1. Backward chaining

2. Resolution
Substitution and Composition

Definition

Given \( p \) a sentence and \( \theta_1, \theta_2 \) two substitutions, the composition of \( \theta_1 \) and \( \theta_2 \) is the substitution \( \theta = \text{COMPOSE}(\theta_1, \theta_2) \) such that:

\[
\text{SUBST}(\theta, p) = \text{SUBST}(\theta_1, \text{SUBST}(\theta_2, p)) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))
\]

Example

Sentence \( p \) is \( P(y) \land Q(x) \Rightarrow R(z) \)
Consider \( \theta_1 = \{y / Toto, z / Titi\}, \theta_2 = \{x / Tata\} \)
\[
\begin{align*}
\text{SUBST}(\theta_1, p) &= P(Toto) \land Q(x) \Rightarrow R(Titi) \\
\text{SUBST}(\theta_2, p) &= P(y) \land Q(Tata) \Rightarrow R(z) \\
\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) &= P(Toto) \land Q(Tata) \Rightarrow R(Titi)
\end{align*}
\]
Backward chaining: main idea

**Definition**

Given a definite clause \( p_1 \land \cdots \land p_n \Rightarrow c \), \( c \) is called the **Head**. \( p_1 \land \cdots \land p_n \) is called the **Body**.

**BC Idea**

*Goal-driven* algorithm.

1. Unification of the goal with the head of a rule
2. Propagation of the substitution to the body. Every premise of the body is a new goal
3. Apply BC recursively on the new goals...

Based on a depth-first search (DFS).
function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions

inputs: KB, a knowledge base

goals, a list of conjuncts forming a query (θ already applied)
θ, the current substitution, initially the empty substitution {}

local variables: answers, a set of substitutions, initially empty

if goals is empty then return {θ}

q’ ← SUBST(θ, FIRST(goals))

for each sentence r in KB

where STANDARDIZE-Apart(r) = (p₁ ∧ ... ∧ pₙ ⇒ q)

and θ’ ← UNIFY(q, q’) succeeds

new_goals ← [p₁, ..., pₙ | REST(goals)]

answers ← FOL-BC-Ask(KB, new_goals, COMPOSE(θ’, θ)) ∪ answers

return answers
Backward chaining: DFS-tree

Example

\{ x / Yannick \}

GoodTeacher(Yannick)
Backward chaining: DFS-tree

Example

S1
Person(Yannick)
S5
GoodTeacher(Yannick)
{ x / Yannick }
{ }
Backward chaining: DFS-tree

Example

Person(Yannick) → GoodLecturesFOL(y) → GoodTeacher(Yannick) → { x / Yannick }
Backward chaining: DFS-tree

Example

S1
GoodTeacher(Yannick)

{ x / Yannick }

S2
GoodLecturesLogic(M1)

{ y / M1 }

S5
Person(Yannick)

{ }

S7
GoodLecturesFOL(y)
Example

Backward chaining: DFS-tree

S1
GoodTeacher(Yannick)

{ x / Yannick }

S2
GoodLecturesLogic(M1)

{ y / M1 }

S3
Gives(Yannick,M1,People)

{ z / People }

compose

S5
Person(Yannick)

S7
GoodLecturesFOL(y)
Backward chaining: DFS-tree

Example

- **Person(Yannick)**
- **GoodLecturesLogic(M1)**
- **GoodTeacher(Yannick)**
- **GoodLecturesFOL(y)**
- **Gives(Yannick,M1,People)**

The composition is shown with the following constraints:

- \( \{ x / Yannick \} \)
- \( \{ z / People \} \)
- \( \{ y / M1 \} \)
- \( \{ \} \)
Backward chaining: DFS-tree

Example

Person(Yannick)
GoodLecturesFOL(y)
GoodLecturesLogic(M1)
Gives(Yannick,M1,People)

{ y / M1 }
{ } { z / People }
compose

S1

{ x / Yannick }

S3

S2
S2
S2
Backward chaining: DFS-tree

Example

\[
\begin{align*}
\text{Person}(\text{Yannick}) & \rightarrow \text{GoodLecturesFOL}(y) \\
\text{GoodLecturesLogic}(M1) & \rightarrow \text{Have}(\text{People},M1) \\
\text{Gives}(\text{Yannick},M1,\text{People}) & \rightarrow \text{Students}(\text{People})
\end{align*}
\]
Backward chaining: DFS-tree

Example

\[
\begin{align*}
\text{Person(Yannick)} & \quad \text{GoodLecturesFOL}(y) \\
& \quad \text{compose} \\
& \quad \text{Gives(Yannick, M1, People)} \\
& \quad \text{compose} \\
& \quad \text{Students(People)} \\
\end{align*}
\]
Backward chaining: DFS-tree

Example

Answer = { x / Yannick, y / M1, z / People }
Depth-first recursive proof search: space is linear in size of the proof.

Incomplete due to infinite loops (DFS). To fix that, we have to check the current goal against every goal in the stack.

Inefficient due to repeated subgoals. To fix that we must use a cache of previous results (memoization)

So what? If BC is not so good, why do we talk about it? Well, it is widely used and with good optimisations it works! (linear algorithm): PROLOG
Outline

1. Backward chaining
2. Resolution
Example

Everyone who loves all animals is loved by someone. Anyone who kills an animal is loved by no one. Jack loves all animals. Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?
Example

“Everyone who loves all animals is loved by someone.”
Another Knowledge base

Example

“Everyone who loves all animals is loved by someone.”

$$\forall x \ [ \forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)]$$
Example

“Everyone who loves all animals is loved by someone.”
∀x [∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)]

“Anyone who kills an animal is loved by no one..”
Another Knowledge base

Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \left[ \forall y \text{Animal}(y) \implies \text{Loves}(x, y) \right] \implies \exists y \text{Loves}(y, x) \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x \left[ \exists y \text{Animal}(y) \land \text{Kills}(x, y) \right] \implies \forall z \neg \text{Loves}(z, x) \]
Another Knowledge base

Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y) \right] \Rightarrow \left[ \exists y \text{Loves}(y, x) \right] \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x \left[ \exists y \text{Animal}(y) \land \text{Kills}(x, y) \right] \Rightarrow \left[ \forall z \lnot \text{Loves}(z, x) \right] \]

“Jack loves all animals”
Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \ [ \forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x [\exists y Animal(y) \land Kills(x, y)] \Rightarrow [\forall z \neg Loves(z, x)] \]

“Jack loves all animals”
\[ \forall x \ Animal(x) \Rightarrow Loves(Jack, x) \]
Example

“Everyone who loves all animals is loved by someone.”
\( \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y) \right] \Rightarrow [\exists y \text{Loves}(y, x)] \)

“Anyone who kills an animal is loved by no one.”
\( \forall x [\exists y \text{Animal}(y) \land \text{Kills}(x, y)] \Rightarrow [\forall z \neg \text{Loves}(z, x)] \)

“Jack loves all animals”
\( \forall x \text{Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x) \)

“Either Jack or Curiosity killed the cat, who is named Tuna”
“Everyone who loves all animals is loved by someone.”
\[ \forall x \ [ \forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

“Anyone who kills an animal is loved by no one..”
\[ \forall x [\exists y \ Animal(y) \land Kills(x, y)] \Rightarrow [\forall z \ \neg Loves(z, x)] \]

“Jack loves all animals”
\[ \forall x \ Animal(x) \Rightarrow Loves(Jack, x) \]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[ Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna) \]
Another Knowledge base

Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow Loves(x, y) \right] \Rightarrow \left[ \exists y \text{Loves}(y, x) \right] \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x \left[ \exists y \text{Animal}(y) \land \text{Kills}(x, y) \right] \Rightarrow \left[ \forall z \neg \text{Loves}(z, x) \right] \]

“Jack loves all animals”
\[ \forall x \text{Animal}(x) \Rightarrow \text{Loves(Jack, x)} \]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[ \text{Kills(Jack, Tuna)} \lor \text{Kills(Curiosity, Tuna)} \]

Tuna is a cat
Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \ [\forall y \text{ Animal}(y) \Rightarrow \text{ Loves}(x, y)] \Rightarrow [\exists y \text{ Loves}(y, x)] \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x [\exists y \text{ Animal}(y) \wedge \text{ Kills}(x, y)] \Rightarrow [\forall z \neg \text{ Loves}(z, x)] \]

“Jack loves all animals”
\[ \forall x \text{ Animal}(x) \Rightarrow \text{ Loves}(\text{Jack}, x) \]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[ \text{ Kills(}\text{Jack}, \text{ Tuna}) \lor \text{ Kills(}\text{Curiosity}, \text{ Tuna}) \]

Tuna is a cat
\[ \text{Cat(} \text{Tuna}) \]
Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y) \right] \Rightarrow \left[ \exists y \text{Loves}(y, x) \right] \]

“Anyone who kills an animal is loved by no one..”
\[ \forall x \left[ \exists y \text{Animal}(y) \land \text{Kills}(x, y) \right] \Rightarrow \left[ \forall z \neg \text{Loves}(z, x) \right] \]

“Jack loves all animals”
\[ \forall x \text{Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x) \]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[ \text{Kills} (\text{Jack}, \text{Tuna}) \lor \text{Kills} (\text{Curiosity}, \text{Tuna}) \]

Tuna is a cat
\[ \text{Cat}(\text{Tuna}) \]

A cat is an animal
Another Knowledge base

Example

“Everyone who loves all animals is loved by someone.”
\[\forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)]\]

“Anyone who kills an animal is loved by no one.”
\[\forall x [\exists yAnimal(y) \land Kills(x, y)] \Rightarrow [\forall z \ \neg Loves(z, x)]\]

“Jack loves all animals”
\[\forall x \ Animal(x) \Rightarrow Loves(Jack, x)\]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna)\]

Tuna is a cat
\[Cat(Tuna)\]

A cat is an animal
\[\forall x \ Cat(x) \Rightarrow Animal(x)\]
Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x [\exists y Animal(y) \land Kills(x, y)] \Rightarrow [\forall z \neg Loves(z, x)] \]

“Jack loves all animals”
\[ \forall x \ Animal(x) \Rightarrow Loves(Jack, x) \]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[ Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna) \]

Tuna is a cat
\[ Cat(Tuna) \]

A cat is an animal
\[ \forall x \ Cat(x) \Rightarrow Animal(x) \]

Question: Did Curiosity kill the cat?
Another Knowledge base

Example

“Everyone who loves all animals is loved by someone.”
\[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y) \right] \Rightarrow \left[ \exists y \text{Loves}(y, x) \right] \]

“Anyone who kills an animal is loved by no one.”
\[ \forall x \left[ \exists y \text{Animal}(y) \wedge \text{Kills}(x, y) \right] \Rightarrow \left[ \forall z \neg \text{Loves}(z, x) \right] \]

“Jack loves all animals”
\[ \forall x \text{Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x) \]

“Either Jack or Curiosity killed the cat, who is named Tuna”
\[ \text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna}) \]

Tuna is a cat
\[ \text{Cat}(\text{Tuna}) \]

A cat is an animal
\[ \forall x \text{Cat}(x) \Rightarrow \text{Animal}(x) \]

Question: Did Curiosity kill the cat?
\[ \text{Kills}(\text{Curiosity}, \text{Tuna}) \]
Conjunctive Normal Form for FOL

Conjunctive Normal Form

A sentence in a Conjunctive Normal Form is a conjunction of clauses, each clause is a disjunction of literals.

Property

Every sentence in FOL (without equality) is logically equivalent to a FOL-CNF sentence.

Example

“Everyone who loves all animals is loved by someone”
\[ \forall x \left( \forall y \text{ Animal}(y) \Rightarrow \text{ Loves}(x, y) \right) \Rightarrow \left[ \exists y \text{ Loves}(y, x) \right] \]

has the following CNF
\[ [\text{Animal}(F(x)) \lor \text{ Loves}(G(x), x)] \land [\neg \text{ Loves}(x, F(x)) \lor \text{ Loves}(G(x), x)]. \]
## Conversion to CNF

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Elimination of implications</td>
<td>$A \Rightarrow B \equiv \neg A \lor B$</td>
</tr>
<tr>
<td>2 Move $\neg$ inwards</td>
<td></td>
</tr>
<tr>
<td>3 Standardize variables</td>
<td></td>
</tr>
<tr>
<td>4 Skolemisation</td>
<td></td>
</tr>
<tr>
<td>5 Drop the universal quantifiers</td>
<td></td>
</tr>
<tr>
<td>6 Distribute $\lor$ over $\land$</td>
<td></td>
</tr>
</tbody>
</table>
Move $\neg$ inwards and variable standardization

**Rules for negated quantifiers**

$\neg\forall x \ p \equiv \exists x \ \neg p$

$\neg\exists x \ p \equiv \forall x \ \neg p$

**Variable standardization**

$$(\forall x \ P(x)) \lor (\exists x \ Q(x))$$

$x$ is used twice but it does not represent the same thing (two different scopes). To avoid confusion, we rename:

$$(\forall x \ P(x)) \lor (\exists y \ Q(y))$$
Skolemization

**Definition**

**Skolemisation** is the process of removing existential quantifiers by elimination.

- Simple case = Existential Instanciation
- Complex case = Use of Skolem functions

**Example**

Simple case: \( \exists x \ P(x) \)
Using EI, we have: \( P(A) \)

Complex case: \( \forall x \ [\exists y \ P(x, y)] \)
Using EI, we have: \( \forall x \ P(x, A) \) wrong

Use of a Skolem function \( F(x): \forall x \ P(x, F(x)) \)
(y in is the scope of x)
Example

∀x [∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)]
Conversion to CNF: example

Example

\[ \forall x \left[ \forall y \ Animal(y) \implies Loves(x, y) \right] \implies \exists y \ Loves(y, x) \]

1. Eliminate implications:
   \[ \forall x \left[ \neg \forall y \neg Animal(y) \lor Loves(x, y) \right] \lor \exists y \ Loves(y, x) \]
Conversion to CNF: example

Example

\[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y) \right] \Rightarrow \left[ \exists y \text{Loves}(y, x) \right] \]

1. Eliminate implications:
   \[ \forall x \left[ \neg \forall y \neg \text{Animal}(y) \lor \text{Loves}(x, y) \right] \lor \left[ \exists y \text{Loves}(y, x) \right] \]

2. Move \( \neg \) inwards
   \[ \forall x \left[ \exists y \neg \left( \neg \text{Animal}(y) \lor \text{Loves}(x, y) \right) \right] \lor \left[ \exists y \text{Loves}(y, x) \right] \]
Conversion to CNF: example

Example

∀x [∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)]

1. Eliminate implications:
   ∀x [¬∀y ¬Animal(y) ∨ Loves(x, y)] ∨ [∃y Loves(y, x)]

2. Move ¬ inwards
   - ∀x [∃y ¬(¬Animal(y) ∨ Loves(x, y))] ∨ [∃y Loves(y, x)]
   - ∀x [∃y ¬¬Animal(y) ∧ ¬Loves(x, y)] ∨ [∃y Loves(y, x)] (De Morgan)
Conversion to CNF: example

Example

\( \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \)

1. Eliminate implications:
   \( \forall x \ [\neg \forall y \ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y \ Loves(y, x)] \)

2. Move \( \neg \) inwards
   - \( \forall x \ [\exists y \ \neg (\neg Animal(y) \lor Loves(x, y))] \lor [\exists y \ Loves(y, x)] \)
   - \( \forall x \ [\exists y \ \neg \neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \) (De Morgan)
   - \( \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \) (double negation)
Conversion to CNF: example

Example

∀x [∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)]

1. Eliminate implications:
   ∀x [¬∀y ¬Animal(y) ∨ Loves(x, y)] ∨ [∃y Loves(y, x)]

2. Move ¬ inwards
   - ∀x [∃y ¬(¬Animal(y) ∨ Loves(x, y))] ∨ [∃y Loves(y, x)]
   - ∀x [∃y ¬¬Animal(y) ∧ ¬Loves(x, y)] ∨ [∃y Loves(y, x)] (De Morgan)
   - ∀x [∃y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃y Loves(y, x)] (double negation)

3. Standardize variables:
   ∀x [∃y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃z Loves(z, x)]
Conversion to CNF: example

Example

\[ \forall x [\forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y \text{Loves}(y, x)] \]

1. Eliminate implications:
   \[ \forall x [\neg \forall y \neg \text{Animal}(y) \lor \text{Loves}(x, y)] \lor [\exists y \text{Loves}(y, x)] \]

2. Move \( \neg \) inwards
   - \[ \forall x [\exists y \neg (\neg \text{Animal}(y) \lor \text{Loves}(x, y))] \lor [\exists y \text{Loves}(y, x)] \]
   - \[ \forall x [\exists y \neg \neg \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists y \text{Loves}(y, x)] \] (De Morgan)
   - \[ \forall x [\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists y \text{Loves}(y, x)] \] (double negation)

3. Standardize variables:
   \[ \forall x [\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists z \text{Loves}(z, x)] \]

4. Skolemization:
   \[ \forall x [\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))] \lor [\text{Loves}(G(x), x)] \]
Conversion to CNF: example

Example

∀x [∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)]

1. Eliminate implications:
   ∀x [¬∀y ¬Animal(y) ∨ Loves(x, y)] ∨ [∃y Loves(y, x)]

2. Move ¬ inwards
   • ∀x [∃y ¬(¬Animal(y) ∨ Loves(x, y))] ∨ [∃y Loves(y, x)]
   • ∀x [∃y ¬¬Animal(y) ∧ ¬Loves(x, y)] ∨ [∃y Loves(y, x)] (De Morgan)
   • ∀x [∃y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃y Loves(y, x)]
   (double negation)

3. Standardize variables:
   ∀x [∃y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃z Loves(z, x)]

4. Skolemization: ∀x [Animal(F(x)) ∧ ¬Loves(x, F(x))] ∨ [Loves(G(x), x)]

5. Drop universal quantifiers:
   [Animal(F(x)) ∧ ¬Loves(x, F(x))] ∨ [Loves(G(x), x)]
### Conversion to CNF: example

#### Example

\[ \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

1. **Eliminate implications:**
   \[ \forall x \ [\neg \forall y \ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]

2. **Move \( \neg \) inwards**
   - \[ \forall x \ [\exists y \ \neg (\neg Animal(y) \lor Loves(x, y))] \lor [\exists y \ Loves(y, x)] \] (De Morgan)
   - \[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \] (double negation)

3. **Standardize variables:**
   \[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists z \ Loves(z, x)] \]

4. **Skolemization:**
   \[ \forall x \ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor [Loves(G(x), x)] \]

5. **Drop universal quantifiers:**
   \[ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor [Loves(G(x), x)] \]

6. **Distribute \( \lor \) over \( \land \):**
   \[ [Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)] \]
Resolution: inference rule

Rule

\[
\ell_1 \lor \cdots \lor \boxed{\ell_i} \lor \cdots \lor \ell_k, \quad \ell'_1 \lor \cdots \lor \boxed{\ell'_j} \lor \cdots \lor \ell'_n
\]

\[
\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor \ell'_1 \lor \cdots \lor \ell'_{j-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n)
\]

with \( \theta \) a substitution such that \( \text{UNIFY}(\boxed{\ell_i}, \neg \boxed{\ell'_j}) = \theta \)

Example

\[
[\text{Animal}(F(x)) \lor \text{Loves}(G(x), x)] \ [\neg \text{Loves}(u, v) \lor \neg \text{Kills}(u, v)]
\]
Resolution: inference rule

**Rule**

\[
\ell_1 \lor \cdots \lor \boxed{\ell_i} \lor \cdots \lor \ell_k, \quad \ell'_1 \lor \cdots \lor \boxed{\ell'_j} \lor \cdots \lor \ell'_n
\]

\[
\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor \ell'_1 \lor \cdots \lor \ell'_{j-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n)
\]

with \(\theta\) a substitution such that UNIFY(\(\boxed{\ell_i}, \neg \boxed{\ell'_j}\)) = \(\theta\)

**Example**

\[
[\text{Animal}(F(x)) \lor \boxed{\text{Loves}(G(x), x)}] \quad [\neg \text{Loves}(u, v) \lor \neg \text{Kills}(u, v)]
\]
Resolution: inference rule

**Rule**

\[
\begin{align*}
\ell_1 \lor \cdots \lor \boxed{\ell_i} \lor \cdots \lor \ell_k, & \quad \ell'_1 \lor \cdots \lor \boxed{\ell'_j} \lor \cdots \lor \ell'_n \\
\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor \ell'_1 \lor \cdots \lor \ell'_{j-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n) & \\
\end{align*}
\]

with \( \theta \) a substitution such that \( \text{UNIFY}(\boxed{\ell_i}, \neg \boxed{\ell'_j}) = \theta \)

**Example**

\[
\begin{align*}
[Animal(F(x)) \lor \boxed{\text{Loves}(G(x), x)}] & \quad [\neg \text{Loves}(u, v) \lor \neg \text{Kills}(u, v)] \\
\text{SUBST}(\theta, \text{Animal}(F(x)) \lor \neg \text{Kills}(u, v)) & \\
\end{align*}
\]
Resolution: inference rule

Rule

\[ \ell_1 \lor \cdots \lor \ell_i \lor \cdots \lor \ell_k, \quad \ell'_1 \lor \cdots \lor \ell'_j \lor \cdots \lor \ell'_n \]

\[
\text{SUBST} \left( \theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor \ell'_1 \lor \cdots \lor \ell'_{j-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n \right)
\]

with \( \theta \) a substitution such that \( \text{UNIFY}(\ell_i, \neg \ell'_j) = \theta \)

Example

\[
[\text{Animal}(F(x)) \lor \text{Loves}(G(x), x)] \quad [\neg \text{Loves}(u, v) \lor \neg \text{Kills}(u, v)]
\]

\[
\text{SUBST}(\theta, \text{Animal}(F(x)) \lor \neg \text{Kills}(u, v))
\]

\( \theta = \{u/G(x), v/x\} \)
Resolution: inference rule

Rule

\[ \ell_1 \lor \cdots \lor \boxed{\ell_i} \lor \cdots \lor \ell_k, \quad \ell'_1 \lor \cdots \lor \boxed{\ell'_j} \lor \cdots \lor \ell'_n \]

\[ \text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor \ell'_1 \lor \cdots \lor \ell'_{j-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n) \]

with \( \theta \) a substitution such that \( \text{UNIFY}(\boxed{\ell_i}, \neg \boxed{\ell'_j}) = \theta \)

Example

\[ [\text{Animal}(F(x)) \lor \boxed{\text{Loves}(G(x), x)}] \quad [\neg \text{Loves}(u, v) \lor \neg \text{Kills}(u, v)] \]

\[ \text{Animal}(F(x)) \lor \neg \text{Kills}(G(x), x) \]

\( \theta = \{u/G(x), v/x\} \)
Resolution algorithm

Definition

Proof by contradiction: given $KB$, to prove $\alpha$, we prove that $KB \land \neg \alpha$ is not satisfiable.
Resolution Proof that Curiosity has killed the cat:

- \( \neg \alpha \) is \( \neg \text{Kills}(\text{Curiosity}, \text{Tuna}) \)
- Use of the factoring rule to infer \( \text{Loves}(G(\text{Jack}), \text{Jack}) \)
Dealing with equality

Paramodulation

There are several ways to deal with $t_1 = t_2$. One of them is 
**Paramodulation:**

\[
\ell_1 \lor \cdots \lor \ell_k \lor t_1 = t_2, \quad \ell'_1 \lor \cdots \lor \ell'_n[t_3] \\
\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell'_n[y])
\]

where

\[
\text{UNIFY}(t_1, t_3) = \theta
\]

This inference rule can be used during the resolution algorithm.

Example

\[
\text{Father}(John) = \text{Father}(Richard) \quad \text{Male}(\text{Father}(x)) \\
\text{Male}(\text{Father}(Richard))
\]

$\theta = \{x/John\} = \text{UNIFY}(\text{Father}(John), \text{Father}(x))$
Unlike logic programming language, Theorem provers cover FOL (no restriction on Definite Clauses). Their algorithm is based on resolution. Theorem prover: OTTER

**Optimisations**

Using the resolution algorithm in a “clever” way.

- **Unit preference**: Inference of sentences with a minimal number of literals (more chance to get the empty clause)
- **Set of support**: What is the set of clauses in KB that will be *useful*?
- **Input resolution**: Always using a sentence from KB or $\alpha$ to apply the resolution rule.
- **Subsumption**: Elimination of sentences that are subsumed by (more specific than) an existing sentence in the KB.
Completeness of resolution

Any set of sentences $S$ is representable in clausal form

Assume $S$ is unsatisfiable, and in clausal form

Some set $S'$ of ground instances is unsatisfiable

Resolution can find a contradiction in $S'$

There is a resolution proof for the contradiction in $S$
Summary

Inference in FOL

- **Propositionalisation**: very slow
- **Unification** techniques: much more efficient
- Generalised Modus Ponens: FC and BC on Definite clauses
  - FC for deductive databases
  - BC for logic programming
- Entailment problem is semi-decidable
- Generalised **resolution**: complete proof system (CNF)
<table>
<thead>
<tr>
<th>A dream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using theorem proving to automatically prove everything...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A first step</th>
</tr>
</thead>
<tbody>
<tr>
<td>To prove everything, we need to prove everything in arithmetic...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic for arithmetic: 0, S(·), ×, +, Expt (extension of FOL, more expressive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gödel’s incompleteness theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gödel said (after a proof on 30 pages):</td>
</tr>
<tr>
<td>“Whatever your logic is, if your logic can express arithmetic, whatever your KB is, I can exhibit a sentence in your logic such that the sentence is entailed by KB but there’s no way to prove it by inference thanks to your KB”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorry for the inconvenience...</td>
</tr>
</tbody>
</table>