Resolution

Until now, we have seen how logical reasoning could be used to discover new facts in a knowledge base (through logical entailment). The reasoning was done by hand, in a kind of informal manner.

We are looking for a procedure that can determine whether or not KB $\models \alpha$, where KB is a given knowledge base and α is a sentence.

Also, if $\beta[x_1,\ldots,x_n]$ is a formula with free variables, we want a procedure that determines terms t_i, if they exist, such that KB $\models \beta[t_1,\ldots,t_n]$.

Resolution

Until now, we have seen how logical reasoning could be used to discover new facts in a knowledge base (through logical entailment). The reasoning was done by hand, in a kind of informal manner.

We are looking for a procedure that can determine whether or not KB $\models \alpha$, where KB is a given knowledge base and α is a sentence.

Also, if $\beta[x_1,\ldots,x_n]$ is a formula with free variables, we want a procedure that determines terms t_i, if they exist, such that KB $\models \beta[t_1,\ldots,t_n]$.

But there is no automated procedure to fully satisfy this requirement (in all cases).

We are looking for a procedure that does deductive reasoning in a manner as sound and complete as possible and in a language as close as possible to FOL.

Resolution

A reasoning process is logically sound if whenever it produces α, the α is guaranteed to be a logical consequence (this would exclude the possibility of producing facts that may be true in the intended interpretation but are not strictly entailed).

A reasoning process is logically complete if it guarantees to produce α, whenever α is entailed (this would exclude the possibility of missing some entailments, for instance when their status is too difficult to determine).

If KB is a finite set of sentences $\{\alpha_1,\ldots,\alpha_n\}$, the deductive reasoning can be formulated in several equivalent ways:

1. $KB \models \alpha$
2. $\models [(\alpha_1 \wedge \ldots \wedge \alpha_n) \supset \alpha]$
3. $KB \cup \{\neg\alpha\}$ is not satisfiable
4. $KB \cup \{\neg\alpha\} \models \neg TRUE$

where TRUE is any valid sentence (for example, $\forall x.x=x$).

Resolution

4→3

If there is $\mathcal{I}$ so that $\mathcal{I} \models$ KB U {¬α}

then KB U {¬α} $\models$ TRUE in $\mathcal{I}$ -- contradiction with 4

If we have a procedure for testing the validity of sentences, or for testing the satisfiability of sentences, or for determining whether or not ¬TRUE is entailed, then that procedure can also be used to find entailments of a finite KB.

The propositional case of resolution

The propositional logic is a restricted form of formulas. Every formula α of propositional logic can be transformed into α´, a conjunction of disjunctions of literals (i.e. atoms or their negation), such that ⊨ (α≡α´)

α´ is in conjunctive normal form CNF

Example (p∨q∨¬r)∧(p∨¬s∨¬q)∧(¬q∨r)

Obs. Lowercase letters are used for propositional symbols to be consistent with common practice.

The propositional case of resolution

The procedure for conversion of any propositional formula to CNF:

1. Replace ⊃ and ≡ with the formulas they represent

2. Move ¬ inward so that it appears in front of an atom
   - ⊨ (¬¬α≡α)
   - ⊨ ¬(α∧β) ≡¬α∨¬β
   - ⊨ ¬(α∨β) ≡¬α∧¬β

3. Distribute ∧ over ∨
   - ⊨ (α∨(β∧γ)) ≡((β∧γ)∨α)≡(α∨β)∧(α∨γ))

4. Collect terms
   - ⊨ (α∨α≡α)
   - ⊨ (α∧α≡α)

The propositional case of resolution

Obs. The result is a logically equivalent CNF formula which can be exponentially larger than the initial formula.

$$((p \supset q) \equiv r) \longrightarrow (\neg(\neg p \vee q) \vee r) \wedge (\neg r \vee (\neg p \vee q))$$

$$((p \wedge \neg q) \vee r) \wedge (\neg r \vee \neg p \vee q)$$

$$(p \vee r) \wedge (\neg q \vee r) \wedge (\neg r \vee \neg p \vee q) \quad \text{CNF}$$

We will write CNF using a shorthand representation.

The propositional case of resolution

A clause is a finite set of literals (understood as a disjunction of its literals).

A clausal formula is a finite set of clauses (understood as a conjunction of its clauses).

Notations:

- $\bar{\rho}$ is the complement of the literal ρ
- $\bar{\rho} \stackrel{\text{def}}{=} \neg\rho$ and $\neg\bar{\rho}=\rho$
- {set of clausal formulas}
- [set of literals]

For example, [p,¬q,r] represents p∨¬q∨r

{[p,¬q,r] ,[q]} represents (p∨¬q∨r)∧q

A clause with a single literal is called a unit clause

[p], [¬q]

The propositional case of resolution

Obs. {}≠{[]}

{} – the empty clausal formula=conjunction of no constraints is a representation of TRUE

[] – disjunction of no possibilities – is a representation of ¬TRUE

{[]} stands for ¬TRUE

In order to determine whether or not KB ⊨ α, it is sufficient to do the following:

1. Convert the sentences in KB and ¬α into CNF;
2. Determine whether or not the resulting set of clauses is satisfiable.

Resolution derivations

The rule of inference called resolution is the following:

Given a clause C_1 U {ρ}, where ρ is a literal, and a clause C_2 U {$\bar{\rho}$}, then C_1 U C_2 is inferred (C_1 or C_2 may be empty).

We say that C_1 U C_2 is a resolvent of the two input cluses with respect to ρ.

For example, [p,q,r] and [q,¬p,s]

[q,r,s] is a resolvent with respect to p.

[p,q] and [¬p, ¬q] have two resolvents:

[q, ¬q] with respect to p

[p, ¬p] with respect to q

Obs. The only way to get [] is by resolving two complementary unit clauses like [p] and [¬p].

Resolution derivations

Def. A resolution derivation of a clause c from a set of clauses S is a sequence of clauses $c_1,\ldots,c_n$, where $c_n=c$ and each c_i is either an element of S or a resolvent of two prior clauses in the derivation.

We write $S \vdash c$ if there is a derivation of c from S.

The resolution derivations are important because these symbol-level operation on finite sets of literals is directly connected to knowledge-level logical interpretations.

Resolution derivations

Prop. The resolvent is always the logical consequence of the two input clauses

$$\{C_1 \cup \{p\}, C_2 \cup \{\neg p\}\} \models C_1 \cup C_2$$

Let $\mathcal{I}$ be an interpretation so that $\mathcal{I} \models C_1 \cup \{p\}$ and $\mathcal{I} \models C_2 \cup \{\neg p\}$

If $\mathcal{I} \models p$ then $\mathcal{I} \not\models \neg p$

but $\mathcal{I} \models C_2 \cup \{\neg p\}$, it follows that $\mathcal{I} \models C_2$, therefore $\mathcal{I} \models C_1 \cup C_2$

If $\mathcal{I} \not\models p$ but $\mathcal{I} \models C_1 \cup \{p\}$, it follows that $\mathcal{I} \models C_1$, therefore $\mathcal{I} \models C_1 \cup C_2$

Resolution derivations

Prop. Any clause derivable by resolution from S is logically entailed by S, that is if $S \vdash c$ then $S \models c$

Proof - by induction on the length of the derivation, we show that

for every c_i it follows that $S \models c_i$

$S \vdash c$ if $\exists c_1,\ldots,c_n=c$ so that either $c_i \in S$ or c_i is the resolvent of two prior clauses in the derivation.

If $c_i \in S$ then $S \models c_i$

If c_i is the resolvent of c_j and c_k, then $\{c_j,c_k\} \models c_i$

From the induction hypothesis, $S \models c_j$ and $S \models c_k$ => $S \models c_i$

Resolution derivations

Prop. Any clause derivable by resolution from S is logically entailed by S, that is if $S \vdash c$ then $S \models c$

Proof - by induction on the length of the derivation, we show that

for every c_i it follows that $S \models c_i$

$S \vdash c$ if $\exists c_1,\ldots,c_n=c$ so that either $c_i \in S$ or c_i is the resolvent of two prior clauses in the derivation.

If $c_i \in S$ then $S \models c_i$

If c_i is the resolvent of c_j and c_k, then $\{c_j,c_k\} \models c_i$

From the induction hypothesis, $S \models c_j$ and $S \models c_k$ => $S \models c_i$

The converse does not hold – we can have $S \not\models c$ without $S \vdash c$.

For example, S={[¬p]} and c=[¬q,q]

$S \models c$ but $c \notin S$ and there are no resolvents, therefore $S \not\vdash c$.

Resolution derivations

Obs. The resolution derivations are not logically complete (do not guarantee to produce α whenever $S \models \alpha$).

Obs. But resolution is both sound and complete when c=[]

$S \vdash []$ iff $S \models []$ (S is unsatisfiable)

Thus, the problem of determining the satisfiability of any set of clauses is reduced to the search for a derivation of the empty clause.

The entailment procedure

We want to determine whether or not KB ⊨ α (equivalent to KB U {¬α} is unsatisfiable).

Let S be the set of clauses obtained by converting KB U {¬α} in CNF.

We check if S is unsatisfiable by searching for a derivation of the empty clause.

The nondeterministic procedure:

Input: a finite set S of propositional clauses

1. If ([]∈S) then return unsatisfiable
   else if (there are two clauses in S that can resolve to produce a new clause (not already in S) then (add the new resolvent clause to S and go to Step 1)
   else return satisfiable

Output: satisfiable or unsatisfiable

The entailment procedure

Obs. The procedure terminates because each added clause is a resolvent of previous clauses and contains only literals from the initial set of clauses S (a finite number) – eventually nothing new can be added.

Obs. The procedure can be made deterministic – we set a strategy for choosing the pair of clauses to produce a new resolvent – e.g. the first pair encountered; the pair that produces the shortest resolvent.

If we are interested in returning the derivation, for each resolvent we should store pointers to its input clauses.

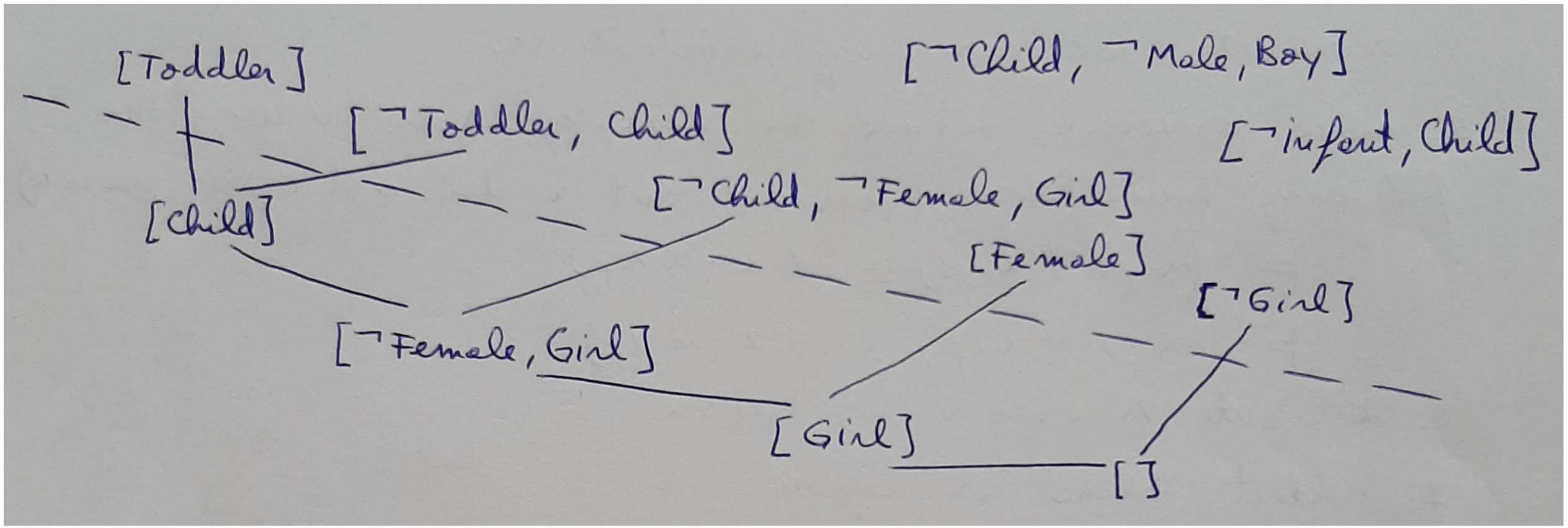
Example 1

KB
- Toddler
- Toddler⊃Child
- Child∧Male⊃Boy
- Infant⊃Child
- Child∧Female⊃Girl
- Female

Question: Girl

KB ⊨ Girl iff KB U {¬Girl} is unsatisfiable

Example 1


[Toddler]
[¬Child, ¬Male, Boy]
[¬Toddler, Child]
[¬Infant, Child]
[Child]
[¬Child, ¬Female, Girl]
[Female]
[¬Girl]
[¬Female, Girl]
[Girl]
[]

Example 2

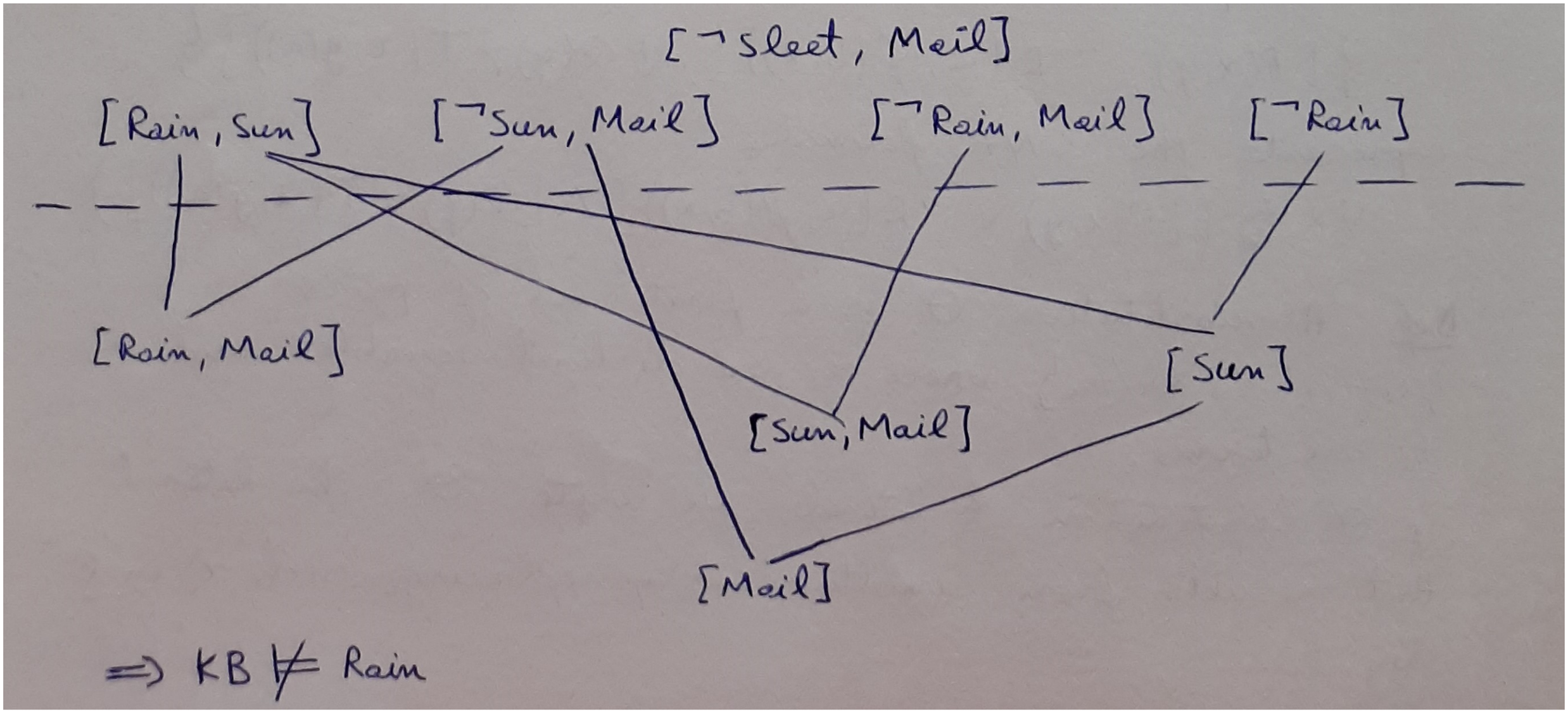
KB ⎡ Sun⊃Mail
   ⎢ (Rain∨Sleet)⊃Mail
   ⎣ Rain∨Sun

Question 1: Rain

Question 2: Mail

KB ⊨ Rain iff KB U {¬Rain} is unsatisfiable

Example 2


[¬sleet, Mail]
[Rain, Sun]
[¬Sun, Mail]
[¬Rain, Mail]
[¬Rain]
[Rain, Mail]
[Sun]
[Sun, Mail]
[Mail]
⇒ KB ⊭ Rain

Handling variables and quantifiers

We transform formulas into an equivalent clausal form:

1. Replace ⊃ and ≡ with the formulas they represent

2. Move ¬ inward so that it appears in front of an atom

   ⊨ (¬¬α≡α)
   ⊨ ¬(α∧β)≡¬α∨¬β
   ⊨ ¬(α∨β)≡¬α∧¬β

   ⊨ ¬∀x.α≡∃x.¬α
   ⊨ ¬∃x.α≡∀x.¬α

3. Rename variables (if necessary) so that the variables in two input clauses of resolution are distinct

Handling variables and quantifiers

4. Move universals outside the scope of ∧ and ∨ using the following equivalences (provided that x does not occur free in α)

$\models (\alpha \wedge \forall x.\beta) \equiv \forall x.(\alpha \wedge \beta)$

$\models (\alpha \vee \forall x.\beta) \equiv \forall x.(\alpha \vee \beta)$

5. Distribute ∧ over ∨

$\models (\alpha \vee (\beta \wedge \gamma)) \equiv ((\beta \wedge \gamma) \vee \alpha) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

6. Collect terms

$\models (\alpha \vee \alpha \equiv \alpha)$

$\models (\alpha \wedge \alpha \equiv \alpha)$

Handling variables and quantifiers

For the beginning, we consider the case where no existentials appear. We drop the quantifiers (they are all universals).

For example, the clausal formula

$$\{[P(x,y),\neg R(a,f(b,x))],[Q(y),T(x,g(a)]\}$$

represents the CNF formula

$$\forall x \forall y.([P(x,y) \vee \neg R(a,f(b,x))] \wedge [Q(y) \vee T(x,g(a)])$$

Def. A substitution θ is a finite set of pairs $\{x_1/t_1,\ldots,x_n/t_n\}$ where x_i are distinct variables and t_i are terms.

If θ is a substitution, ρ literal, we write by ρθ the literal that results from simultaneously replacing each x_i in ρ by t_i.

Handling variables and quantifiers

For example,

$\theta = \{x/f(a,y), y/g(x,z)\}$

$\rho = P(h(x,b,y),z)$

$\rho\theta = P(h(f(a,y),b,g(x,z)),z)$

If c is a clause, $c\theta$ is the clause resulting from making substitution on each literal.

We say that a term, literal or clause is ground if it contains no variables.

We say that ρ is an instance of ρ' if there is θ so that $\rho = \rho'\theta$.

First-order resolution

Since the clauses with variables are universally quantified, we want to allow resolution to be applied to any of their instances.

For example

[P(x,f(a))] and [¬P(g(b,z),y),¬Q(z,f(b))]

x/g(b,z) | y/f(a)

[P(g(b,z),f(a))] and [¬P(g(b,z),f(a)),¬Q(z,f(b))]

The resolvent is [¬Q(z,f(b))].

First-order resolution

The general rule of resolution is defined as following:

We are given the clauses $C_1 \cup \{\rho_1\}$ and $C_2 \cup \{\bar{\rho}_2\}$, where ρ_1 and ρ_2 are literals.

We rename the variables in the two clauses if necessary, so that each clause has distinct variables.

Suppose that there is a substitution θ such that $\rho_1\theta = \rho_2\theta$.

Then we can infer the clause $(C_1 \cup C_2)\theta$.

We say that θ is a unifier of ρ_1 and ρ_2.

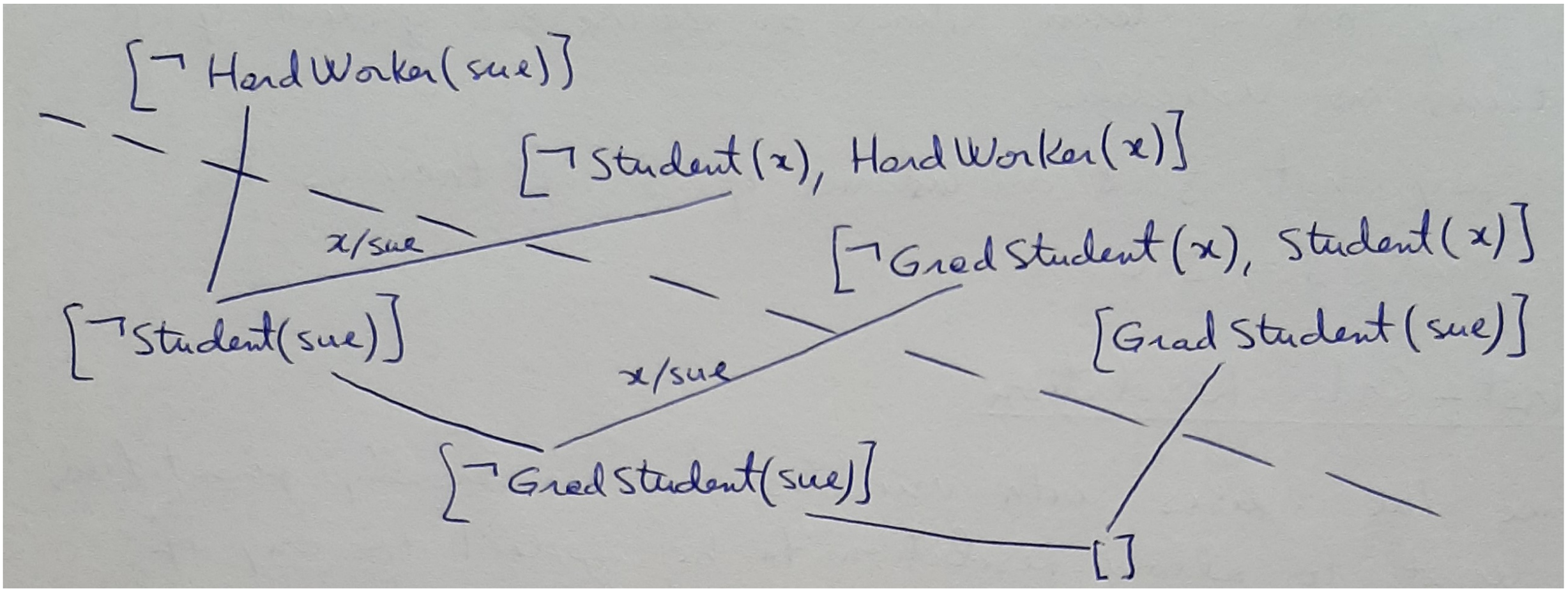
With this general rule of resolution, it is the case that $S \vdash []$ iff $S \models []$.

Example 3

KB
- ∀x.GradStudent(x)⊃Student(x)
- ∀x.Student(x)⊃HardWorker(x)
- GradStudent(sue)

KB ⊨ HardWorker(sue)

Example 3


[¬ HardWorker(sue)]
[¬ Student(x), HardWorker(x)]
x/sue
[¬ GradStudent(x), Student(x)]
[¬ Student(sue)]
x/sue
[GradStudent(sue)]
[¬ GradStudent(sue)]
[]

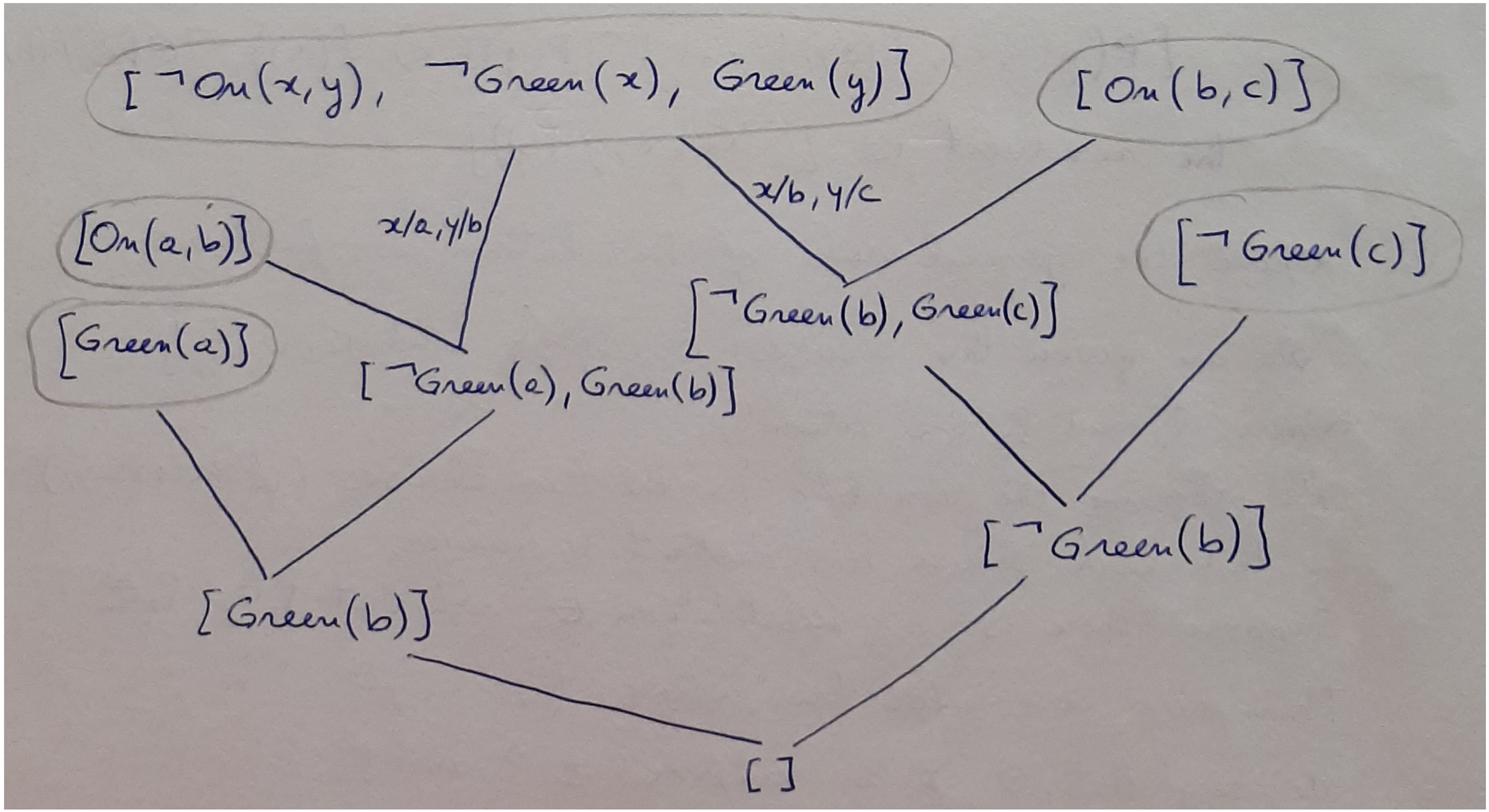
Example 4

The three-block problem

KB: On(a,b), On(b,c), Green(a), ¬Green(c)

Question: ∃x∃y.Green(x)∧¬Green(y)∧On(x,y)

Example 4


[¬On(x,y), ¬Green(x), Green(y)]
[On(b,c)]
x/a, y/b
x/b, y/c
[On(a,b)]
[¬Green(c)]
[Green(a)]
[¬Green(b), Green(c)]
[¬Green(a), Green(b)]
[¬Green(b)]
[Green(b)]
[]

Example 5 – The necessity of renaming variables

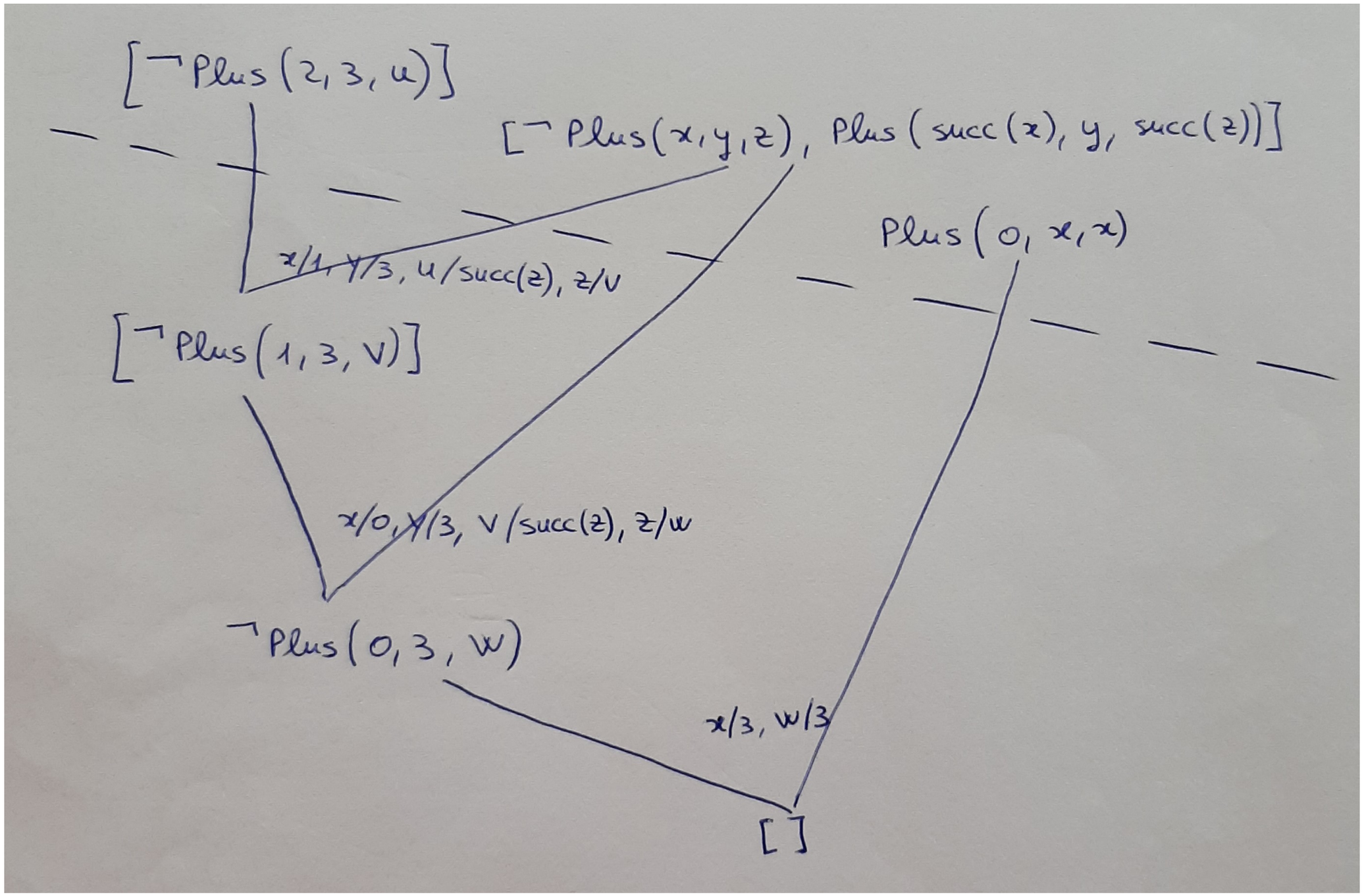
KB
- ∀x.Plus(zero,x,x)
- ∀x∀y∀z.Plus(x,y,z))⊃Plus(succ(x),y,succ(z))

Question: ∃u.Plus(2,3,u)

Plus(x,y,z) represents x+y=z

succ(succ(succ(zero))) represents 3

Example 5 – The necessity of renaming variables



We can identify the value of u:

u is bound to succ(v); v is bound to succ(w); w is bound to 3 =>u=5

To Do

KB
1. Anybody who passes some exam studies or is brilliant or is lucky.
2. Anybody who takes a 5 passes some exam.
3. No FMI student is lucky.
4. Anyone who drinks beer does not study.

5. (Question) Every FMI student who takes a 5 and drinks beer is brilliant.

a) Represent 1.-4. in FOL, using a vocabulary that you will define.
b) Prove that 5. is logically entailed from 1.-4., by applying Resolution.