# 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 $\alpha$ is a sentence.

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

A reasoning process is logically sound if whenever it produces $\alpha$, then $\alpha$ 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 $\alpha$, whenever $\alpha$ 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 $\{\alpha_1, \dots, \alpha_n\}$, the deductive reasoning can be formulated in several equivalent ways:

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

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

$4 \Rightarrow 3$

if there is $\mathcal{I}$ so that $\mathcal{I} \models KB \cup \{\neg\alpha\}$ then $KB \cup \{\neg\alpha\} \models TRUE$ in $\mathcal{I}$ - contradiction with $KB \cup \{\neg\alpha\} \models \neg TRUE$

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

The propositional case of Resolution

The propositional logic is a restricted form of formulas.

Every formula $\alpha$ of propositional logic can be transformed into $\alpha'$, a conjunction of disjunctions of literals (i.e. atoms or its negation), such that $\models (\alpha \equiv \alpha')$

$\alpha'$ is in conjunctive normal form CNF

Example: $(p \vee q \vee \neg r) \wedge (p \vee \neg s \vee \neg q) \wedge (\neg q \vee r)$

Att: lowercase letters are used for propositional symbols to be consistent with common practice

The procedure for convertion of any propositional formula to CNF:

1. replace $\supset, \equiv$ with the formulas they represent
2. move $\neg$ inward so that it appears in front of an atom

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

3. distribute $\wedge$ over $\vee$

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

4. collect terms

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

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.

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 $\rho$

$\bar{\rho} \overset{def}{=} \neg \rho$ and $\neg \bar{\rho} = \rho$

{set of clausal formulas}

[set of literals]

For example, $[p, \neg q, r]$ represents $(p \vee \neg q \vee r)$

$\{[p, \neg q, r], [q]\}$ represents $(p \vee \neg q \vee r) \wedge q$

A clause with a single literal is called a unit clause

$[p]$, $[\neg q]$

Obs. $\{\ \} \neq \{[\ ]\}$

$\{\ \}$ - the empty clausal formula = conjunction of no constraints is a representation of TRUE

$[\ ]$ - disjunction of no possibilities - is a representation of $\neg$TRUE

$\{[\ ]\}$ stands for $\neg$TRUE

In order to determine whether or not $KB \models \alpha$, it is sufficient to do the following:

1. convert the sentences in KB and $\neg\alpha$ 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 $c_1 \cup \{\rho\}$ where $\rho$ is a literal, and a clause $c_2 \cup \{\bar{\rho}\}$, then $c_1 \cup c_2$ is inferred ($c_1$ and $c_2$ may be empty).

We say that $c_1 \cup c_2$ is a resolvent of the two input clauses with respect to $\rho$.

For example $[\underline{p}, q, r]$ and $[q, \underline{\neg p}, s]$

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

$[p, q]$ and $[\neg p, \neg q]$ have two resolvents:

$[q, \neg q]$ with respect to $p$

$[p, \neg p]$ with respect to $q$

<u>Obs</u>. The only way to get $[\,]$ is by resolving two complementary unit clauses like $[p]$ and $[\neg p]$.

<u>Def</u>. 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.

<u>Obs</u>. 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 $\mathfrak{I}$ be an interpretation so that $\mathfrak{I} \models c_1 \cup \{p\}$ and $\mathfrak{I} \models c_2 \cup \{\neg p\}$

1. if $\mathcal{I} \models p$ then $\mathcal{I} \not\models \neg p$
but $\mathcal{I} \models c_2 \cup \{\neg p\}$ $\Big\} \Rightarrow \mathcal{I} \models c_2 \Rightarrow \mathcal{I} \models c_1 \cup c_2$

2. if $\mathcal{I} \not\models p$ but $\mathcal{I} \models c_1 \cup \{p\} \Rightarrow \mathcal{I} \models c_1 \Rightarrow \mathcal{I} \models c_1 \cup c_2$

<u>Obs</u>. 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 $\exists\, c_1, \dots, c_n = c$ so that either $c_i \in S$ or $c_i$ is the resolvent of two earlier clauses in the derivation.

if $c_i \in S$ then $S \models c_i$

if $c_i$ is a resolvent of $c_j$ and $c_k \Rightarrow \{c_j, c_k\} \models c_i$
from the induction hypothesis $S \models c_j$, $S \models c_k$ $\Big\} \Rightarrow$

$S \models c_i$.

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

For example, $S = \{[\neg p]\}$ and $c = [\neg q, q]$

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

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

<u>Obs</u>. But Resolution is both sound and complete when $c = [\,]$.

$$S \vdash [\,] \text{ iff } S \models [\,] \quad (S \text{ is unsatisfiable})$$

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 $KB \models \alpha$ (equivalent to $KB \cup \{\neg\alpha\}$ is unsatisfiable).
Let $S$ be the set of clauses obtained by converting $KB \cup \{\neg\alpha\}$ 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 ($[\,] \in S$) then return unsatisfiable
   else if (there are two clauses in $S$ that can resolve to produce another clause not already in $S$)
   then (add the new resolvent clause ~~to~~ to $S$ and go to step 1)
   else return satisfiable

output: satisfiable or unsatisfiable

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.

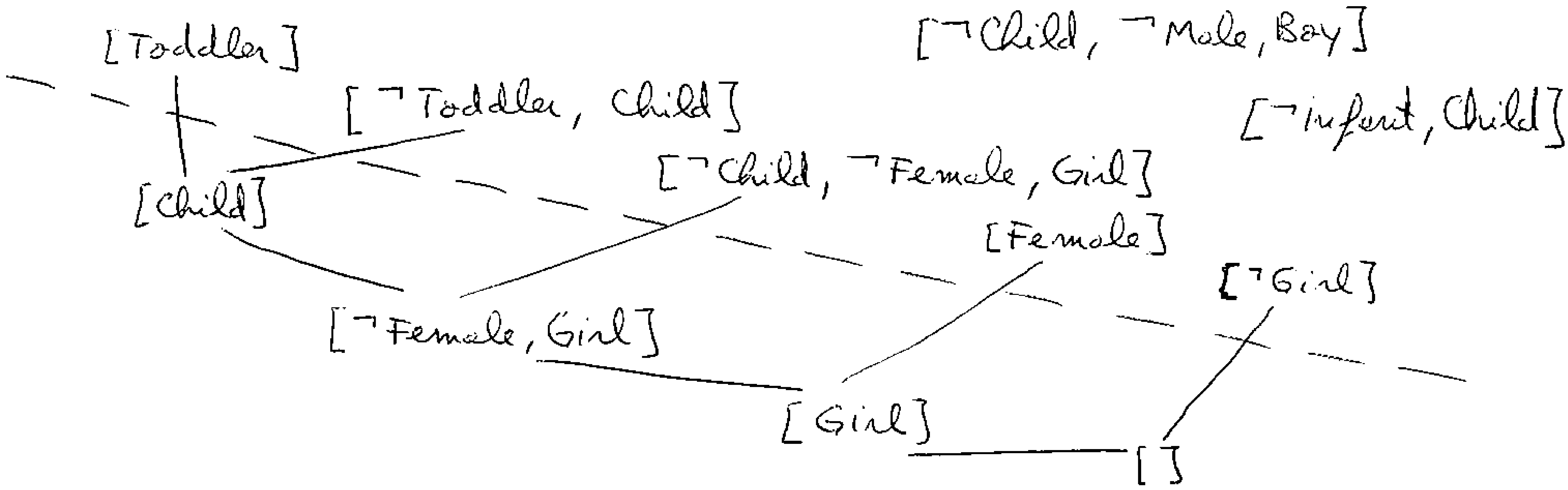
If we are interested in returning the derivation, for each resolvent we should store pointers to its input clauses.

Example 1 We have the following knowledge base

$$KB \begin{cases} Toddler \\ Toddler \supset Child \\ Child \wedge Male \supset Boy \\ infant \supset Child \\ Child \wedge Female \supset Girl \\ Female \end{cases}$$

Question: Girl

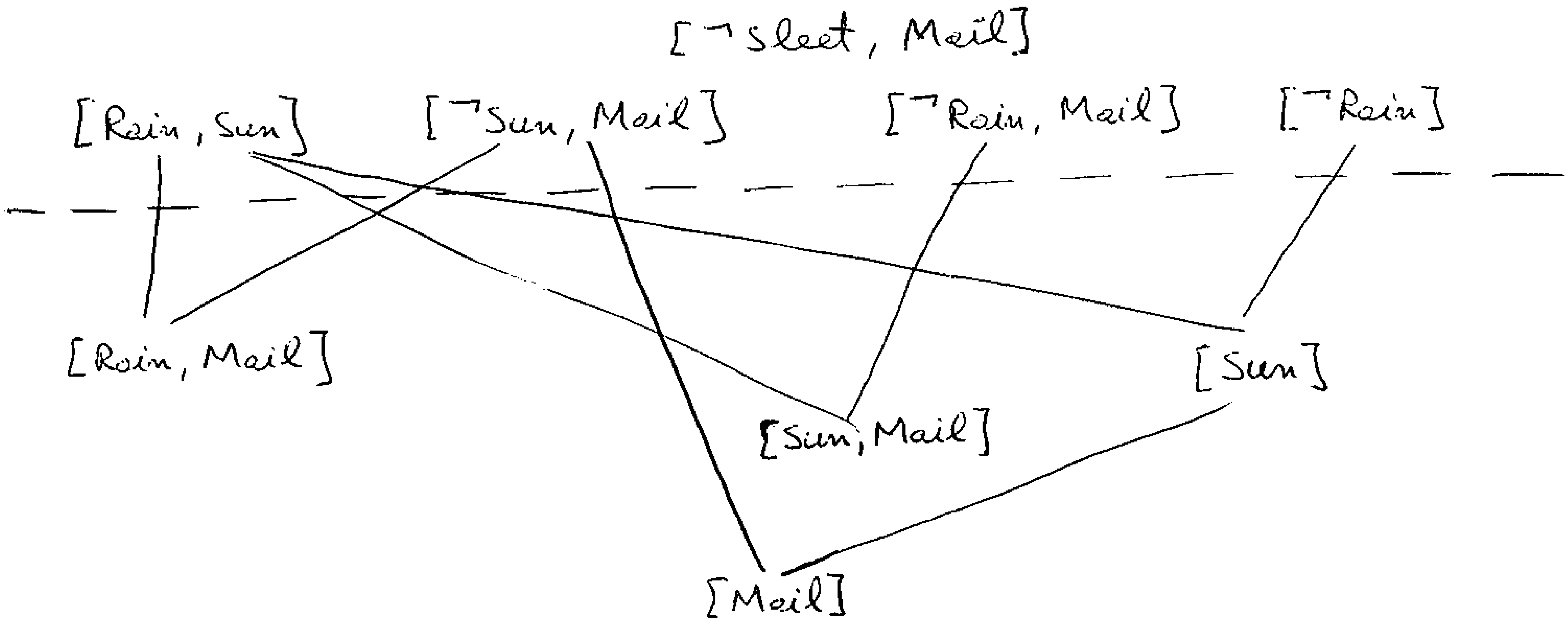
$KB \models Girl$ iff $KB \cup \{\neg Girl\}$ is unsatisfiable




Example 2

$$KB \begin{cases} Sun \supset Mail \\ (Rain \vee Sleet) \supset Mail \\ Rain \vee Sun \end{cases}$$

Question 1: Mail Question 2: Rain




$\Rightarrow KB \not\models Rain$

## Handling variables and quantifiers

We transform formulas into an equivalent clausal form:

1. replace $\supset$ and $\equiv$ as indicated before (page 2)
2. move $\neg$ inward, adding the following two:

$$\models \neg \forall x . \alpha \equiv \exists x . \neg \alpha$$

$$\models \neg \exists x . \alpha \equiv \forall x . \neg \alpha$$

3. rename variables (if necessary) so that the variables in the two input clauses of Resolution are distinct.
4. eliminate all remaining existentials.
5. move universals outside the scope of $\wedge$ and $\vee$ using the following equivalences (provided that $x$ does not occur free in $\alpha$)

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

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

6. distribute $\wedge$ over $\vee$ as before
7. collect terms as before

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

Atoms have the form $P(t_1, \ldots, t_n)$ (we ignore now $t_1 = t_2$).

For example, the clausal formula

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

represents the CNF formula

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

<u>Def</u>. A substitution $\theta$ 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 $\theta$ substitution, $\rho$ literal, we write $\rho\theta$ = the literal that results from simultaneously replacing each $x_i$ in $\rho$ by $t_i$.

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 the substitution on each literal.

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

We say that $\rho$ is an instance of $\rho'$ if there is $\theta$ 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 $[\neg P(g(b,z), y), \neg Q(z, f(b))]$

$$x/g(b,z) \qquad y/f(a)$$

$$[P(g(b,z), f(a))] \text{ and } [\neg P(g(b,z), f(a)), \neg Q(z, f(b))]$$

the resolvent is $[\neg Q(z, f(b))]$

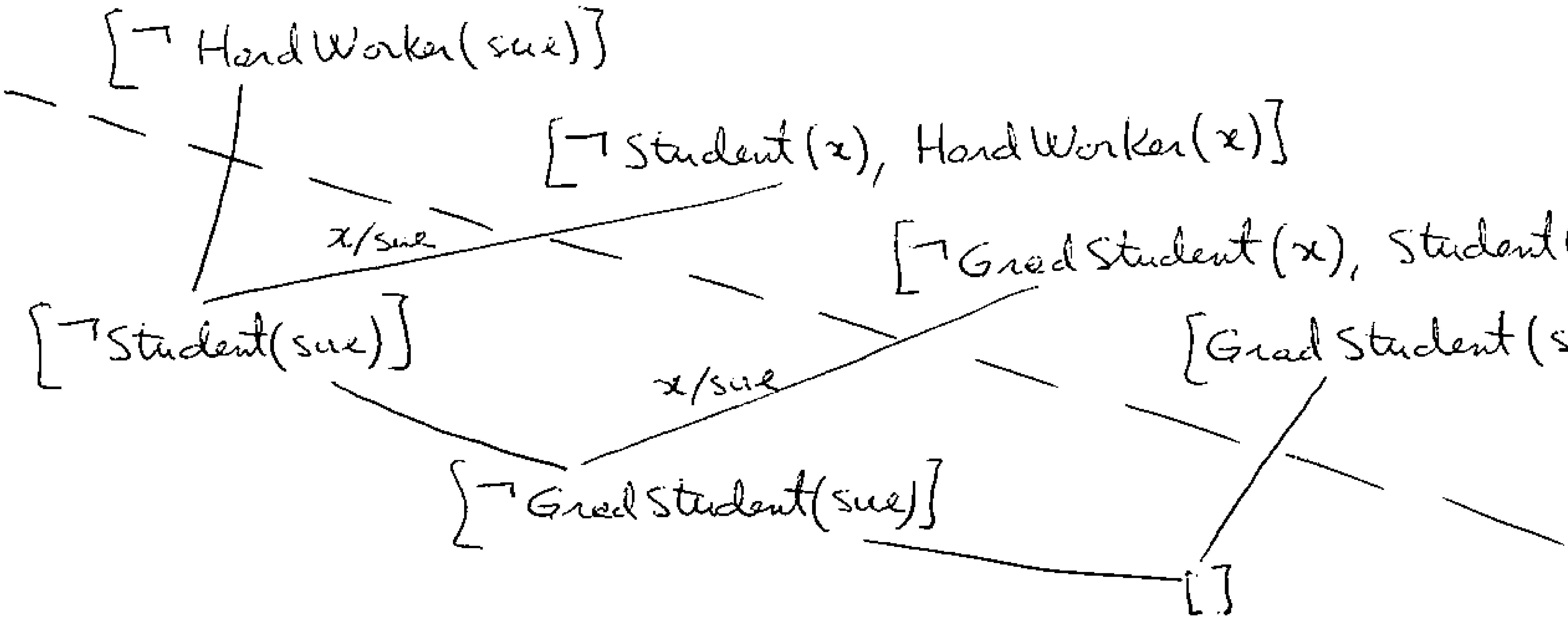
We define the general rule of Resolution as follows:

We are given the clauses $c_1 \cup \{\rho_1\}$ and $c_2 \cup \{\overline{\rho}_2\}$, where $\rho_1$ and $\rho_2$ are literals.
We rename the variables in the two clauses (if necessary) so that each clause has distinct variables.
Suppose there is a substitution $\theta$ such that $\rho_1\theta = \rho_2\theta$.
Then we can infer the clause $(c_1 \cup c_2)\theta$.
We say that $\theta$ is a unifier of $\rho_1$ and $\rho_2$.
With this general rule of Resolution, it is the case that $S \vdash [\,]$ iff $S \models [\,]$.

## Example 3

$$
KB \begin{cases} \forall x.\ GradStudent(x) \supset Student(x) \\ \forall x.\ Student(x) \supset HardWorker(x) \\ GradStudent(sue) \end{cases}
$$

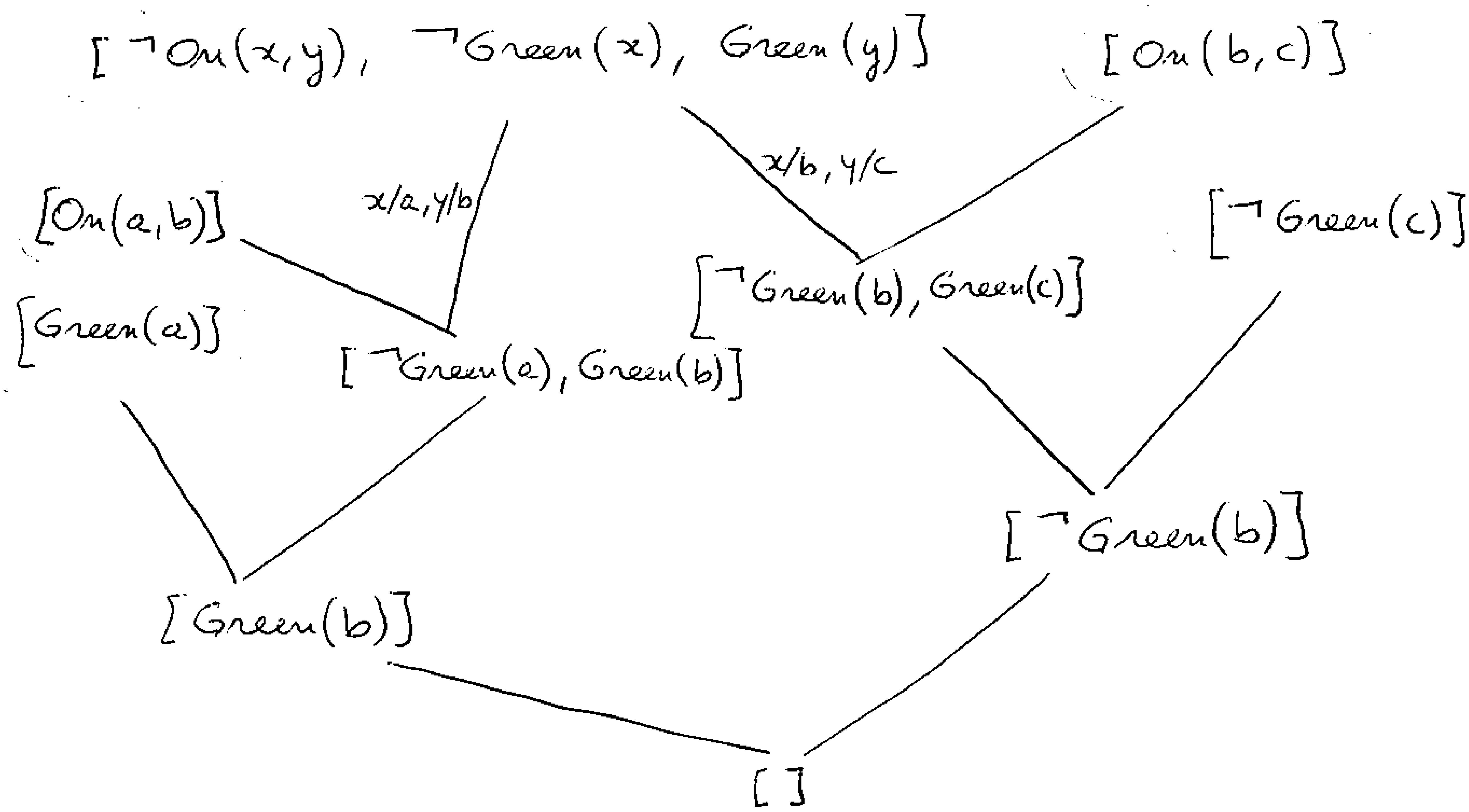
$$KB \models HardWorker(sue)$$




## Example 4 The three-block problem

KB: $On(a,b)$, $On(b,c)$, $Green(a)$, $\neg Green(c)$

Question: $\exists x \exists y.\ On(x,y) \wedge Green(x) \wedge \neg Green(y)$



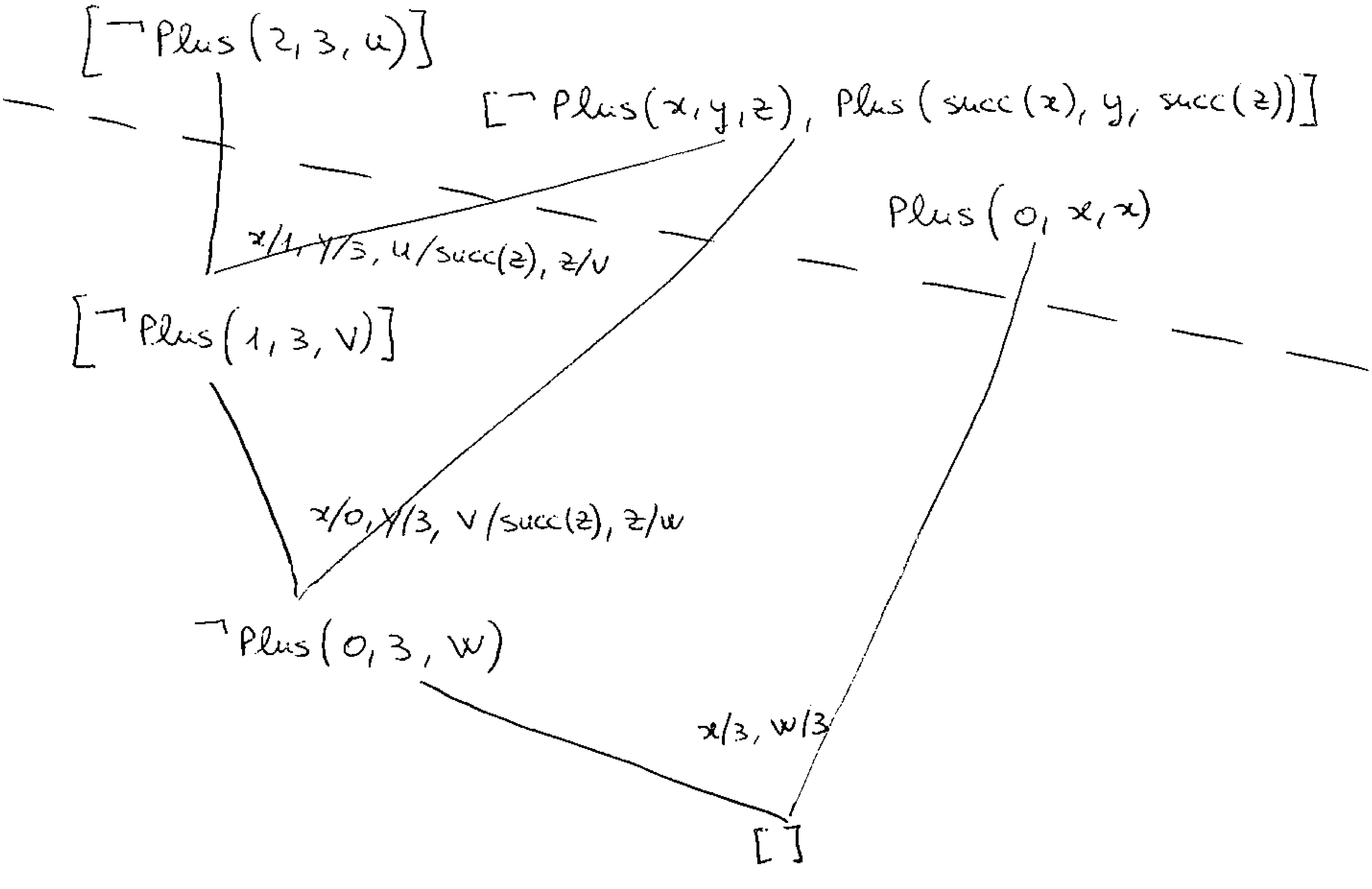
Example 5 - The necessity of renaming variables

KB
$$\begin{cases} \forall x.\ Plus(zero, x, x) \\ \forall x \forall y \forall z.\ Plus(x, y, z) \supset Plus(succ(x), y, succ(z)) \end{cases}$$

Question: $\exists u.\ Plus(2, 3, u)$

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

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




We can identify the value of u:

u is bound to succ(v); v is bound to succ(w);

w is bound to 3 $\Rightarrow$ u = 5