Automated Reasoning – Assignment

Sofia Santilli – 1813509

13th September 2022

Exercise 1

Mostrare un esempio di esecuzione di GSAT su una formula di quattro variabili in cui il primo tentativo non trova un modello dopo sette passi e il secondo invece lo trova dopo tre passi, dove un passo e' il passaggio da una interpretazione a un'altra.

GSAT is an incomplete method for finding a model of a formula F. It works on CNF formulae.

The algorithm outputs a model for F, if found, and takes as input a formula F and two parameters:

- the maximum number of steps N_{STEPS}, that is the maximum number of times a variable is chosen in order to be flipped (if x=true then it is set to false. If x=false, it is set to true) and so a new interpretation is considered.
- the maximum number of tries N_{TRIES}. At the beginning of each new try, a new random interpretation is taken.

The pseudocode describing the algorithm is the following:

GSAT algo:

```
variables = [w, x, y, z]
for (i<N_TRIES, i=0, i++) {
        pick a random interpretation I
        if I satisfies all clauses, return true
        for (j<N_STEPS, j=0, j++) {
                dict_I = {}
                for (k in variables) {
                        I_k' = interpretation obtained by inverting the value of the variable k in I
                        N_SAT = number of satisfied clauses of I_k'
                        add the (key, value) couple (I_k', N_SAT) to dict_I
                }
                I = the interpretation from dict_I with the maximum N_SAT value associated(**)
                if I satisfies all clauses, return true
        }
}
return false
```

(**) if two interpretations in *dict_i* both maximize N_{SAT}, we choose the one with the changed variable coming first in alphabetical order.

If the algorithm returns false, it means that GSAT wasn't able to find a model with that N_{STEPS} and N_{TRIES}. The running time of the algorithm is given by $N_{STEPS}*N_{TRIES}$. In this case, we don't know if the formula is satisfiable or not (incompleteness).

For the required example, since I have to find the model during the second try, N_{TRIES} is at least 2: so we can set $N_{TRIES}=2$. While the words *"al primo tentativo non trova un modello dopo 7 passi"* brought me to set $N_{STEPS}=7$.

The formula F proposed for the example has 4 variables: *w*, *x*, *y* and *z* and is made up of 5 clauses:

$$\boxed{F = \{z,\ x \vee y,\ w \vee \neg y,\ \neg z \vee \neg y,\ y \vee w\}}$$

N_{SAT} = number of satisfied clauses in F. ***given a certain interpretation I***

$\boxed{1^{ST}\ \text{TRY:}}$

I take a random interpretation:

$$I = \{w=F,\ x=F,\ y=T,\ z=F\} \qquad N_{SAT} = 0+1+0+1+1 = 3.$$

$\boxed{\text{STEP } 1}$

$$I'_w = \{\underline{w=T},\ x=F,\ y=T,\ z=F\} \qquad N_{SAT} = 0+1+1+1+1 = 4.$$

$$I'_x = \{w=F,\ \underline{x=T},\ y=T,\ z=F\} \qquad N_{SAT} = 0+1+0+1+1 = 3.$$

$$I'_y = \{w=F,\ x=F,\ \underline{y=F},\ z=F\} \qquad N_{SAT} = 0+0+1+1+0 = 2.$$

$$I'_z = \{w=F,\ x=F,\ y=T,\ \underline{z=T}\} \qquad N_{SAT} = 1+1+0+0+1 = 3.$$

N_{SAT} is maximized by changing w to true.

$$I = I'_w = \{w=T,\ x=F,\ y=T,\ z=F\}.$$

Step 2 starts from this new interpretation I

$\boxed{\text{STEP } 2}$

$$I'_w = \{\underline{w=F},\ x=F,\ y=T,\ z=F\} \qquad N_{SAT} = 0+1+0+1+1 = 3.$$

$$I'_x = \{w=T,\ \underline{x=T},\ y=T,\ z=F\} \qquad N_{SAT} = 0+1+1+1+1 = 4.$$

$$I'_y = \{w=T,\ x=F,\ \underline{y=F},\ z=F\} \qquad N_{SAT} = 0+0+1+1+1 = 3.$$

$$I'_z = \{w=T,\ x=F,\ y=T,\ \underline{z=T}\} \qquad N_{SAT} = 1+1+1+0+1 = 4.$$

N_{SAT} is maximized by changing x or z. I choose x, because it comes first in alphabetical order. $\Rightarrow x=T$.

$$I = I'_x = \{w=T,\ x=T,\ y=T,\ z=F\}.$$

$\boxed{\text{STEP } 3}$

$I'_w = \{\underline{w=F},\ x=T,\ y=T,\ z=F\}$ $\quad N_{SAT} = 0+1+0+1+1 = 3.$

$I'_x = \{w=T,\ \underline{x=F},\ y=T,\ z=F\}$ $\quad N_{SAT} = 0+1+1+1+1 = 4,$

$I'_y = \{w=T,\ x=T,\ \underline{y=F},\ z=F\}$ $\quad N_{SAT} = 0+1+1+1+1 = 4.$

$I'_z = \{w=T,\ x=T,\ y=T,\ \underline{z=T}\}$ $\quad N_{SAT} = 1+1+1+0+1 = 4.$

I choose, for alphabetical order, $x=F$.

$$I = I'_x = \{w=T,\ x=F,\ y=T,\ z=F\}.$$

Now, we can notice that the last I found is the same obtained at the end of step 1. Therefore:

- <u>step 4</u> will be exactly the same as step 2.
- <u>step 5</u> will be the same as step 3.
- <u>step 6</u> will be the same as step 2.
- <u>step 7</u> will be the same as step 3.

Therefore the algorithm, during this first try, is stuck in a LOCAL MAXIMA. After 7 steps, it didn't manage to find a model for the formula F.

2nd TRY

I take a different random interpretation:

$I = \{ w=F, x=F, y=F, z=F \}$. $N_{SAT} = 0+0+1+1+0 = 2$.

STEP 1

$I'_w = \{ \underline{w=T}, x=F, y=F, z=F \}$ $N_{SAT} = 0+0+1+1+1 = 3$

$I'_x = \{ w=F, \underline{x=T}, y=F, z=F \}$ $N_{SAT} = 0+1+1+1+0 = 3.$

$I'_y = \{ w=F, x=F, \underline{y=T}, z=F \}$ $N_{SAT} = 0+1+0+1+1 = 3.$

$I'_z = \{ w=F, x=F, y=F, \underline{z=T} \}$ $N_{SAT} = 1+0+1+1+0 = 3.$

Choose $w=T$ for alphabetical order. $I = I'_w = \{ w=T, x=F, y=F, z=F \}$.

STEP 2

$I'_w = \{ \underline{w=F}, x=F, y=F, z=F \}$ $N_{SAT} = 0+0+1+1+0 = 2$

$I'_x = \{ w=T, \underline{x=T}, y=F, z=F \}$ $N_{SAT} = 0+1+1+1+1 = 4.$

$I'_y = \{ w=T, x=F, \underline{y=T}, z=F \}$ $N_{SAT} = 0+1+1+1+1 = 4.$

$I'_z = \{ w=T, x=F, y=F, \underline{z=T} \}$ $N_{SAT} = 1+0+1+1+1 = 4.$

Choose $x=T$ for alphabetical order. $I = I'_x = \{ w=T, x=T, y=F, z=F \}$.

STEP 3.

$I'_w = \{ \underline{w=F}, x=T, y=F, z=F \}$ $N_{SAT} = 0+1+1+1+0 = 3.$

$I'_x = \{ w=T, \underline{x=F}, y=F, z=F \}$ $N_{SAT} = 0+1+1+1+1 = 4.$

$I'_y = \{ w=T, x=T, \underline{y=T}, z=F \}$ $N_{SAT} = 0+1+1+1+1 = 4.$

$I'_z = \{ w=T, x=T, y=F, \underline{z=T} \}$ $N_{SAT} = 1+1+1+1+1 = 5$

Choose to change z in true.

N_{SAT} has maximum value $\Rightarrow$ $I = I'_z = \{ w=T, x=T, y=F, z=T \}$ is a MODEL for formula F.

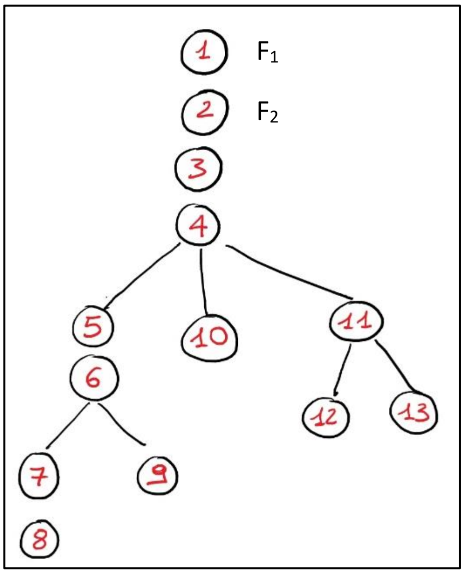
Exercise 2

Esiste un algoritmo che trasforma un qualsiasi tableau ground chiuso in un tableau con unificazione chiuso per la stessa formula? Ne esiste uno per la trasformazione opposta? In caso positivo descrivere l'algoritmo nei suoi singoli passi.

Tableau is a method to prove the unsatisfiability of a set of formulae. The principle consists in dividing the formula in its components more and more, until we come up to the simplest components. In this way it is easier to find contraddiction if it exists. If a branch contains complementary literals, close the branch. If all the branches were closed, the tableau is closed and the set of formulae is unsatisfiable.

Let's see how to tanslate the two different types of tableau one in the other.

It is given a set S of formulae (F_1, F_2, …), that needs to be checked for satisfiability through the tableaux method.



I treat the tableau as a tree. So I visit it with a pre-order visit. This allows me to visit the nodes of the tableaux in the same order of appearance of when the tableaux was built. In this way the formulae in S will be visited first (*in the example on the left 1 and 2*); the nodes that develop in a line are visited from the top to the bottom (*1-2-3-4, 5-6, 7-8*); when there is a ramification, I'll first visit nodes on the leftmost branch, to arrive finally to nodes in the rightmost branch (regarding the ramification starting in 4, first I'll visit all the nodes in the branch starting in 5, then the nodes starting in 10 (only itself here) and finally the nodes starting in 11. Same for the other ramifications, when met). In this way the whole tableau will be certainly transposed. So the nodes, in the figure, are enumerated according to the order in which they are visited.

In order to write the algorithms, we define some rules (rule names were established by me for convenience). Some aspects remain the same when translating, while some others change between the two types of tableaux: based on this I distinguished between two types of rules.

In the following: "tableau A" is the tableau we want to translate, whatever type it is, and "tableau B" the translation to the other type.

Rules that do Not cause changes in the other tableau:

1) (**starting rule**) the first nodes visited in tableau A are the ones containing the formulae of the initial set S. These nodes are simply rewritten in tableau B as first nodes.

In order to distinguish these initial nodes, the algorithm needs to know what are the nodes that weren't originated starting from other nodes. So these information need to be originally associated to each node of the tree structure in input to the algorithm.

2) (**ramification rule**) if after the node of tableau A there is a ramification, the *ramification* is also reported in tableau B. In fact it means a disjunction will be developed. And disjunctions, in both types of tableaux, develop following the same rule:

$$\frac{A \vee B}{A \mid B}$$

Also conjunction in both tableaux is developed following the same rule $\frac{A \wedge B}{\substack{A \\ B}}$ but it is not necessary to create a translating rule dedicated to it, since the case is already managed by rule 3, that tells how to deal with both the premise $A \ \wedge B$ and the consequences A, B.

3) (**non-quantified-origin rule**) if in the node of tableau A there is:

   a) a universally or existentially quantified formula F that was not obtained from the expansion of another quantified formula, OR

   b) a literal that was not obtained from the expansion of a quantified formula, OR

   c) a formula with conjunctions (∧) and/or disjunctions (∨) of literals, or of subformulae (quantified or not), that was not obtained from the expansion of a quantified formula,

   then F is written unchanged also in tableau B.

4) (**from-existential-1 rule**) If in a node of tableau A there is the expansion of an existential quantifier in the form F[c/x] (where the variable x has been substituted by a new constant c), F[c/x] is rewritten unchanged in tableau B in the case there are no free variables in F. If free variables are in F, see rules 6 and 8.

5) (**conj-disj order**) It is more an *observation* than a rule. In fact it can not be applied to the single node we are visiting in that moment. If a tableau A expands conjunctions and disjunctions in a certain order, it will be mantained also in the translated tableau B. Generally conjunctions are expanded before disjunctions in order to avoid the creation of too many branches.

Rules that cause Changes in the other tableau:

6) (**from-existential-2-GU rule**) If in a node of the Ground tableau there is the expansion of an existential quantifier, and so there is a formula as F[c/x] (where the variable x has been substituted by a new constant c), in the case there are free variables x_1, x_2, …, x_N in F, the formula is translated in the Tableau with Unification as F[f(x_1, x_2, …, x_N)/x]. That is a formula in which the constant c has been replaced by f(x_1, x_2, …, x_N), meaning that the variable that was existentially quantified is function of the free variables in F.

7) (**from-universal-1-GU rule**) If in a node of the Ground tableau there is the expansion of an universal quantifier, that is a formula as F[t/x] (where t is a ground term), it is translated in the Tableau with

Unification as F[x_i/x], where x_i is a placeholder. Therefore x_i will remain in the tableau until Unification algorithm will be applied to close some branch. Remember that x_i is a rigid variable, therefore it can be assigned only to a value that remains in the whole tableau.

A particular case is when in the ground tableau F[t/x] is inside an existential quantifier: for example $\exists y\, F[t, f(y)]$, where the ground term t replaced the universally quantified x. In this case, when tanslating to the tableau with unification, it is possible to put f(y) in place of t, obtaining $\exists y\, F[f(y), f(y)]$ (substitution with variables).

8) (**from-existential-3-UG rule**) If in a node of the Tableau with Unification there is the expansion of an existential quantifier in the form F[f(x_1, x_2, …, x_N)/x], where x_1, x_2, …, x_N are the free variables contained in F, it is translated in the Ground tableau as F[c/x], where c is a new constant.

9) (**from-universal-2-UG rule**) If in a node of the Tableau with Unification there is the expansion of an universal quantifier, so in the form F[x_i/x], it is translated in the Gound tableau as F[t/x] where t, if possible, has to be put equal to an already existing ground term in the ground tableau. About that, pay attention to the rule 10.

10) (**quantifiers order rule**) In ground tableau existential quantifiers are expanded first: in this way new constants are introduced that can be used by universal quantifiers as ground terms.

Instead, in tableau with unification, quantifiers are expanded in the order of their appearance, since universal quantifiers use placeholders and so they do not need to know immediately the term to replace the universally quantified variable with.

Therefore, when translating a tableau with unification in a ground tableau, before expanding a universal quantifier you need to check if a useful ground term is already present in the tableau, or if an existential quantifier has already been developed. Otherwise, keep in memory the node, and come back later to expand it.

Pay attention: in rules 4, 6, 7, 8, 9, the formula in the node examined may also be in the scope of another quantifier. In this case, when translating, do what said by the rule, rewriting also the other quantifier outside the F[…].

Algorithm for translating a ground tableaux into a tableaux with unification

Main_algo:

- input: ground tableau T
- output: *new_tableau*, the corresponding tableau with unification

```
begin
    new_tableau = empty tree structure    # global structure that will contain the translated tableau
    translating_algo(T)
    apply the Unification Algorithm on new_tableau in order to close branches
end
```

translating_algo:

- input: ground tableau
- output: builds the correspondent tableau with unification (before applying Unification Algorithm)

```
begin
    # the tableau is visited in pre-order
    if (nodo N == null) return;
    Visita(N)
    while N has children C:      # loop from the leftmost child to the rightmost one
            translating_algo(C)     # recursive call
 end
```

Visita:

- input: node N (with the correspondent formula) of the ground tableau
- output: translation of N in the tableaux with unification

```
begin
    if N is a starting node:
            update new_tableau by applying  (1) starting rule
            return
    update new_tableau by applying rules: 2, 3, 4, (5), 6, 7, (10)
end
```

<u>Algorithm for translating a tableaux with unification into a ground tableaux</u>

The ***Main_algo*** function is the same except for the fact that, when translating the tableau with unification to the corresponding ground tableau, you don't have to perform the last operation, that is the application of the Unification algorithm.

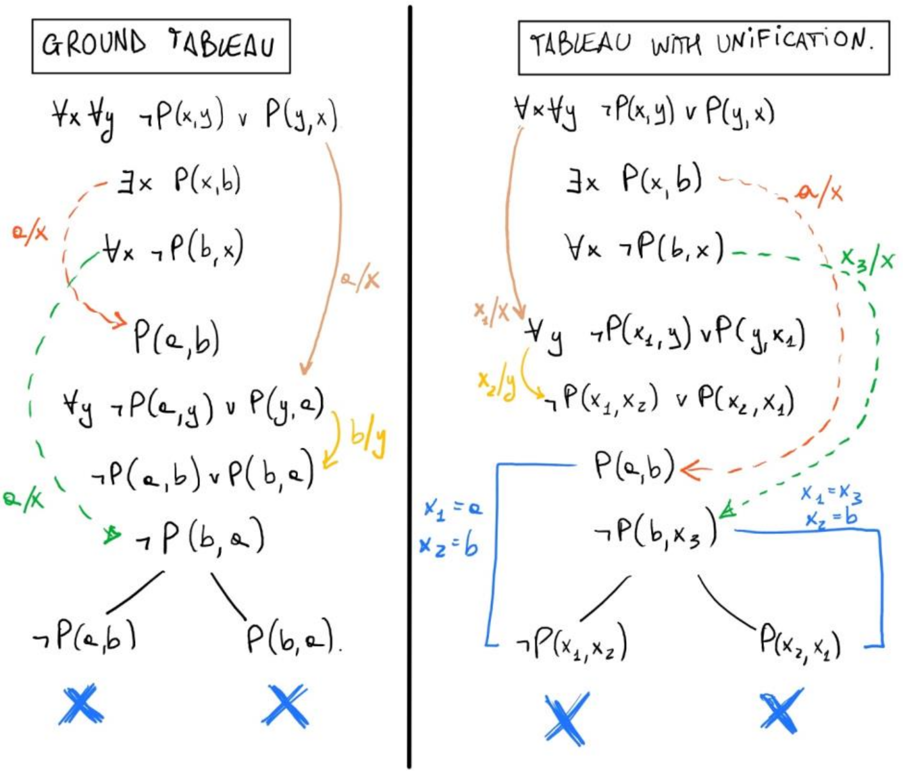
The ***translating_algo*** function is identical to the precedent case.

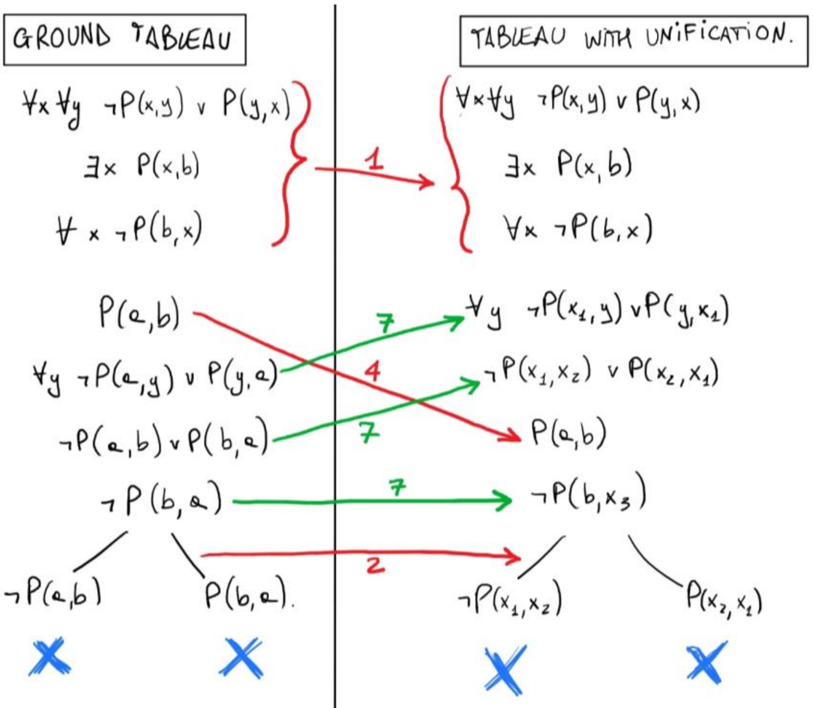
The ***Visita*** function changes in the rules to apply outside the "if". So, the update based on rule 1 done inside the "if" doesn't change. While the second update performed on the new ground tableau is now based on some different rules: 2, 3, 4, (5), 8, 9, (10).

The other thing that changes in the three functions is the input, that is a tableau with unification, and the output, that is a ground tableau now.

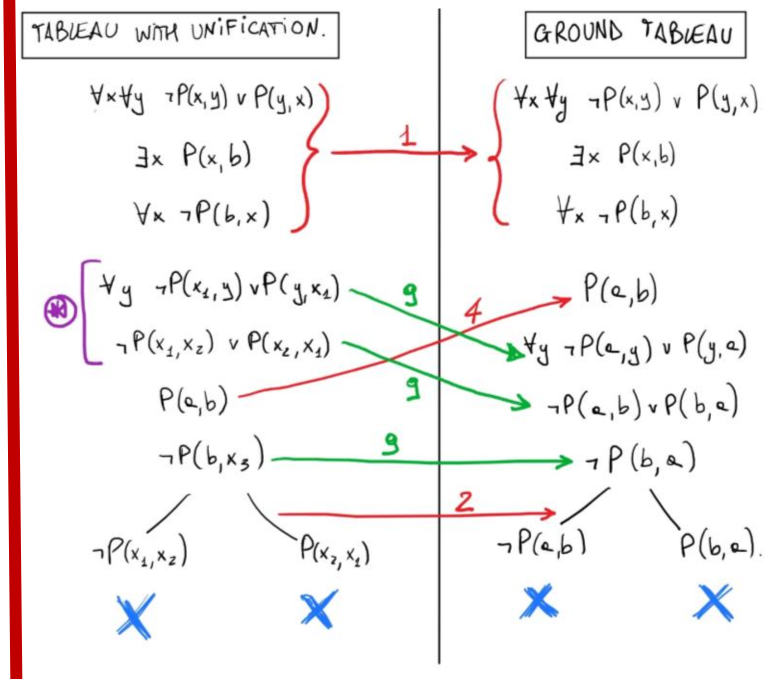
Example were some of the presented rules are applied:

$$S = \{\ \forall x \forall y\ (\neg P(x,y) \lor P(y,x)),\ \ \exists x\ P(x,b),\ \forall x\ \neg P(b,x) \}$$





ends by applying Unification algo: $\begin{cases} x_1 = x_3 = a. \\ x_2 = b. \end{cases}$



As told in rule 10, when translating to the ground tableau, ⊛ these nodes aren't expanded until the existential quantifier was expanded. In this way, when expanding universal quantifiers, we had a better idea of what ground terms to use.

Exercise 3

Scrivere una dimostrazione in deduzione naturale in cui si usano sia la regola di eliminazione dei quantificatori esistenziali che quella di introduzione dei quantificatori universali; usare queste due regole deve essere necessario per ottenere la formula finale.

Natural deduction is a deduction method that allows to prove entailment. For example $A \vDash B$ if from A (premise) is possible to infer B. Proof is always a line.

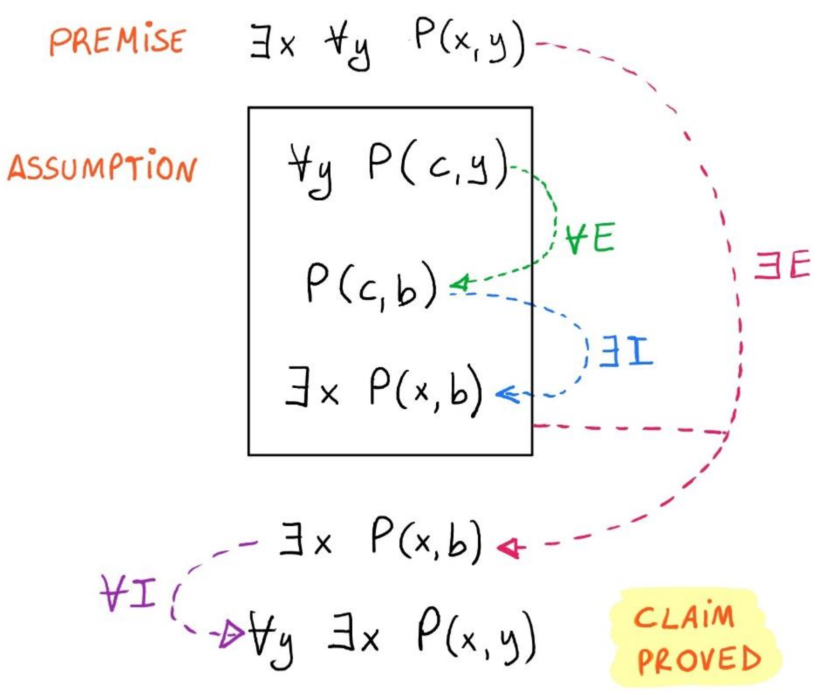
When working with First-order Natural deduction, we have the following rules in order to deal with quantifiers:

- Universal elimination: $\dfrac{\forall x\ A}{A[t/x]}\ (\forall E)$ where t is a term;
- **Universal instantiation**: $\dfrac{A[y/x]}{\forall x\ A}\ (\forall I)$ where y is a variable not occurring free in the premises and in the active assumptions;
- Existential instantiation: $\dfrac{A[t/x]}{\exists x\ A}\ (\exists I)$ where t is a term;
- **Existential elimination**: $\dfrac{\exists x\ A \quad \begin{pmatrix} A[y/x] \\ \vdots \\ C \end{pmatrix}}{C}\ (\exists E)$ where y does not occur free in C, in the premises and in the active assumptions;

$A[z/x]$ means that in formula A the variable x has been replaced by z.

A demonstration through Natural deduction has been applied to the following entailment:

$$\exists x\, \forall y\, P(x,y) \vDash \forall y\, \exists x\, P(x,y)$$



Regarding existential elimination ($\exists E$), first I took as assumption the argument of $\exists x$, $\forall y\, P(x,y)$, in which *x* has been replaced with c (c that does not occur free in the premise or in $\exists x\, P(x,b)$, entailed from the assumption). From this assumption, in the box, I derived $\exists x\, P(x,b)$ and following the rule, I derived it also outside the box.

Regarding the universal instantiation ($\forall I$), I can apply it since *b* is a variable not occurring free in the premise or in the assumption.