Knowledge Representation and Reasoning

Lecture 10:
Resolution Principle II

dr Anna Maria Radzikowska
Warsaw University of Technology
Faculty of Mathematics and Information Science
E-mail: annrad@mini.pw.edu.pl
Recall

- $T \vdash \beta$ iff $T \cup \{\neg \beta\}$ is inconsistent.

- A **Literal**: an atom or its negation.

- A **Clause**: $\gamma = \lambda_1 \land \ldots \land \lambda_n$, where $\lambda_i$ are literals.

- **Resolution rule**:

  $\frac{\lambda \lor \alpha, \neg \lambda \lor \beta}{\alpha \lor \beta}$

  where $\lambda$ is a literal and $\alpha, \beta$ are clauses.
What do we need?

In order to show $T \vdash \beta$ using RR we need:

- Transform each sentence $\alpha \in T$ and $\neg \beta$ into the set of clauses.
- What to do with quantifiers?
In order to show $T \vdash \beta$ using RR we need:

- Transform each sentence $\alpha \in T$ and $\neg \beta$ into the set of clauses.
  - What to do with quantifiers?
- Unification of literals.
  - E.g., $Student(x)$ and $Student(Peter)$ are **NOT** the same literals!
What do we need?

In order to show $T \vdash \beta$ using RR we need:

- Transform each sentence $\alpha \in T$ and $\neg \beta$ into the set of clauses.
  What to do with quantifiers?

- Unification of literals.
  E.g., $Student(x)$ and $Student(Peter)$ are **NOT** the same literals!

- Some additional techniques will be also useful.
A (first–order) formula \( \alpha \) is in the **PCNF** iff

\[
\alpha = Q_1x_1 \ Q_2x_2 \ldots Q_mx_m \ \land \ \lor_{\ i=1}^{n} \ \lor_{\ j=1}^{k(i)} \lambda_{ij}
\]

where

- \( Q_1, \ldots, Q_m \) are quantifiers
- \( x_1, \ldots, x_m \) are distinct variables
- \( \lambda_{ij} \) are literals.

**Prefix** – \( Q_1x_1 \ Q_2x_2 \ldots Q_mx_m \)

**Matrix** – \( \land_{i=1}^{n} \lor_{j=1}^{k(i)} \lambda_{ij} \)
A (first–order) formula $\alpha$ is in the PCNF iff

$$\alpha = Q_1 x_1 \ Q_2 x_2 \ \ldots \ Q_m x_m \ (n \ \wedge \ \bigvee_{i=1}^{k(i)} \lambda_{ij})$$

where

- $Q_1, \ldots, Q_m$ are quantifiers
- $x_1, \ldots, x_m$ are distinct variables
- $\lambda_{ij}$ are literals.

**Prefix** – $Q_1 x_1 \ Q_2 x_2 \ \ldots \ Q_m x_m$

**Matrix** – $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k(i)} \lambda_{ij}$

**Theorem 10.1** For every formula $\alpha$ there exists a formula $\alpha'$ in PCNF such that $\alpha \iff \alpha'$. 

Translation to PCNF

**Input:** a formula $\alpha$

**Output:** PCNF of $\alpha$

**ALGORITHM:**

**Step 1.** Eliminate redundant quantifiers.

Eliminate in $\alpha$ any $\forall x$ (resp. $\exists x$), which scope does not contain $x$. Denote the resulting formula by $\alpha_1$.

E.g., $\forall x \exists y. P(x) \rightarrow P(a, x)$

$\exists y$ is redundant
Step 2 Rename variables.

Repeat

- Take the leftmost subformula of $\alpha_1$ of the form $\forall x. \phi(x)$ (resp. $\exists x. \phi(x)$) such that $x$ occurs in other part of $\alpha_1$.
- Replace this subformula by $\forall y. \phi(y)$ (resp. $\exists y. \phi(x)$), where $y$ is a new variable.

until all quantified variables are different and no variable is both bound and free.

Denote the resulting formula by $\alpha_2$.

E.g., $\alpha_1 = (\forall x. A(x) \rightarrow B(x)) \land (\exists x. C(x))$

$\alpha_2 = (\forall y. A(y) \rightarrow B(y)) \land (\exists x. C(x))$. 
**Step 3** Eliminate $\neg$ and $\equiv$:

\[
\begin{align*}
\beta \to \phi & \leadsto \neg \beta \lor \phi \\
\beta \equiv \phi & \leadsto (\neg \beta \lor \phi) \land (\neg \phi \lor \beta)
\end{align*}
\]

Denote the resulting formula by $\alpha_3$.

**Step 4** Move $\neg$ inward:

\[
\begin{align*}
\neg \forall x \beta(x) & \leadsto \exists x \neg \beta(x) \\
\neg \exists x \beta(x) & \leadsto \forall x \neg \beta(x) \\
\neg (\beta \lor \phi) & \leadsto \neg \beta \land \neg \phi \\
\neg (\beta \land \phi) & \leadsto \neg \beta \lor \neg \phi \\
\neg \neg \beta & \leadsto \beta.
\end{align*}
\]

Denote the resulting formula by $\alpha_4$. 
Translation to PCNF (cont.)

**Step 5** Move quantifiers to the left

\[\begin{align*}
\forall x (\beta \land \phi) & \leadsto \forall x (\beta \land \phi) \\
\phi \land (\forall x \beta) & \leadsto \forall x (\phi \land \beta) \\
(\forall x \beta) \lor \phi & \leadsto \forall x (\beta \lor \phi) \\
\phi \lor (\forall x \beta) & \leadsto \forall x (\phi \lor \beta) \\
(\exists x \beta) \land \phi & \leadsto \exists x (\beta \land \phi) \\
\phi \land (\exists x \beta) & \leadsto \exists x (\phi \land \beta) \\
(\exists x \beta) \lor \phi & \leadsto \exists x (\beta \lor \phi) \\
\phi \lor (\exists x \beta) & \leadsto \exists x (\phi \lor \beta).
\end{align*}\]

Denote the resulting formula by \(\alpha_5\).
Step 6 Distribution:

\[(\beta \land \phi) \lor \psi \leadsto (\beta \lor \psi) \land (\phi \lor \psi)\]
\[\beta \lor (\phi \land \psi) \leadsto (\beta \lor \phi) \land (\beta \lor \psi).\]

Denote the resulting formula by \(\alpha_6\).
Step 6  Distribution:

\[(\beta \land \phi) \lor \psi \leadsto (\beta \lor \psi) \land (\phi \lor \psi)\]

\[\beta \lor (\phi \land \psi) \leadsto (\beta \lor \phi) \land (\beta \lor \psi).\]

Denote the resulting formula by \(\alpha_6\).

**Theorem 10.2** The resulting formula \(\alpha_6\) is in PCNF and \(\alpha \iff \alpha_6\).
Example 9.1

Consider a formula $\alpha = P(x, y) \rightarrow (\exists x. Q(x) \land \exists z. R(x, y))$.

$\alpha_1 = P(x, y) \rightarrow (\exists x. Q(x) \land R(x, y))$

$\alpha_2 = P(u, y) \rightarrow (\exists x. Q(x) \land R(x, y))$

$\alpha_3 = \neg P(u, y) \lor (\exists x. Q(x) \land (x, y))$

$\alpha_4 = \alpha_3$

$\alpha_5 = \exists x. (\neg P(u, y) \lor (Q(x) \land R(x, y)))$

$\alpha_6 = \exists x. ((\neg P(u, y) \lor Q(x)) \land (\neg P(u, y) \lor R(x, y)))$. 
Assume that $\alpha$ is in PCNF, i.e., $\alpha = Q_1 x_1 \ldots Q_n x_n \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k(i)} \lambda_{ij}$. Consider the following two cases:

- All $Q_i$ are $\forall$ — $\alpha$ can be identified with the set $CL$ of $m$ clauses:

$$CL = \{ \gamma_1(x), \ldots, \gamma_m(x) \},$$

where $x = (x_1 \ldots x_n)$ and $\gamma_i = \bigvee_{j=1}^{k(i)} \lambda_{ij}(x)$.

- For some $i = 1, \ldots, n$, $Q_i = \exists$. We need a method to eliminate existential quantifiers. This process is called a **Skolemization**.
Skolemization is a procedure of elimination of existential quantifiers. Let $\alpha$ be in PCNF, i.e. $\alpha = Q_1 x_1 \ldots Q_n x_m \cdot \phi$

- Let $Q_k$ be the leftmost existential quantifier
  - If $k = 1$ then substitute all occurrence of $x_k$ in $\phi$ by a new constant $a$, called a **Skolem constant**, and remove $\exists x_1$ from the matrix.
  - If $k > 1$ then substitute all occurrences of $x_k$ in $\phi$ by a new function $f(x_1, \ldots, x_{k-1})$, called a **Skolem function**, and remove $\exists x_k$ from the matrix.

- Repeat the process until all $\exists$ are eliminated.
Example 10.1

\[ \alpha = \exists x \ \forall y. \text{Teaches}(x,y). \]

After Skolemization:

\[ \alpha' = \forall y. \text{Teaches}(a, y), \text{ where } a \text{ is a Skolem constant.} \]
Example 10.1

\[ \alpha = \exists x \ \forall y. \text{Teaches}(x, y). \]

After Skolemization:
\[ \alpha' = \forall y. \text{Teaches}(a, y), \text{where } a \text{ is a Skolem constant.} \]

\[ \beta = \forall x \ \exists y. \text{Likes}(x, y). \]

After Skolemization:
\[ \beta' = \forall x. \text{Likes}(x, f(x)), \text{where } f \text{ is a Skolem function.} \]
Assume that the following two clauses are given:

\[ \gamma_1 = \neg \text{Student}(x) \lor \text{Adult}(x) \]
\[ \gamma_2 = \text{Student}(Peter). \]

Note that \( \gamma_1 \) corresponds to the formula

\[ \forall x. \text{Student}(x) \rightarrow \text{Adult}(x), \]

so \( \gamma_1 \) and \( \gamma_2 \) imply \( \text{Adult}(Peter) \).

However, RR is not applicable to \( \gamma_1 \) and \( \gamma_2 \).
Substitution

Assume that the following two clauses are given:

\[ \gamma_1 = \neg \text{Student}(x) \lor \text{Adult}(x) \]
\[ \gamma_2 = \text{Student}(Peter). \]

Note that \( \gamma_1 \) corresponds to the formula

\[ \forall x. \, \text{Student}(x) \rightarrow \text{Adult}(x), \]

so \( \gamma_1 \) and \( \gamma_2 \) imply \( \text{Adult}(Peter) \).

However, RR is not applicable to \( \gamma_1 \) and \( \gamma_2 \).

We need a substitution, say \( x \leftarrow Peter \).
Definition 10.1 A **substitution** $\Theta$ is a set

$$\Theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

such that

- $x_1, \ldots, x_n$ are distinct variables
- $t_1, \ldots, t_n$ are terms not containing any $x_i, i = 1, \ldots, n$.

An empty substitution is denoted by $\varepsilon$.

**Example 10.2**

- $\Theta_1 = \{x \leftarrow a, y \leftarrow f(z)\}$ is a substitution
- $\Theta_2 = \{x \leftarrow a, y \leftarrow f(x)\}$ is **NOT** a substitution.

Substitutions are applied to formulas and terms.
Application of substitutions

Let

- $e$ – an expression (either a term or a formula)
- $\Theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ – a substitution.

An application of $\Theta$ to $e$, written $e\Theta$, is the expression $e'$ such the all occurrences of $x_1, \ldots, x_n$ in $e$ are respectively replaced by $t_1, \ldots, t_n$. 
Application of substitutions

Let

- $e$ – an expression (either a term or a formula)
- $\Theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ – a substitution.

An application of $\Theta$ to $e$, written $e\Theta$, is the expression $e'$ such the all occurrences of $x_1, \ldots, x_n$ in $e$ are respectively replaced by $t_1, \ldots, t_n$.

Example 10.3

- $e = \neg P(x, y) \lor Q(x, f(x))$
- $\Theta = \{x \leftarrow a, y \leftarrow g(z)\}$
- $e\Theta = \neg P(a, g(z)) \lor Q(a, f(a))$. 
Composition of substitutions

Given two substitutions

- \( \Theta_1 = \{ x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n \} \)
- \( \Theta_2 = \{ y_1 \leftarrow v_1, \ldots, y_k \leftarrow v_k \} \),

the *composition* \( \Theta_1 \Theta_2 \) is the set obtained from

\[
\{ x_1 \leftarrow t_1 \Theta_2, \ldots, x_n \leftarrow t_n \Theta_2, y_1 \leftarrow v_1, \ldots, y_k \leftarrow v_k \}
\]

by removing expressions of the form

- \( x_i \leftarrow t_i \Theta_2 \) such that \( x_i = t_i \Theta_2 \)
- \( y_i \leftarrow v_i \) such that \( y_i \in \{ x_1, \ldots, x_n \} \).
Composition of substitutions

Given two substitutions
- $\Theta_1 = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$
- $\Theta_2 = \{y_1 \leftarrow v_1, \ldots, y_k \leftarrow v_k\}$,
the composition $\Theta_1 \Theta_2$ is the set obtained from

\[\{x_1 \leftarrow t_1 \Theta_2, \ldots, x_n \leftarrow t_n \Theta_2, y_1 \leftarrow v_1, \ldots, y_k \leftarrow v_k\}\]

by removing expressions of the form
- $x_i \leftarrow t_i \Theta_2$ such that $x_i = t_i \Theta_2$
- $y_i \leftarrow v_i$ such that $y_i \in \{x_1, \ldots, x_n\}$.

Example 10.4

\[
\begin{align*}
\Theta_1 &= \{x \leftarrow g(y), z \leftarrow y\} \\
\Theta_2 &= \{x \leftarrow a, y \leftarrow b, z \leftarrow f(c)\} \\
\Theta_1 \Theta_2 &= \{x \leftarrow g(b), z \leftarrow b, y \leftarrow b\}.
\end{align*}
\]
A set $Ex = \{e_1, \ldots, e_n\}$ of expressions is **unifiable** iff there is a substitution $\Theta$ such that $e_1\Theta = e_2\Theta = \ldots = e_n\Theta$. 
A set $Ex = \{e_1, \ldots, e_n\}$ of expressions is **unifiable** iff there is a substitution $\Theta$ such that $e_1\Theta = e_2\Theta = \ldots = e_n\Theta$.

If $Ex$ is unifiable then any substitution which makes it unifiable is called a **unifier** of $Ex$. 

---

**Unifiers**
A set $Ex = \{e_1, \ldots, e_n\}$ of expressions is **unifiable** iff there is a substitution $\Theta$ such that $e_1 \Theta = e_2 \Theta = \ldots = e_n \Theta$.

If $Ex$ is unifiable then any substitution which makes it unifiable is called a **unifier** of $Ex$.

For example, for $Ex = \{P(x), P(y)\}$ the following substitutions are unifiers of $Ex$:

\begin{align*}
\Theta_1 &= \{x \leftarrow y\} \\
\Theta_2 &= \{y \leftarrow x\} \\
\Theta_3 &= \{x \leftarrow a, y \leftarrow a\} \\
\Theta_4 &= \{x \leftarrow f(z), y \leftarrow f(z)\}.
\end{align*}
Let $Ex$ be a unifiable set of expressions. A unifier $\Theta$ of $Ex$ is called the **most general unifier**, in symbols $MGU(Ex)$, iff for every unifier $\theta$ of $Ex$ there is a substitution $\delta$ such that $\theta = \Theta \delta$.

**Example 10.5** For $Ex = \{P(x), P(y)\}$, there are two MGUs:

$$\Theta_1 = \{x \leftarrow y\}$$
$$\Theta_2 = \{y \leftarrow x\}$$

since

$$\Theta_3 = \{x \leftarrow a, y \leftarrow a\} = \Theta_1\{y \leftarrow a\}$$
$$\Theta_4 = \{x \leftarrow f(z), y \leftarrow f(z)\} = \Theta_2\{x \leftarrow f(z)\}.$$
Most General Unifier

Let $Ex$ be a unifiable set of expressions. A unifier $\Theta$ of $Ex$ is called the **most general unifier**, in symbols $MGU(Ex)$, iff for every unifier $\theta$ of $Ex$ there is a substitution $\delta$ such that $\theta = \Theta\delta$.

**Example 10.5** For $Ex = \{P(x), P(y)\}$, there are two MGUs:

$$
\Theta_1 = \{x \leftarrow y\}
$$

$$
\Theta_2 = \{y \leftarrow x\}
$$

since

$$
\Theta_3 = \{x \leftarrow a, y \leftarrow a\} = \Theta_1\{y \leftarrow a\}
$$

$$
\Theta_4 = \{x \leftarrow f(z), y \leftarrow f(z)\} = \Theta_2\{x \leftarrow f(z)\}.
$$

Every unifiable set $Ex$ of expressions has at least one MGU.
How to determine $MGU(Ex)$ if it exists?
How to determine $\text{MGU}(\text{Ex})$ if it exists?

$\text{Ex} = \{e_1, \ldots, e_n\}$ — set of expressions.

A disagreement set of $\text{Ex}$ is the set $\text{DS}(\text{Ex}) = \{e'_1, \ldots, e'_m\}$ of expressions defined as:

- Consider $e_1, \ldots, e_n$ as a string of symbols.
- Determine the leftmost position $k$ at which at least two expressions from $\text{Ex}$ have different symbols.
- For each $i = 1, \ldots, k$, put $e'_i$ to be the subexpression of $e_i$ starting from the position $k$. 
How to determine $MGU(Ex)$ if it exists?

$Ex = \{e_1, \ldots, e_n\}$ — set of expressions.

A **disagreement set** of $Ex$ is the set $DS(Ex) = \{e'_1, \ldots, e'_m\}$ of expressions defined as:

- Consider $e_1, \ldots, e_n$ as a string of symbols.
- Determine the leftmost position $k$ at which at least two expressions from $Ex$ have different symbols.
- For each $i = 1, \ldots, k$, put $e'_i$ to be the subexpression of $e_i$ starting from the position $k$.

**Example 10.6**

$$Ex = \{P(x, f(a)), P(x, c), P(f(y), d)\}$$

$$DS(Ex) = \{x, f(y)\}.$$
Unification algorithm

**Input**: any set \(Ex\) of expressions

**Output**: \textsc{yes} iff \(Ex\) is unifiable (and \(MGU(Ex)\)); otherwise \textsc{no}.

**Step 1.** Set \(k = 0, E_0 = Ex, \Theta_0 = \varepsilon\)

**Step 2.** if \(E_k\) is a singleton then STOP — \(Ex\) is unifiable and \(MGU(Ex) = \Theta_k\); otherwise go to **Step 3**.

**Step 3.** Determine \(DS(E_k)\).

**Step 4.** If \(DS(E_k)\) contains a variable \(x\) and a term \(t\) not containing \(x\), then put

\[
\Theta_{k+1} = \Theta_k \{ x \leftarrow t \} \\
E_{k+1} = E_k \Theta_k \Theta_{k+1} \\
k := k + 1
\]

otherwise STOP — \(Ex\) is not unifiable.
Example 10.7 $Ex_1 = \{Q(x, f(b)), Q(a, f(y))\}$.

1. $k = 0, \Theta_0 = \varepsilon, E_0 = Ex_1$.
2. $DS(E_0) = \{x, a\}, \Theta_1 = \{x \leftarrow a\}$.
3. $E_1 = \{Q(a, f(b)), Q(a, f(y))\}, k = 1$.
4. $DS(E_1) = \{b, y\}, \Theta_2 = \{x \leftarrow a, y \leftarrow b\}$.
5. $E_2 = \{Q(a, f(b))\}$.

$Ex_1$ is unifiable and $MGU(Ex_1) = \{x \leftarrow a, y \leftarrow b\}$. 
Example 10.8 $Ex_2 = \{Q(x, x), Q(a, b)\}$.

1. $k = 0$, $\Theta_0 = \varepsilon$, $E_0 = Ex_2$.
2. $DS(E_0) = \{x, a\}$, $\Theta_1 = \{x \leftarrow a\}$.
3. $E_1 = \{Q(a, a), Q(a, b)\}$, $k = 2$.
4. $DS(E_1) = \{a, b\}$.

$Ex_2$ is NOT unifiable!
Binary resolvents

For a literal $\lambda$, $\bar{\lambda}$ stands for the atom $\alpha$ corresponding to $\lambda$, that is
1. if $\lambda$ is an atom $\alpha$, then $\bar{\lambda} = \alpha$
2. if $\lambda$ is the negation of an atom $\alpha$ (i.e. $\lambda = \neg \alpha$), then $\bar{\lambda} = \alpha$.

Let
- $\gamma_1$, $\gamma_2$ be clauses with different variables
- $\gamma_1$ contains a literal $\lambda_1$ and $\gamma_2$ contains a literal $\lambda_2$ such that $\lambda_1$ and $\lambda_2$ are complementary
- $\Theta = MGU(\bar{\lambda_1}, \bar{\lambda_2})$.

A binary resolvent of $\gamma_1$ and $\gamma_2$ is a clause

$$ResBin(\gamma_1, \gamma_2) = (\gamma_1 \Theta \setminus \lambda_1 \Theta) \lor (\gamma_2 \Theta \setminus \lambda_2 \Theta)$$
Observe first that the clauses

\[ P(x) \lor P(y) \]
\[ \neg P(z) \lor \neg P(u) \]

are inconsistent. However, by successively applying RR we cannot derive \( \square ! \)
Observe first that the clauses

\[ P(x) \lor P(y) \]
\[ \neg P(z) \lor \neg P(u) \]

are inconsistent. However, by successively applying RR we cannot derive \( \square \)!

To solve this problem, a \textit{factorization process} is performed.
Let $\gamma = \lambda_1 \land \ldots \land \lambda_n$, which can be viewed as a set $\{\lambda_1, \ldots, \lambda_n\}$ of its literals.

Take a unifiable subset $L \subseteq \{\lambda_1, \ldots, \lambda_n\}$ of literals from $\gamma_1$.

Put $\Theta = MGU(L)$.

A factor of $\gamma$ is a clause $\gamma'$ obtained from $\gamma\Theta$ by removing repeating literals.
Example 10.9 Consider a clause \( \gamma_1 = P(x) \lor P(f(z)) \lor \neg Q(x) \).

- Take \( L = \{ P(x), P(f(z)) \} \)
- \( \Theta = MGU(L) = \{ x \leftarrow f(z) \} \)
- \( Factor(\gamma_1) = P(f(z)) \lor \neg Q(f(z)) \).
Example 10.9 Consider a clause $\gamma_1 = P(x) \lor P(f(z)) \lor \neg Q(x)$.

- Take $L = \{P(x), P(f(z))\}$
- $\Theta = MGU(L) = \{x \leftarrow f(z)\}$
- $Factor(\gamma_1) = P(f(z)) \lor \neg Q(f(z))$.

Let $\gamma_2 = \neg P(y, a) \lor \neg P(y, x) \lor \neg P(x, y)$.

- Take $L = \{\neg P(y, a), \neg P(y, x), \neg P(x, y)\}$
- $\Theta = MGU(L) = \{x \leftarrow a, y \leftarrow b\}$
- $Factor(\gamma_2) = \neg P(a, a)$ (unit factor).
Definition 10.2 A resolvent of two clauses $\gamma_1$ and $\gamma_2$, in symbols $Res(\gamma_1, \gamma_2)$, is any of the following binary resolvent

- $ResBin(\gamma_1, \gamma_2)$
- $ResBin(\gamma_1, Factor(\gamma_2))$
- $ResBin(Factor(\gamma_1), \gamma_2)$
- $ResBin(Factor(\gamma_1), Factor(\gamma_2))$. 
A *resolution proof* of $\beta$ from $T$ is a sequence $(\gamma_0, \ldots, \gamma_n)$ of clauses such that

- $\gamma_0 \in CL(T, \neg \beta)$
- $\gamma_n = \Box$
- for every $i = 1, \ldots, n$, either $\gamma_i \in CL(T, \neg \beta)$ or $\gamma_i = Res(\gamma_k, \gamma_j)$, for $0 \leq k, j < i$. 


A resolution proof of $\beta$ from $T$ is a sequence $(\gamma_0, \ldots, \gamma_n)$ of clauses such that

- $\gamma_0 \in CL(T, \neg \beta)$
- $\gamma_n = \square$
- for every $i = 1, \ldots, n$, either $\gamma_i \in CL(T, \neg \beta)$ or $\gamma_i = Res(\gamma_k, \gamma_j)$, for $0 \leq k, j < i$.

**Theorem 10.3** (Soundness and Correctness) $T \vdash \beta$ iff there is a resolution proof of $\beta$ from $T$. 
Resolution method – outline

Given $T$ and $\beta$, show $T \vdash \beta$.

**Step 1.** For every $\alpha \in T$ and for $\neg \beta$ determine its PCNF.

**Step 2.** If necessary, apply Skolemization. Get $CL = CL(T, \neg \beta)$.

**Step 3.** If $\Box \in CL$, then STOP and return **Yes**.

**Step 4.** For any clauses $\gamma_1, \gamma_2 \in CL(T, \neg \beta)$, try to find unifiable literals $\lambda_1 \in \gamma_1$ and $\lambda_2 \in \gamma_2$ such that $\lambda_1 \iff \neg \lambda_2$. If no such clauses can be found, then STOP and return **NO**.

**Step 5.** Calculate $\gamma = Res(\gamma_1, \gamma_2)$.

**Step 6.** Put $CL := CL \cup \{\gamma\}$ and return to **Step 3**.

**Remark:** This process need not be finite!
Answer Extraction

In many cases the query is of the form $\beta = \exists x. \phi(x)$ and we are actually interested in the question: “Who has the property $\phi$”. Obtaining answers for such queries is called an answer extraction.

**General idea:**

Replace a query $\beta = \exists x. \phi(x)$ by

$$\beta' = \exists x. \phi(x) \land \neg \text{Answer}(x)$$

where $\text{Answer}$ is a new predicate occurring nowhere else. Since it does not occur elsewhere, we cannot derive $\Box$. Instead, we terminate the derivation process as soon as a clause containing only the answer predicate is produced.
Consider the set $T$ of the following sentences:

$$\alpha_1 = Bird(Tweety)$$
$$\alpha_2 = \forall x. Bird(x) \rightarrow Flies(x)$$
$$\alpha_3 = \forall x. Flies(x) \rightarrow HasWings(x)$$

and the formula $\beta = HasWings(tweety)$. 
Example 10.10a

Consider the set $T$ of the following sentences:

\[
\begin{align*}
\alpha_1 &= \text{Bird}(\text{Tweety}) \\
\alpha_2 &= \forall x. \text{Bird}(x) \rightarrow \text{Flies}(x) \\
\alpha_3 &= \forall x. \text{Flies}(x) \rightarrow \text{HasWings}(x)
\end{align*}
\]

and the formula $\beta = \text{HasWings}(\text{tweety})$.

The corresponding set $CL(T, \neg \beta)$ consists of the following clauses:

\[
\begin{align*}
\text{Bird}(\text{Tweety}) \\
\neg \text{Bird}(x) \lor \text{Flies}(x) \\
\neg \text{Flies}(x) \lor \text{HasWings}(x) \\
\neg \text{HasWings}.
\end{align*}
\]
Example 10.10a: Derivation tree

\[ \neg Wings(tweety) \quad \neg Flies(x) \lor Wings(x) \quad \neg Bird(x) \lor Flies(x) \quad Bird(tweety) \]

\[ \neg Flies(tweety) \quad Flies(tweety) \]

\[ \square \]
Example 10.10b

Consider the same set $T$ of sentences:

\[
\begin{align*}
Bird&(Tweety) \\
\forall x. \ Bird(x) &\rightarrow Flies(x) \\
\forall x. \ Flies(x) &\rightarrow HasWings(x)
\end{align*}
\]

and the formula $\beta = \exists x. \ HasWings(x)$. 

Example 10.10b

Consider the same set $T$ of sentences:

$$
\begin{align*}
&\text{Bird(Tweety)} \\
&\forall x. \text{Bird}(x) \rightarrow \text{Flies}(x) \\
&\forall x. \text{Flies}(x) \rightarrow \text{HasWings}(x)
\end{align*}
$$

and the formula $\beta = \exists x. \text{HasWings}(x)$.

The set $CL(T, \neg \beta)$ consists of clauses:

$$
\begin{align*}
&\text{Bird(Tweety)} \\
&\neg \text{Bird}(x) \lor \text{Flies}(x) \\
&\neg \text{Flies}(x) \lor \text{HasWings}(x) \\
&\neg \text{HasWings}(x) \lor \text{Answer}(x).
\end{align*}
$$
Example 10.10b: Derivation tree

\[
\begin{align*}
\neg \text{Wings}(x') \lor \text{Answer}(x') & \quad \neg \text{Flies}(x) \lor \text{Wings}(x) & \quad \neg \text{Bird}(x) \lor \text{Flies}(x) & \quad \text{Bird(tweedy)} \\
\neg \text{Flies}(x) \lor \text{Answer}(x) & \quad & \text{Flies(tweeety)} & \quad \text{Answer(tweety)}
\end{align*}
\]
Now let $T$ consists of sentences:

\[
\begin{align*}
&\text{Bird(Tweety)} \lor \text{Bird(Clyde)} \\
&\forall x. \text{Bird}(x) \rightarrow \text{Flies}(x) \\
&\forall x. \text{Flies}(x) \rightarrow \text{HasWings}(x)
\end{align*}
\]

and let $\beta = \exists x. \text{HasWings}(x)$. 
Now let $T$ consists of sentences:

$$\text{Bird(Tweety)} \lor \text{Bird(Clyde)}$$
$$\forall x. \text{Bird}(x) \rightarrow \text{Flies}(x)$$
$$\forall x. \text{Flies}(x) \rightarrow \text{HasWings}(x)$$

and let $\beta = \exists x. \text{HasWings}(x)$.

Intuitively, $T$ implies that Tweety has wings, or Clyde has wings, or both have wings, i.e., $T \vdash \text{HasWing(Tweety)} \lor \text{HasWings(Clyde)}$. 
Example 10.10c

Now let $T$ consists of sentences:

$$
\begin{align*}
\text{Bird}(\text{Tweety}) & \lor \text{Bird}(\text{Clyde}) \\
\forall x. \text{Bird}(x) & \rightarrow \text{Flies}(x) \\
\forall x. \text{Flies}(x) & \rightarrow \text{HasWings}(x)
\end{align*}
$$

and let $\beta = \exists x. \text{HasWings}(x)$.

Intuitively, $T$ implies that Tweety has wings, or Clyde has wings, or both have wings, i.e., $T \vdash \text{HasWing}(\text{Tweety}) \lor \text{HasWings}(\text{Clyde})$.

The set $CL(T, \neg \beta)$ is as before.
Example 10.10c: Derivation tree

\[ \neg \text{Wings}(x) \vee \text{Answer}(x) \]
\[ \neg \text{Flies}(x) \vee \text{Wings}(x) \]
\[ \neg \text{Bird}(x) \vee \text{Flies}(x) \]
\[ \text{Bird}(t) \vee \text{Bird}(c) \]

\[ \neg \text{Flies}(x) \vee \text{Answer}(x) \]
\[ \text{Flies}(t) \vee \text{Bird}(c) \]

\[ \text{Flies}(t) \vee \text{Flies}(c) \]
\[ \text{Answer}(t) \vee \text{Flies}(c) \]

\[ \text{Answer}(t) \vee \text{Answer}(c) \]
Thank you for your attention!