Forward and Backward Chaining

Outline

I. Translation of sentences into FOL

II. Forward chaining

III. Backward chaining

IV. Logic programming & Prolog (optional)

V. Conversion into the Conjunctive Normal Form

* Figures are from the textbook site.
I. First-Order Definite Clauses

A first-order definite clause is a disjunction of literals of which exactly one is positive.
A first-order definite clause is a disjunction of literals of which exactly one is positive.

- single positive literal (i.e., fact in propositional logic if no variable)

Bird(Ostrich)
A *first-order definite clause* is a disjunction of literals of which exactly one is positive.

- single positive literal (i.e., fact in propositional logic if no variable)
  
  \[ \text{Bird(Ostrich)} \]

- an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal (i.e., definite clause in PL)
  
  \[ \text{Human(Socrates) } \Rightarrow \text{Fallible(Socrates)} \]
I. First-Order Definite Clauses

A *first-order definite clause* is a disjunction of literals of which exactly one is positive.

- single positive literal (i.e., fact in propositional logic if no variable)
  
  _Bird(Ostrich)_

- an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal (i.e., definite clause in PL)
  
  _Human(Socrates) ⇒ Fallible(Socrates)_

- variables allowed and implicitly under universal quantification
  
  _Human(x) ⇒ Fallible(x)_ // interpreted as ∀x _Human(x) ⇒ Fallible(x)_
I. First-Order Definite Clauses

A *first-order definite clause* is a disjunction of literals of which exactly one is positive.

- single positive literal (i.e., fact in propositional logic if no variable)
  
  $\text{Bird(Ostrich)}$

- an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal (i.e., definite clause in PL)
  
  $\text{Human(Socrates)} \Rightarrow \text{Fallible(Socrates)}$

- variables allowed and implicitly under universal quantification
  
  $\text{Human}(x) \Rightarrow \text{Fallible}(x)$ // interpreted as $\forall x \ \text{Human}(x) \Rightarrow \text{Fallible}(x)$

  $\text{Gate}(g) \land \text{Terminal}(t) \Rightarrow g \neq t$ // interpreted as $\forall g, t \ \text{Gate}(g) \land \text{Terminal}(t) \Rightarrow g \neq t$
I. First-Order Definite Clauses

A *first-order definite clause* is a disjunction of literals of which exactly one is positive.

- single positive literal (i.e., fact in propositional logic if no variable)

  \[ \text{Bird(Ostrich)} \]

- an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal (i.e., definite clause in PL)

  \[ \text{Human(Socrates)} \Rightarrow \text{Fallible(Socrates)} \]

- variables allowed and implicitly *under universal quantification*

  \[ \text{Human}(x) \Rightarrow \text{Fallible}(x) \]  // interpreted as \( \forall x \; \text{Human}(x) \Rightarrow \text{Fallible}(x) \)

  \[ \text{Gate}(g) \land \text{Terminal}(t) \Rightarrow g \neq t \]  // interpreted as \( \forall g, t \; \text{Gate}(g) \land \text{Terminal}(t) \Rightarrow g \neq t \)

- existential quantifiers *not* allowed
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”:
Translation of Sentences

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”:  // ∀x  ...
The law says that it is a crime for an American to sell weapons to hostile nations.
The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”: $\forall x \ldots$

$American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z) \Rightarrow Criminal(x)$
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”: \[ \forall x \ldots \]

\[
\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)
\]

“Nono … has some missiles”: 
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”:  \[ \forall x \ldots \]

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x) \]

“Nono … has some missiles”:  \[ \exists x \text{ Owns}(Nono, x) \land \text{Missile}(x) \]
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”:  // ∀x  ...

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x) \]

“Nono … has some missiles”:  // ∃x  \text{Owns}(Nono, x) \land \text{Missile}(x)

\[ \text{Owns}(\text{Nono}, M_1) \]

\[ \text{Missile}(M_1) \]

introducing a Skolem constant to eliminate ∃
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“… it is a crime for an American to sell weapons to hostile nations”:

\[
\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)
\]

“Nono … has some missiles”:

\[
\exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x)
\]

“All of its missiles were sold by Colonel West”:

\[
\forall x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x)
\]
The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

“... it is a crime for an American to sell weapons to hostile nations”: \[ \forall x \quad \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \implies \text{Criminal}(x) \]

“Nono ... has some missiles”: \[ \exists x \quad \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \]

“... all of its missiles were sold by Colonel West”: \[ \forall x \quad \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono}) \]
Completing the KB
Completing the KB

Need to know that missiles are weapons.

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
Completing the KB

Need to know that missiles are weapons.

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

Also need to know that an enemy of America counts as “hostile”.

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]
Completing the KB

Need to know that missiles are weapons.

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

Also need to know that an enemy of America counts as “hostile”.

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

“West, who is American …

\[ \text{American}(\text{West}) \]
Completing the KB

Need to know that missiles are weapons.

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

Also need to know that an enemy of America counts as “hostile”.

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

“West, who is American …

\[ \text{American}(\text{West}) \]

“The country Nono, an enemy of America …”

\[ \text{Enemy}(\text{Nono}, \text{America}) \]
Completing the KB

Need to know that missiles are weapons.

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

Also need to know that an enemy of America counts as “hostile”.

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

“West, who is American …

\[ \text{American}(\text{West}) \]

“The country Nono, an enemy of America …”

\[ \text{Enemy}(\text{Nono}, \text{America}) \]

The KB consists of first-order definite clauses with no function symbols. It is called a Datalog.
II. Simple Forward Chaining

1. Start from the known facts.

2. Trigger all the rules whose premises are satisfied.

3. Add their conclusions to the known facts.

4. Repeat steps 2 and 3 until one of the following situations occurs:
   
   a. The query is answered.
   
   b. No new facts are added.
II. Simple Forward Chaining

1. Start from the known facts.

2. Trigger all the rules whose premises are satisfied.

3. Add their conclusions to the known facts.

4. Repeat steps 2 and 3 until one of the following situations occurs:
   a. The query is answered.
   b. No new facts are added.

A new fact is not a renaming of a known fact.
II. Simple Forward Chaining

1. Start from the known facts.

2. Trigger all the rules whose premises are satisfied.

3. Add their conclusions to the known facts.

4. Repeat steps 2 and 3 until one of the following situations occurs:
   a. The query is answered.
   b. No new facts are added.

A *new fact* is not a renaming of a known fact.

*Likes(x, IceCream)* is a renaming of *Likes(y, IceCream)*. Both have the meaning: “Everyone likes ice cream”.
Execution of Forward Chaining

KB:

\[\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)\]

\[\text{Owns}(\text{Nono}, M_1)\]

\[\text{Missile}(M_1)\]

\[\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})\]

\[\text{Missile}(x) \Rightarrow \text{Weapon}(x)\]

\[\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)\]

\[\text{American}(\text{West})\]

\[\text{Enemy}(\text{Nono}, \text{America})\]
Execution of Forward Chaining

**KB:**

- \( \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x) \)
- \( \text{Owns}(\text{Nono}, M_1) \)
- \( \text{Missile}(M_1) \)
- \( \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \)
- \( \text{Missile}(x) \Rightarrow \text{Weapon}(x) \)
- \( \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \)
- \( \text{American}(\text{West}) \)
- \( \text{Enemy}(\text{Nono}, \text{America}) \)

Iteration 1 adds:
Execution of Forward Chaining

**KB:**

- $\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)$
- $\text{Owns}(\text{Nono}, M_1)$
- $\text{Missile}(M_1)$
- $\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
- $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
- $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$
- $\text{American}(\text{West})$
- $\text{Enemy}(\text{Nono}, \text{America})$

Iteration 1 adds:

- $\text{Sell}(\text{West}, M_1, \text{Nono})$
Execution of Forward Chaining

**KB:**

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x,y,z) \Rightarrow \text{Criminal}(x) \]

\[ \text{Owns}(\text{Nono}, M_1) \]

\[ \text{Missile}(M_1) \]

\[ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

\[ \text{American}(\text{West}) \]

\[ \text{Enemy}(\text{Nono}, \text{America}) \]

**Iteration 1 adds:**

\[ \text{Sell}(\text{West}, M_1, \text{Nono}) \]

\[ \text{Weapon}(M_1) \]
Execution of Forward Chaining

**KB:**

\[
\begin{align*}
\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) & \Rightarrow \text{Criminal}(x) \\
\text{Owns}(\text{Nono}, M_1) \\
\text{Missile}(M_1) \\
\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) & \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \\
\text{Missile}(x) & \Rightarrow \text{Weapon}(x) \\
\text{Enemy}(x, \text{America}) & \Rightarrow \text{Hostile}(x) \\
\text{American}(\text{West}) \\
\text{Enemy}(\text{Nono}, \text{America})
\end{align*}
\]

**Iteration 1 adds:**

\[
\begin{align*}
\text{Sell}(\text{West}, M_1, \text{Nono}) \\
\text{Weapon}(M_1) \\
\text{Hostile}(\text{Nono})
\end{align*}
\]
Execution of Forward Chaining

KB:

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x) \]

\[ \text{Owns}(\text{Nono}, M_1) \]

\[ \text{Missile}(M_1) \]

\[ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

\[ \text{American}(\text{West}) \]

\[ \text{Enemy}(\text{Nono}, \text{America}) \]

Iteration 1 adds:

\[ \text{Sell}(\text{West}, M_1, \text{Nono}) \]

\[ \text{Weapon}(M_1) \]

\[ \text{Hostile}(\text{Nono}) \]

Iteration 2 adds:
Execution of Forward Chaining

**KB:**

\[
\begin{align*}
&\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x) \\
&\text{Owns}(\text{Nono}, M_1) \\
&\text{Missile}(M_1) \\
&\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \\
&\text{Missile}(x) \Rightarrow \text{Weapon}(x) \\
&\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \\
&\text{American}(\text{West}) \\
&\text{Enemy}(\text{Nono}, \text{America})
\end{align*}
\]

**Iteration 1 adds:**

\[
\begin{align*}
&\text{Sell}(\text{West}, M_1, \text{Nono}) \\
&\text{Weapon}(M_1) \\
&\text{Hostile}(\text{Nono})
\end{align*}
\]

**Iteration 2 adds:**

\[
\begin{align*}
&\text{Criminal}(\text{West})
\end{align*}
\]
Execution of Forward Chaining

**KB:**

- \(\text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)\)
- \(\text{Owns}(\text{Nono}, M_1)\)
- \(\text{Missile}(M_1)\)
- \(\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})\)
- \(\text{Missile}(x) \Rightarrow \text{Weapon}(x)\)
- \(\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)\)
- \(\text{American}(\text{West})\)
- \(\text{Enemy}(\text{Nono}, \text{America})\)

Iteration 1 adds:
- \(\text{Sell}(\text{West}, M_1, \text{Nono})\)
- \(\text{Weapon}(M_1)\)
- \(\text{Hostile}(\text{Nono})\)

Iteration 2 adds:
- \(\text{Criminal}(\text{West})\)

KB has now reached a **fixed point**, meaning that no new sentences are possible.
Proof Tree

- **American(West)**
- **Missile(M₁)**
- **Weapon(M₁)**
- **Sells(West,M₁,Nono)**
- **Owns(Nono,M₁)**
- **Hostile(Nono)**
- **Enemy(Nono,America)**
Proof Tree

Soundness of forward chaining

Every inference is an application of Generalized Modus Ponens.
Soundness of forward chaining

Every inference is an application of Generalized Modus Ponens.

Completeness

- Easy to establish if no function symbols appears in the KB.
Soundness of forward chaining

Every inference is an application of Generalized Modus Ponens.

Completeness

- Easy to establish if no function symbols appear in the KB.
- Guaranteed except for a query with no answer, if function symbols appear in the KB.
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- ♠ Exhaustively matches every rule against every fact.
- ♠ Rechecks every rule on each iteration (even with very few additions to $KB$).
- ♠ Generates many facts that are irrelevant to the goal.
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

$Missile(x) \land Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

Missile($x$) $\land$ Owns(Nono, $x$) $\Rightarrow$ Sells(West, $x$, Nono)

- Suppose Nono owns many objects among which very few are missiles. They are two approaches:
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

$$\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$$

- Suppose Nono owns many objects among which very few are missiles. They are two approaches:
  - Find all the objects owned by Nono and, for each, check if it is a missile.
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

$\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

- Suppose Nono owns many objects among which very few are missiles. They are two approaches:
  - Find all the objects owned by Nono and, for each, check if it is a missile.
  - Find all the missiles first and check if they are owned by Nono.
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

\[ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

Suppose Nono owns many objects among which very few are missiles. They are two approaches:

- Find all the objects owned by Nono and, for each, check if it is a missile.
- Find all the missiles first and check if they are owned by Nono. More efficient!
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

$\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

- Suppose Nono owns many objects among which very few are missiles. They are two approaches:
  - Find all the objects owned by Nono and, for each, check if it is a missile.
  - Find all the missiles first and check if they are owned by Nono. More efficient!

- How to order the conjuncts of the rule premise so they can be solved with the minimum total cost?
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to $KB$).
- Generates many facts that are irrelevant to the goal.

$\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

- Suppose Nono owns many objects among which very few are missiles. They are two approaches:
  - Find all the objects owned by Nono and, for each, check if it is a missile.
  - Find all the missiles first and check if they are owned by Nono. More efficient!

- How to order the conjuncts of the rule premise so they can be solved with the minimum total cost?

NP-hard!
Improvement 1: Matching Rules Against Known Facts

Inefficiency of simple forward chaining:

- Exhaustively matches every rule against every fact.
- Rechecks every rule on each iteration (even with very few additions to KB).
- Generates many facts that are irrelevant to the goal.

\[ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

- Suppose Nono owns many objects among which very few are missiles. They are two approaches:
  - Find all the objects owned by Nono and, for each, check if it is a missile.
  - Find all the missiles first and check if they are owned by Nono. More efficient!

- How to order the conjuncts of the rule premise so they can be solved with the minimum total cost?

  NP-hard! Use a heuristic, e.g., the minimum-remaining-values (MRV) heuristic for CSPs.
CSP as a Definite Clause

View every conjunct in the premise as a constraint on the variables it contains.

Diff(wa, nt) \land Diff(wa, sa) \land 
Diff(nt, q) \land Diff(nt, sa) \land 
Diff(q, nsw) \land Diff(q, sa) \land 
Diff(nsw, v) \land Diff(nsw, sa) \land 
Diff(v, sa) \Rightarrow Colorable()

Diff(\text{Red}, \text{Blue}) \quad \text{Diff(\text{Red}, \text{Green})}
\quad \text{Diff(\text{Green}, \text{Red})} \quad \text{Diff(\text{Green}, \text{Blue})}
\quad \text{Diff(\text{Blue}, \text{Red})} \quad \text{Diff(\text{Blue}, \text{Green})}

Graph coloring
CSP as a Definite Clause

View every conjunct in the premise as a constraint on the variables it contains.

Graph coloring

Constraint satisfaction is NP-hard.
CSP as a Definite Clause

View every conjunct in the premise as a constraint on the variables it contains.

```
Diff(wa, nt) ∧ Diff(wa, sa) ∧
Diff(nt, q) ∧ Diff(nt, sa) ∧
Diff(q, nsw) ∧ Diff(q, sa) ∧
Diff(nsw, v) ∧ Diff(nsw, sa) ∧
Diff(v, sa) ⇒ Colorable()
```

Graph coloring

Constraint satisfaction is NP-hard.

Matching a definite clause against a set of facts is NP-hard.
CSP as a Definite Clause

View every conjunct in the premise as a constraint on the variables it contains.

\[ \text{Diff}(wa, nt) \land \text{Diff}(wa, sa) \land \text{Diff}(nt, q) \land \text{Diff}(nt, sa) \land \text{Diff}(q, nsw) \land \text{Diff}(q, sa) \land \text{Diff}(nsw, v) \land \text{Diff}(nsw, sa) \land \text{Diff}(v, sa) \Rightarrow \text{Colorable}() \]

\[ \text{Diff}(\text{Red}, \text{Blue}) \land \text{Diff}(\text{Red}, \text{Green}) \land \text{Diff}(\text{Green}, \text{Red}) \land \text{Diff}(\text{Green}, \text{Blue}) \land \text{Diff}(\text{Blue}, \text{Red}) \land \text{Diff}(\text{Blue}, \text{Green}) \]

Constraint satisfaction is NP-hard.

Matching a definite clause against a set of facts is NP-hard.

Good news View every Datalog clause as a CSP and apply heuristics for CSPs (e.g., tree structure, cutset conditioning, etc.).
Improvement 2: Incremental FC

Observations

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.
Improvement 2: Incremental FC

**Observations**

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.

Incremental forward chaining does the following during iteration $i$: 
Improvement 2: Incremental FC

Observations

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.

Incremental forward chaining does the following during iteration $i$:

1. Check a rule only if its premise includes a conjunct $p_i$ that unifies with a fact $p_i'$ inferred at iteration $i - 1$. 
Improvement 2: Incremental FC

Observations

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.

Incremental forward chaining does the following during iteration $i$:

1. Check a rule only if its premise includes a conjunct $p_i$ that unifies with a fact $p_i'$ inferred at iteration $i - 1$.
2. Extends the substitution to match $p_i$ with $p_i'$. 
Improvement 2: Incremental FC

Observations

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.

Incremental forward chaining does the following during iteration $i$:

1. Check a rule only if its premise includes a conjunct $p_i$ that unifies with a fact $p_i'$ inferred at iteration $i - 1$.
2. Extends the substitution to match $p_i$ with $p_i'$.
3. Repeat for every such conjunct in the premise of the same rule.
Improvement 2: Incremental FC

Observations

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.

Incremental forward chaining does the following during iteration $i$:

1. Check a rule only if its premise includes a conjunct $p_i$ that unifies with a fact $p_i'$ inferred at iteration $i - 1$.
2. Extends the substitution to match $p_i$ with $p_i'$.
3. Repeat for every such conjunct in the premise of the same rule.
4. The remaining conjuncts are matched with facts from iterations before $i - 1$. 
Improvement 2: Incremental FC

Observations

- Every new fact inferred on iteration $i$ must be derived from at least one new fact inferred on iteration $i - 1$.
- Only a small fraction of the rules are triggered by a fact.

Incremental forward chaining does the following during iteration $i$:

1. Check a rule only if its premise includes a conjunct $p_i$ that unifies with a fact $p_i'$ inferred at iteration $i - 1$.
2. Extends the substitution to match $p_i$ with $p_i'$.
3. Repeat for every such conjunct in the premise of the same rule.
4. The remaining conjuncts are matched with facts from iterations before $i - 1$.

E.g., the Rete algorithm
III. Backward Chaining

Works like AND/OR search:

- **OR**
  - ♠ The goal query can be proved by any rule in the $KB$.
  - ♠ A query containing a variable, e.g., $Person(x)$ can be proved in multiple ways.

- **AND**: all the conjuncts in the premise of a clause must be proved.
III. Backward Chaining

Works like AND/OR search:

- **OR**
  - The goal query can be proved by any rule in the *KB*.
  - A query containing a variable, e.g., *Person(x)* can be proved in multiple ways.

- **AND**: all the conjuncts in the premise of a clause must be proved.

How does it work?
III. Backward Chaining

Works like AND/OR search:

- **OR**
  - The goal query can be proved by any rule in the $KB$.
  - A query containing a variable, e.g., $Person(x)$ can be proved in multiple ways.

- **AND**: all the conjuncts in the premise of a clause must be proved.

How does it work?

- Fetch all clauses that unify with the goal.

- Rename the variables in every such clause to be brand-new.

- Prove every conjunct in the clause by keeping track of the expanded substitution as it goes.
\begin{align*}
\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) & \Rightarrow \text{Criminal}(x) \\
\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) & \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \\
\text{Missile}(x) & \Rightarrow \text{Weapon}(x) \\
\text{Enemy}(x, \text{America}) & \Rightarrow \text{Hostile}(x)
\end{align*}
Atomic fact is considered as a clause whose left-hand side is an empty list.
Algorith \( = \) Logic + Control \hfill (Robert Kowalski)

Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compilers, parsing natural languages)
IV. Logic Programming (Optional)

Algorithm = Logic + Control  

(Robert Kowalski)

Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compilers, parsing natural languages)

✦ A Prolog program is a set of definite clauses.
Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compliers, parsing natural languages)

♦ A Prolog program is a set of definite clauses.

```prolog
// American(x) ∧ Weapon(y) ∧ Hostile(z) ∧ Sells(x, y, z) ⇒ Criminal(x)
```
IV. Logic Programming (Optional)

Algorithm = Logic + Control  
(Robert Kowalski)

Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compilers, parsing natural languages)

- A Prolog program is a set of definite clauses.

// American(x) ∧ Weapon(y) ∧ Hostile(z) ∧ Sells(x, y, z) ⇒ Criminal(x)

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
IV. Logic Programming (Optional)

Algorithm = Logic + Control  
(Robert Kowalski)

Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compilers, parsing natural languages)

A Prolog program is a set of definite clauses.

```prolog
// American(X) ∧ Weapon(Y) ∧ Hostile(Z) ∧ Sells(X,Y,Z) ⇒ Criminal(X)
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

\[ \rightarrow \]
IV. Logic Programming (Optional)

Algorithm = Logic + Control  (Robert Kowalski)

Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compliers, parsing natural languages)

A Prolog program is a set of definite clauses.

```
// American(x) ∧ Weapon(y) ∧ Hostile(z) ∧ Sells(x, y, z) ⇒ Criminal(x)

criminal(X) :- american(X), weapon(Y), sells(X, Y, Z), hostile(Z).
```
Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compilers, parsing natural languages)

A Prolog program is a set of definite clauses.

```prolog
// American(x) ∧ Weapon(y) ∧ Hostile(z) ∧ Sells(x, y, z) ⇒ Criminal(x)
criminal(X) :- american(X), weapon(Y), sells(X, Y, Z), hostile(Z).
```

- uppercase letters for variables
IV. Logic Programming (Optional)

*Algorithm = Logic + Control*  
(Robert Kowalski)

Prolog (1972) is the most widely used logic programming language.

- Rapid prototyping
- Symbolic manipulation (e.g., writing compilers, parsing natural languages)

♦ A Prolog program is a set of definite clauses.

```
// American(x) ∧ Weapon(y) ∧ Hostile(z) ∧ Sells(x, y, z) ⇒ Criminal(x)

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- uppercase letters for variables
- end of a clause
Backward Chaining in Prolog

- Prolog recursively defines a function.
Backward Chaining in Prolog

- Prolog recursively defines a function.

**Example** List appending.

```prolog
append([], Y, Y).
append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```
Backward Chaining in Prolog

Prolog recursively defines a function.

**Example** List appending.

```prolog
// appending the empty list and the list Y produces the same list Y.
append([], Y, Y).

append([A|X], Y, [A|Z]) :- append(X, Y, Z)
```
Backward Chaining in Prolog

Prolog recursively defines a function.

**Example** List appending.

// appending the empty list and the list Y produces the same list Y.

```prolog
append([], Y, Y).
```

```prolog
append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

a list whose first element is A and rest is X.
Prolog recursively defines a function.

**Example** List appending.

- appending the empty list and the list Y produces the same list Y.

\[
\text{append}([], Y, Y).
\]

- [A | Z] is the result of appending [A | X] and Y provided that Z is the result of appending X and Y.

\[
\text{append}([A|X], Y, [A|Z]) :- \text{append}(X, Y, Z)
\]

a list whose first element is A and rest is X.
Prolog recursively defines a function.

**Example** List appending.

\[
\text{append}([], Y, Y).
\]

// appending the empty list and the list Y produces the same list Y.

\[
\text{append}([A|X], Y, [A|Z]) : - \text{append}(X,Y,Z).
\]

// [A | Z] is the result of appending [A | X] and Y provided that Z is
// the result of appending X and Y.

a list whose first element
is A and rest is X.

Describes the relations among the three arguments of append.
Query Example

(1) \texttt{append([], Y, Y).}

(2) \texttt{append([A|X], Y, [A|Z]) :- append(X,Y,Z)}

\textbf{Query:} \texttt{append(X, Y, [1, 2, 3])}
Query Example

(1)  `append([], Y, Y).`

(2)  `append([A|X], Y, [A|Z]) :- append(X,Y,Z)`

**Query**: `append(X, Y, [1, 2, 3])`

Solutions returned by Prolog:

`X=[]      Y=[1,2,3]  // matches (1)`
Query Example

(1)  append([], Y, Y).

(2)  append( [A|X], Y, [A|Z] ) :- append(X,Y,Z)

Query: append(X, Y, [1, 2, 3])

Solutions returned by Prolog:

X=[ ]   Y=[1,2,3]  // matches (1)
X=[1]   Y=[2,3]   // matches (2) to obtain substitution {A/1}, and then
// matches append(X, Y, [2, 3]) against (1).
Query Example

(1) append([], Y, Y).

(2) append([A|X], Y, [A|Z]) :- append(X, Y, Z)

**Query**: append(X, Y, [1, 2, 3])

Solutions returned by Prolog:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>[1,2,3]</td>
<td>matches (1)</td>
</tr>
<tr>
<td>[1]</td>
<td>[2,3]</td>
<td>matches (2) to obtain substitution {A/1}, and then matches append(X, Y, [2, 3]) against (1).</td>
</tr>
<tr>
<td>[1,2]</td>
<td>[3]</td>
<td>applies (2) twice and then (1).</td>
</tr>
</tbody>
</table>
Query Example

(1)  \text{append}([], Y, Y).
(2)  \text{append}([A|X], Y, [A|Z]) :- \text{append}(X, Y, Z)

Query: \text{append}(X, Y, [1, 2, 3])

Solutions returned by Prolog:

\begin{align*}
X=| & \quad Y=[1, 2, 3] & \text{// matches (1)} \\
X=[1] & \quad Y=[2, 3] & \text{// matches (2) to obtain substitution \{A/1\}, and then} \\
& \quad \quad \quad \quad \quad \text{// matches } \text{append}(X, Y, [2, 3]) \text{ against (1).} \\
X=[1, 2] & \quad Y=[3] & \text{// applies (2) twice and then (1).} \\
X=[1, 2, 3] & \quad Y=| & \text{// applies (2) thrice and then (1).}
\end{align*}
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

\[
\text{path}(X, Z) :- \text{link}(X, Z).
\]
\[
\text{path}(X, Z) :- \text{path}(X, Y), \text{link}(Y, Z).
\]
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

\[
\text{path}(X,Z) :- \text{link}(X, Z).
\]

\[
\text{path}(X,Z) :- \text{path}(X,Y), \text{link}(Y,Z).
\]
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

\[
\text{path}(X,Z) \leftarrow \text{link}(X, Z).
\]

\[
\text{path}(X,Z) \leftarrow \text{path}(X,Y), \text{link}(Y,Z).
\]
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

path(X,Z) :- link(X, Z).
path(X,Z) :- path(X,Y), link(Y,Z).

Query  path(a, c)
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

\[
\text{path}(X,Z) : \text{link}(X, Z).
\]

\[
\text{path}(X,Z) : \text{path}(X,Y), \text{link}(Y,Z).
\]

**Query** \(\text{path}(a, c)\)
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

\[
\text{path}(X,Z) : \text{link}(X, Z).
\]
\[
\text{path}(X,Z) : \text{path}(X,Y), \text{link}(Y,Z).
\]
\[
\text{path}(X,Z) : \text{link}(X, Z).
\]

Query \ \text{path}(a, c)
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

path(X,Z) :- link(X, Z).
path(X,Z) :- path(X,Y), link(Y,Z).

Query path(a, c)
Infinite Loop

Finds if a path exists between two nodes in a directed graph.

path(X, Z) :- link(X, Z).
path(X, Z) :- path(X, Y), link(Y, Z).

Query path(a, c)

link(A, B) link(B, C)

Infinite loop!
Redundant Inference

\[
\text{path}(X,Z) \triangleq \text{link}(X, Z).
\]

\[
\text{path}(X,Z) \triangleq \text{path}(X,Y), \text{link}(Y,Z).
\]
Redundant Inference

\[
\text{path}(X,Z) : - \text{link}(X, Z).
\]
\[
\text{path}(X,Z) : - \text{path}(X,Y), \text{link}(Y,Z).
\]

Query \( \text{path}(A_1, J_4) \)
Redundant Inference

\[
\text{path}(X,Z) :\ - \ 
\text{link}(X, Z).
\]
\[
\text{path}(X,Z) :\ - \ 
\text{path}(X,Y), \ 
\text{link}(Y,Z).
\]

**Query**  \( \text{path}(A1, J4) \)

- Prolog performs 877 inferences (most of which involve nodes from which the goal is unreachable).
Redundant Inference

\[ \text{path}(X, Z) :\ - \text{link}(X, Z). \]
\[ \text{path}(X, Z) :\ - \text{path}(X, Y), \text{link}(Y, Z). \]

**Query** \[ \text{path}(A1, J4) \]

- Prolog performs 877 inferences (most of which involve nodes from which the goal is unreachable).
- Forward chaining performs only 62 inferences.
Redundant Inference

\[
\text{path}(X,Z) :\ - \ \text{link}(X, Z).
\]
\[
\text{path}(X,Z) :\ - \ \text{path}(X,Y), \ \text{link}(Y,Z).
\]

Query \ path(A1, J4)

- Prolog performs 877 inferences (most of which involve nodes from which the goal is unreachable).

- Forward chaining performs only 62 inferences.
V. Resolution in FOL

- Forward and backward chaining work with definite clauses only.

- Resolution works for any knowledge base.

- Before using resolution, we need to convert FOL sentences in the KB into the conjunctive normal form.
Conjunctive Normal Form (CNF)

Before inference, we need to convert sentences to CNF.

\[(l_{11} \lor l_{12} \lor \cdots \lor l_{1n_1}) \land \cdots \land (l_{k1} \lor l_{k2} \lor \cdots \lor l_{kn_k})\]

Literals can contain variables (assumed to be universally quantified).
Conjunctive Normal Form (CNF)

Before inference, we need to convert sentences to CNF.

$$(l_{11} \lor l_{12} \lor \cdots \lor l_{1n_1}) \land \cdots \land (l_{k1} \lor l_{k2} \lor \cdots \lor l_{kn_k})$$

Literals can contain variables (assumed to be universally quantified).

$$\forall x, y, z \ American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z) \Rightarrow Criminal(x)$$
Before inference, we need to convert sentences to CNF.

\[(l_{11} \lor l_{12} \lor \cdots \lor l_{1n_1}) \land \cdots \land (l_{k1} \lor l_{k2} \lor \cdots \lor l_{kn_k})\]

Literals can contain variables (assumed to be universally quantified).

\[\forall x, y, z \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)\]

\[\downarrow\]

\[\neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Hostile}(z) \lor \neg \text{Sells}(x, y, z) \lor \text{Criminal}(x)\]
Conjunctive Normal Form (CNF)

Before inference, we need to convert sentences to CNF.

\[(l_{11} \lor l_{12} \lor \cdots \lor l_{1n_1}) \land \cdots \land (l_{k1} \lor l_{k2} \lor \cdots \lor l_{kn_k})\]

 Literals can contain variables (assumed to be universally quantified).

\[\forall x, y, z \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)\]

\[\downarrow\]

\[\neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Hostile}(z) \lor \neg \text{Sells}(x, y, z) \lor \text{Criminal}(x)\]

• Every FOL sentence can be converted into an inferentially equivalent CNF sentence.
Conjunctive Normal Form (CNF)

Before inference, we need to convert sentences to CNF.

$$(l_{11} \lor l_{12} \lor \cdots \lor l_{1n_1}) \land \cdots \land (l_{k1} \lor l_{k2} \lor \cdots \lor l_{kn_k})$$

Literals can contain variables (assumed to be universally quantified).

$$\forall x, y, z \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x)$$

$$\Downarrow$$

$$\neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Hostile}(z) \lor \neg \text{Sells}(x, y, z) \lor \text{Criminal}(x)$$

- Every FOL sentence can be converted into an inferentially equivalent CNF sentence.

- The conversion procedure is similar to the propositional logic case, except for the need to eliminate $\exists$. 
Conversion to CNF

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

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

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

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

Conversion steps:

a) Eliminate implications: Replace $P \Rightarrow Q$ with $\neg P \lor Q$. 
Conversion to CNF

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

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

Conversion steps:

a) Eliminate implications: Replace \( P \Rightarrow Q \) with \( \neg P \lor Q \).

\[ \forall x \ \neg (\forall y \ \text{Animal}(y) \Rightarrow \text{Loves}(x, y)) \lor (\exists y \ \text{Loves}(y, x)) \]
Conversion to CNF

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

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

Conversion steps:

a) Eliminate implications: Replace \( P \Rightarrow Q \) with \( \neg P \lor Q \).

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

\[
\forall x \neg (\forall y \neg \text{Animal}(y) \lor \text{Loves}(x, y)) \lor (\exists y \text{Loves}(y, x))
\]
Conversion to CNF

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

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

Conversion steps:

a) Eliminate implications: Replace $$P \Rightarrow Q$$ with $$\neg P \lor Q$$.

$$\forall x \ \neg(\forall y \ Animal(y) \Rightarrow Loves(x, y)) \lor (\exists y \ Loves(y, x))$$

b) Move $$\neg$$ inward:
Conversion to CNF

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

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

Conversion steps:

a) Eliminate implications: Replace $P \Rightarrow Q$ with $\neg P \lor Q$.

∀x ¬(∀y Animal(y) ⇒ Loves(x, y)) ∨ (∃y Loves(y, x))

b) Move $\neg$ inward:

¬∀x P
¬∃x P
Conversion to CNF

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

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

Conversion steps:

a) Eliminate implications: Replace \( P \Rightarrow Q \) with \( \neg P \lor Q \).

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

b) Move \( \neg \) inward:

\[ \neg \forall x P \iff \exists x \neg P \]
\[ \neg \exists x P \iff \forall x \neg P \]
Moving \neg \text{ Inward}

\forall x \neg (\forall y \neg \text{Animal}(y) \lor \text{Loves}(x, y)) \lor (\exists y \text{Loves}(y, x))
Moving \rightarrow \text{ Inward}

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

\[ \forall x \ (\exists y \neg (\neg \text{Animal}(y) \lor \text{Loves}(x, y))) \lor (\exists y \text{Loves}(y, x)) \]
Moving $\neg$ Inward

$$\forall x \neg (\forall y \neg \text{Animal}(y) \lor \text{Loves}(x, y)) \lor (\exists y \text{Loves}(y, x))$$

$$\downarrow$$

$$\forall x (\exists y \neg (\neg \text{Animal}(y) \lor \text{Loves}(x, y)) \lor (\exists y \text{Loves}(y, x)))$$

$$\downarrow$$

$$\forall x (\exists y \neg \neg \text{Animal}(y) \land \neg \text{Loves}(x, y)) \lor (\exists y \text{Loves}(y, x))$$
Moving $\nrightarrow$ Inward

$$
\forall x \ (\exists y \ (\neg Animal(y) \land \neg Loves(x, y)) \lor (\exists y \ Loves(y, x)))
$$
Moving ¬ Inward

∀x (¬(∀y ¬Animal(y) ∨ Loves(x, y)) ∨ (∃y Loves(y, x)))

∀x (∃y ¬(¬Animal(y) ∨ Loves(x, y))) ∨ (∃y Loves(y, x))

∀x (∃y ¬¬Animal(y) ∧ ¬Loves(x, y)) ∨ (∃y Loves(y, x))

∀x (∃y Animal(y) ∧ ¬Loves(x, y)) ∨ (∃y Loves(y, x))

“Either there is some animal a person doesn’t love, or (otherwise) someone loves that person.”
"Either there is some animal a person doesn’t love, or (otherwise) someone loves that person."

"Everyone who loves all animals is loved by someone."
Variable Standardization

C) Standardize variables

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists y \ Loves(y, x)) \]
Variable Standardization

C) Standardize variables

$$\forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists y \ Loves(y, x))$$

Change the name of one of $y$ and $y$ to avoid confusion later when we drop the quantifiers.
C) Standardize variables

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists y \ Loves(y, x)) \]

Change the name of one of \(y\) and \(y\) to avoid confusion later when we drop the quantifiers.

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z \ Loves(z, x)) \]
d) Skolemize:

$$
\forall x (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z Loves(z, x))
$$
d) Skolemize:

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y) ) \lor (\exists z \ Loves(z, x)) \]

1\textsuperscript{st} try: introduce constants

\( A \) and \( B \) respectively for \( y \) and \( z \).
Skolemization

d) Skolemize:

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z Loves(z, x)) \]

1\textsuperscript{st} try: introduce constants 
A and B respectively for y and z.

\[ \forall x \ (Animal(A) \land \neg Loves(x, A)) \lor Loves(B, x) \]
d) Skolemize:

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z \ Loves(z, x)) \]

1\textsuperscript{st} try: introduce constants 
A and B respectively for y and z.

\[ \forall x \ (Animal(A) \land \neg Loves(x, A)) \lor Loves(B, x) \]

“Everyone either fails to love an animal A or is loved by some particular entity B.”
d) Skolemize:

∀𝑥 (∃𝑦 Animal(𝑦) ∧ ¬Loves(𝑥, 𝑦)) ∨ (∃𝑧 Loves(𝑧, 𝑥))

1st try: introduce constants
A and B respectively for 𝑦 and 𝑧.

∀𝑥 (Animal(A) ∧ ¬Loves(𝑥, A)) ∨ Loves(B, 𝑥)

“Everyone either fails to love an animal A or is loved by some particular entity B.”
d) Skolemize:

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z \ Loves(z, x)) \]

1st try: introduce constants A and B respectively for y and z.

\[ \forall x \ (Animal(A) \land \neg Loves(x, A)) \lor Loves(B, x) \]

“Everyone either fails to love an animal A or is loved by some particular entity B.”

Both y and z depends on x, and in different ways.
∀ x (∃ y Animal(y) ∧ ¬ Loves(x, y)) ∨ (∃ z Loves(z, x))
Skolemization (cont’d)

∀x (∃y Animal(y) ∧ ¬Loves(x, y)) ∨ (∃z Loves(z, x))

2\textsuperscript{nd} try: introduce Skolem functions 

F(x) and G(x) respectively for y and z.
Skolemization (cont’d)

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z \ Loves(z, x)) \]

2nd try: introduce Skolem functions 
\( F(x) \) and \( G(x) \) respectively for \( y \) and \( z \).

\[ \forall x \ (Animal(F(x)) \land \neg Loves(x, F(x))) \lor Loves(G(x), x) \]
Skolemization (cont’d)

∀x (∃y Animal(y) ∧ ¬Loves(x, y)) ∨ (∃z Loves(z, x))

2nd try: introduce Skolem functions F(x) and G(x) respectively for y and z.

∀x (Animal(F(x)) ∧ ¬Loves(x, F(x))) ∨ Loves(G(x), x)

General case:

∀x₁, ..., xₙ∃y P(y, x₁, ..., xₙ)  // y depends on x₁, ..., xₙ
Skolemization (cont’d)

\[ \forall x \ (\exists y \ Animal(y) \land \neg Loves(x, y)) \lor (\exists z \ Loves(z, x)) \]

\[ 2^{\text{nd try}}: \text{introduce Skolem functions} \\
\ F(x) \text{ and } G(x) \text{ respectively for } y \text{ and } z. \]

\[ \forall x \ (\Animal(F(x)) \land \neg Loves(x, F(x))) \lor Loves(G(x), x) \]

General case:

\[ \forall x_1, \ldots, x_n \exists y \ P(y, x_1, \ldots, x_n) \]

\[ \text{eliminate } y \text{ by introducing function } f \]

// y depends on \( x_1, \ldots, x_n \)
Skolemization (cont’d)

General case:

\[ \forall x \left( \exists y \ Animal(y) \land \neg Loves(x, y) \right) \lor \left( \exists z \ Loves(z, x) \right) \]

2nd try: introduce Skolem functions \( F(x) \) and \( G(x) \) respectively for \( y \) and \( z \).

\[ \forall x \left( Animal(F(x)) \land \neg Loves(x, F(x)) \right) \lor Loves(G(x), x) \]

\[ \forall x_1, \ldots, x_n \exists y \ P(y, x_1, \ldots, x_n) \quad \text{// } y \text{ depends on } x_1, \ldots, x_n \]

eliminate \( y \) by introducing function \( f \)

\[ P(f(x_1, \ldots, x_n), x_1, \ldots, x_n) \]
Skolemization – One More Example

\[ \exists s \forall t \forall u \forall v \forall w \forall x \forall y \exists z \ P(s, t, u, v, w, x, y, z) \]
Skolemization – One More Example

\[ \exists s \forall u \forall v \forall w \forall x \forall y \forall z \ P(s, t, u, v, w, x, y, z) \]

Replace \( s \) with a constant \( C_1 \) (i.e., a function with no argument).

\[ \exists t \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, t, u, v, w, x, y, z) \]
Skolemization – One More Example

$$\exists s \forall t \forall u \forall v \forall w \forall x \forall y \forall z \ P(s, t, u, v, w, x, y, z)$$

Replace $s$ with a constant $C_1$ (i.e., a function with no argument).

$$\exists t \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, t, u, v, w, x, y, z)$$

Replace $t$ with another constant $C_2$. ($t$ depends $s$ and is a function of $C_1$. It is thus a constant as well.)

$$\forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z)$$
Skolemization – One More Example

\[ \exists s \forall u \forall v \forall w \forall x \forall y \forall z \ P(s, t, u, v, w, x, y, z) \]

- Replace \( s \) with a constant \( C_1 \) (i.e., a function with no argument).

\[ \exists t \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, t, u, v, w, x, y, z) \]

- Replace \( t \) with another constant \( C_2 \). (\( t \) depends on \( s \) and is a function of \( C_1 \). It is thus a constant as well.)

\[ \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z) \]

- Eliminate the two universal quantifiers in front of \( u \) and \( v \).

\[ \exists w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z) \]
Skolemization – One More Example

\[ \exists s \exists t \forall u \forall v \exists w \forall x \forall y \exists z \ P(s, t, u, v, w, x, y, z) \]

- Replace \( s \) with a constant \( C_1 \) (i.e., a function with no argument).

\[ \exists t \forall u \forall v \exists w \forall x \forall y \exists z \ P(C_1, t, u, v, w, x, y, z) \]

- Replace \( t \) with another constant \( C_2 \). \((t \) depends on \( s \) and is a function of \( C_1 \). It is thus a constant as well.\)

\[ \forall u \forall v \exists w \forall x \forall y \exists z \ P(C_1, C_2, u, v, w, x, y, z) \]

- Eliminate the two universal quantifiers in front of \( u \) and \( v \).

\[ \exists w \forall x \forall y \exists z \ P(C_1, C_2, u, v, w, x, y, z) \]

- \( w \) depends on \( C_1, C_2, u, v \), among which only \( u, v \) are variables. Introduce a Skolem function \( f_1 \).

\[ \forall x \forall y \exists z \ P(C_1, C_2, u, v, f_1(u, v), x, y, z) \]
Skolemization – One More Example

\[ \exists s \forall u \forall v \forall w \forall x \forall y \forall z \ P(s, t, u, v, w, x, y, z) \]

Replace \( s \) with a constant \( C_1 \) (i.e., a function with no argument).

\[ \exists t \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, t, u, v, w, x, y, z) \]

Replace \( t \) with another constant \( C_2 \). (\( t \) depends on \( s \) and is a function of \( C_1 \). It is thus a constant as well.)

\[ \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z) \]

Eliminate the two universal quantifiers in front of \( u \) and \( v \).

\[ \exists w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z) \]

\( w \) depends on \( C_1, C_2, u, v \), among which only \( u, v \) are variables. Introduce a Skolem function \( f_1 \).

\[ \forall x \forall y \forall z \ P(C_1, C_2, u, v, f_1(u, v), x, y, z) \]

Eliminate two more universal quantifiers.

\[ \exists z \ P(C_1, C_2, u, v, f_1(u, v), x, y, z) \]
Skolemization – One More Example

\[ \exists s \exists t \forall u \forall v \forall w \forall x \forall y \forall z \ P(s, t, u, v, w, x, y, z) \]

Replace \( s \) with a constant \( C_1 \) (i.e., a function with no argument).

\[ \exists t \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, t, u, v, w, x, y, z) \]

Replace \( t \) with another constant \( C_2 \). (\( t \) depends \( s \) and is a function of \( C_1 \). It is thus a constant as well.)

\[ \forall u \forall v \forall w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z) \]

Eliminate the two universal quantifiers in front of \( u \) and \( v \).

\[ \exists w \forall x \forall y \forall z \ P(C_1, C_2, u, v, w, x, y, z) \]

\( w \) depends on \( C_1, C_2, u, v \), among which only \( u, v \) are variables. Introduce a Skolem function \( f_1 \).

\[ \forall x \forall y \forall z \ P(C_1, C_2, u, v, f_1(u, v), x, y, z) \]

Eliminate two more universal quantifiers.

\[ \exists z \ P(C_1, C_2, u, v, f_1(u, v), x, y, z) \]

\( z \) depends on \( C_1, C_2, u, v, x, y \), among which only \( u, v, x, y \) are variables. Introduce a second Skolem function \( f_2 \).

\[ P(C_1, C_2, u, v, f_1(u, v), x, y, f_2(u, v, x, y)) \]
Handling $\forall, \lor, \text{ and } \land$

e) Drop universal quantifiers:

$$\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)$$
Handling \( \forall, \lor, \text{ and } \land \)

e) Drop universal quantifiers:

\[
\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)
\]

\[
\Downarrow
\]

\[
(\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)
\]
Handling ∀, ∨, and ∧

e) Drop universal quantifiers:

$$\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)$$

↓

$$(\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)$$

f) Distribute ∨ over ∧:
Handling ∀, ∨, and ∧

e) Drop universal quantifiers:

\[ ∀x \ (Animal(F(x)) \land \neg Loves(x, F(x))) \lor Loves(G(x), x) \]

\[ \downarrow \]

\[ (Animal(F(x)) \land \neg Loves(x, F(x))) \lor Loves(G(x), x) \]

f) Distribute ∨ over ∧:

\[ (Animal(F(x)) \lor Loves(G(x), x)) \land (\neg Loves(x, F(x)) \lor Loves(G(x), x)) \]
Handling $\forall$, $\lor$, and $\land$

e) Drop universal quantifiers:

$$\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)$$

f) Distribute $\lor$ over $\land$:

$$\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))) \lor \text{Loves}(G(x), x)$$

clause 1
Handling ∀, ∨, and ∧

e) Drop universal quantifiers:

\[ ∀x \ (\text{Animal}(F(x)) ∧ \neg \text{Loves}(x, F(x))) ∨ \text{Loves}(G(x), x) \]

downarrow

\[ (\text{Animal}(F(x)) ∧ \neg \text{Loves}(x, F(x))) ∨ \text{Loves}(G(x), x) \]

f) Distribute ∨ over ∧:

\[ (\text{Animal}(F(x)) ∨ \text{Loves}(G(x), x)) ∧ (\neg \text{Loves}(x, F(x)) ∨ \text{Loves}(G(x), x)) \]

clause 1

clause 2