A LIGHT-WEIGHT DESCRIPTION LOGICS

EL

As we have seen, EL is an inexpressive DL allowing only ∏ (conjunction) and ∃ as constructors

We consider a slight extensions to handle disjointness

**Syntax of EL⊥**

The main building blocks of EL⊥ (and all DLs) are:

* concepts unary predicates. Express classes. Label the notes of the graph
* Roles binary predicates. Express relationship. Edges between objects

Concepts: represent classes of objects

Roles: set of pairs, relation between objects

For now, we will ignore constants (we will use them again with ALC)

**Syntax of EL⊥ II**

Consider infinite, disjoints sets

* Nc concept names. Like *female*
* NR role names. Like *hasChild*

Formally they are infinite but they are unbounded, there are arbitrary many

They are disjointed, we can identity concept names and role name

Concepts are capital letters, usually the first letter of the alphabet (A,B,C)

Role names are lowercase called (r,s,t…)

EL⊥ concepts are defined by (A € Nc (concept name), r € NR(role name))

C: concepts

Bars separate different possibility

**C :: = A | T | ⊥| C ∏ C | ∃ r.C**

C: class of all concepts concepts names, sometimes are called complex concepts to differentiate from concept names

Bars (|) separate different possibility

A: concept name

T: tautology (always true)

⊥ : contradiction

C: is a general concept, the two C can be different

*The class of all contracts can be A (every concepts name is a concepts), T is a concept, ⊥ is a concept, if you have two concepts ( C ) the conjunction of two concepts (C ∏ C) is a concept, r is a role nome and ∃ r.C is a concept*

∃ r.C (like there's exist a child is a female)

* A = every concept name is a concept
* T and ⊥are concepts
* If C and D are concepts then C ∏ D (conjunction of C and D) is a concept
* If C is a concept and r a role name, then ∃ r.C is a concept

**Example**

* Female is a concept
* T is a concept
* ∃ hasChild.Female (hasChild is a role name, Female is a concept, so ∃ hasChild.Female is a concert)
* Female ∏ ∃ hasChild.T it is a concept. Refers to the class that have a female class
* ∃ hasChild.Female ∏ ∃ hasChild.Male (Male is a concept, that total is a concept). Class of people that have one female and one male
* ∃ hasChild.(Female ∏ ∃ hasChild.Male) is a concept . People that have a child that is a female that have a male child. Have a daughter and that daughter have a son
* ∃hasChid.⊥ is a concept (it actually is a contraction) people that have a child that cannot exists
* …

**Semantics of EL⊥**

The fundamentals:

* **concept** (names) are unary predicates sets. They are interpreted as sets
* **Role** (names) are binary predicates pairs. Binary relationship
* T is a tautology always true. We will interpret it as a problems that holds for all the individuals
* ⊥is a contradiction always false. Is the empty set, no object can satisfy this property
* ∏ is the conjunction In logical terms it is ^. The conjunction of two property is satisfied iff both properties are satisfied
* ∃ is about role successor. Is a binary predicate, property about the successor of a role. It is a sequence of arrows

**Semantics of EL⊥ II**

An interpretation is a pair I = (Δi, •i) where

* Δ i is a non-empty set called **domain**, set of objects
* •i is the **interpretation function** which maps
  + Every A € Nc to a set Ai ⊆ Δi. If A is a concept name, than the interpretation of A is a subject of the domain
  + Every r € Nr to a binary relation ri ⊆ Δi x Δi

The interpretation function is extended to complex concepts:

Ti = Δi (T is tautology, implicitly we know that all objects satisfy top). The interpretation of top is the interpretation domain

⊥ i = ø no objects satisfy a contradiction. The interpretation of bottom (⊥) must be the empty set

(C ∏ D)i = Ci ∩ Di must satisfy C and D, must belong to both set so it must belong to the interception

( ∃ .r.C)i = {δ € Δi | ∃η € Ci. (δ, η) € ri}

↓

the interpretation of ∃r.C must be a set of the domain, but they have to satisfy some properties. Find an interpretation η that is reachable from δ using an r-edge. Check that η belongs to the concept C

**Example**

Δi= {δ , η, θ , k}

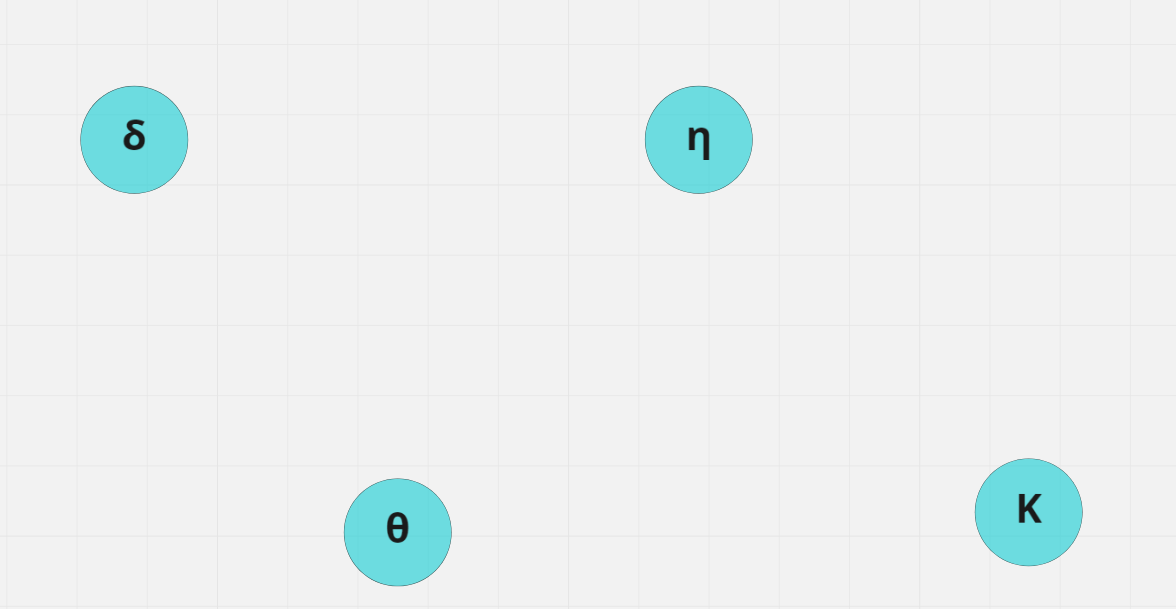
Ai= (θ , k) θ , k satisfy the property A

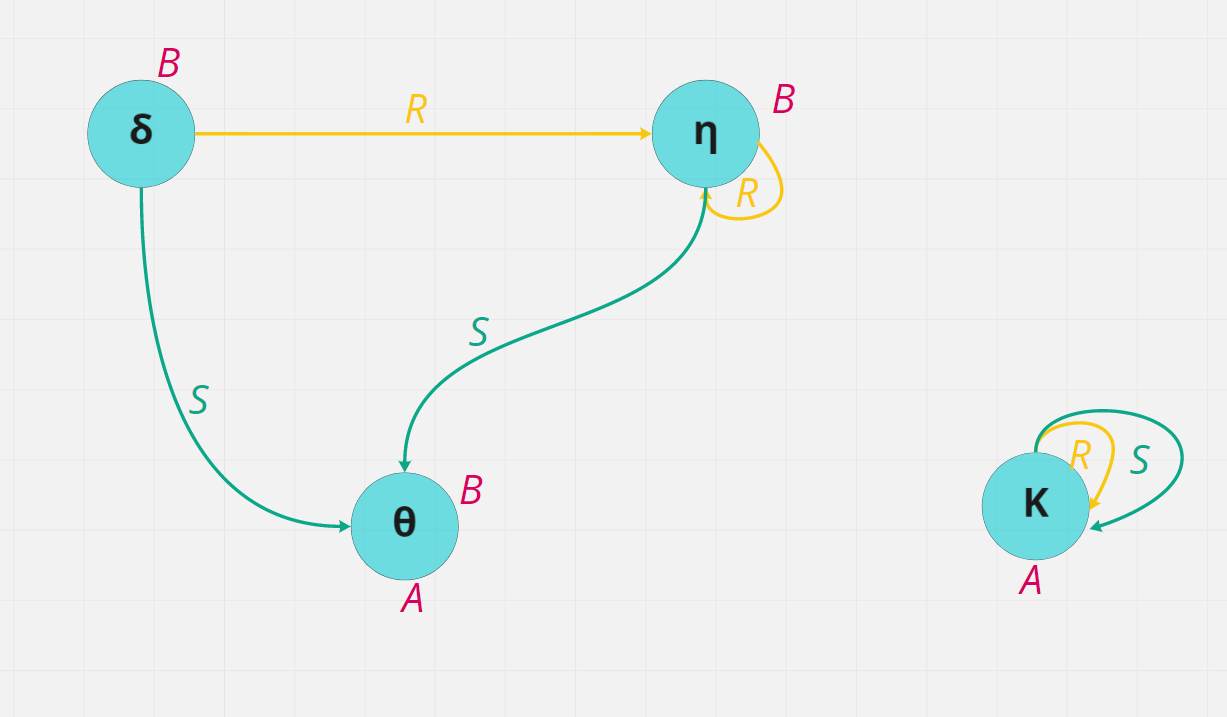
Bi = (δ, η, θ) satisfy B

θ satisfy both

r*i* = {(δ, η ), (η, η), (k,k)}

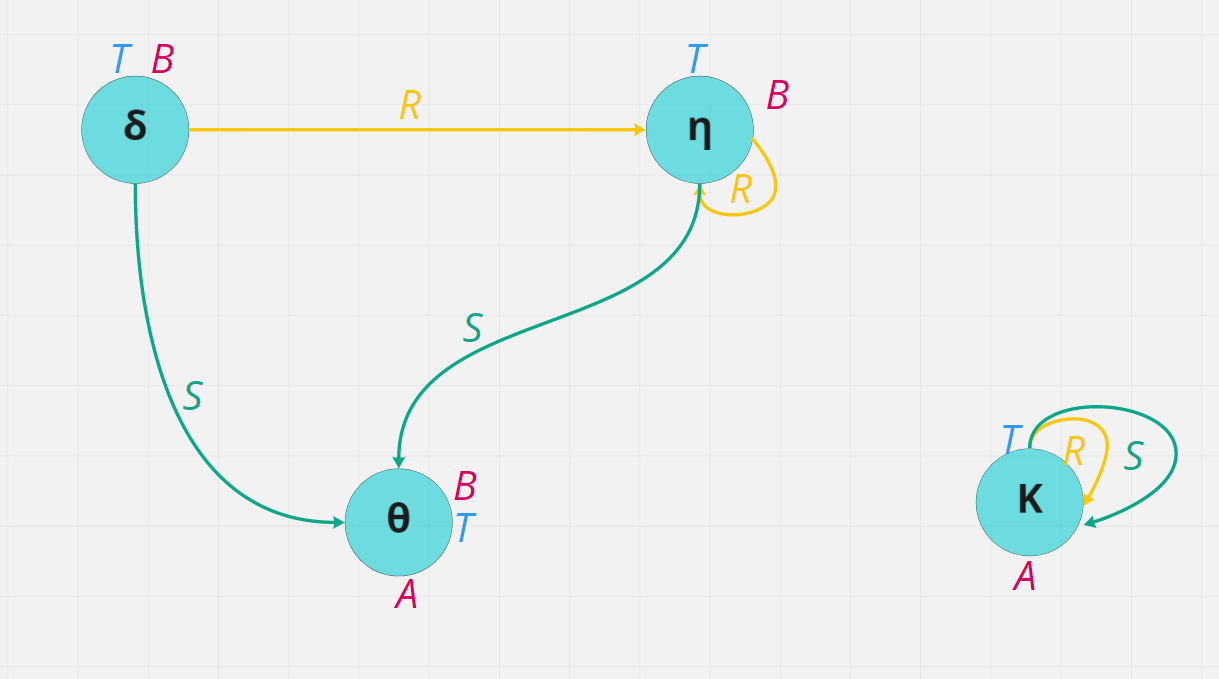
si = {(δ, θ), (η θ) (k,k)}

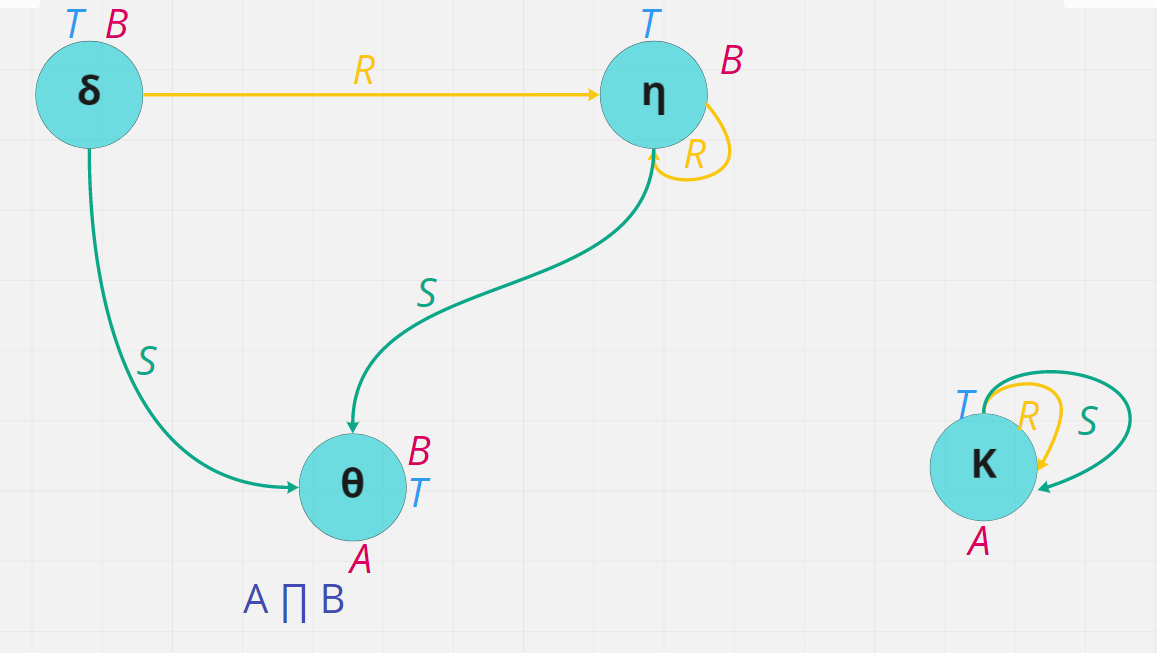




Domain have 4 objects

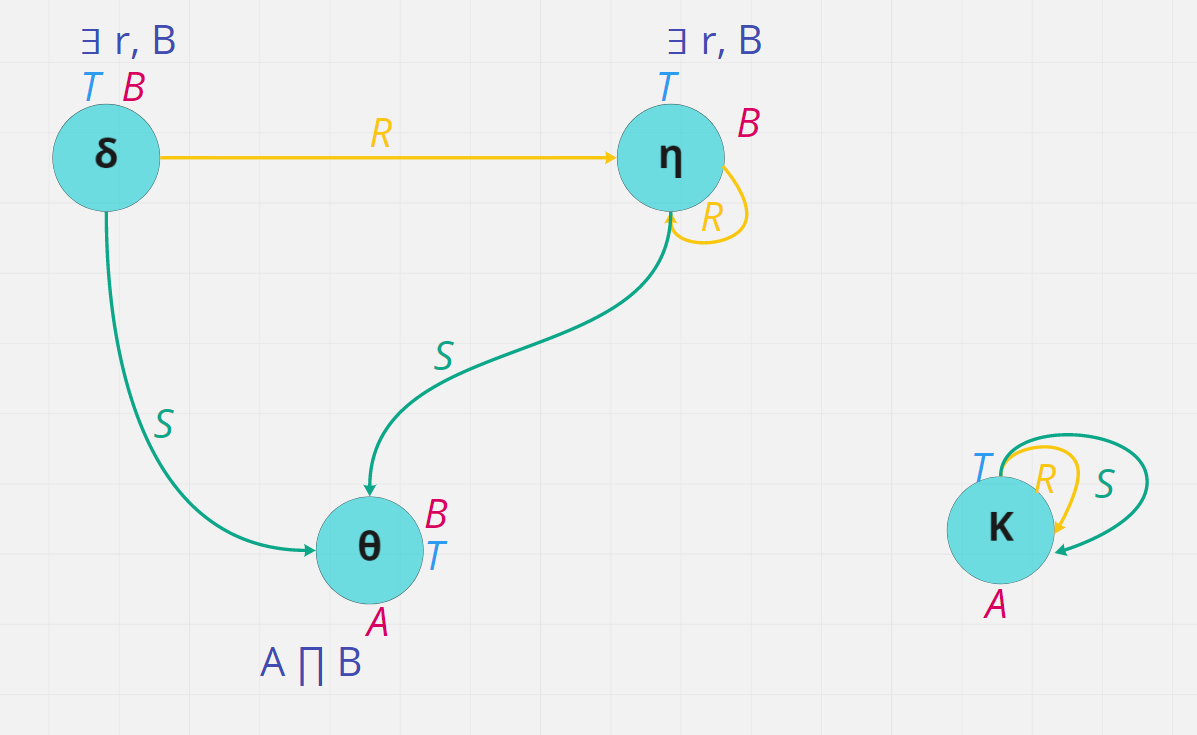
All objects satisfy top T



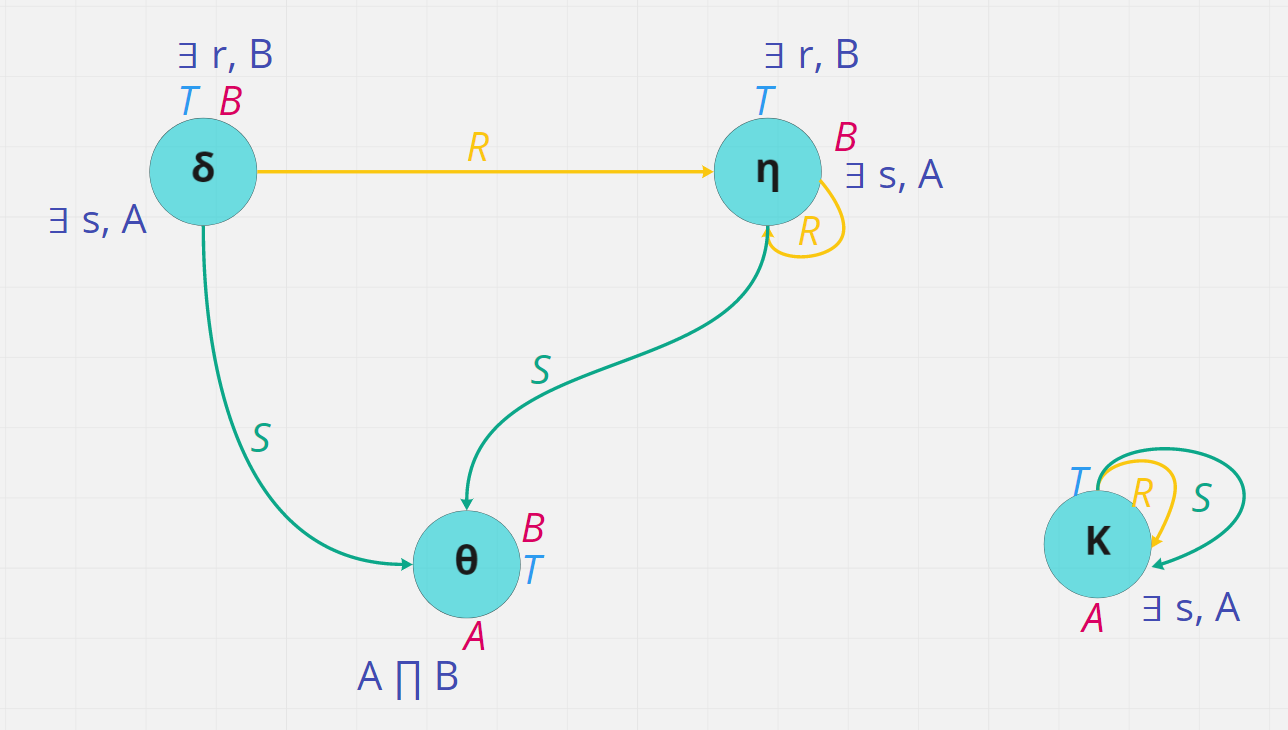


A ∏ B belong to θ

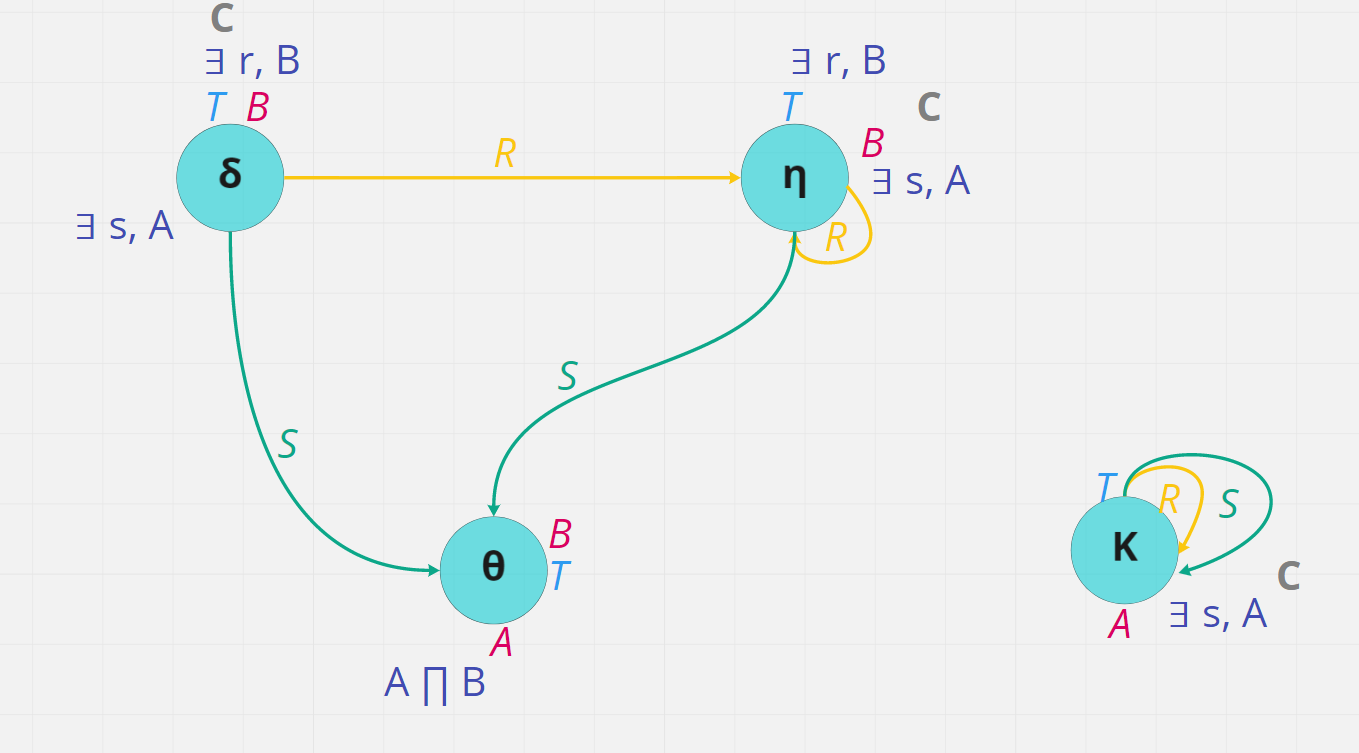
∃ r, B things that can reach an object that belong to B. Have an r edge that reach B



∃ s. A s successor tha belongs to A



C = ∃ r ∃ s.A C is the complicated concept since we already find out ∃ s.A . Things that we can reach with r that goes to ∃ s.A



**First reasoning problem**

A concept C is **satisfiable** iff there is an interpretation *I* such that Ci ≠ ø

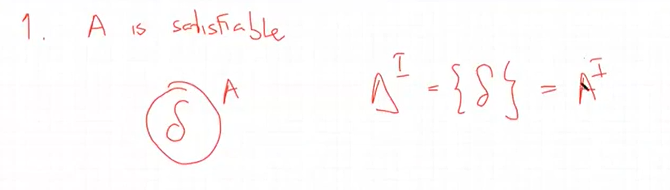
A concept (or a complex one) is satisfiable if we can find some interpretations that doesn't make it empty

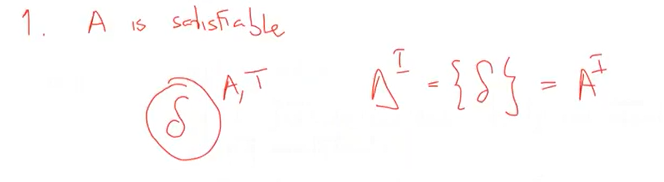
We show that this problem is **trivial** in EL⊥

Deciding whether a concept is satisfiable or not is a trivial problem

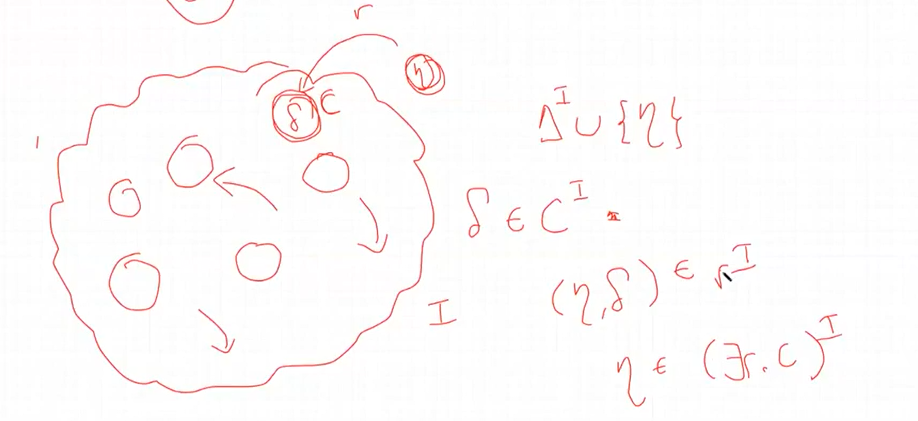
**INSIGHTS**

**Satisfiability**

* every concept name A € Nc is satisfiable. Satisfiable means that I can find at last one interpretation that interpret the concept
* T Top is satisfiable (in the same example it also belongs to T). The domain is always not empty



* If C (an arbitrary concept) is satisfiable, then ∃ r.C is satisfiable



* if C, D are satisfiable the C ∏ D is satisfiable !

**Unsatisfiable**

* ⊥ is unsatisfiable
* If C is unsatisfiable then ∃r.C is unsatisfiable
* If C is unsatisfiable than C ∏ D is unsatisfiable. Intersection the empty set with another set is the empty set

**Trivial of satisfiability**

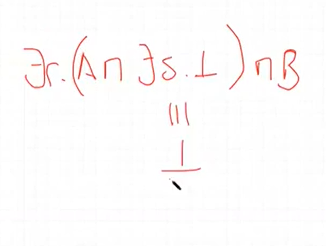
Call EL the sublogic of EL⊥ without ⊥

An EL⊥ concept C is satisfiable iff it is an EL concept (if it uses no ⊥ )

**…In other words…**

Any EL⊥ concept with bottom as a **subconcept** is equivalent to botton

Bottom is the concept that is interpret with the empty set



if we have a concept that have ⊥ it is equivalent to ⊥ , we can avoid more complicated structure

**Concept subsumption**

Let C and D be EL concepts (no ⊥ )

C is **subsumed** by D iff for all interpretations I Ci ⊆ Di. For any possible interpretation make the interpretation of C a subclass of the interpretation of D

We will denote this as C ⊑ D

Whichever interpretation we give the interpretation of concept C will always be a subclass of the interpretation of the concept D

We excluded ⊥ because if we have it become trivial to check.

25 October 2021

**Example**

A ∏ B ∏ C ⊑ A ∏ C

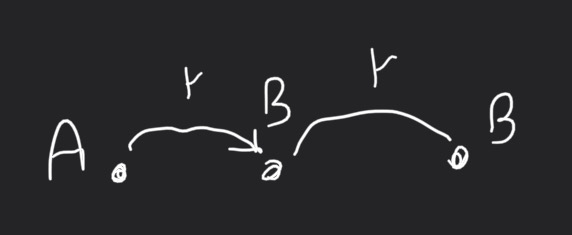
A ∏ B ∏ C belongs to all of three, they are intersection, so it is a subclass of A ∏ C

C ∏ D ⊑ C If I have an arbitrary set C intersect a set D it is a subset of C

(C ∏ D)i = Ci ∩ Di ⊑ Ci

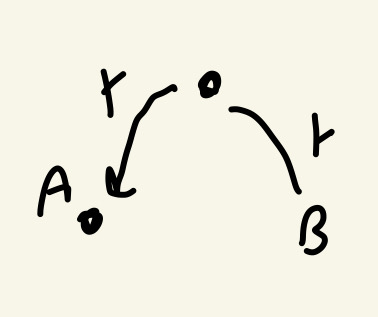
∃ r. (A ∏ B ) ⊑ ∃r.A ∏ ∃r.B If I have a child that is a student worker than I have a child that studies and a child that works

A ∏ ∃ r. (B ∏ ∃ r.B) ⊑ ∃ r. ∃ .r. T: If I belong to A and have an r successor that is B and also have an r successor that is in B than I have an r successor that have an r successor than have any arbitrary properties



∃ r.A ∏ ∃ r.B ⋢ ∃ r.(A ∏ B)

*If I have a child that is male and a child that is female I must have a child that is male and female at the same time*. It is inutiltelvy wrong



If one is a subset of another doesn't need the opposite is also true

To check if one is subsumed by another we build a tree that represent the concepts

**Concept trees**

Every EL concept can be represented by a (labelled) tree:

* nodes are labelled with concept names.
* Branches are labelled with role names

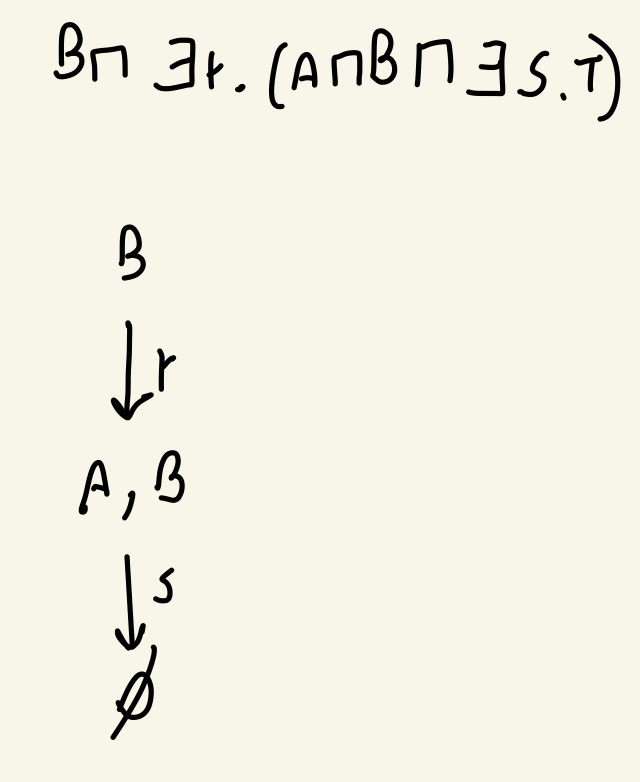
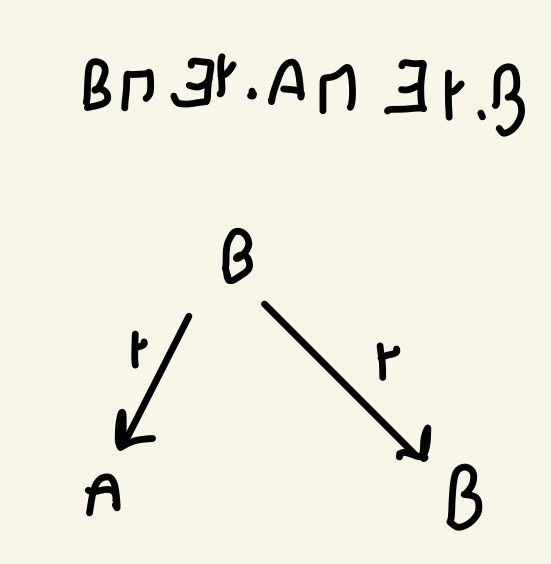
Built recursively on the structure of the concept

**For each** top level conjunction X:

* if X € Nc label the root with X
* If X is ∃ r.D create r-child with the concept tree of D

**Example**

B ∏ ∃ r.(A ∏ B ∏ ∃ s.T) B ∏ ∃ r.A ∏ ∃ r.B

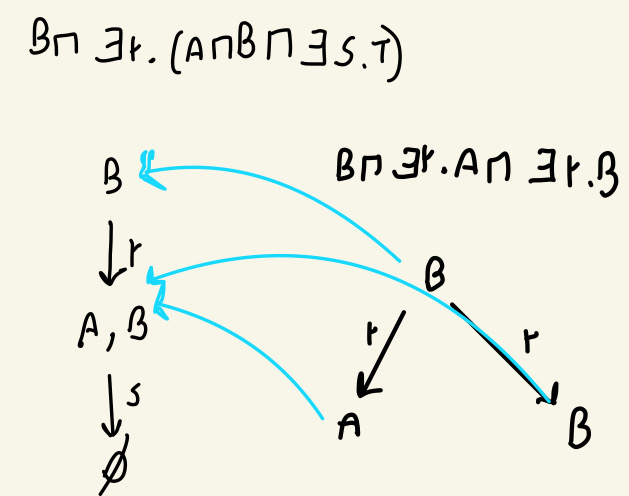
Sort of a **Canonical model** of the concept

**Subsumption through homomorphism**

A **homomorphism** between two concept trees T1 and T2 is a function h mapping

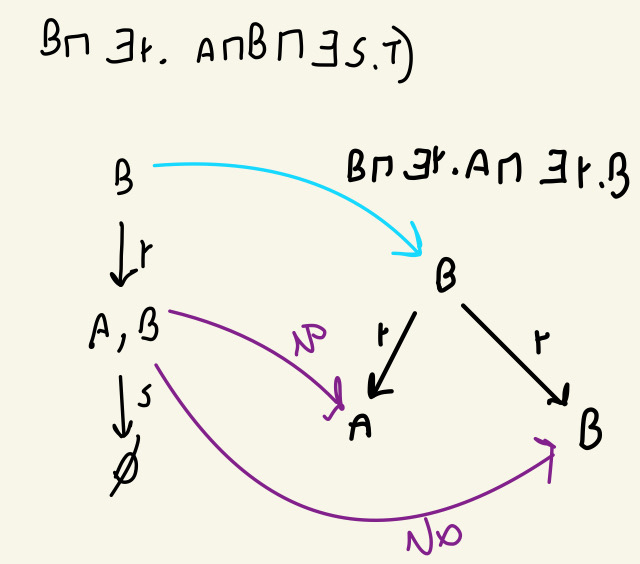
Each node in nodes(T1) to a node in nodes (T2) such hat:

* h(root(T1)=root(T2)) (maps the root to the root). Every element belong to both trees
* For every n € nodes(T1), label(n) ⊆ label(h(n)) (label of the image is increasing). Put more constraints in the image. If the node is A it needs to be map to at least A
* If m is an r-successor of n, then h(m) is an r-successor of h(n) (successor preservation)



homomorphism from the tree to the right to the one on the left

Check opposite direction



The images must have all the elements, it does not satisfy the condition of the construction, can’t build a homomorphism from the tree on the left to the right

**Theorem**

C is subsumed by D iff I can find a homomorphism from T(D) to T(C)

Everything that satisfy the images must satisfy the object

**Proof (Idea)**

Homomorphic image simply **adds** constraints to an object

If δ satisfies all constraints in T(C) then is also satisfy in T(D)

T(C) is more constrained than T(D)

All objects satisfying T(C) also satisfy T(D) thus all objects in Ci belong to Di. This is a **subsumption**

Ci ⊆ Di

**Complexity**

Deciding subsumption requires polynomial time on the size of the concepts.

Subsumptions mean homomorphism we know that we have the map the root to the root and every time we have an r we need to try all possibility

We can check subsumption in a stronger resolut. We can check subsumption with respect to background knowledge

**Equivalence**

Two concepts C and D are equivalent if C subsumed D and D subsumed C. (C≡D)

they are interpreted in the same way, can’t be distinguished

Two concepts are equivalent if:

All the interpretation interpret them as the same set

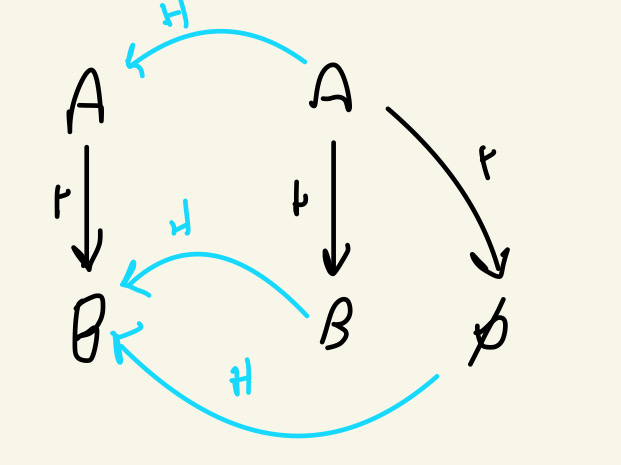
C ⊑ D and D ⊑ C

**Examples**

∃ r.( ∃ s. A ∏ ⊥ ) ≡ ⊥ any concept that have ⊥ is equivalent to ⊥

A ∏ B ≡ B ∏ A Conjunction is commutative

A ∏ ∃ r.B ≡ A ∏ ∃ r. B ∏ ∃ r. T the concept on the right is subsumed to the concept on the left. In the other direction having an r successor to T is a weaker property that ∃ r. B is not adding any constraint. Check example ↓



Empty set is a upset of B

A ∏ ∃ r.B ≠ A ∏ ∃ r. B ∏ ∃s. T if we have an s successor the right is a subclass of the one on the right. No homomorphism

A ∏ ∃ r.B ≠ A ∏ ∃ r. T

Must belong to A and r successor that belong to B, the right one is not a ⊆ of the one on the left. Left is a subclass of right but not vice versa

**From concepts to Knowledge**

So far, we have described concepts and their relationship

But how do we express knowledge?

Through constraints on the relevant interrelations (TBoxes)

**The terminological box**

The terminological box (TBox) imposes restrictions on the potential interpretation of concepts

For examples *Platypus ⊑ mammals* requires that the class of platypus is always contained in that of mammals

interpretations which do not satisfy this are **ignored**.

When I will do reasoning they will not be take to account

**TBoxes**

A (EL⊥ ) **general concept inclusion (GCI)** (also called axiom) is an expression C ⊑ D

With C and D EL⊥ concepts

A TBox is a finite set of GCIs. Impose a fine set of restrictions

The GCI will be see as a restriction

**TBox semantics**

An interpretation I satisfies the GCI C subsumed D iff Ci subsumed Di

I is a model of the TBox T iff is satisfies all GCIs in T

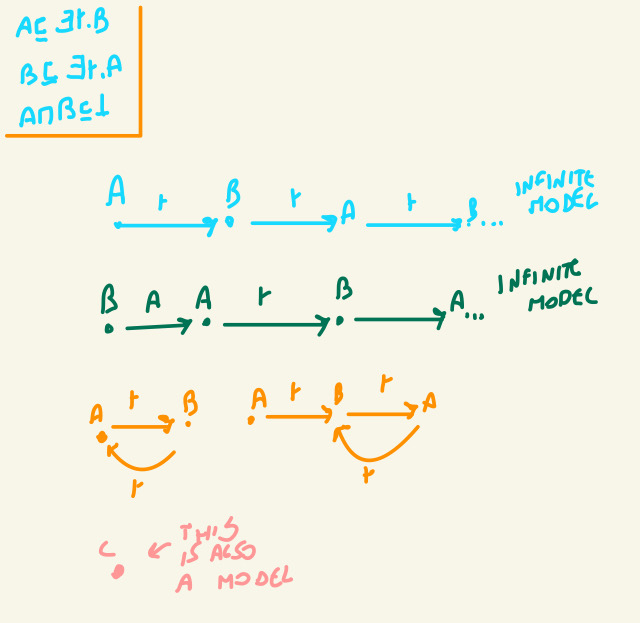
**Example**

Consider the TBox T with the GCIs

A ⊑ ∃ r.B

B ⊑ ∃ r.A

A ∏ B ⊑ ⊥



Anything that will put a node with A and B will not be a model

**TBox reasoning**

In the presence of a TBox T reasoning is **restricted** to the class of modes only. Think only about the preparations that satisfy the constraint

* Concept satisfiability (with respect to T) is there a **model** i of T such that Ci ≠ ø ?

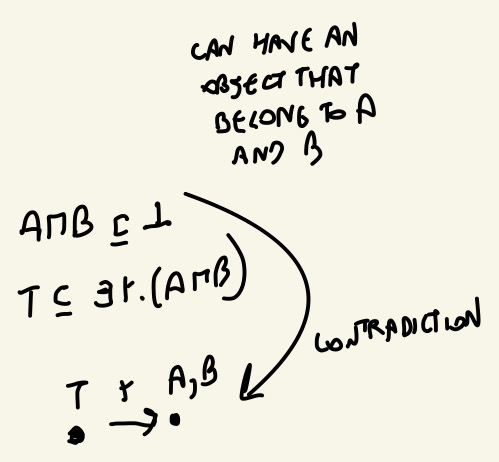
With respect to the example before A ∏ B is unsat, interpret as the empty set

With respect to T ∃r.(A ∏ B) is unsat

* Concept subsumption (with respect to T) does Ci ⊆ Di hold for all **models** i of T?

A ⊑ ∃ r. T with respect to T

TBox consistency does T have a model?



Can satisfy both axiom, it is inconsistent

Need T and ⊥ to make a TBox inconsistent

A ∏ B ⊑ ⊥ no intersection between A and B

**Consistency vs. Satisfiability**

Inconsistency: refers to the Tbox, all set of axiom together, ask if there is a model of not

Unsatisfiability: there is a model that makes it not empty.

If a TBox is inconsenty, every concept is unsat

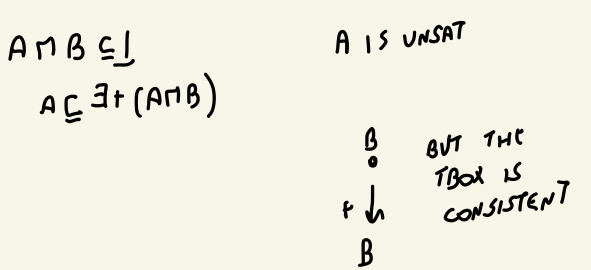
If a concept is unsat doesn't mean that the Tbox is inconsistent

Note that concepts may be unsatisfiable with respect to consistent TBoxes

Satisfiability: concepts that we built and whether there is a model that makes it not empty

A ⊑ ∃ r. (A ∏ B) A ∏ B ⊑ ⊥

Cannot be an object that belong to A, A is empty set, but there is still a model



T has a trivial model, but A is unsatisfiable

**Forget consistency**

C is subsumed by D iff for every model I,Ci ⊆ Di

T is not subsumed by D if I can find a model that violates that proof.

T is not subsumed by ⊥ iff there exist a model I such that Δi = Ti ≠ ø = ⊥i iff there exist a model i of T iff T is consistent

If we can decide subsumption, we can also decide consistency

**Forget satisfiability**

The concept C is satisfiable iff C is **not subsumed** by ⊥

Not subsumed: if it not true that all models made Ci a subset of ⊥

If we can decide subsumption, we can also decide satisfiability

NOATION: We focus on the former T ⊢ C ⊑ D or C ⊑T D

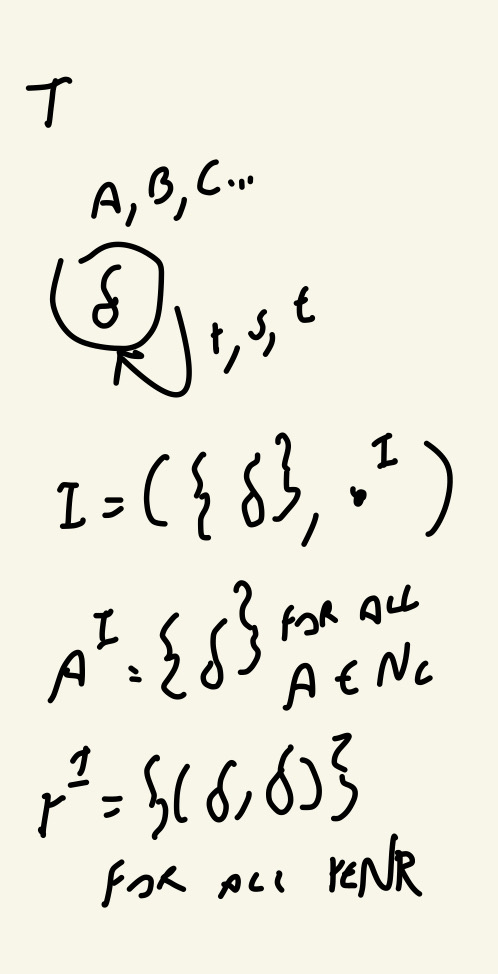
Our goal is to find an algorithm that decide subsumed relationship

**Simplification without ⊥**

An EL TBox is an EL⊥ without ⊥

* every EL TBox is consistent, whatever TBox I build is consistent
* Every EL concept is satisfiable with respect to an EL Tbox

an interpretation need to have at least one element

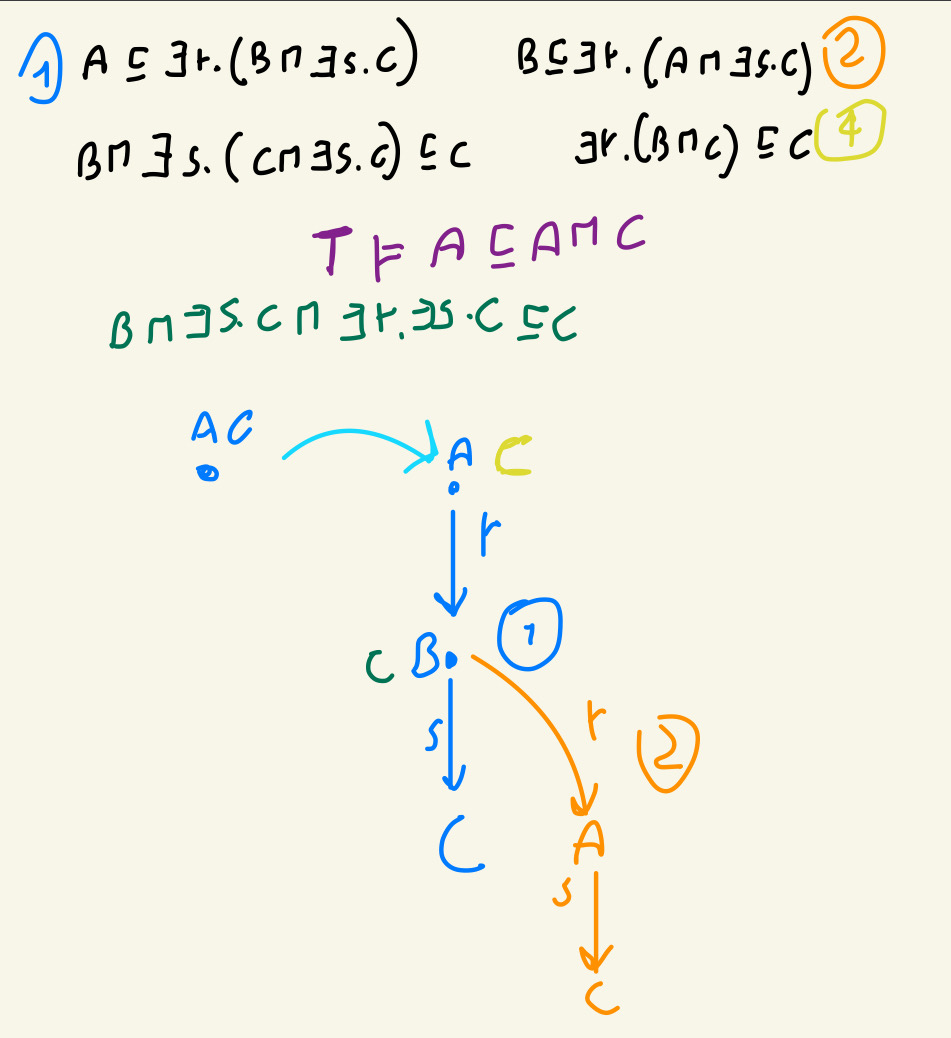


**Deciding subsumption**

Homomorphism approach does not apply directly

* GCIs add objects to the concept tree
* How to avoid infinite constructions?

**Example T:**

****

Avoid infinite construction

**Goal**

We will introduce a new method based on **consequence propagation**

The basic dea is similar to **resolution** in propositional logic

Combine two clauses (GCIs) to derive a new one

If the two original clauses are true the third one combined by the other two is also true.

First we need some simplifications

Need to have conjunction of clauses

**Normal form**

A GCI is in **normal form** it is has one of the shapes:

A and B are concept names

A ⊑ B

A1 ∏ A2 ⊑ B

A ⊑ ∃ r.B

∃r.A ⊑ B

Where A, A1, A2, € Nc ∪ {T} and B € NC ∪ {T, ⊥}

A Tbox is in normal form if all its GCIs are in normal form

C ⊑ D ∏ E

Is equivalent to

C ⊑ D and C ⊑ E

**Example**

*Person ⊑ ∃hasParent.Person* YES

*Mammal ∏ Oviparous ⊑ Monotreme* YES

*Mother ⊑ Female ∏ ∃ hasChild.T* NO. Conjunction and ∃ on the right, we can transform it

*Monotreme* ⊑ *Mammal ∏ oviparous* NO it has a conjunction on the right

In essence **at most one construction** can appear in the GCI with the exception of ∏ on the right

**Relationship with rules**

GCIs in normal for are basically existential rules with constraints

A ⊑ B B(x) ← A(x) if an object x belong to A it also belong to B

A1 ∏ A2 ⊑ B B(x) ← A1(x), A2(x)

∃ r. A ⊑ B B(x) ← r(x,y), a(y)

A ⊑ ∃ r.B SrB(x,y) ← A(x) r(x,y) ← SrB(x,y) B(y) ← SrB(x,y) three rules at the same time

T in body and ⊥ in the head are removed

**Example (with simplification)**

∃ hasChild.T ⊑ parent parent(x) ← HasChild(x,y)

Parent ⊑ ∃ hasChild.T hasChild(x,y) ← parent(x) don’t care about any property of T

Mammal ∏ reptile ⊑ ⊥ ← Mammal(x), reptile(x). No mammal is a reptile in the same moment

28 oct 2021

GCI in normal form:

* at most one construct (no conjunction on the right)

**Normalisation is NOT a restriction**

Every TBox can be transformed to normal form exhaustively applying the normalisation rules (take complex GCI and make easier one)

Every finite set of GCI can be transformed to a TBox in normal form

C^ ⊑ D^ C^ ⊑ A A ⊑ D^ separe in two GCI by putting an intermediate new concept

(With a hat (^) are complex concepts, A new concept name, at least one constructor)

B ⊑ C ∏ D B ⊑ C B ⊑ D. Saying that something is a subset on the intersection of two set is the same as being part of both set at the same time

If we have conjunction on the left

B ∏ C^ ⊑ D introduce an abbreviation, new concept A B ∏ A ⊑ D C^ ⊑ A

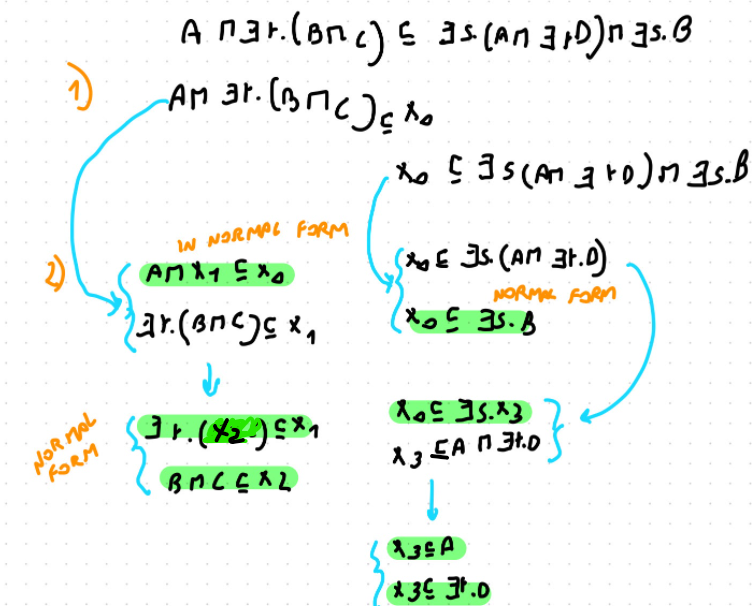
∃ r.C^ ⊑ D ∃ r.A ⊑ D C^ ⊑ A

D ⊑ Er.C^ D ⊑ ∃ r.A A ⊑ C^

Substitute something on the left, will stay still on the left in the substitution

**Example**

A ∏ ∃ r.(B ∏ C) ⊑ ∃.s(A ∏ ∃r.D ) ∏ ∃ s.B



End up with a set of GCI

Can always extend any model original of CGI in normal form

If I have an arbitrary TBox I can transform it in normal form

**Conservative extension**

T and TBox and *norm(T)* the TBox obtained from normalising T

Norm(T) is a conservative extension of T: (non equal because have new names, add something more)

If C and D use only symbols in T, then

T ⊢ C ⊑ D iff norm(T) ⊢ C ⊑ D

That is, normalisation does not add or remove subsumption relationship (\*)

**Proof of Conservative Extension**

1.

Every model of norm(T) is a model of T (by construction). If I have an interpretation that satisfies all the GCI it also satisfy the original GCI

T ⊢ C ⊑ D ⟹ Norm(T) ⊢ C ⊑ D

Every model of T satisfy that C is a subclass of D

2.

Every model of T can be extended to a model of norm(T) (interpreting A as C^)

Norm(T) ⊢ C ⊑ D ⟹ T ⊢ C ⊑ D

**Consequences**

What this all means is that to decide subsumption it suffices to consider TBoxes in normal form

We will explore how to decide these consequences

**Motivation**

T ⊢ C ⊑ D means that for ever model I of T, Ci ⊆ Di

Propagate the consequences of the TBox

Suppose that T ⊢ A ⊑ B1 T ⊢ A ⊑ B2 T ⊢ D ⊑ E

If B1 ⊑ C € T T ⊢ A ⊑ C (for transitivity with T ⊢ A ⊑ B1)

If B1 ∏ B2 ⊑ C € T T ⊢ A ⊑ C

If B1 ⊑ ∃ r.D € T T ⊢ A ⊑ ∃ r.E ( aslo A ⊑ ∃ r.D but every D element is subsumed by E, so ∃ r.E)

If T ⊢ A ∏ ∃ r.B

If ∃ r.B ⊑ C € T T ⊥ A ⊑ C (transitivity)

Propagate consequences that we have GCI that have in a TBox and propagate them

**Completion algorithm**

Propagated consequences

**Initialisation:** start with all the obvious tautologies

A € NC

T ⊢ A ⊑ A true in every model, every model is a subclass of himself

T ⊢ A ⊑ T also true, T is the entire domain

Provate knowledge

**Saturation:** apply the completion rule while possible

**Completion rule**

1. If T ⊢ A ⊑ B and B ⊑ C € T then conclude T ⊢ A ⊑ C
2. If T ⊢ A ⊑ B1, If T ⊢ A ⊑ B2 and B1 ∏ B2 ⊑ C € T then conclude T ⊢ A ⊑ C
3. If T ⊢ A ⊑ B, If T ⊢ C ⊑ D and B ⊑ ∃ r.C € T then conclude T ⊢ A ⊑ ∃ r.D.

1. If T ⊢ A ⊑ ∃ r.B and ∃ r.B ⊑ C € T then conclude T ⊢ A ⊑ C

**Example**

A ⊑ ∃ r.B . . B ⊑ ∃ s.C ∃ s.C ⊑ C

B ⊑ E C ∏ E ⊑ D ∃ r.D ⊑ D

T ⊢ :

A ⊑ T A ⊑ A .

B ⊑ T B ⊑ B .

C ⊑ T C ⊑ C

D ⊑ T D ⊑ D

E ⊑ T E ⊑ E

those are obius tautology

1. rule B ⊑ E . .

3. rule A ⊑ ∃ r.B .

3. rule A ⊑ ∃ r.E

4. Rule

B ⊑ ∃ s.C .

4. Rule B ⊑ C . .

A ⊑ ∃r.C

2. Rule B ⊑ D .

A ⊑ ∃ r.D .

4. rule A ⊑ D

Can’t do anything more

**Reminder**

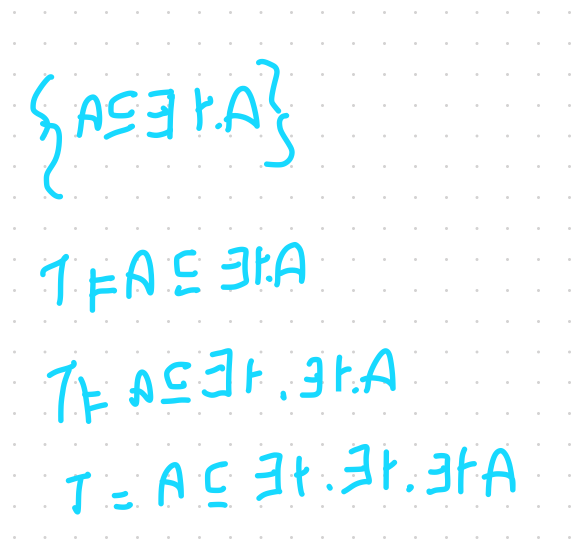
The goal of the method is to make consequences **explicit**

Make consequences of our KB explicit

based on other explicit consequences and GCIs

**Reasoning limitations**

By construction. The completion algorithm does not derive **all** consequences of T



can have infinite consequences, can’t be derived

But it derives all **atomic subsumptions**

A ⊑ B with A, B € NC ∪ {T, ⊥ } (\*)

The task of extracting all these consequences is called **classification**

Problem of deriding all these atomic subsumptions, knowing which are subclass of other

**\* One missing rule**

⊥ has very special properties

Consider the TBox T with GCIs:

A ⊑ ∃ r.B B ⊑ ∃ s.C C ⊑ ⊥

It entails that

T ⊢ A ⊑ ∃ r. ∃ .s ⊥ this is a complicated way to express ⊥ T ⊢ A ⊑ ⊥

A is unsatisfiable

However the completion algorithm does not derive this. We need a **new rule**

1. If T ⊢ A ⊑ ∃ r.⊥ then conclude T ⊢ A ⊑ ⊥

Propagate the ⊥ . There is no element that have an r successor that belong to the empty set

**Let’s recall**

Starting from a TBox T in *normal form*

The completion algorithm makes the normal form consequences of T explicit

**Claim**

A ⊑T B iff the algorithm derives A ⊑ B A ⊑ ⊥ or A ⊑ ∃ r.⊥

(Or we could extend the initialisation step)

A subsumption relation hold if we can derive it from the competition algorithm or if the concept on the left sight is unsatisfiable

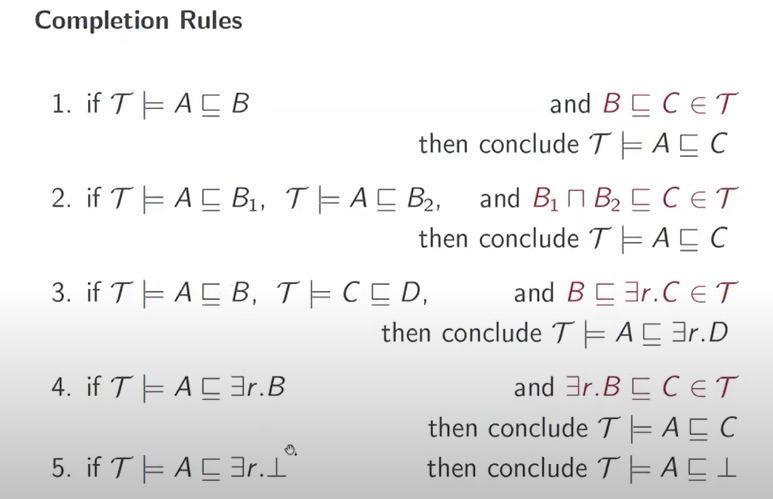
Every consequences is derived and if it derived is a consequences

**Soundness**

We need to show that every consequence derived is entailed by T

This is a consequence of the soundness of the competition rules:

1. Initialisation is trivially true (A ⊑ A, A ⊑ T are tautologies)
2. Rules do not introduce artefacts: if a model satisfies the preconditions of a rule, it also satisfies the added knowledge.



**Rule soundness**

Consider a model i

1. If i satisfies A ⊑ B then Ai ⊆ Bi

If B ⊑ C € T then Bi ⊆ Ci

thus Ai ⊆ Ci (for transitivity)

4. If i satisfies A ⊑ ∃ r.B

Then Ai ⊆ { δ | ∃ η € Bi (δ, η ) € ri}

if ∃ r.B ⊑ C € T

Then { δ | ∃ η € Bi (δ, η ) € ri} ⊆ Ci

Thus Ai ⊆ Ci

Check the remaking there rules

**Completeness**

We need to show that if a consequence is **not** derived then T does not entail it

If A then B is equal to no B then not A

We will build a **model** I that verifies this, find a model that does not satisfy it, so we know that is not true is any model

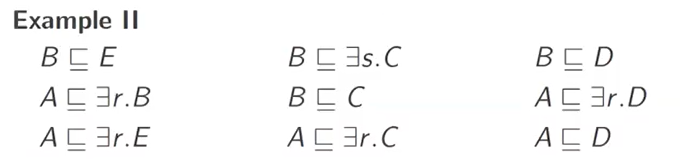
Specifically d A ⊑ B is **not** survived (Ai ⊈ Bi not subset)

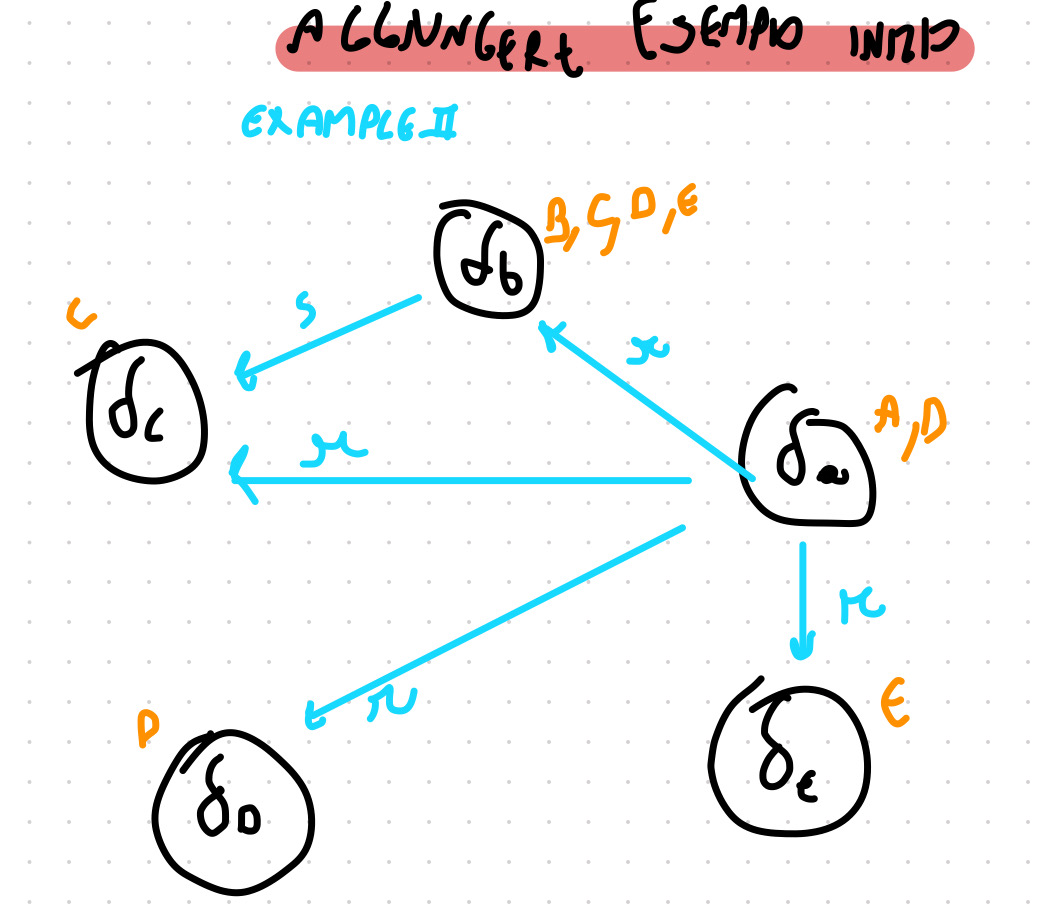
**Model construction**

Consider the set **PC** of all concept names A (in T) such that T ⊢ A ⊑ ⊥ is **not** derived by the algorithm

We build the interpretation I = (Δi, . i) where

* Δi = {δa | A € PC} (concept representatives)
* Ai = {δB | B ⊑A is derived}
* ri = {δa, δb) | A ⊑ ∃ r.B is derived}





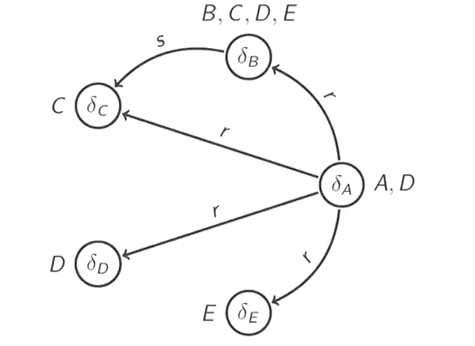
5 elements δa δb δc δd δe

Only δa belongs to A

Only δb belong to B

…

δa have r edges to δb δc δd δe



**Complexity**

The completion algorithm generates only consequences in **normal form**

GCIs in normal form have:

* at most there concepts names, or (A1 ∏ A2 ⊑ B)
* ⊥ most two concept names and a role name (A ⊑ ∃r.B, ∃r.A ⊑ B)

If there are n symbols in T, at most n3 consequences derived (worst case)

How many symbols are there in T?

**Cost of rule applications**

To apply a rule we should find:

* at most two consequences
* One GCI

At most n4 checks (n3 \* n) we can do much better

Overall, **polynomial time**

**Complex consequences**

The completion algorithm can only derive atomic subsumptions

What if we are interested in a more complex consequence?

T ⊢ C ∏ ∃r.B ⊑ D ∏ ∃ r.C

We can make some reductions

**Reduction to atomic subsumptions**

If C and D are complex concepts and X, Y are **new** concept names (not in T)

T ⊢ C ⊑ D iff T ∪ {X ⊑ C, D ⊑ Y} ⊢ X ⊑ Y

Intuitively we give a name to the complex concepts and check subsumption between these names

**ELI**

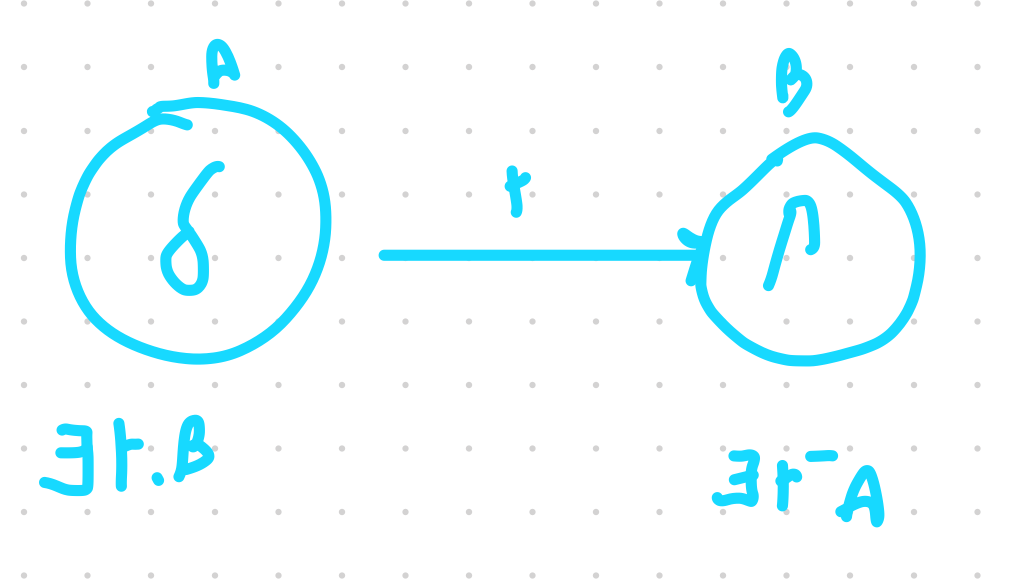
Consider the extension EL that allows for **inverse roles**

C::=A | T | ⊥ | C ∏ C | ∃ r.C. | ∃ r - .C

If ∃ r.C is *has a child with this properties*

∃ r - .C is *has a parent with this properties*

(R-)i = {(δ η )(η δ) € ri}



**From ELI to rules**

ElI GCIs in normal form can also be understood as predicate rule

* ∃ r - . A ⊑ B B(x) ← r(y,x), A(y)
* A ⊑ ∃ r - B SrmB(y,x) ← A(x) r(x,y) ← SrmB(y,x) B(x) ← SrmB(y,x)

Just reverse the arguments of r

**Similar but different**

ELI may look very similar to EL

But reasoning in ELI requires **exponential time**