*Made by: Tito Nicola Drugman*

[**Compute the redux of a knowledge base**](#_a83pcd2rysg1) **3**

[**Decide if a clauses is a consequence of K**](#_fiswu96s2pir) **3**

[**Using predicate rules to create knowledge base**](#_9c4peliyzaua) **3**

[**Build the canonical model**](#_y2gt7bpn0v44) **4**

[**Subsumption**](#_nvq30dpxs8y2) **5**

[**Concept equivalence (EL)**](#_ljds327hz05p) **6**

[**Find an interpretation I such that Ci ⊈ Di**](#_85v022l34fji) **7**

[**Construct a model that has at least three elements**](#_jkz0bh2zatwd) **8**

[**Transform a TBox in normal form**](#_ynf198q12p7l) **8**

[**Completion algorithm**](#_6feqoi3wylh3) **9**

[Completion rules](#_jwq1h8w1pssm) 9

[**Inverse role**](#_wkzplw8ccata) **10**

[**NNF**](#_cnv0ixy7e2xu) **11**

[**Satisfiability**](#_dvx5nbc5a82q) **11**

[**Disjunction**](#_ny5chamuys1d) **12**

[**Construct concepts**](#_ev98gqn4r7uj) **12**

[**TBox Consistency**](#_36dmkfdt2ttw) **13**

[**Satisfiability**](#_be2mkdm2jx9w) **14**

[**Subsumption w.r.t a TBox**](#_m4ljpxfb85x6) **14**

[**Knowledge Base Consistency - ABox consistent with respect to the TBox**](#_sr8nxnj9hm4k) **15**

[What to check](#_f2yzdtqe1yvd) 16

[Consistency vs. Satisfiability](#_v9uxztlhe94y) 17

[**Build type graph**](#_t7096tjsgzrd) **17**

[How to name a type](#_onywiklhfdc7) 19

[How to recognise if a type is initial](#_j2svnh4z678g) 19

[How to recognise if a type is final](#_ovr72ppi51dd) 19

[Actually build the graph](#_l2ngnvxlu2x7) 20

[Draw the nodes](#_gx2csr98or5g) 20

[Draw the edges](#_6g1r17lm39zw) 20

[Initial or final?](#_xey7gurjqyol) 20

[Which temporal model satisfies the formula?](#_mrkbc0619kze) 21

[Length of a temporal model](#_uknk5l7xhxsi) 21

[First method (graphical approach)](#_6e53b9s8vvt1) 21

[Second method (mechanical approach)](#_t5laymtn14q0) 22

[**Construct an LTLF formula**](#_bwx0d2kizu39) **22**

[Extend the specification](#_1n8bzpb053l0) 23

[**Reasoning with probabilistic knowledge base**](#_t90nfsrjo7wo) **23**

[Compute the probability of the sequences](#_hoof8wlldhw0) 23

[Find a probabilistic distribution](#_ekaihhi3fhox) 24

[Extreme probabilities](#_k7zhz1orvnn) 25

[**Exercises from Mock**](#_mcfp1xxhxjg2) **25**

[Mock 1 - Boolean Functions](#_e8d3m8qqympa) 25

[Mock 2 - EL](#_8jay7q488ke7) 26

[Mock 3 - ALC](#_wv2kh8eggh0q) 27

[Mock 4 - Probabilistic Reasoning](#_tsv6exhz4z9t) 29

[Mock 5 - Temporal Logic](#_8kruwwiyu03d) 29

[**Exercise from Exam 220207**](#_4pemq994qsy1) **30**

[Exam 1 - Boolean Functions](#_phnqt3xary0a) 30

[Exam 2 - EL I](#_xsx9493taa50) 30

[Exam 3 - EL II](#_ody5ydz4a1gz) 30

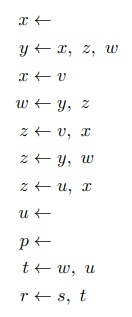
[Exam 4 - ALC](#_ybu1hjafog71) 32

[Exam 5 - Probabilistic Reasoning](#_x23wgi7nk0ol) 32

[Exam 6 - Temporal Logic](#_omuuzuxmxr76) 34

| **EL** | C | T | C ∏ D | ∃r.C |
| --- | --- |
| **EL⊥** | C | T | ⊥ | C ∏ D | ∃r.C |
| **ALC** | C | C ∏ D | ¬C | ∃r.C ( ∀r.C | C ⊔ D | T | ⊥ ) |
| Temporal |  |
| Probabilistic |  |

# Compute the redux of a knowledge base



A redux is a Knowledge Base without facts in the body ( x ← is a fact ), you will get a simpler Knowledge Base.

If x is a fact you can remove it from every body of all the clauses, keep doing so until no fact is left in the body of the knowledge base. The worst case scenario is when we must remove each variable from each rule. If K has *n* rules and *m* variables, we need maximum *n\*m* steps.

# Decide if a clauses is a consequence of K

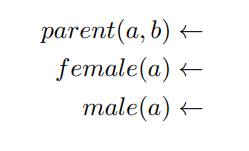
K is the Knowledge Base, to check if a clauses *a ← b* is a consequence of K you need to:

* make b a fact, so add *b ←*  to K
* check if doing so if *a* become true, then the clause *a ← b* is a consequence of K

Be careful that if the clause to check have the body equal to the head (so a ← a ) it is always be a consequence of K, because by adding *a ←* to K you already satisfied that a is true.

# Using predicate rules to create knowledge base

In the case you have the knowledge like



You can use predicate rules to create a knowledge base which describes family relations like *aunt*, *uncle*…

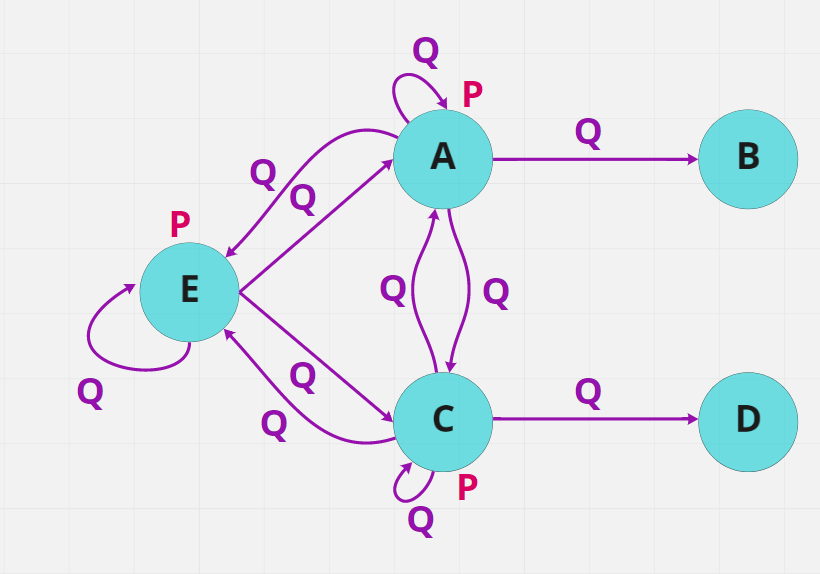
ex: *aunt(a,b) ← female(a), parent(x,b), parent(y,x), parent(y,a)*

# Build the canonical model

Canonical model: minimal representation of all possible models. A canonical model is a small model that has the minimal constraint, if we can conclude that something is true for the canonical model is true for all possible models.

How to build it

* Start with the empty graph
* Create all relevant nodes (constants), take all the constants in the knowledge base. Make the assumption that each constant refers to a different element of the domain.
  + The domain of the canonical model is the set of all constants that appear in the knowledge base
* Add all the facts
  + for each fact *P(a) ←* add the label P to the node *a*
  + for each fact *Q(a,b) ←* add a Q label edge from *a* to *b*
* Propagate rules
  + take a rule P(x) ← Q(x,y), … Q(y,x), P(z)
  + if all those predicates in the body are true, all the predicates in the head must be true, substitute the variables with constants and try to check if the body is true.
  + !!! remember that two different variables can represent the same object !!! when we substitute we can give the same value to multiple variables
  + We need to be sure that every consequence is found, show that our method is complete, everything that follows is derived.
  + Sometimes it can have no sense, everyone that have a parent can be sibling of themselves, to avoid so, you should add some constraints to remove any unexpected consequences
    - ← sibling(x,x) specifies that no one is a sibling of himself



*A canonical model*

# Subsumption

A ∏ C ∏ ∃r.B ⊑ C ∏ ∃r.T

You can use the homomorphism method to verify if a subsumption relations holds

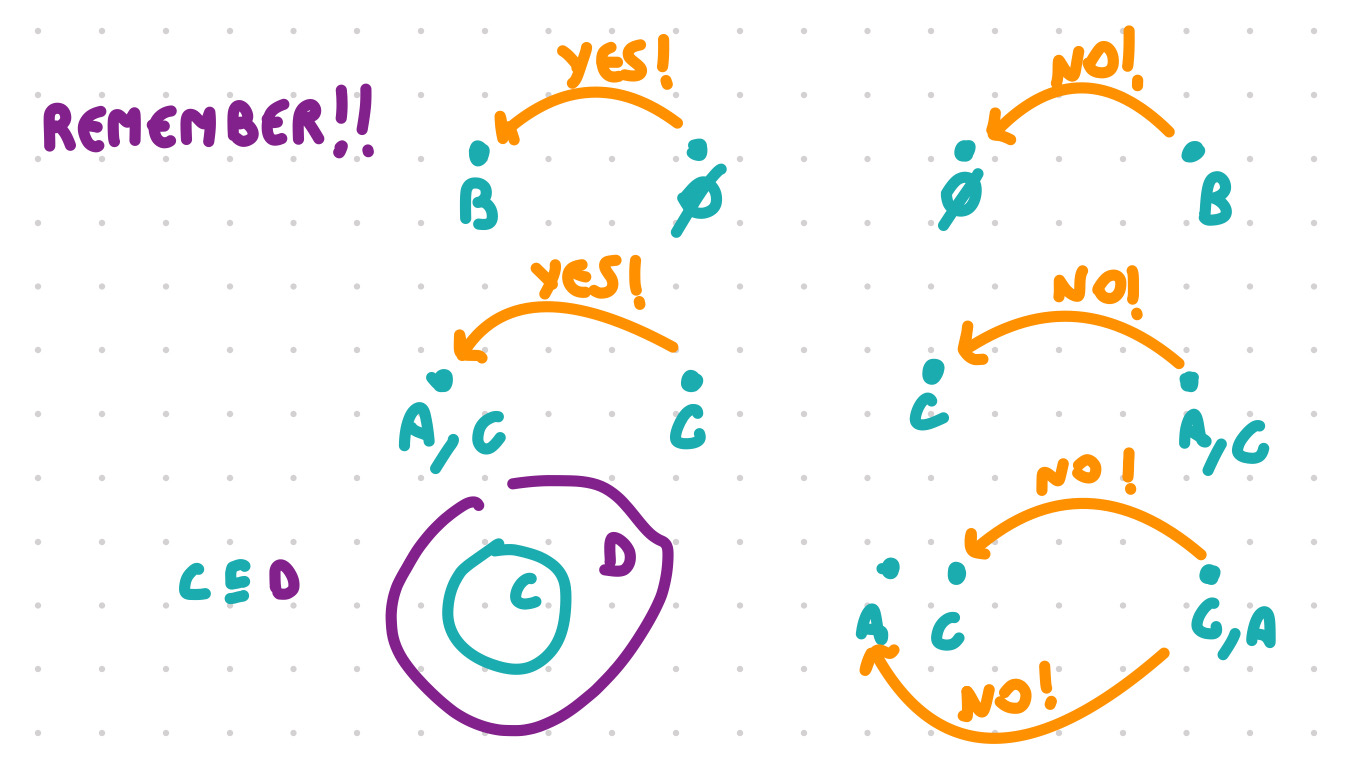
A subsumption relation is A ⊑ B means that A is a subset of B, so A have more constraints (because is smaller) than B.

A is subsumed by B iff for all interpretation *I* Ai ⊆ Bi. For any possible interpretation, make the interpretation of A a subclass of the interpretation of B.

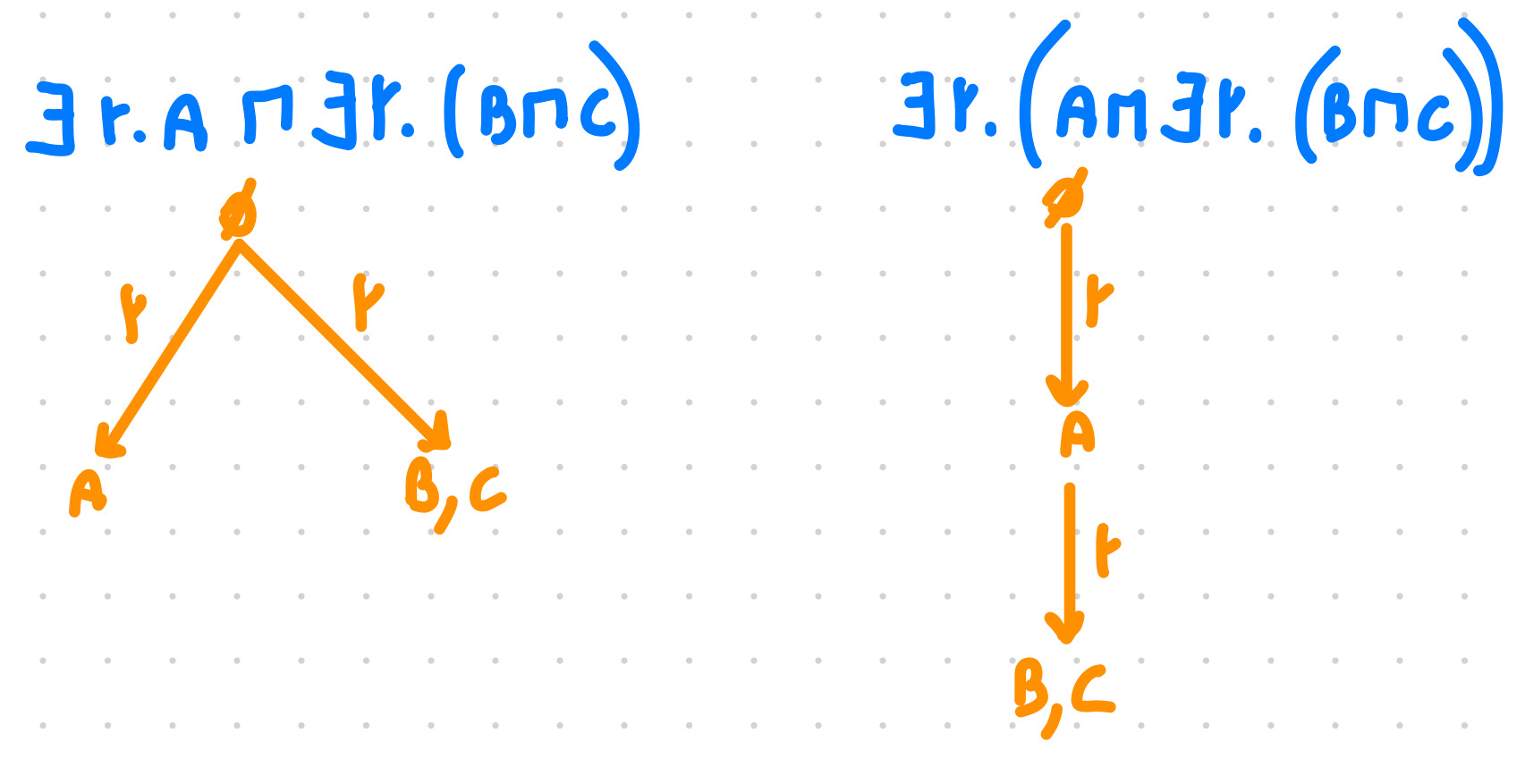
To check if A is subsumed by B we need to build a tree to represent the concepts

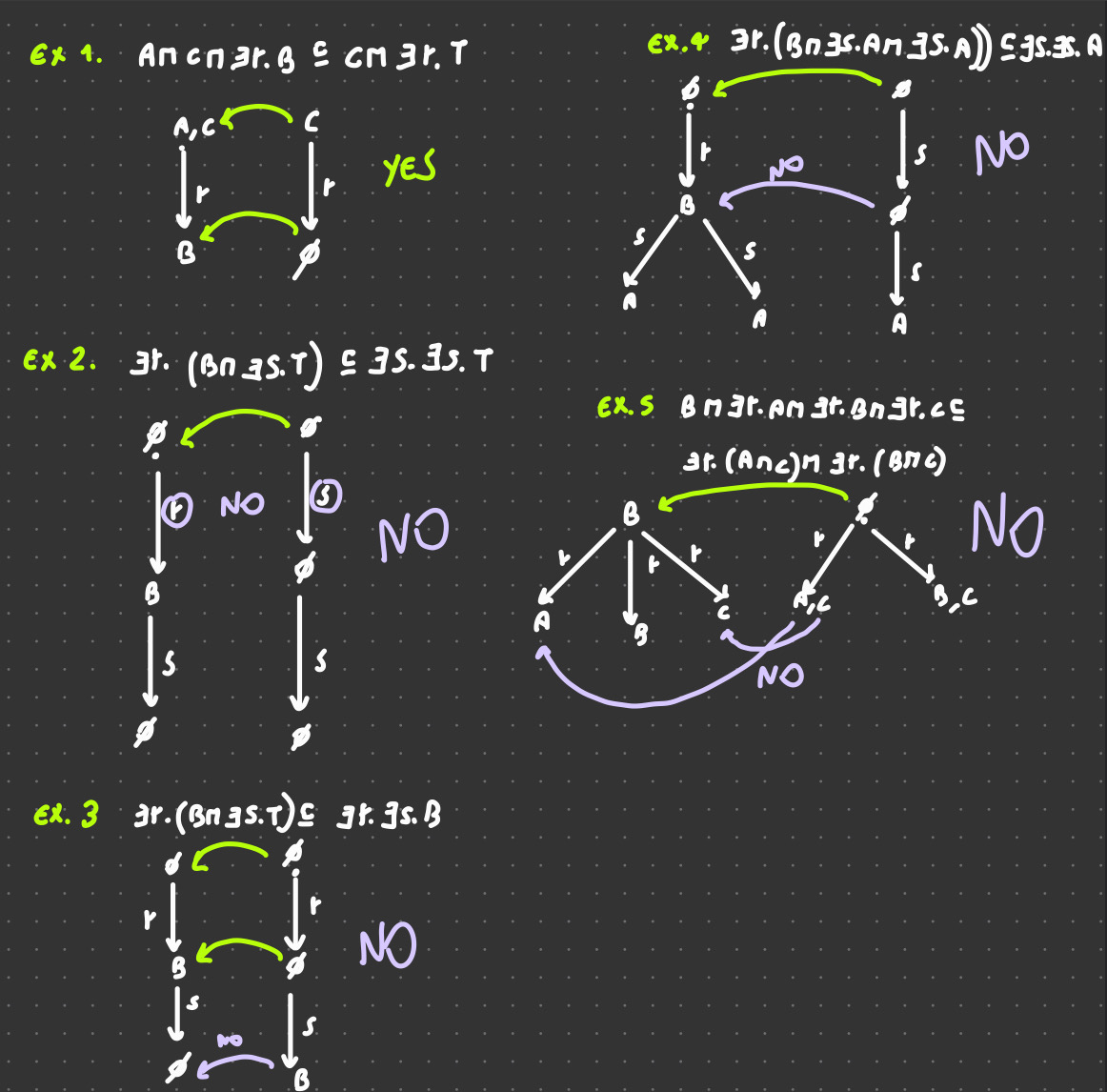
A homomorphism between two concept trees T1 and T2 is a function *h* mapping each node T1 to a node in T2

* maps the root to the root
* need to go from the right to the left (from the bigger B to the smaller A), check if the left side has more constraint of the right one, if there is the node is B it needs to be map to at least B
* check if the r-successor is right
* As soon as there is something not working you can stop because the subsumption relation does not hold
* T (top) is written as ø and also if there is no concept (like an r-relation with nothing at the beginning) keep using ø
* !!! going from ø to A is ok (because you are adding more constraints), but going from A to ø is wrong !!!



* When I have a conjunction of r-edges should I start from the beginning or keep going? Sometimes you might have ∃r.A ∏ ∃r.(B ∏ C) this is different from ∃r. ( A ∏ ∃r.(B ∏ C) ) because:





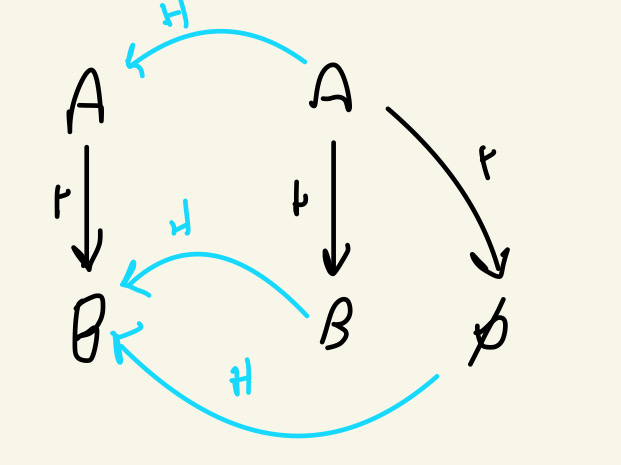
# Concept equivalence (EL)

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

∃ 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



Empty set is a subset of B

# Find an interpretation *I* such that Ci ⊈ Di

You start with two pairs of concepts C, D

C = ∃r.T D = ∃r.A

You need to find a situation in which C is not a subset of D

You need an interpretation that satisfy C but does not satisfy D, need an object that belongs to C (that satisfy C) but not D (don’t safiy D)

So if C = ∃r.T D = ∃r.A

It needs to have an r successor but it must not belong to A



I = (Δi , ・i )

Δi = { α, Ω }

Ai = { α }

Bi = { α, Ω }

ri = { (α, Ω) }

* it needs to belong to the concept on the left but not to the concept on the right
* start by representing the left concept, than add some constraints to be sure that the right concept is not satisfied
* to make it more clear, you can add some property to both α and Ω

If we have a concept C = *<something>* and D = ⊥ than we can just build C and we already found an interpretation Ci ⊈ Di

!!! remember: In EL⊥ any concept with ⊥ is just ⊥, so if D = ∃r.(A ∏ ∃s.⊥) that D = ⊥ !!!

# Construct a model that has at least three elements

∃r.(A ∏ B) ⊑ A ∏ ∃r.C

Means that if(!) I have ∃r.(A ∏ B) I must also have A ∏ ∃r.C, this is because the left one have more constraint (so it is smaller) than the one on the right, so the left one is a subset of the one on the right, having the left one require of having also the right one.

But if I have not the one on the left I have satisfied the model already, so I can create a model that does not have ∃r.(A ∏ B) and I will be completely fine.

# Transform a TBox in normal form

A terminological box (TBox) imposes some restrictions on the potential interpretation of concepts.

A TBox is in normal form if it has one of these shapes (A and B concept names):

A ⊑ B

A1 ∏ A2 ⊑ B

A ⊑ ∃ r.B

∃r.A ⊑ B

!!! No conjunction on the right, but one conjunction on the left is allowed !!!

In essence at most one constructioncan appear in the GCI, with the exception of ∏ on the right.

Take a TBox

∃r.(A ∏ B) ⊑ A ∏ ∃r.C

First of all you need to split it, that is

∃r.(A ∏ B) ⊑ x0 and x0 ⊑ A ∏ ∃r.C

!!! when you add a xn replacing something what you replace must be on the same side of the subsumption !!!

∃r.(A ∏ B) ⊑ x0  ⟹ ∃r.(x1) ⊑ x0  A ∏ B ⊑ x1

!!! with the ∏ on the right you can just split it in two pieces without creating additional xn !!!

x0 ⊑ A ∏ ∃r.C ⟹ x0 ⊑ A x0 ⊑ ∃r.C

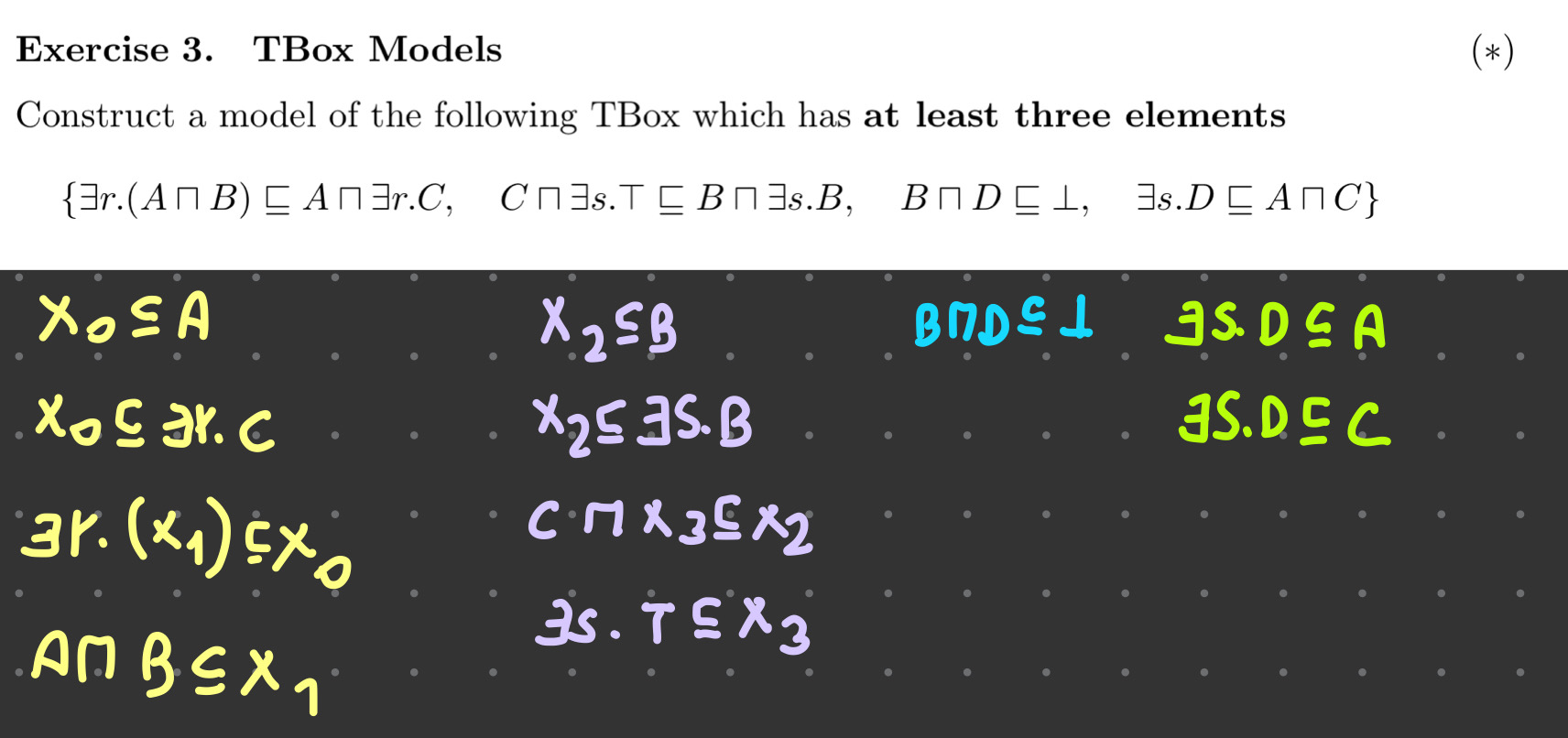
# Completion algorithm

Let *T* be the TBox



Apply the completion algorithm to check whether the consequence ∃r.∃s.D ⊑ ∃r.∃s.B holds

First of all, normalise the TBox



Then normalise the consequence

y0 ⊑ ∃r.∃s.D and ∃r.∃s.B ⊑ z0  (invert the ⊑ and use two different variables)

We would like to have y0 ⊑ z0 to check if a consequence holds

Normalise the consequence

y0 ⊑ ∃r.∃s.D becomes

y0 ⊑ ∃r.x4

x4 ⊑ ∃s.D

∃r.∃s.B ⊑ z0 becomes

∃r.x5 ⊑ z0

∃s.B ⊑ x5

So, I have the normalisation of the TBox and the normalisation of the consequence.

Now I have to apply the completion rules:

## Completion rules

| **IF I HAVE** | **THEN** |
| --- | --- |
| A ⊑ B and B ⊑ C | A ⊑ C |
| A ⊑ ∃r.B and B ⊑ C | A ⊑ ∃r.C |
| A ⊑ ∃r.B and ∃r.B ⊑ C | A ⊑ C |
| A ⊑ B and A ⊑ C and B ∏ C ⊑ D | A ⊑ D |

The completion algorithm starts with all the concepts names subsumed by themselves and subsumed by T

P ⊑ P

P ⊑ T

P € Nc P represents X0, X1, A, B, C…

* First derive all the simple axiom, than is a concept subsumed by a concept

x0 ⊑ A X2 ⊑ B

* Derive what is subsumed by x0 and x2

X0 ⊑ ∃r.C

X2 ⊑ ∃s.B

* Since X2 ⊑ ∃s.B and B ⊑ T we can derive

X2 ⊑ ∃s.T

* Since ∃s.T ⊑ X3

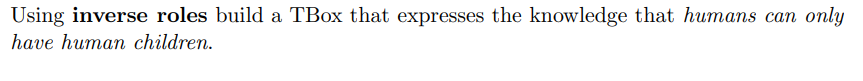
X2 ⊑ X3

This is the completion algorithm for the TBox, now add the consequence in normal form add all the formulas

I have to derive everything possible using the completion rules applied on the normalisation of the TBox and the normalisation of the consequence.

My hope is to arrived at y0 ⊑ z0

# Inverse role



Inverse role allows to traverse the role relationship backwards

Syntax: r -,r ∈ NR



Want to build a concept that says that being a child of a human, not the parent has a child that is human, something like:

ChilfOfHuman ⊑ Human

∃HasChild - .Human ⊑ Human

# NNF

Transform a ALC concept to Negation Normal Form

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).

Also NNF is a conjunction of disjunction, that is

( A ⊔ B ) ∏ ( B ⊔ ¬C ) ∏ D

!!! ¬∃r.A ⇔ ∀r.¬A !!!

# Satisfiability

Check if a concept is satisfiable or not



* First of all the concept must be in NNF

A ∏ ∃r.( ¬A ∏ ¬B) ∏ ∀r.B

* When there is a conjunction make a split and add the element *a* to the concept A

A(a) ∃r.(¬A ∏ ¬B)(a) ∀r.B(a)

* When there is an ∃ you need to create a new object

∃r.(¬A ∏ ¬B)(a) ⟹ r(a,b) ¬A(b), ¬B(b)

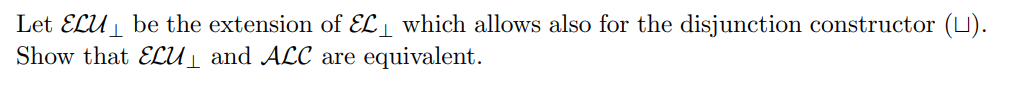
* Since there is a r-edge that goes from *a* to *b* and ∀r.B(a) means that the object *b* (that is at the end of the r-edge) must have the concept B
* There is a clash, between ¬B(b) and B(b) so the concept is not satisfiable

!!! A ∏ ∃r.⊥ ⇔ ⊥ a ⊥ with conjunctions is always ⊥ !!!

If there is an ⊔ make a choice and check the cases

∀r.⊥ is satisfiable as long you don’t have also ∃r.A .

# Disjunction



ALC C ::= C | C ∏ C | ∃r.C | ¬C

ELU⊥ C ::= C | C ∏ C | ∃r.C | ⊥ | T | C ⊔ C

To prove that they are equivalent everything that can be expressed with ELU⊥ must also be expressed with ALC and vice versa

* From ELU⊥ to ALC

⊥ = A ∏ ¬A

T = ¬⊥

C ⊔ C = ? yes! ¬ ( ¬C ∏ ¬D )

* From ALC to ELU⊥

¬C = ? Call it Xa

A ∏ Xa ⊑ ⊥

T ⊑ A ⊔ Xa every object must belong to A or Xa

# Construct concepts

Such that for any interpretation *I* holds that:

if DI ≠ ø ΔI must have at least 7 elements

Create a first element with concept A and a r-edge that goes to another element, that element should not collapse to the first one, so it must have ¬A

Then create a third concept, it must not collapse to A or ¬A, so add to the second one B (so it is ¬A, B) and to the third ¬A, ¬B

Remember that if an element has concept A and another has concept A,B they can collapse, you need to avoid this by adding as many negated or positive concepts, you can always add as many concepts as you like as your elements.

# TBox Consistency

Take a TBox

T1 = { A ⊑ ∃r.A, A ⊑ ∀r.¬A }

Transform the TBox

A ⊑ B ⟹ T ⊑ ¬A ⊔ B

The ⊑ is like an →

T ⊑ ¬A ⊔ ∃r.A

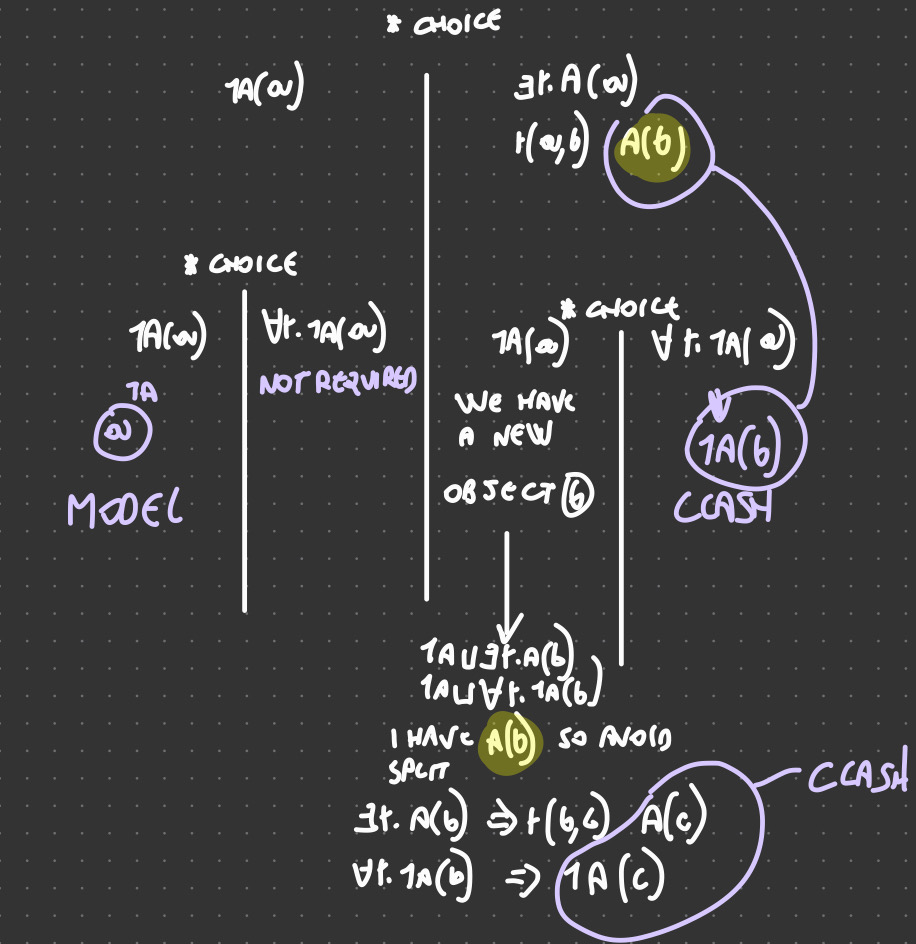
T ⊑ ¬A ⊔ ∀r.¬A

Since there is an ⊔ in the first GCI make a choice, split the paper, in the left the choice ¬A(a), on the right it is ∃r.A(a)

The left side is already satisfied, since ¬A is also in the second GCI, so check the right part. Every time there is an ∃r.A(a) you need to create an r edge r(a,b) that goes from *a* to *b* and give the concept A to b A(b)

Stop as soon there is a clash

if a new object is created you need to start from the beginning and make all the choice for that object too



# Satisfiability

Decide whether a concept is satisfiable with respect to a TBox

Given a TBox

T1 = { A ⊑ ∃r.A, A ⊑ ∀r.¬A }

Given a concept

∀r∃r.A

* Transform the TBox

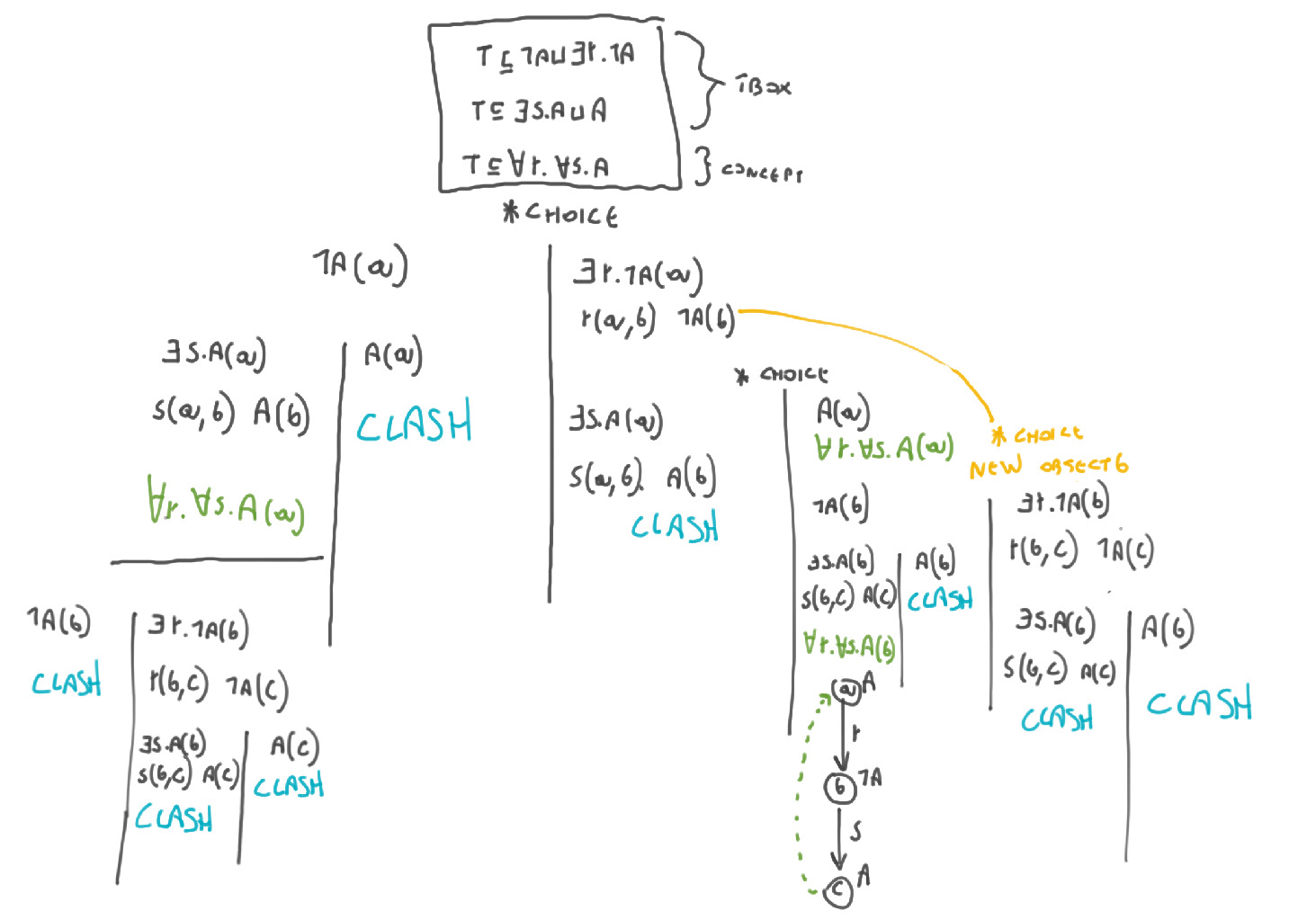
A ⊑ B ⟹ T ⊑ ¬A ⊔ B

The ⊑ is like an →

* Transform the concept

Just add T ⊑ in front

Follow the same strategy used in the TBox consistency, but if you find a possible model (even just one is fine) the concept is satisfiable with respect to the TBox. If all the model give a clash the concept is not satisfiable with respect to the TBox



# Subsumption w.r.t a TBox

Check if a subsumption hold with respect to a TBox

Given a TBox

T1 = { A ⊑ ∃r.A, A ⊑ ∀r.¬A }

Given a subsumption

B ⊔ C ⊑ ∀r.A

Check that does not exist any object that belong to the left side of the subsumption but not to the right, that is impossible to belong to ∃r.( A ∏ B ) and not to ∀r.B

To check subsumption of C ⊑ D we have to check that C ∏ ¬D is unsat

Negate the ⊑, that is a negation of the implies ( → )

If we get that all the models are unsat the subsumption holds, study all the models

If we get a model (or more) is sat the subsumption doesn’t hold, we can stop there

* Transform the TBox

A ⊑ B ⟹ T ⊑ ¬A ⊔ B

The ⊑ is like an →

* Transform the sumsumption

!!! it must be all negated !!!

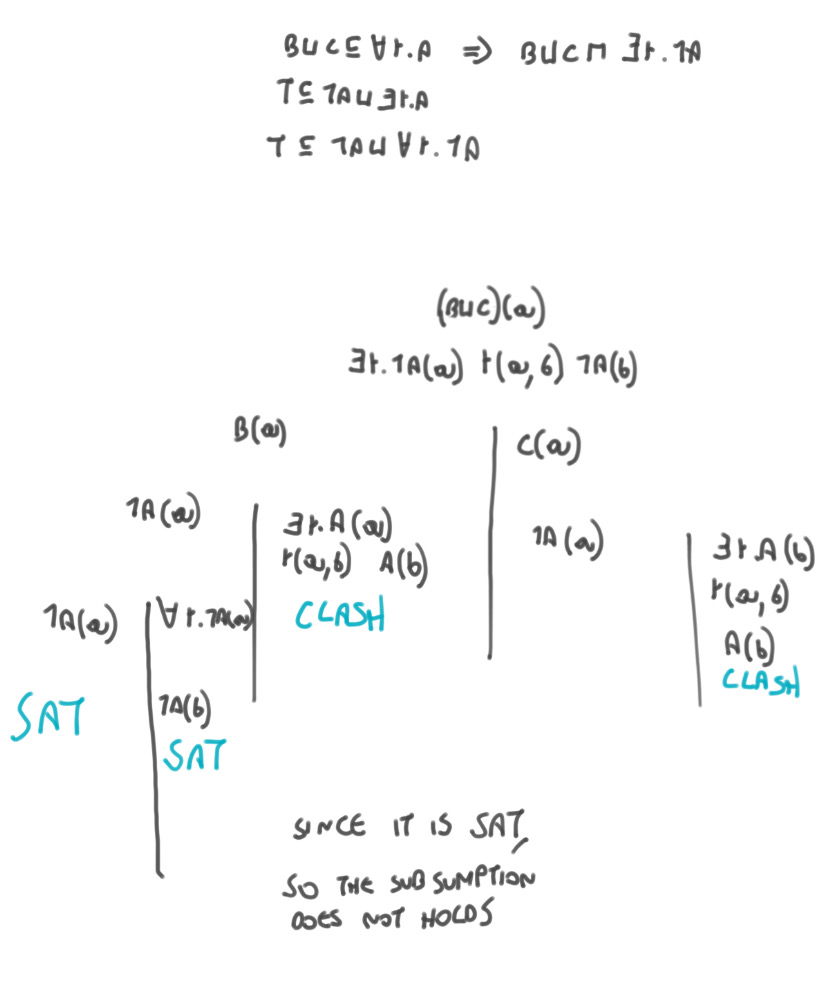
A ⊑ B ⟹ A ∏ ¬B

B ⊔ C ⊑ ∀r.A ⟹ B ⊔ C ∏ ∃r.¬A

Add the T ⊑ before

T ⊑ B ⊔ C ∏ ∃r.¬A

As soon as you find a SAT you can stop



# Knowledge Base Consistency - ABox consistent with respect to the TBox

We need a ABox and a TBox

Given a TBox

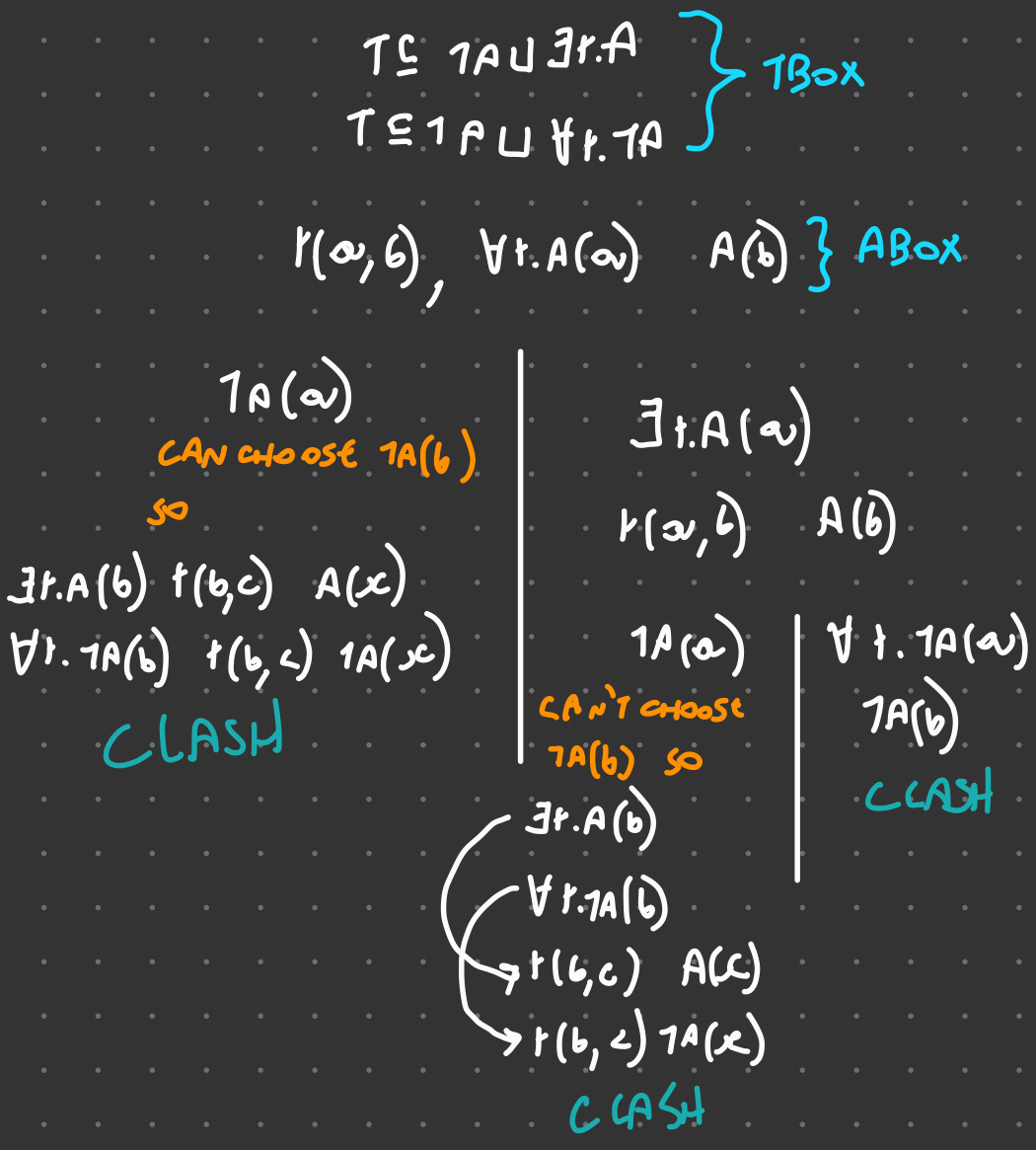
T1 = { A ⊑ ∃r.A, A ⊑ ∀r.¬A }

Given a ABox

{ r(a,b), ∀r.A(a) }

Start with the ABox and then do the same asTBox consistency

!!! It is a good idea to start with the ABox because ABox is certain and it must be in that way, so do the ABox before the split of the TBox in that way you save some time later !!



## What to check

| **Subsumption of a concept** | Do the homomorphism (EL) |  |  |
| --- | --- | --- | --- |
| **Satisfiability of a concept** | Put the concept in Negation Normal Form (negation only in front of literals) | Apply all the rule in the correct way and see if there is all clash (not satisfiable) or if there is a saturated open set of assertion (SAT) | !!! If you create a new object you have to repeat the same method and split cases as before |
| **Tbox consistency** | Rewrite the Tbox in the form T⊑A⊔B (to do so remember that ⊑ is like →) | Apply all the rule in the correct way and see if there is all clash (not consistent) or if there is a saturated open set of assertion (consistent) | !!! If you create a new object you have to repeat the same method and split cases as before |
| **Satisfiability with respect to a Tbox** | Rewrite the concept in Normal Form.  Rewrite the Tbox in the form T⊑A⊔B (to do so remember that ⊑ is like →) | Apply all the rule in the correct way and see if there is all clash (not satisfiable) or if there is a saturated open set of assertion (SAT) | !!! If you create a new object you have to repeat the same method and split cases as before |
| **Subsumption relations hold with respect to a Tbox** | Rewrite the subsumption relation in the correct way (⊑ = →) and negate it.  Rewrite the Tbox in the form T⊑A⊔B (to do so remember that ⊑ is like →) | If we find a clash it’s UNSAT and so the subsumption holds. As soon as we find a SAT we can stop and says that the sumbumption does not hold | !!! If you create a new object you have to repeat the same method and split cases as before |
| **Abox consistency with respect to aTBox** | Apply the classical rule | Apply all the rule in the correct way and see if there is all clash (not satisfiable) or if there is a saturated open set of assertion (SAT) |  |

## 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

# Build type graph

Let φ = x ų ¬y

A type is a maximally consistent subset of csub

Split φ in all the subformulas (and write also the initial formula as a subset)

* x ų ¬y
* x
* ¬y
* y

Add the negation

* ¬ (x ų ¬ y )
* ¬x

(don’t add y and ¬y because they are already there)

!!! the until formula requires two other formulas ⚪ <until formula> and ¬⚪<until formula>

* ⚪(x ų ¬ y )
* ¬⚪(x ų ¬ y )

Now because there are 8 formulas (4 positive and 4 negated) it is easier to consider only the positive one and make them true (1) or false (0). To be maximally consistent we could not have y and ¬y at the same time, choose 1 or in positive form on in the negated form

It is easier to start from the smaller formula (x and y) and then check the positive formulas

| ⚪(x ų ¬ y ) | x ų ¬y | x | y |
| --- | --- | --- | --- |
|  |  | 0 | 0 |
|  |  | 0 | 1 |
|  |  | 1 | 0 |
|  |  | 1 | 1 |

If y is false

* The until formula ( x ų ¬y ) is already satisfied, so it is 1
* It does not matter if x is true or not, in both cases the until formula is already satisfied
* ⚪(x ų ¬ y ) can be true of false, it is possible to have it or not

| ⚪(x ų ¬ y ) | x ų ¬y | x | y |
| --- | --- | --- | --- |
| 0 / 1 | 1 | 0 | 0 |
|  |  | 0 | 1 |
| 0 / 1 | 1 | 1 | 0 |
|  |  | 1 | 1 |

If y is true

* and x is not true
  + the until formula is definitely false. x has to be true until I find ¬y, but since x is not satisfied the only way would be to have ¬y true but it is false
  + even if the until formula is not satisfied now it could be satisfied later so ⚪(x ų ¬ y ) can be either true or false
* and x is true
  + the until formula is satisfied
  + The problem is that you don’t know if x ų ¬y it satisfied or not, because it could be (or not) satisfied in the future, so it can or can not holds, so there are 2 possibilities
    - The until formula is satisfied only if in the next point in time it is satisfied
    - The until formula is not satisfied only if in the next point in time it is not satisfied

| ⚪(x ų ¬ y ) | x ų ¬y | x | y |
| --- | --- | --- | --- |
| 0 / 1 | 1 | 0 | 0 |
| 0 / 1 | 0 | 0 | 1 |
| 0 / 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

No more choices, there are 8 types ( 2 for the first, 2 for the second, 2 for the third and 1 for fourth and fifth )

## How to name a type

T0100

T1100

T0001

T1001

T0110

T1110

T0011

T1111

## How to recognise if a type is initial

A type is initial if it contains the formula (it has to be true) that we are interested in.

In this example a type is initial if the second number is 1 ( so if φ = x ų ¬y is true ), there are 5 initial types

## How to recognise if a type is final

A type is finial if all the next formula are negated, it can contain ¬⚪

!!! ( (⚪x) ų ¬ y ) is not a next formula but an until formula, a next formula would be ⚪x or ⚪(x ų ¬ y ) !!!

In this example the only formula with a ⚪ is the first one, so if the first number of the type is 0, the type is final. There are 4 final types.

!!! a formula can be initial / final, both or none !!!

## Actually build the graph

### Draw the nodes

There are 8 types, so there are 8 nodes

### Draw the edges

We have to check how they are connected

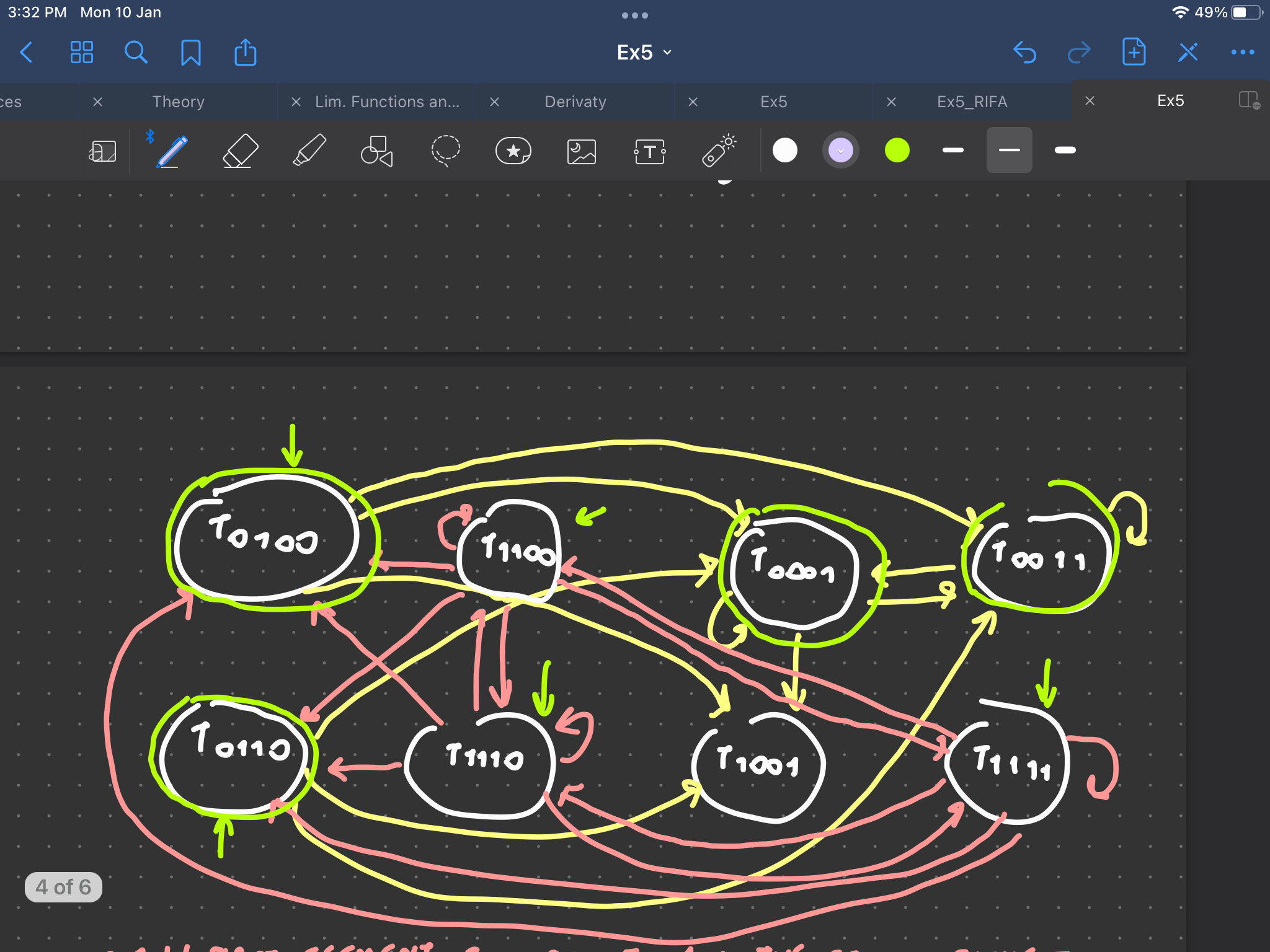
* Start by checking the next formula ⚪(x ų ¬ y ), if is 0 (so the first number of the type is 0) it means that in the next point the second number of the type ( x ų ¬y ) must be 0, otherwise it would be inconsistent
  + connect all the types that have a 0 as first number to all the types that have a 0 in the second number
* Again check the formula ⚪(x ų ¬ y ), if the first number of the type is 1, in the next point in time the formula x ų ¬ y (that is the second one) must be true
  + connect all the types that have a 1 in the first number to all the types that have a 1 in the second number

!!! some edges may be connected to themself !!!

### Initial or final?

After drawing all the edges, we have to specify which type is initial and which is final

* An initial type has the interested formula true, it is marked with an arrow coming from nowhere and going to the initial type
* A final type has all the next formula negated, it is marked with a double circle



## Which temporal model satisfies the formula?

The class of all temporal model that satisfies are all the paths in the graph that goes from an initial state to a final state

!!! can do a loop as long as you want and then move to a final type, not always is a straight path !!!

!!! can be also one type that is initial and final at the same time !!!

## Length of a temporal model

Temporal model: sequence of propositional valuation

V0, V1, V2… Vn

The length of a temporal model is the number of valuation n+1

length (V0, V1, V2… Vn ) = n+1

A temporal model of length 2 need 3 evaluations

There are two different methods to find how many temporal models of length 2 satisfy the formula (x ų ¬y )

### First method (graphical approach)

* check in the type graph how many models that starts with an initial type and ends with a final type that have length 2 (that looks at 3 types)
* It is very difficult to keep track

### Second method (mechanical approach)

* try to analyse the formula and check the possibilities
* a valuation is a function that takes x and y (the only two variables in this case) and maps them to 0 or 1. At each point in time we can choose 4 possible valuations (00, 01, 10, 11). Not all the valuations satisfy the formula. We have 4x4x4 possibilities that have length 2, but not all of them satisfy the formula
* To satisfy ( x ų ¬y ) at some point I have to find ¬y it can be in V0 or V1 or V2 (can’t be found later)
* In the case ¬y is found in the valuation V0 the formula is satisfied regardless of the value of x. I don't care what happens later
* in the case ¬y is found in the vautation V1 we are forced to have y in V0 (otherwise V1 will not be the first time of ¬y), but since I have to satisfy ( x ų ¬y ) we must have x in V0. In V1 we can have either x or ¬x and in V2 all the valuations are fine
* in the case ¬y is found in the valuation V2 we can have in V2 x or ¬x. In V0 and V1 I must have y (otherwise V2 will not be the first time of ¬y). Since this x must be true in V0 and V1 otherwise the until formula would not be satisfied

| V0 | V1 | V2 | Total Temporal Models |
| --- | --- | --- | --- |
| ¬y and (x xor ¬x) | (x xor ¬x) and ( y xor ¬y). 4 possibilities | (x xor ¬x) and ( y xor ¬y). 4 possibilities | 2 x 4 x 4 = 32 |
| x and y | ¬y and (x xor ¬x) | (x xor ¬x) and ( y xor ¬y). 4 possibilities | 1 x 2 x 4 = 8 |
| y and x | y and x | ¬y and (x xor ¬x) | 1 x 1 x 2 = 2 |

In total there are 32 + 8 + 2 = 42 possible temporal model of length 2 that satisfies the formula ( x ų ¬y )

# Construct an LTLF formula

Traffic light describing the specification:

* the light is either green or red, but never both
* whenever the light is red, it will eventually turn green

It must be a conjunction of the two properties

( green v red ) ^ ¬( green ^ red ) ⟹ (green XOR red)

^

(red → ♢green) ♢ eventually

!!! add the box ☐ (always) in front, you are not thinking locally but in general, so you need ☐ !!!

☐(green XOR red) ^ ☐(red → ♢green)

## Extend the specification

Two traffic lights (with variables greeni and redi (i = 1, 2) such that the two green lights are never simultaneously on.

Use the specification before

☐(green1 XOR red1) ^ ☐(red1 → ♢green1) ^ ☐(green2 XOR red2) ^ ☐(red2 → ♢green2)

and add with a the new specification

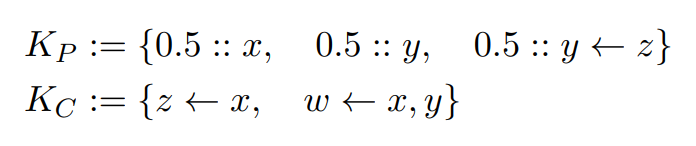
^ ☐( ¬(green1 ^ green2))

!!! It is not satisfiable !!!

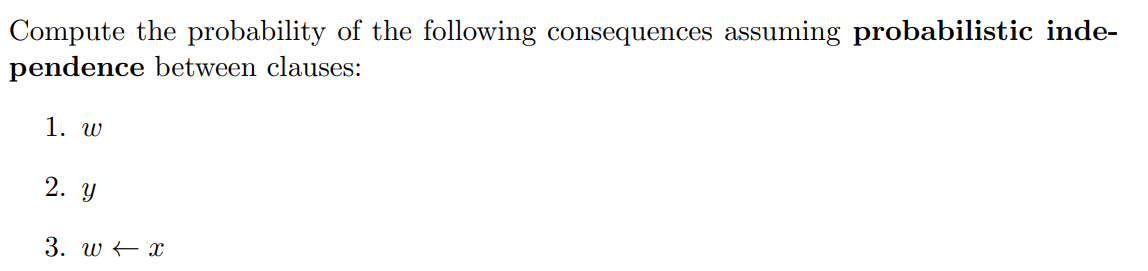
Because a temporal model of a traffic light always finishes when it is green, for the temporal model to finish both lights have to be green, but can have both green at the same time. A finite model does not exist a finite model.

# Reasoning with probabilistic knowledge base

## Compute the probability of the sequences



There are three probabilistic clause, each one can be true or false, so there are 23 = 8 possible words



The first number represents the first probabilistic clause…

|  | 000 | 001 | 010 | 100 | 011 | 110 | 101 | 111 |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P(w) | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3/8 |
| P(y) | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 5/8 |
| P(w←x) | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 6/8 |

P(w) = 3/8

P(y) = 3/8

p(w←x) =

In the last example 001 that is:

y←z

z←x

w←x,y

The first two clause of can be reduce to y←x, that transform the third one in w←x, so P(w←x) is 1

## Find a probabilistic distribution

If 0.5::y is not a probabilistic clause but a constraint it means y must have probability 0.5 (and not 5/8 from the exercise before)

Kp = { 0,5::x, 0,5::y←z }

Kc = { z←x, w←x,y }

Now there are 4 worlds (22)

The only way that we can entail y is with the axion 0,5::y←z because no other axiom have y in the head, so the probability 0,5 of y must be distributed between 01 and 11 (so when 0,5::y←z is true). But the only way is actually 11 because we need z to be also true, so x must be true. The world 11 must have probability 0,5 then 01 has probably 0 (because the sum must give 0,5), also 10 must have probability 0 because (P(10) +P(11)) must be 0.5 and P(11)=0,5. P(00) must be 0,5 because the total sum must be 1

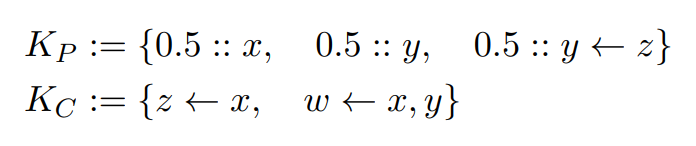
Compute P(w), the first number represent x, the second y←z

| **00** | **01** | **10** | **11** |
| --- | --- | --- | --- |
| 0.5 | 0 | 0 | 0.5 |

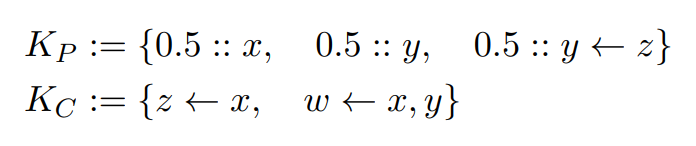
P(w) = 0.5

## Extreme probabilities

I need a probabilistic knowledge base



And a classical knowledge base



Find the smallest and biggest probability that can be assigned in any distribution that satisfy this constraint ( in this example w )

First of all we have to check which distribution satisfy w

Than check which one in particular minimise or maximise the probability of the words

The word that satisfy w are 110, 101, 111

Lower probability of w P(w) = minP € coh(k) P(101)+P(110)+P(111)

Is the minimum over all the probability distribution that are coherent with this Knowledge base (K)

Maximum probability of w (w) = maxP € coh(k) P(101)+P(110)+P(111)

Is the maximum over all the probability distribution that are coherent with this Knowledge base (K)

# Exercises from Mock

## Mock 1 - Boolean Functions

To construct a ternary boolean function that returns true in exactly five inputs you need to create a truth table composed of 3 variables (x, y, z) and each of them can be true or false. That is there are 23 = 8 possible combinations of inputs.

Is it always possible to express the ternary boolean function as a propositional formula. To do so you need to write all the possible combinations that make the formula true and connect them with a disjunction.

| X | Y | Z | Φ |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Becomes as a propositional formula:

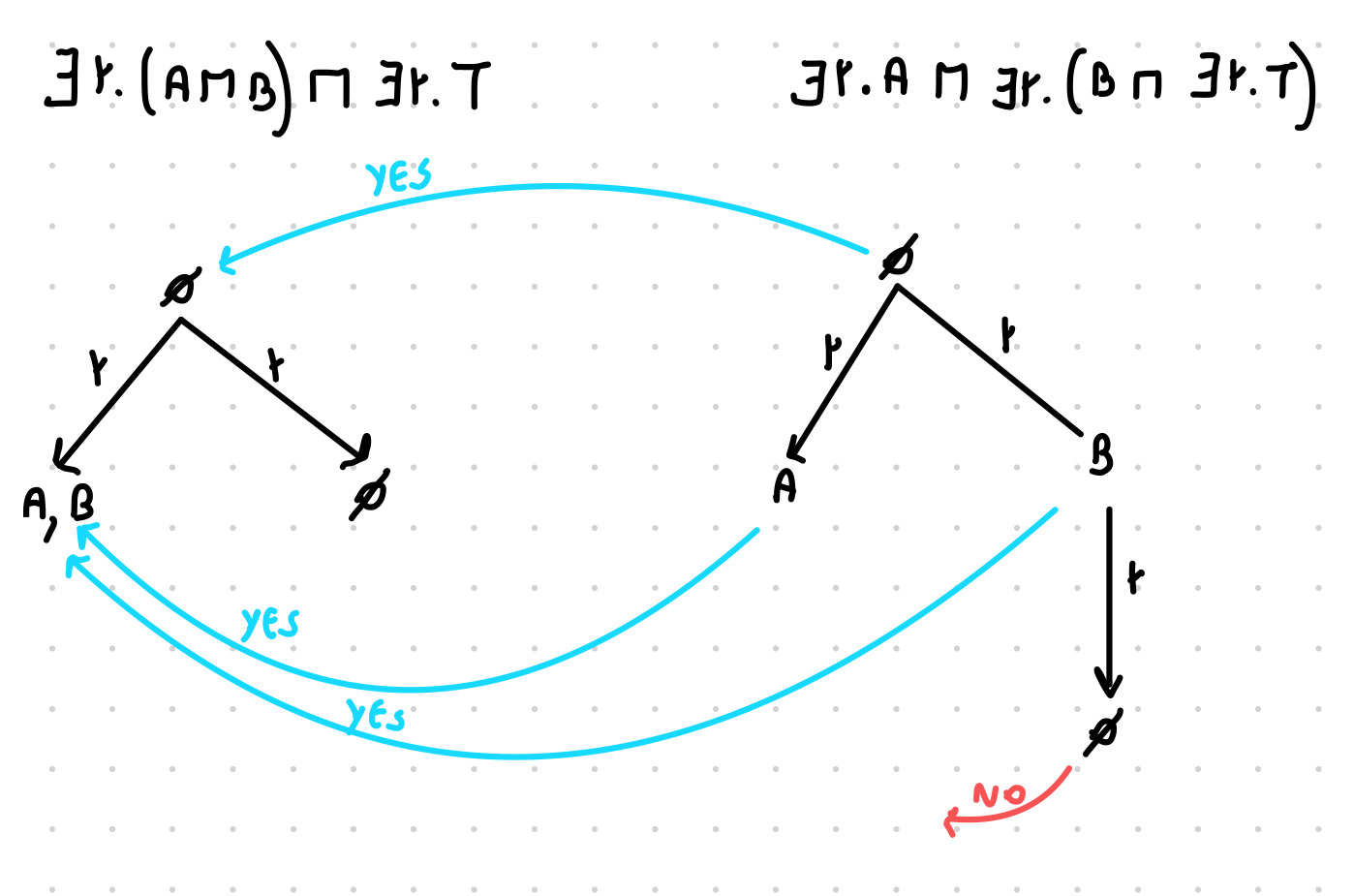
( x ^ ¬y ^ ¬z) v ( ¬x ^ y ^ z ) v ( x ^ ¬y ^ z ) v ( x ^ y ^ ¬z ) v ( x ^ y ^ z )

It is not always possible to express a propositional formula as a propositional Knowledge Base. This is because it requires an additional step, that is to be a set of Horn Clauses, that is a conjunction (!!! apply first the distributivity rules) of Horn Clauses (a Horn Clause is a disjunction with exactly one positive literal). As an example (x v y v ¬z) can not be expressed as a propositional Knowledge Base.

## Mock 2 - EL

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

To check if the subsumption relation holds or not without a TBox we need to apply the homomorphism.



The subsumption relation does not hold because on the left we have a concept of depth 1, while on the right there is a concept of depth 2.

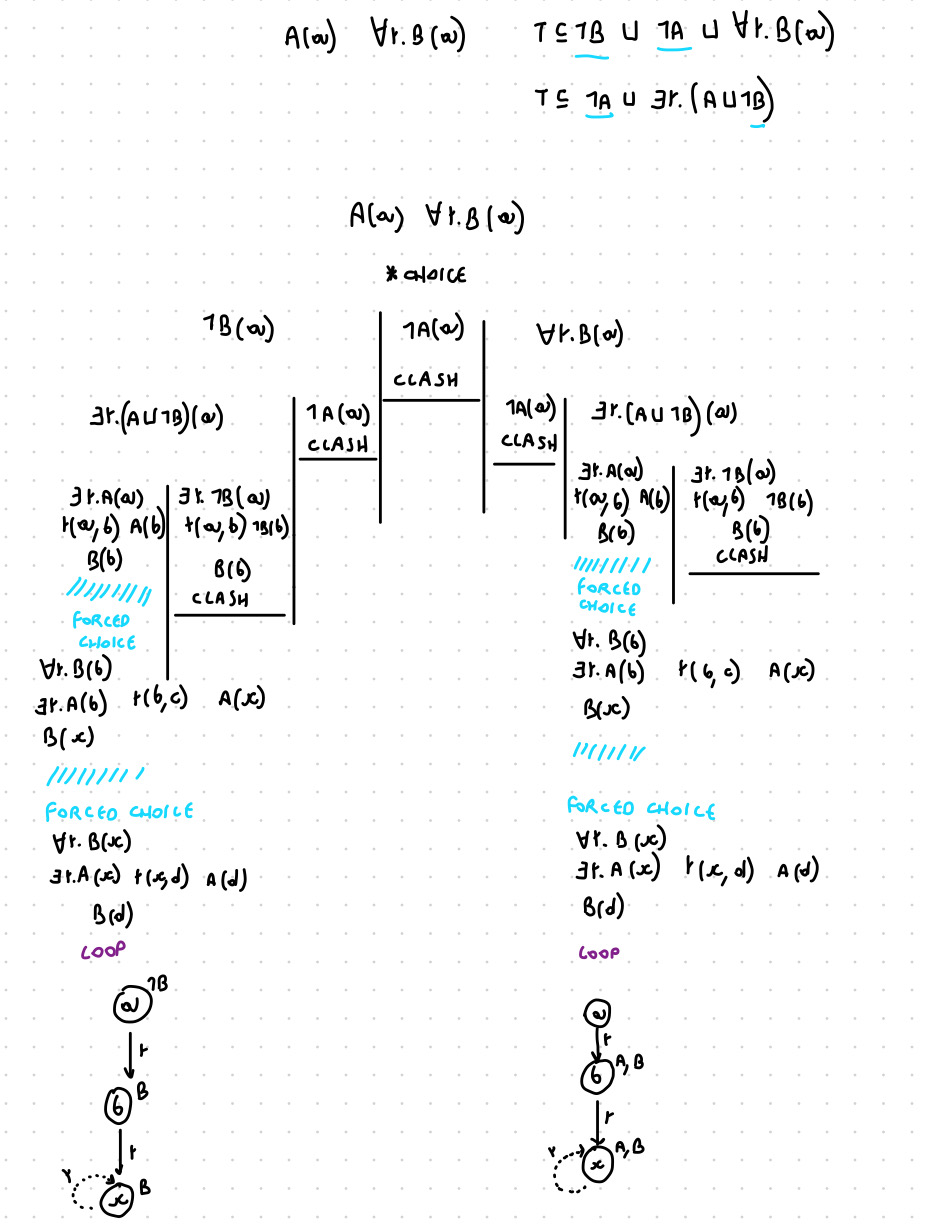
Adding a TBox of the type T ⊑ ∃r.B will make the subsumption relation holds. This is because you will have on the left side a new r-edge going from A,B to B.

Otherwise to check if you can apply the completion algorithm.

## Mock 3 - ALC

To use the tableau algorithm to check if a concept is satisfiable with respect to a TBox we need to

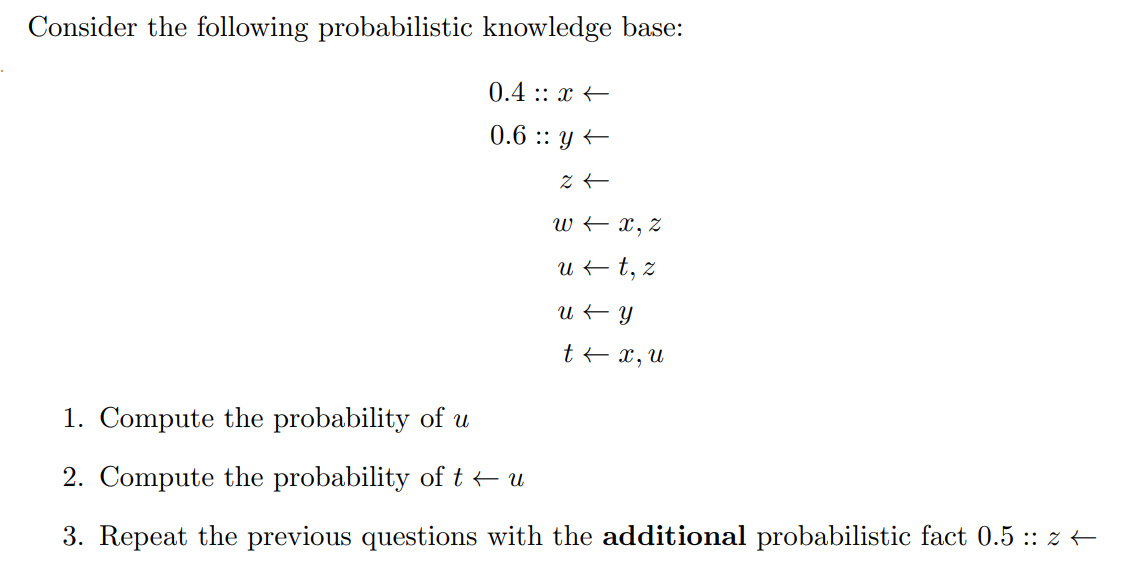
* Put the TBox in normal form: B ⊑ ¬(A ∏ ∃r. ¬B) becomes ( B ⊑ ¬A ⊔ ∀r. B )
* Rewrite the Tbox: ( B ⊑ ¬A ⊔ ∀r. B ) becomes T ⊑ ¬B ⊔ ¬A ⊔ ∀r. B and A ⊑ ∃r.( A ⊔ ¬B) becomes T ⊑ ¬A ⊔ ∃r.( A ⊔ ¬B)



To make it inconsistent by adding a new GCI it is sufficient to add to the TBox ∃r.A ⊑ ⊥

Or technically you can add A ⊔ B ⊔ ¬A ⊔ ¬B ⊔ ∃r.A ⊑ ⊥ or ¬B ∏ ∃r.T ⊑ ⊥

## Mock 4 - Probabilistic Reasoning



!!! Because the probability is not all equal to 0.5, the possible words have different weights !!!

| **0.4:: x** | **0.6:: y** | **u** | **t ← u** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |

P(u) = [ (1-0.4) \* 0.6] + [ (0.4 \* 0.6) ] = 0.6

P(t←u) = [ 0.4 \* (1 - 0.6) ] + [ 0.4 \* 0.6 ] = 0.4

Adding the additional probabilistic fact 0.5:: z ← will not change the probability of *u* and *t ← u* because they are independent from z and also because is present the classical fact z ← already.

## Mock 5 - Temporal Logic

*If b is observed then a must have been observed before*

If *before* is considered as just the spot exactly before of b than it can be written as:

a → ⚪b

# Exercise from Exam 220207

## Exam 1 - Boolean Functions

Since it is a ternary boolean function there are 23 = 8 different inputs and there are 28 different ternary boolean functions.

Not all the propositional formulas composed by the variables x, y, z can be expressed as a propositional knowledge base. This is because to do so is also required to be a Horn Clause, (that is a clause with exactly one positive literal). As an example (x v y v ¬z) or ( x v y v z ) or ( ¬x v ¬v v ¬z ) can not be expressed as Horn clauses and so they can not be expressed as a propositional knowledge base.

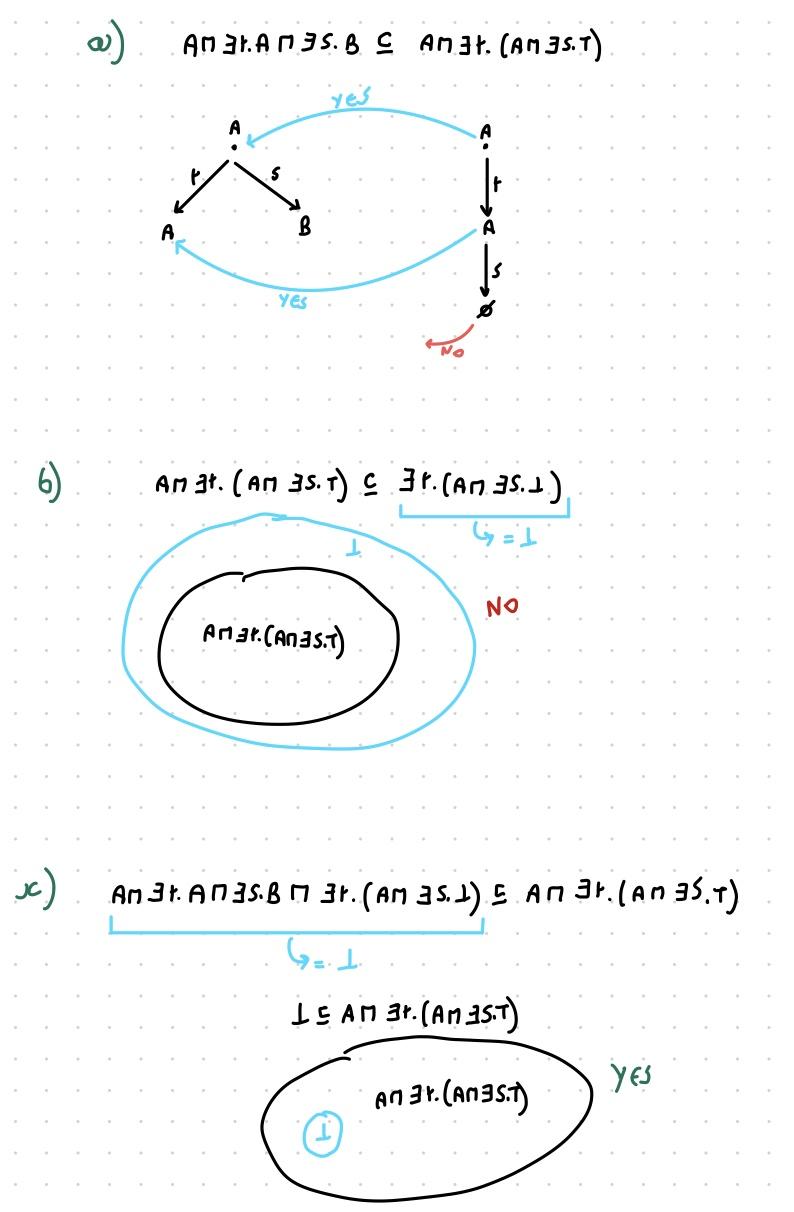
## Exam 2 - EL I

Since in EL there is no ¬ and no ⊥ then I can not add any constraint, so there will be always an interpretation that satisfies C

An EL⊥ might have some inconsistent model, so there might be no model

## Exam 3 - EL II

To check if a subsumption relation hold without a TBox is sufficient to do the homomorphism method.

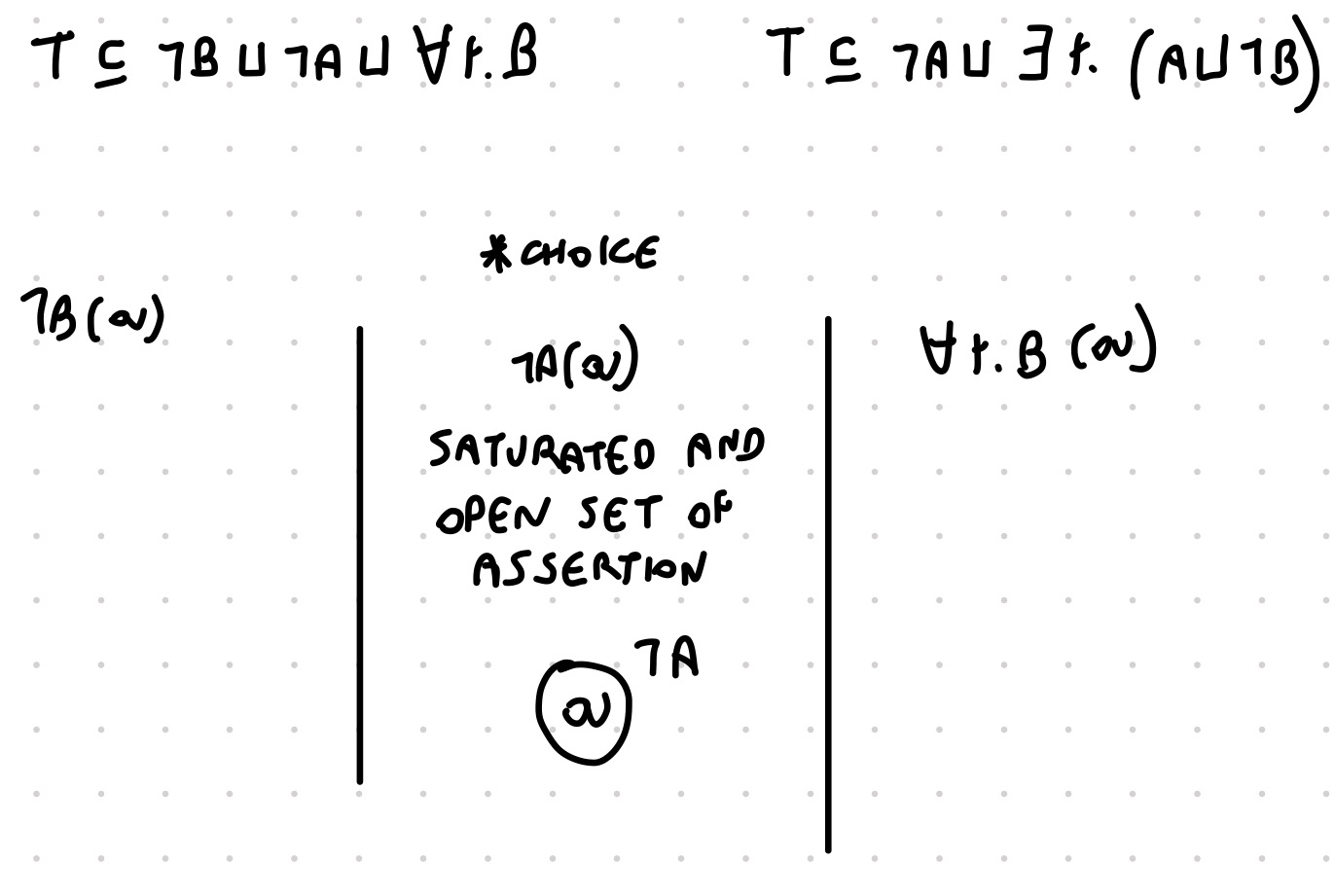


Construct one GCI three subsumption relations holds???

## Exam 4 - ALC

Use the tableau algorithm to show that the TBox is consistent.

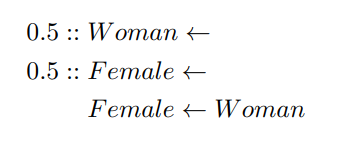
To be consistent we just need to find at least one saturated and open set of assertions.



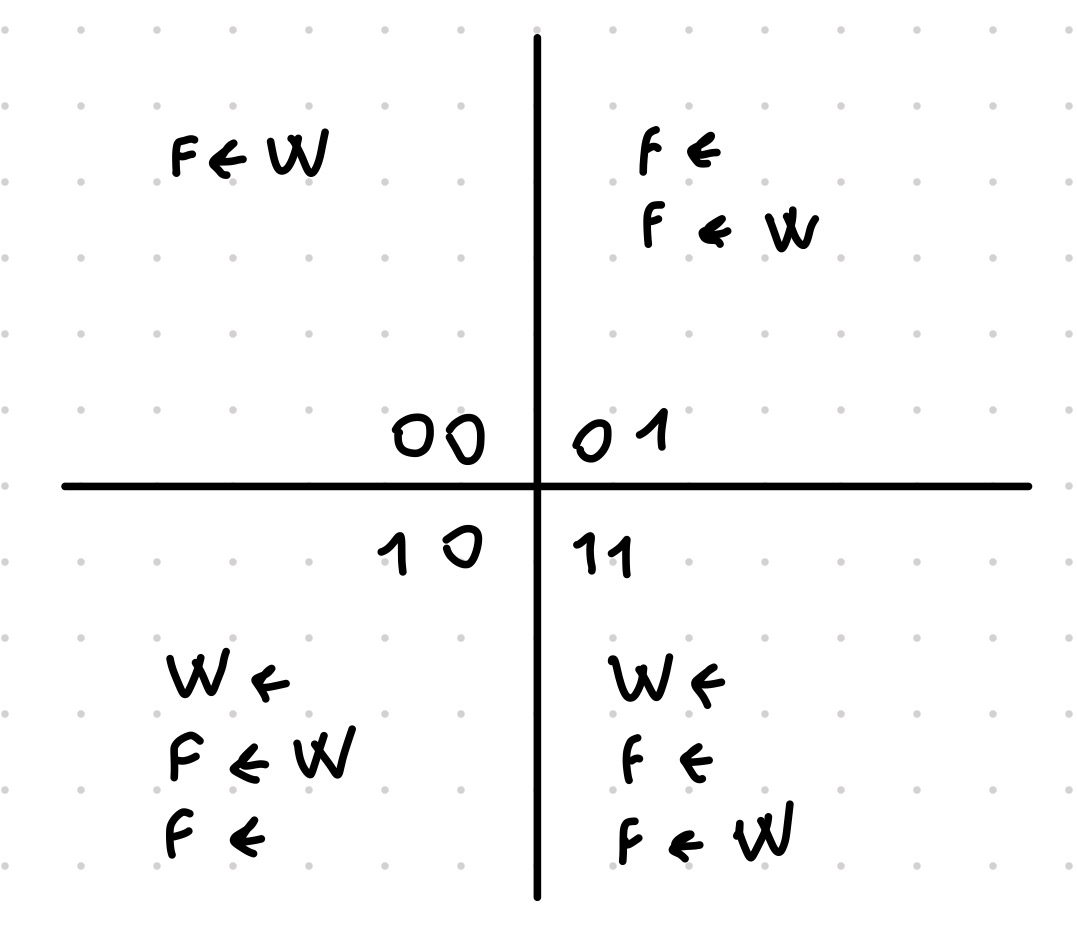
To rewrite the TBox as only one GCI is sufficient to use a conjunction

T ⊑ ( ¬B ⊔ ¬A ⊔ ∀r.B ) ∏ ( ¬A ⊔ ∃r.( A ⊔ ¬B ))

## Exam 5 - Probabilistic Reasoning



Compute the probability of *Woman* and *Female*



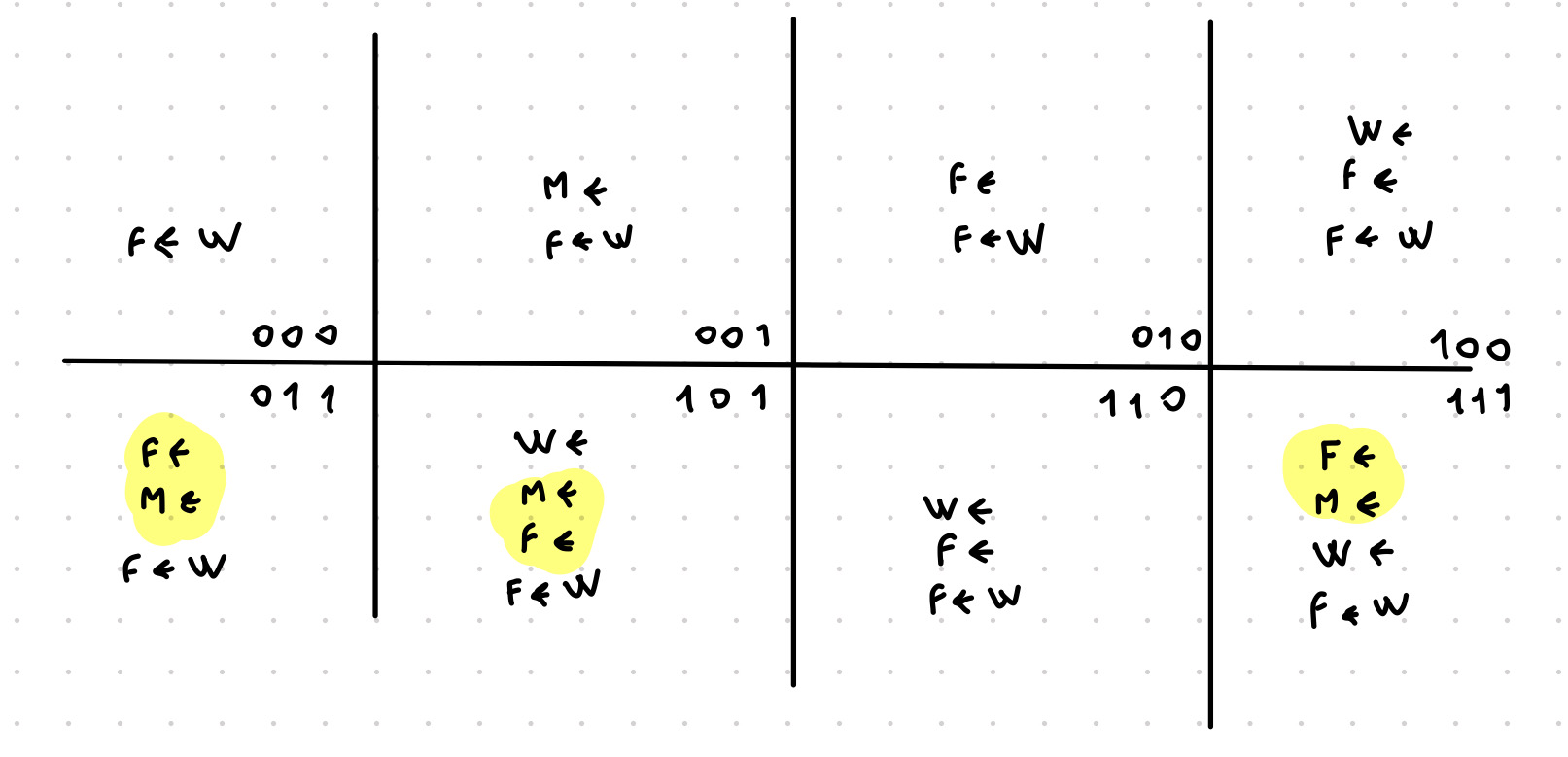
P(*female*) = 3/4 = 0.75

P(*woman*) = 2/4 = 0.5

This result is reasonable in two main possible scenarios

* If we are considering female the sex (including all the living species) and woman just the female gender of human. In this case it is reasonable because all women are female but not all females are women.
* If we are considering just the human race and female is the gender and women are female with at least 18 years old.

If we add *0.5 :: Male ←*



We would have the probability of being female and male at the same time equal to 3/8

## Exam 6 - Temporal Logic

To build a temporal model M such that M,0 ⊨ ¬a of ¬a ų ( ¬⚪T v a )

Why is the formula a tautology?

The model ¬a, a, ¬a will satisfy it.