# 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 ⊨ α, 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 ⊨ $\beta[t_1,\ldots,t_n]$.

# 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 $KB \vDash \alpha$, where KB is a given knowledge base and $\alpha$ 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 $KB \vDash \beta[t_1,\ldots,t_n]$.

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

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

# Resolution

4→3 (reductio ad absurdum)

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

then KB U {¬α} $\vDash$ 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.

$$((p \supset q) \equiv r) \rightarrow (\neg(\neg p \lor q) \lor r) \land (\neg r \lor (\neg p \lor q))$$

$$((p \land \neg q) \lor r) \land (\neg r \lor \neg p \lor q)$$

$$(p \lor r) \land (\neg q \lor r) \land (\neg r \lor \neg p \lor q) \quad \text{CNF}$$

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

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

- {set of clauses}

- [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]

**Obs.** If $C_1$ and $C_2$ are two clauses, by $C_1 \cup C_2$ is denoted the clause resulted from the disjunction of the elements of $C_1$ and the elements of $C_2$ .

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

We say that $C_1 \cup C_2$ is a resolvent of the two input cluses with respect to $\rho$.

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

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

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

If $\mathcal{I} \vDash p$ then $\mathcal{I} \nvDash \neg p$

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

If $\mathcal{I} \nvDash p$ but $\mathcal{I} \vDash C_1 \cup \{p\}$, it follows that $\mathcal{I} \vDash C_1$, therefore $\mathcal{I} \vDash C_1 \cup C_2$

# Resolution derivations

Prop. Any clause derivable by resolution from S is logically entailed by S, that is if

$S \vdash c$ then $S \vDash c$

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

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

$S \vdash c$ 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 $S \vDash c_i$

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

From the induction hypothesis, $S \vDash c_j$ and $S \vDash c_k \Rightarrow S \vDash c_i$

# Resolution derivations

Prop. Any clause derivable by resolution from S is logically entailed by S, that is if

$S \vdash c$ then $S \vDash c$

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

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

$S \vdash c$ 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 $S \vDash c_i$

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

From the induction hypothesis, $S \vDash c_j$ and $S \vDash c_k \Rightarrow S \vDash c_i$

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

For example, S={[¬p]} and c=[¬q,q]

$S \vDash c$ but $c \notin S$ and there are no resolvents, therefore $S \nvdash c$.

# Resolution derivations

Obs. The resolution derivations are not logically complete (do not guarantee to produce α whenever S ⊨ α).

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

S ⊢ [ ] iff S ⊨ [ ] (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.
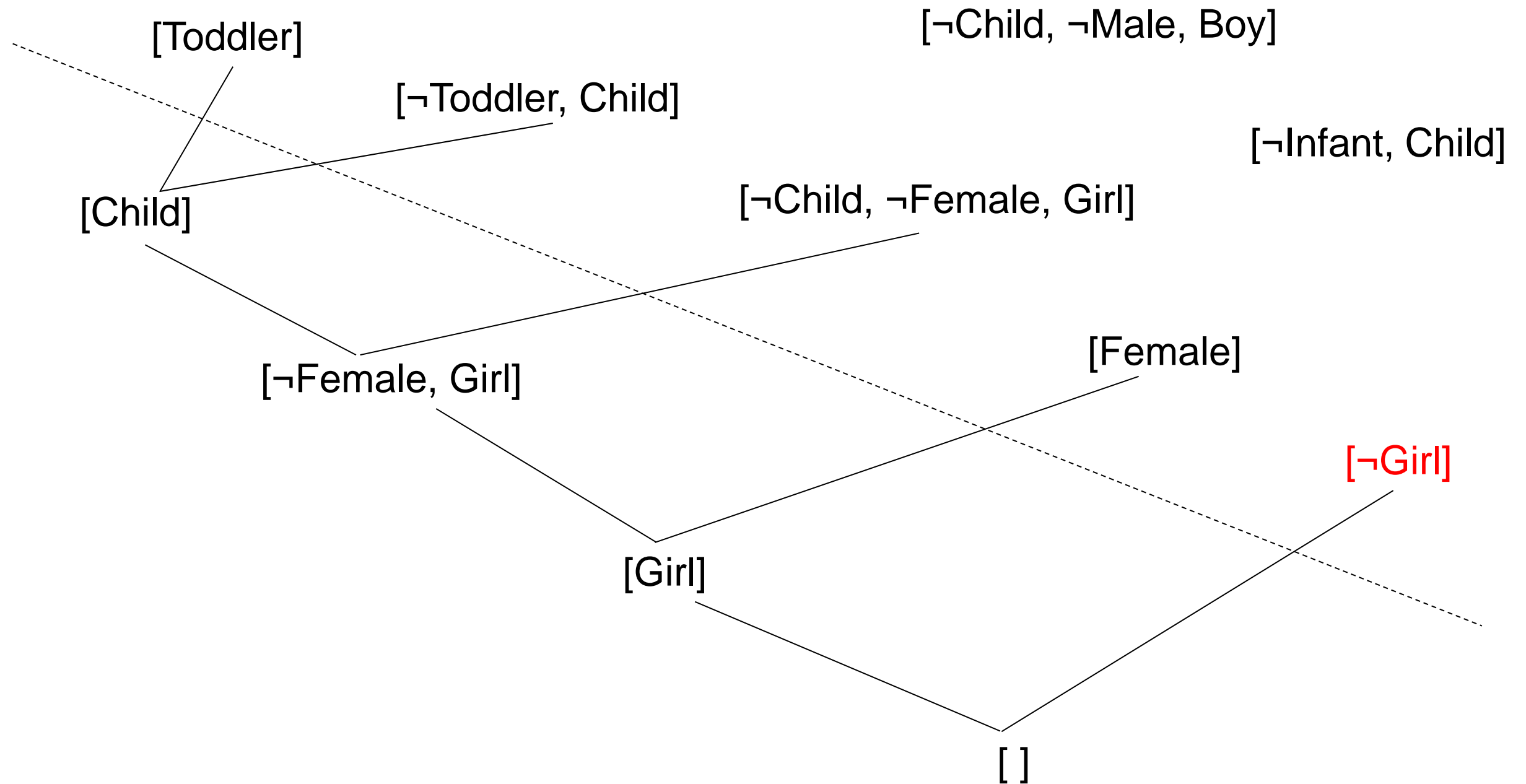
# Example 1

KB
> Toddler
>
> Toddler$\supset$Child
>
> Child$\wedge$Male$\supset$Boy
>
> Infant$\supset$Child
>
> Child$\wedge$Female$\supset$Girl
>
> Female

Question: Girl

KB $\vDash$ Girl iff KB U {¬Girl} is unsatisfiable

# Example 1

**KB + the negated question:**

[Toddler]

[¬Child, ¬Male, Boy]

[¬Toddler, Child]

[¬Infant, Child]

[Child]

[¬Child, ¬Female, Girl]

[¬Female, Girl]

[Female]

[¬Girl]

[Girl]

[ ]

# Example 2

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

Question 1: Rain
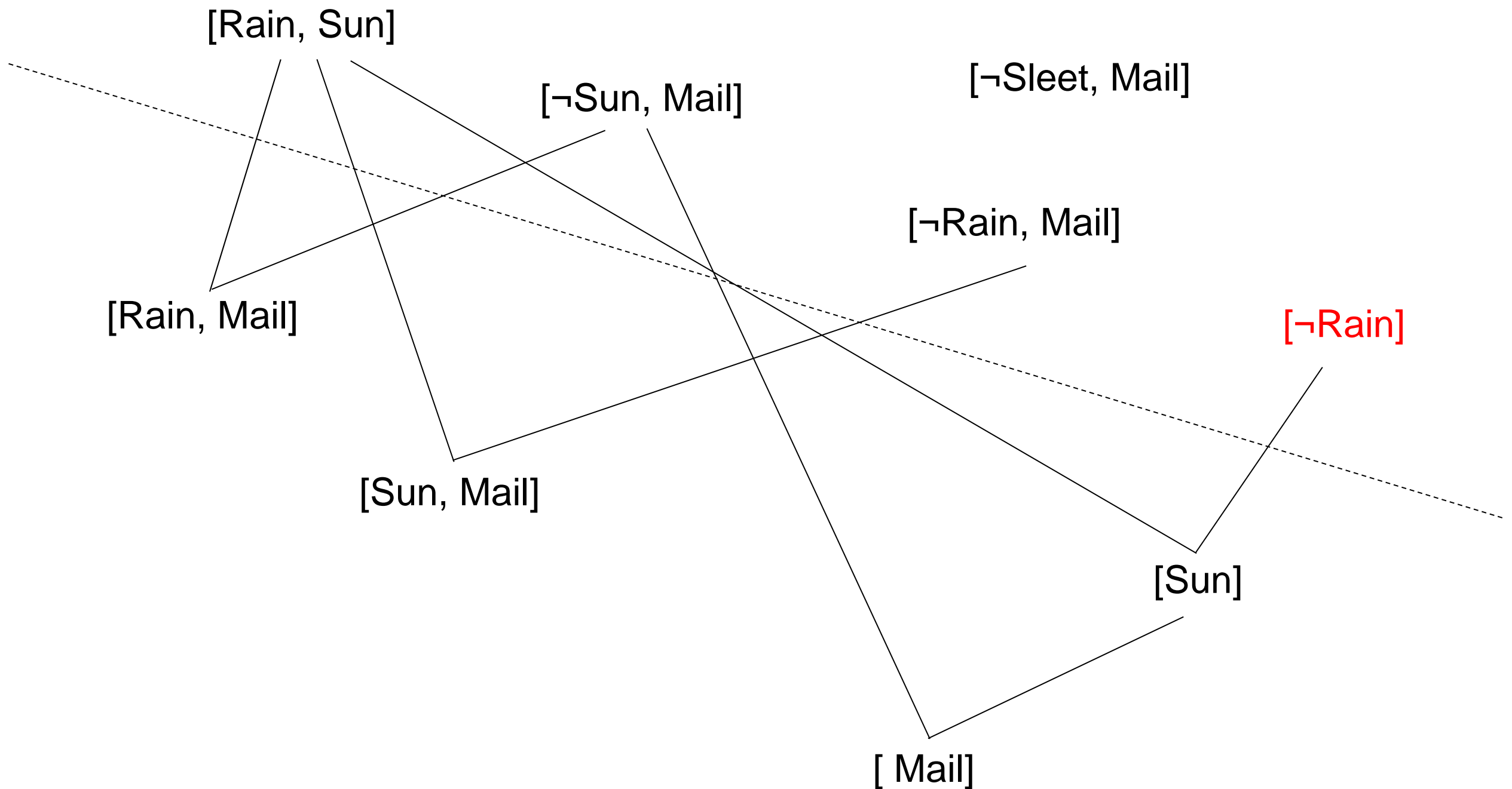
Question 2: Mail

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

# Example 2

**KB:**

[Rain, Sun]

[¬Sleet, Mail]

[¬Sun, Mail]

[¬Rain, Mail]

[Rain, Mail]

[¬Rain]

[Sun, Mail]

[Sun]

[ 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

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

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

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)

$$⊨ (α∧∀x.β)≡∀x.(α∧β)$$
$$⊨ (α∨∀x.β)≡∀x.(α∨β)$$

5. Distribute ∧ over ∨

$$⊨ (α∨(β∧γ))≡((β∧γ)∨α)≡(α∨β)∧(α∨γ))$$

6. Collect terms

$$⊨ (α∨α≡α)$$
$$⊨ (α∧α≡α)$$

# 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 $\theta$ 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 $\theta$ is a substitution, $\rho$ literal, we write by $\rho\theta$ the literal that results from simultaneously replacing each $x_i$ in $\rho$ 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 $\rho$ 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

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

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

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

The resolvent is [¬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 $\rho_1$ and $\rho_2$ are literals.

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

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

Then we can infer the clause $(C_1 \cup C_2)\theta$.
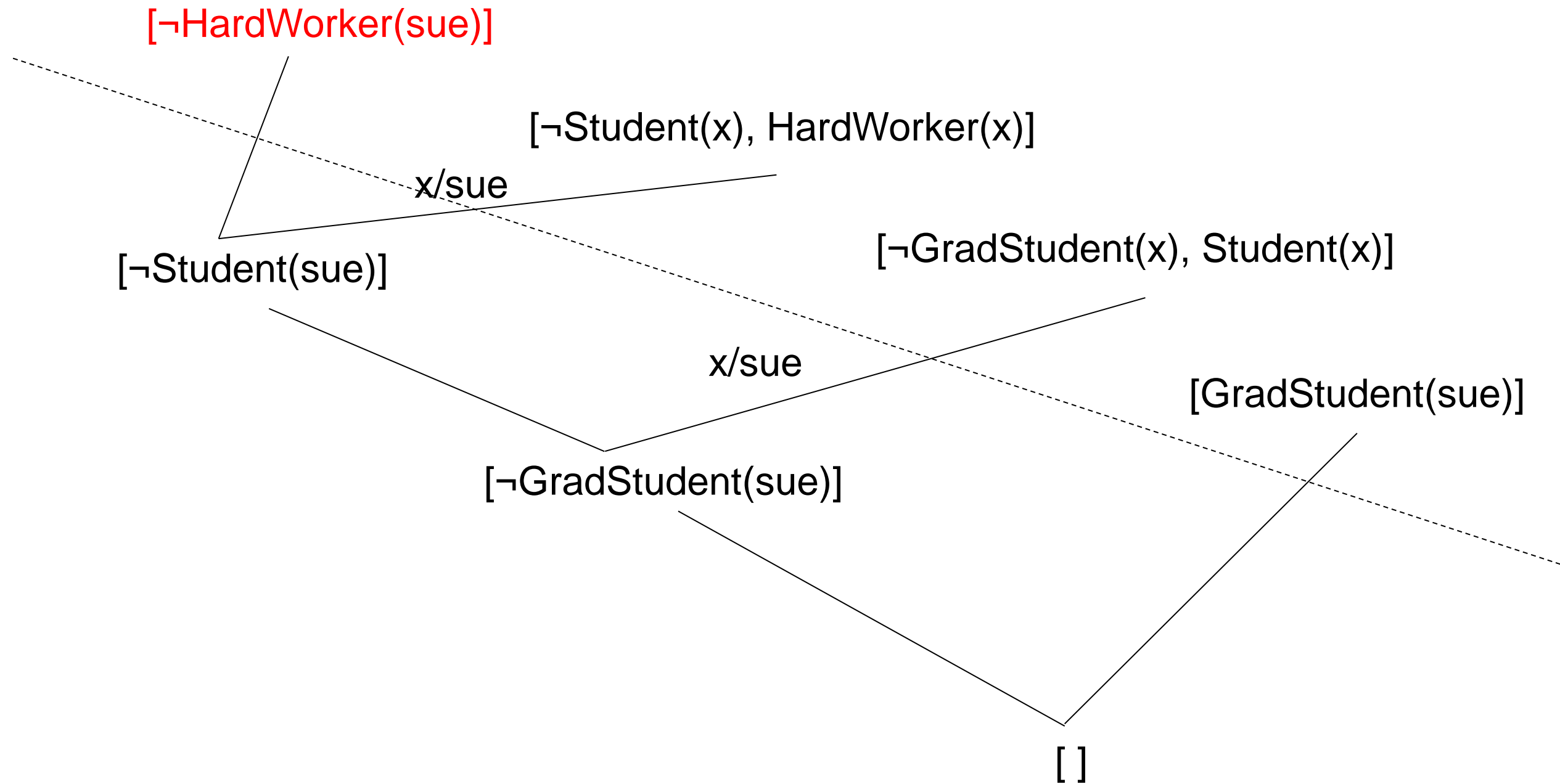
We say that $\theta$ is a unifier of $\rho_1$ and $\rho_2$.

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

# Example 3

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

KB ⊨ HardWorker(sue)

# Example 3

[¬HardWorker(sue)]

[¬Student(x), HardWorker(x)]

x/sue

[¬Student(sue)]

[¬GradStudent(x), Student(x)]

x/sue

[GradStudent(sue)]

[¬GradStudent(sue)]

[ ]

# Example 4

The three-block problem
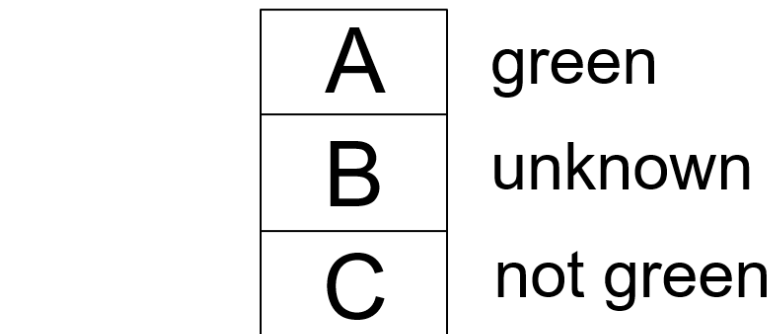


KB: On(a,b), On(b,c), Green(a), ¬Green(c)

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

# Example 4

[¬On(x,y), ¬Green(x), Green(y)]

[On(b,c)]

x/b,y/c

[On(a,b)]

x/a,y/b

[¬Green(c)]

[¬Green(b), Green(c)]

[¬Green(a), Green(b)]

[Green(a)]

[Green(b)]

[¬Green(b)]

[ ]
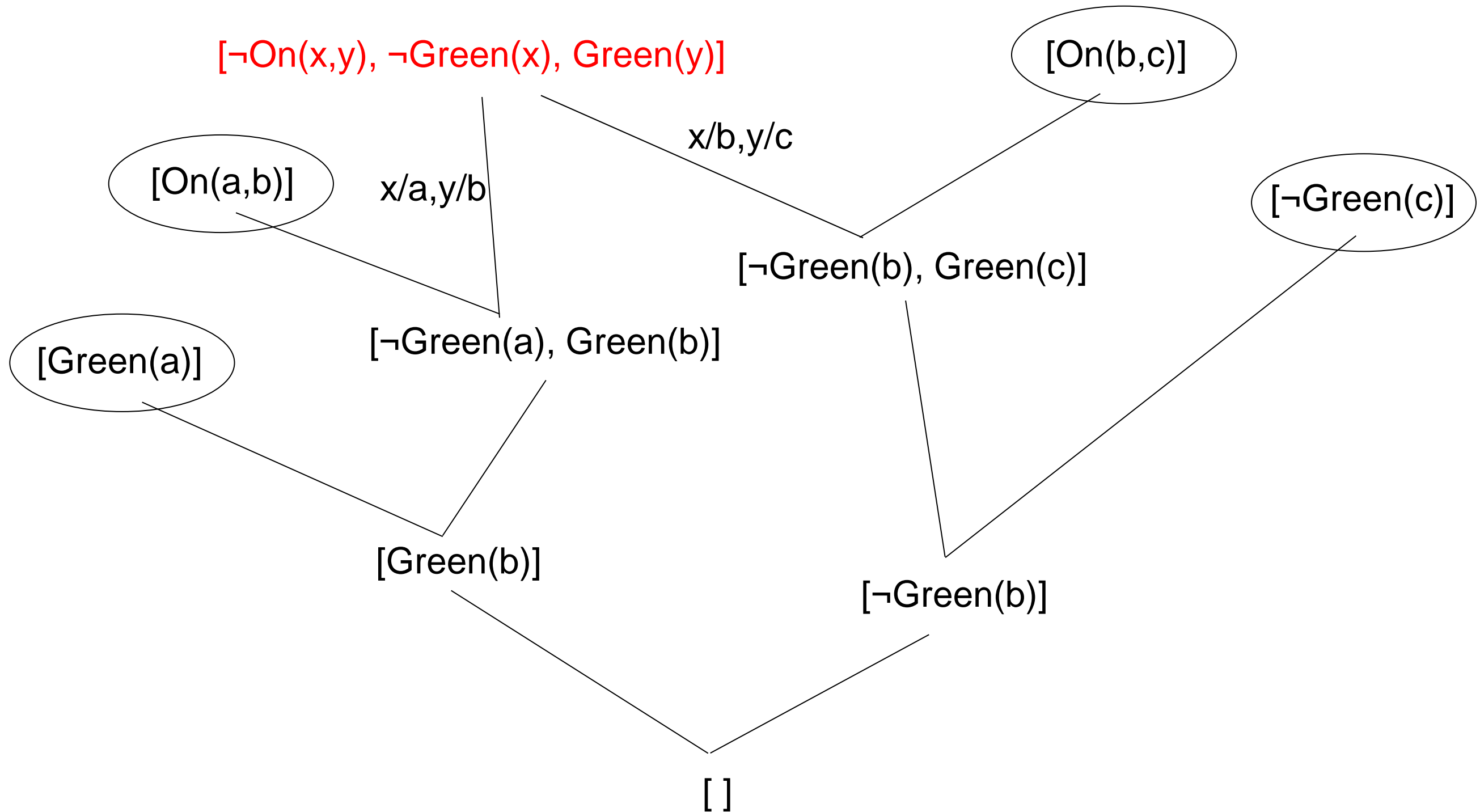
# Example 5 – The necessity of renaming variables

KB
$$\forall x.\text{Plus(zero,x,x)}$$

$$\forall x \forall y \forall z.\text{Plus(x,y,z))} \supset \text{Plus(succ(x),y,succ(z))}$$

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

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

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

# Example 5 – The necessity of renaming variables

[¬Plus(2,3,u)]

[¬Plus(x,y,z), Plus(succ(x),y, succ(z))]

x/1, y/3, u/succ(z), z/v

[Plus(0,x,x)]

[¬Plus(1,3,v)]

x/0, y/3, v/succ(z), z/w

[¬Plus(0,3,w)]

x/3, w/3

[ ]
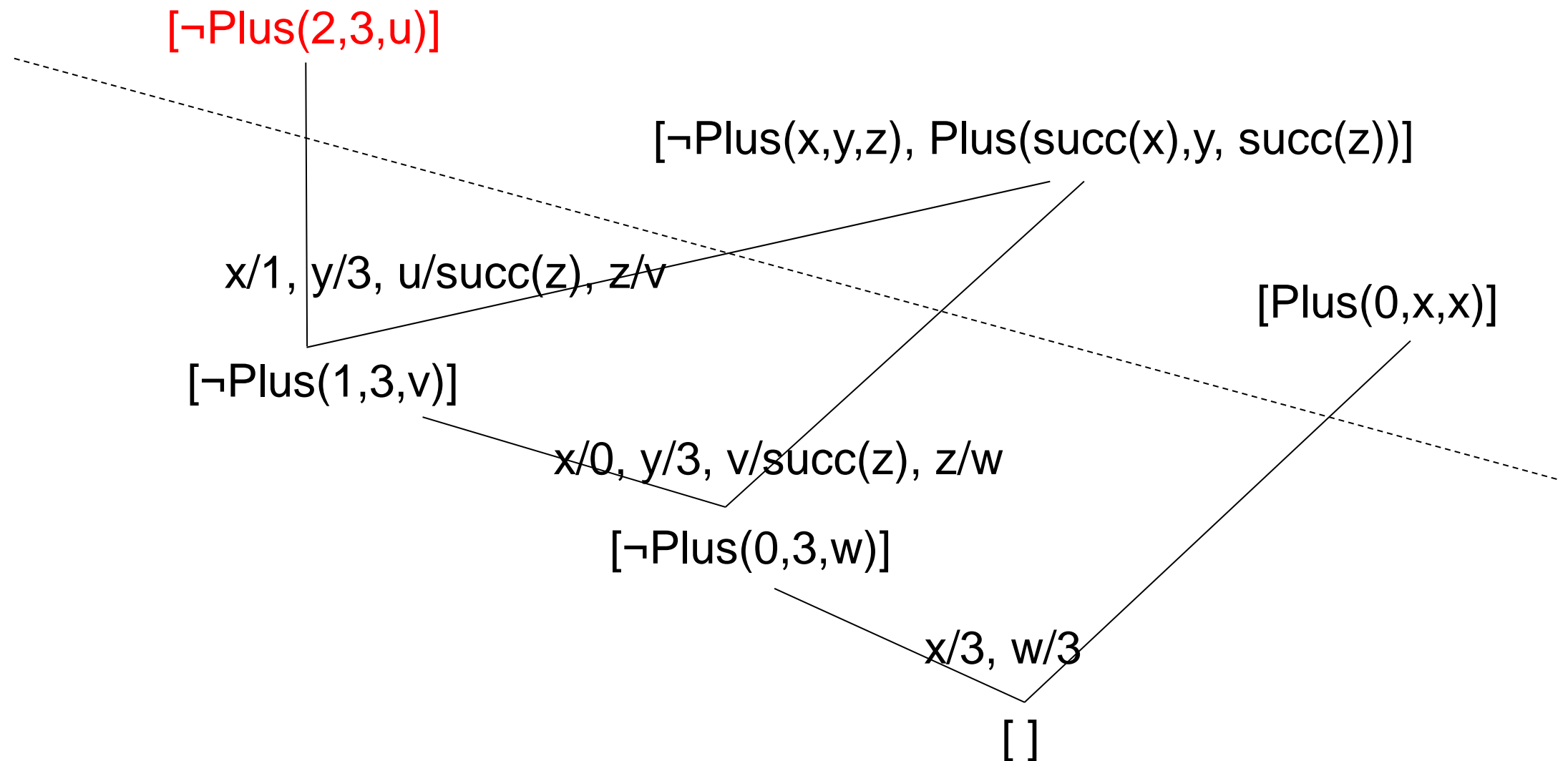
We can identify the value of u:
u is bound to succ(v); v is bound to succ(w); w is bound to 3 ⇒ u=5

# 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.