Knowledge Representation (II)

Paolo Turrini

Department of Computing, Imperial College London

Introduction to Artificial Intelligence
The main reference

Stuart Russell and Peter Norvig
Artificial Intelligence: a modern approach
Chapter 9
Today’s class

- Start with a knowledge base and try to prove something interesting
- Various methods for doing so, with different computational properties
Artificial Intelligence and Law

The formalization of legislation and the development of computer systems to assist with legal problem solving provide a rich domain for developing and testing artificial-intelligence technology.

Marek Sergot, Fariba Sadri, Robert Kowalski, (and others)
*The British Nationality Act as a logic program*
Communications of the ACM, 1986
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.
An informal knowledge base

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.

Prove that Colonel West is a criminal
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

... has some missiles

Owns (Nono, M1) and Missile (M1)

... all of its missiles were sold to it by Colonel West

∀ x Missile (x) ∧ Owns (Nono, x) ⇒ Sells (West, x, Nono)
Example knowledge base contd.

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]
Example knowledge base contd.

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

Nono . . . has some missiles
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]
Nono ... has some missiles
\[ \text{Owns}(\text{Nono}, M_1) \land \text{Missile}(M_1) \]
Example knowledge base contd.

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

Nono ... has some missiles

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

... all of its missiles were sold to it by Colonel West
Example knowledge base contd.

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

Nono ... has some missiles
\[\text{Owns}(\text{Nono}, M_1) \quad \text{and} \quad \text{Missile}(M_1)\]

... all of its missiles were sold to it by Colonel West
\[\forall x \quad \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})\]
Example knowledge base contd.

Missiles are weapons:
Example knowledge base contd.

Missiles are weapons:

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
Example knowledge base contd.

Missiles are weapons:

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

An enemy of America "counts as" hostile:
Example knowledge base contd.

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America "counts as" hostile:

$Enemy(x, America) \Rightarrow Hostile(x)$
Example knowledge base contd.

Missiles are weapons:

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

An enemy of America "counts as" hostile:

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

West, who is American.
Example knowledge base contd.

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America ”counts as” hostile:

Enemy(x, America) ⇒ Hostile(x)

West, who is American.

American(West)
Example knowledge base contd.

Missiles are weapons:

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

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 . . .
Example knowledge base contd.

Missiles are weapons:

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

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 . . .

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

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
Forward chaining proof

Diagram:

- Weapon(M1)
- Sells(West, M1, Nono)
- Hostile(Nono)
- American(West)
- Missile(M1)
- Owns(Nono, M1)
- Enemy(Nono, America)
Forward chaining proof

- Criminal(West)
  - Weapon(M1)
    - American(West)
    - Missile(M1)
  - Sells(West,M1,Nono)
    - Owns(Nono,M1)
  - Hostile(Nono)
    - Enemy(Nono,America)
Footage from Tuesday
GMP: recall...

Definite clause:
disjunction of literals, \textbf{exactly} one of which positive
e.g., \((p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)\)

\[
\begin{array}{c}
p_1', \ p_2', \ \ldots, \ p_n', \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \\
q\theta
\end{array}
\]

where \(p_i'\theta = p_i\theta\) for all \(i\)

Assuming all variables are universally quantified...

\begin{align*}
p_1' & \text{ is } \text{King}(\text{Joffrey}) \quad & p_1 & \text{ is } \text{King}(x) \\
p_2' & \text{ is } \text{Greedy}(y) \quad & p_2 & \text{ is } \text{Greedy}(x) \\
\theta & \text{ is } \{x/\text{Joffrey}, y/\text{Joffrey}\} \quad & q & \text{ is } \text{Evil}(x) \\
q\theta & \text{ is } \text{Evil}(\text{Joffrey})
\end{align*}
Forward chaining algorithm

function FOL-FC-Ask\((KB, \alpha)\) returns a substitution or \textit{false}

inputs: \(KB\), the knowledge base, a set of definite clauses
\(\alpha\), the (atomic) query

local variables: \(\textit{new}\), the new sentences inferred at each iteration

repeat until \(\textit{new}\) is empty
\(\textit{new} \leftarrow \{\}\)

for each sentence \(r\) in \(KB\) do

\((p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow \text{Standardize-Apart}(r)\)

for each \(\theta\) such that \((p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta\)

for some \(p'_1, \ldots, p'_n\) in \(KB\) do

\(q' \leftarrow q\theta\)

if \(q'\) does not already unify in \(KB\) or \(\textit{new}\) then

add \(q'\) to \(\textit{new}\)

\(\phi \leftarrow \text{Unify}(q', \alpha)\)

if \(\phi\) is not \textit{fail} then return \(\phi\)

add \(\textit{new}\) to \(KB\)

return \textit{false}
First iteration

On the first iteration...

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

On the first iteration...

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

has unsatisfied premises
First iteration

\[ \forall x \: \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]
∀ x  Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)

is satisfied with {x/M₁} and Sells(West, M₁, Nono) is added
First iteration

\[ \text{Missile}(x) \implies \text{Weapon}(x) \]
First iteration

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

is satisfied with \( \{x/M_1\} \) and \( \text{Weapon}(M_1) \) is added
First iteration

\[ \text{Enemy}(x, \text{America}) \implies \text{Hostile}(x) \]
First iteration

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

is satisfied with \( \{x/\text{Nono}\} \) and \( \text{Hostile}(\text{Nono}) \) is added
First iteration

So in total we have added...
First iteration

So in total we have added...

\[ \text{Sells}(\text{West}, M_1, \text{Nono}) \]
First iteration

So in total we have added...

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

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

So in total we have added...

\[ Sells(West, M_1, Nono) \]

\[ Weapon(M_1) \]

\[ Hostile(Nono) \]
Second iteration

On the second iteration...
On the second iteration...

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

On the second iteration...

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

is satisfied with \( \{x/\text{West}, y/M_1, z/\text{Nono}\} \)
and \( \text{Criminal(West)} \) is added
On the second iteration...

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

is satisfied with \(\{x/\text{West}, y/M_1, z/\text{Nono}\}\)
and \(\text{Criminal}(\text{West})\) is added

Yay! :)

Paolo Turrini  Intro to AI
Forward chaining proof

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
Forward chaining proof

- Weapon(M1)
- Sells(West,M1,Nono)
- Hostile(Nono)
- American(West)
- Missile(M1)
- Owns(Nono,M1)
- Enemy(Nono,America)
Forward chaining proof

- **Criminal(West)**
  - **Weapon(M1)**
    - **American(West)**
    - **Missile(M1)**
  - **Sells(West,M1,Nono)**
    - **Owns(Nono,M1)**
  - **Hostile(Nono)**
    - **Enemy(Nono,America)**
Properties of forward chaining

- Sound (because of soundness of GMP)
Properties of forward chaining

- Sound (because of soundness of GMP)
- Complete for first-order (entailed!) definite clauses
Properties of forward chaining

- Sound (because of soundness of GMP)
- Complete for first-order (entailed!) definite clauses
- May not terminate in general if $\alpha$ is not entailed

It's inefficient: but there can be improvements well... matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases
Properties of forward chaining

- Sound (because of soundness of GMP)
- Complete for first-order (entailed!) definite clauses
- May not terminate in general if $\alpha$ is not entailed
- It’s inefficient:
Properties of forward chaining

- Sound (because of soundness of GMP)
- Complete for first-order (entailed!) definite clauses
- May not terminate in general if $\alpha$ is not entailed
- It’s inefficient:
  - but there can be improvements
Properties of forward chaining

- Sound (because of soundness of GMP)
- Complete for first-order (entailed!) definite clauses
- May not terminate in general if $\alpha$ is not entailed
- It’s inefficient:
  - but there can be improvements
  - well... matching conjunctive premises against known facts is NP-hard
Properties of forward chaining

- Sound (because of soundness of GMP)
- Complete for first-order (entailed!) definite clauses
- May not terminate in general if $\alpha$ is not entailed
- It’s inefficient:
  - but there can be improvements
  - well... matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases
Backward chaining example

Criminal(West)
Backward chaining example

![Diagram]

- Criminal(West)
- American(x)
- Weapon(y)
- Sells(x, y, z)
- Hostile(z)
Backward chaining example

- Criminal(West)
- American(West) with the empty set { }
- Weapon(y)
- Sells(x,y,z)
- Hostile(z)
Backward chaining example

```
American(West)  
{}                  

Weapon(y)      

Sells(x,y,z)  

Missile(y)       

Hostile(z)     

Criminal(West)   
{x/West}          
```
Backward chaining example

- **Criminal(West)**
  - **American(West)**
    - **Weapon(y)**
      - **Missile(y)**
        - `{y/M1}`
  - **Sells(x,y,z)**
  - **Hostile(z)**
  - `{x/West, y/M1}`
Backward chaining example

- **Criminal(West)**
  - **American(West)** { }
  - **Weapon(y)**
    - **Missile(y)** { y/M1 }
  - **Sells(West,M1,z)** { z/Nono }
    - **Missile(M1)***
    - **Owns(Nono,M1)***
  - **Hostile(z)**
Backward chaining example

```
Criminal(West) \{x/West, y/M1, z/Nono\}

American(West) \{\}\nWeapon(y) \{\}\nSells(West,M1,z) \{z/Nono\}
Hostile(Nono) \{\}\nMissile(y) \{y/M1\}
Missile(M1) \{\}\nOwns(Nono,M1) \{\}\nEnemy(Nono,America) \{\}\n```
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' ← FIRST(goals)θ

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 example

\[
\text{Criminal(\textit{West})}
\]
Backward chaining example

Diagram:

- `Criminal(West)`
- `American(x)`
- `Weapon(y)`
- `Sells(x, y, z)`
- `Hostile(z)`

Variables:
- `{x/West}`
Backward chaining example
Backward chaining example

![Diagram of a backward chaining example]

- **Criminal(West)**
- **American(West)**
  - {}  
- **Weapon(y)**
- **Sells(x,y,z)**
- **Hostile(z)**
- **Missile(y)**
Backward chaining example

- Criminal(West)
- American(West) {}
- Weapon(y)
- Sells(x,y,z)
- Hostile(z)
- Missile(y) y/M1
  - { y/M1 }
Backward chaining example
Backward chaining example

Diagram:

- **Criminal(West)**
  - **American(West)**: \{\} 
  - **Weapon(y)**: \{\} 
  - **Sells(West,M1,z)**: \{z/Nono\} 
    - **Missile(y)**: \{y/M1\} 
    - **Missile(M1)**: \{\} 
    - **Owns(Nono,M1)**: \{\} 
  - **Hostile(Nono)**: \{\} 
    - **Enemy(Nono,America)**: \{\} 

The diagram represents a backward chaining example in AI, illustrating the relationships and conditions involved.
Properties of backward chaining

Depth-first recursive proof search
Properties of backward chaining

Depth-first recursive proof search

- space is linear in size of proof
Properties of backward chaining

Depth-first recursive proof search

- space is linear in size of proof
- Incomplete due to infinite loops
Properties of backward chaining

Depth-first recursive proof search

- space is linear in size of proof
- Incomplete due to infinite loops

Widely used for logic programming
Resolution

Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n \\
\frac{(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta}{\text{where } \text{UNIFY}(\ell_i, \neg m_j) = \theta.}
\]
Resolution

Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

\[\neg \text{Rich}(x) \lor \text{Unhappy}(x)\]
Resolution

Full first-order version:

\[
\frac{\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n}{(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta}
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

\[
\begin{align*}
\neg \text{Rich}(x) & \lor \text{Unhappy}(x) \\
\text{Rich}(\text{Berlusconi}) &
\end{align*}
\]
Resolution

Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

\[
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\]

\[
\text{Rich}(\text{Berlusconi})
\]

\[
\text{Unhappy}(\text{Berlusconi})
\]
Resolution

Full first-order version:

\[
\begin{align*}
\ell_1 \lor \cdots \lor \ell_k, & \quad m_1 \lor \cdots \lor m_n \\
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta
\end{align*}
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

\[
\begin{align*}
\neg \text{Rich}(x) \lor \text{Unhappy}(x) \\
\text{Rich}(\text{Berlusconi}) \\
\underline{\text{Unhappy}(\text{Berlusconi})}
\end{align*}
\]
Resolution

Full first-order version:

\[
x_1 \lor \cdots \lor x_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta
\]

where Unify(\(\ell_i, \neg m_j\)) = \(\theta\).

For example,

\[
\neg Rich(x) \lor Unhappy(x)
\]

\[
Rich(Berlusconi)
\]

\[
Unhappy(Berlusconi)
\]

with \(\theta = \{x/Berlusconi\}\)
Resolution

Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta
\]

where \(\text{UNIFY}(\ell_i, \neg m_j) = \theta\).

For example,

\[
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\]

\[
\text{Rich}(\text{Berlusconi})
\]

\[
\underline{\text{Unhappy}(\text{Berlusconi})}
\]

with \(\theta = \{x/\text{Berlusconi}\}\)

Apply resolution steps to \(\text{CNF}(KB \land \neg \alpha)\)
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

\( \forall x \ [\forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)] \)
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

\[ \forall x \ [ \forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)] \]

1. Eliminate biconditionals and implications
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

\[ \forall x \ [\forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg \forall y \ \neg Pizza(y) \lor Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

\[ \forall x \ [\forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg\forall y \ \neg Pizza(y) \lor Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]

2. Move \( \neg \) inwards:

\[ \neg\forall x \varphi \equiv \exists x \ \neg \varphi, \quad \neg\exists x \varphi \equiv \forall x \ \neg \varphi : \]
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

$$\forall x \ [\forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \ [\neg \forall y \neg Pizza(y) \lor Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)]$$

2. Move $$\neg$$ inwards: $$\neg \forall x \varphi \equiv \exists x \ \neg \varphi, \ \neg \exists x \varphi \equiv \forall x \ \neg \varphi$$:

$$\forall x \ [\exists y \ \neg (\neg Pizza(y) \lor Dislikes(x, y))] \lor [\exists y \ Dislikes(y, x)]$$
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

$$\forall x \ [\forall y \ Pizza(y) \Rightarrow \ Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \ [\neg \forall y \ \neg Pizza(y) \lor Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)]$$

2. Move $\neg$ inwards: $\neg \forall x \varphi \equiv \exists x \ \neg \varphi, \ \neg \exists x \varphi \equiv \forall x \ \neg \varphi$:

$$\forall x \ [\exists y \ \neg (\neg Pizza(y) \lor Dislikes(x, y))] \lor [\exists y \ Dislikes(y, x)]$$
$$\forall x \ [\exists y \ \neg \neg Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)]$$
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

\[ \forall x \ [\forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg \forall y \ \neg Pizza(y) \lor Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]

2. Move \(\neg\) inwards:

\[ \neg \forall x \ \varphi \equiv \exists x \ \neg \varphi, \quad \neg \exists x \ \varphi \equiv \forall x \ \neg \varphi: \]

\[ \forall x \ [\exists y \ \neg((\neg Pizza(y) \lor Dislikes(x, y)))] \lor [\exists y \ Dislikes(y, x)] \]
\[ \forall x \ [\exists y \ \neg Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]
\[ \forall x \ [\exists y \ Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]
Conversion to CNF

Everyone who dislikes pizzas is disliked by someone:

\[ \forall x \ [\forall y \ Pizza(y) \Rightarrow Dislikes(x, y)] \Rightarrow [\exists y \ Dislikes(y, x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg\forall y \ \neg Pizza(y) \lor Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]

2. Move \( \neg \) inwards:

\[ \forall x \ [\exists y \ \neg(\neg Pizza(y) \lor Dislikes(x, y))] \lor [\exists y \ Dislikes(y, x)] \]

\[ \forall x \ [\exists y \ \neg\neg Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]

\[ \forall x \ [\exists y \ Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists y \ Dislikes(y, x)] \]
Conversion to CNF contd.

3. Standardise variables: each quantifier should use a different one
Conversion to CNF contd.

3. Standardise variables: each quantifier should use a different one

\[ \forall x \ [\exists y \ Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists z \ Dislikes(z, x)] \]
Conversion to CNF contd.

3. Standardise variables: each quantifier should use a different one

\[ \forall x \ [\exists y \ Pizza(y) \land \neg \text{Dislikes}(x, y)] \lor [\exists z \ \text{Dislikes}(z, x)] \]

4. Skolemize: a more general form of existential instantiation.
3. Standardise variables: each quantifier should use a different one

\[
\forall x \ [\exists y \ Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists z \ Dislikes(z, x)]
\]

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function
Conversion to CNF contd.

3. Standardise variables: each quantifier should use a different one

\[ \forall x \left( \exists y \ Pizza(y) \land \neg\text{Dislikes}(x, y) \right) \lor \left( \exists z \ \text{Dislikes}(z, x) \right) \]

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
Conversion to CNF contd.

3. Standardise variables: each quantifier should use a different one

\[ \forall x \ [\exists y \ Pizza(y) \land \neg Dislikes(x, y)] \lor [\exists z \ Dislikes(z, x)] \]

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

\[ \forall x \ [Pizza(F(x)) \land \neg Dislikes(x, F(x))] \lor Dislikes(G(x), x) \]
Conversion to CNF contd.

5. Drop universal quantifiers:

\[
\text{Pizza}(F(x)) \land \neg \text{Dislikes}(x, F(x)) \lor \text{Dislikes}(G(x), x)
\]
5. Drop universal quantifiers:

\[ \text{Pizza}(F(x)) \land \neg \text{Dislikes}(x, F(x)) \lor \text{Dislikes}(G(x), x) \]

6. Distribute \( \land \) over \( \lor \):

\[ \text{Pizza}(F(x)) \lor \text{Dislikes}(G(x), x) \land \]

\[ \land [\neg \text{Dislikes}(x, F(x)) \lor \text{Dislikes}(G(x), x)] \]
Resolution proof: definite clauses

\[
\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor Criminal(x) \rightarrow \neg Criminal(West)
\]

\[
\neg American(West) \lor \neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)
\]

\[
\neg Missile(x) \lor Weapon(x)
\]

\[
\neg Missile(M1) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)
\]

\[
\neg Missile(x) \lor \neg Owns(Nono,x) \lor Sells(West,x,Nono) \rightarrow \neg Sells(West,M1,z) \lor \neg Hostile(\tau)
\]

\[
\neg Missile(M1) \lor \neg Owns(Nono,M1) \lor \neg Hostile(Nono)
\]

\[
\neg Owns(Nono,M1) \lor \neg Hostile(Nono)
\]

\[
\neg Enemy(x,America) \lor Hostile(x)
\]

\[
\neg Hostile(Nono)
\]

\[
\neg Enemy(Nono,America)
\]

\[
\neg Enemy(Nono,America)
\]
What we have seen

Inference methods with different structure and properties

- Forward-chaining (deductive databases)
- Backward-chaining (logic programming)
- Resolution (full-first order logic)
Coming next

- Knowledge and uncertainty: what if we don’t know exactly?
- How to handle uncertain information:
  - representation
  - reasoning
Every instantiation of a universally quantified sentence is entailed by it:

$$\forall \, v \alpha \quad \frac{}{\alpha(\{v/g\})}$$

for any variable $v$ and ground term $g$. 

E.g.,

$$\forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$ yields

$$\text{King}(\text{Joffrey}) \land \text{Greedy}(\text{Joffrey}) \Rightarrow \text{Evil}(\text{Joffrey})$$

$$\text{King}(\text{Aerys}) \land \text{Greedy}(\text{Aerys}) \Rightarrow \text{Evil}(\text{Aerys})$$

$$\text{King}(\text{Father}(\text{Joffrey})) \land \text{Greedy}(\text{Father}(\text{Joffrey})) \Rightarrow \text{Evil}(\text{Father}(\text{Joffrey}))$$...
Appendix: Universal Instantiation

Every instantiation of a universally quantified sentence is entailed by it:

\[ \forall v \alpha \rightarrow \alpha(\{v/g\}) \]

for any variable \( v \) and ground term \( g \)
E.g., \( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \) yields
Appendix: Universal Instantiation

Every instantiation of a universally quantified sentence is entailed by it:

$$\forall v \alpha \\
\alpha(\{v/g\})$$

for any variable $v$ and ground term $g$

E.g., $\forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields

$$\text{King}(\text{Joffrey}) \land \text{Greedy}(\text{Joffrey}) \Rightarrow \text{Evil}(\text{Joffrey})$$
$$\text{King}(\text{Aerys}) \land \text{Greedy}(\text{Aerys}) \Rightarrow \text{Evil}(\text{Aerys})$$
$$\text{King}(\text{Father(}\text{Joffrey})\text{)} \land \text{Greedy}(\text{Father(}\text{Joffrey})\text{)} \Rightarrow \text{Evil}(\text{Father(}\text{Joffrey})\text{)}$$

...
Appendix: Existential instantiation (EI)

For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\exists v \, \alpha \quad \frac{\exists v \, \alpha}{\alpha(\{v/g\})}$$

E.g., $\exists x \text{Crown}(x)$ and $\text{OnHead}(x, \text{John})$ yields $\text{Crown}(C_1)$ and $\text{OnHead}(C_1, \text{John})$, provided $C_1$ is a new constant symbol, called a Skolem constant.
Appendix: Existential instantiation (EI)

For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$
\exists v \alpha \\
\alpha(v/g)
$$

E.g., $\exists x \ Crown(x) \land OnHead(x, John)$ yields

provided $C_1$ is a new constant symbol, called a Skolem constant.
Appendix: Existential instantiation (EI)

For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\exists v \alpha \frac{\exists v \alpha}{\alpha(v/g)}$$

E.g., $\exists x \ Crown(x) \land \text{OnHead}(x, John)$ yields

$$\text{Crown}(C_1) \land \text{OnHead}(C_1, John)$$
Appendix: Existential instantiation (EI)

For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\exists v \alpha \frac{\alpha(v/g)}{\alpha}\)$$

E.g., $\exists x \ Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

provided $C_1$ is a **new** constant symbol, called a **Skolem constant**
UI can be applied several times to **add** new sentences; the new KB is logically equivalent to the old.
UI can be applied several times to add new sentences; the new KB is logically equivalent to the old.

EI can be applied once to replace the existential sentence;
UI can be applied several times to add new sentences; the new KB is logically equivalent to the old.

EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old,
UI can be applied several times to **add** new sentences; the new KB is logically equivalent to the old.

EI can be applied once to **replace** the existential sentence; the new KB is **not** equivalent to the old, but is satisfiable iff the old KB was satisfiable.