element belong to C. With tableau rule we give the name of that element. This is the **potentially dangerous rule** because we are going to add more information than the original knowledge base. However, it can be proven that this additional information that we add to knowledge base **doesn't change** the outcome of the *satisfiability problem*,**doesn't lead** this algorithm to answer YES or NO.
All the others tableau rules are *obvious* equivalent step in term of information content or semantic of the ABox produced by the tableau expansion. Also other previous step of the algorithm are easy to prove because they are all made of **rewriting** of concept expression that are equivalent to the one that are replaced.
Completeness property is more complicated to prove from mathematical formal logic viewpoint. Because we should explore all models, interpretations.

Role depth of concepts $d(C)$:

$$d(A) = 0 \qquad A \in N_C$$
$$d(\neg C) = d(C)$$
$$d(C \sqcap D) = d(C \sqcup D) = \max\{d(C), d(D)\}$$
$$d(\exists r.C) = d(\forall r.C) = d(C) + 1$$

Maximal nesting of quantifiers in a concept description.

sub-concept descriptions of concepts $sub(C)$:

$C \in sub(C)$

$C = \neg D$, then $D \in sub(C)$

$C = C_1 \sqcap C_2$ or $C = C_1 \sqcup C_2$, then $C_1, C_2 \in sub(C)$

$C = \exists r.D$ or $C = \forall r.D$, then $D \in sub(C)$

sub-concept descriptions of ABoxes $sub(\mathcal{A})$:

$$sub(\mathcal{A}) := \bigcup_{C(a) \in \mathcal{A}} sub(C)$$

The algorithm terminates since:

1. depth of the proof ABox bounded by $d(C_0)$.
2. for each individual, at most $\#sub(C_0)$ successors are generated
3. each individual has at most $\#sub(C_0)$ concept assertions
4. concepts are never deleted from node labels

How do we prove that the algorithm terminates on every TBox and concept expression?
The *unfolding process* and *NNF transformation* are steps that are quite easy to prove that terminates because NNF is a transformation that requires **linear time** (number of steps is proportional to the length of formulas that are process) and also unfolding, although exponential, is *finite* until terminating **finite time**.
What it is little bit more complicated to verify the termination is the *tableau expansion*. But after finite number of step we finish the expansion of all ABoxes produced by the tableau.
To prove this, we can say that every tableau rule simplifies the formula from the syntactic viewpoint, it **always** decompose the formula, and it is good intuition although it **not always** true or at least **not completely obvious** that leads to the termination.
On the picture there a *pseudoformal proof* of the termination. We can see **role depth** of concept d(C).
If A is a concept name then its role depth is 0. Then the role depth of negation is the same without negation of this concept expression and so on.
Role depth represent for a concept expression the *maximal nesting of quantifiers in a concept description.*

The idea of the proof is that the *role depth* expression produced by the tableau algorithm **always** decreases. In the end the role depth will be zero, and we have **no** existential role description or universal role description. This prove that tableau eventually ends this decomposition.
Sub-concept description of the concept is the collection of all the sub-concept expression inside concept expression.
Sub-concept description of the ABoxes we collect in the Abox all subexpressions that appear in every concept assertions on the ABox.

There is a **bound** for the depth of all the formulas produced by the tableau. Tableau **never** increase the role depth in its expansion rules.
Most of the rules **reduce** the role depth, we can prove important property, 2), so that **at most** a finite number of successors we generate, where successors are individual that appear after the individual that we have consider. For instance, in $\exists r.C(x)$ with tableau rule we add $r(x, z), C(z)$, z is the *successor* of x. Also the successor of z is a successor of x because property is transitive.
Also for assertions, 3), we have a *bound* but in this case in the case of assertions, so the ABoxes generate by the tableau are finite.
4) is important because the tableau method *monotonically* increases the sixe of the ABox but **never** delete formulas. This is important in termination because deletion should anticipate the termination or could loop by adding and deleting the same formulas.

Complexity of unfolding: exponential

Complexity of transformation into NNF: linear

Complexity of application of tableau rules: polynomial space

$\mathcal{A}_0$ $\mathcal{A}_1$ $\cdots$ $\mathcal{A}_{\#\mathsf{sub}(C_0)}$

- all ABoxes need to be considered, but only one at a time
- the whole tree may be generated, but only one path needs to be stored

We know that after a *finite number of steps* this algorithm **terminates**. But now the problem is how complex it is.

What is the computational cost of running the reasoning algorithm of unfoldable TBox?

The cost is very high.

The *unfolding step*, in which we use the concept definition as rewriting rules for the concept expression, can produce **exponential** blow up of the concept expression. The *NNF step* is **linear**.

The cost of the *tableau method* run in **polynomial space**, complexity class in which the algorithm can use only a polynomial amount of memory, this **does not** mean that the time to take the algorithm is polynomial. Running the tableau is also an **exponential time** computational cost. Also in tableau algorithm requires *exponential space* and **not** polynomial space. we are taling about the *worst case*. The choice of the ABoxes to expand is very important for computational cost because may be one could lead use to arrive an answer without looking at another one and the time descrease. If we are unluckly and, we make the wrong choice we need exponential time to have our answer.

- simple expansion does not work in the presence of GCIs:
  - replace a name by which part of the TBox?
  - cyclic axioms: termination?

- Applying the GCIs like rules does not work either!

$$\exists r.(C \sqcap \exists s.D) \sqsubseteq \neg E \sqcup \exists r.D$$

'Precondition' may never appear at relevant element

- Recall: GCIs hold at every point in the model
  → new tableau rule for GCIs needed

The algorithm that we have just seen is not for all knowledge bases that we have write in ALC, it works **only** for TBoxex that are unfoldable. But in DL we can also write cyclic statements in the TBox, GCIs.

If we have cyclic axioms in the TBox we **cannot** apply the technique that we have already defined.

Why?

Because the *unfolding* would **never** terminates.

This means that GCIs **cannot** be unfolded.

What do we do?

We cannot apply the same technique because the unfolding will get into infinite expansion. But a part of cyclicity we have another obstacle, because we **don't have names** but we have expression so semantically it could be equivalent to infinite other concept expressions. Because, see example in left, we could switch C and $\exists s$.D, and semantically is equivalent of $C \sqcap \exists s.D$ but syntactically they are different.

Another problem is that $\exists r.(C \sqcap \exists s.D)$ can **never** appear in any other formulas but this rule could have an effect. It is impossible to just extend the idea that apply GCIs like rewriting rules.

The conclusion is that we cannot **get rid** of the TBox before arriving at the tableau. We **cannot** encode in a pre-processing step all the *semantics* of the TBox into the concept expression in the case of GCIs.

We **change the tableau method**.

Tableau rule for GCIs

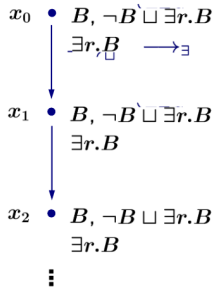
1. Code all GCIs into one.

For $\mathcal{T} = \{ C_1 \sqsubseteq D_1,\ C_2 \sqsubseteq D_2, \ldots,\ C_n \sqsubseteq D_n\}$
build the GCI $\top \sqsubseteq C_{GCI}$ with
$C_{GCI} \equiv (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$

2. Assert C_{GCI} for every individual: new tableau rule

$\longrightarrow_{\top \sqsubseteq C_{GCI}}$: If x in $\mathcal{A}$ and $C_{GCI}(x) \notin \mathcal{A}$,
then replace $\mathcal{A}$ with $\mathcal{A}' = \mathcal{A} \cup \{C_{GCI}(x)\}$

Consider: $\mathcal{T} = \{B \sqsubseteq \exists r.B\}$
with $C_{GCI} = \neg B \sqcup \exists r.B$



Remedy:
Block of application of $\longrightarrow_\exists$

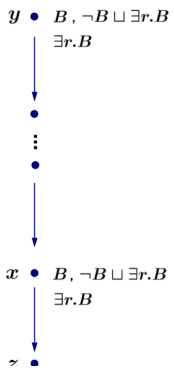
An individual x is directly blocked
by an individual y, iff:

- there is a path from y to x in $\mathcal{A}$
- x was generated by $\longrightarrow_\exists$ after y
  'y is older than x.'
- $\{C \mid C(x) \in \mathcal{A}\} \subseteq \{D \mid D(y) \in \mathcal{A}\}$

An individual x is indirectly blocked if:

- there is a path from y to x in $\mathcal{A}$
- y is directly blocked

An individual x is blocked
if it is blocked or indirectly blocked.



We **do not** have the *unfolding step* anymore. The we do the *NNF transformation* and finally we run the tableau. but tableau contain a **new rule**, a rule that takes into account the TBox. Given a TBox compose by concept subsumption or GCIs we build directly the concept expression C_{GCI} that it is obtained turning every concept subsumption axioms into the concept with $\sqcup$ and we make an $\sqcap$ between every formulas in the TBox, see figure. This is **not** an axiom of the TBox it is a definition. The we add the rule for C_{GCI} expression. Essentially, we have to assert that every individual appearing into the ABox belong to C_{GCI} expression. This solve the presence of general concept inclusion, GCI, in the TBox.

Adding this rule affects the termination property.
We can see that if we have $B \sqsubseteq \exists r.B$ with
$C_{GCI} = \neg B \sqcup \exists r.B$.
We start with ABox $B(x_0)$, so this formula is atomic it *cannot* be expanded by any other rules. For the property that we said before we must add C_{GCI} to x_0.
$A_0 = B(x_0)$

$$A_1 = A_0 \cup \{(\neg B \sqcup \exists r.B)(x_0)\}$$

$$\textit{(or-rule)} \quad \begin{array}{l} A_2 = A_1 \cup \{\neg B(x_0)\} - \textbf{clash} \\ A_3 = A_1 \cup \{(\exists r.B)(x_0)\} \end{array}$$

$$(\exists - \textit{rule}) \quad A_4 = A_3 \cup \{r(x_0, x_1), B(x_1)\}$$

Now $B(x_1)$ is an atomic formula but unfortunately we are **not** done because C_{GCI} rule come to play again because we have introduced new individual in the ABox and we must assert C_{GCI} to x_1. So we repeat same things done for x_0. We **always** generate a new individual, we have *loop* again, exactly in the unfolding case fo GCIs, but now the *infinite expansion* is inside the tableau. If we **always** choose a different name of individual for the successor we **never stop**. And this is **not** wrong, there is in fact a model for this TBox and ABox that it is exaclty like this, it is a *infinite chain* of individuals, all related by an r, and every individual belongs to the concept B. In the possible world it is **not** a problem, we can generate infinite domain domain of interpretations with infinite number of elements and we can say that all elements that belong to B are infinite and belong to role r.

How can we solve this problem?
Generate a *blocking condition* in the application of exist rule. The application of exist rule is **not** conditioned **only** by the existence already of the assertion that we are going to introduce like the standard tableau but in the *tableau of* C_{GCI} we have a **blocking condition** that is able to detect situations that are an exact replay of the previous situation generated in the Abox. We stop the expansion when the method realize that it is just reapiting over and over the same action, same steps. Remember that we are try to build a *open and complete* Abox but when we have cycle we build and infinite ABox. At infinite time the ABox that we obtain will be **open**.

If it realize that it are just repeating the same things that we have done before and these things **doesn't generate** conflicts it can understand that it goes on forever like this and make the same choices it will **never** makes conflicts. We arrive at the situation in which the same formula that were contained by the predecessor is also the formula that we apply. See the case y and x in the previous example. And we see that we **don't** produce any conflict, so I can block it becaause it would produce the same expansion that I have already obtained previously.

Replace the exists rule $\longrightarrow_{\exists}$ by a exists rule with blocking $\longrightarrow_{\exists\Box}$:

	Precondition	Replace $\mathcal{A}$ by:
$\longrightarrow_{\exists\Box}$	$(\exists r.C)(x) \in \mathcal{A}$, and x is not (indirectly) blocked but no z in $\mathcal{A}$ s.t. $\{r(x,z), C(z)\} \subseteq \mathcal{A}$	$\mathcal{A}' := \mathcal{A} \cup \{r(x,z), C(z)\}$

The final tableau method for GCIs says that we replay the original GCIs rule with one with *blocking*. Apply $\exists$-rule only to individual that **are not** blocked.

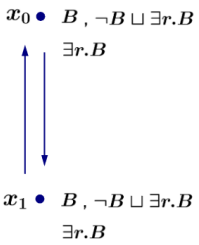
Have we obtained a model?

Some role-successors are missing in the 'blocked' ABox!

Build model w.r.t. blocking:

How to obtain a model for:
$\mathcal{T} = \{B \sqsubseteq \exists r.B\}$?

Introduce 'back links'.



Why new method of tableau of GCIs works?
We just say it is **not necessary** to make infinite expansion even though, in principle, it may lead in the end in correct answer. However, even in presence of the *blocked* **non expanded** existential fprmula we still can build a model for the ABox and TBox.
Because of the blocking we **don't** know the successor of x_1 we have a problem. We solve it put and arrow back, *back link*, so we can generate a model *going back* to the predecessor.
Instead of creating a new individual, that is a successor of x_1 w.r.t r and satisfies B, we use the precedessor of the duplicate x_1. We can still extract a model from a open ABox with blocking condition.

Soundness of the procedure:
is shown by local correctness of each tableau rule.

Local correctness:
Let $\mathcal{S}'$ be obtained from $\mathcal{S}$ by the application of a tableau rule.
Then $\mathcal{S}$ is consistent iff $\mathcal{S}'$ is consistent.

We can repeat and prove formally the three properties of decision procedure. The tableau is still *sound* and *complete* method and it **always** *terminate* because of the **blocking condition**. Blocking condition prevent from provide infinite branching.

Completeness of the procedure:
Directly follows from the definition of a clash.

Tableau method is a **non deterministic** method.

Let $\mathcal{T}$ be a TBox and C a concept description.
The interpretation $\mathcal{I}$ is a finite model of C w.r.t. $\mathcal{T}$ iff

- $\mathcal{I}$ is a model of $\mathcal{T}$ and
- $C^{\mathcal{I}} \neq \emptyset$, and $\Delta^{\mathcal{I}}$ is finite.

Theorem:
$\mathcal{ALC}$ has the finite model property.
i.e., if $\mathcal{T}$: $\mathcal{ALC}$-TBox and C: $\mathcal{ALC}$-concept description such that C is satisfiable w.r.t. $\mathcal{T}$, then C has a finite model w.r.t. $\mathcal{T}$.

Finite model property is *theoretical property*. An interpretation is called *finite model* **if and only if** it is a model and the domain of the interpretation is **finite**.
Everytime a concept expression is *satisfiable* there exists a finite model for this concept expression or for knowledge base, and this called **finite model property**. The fact that we can have infinite interpretations in our semantic, **doesn't change** the satisfiability property.
Why?
Because everytime a knowledge base has a model there is also a *finite model*.