Introduction to Artificial Intelligence

Chapter 3: Knowledge Representation and Reasoning

(4) Inference with FOL

Nguyễn Hải Minh, Ph.D
nhminh@fit.hcmus.edu.vn
Outline

- Reducing first-order inference to propositional inference
- Unification
- Forward chaining
- Backward chaining
- Resolution
Reducing first-order inference to propositional inference

First-order inference can be done by converting the knowledge base to propositional logic and using propositional inference.
Universal Instantiation (UI)

The rule of Universal Instantiation:
- We can infer any sentence obtained by substituting a ground term (a term without variables) for the variable.

\[ \forall v \alpha \]
\[ \text{SUBST}\left(\{v/g\}, \alpha\right) \]

for any variable \( v \) and ground term \( g \)

E.g., \( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \) yields:
- \( King(John) \land Greedy(John) \Rightarrow Evil(John) \)
- \( King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \)
- \( King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John)) \)
Existential Instantiation

The rule of **Existential Instantiation**: for any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in KB

$$\exists v \alpha$$

$\text{SUBST}\{\{v/k\}, \alpha\}$

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

$$\text{Crown}(C_1) \land \text{OnHead}(C_1, \text{John})$$

provided $C_1$ is a new constant symbol, called a **Skolem constant**
Reduction to propositional inference

- Suppose the KB contains just the following:
  - $\forall x \, \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  - $\text{King}(\text{John})$
  - $\text{Greedy}(\text{John})$
  - $\text{Brother}(\text{Richard}, \text{John})$

- Instantiating the universal sentence in all possible ways, we have:
  - $\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
  - $\text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
  - $\text{King}(\text{John})$
  - $\text{Greedy}(\text{John})$
  - $\text{Brother}(\text{Richard}, \text{John})$

- The new KB is **propositionalized**: proposition symbols are
  - $\text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}), \text{etc.}$
Reduction to propositional inference

- Every FOL KB can be propositionalized so as to preserve entailment
  - A ground sentence is entailed by new KB iff entailed by original KB

- Idea:
  - Propositionalize KB and query,
  - Apply resolution,
  - Return result

- Problem: with function symbols, there are infinitely many ground terms,
  - E.g., Father(Father(Father(John)))
Reduction to propositional inference

❑ Theorem: Herbrand (1930)
  o If an original FOL KB |- \( \alpha \), then \( \alpha \) is entailed by a finite subset of the propositionalized KB
  o Idea: For \( n = 0 \) to \( \infty \) do
    • create a propositional KB by instantiating with depth-\( n \) terms
    • see if \( \alpha \) is entailed by this KB

→ Problem: works if \( \alpha \) is entailed, loops if \( \alpha \) is not entailed

❑ Theorem: Turing (1936), Church (1936)
  o Entailment for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.)
Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
  - E.g., from:
    - $\forall x \text{ King}(x) \land \text{ Greedy}(x) \Rightarrow \text{ Evil}(x)$
    - $\text{ King}(\text{ John})$
    - $\forall y \text{ Greedy}(y)$
    - $\text{ Brother}(\text{ Richard, John})$
  - It seems obvious that $\text{ Evil}(\text{ John})$, but propositionalization produces lots of facts such as $\text{ Greedy}(\text{ Richard})$ that are irrelevant.

- With $p\, k$-ary predicates and $n$ constants, there are $p \cdot n^k$ instantiations.
Unification

We can get the inference immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(\text{y})$.

$\rightarrow$ The substitution $\theta = \{x/\text{John}, y/\text{John}\}$ works
Generalized Modus Ponens (GMP)

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]

\[
\text{SUBST}(\theta, q)
\]

where \( \text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i) \) for all \( i \)

- \( p_1' \) is King(John)  \( p_1 \) is King(\( x \))
- \( p_2' \) is Greedy(y)  \( p_2 \) is Greedy(\( x \))
- \( \theta \) is \{\( x/\text{John}, y/\text{John} \} \)  \( q \) is Evil(\( x \))
- \( \text{SUBST}(\theta, q) \) is Evil(John)

\[ \square \text{All variables assumed universally quantified} \]

\( \rightarrow \text{GMP is a lifted version of Modus Ponens} \)
Unification

- Recall: \( \text{SUBST}(\theta, p) = \) result of substituting \( \theta \) into sentence \( p \)
- Unify algorithm:
  - takes 2 sentences \( p \) and \( q \) and returns a unifier if one exists
  - \( \text{UNIFY}(p, q) = \theta \) where \( \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \)
- Example: \( \text{UNIFY}(p, q) \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, ( x ))</td>
<td>Knows(John, Jane)</td>
<td>( {x/Jane} )</td>
</tr>
<tr>
<td>Knows(John, ( x ))</td>
<td>Knows(( y ), Steve)</td>
<td>( {x/\text{Steve}, y/\text{John}} )</td>
</tr>
<tr>
<td>Knows(John, ( x ))</td>
<td>Knows(( y ), Mother(( y )))</td>
<td>( {x/\text{Mother(John)}, y/\text{John}} )</td>
</tr>
<tr>
<td>Knows(John, ( x ))</td>
<td>Knows(( x ), Steve)</td>
<td>fail</td>
</tr>
</tbody>
</table>

Problem is due to use of same variable \( x \) in both sentences

Standardizing apart eliminates overlap of variables
Knows(\( z \), Steve)
Unification

- **UNIFY**(Knows(John, x), Knows(y, z)) = θ,
  1. θ = {y/John, x/z}
  2. θ = {y/John, x/John, z/John}

- The first unifier is *more general* than the second
- There is a single **Most General Unifier** (MGU) that is unique up to renaming of variables.

  MGU = {y/John, x/z}

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(x)</td>
<td>P(A)</td>
<td>{x/A}</td>
</tr>
<tr>
<td>P(f(x), y, g(x))</td>
<td>P(f(x), x, g(x))</td>
<td>{y/x} or {x/y}</td>
</tr>
<tr>
<td>P(f(x), y, g(y))</td>
<td>P(f(x), z, g(x))</td>
<td>{y/x, z/x}</td>
</tr>
<tr>
<td>P(x, B, B)</td>
<td>P(A, y, z)</td>
<td>{x/A, y/B, z/B}</td>
</tr>
<tr>
<td>P(g(f(v)), g(u))</td>
<td>P(x, x)</td>
<td>{x/g(f(v)), u/f(v)}</td>
</tr>
<tr>
<td>P(x, f(x))</td>
<td>P(x, x)</td>
<td>fail</td>
</tr>
</tbody>
</table>
The unification algorithm

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound expression
        y, a variable, constant, list, or compound expression
        θ, the substitution built up so far (optional, defaults to empty)

if θ = failure then return failure
else if x = y then return θ
else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
else if LIST?(x) and LIST?(y) then
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
else return failure
```
The unification algorithm

function \textsc{Unify-Var}(\textit{var}, x, \theta) returns a substitution

\textbf{if } \{\textit{var}/\textit{val}\} \in \theta \textbf{ then return } \textsc{Unify} (\textit{val}, x, \theta)
\textbf{else if } \{x/\textit{val}\} \in \theta \textbf{ then return } \textsc{Unify} (\textit{var}, \textit{val}, \theta)
\textbf{else if } \textsc{Occur-Check}(\textit{var}, x) \textbf{ then return } \textbf{failure}
\textbf{else return } \text{add}\ \{\textit{var}/x\} \text{ to } \theta
Exercise

Find the MGU when we UNIFY(p, q)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>MGU?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(B, C)</td>
<td>A(x, y)</td>
<td></td>
</tr>
<tr>
<td>A(x, F(D, x))</td>
<td>A(E, F(D, y))</td>
<td></td>
</tr>
<tr>
<td>A(x, y)</td>
<td>A(F(C, y), z)</td>
<td></td>
</tr>
<tr>
<td>P(A, x, F(G(y)))</td>
<td>P(y, F(z), F(z))</td>
<td></td>
</tr>
<tr>
<td>P(x, F(y))</td>
<td>P(z, G(w))</td>
<td></td>
</tr>
</tbody>
</table>
Forward Chaining
First-order Horn Clauses

- Other name: First-order definite clauses
  - Disjunctions of literals of which exactly one is positive
  - Can be atomic or implication

- Example:
  - King(x) ∧ Greedy(x) => Evil(x)
  - King(John)
  - Greedy(y)
Example knowledge base

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nonô, 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:

\[ R1: \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]

Nono ... has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono},x) \land \text{Missile}(x) \):

\[ R2: \text{Owns}(\text{Nono},M_1); \quad R3: \text{Missile}(M_1) \]

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

\[ R4: \text{Missile}(x) \land \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono}) \]

Missiles are weapons:

\[ R5: \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

An enemy of America counts as "hostile“:

\[ R6: \text{Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x) \]

West, who is American ...

\[ R7: \text{American}(\text{West}) \]

The country Nono, an enemy of America ...

\[ R8: \text{Enemy}(\text{Nono},\text{America}) \]
Forward chaining algorithm

```python
function FOL-FC-ASK(KB, α) returns a substitution or false
    inputs: KB, the knowledge base, a set of first-order definite clauses
             α, the query, an atomic sentence
    local variables: new, the new sentences inferred on each iteration

    repeat until new is empty
        new ← {}
        for each rule in KB do
            (p₁ ∧ ... ∧ pₙ ⇒ q) ← STANDARDIZE-VARIABLES(rule)
            for each θ such that SUBST(θ, p₁ ∧ ... ∧ pₙ) = SUBST(θ, p'₁ ∧ ... ∧ p'ₙ)
                for some p'₁, ..., p'ₙ in KB
                    q' ← SUBST(θ, q)
                    if q' does not unify with some sentence already in KB or new then
                        add q' to new
                        φ ← UNIFY(q', α)
                        if φ is not fail then return φ
        add new to KB
    return false
```
Forward Chaining Example: The crime KB

- **Iteration 1:**
  - R4 satisfied with \{x/M1\} → R9: \text{Sells}(\text{West}, M1, \text{Nono}) is added to KB
  - R5 satisfied with \{x/M1\} → R10: \text{Weapon}(M1) is added
  - R6 satisfied with \{x/Nono\} → R11: \text{Hostile}(\text{Nono}) is added

- **Iteration 2:**
  - R1 is satisfied with \{x/\text{West}, y/M1, z/\text{Nono}\} → \text{Criminal}(\text{West}) is added.
Forward chaining proof
Forward chaining proof
Forward chaining proof
Properties of forward chaining

- Sound and complete for first-order definite clauses
- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable
Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration \( k \) if a premise wasn't added on iteration \( k-1 \)

\[ \Rightarrow \text{match each rule whose premise contains a newly added positive literal} \]

Matching itself can be expensive:

**Database indexing** allows \( O(1) \) retrieval of known facts

- e.g., query \( \text{Missile}(x) \) retrieves \( \text{Missile}(M_1) \)

Forward chaining is widely used in **deductive databases**
Backward Chaining
Backward chaining algorithm

```plaintext
function FOL-BC-ASK(KB, query) returns a generator of substitutions
    return FOL-BC-OR(KB, query, { })

generator FOL-BC-OR(KB, goal, θ) yields a substitution
    for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) do
        (lhs, rhs) ← STANDARDIZE-VARIABLES((lhs, rhs))
        for each θ' in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
            yield θ'

generator FOL-BC-AND(KB, goals, θ) yields a substitution
    if θ = failure then return
    else if LENGTH(goals) = 0 then yield θ
    else do
        first, rest ← FIRST(goals), REST(goals)
        for each θ' in FOL-BC-OR(KB, SUBST(θ, first, θ)) do
            for each θ'' in FOL-BC-AND(KB, rest, θ') do
                yield θ''
```
Backward chaining example

Criminal(West)
Backward chaining example

![Backward Chaining Diagram]
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

\[ \text{Criminal(West)} \] \[ \{ x/\text{West}, y/M1, z/\text{Nono} \} \]

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

\[ \text{Weapon(y)} \]
\[ \{ z/\text{Nono} \} \]

\[ \text{Sells(West,M1,z)} \]
\[ \{ z/\text{Nono} \} \]

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

\[ \text{Missile(y)} \]
\[ \{ y/M1 \} \]

\[ \text{Missile(M1)} \]

\[ \text{Owns(Nono,M1)} \]
Backward chaining example
Backward chaining example

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

American(West) { }

Weapon(y) { }

Sells(West, M1, z) { z/Nono}

Hostile(Nono) { }

Missile(y) { y/M1}

Missile(M1) { }

Owns(Nono, M1) { }

Enemy(Nono, America) { }
```
Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  → fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  → fix using caching of previous results (extra space)
- Widely used for logic programming
Resolution
Resolution: brief summary

- Full first-order version:
  \[
  \begin{array}{c}
  \ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n \\
  \text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)
  \end{array}
  \]

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

- The two clauses are assumed to be standardized apart so that they share no variables.

- For example,
  \[
  \neg \text{Rich}(x) \lor \text{Unhappy}(x) \\
  \text{Rich}(x) \\
  \underline{\text{Rich}(Ken)} \\
  \text{Unhappy}(Ken)
  \]
  with \( \theta = \{x/\text{Ken}\} \)

- Apply resolution steps to \( \text{CNF}(KB \land \neg \alpha) \); complete for FOL
Conversion to CNF

*Everyone who loves all animals is loved by someone:*

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

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:** \( \neg \forall x p \equiv \exists x \neg p, \quad \neg \exists x p \equiv \forall x \neg p \)

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

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

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

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

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

4. Skolemize: removing existential quantifiers by elimination. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

5. Drop universal quantifiers:

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

6. Distribute $\lor$ over $\land$:

$$\left[ Animal(F(x)) \lor Loves(G(z),x) \right] \land \left[ \neg Loves(x,F(x)) \lor Loves(G(z),x) \right]$$
Resolution proof

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

\[ \neg \text{Criminal}(\text{West}) \]

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

\[ \neg \text{Missile}(x) \lor \text{Weapon}(x) \]

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

\[ \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono},x) \lor \text{Sells}(\text{West},x,\text{Nono}) \]

\[ \neg \text{Sells}(\text{West},\text{M1},z) \lor \neg \text{Hostile}(z) \]

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

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

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

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

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

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

Logical inference in FOL:

- Inference Rules (UI & Existential Instantiation) is used to propositionalize the inference problem. It is typically slow.
- The use of Unification to identify appropriate substitutions for variables eliminates the instantiation step in first-order proofs → more efficient.
- Forward/Backward chaining apply Generalized Modus Ponens and unification to sets of definite clause.
  - FW chaining is complete for Datalog and runs in polynomial time
  - BW chaining suffers from redundant inferences and infinite loops
- Generalized Resolution inference rule provides a complete proof system for FOL, using KB in CNF
Next week

- Chapter 4: Learning
  - Learning Decision Trees
  - Artificial Neural Network