**EL and Negations**

EL (as predicate rules) has no way to express **negations**

In EL⊥, we can partially achieve it:

A ⊓ B ⊑ ⊥ (the conjunction of A and B is a contradiction) means that no A-element can belong to B

in symbols: A ⊑ ¬B

If you are in A you are definitely not in B (and vice versa)

This **propagates** further:

if C ⊑ B and A ⊓ B ⊑ ⊥, then A ⊑ ¬C

However, this **does not** characterise negation

It tells us that something must be in the negation of a concept, but don’t tell what to do with that negation

C ⊑ B propagate to ¬ B ⊑ ¬ C

Something has to be in the negation but don’t tell the full negation

**Immortality as non-mortality**

Consider a knowledge base about living beings

*Mortal ⊓ Immortal ⊑ ⊥*

(no living being can be mortal and immortal) if somebody is mortal is not immortal (and vice versa)

The intersection has to be empty, but doesn't tell anything about the union. We can have things that are none of them

But there can still be objects that are

**neither** *Mortal* **nor** *Immortal*

How to express that

if a being is **not mortal**, then it must be immortal?

We want to express that if someone is not mortal it must be immortal

Define immortality as the negation of mortality (and vice versa) can’t do this in EL

**The Need for Negation**

What we want is to express. *¬Mortal ⊑ Immortal*

Everyone that is not mortal must be immortal, execute the possibility of having objects neither mortal or immortal

But this is not allowed in EL⊥

Let’s just include it in the language ALC

**A Minimal Definition of ALC**

ALC is EL extended with the negation (¬) operator

C ::= A | C ⊓ C | ∃r.C | **¬C**

(the first three elements is what we have for EL)

*A concept can be a concept name, conjunction of concepts, an existential restriction AND a new kind of constructor that allows to negate a concept*

With the obvious semantics (¬C )i = ∆i \ Ci

Semantics: given by interpretations, take an interpretation that define a domain and gives to every concept name a set of object and to every role name a binary relation (set of pairs)

Is the complement of C with respect to the domain

(¬C)i  contains all objects that **do not belong** to Ci

Set of all objects that do not belong to the interpretation of C

**Knowledge and reasoning**

All notions that we defined for EL⊥ transfer to ALC

• (ALC-)GCI: C ⊑ D with C,D (ALC-)concepts

Want to allow negations GCI refers to ALC

GCI: expression of C ⊑ D ad bot C and D are concept constructed in ALC-

• (ALC-)TBox: finite set of (ALC-)GCIs

• **satisfiability**: possibility of a non-empty interpretation (in a model)

A concept that has at least one element (that is not interpreted as the empty set)

Satisfiability wtc Tbox it need to be a model of the TBox

• consistency: existence of a model

• subsumption C ⊑ D: Ci ⊆ D*i* (in all models)

GCI: context in bio find a TBOx( set of constraints)

Subsumption: reasoning problem

TBox: two constraints, when I am doing reasoning check interpretations that satisfies those 2 constraints

P ⊑ ∃c.T (every parent have a child)

G ⊑ ∃ c.P (grand parent, person that have a child that is a parent)

Query

G ⊑ ∃ c. ∃ c.T? Yes

Model: interpretation that satisfies all the GCI

G→ P→ 0 this interpretation is a model

G→ P this interpretation but it is not a model because to be a parent it needs to have a child

**Some useful abbreviations**

• ⊥:=A⊓¬A (⊥i =Ai ∩ (∆i \ Ai) =∅) ⊥ is the contradiction

[∆i \ Ai  is the complement of A]

Intersection of a object and its opposite is always an empty set

Intersection of A and its complement. No object being to an object and its complement at the same time

• ⊤ := ¬⊥

Complement of the empty set

Top is the negation of bottom

• C⊔D:=¬(¬C⊓¬D) *(DeMorgan)* ((C⊔D)i =Ci ∪ Di )

C⊔D is disjunction

union of the two interpretation, belong to C or belongs to D (or both)

• ∀r.C := ¬(∃r.¬C) ( (∀r.C)i = {δ|∀η∈∆i.(δ,η)∈rI ⇒ η∈Ci} )

Universal restriction. There is an r successor that belongs to the concept C

∀r.C. All object that I can reach truth a r edges and if all of them belongs to the concept C

Only has daughters, is different to have daughter (can have one +one male)

If an object does not have any r successor than they object belongs to ∀ r.C

**Understanding value restriction**

*Person ⊑ ∀ hasChild.Person*

Defining property of people: if someone is a person, all the children of that person are a person

*∀ hasAccess.Stairs ⊑ Inaccessible*

If all the entrances have stairs that this place is inaccessible (for wheelchairs)

careful with lack of successors!

**Two set of constructors, one language**

From now on, we simply consider all constructors

¬, ⊓, ⊔, ∃, ∀

as part of the language ALC

Use smallest set of constructor so that you have to do less work

Useful for having a negation normal form (NNF)

**NNF**

An ALC concept is in negation normal form (NNF) iff negations apply to concept names only. Only to propositional variables (in this case is about concepts)

Every concept can be equivalently rewritten in NNF using

• De Morgan laws

¬(C ⊓ D) = ¬C ⊔ ¬D ¬(C ⊔ D) = ¬C ⊓ ¬D

Push the negation inside for having in Negation normal form

• duality of quantification

¬(∃r.C) = ∀r.¬C ¬(∀r.C) = ∃r.¬C

Use the definition of ∀ and ∃

• ¬¬ C = C

Remove double negation

not in NNF

in NNF

**Example**

¬(∃r.(¬A ⊓ B) ⊔ ∀s.C) it is not in NNF because of the negation at the beginning

Apply DeMogan

≡ ¬(∃r.(¬A ⊓ B)) ⊓ ¬(∀s.C)

¬ ∃D → ∀ ¬ D ¬ ∀D → ∃ ¬D

≡ ∀r.¬(¬A ⊓ B) ⊓ ∃s.¬C

DeMorgan

≡ ∀r.(A ⊔ ¬B) ⊓ ∃s.¬C

Now is in NNF

**NNF Assumption**

From now on, we assume concepts are **always** expressed in NNF

Implicit transformation whenever needed

**Roadmap**

We will describe algorithms for deciding:

• concept satisfiability (without TBox)

• TBox consistency

• knowledge base consistency (with **individuals**)

through a technique called **tableaux** a model-construction mechanism

Introduce a new kind of reasoning process (the idea behind is very simple)

Tableaux: process that is trying to construct a model by decomposing complex constraints in simple one

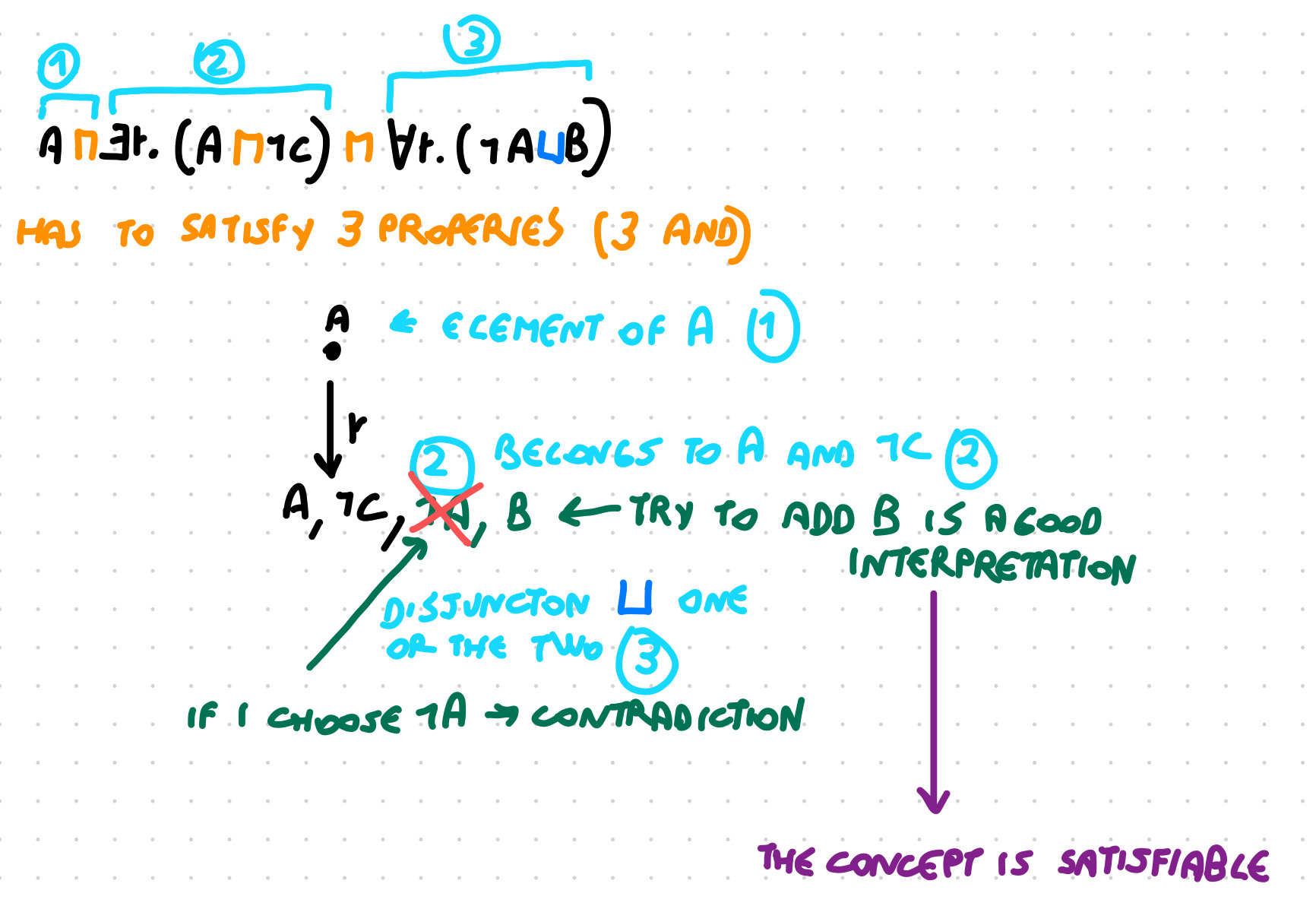
**Basic Idea**

Construct a model as one would try decomposing a concept into **smaller parts**

Find an interpretation that makes the concept not empty

**Example**:

A⊓∃r.(A⊓¬C)⊓∀r.(¬A⊔B)



**The Tableaux Algorithm I**

If we want to check whether the concept C is **satisfiable**

we want to build an object that belongs to C

Check if a concept C is satisfiable: try to do a constructing proof of satisfiability

A tableau is a model-constructing mechanism

that works over sets of assertions

Intuitively: interpretations

but not fully expressed (abstract)

**The Tableaux Algorithm II**

**Initialization**: assert C(a)

a is an arbitrary domain element. We created the object a

There is a object a that belong to the concept C

we start with the singleton set A = {C(a)}

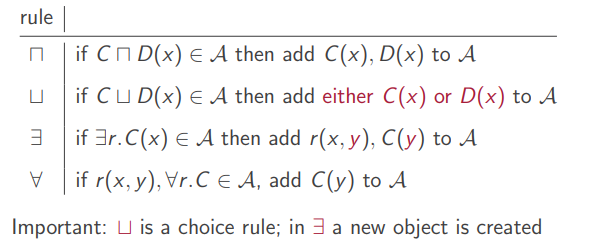
**Decomposition**: apply the tableaux rules (the decomposition rules, from complicate expression to simple expression)

Until we can find the interpretation or guarantee that we can’t find it

**Tableaux rules**

Remember that concepts are in **negation normal form**

A tableau rule takes an assertion from A and **adds** one or two assertions as follows:



∏ the object need to belongs to both concept

⊔ is a choice rule, we guess to add C or D

In ∃ a new object is created

**No negation rule**

Note that there is no rule for handling **negation**

Negation appears as the **last** item in the decomposition

It plays a role in deciding the existence of a satisfying interpretation

don't play a role in decomposition

**Saturation and closure**

A set of assertions A is

• **saturated** if no rule is applicable to it

• **closed** if {A(a), ¬A(a)} ⊆ A for some A ∈ NC

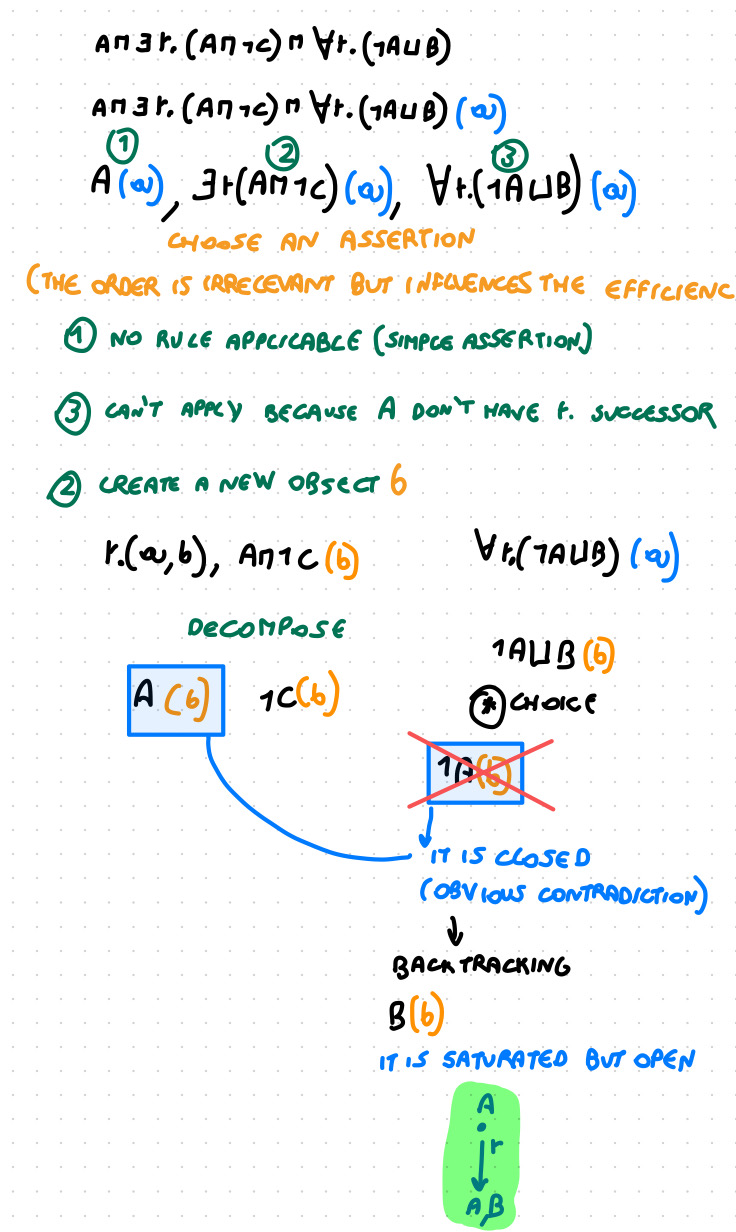
Is the obvious contradiction. Pair of assertion for the same object that belong to A and to the negation of A (needs to be the same object)

• **open** if it is not closed

If it doesn't have an obvious contradiction

**Example**

A⊓∃r.(A⊓¬C)⊓∀r.(¬A⊔B)



**Reasoning**

The concept C is **satisfiable** if the tableaux algorithm yields a **saturated and open** set of assertions

saturated: apply rules until we can’t apply anymore rules

Satisfiability: at least one way to get to the open set of assertion.

**Correctness**

The resulting set of assertions is a witness for satisfiability (a satisfying interpretation)

A saturated set of assertions has decomposed concepts

to atomic assertions

A(a) ¬A(a) r(a, b)

Decompose the complex concepts in atomic assertion, so that we have a saturated set of assertions

An open set of assertions contains no obvious contradiction

**Correctness II**

Given a saturated and open set of assertions A

construct the interpretation IA = (∆iA , ·iA ) where

• ∆iA = { a | a appears in A }

• AiA = { a | A(a) ∈A} A∈NC

• riA = { (a,b) | r(a,b) ∈A } r∈NR

First we see that IA satisfies all atomic assertions in A:

if C(a)∈A, then a∈CiA; if r(a,b)∈A, then(a,b)∈riA

Afterwards, we show that this property propagates to complex assertions

**Atomic Model**

Let A be saturated and open, and IA the interpretation constructed before

IA satisfies all atomic assertions in A:

• if A(a)∈A, by definition a∈AIA

• if r(a,b) ∈ A, by definition (a,b) ∈ rIA

• if ¬A(a)∈A, since A is open, A(a) ∈/A.

By definition, a ∉ AiA , and **a ∈ (¬A)iA**

**Model**

We show (by induction) that IA satisfies all assertions in A

• if C⊓D(a) ∈ A, since A is saturated, C(a) ,D(a)∈A

By induction, a∈CIA, a∈DIA. Thus, a∈(C⊓D)IA

If belongs to both interpretations it belongs to the interception

• if C ⊔ D(a) ∈ A,

since A is saturated, C(a) ∈ A, or D(a) ∈ A In the first case, a∈CIA. Thus,a∈(C⊔D)iA

• if ∃r.C(a) ∈ A, then r(a,b),C(b) ∈ A

By induction, b ∈ CIA, (a,b) ∈ rIA. Thus, a ∈ (∃r.C)iA

• if ∀r.C(a), r(a,b) ∈ A, then C(b) ∈ A

By induction, b ∈ CiA . As this holds for all r-successors, a ∈ (∀r.C)iA

**Summary**

Starting from {C(a)} we construct a saturated set of assertions A

• C(a) ∈ A (each rule adds new assertions)

• if A is open, then IA satisfies all assertions in A

In particular a ∈ CiA then C is satisfiable

If we reach a saturated and open set of a assertion C is satisfiable

**Converse**

If the tableaux (on C) produces a saturated and open set A, then C is satisfiable

We still need to show the **converse**:

if C is satisfiable, then there are choices such that the tableau yields a saturated, open set

We must make the correct choice in the ⊔ rule

**Model-Guided Construction**

If C is satisfiable, then there exists an interpretation I and δ∈∆I such that δ∈CI

This interpretation satisfies the set {C(a)} via a function f (a) = δ

We show that rule applications preserve this property

I⊨ A0 → I ⊨ A1 → ... →I ⊨ An

each Ai is open

**Model-Guided Construction II**

If I satisfies A and a rule application on A yields B

then I satisfies B

Case analysis on rule

⊓ if I satisfies C⊓D(a) then f(a) ∈ Ci∩Di.

Thus I satisfies C(a) and D(a)

⊔ if I satisfies C⊔D(a) then f(a) ∈ Ci∪Di.

Thus, f(a) ∈ Ci or f(a) ∈ Di. In the first case, **choose** to add C(a) to the set of assertions. I satisfies C(a)

∃ if I satisfies ∃r.C(a) then f (a) ∈ ∃r.Ci.

There exists η ∈ ∆i  s.t. (f(a),η) ∈ ri and η ∈ Ci. Define f(b) = η. Then I satisfies r(a,b) and C(b)

∀ if I satisfies ∀r.C(a) and r(a,b), then f(b) ∈ CI. Hence, I satisfies C(b)

**Termination**

Note that each rule application adds **simpler** concepts to A

A saturated set is found after **finitely many** rule applications (perhaps exponentially many, in the length of input concept)

Adding a T-Box

11 nov 2021

Decomposing until we get to the atomic sentences

Unsat: if it is equivalent to ⊥

**GCIs**

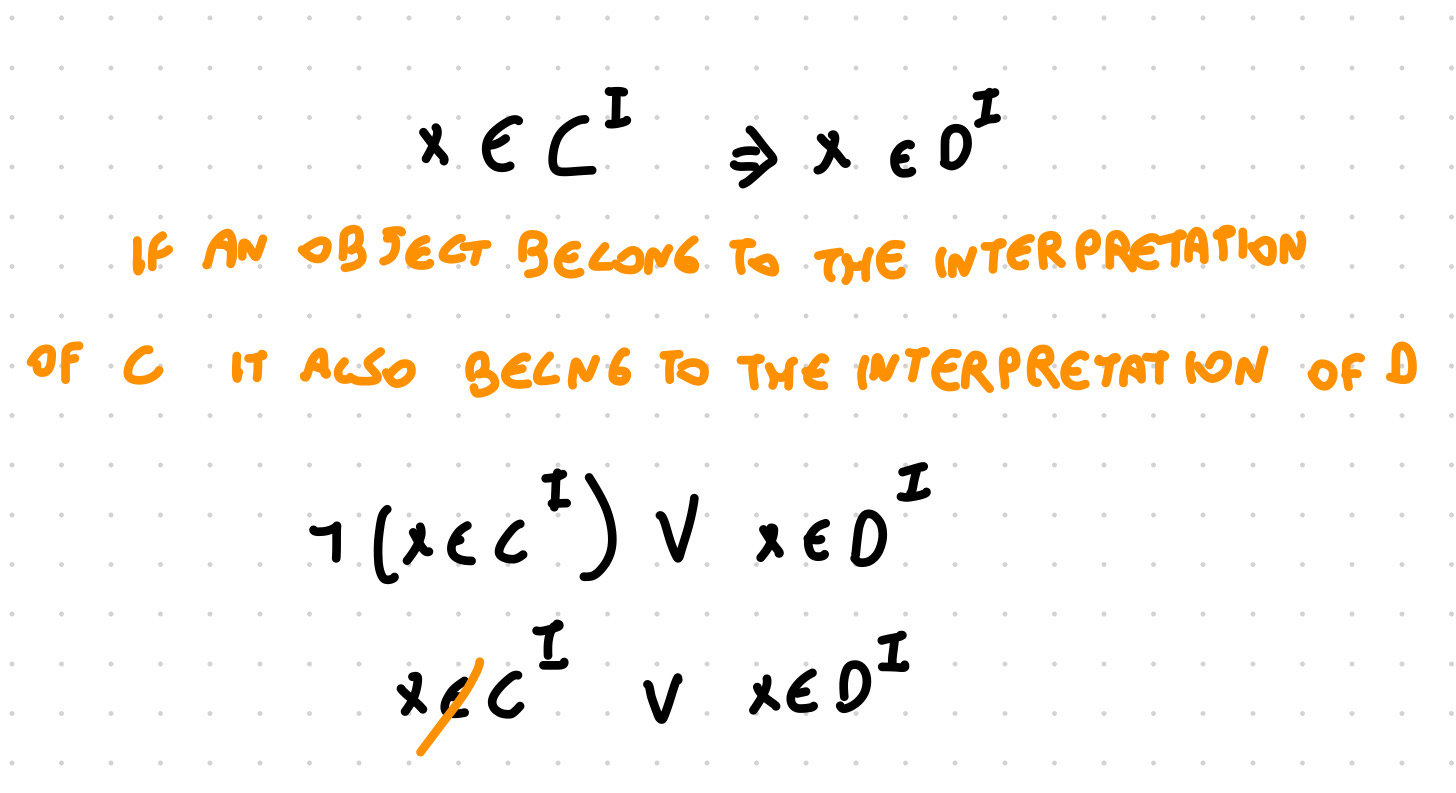
The tableau algorithm discussed before does not consider GCIs

Doesn't take any knowledge in account

**Recall**: the GCI C⊑D requires every model I to satisfy Ci ⊆ Di

In other words, every element of Ci must also be an element of Di (this is the definition of subset)

Alternatively, an element of ∆i is **not** in Ci **or** is in Di



**GCI Equivalence**

The GCI C ⊑ D is equivalent to T ⊑ ¬C ⊔ D

(In EL we couldn't do this, didn’t have negation)

Every elements of the domain must satisfy either ¬C or in D

C1 ⊑ D1 equivalent to T ⊑ ¬ C1 ⊔ D1

C2 ⊑ D2 equivalent to T ⊑ ¬ C2 ⊔ D2

A model needs to satisfy both conditions

T ⊑ ( ¬ C1 ⊔ D1) ∏ ( ¬ C2 ⊔ D2)

The second form suggests a requirement for model building

We want to build a model that satisfy all the constraint

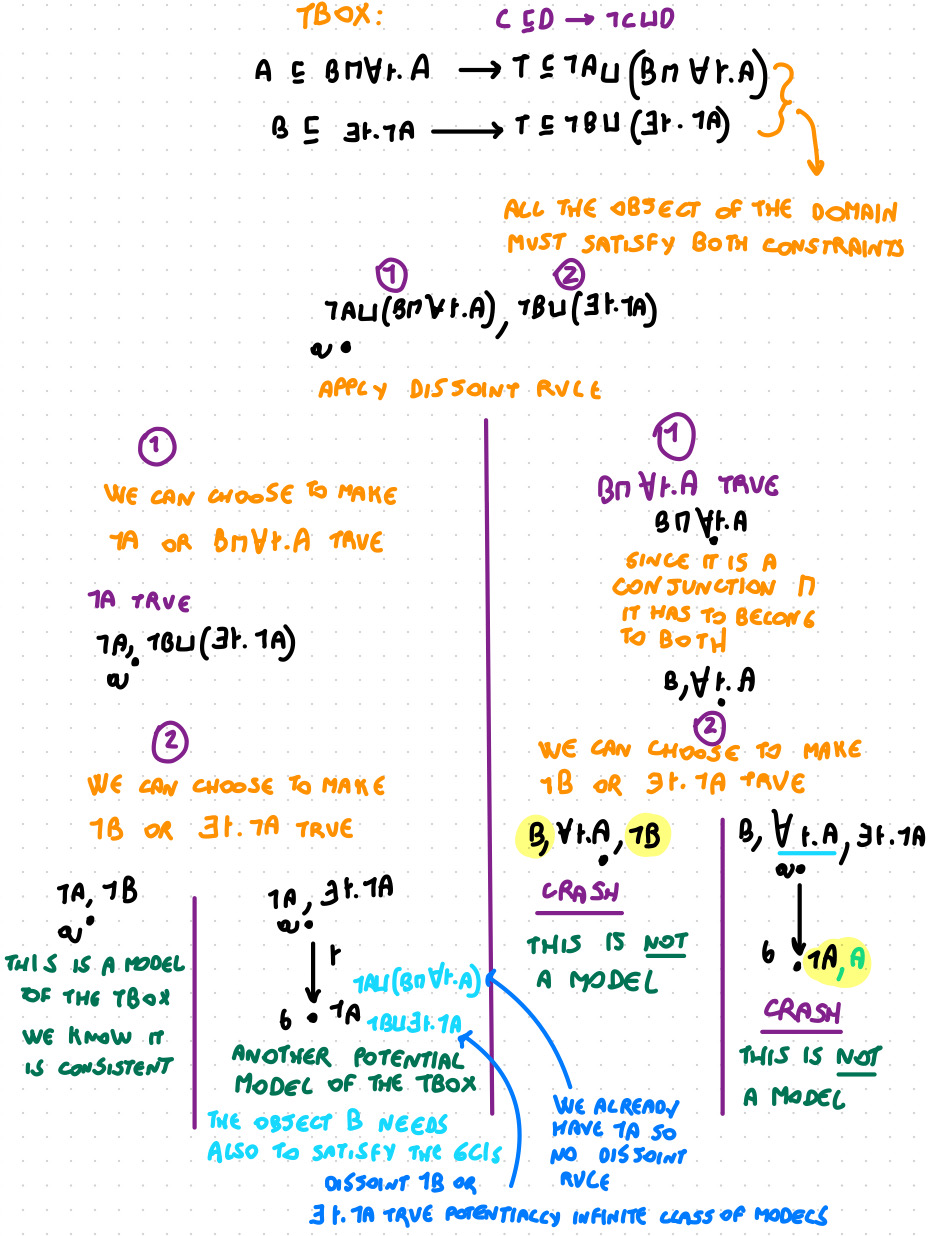
C ⊑ D: Every element that we generate must belong to ¬C⊔D

**Example**

Consider the TBox T with two GCIs:

A ⊑ B ∏ ∀ r.A

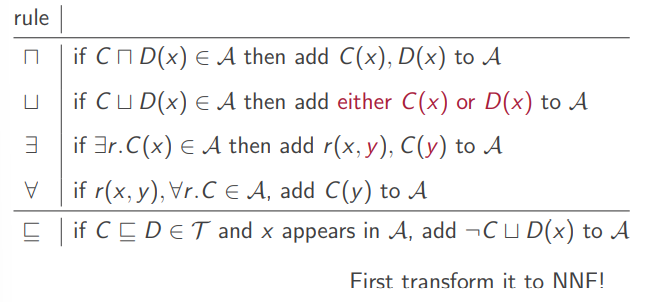
B ⊑ ∃ r. ¬ A



We have the exact same idea of the tableaux algorithm, decomposing the concepts into simpler parts. But every time we have an object we have to be sure that that object satisfy the GCIs

So add to the object the assertions of the GCIs, include them in any object

**A new rule**

****

The additional rule is the rule for GCIs

every time we include an new object we have to add the assertion that x must belong to ¬C⊔D(x) in NNF (negation normal form) and add it to the set of assertion we are dealing with

**TBox Consistency**

To check whether the TBox T is consistent:

Start with the set {T(a)} (there must be at least one element)

and apply the tableau rules to find a saturated, open set

If we find a saturated open set of assertion that the TBox is consistent, if we cannot find it the TBox is not consistent and we can’t build a model

**Is that all?**

Everything seems to work but there is a problem

The ⊑ rule does **not** add simpler concepts

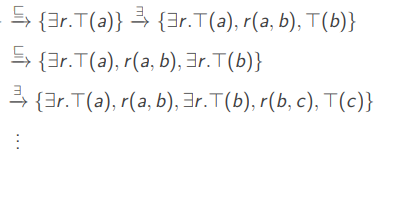
Does the process still terminate?

**Example**

Consider the TBox with only the GCI T ⊑ ∃r.T

no contradiction, is an EL TBox it is consistent

Start with simple assertion {T(a)}



this approach never terminates

We can see that this is an infinite sequence with a pattern, as human we have an idea of how it will continue

**Pattern Repetition**

In the previous example, we built an **infinite** sequence of nodes (elements)

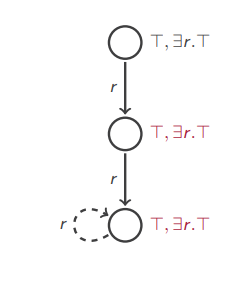
But they are all **the same**

After recognising the pattern, further enumeration is not informative (using like …)

The next still will not give me any additional information

We find a pattern and realise it is a cycle that the model construction is doing

**Cycle**

****

Obviously, cycles (patterns) may be longer

**Reasoning with Cycles**

The cycle is not intended to represent a cyclic model (although in ALC it happens to be one)

Sometimes those models are counter intuitive. If this a finite representation of an infinite model

Instead, it gives the instructions on how to construct one repeating a found pattern

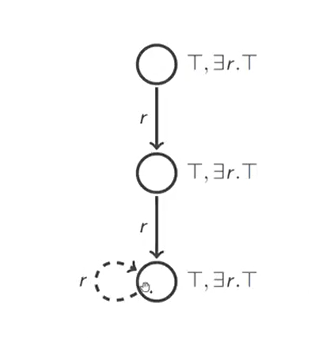
Find a pattern and expand the cycle to an infinite length

If the cyclic pattern is contradiction free then so would the full unfolded model be

Trying to decide whether a TBox is consistent

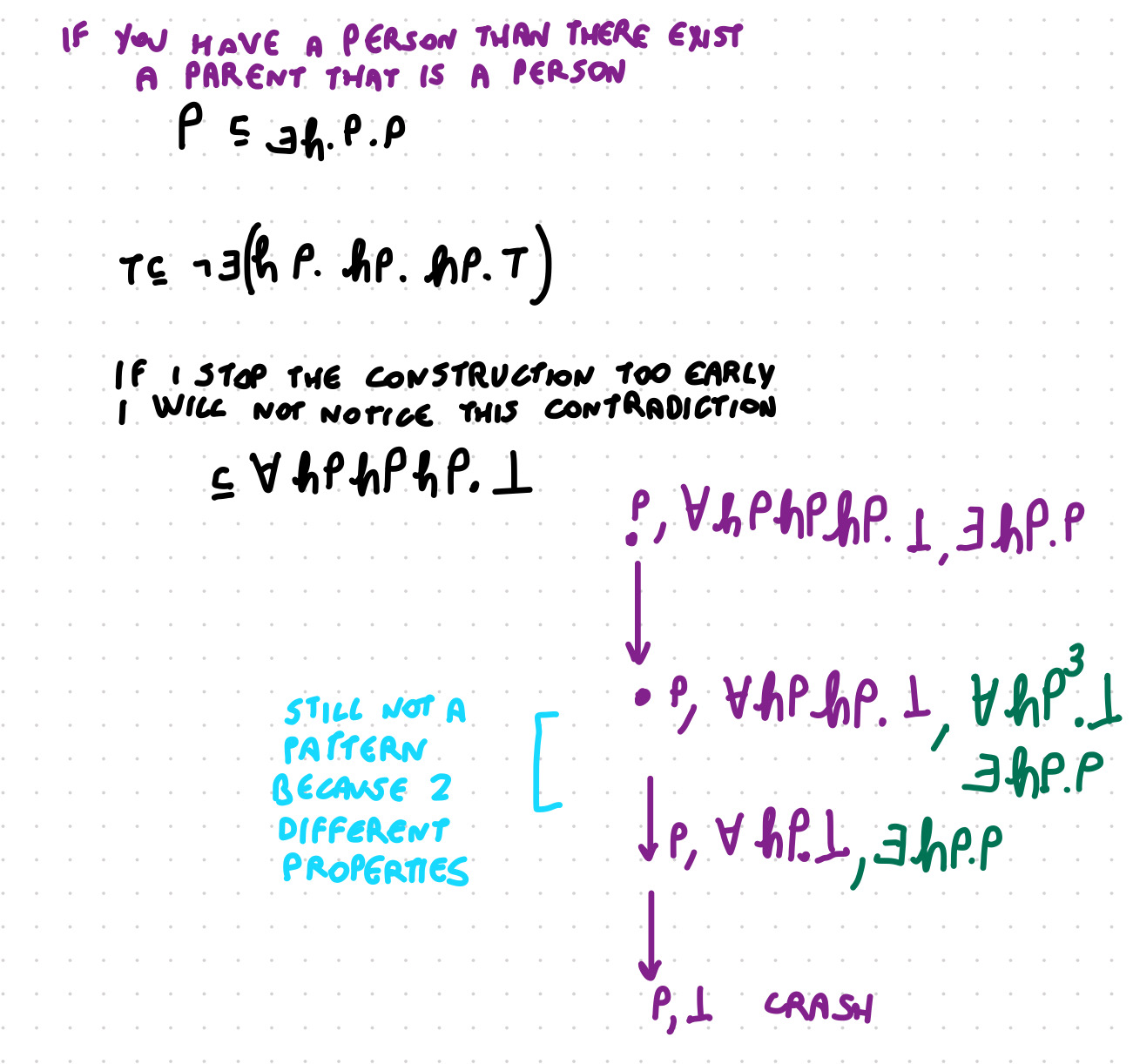
We know the existence of a model even if we do not fully construct it

The model exists even if we don’t fully construct it

****

If I construct a model: if up to here I didn’t find any contradiction (no crash) the set of assertion is open, even if I unrevil it since I will be copying the information again and again I will still have no contradiction

How to we decide when we have find a pattern

****

If we find a pattern and we saturate it and we don’t find a contradiction we can state that there will be a model even if we don’t fully construct the model

How do we decide when we have found a pattern? how do we decide to stop without losing informations

**Blocking**

We see the set of assertions as trees (in the obvious manner)

blocking to avoid infinite construction

Start with an object and whenever we have an existential rule we create a new successor of that object. If that have another existential restriction or value restriction that propagates it might have more successor

Every time we add a new object we add it with the existential rule

Every time we create a new object it will be connected to a previously object with an r edge

∃ r.C(a) → r(a,b) c(b)

Given a node a, and a set of assertions A define ConA(a) = {C | C(a) ∈ A} (the set of concepts that a satisfies in A)

The node a is blocked by the node b in A iff

• b is a non-root predecessor of a and (not the first object in the algorithm) (should not be the first object but something generated)

• ConA(a) ⊆ ConA(b) has to require less condition than the object B. Everything that a must satisfy is also satisfy by b

We go back in the tree and find another note that have all the constraints that we have seen (but it is not the root) than it is a blocking note

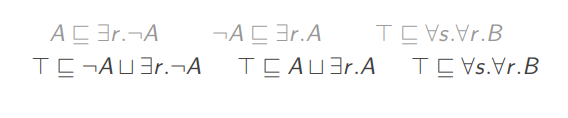
If a note is blocked I can’t generate new object

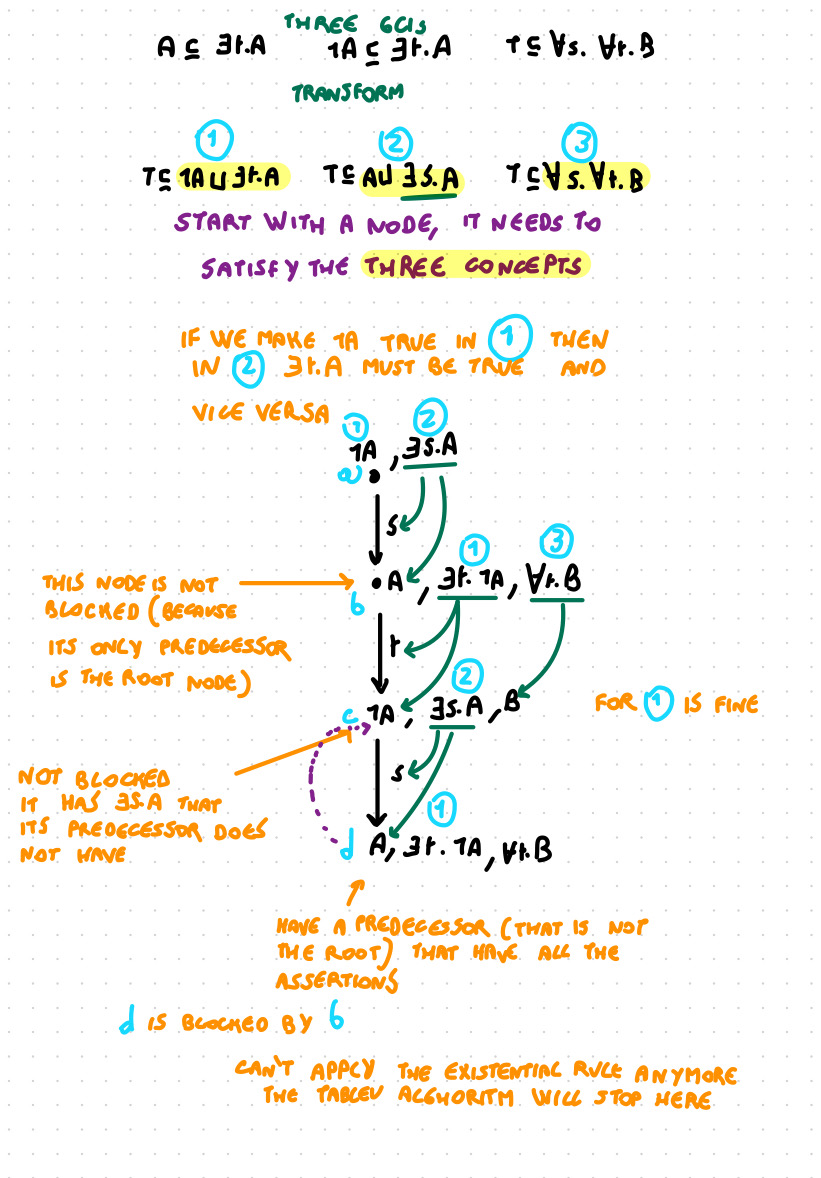
**Modified Rule Application**

The ∃ rule is applicable to ∃r.C(a) only if a is not blocked and none of its predecessors are blocked

Don’t want to expand anything that is below a block node (ro the block note itself)

**Example**

Transform 



**Reasoning**

The TBox T is consistent if the tableaux algorithm yields a saturated and open set of assertions

Saturation: can’t apply any rule, now depends on the blocking condition also

Similar argument to concept satisfiability (\*)

Proof of correctness: check that the rule application preserve models

A(a) € A than a € Ai

If we a saturated and open set of assertions we build the domain Δ iA set of objects in A

after we find a saturated an open set of assertion we build the domain ΔiA as the set of objects that appears in *A*

*A* = { a € Δ i | A(a) € A}

riA = {(a,b) | r(a,b) € A a is not blocked and has no blocked predecessor

Create cycle: add others pairs { (a,c) | a blocked by b, r(b,c) € A }

**Termination**

Are we sure that no infinite derivations are possible?

Note that for each node a, ConsA(a) contains only subconcepts appearing in T

All the concepts that appear here are subconcepts of the one appeariningin the TBox

If T has n subconcepts, there are at most 2n different sets ConsA

I can choose for each concept if it is there o no

but it is finite

The tableau has depth at most 2n + 2 and finite width (number of ∃r.C subconcepts)

the root is not counted +1

two will have exact same set of construct

can branch finitely, depends on how many existential constraints

**Complexity**

Overall, deciding TBox consistency is ExpTime-complete

Tableau not very good, we have a choice rule, or we have an exponential blow one, if we make t, require double exponential time

complexity is the worst case scenario, in practice the TBox that we encounter have a tableaux that goes fast

**Other Reasoning Problems**

• **satisfiability**: to check if C is satisfiable w.r.t. T , just start the tableau algorithm with {C(a)}

• **subsumption**: T ⊨ C ⊑ D iff C ∏ ¬D is unsatisfiable if there is an object that belong to C but no to D. if there is not object it is empty and unsat, so the subsumption holds. We couldn’t do it in EL

**ADDING INDIVIDUALS**

**From Terminological Knowledge . . .**

So far, we have focused on the terminological knowledge of a domain

That which defines the meaning of terms in the domain

This allows to express (and reason about) general notions and properties

But what about specific objects?

**. . . to Assertional Knowledge**

It is not sufficient to know the definition of a Parent

We want to make sure that if *hasChild(rafael, joe)* then *rafael* is a *Parent*

Want to deduce that Rafel is a parent

But then we need to add these “facts” to the knowledge base

These form the assertional knowledge of the domain

**ABoxes (assertion box)**

TBox is a terminological box

Consider an infinite set NI of individual names disjoint with NC and NR

An assertion is an expression of the form

• C(a) where a ∈ NI and C is an ALC concept (a in an individual name that belongs to C)

or

• r(a, b) where a, b ∈ NI and r ∈ NR

An **assertion box (ABox)** is a finite set of assertions (finite set of facts)

A knowledge base is a pair K = (T , A) where T is a TBox and A is an ABox

**ABox Semantics**

Need to give a meaning to the individuals names

An interpretation in ALC pairs that gives a domain

Interpretation functions maps concept names to sets and role names to binary relations

Individual names are now my map to an object of the domain

Interpretations need to give a meaning to individual names

The interpretation function ·i maps each a ∈ NI to an element ai ∈ ∆i

I satisfies the assertion

• C(a) iff ai ∈ Ci when the interpretation of A is an element of C *raphael is a parent, whatever object represent raphael must belong to the set of parents in that interpretation*

• r(a, b) iff (ai , bi ) ∈ ri

I is a model of the ABox A if it satisfies all assertions in A

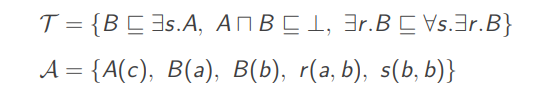
**Reasoning**

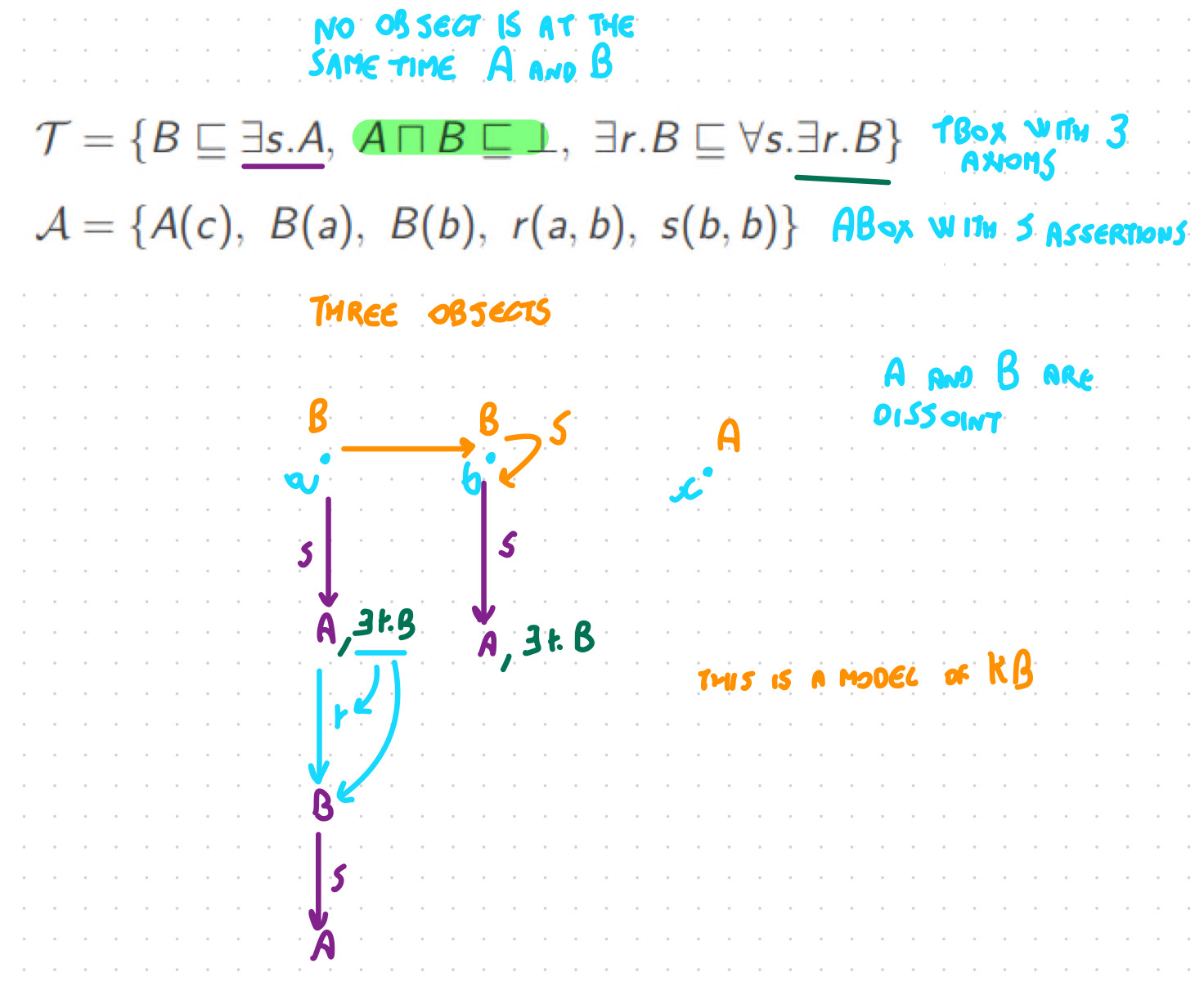
A KB is consistent iff it has a model. If I can find an interpretation that satisfy all the GCIs in the TBox and al the assertion in the ABox

We decide consistency by applying the tableau algorithm to a large initial set of assertions

We divide the problem into simpler cases

**Example**

****

****

**Root Pre-completion and Expansion**

Avoid to generate too many object

The tableau algorithm works in two steps Starting from the ABox as a given set of assertions:

1. pre-completion: apply all non-generating rules (not ∃) until saturation

extract all the informations about the object that are there without generating new objects

2. expansion: apply tableaux rules to each pre-completed individual. Do the all tableau algorithm

**Theorem**

The KB (T, A) is consistent iff the tableau algorithm produces a saturated and open ABox (do not forget blocking)

**Forest Models**

In ALC if a TBox is consistent than it has a tree shape model

The tableau algorithm, when successful, generates a forest-like model

Root: speaks about all the object that are at the begging

• an arbitrarily interconnected root system (ABox)

• with trees growing from each element



**Other Reasoning Problems**

• Instance checking

• Query answering

• Generalisation inferences

• Explanations and Debugging

• . . .

Instant checking

I ⊨ C(a) for all models I of (IA) (of the knowledge base)

In every model C(a) has be mapped to true

it is true if (I, A u { ¬ C(a) }) is inconsistent If it is consistent I have found a model of the KB. If it is inconsistent no model satisfy this