Inference with Quantifiers

• Universal Instantiation:
  • Given $\forall x, \text{person}(x) \Rightarrow \text{likes}(x, \text{McDonalds})$
  • Infer $\text{person}(\text{John}) \Rightarrow \text{likes}(\text{John}, \text{McDonalds})$

• Existential Instantiation:
  • Given $\exists x, \text{likes}(x, \text{McDonalds})$
  • Infer $\Rightarrow \text{likes}(\text{S1}, \text{McDonalds})$
  • S1 is a “Skolem Constant” that is not found anywhere else in the KB and refers to (one of) the individuals that likes McDonalds.
Universal Instantiation

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

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

• for any variable v and ground term g
  • ground term…a term with out variables

• Example:
  • \(\forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)\) yields
    • \(\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})\)
    • \(\text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})\)
    • \(\text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))\)
    • ...

Existential Instantiation

• For any sentence $\alpha$, variable $v$, and constant $k$ that does not appear in the KB:

$$\exists v \quad \alpha$$

$$\text{SUBST} \left( \{v/k\}, \alpha \right)$$

• Example:
  • $\exists x \quad \text{Crown}(x) \land \text{OnHead}(x, \text{John})$ yields:
    • $\text{Crown}(C_1) \land \text{OnHead}(C_1, \text{John})$
    • provided $C_1$ is a new constant (Skolem)

• Like $e$ or $\pi$ in mathematics. Just a name for a specific constant
Existential Instantiation

• Universal Instantiation can be applied several times to *add* new sentences
  • The KB is logically equivalent to the old

• Existential Instantiation 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
Reduction to Propositional Inference

• Use instantiation rules to create *relevant* propositional facts in the KB, then use propositional reasoning

• Problems:
  • May generate infinite sentences when it cannot prove
  • Will generate many irrelevant sentences along the way!

• Instead, will use Unification
Unification

• **Unification**: The process of finding all legal substitutions that make logical expressions look identical.

• The UNIFY algorithm takes two sentences and returns a unifier $\theta$ for them, if one exists.

• UNIFY ($p$, $q$) = $\theta$ where SUBST ($\theta$, $p$) = SUBST ($\theta$, $q$)
## Unification examples

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

• This is a general inference rule for FOL

• Given: 

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

\[
\text{SUBST } (\theta, \ p_i') = \text{SUBST } (\theta, \ p_i) \quad \text{for all } 1 \leq i \leq n
\]

• Conclude: 
  • \(q^\theta = \text{SUBST } (\theta, \ q)\)

• This is a *lifted* version of Modus Ponens (raises MP from ground (variable-free) propositional logic to FOL.)
GMP in “CS terms” and example

• Given a rule containing variables

• If there is a consistent set of bindings for all of the variables of the left side of the rule (before the arrow)
• Then you can derive the result of substituting all of the same variable bindings into the right side of the rule

∀x, Parent(x,y) ∧ Parent(y,z) ⇒ GrandParent(x,z)

Parent(James, John), Parent(James, Richard), Parent(Harry, James)

We can derive:
• GrandParent(Harry, John), bindings: ((x Harry) (y James) (z John)
• GrandParent(Harry, Richard), bindings: ((x Harry) (y James) (z Richard)
Most General Unifier (MGU)

• The MGU places the fewest restriction on the values of variables as possible.

• Process is recursive (see Figure 9.1, if interested).

• Explore the two expressions simultaneously “side by side” building up a unifier along the way, but failing if two corresponding points in the structures do not match.

• When finding unifiers:
  • Give each rule a separate set of variables
  • If a variable is already bound to something, it must retain the same value throughout the computation
Forward Chaining (déjà vu)

- Forward Chaining
  - Start with atomic sentences in the KB and apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made.

- General steps:
  - Given a new fact, generate all consequences
  - Assumes all rules are of the form
    - $C_1 \land C_2 \land C_3 \land \ldots \Rightarrow \text{Result}$
  - Each rule and binding generates a new fact
  - This new fact will “trigger” other rules
  - Keep going until the desired fact is generated
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 Col. West, who is an American.

Prove that Col. West is a criminal.
FC: Example Knowledge Base

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

• Nono...has some missiles
  \[\exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missiles}(x)\]
  \[\text{Owns}(\text{Nono}, M_1) \text{ and Missle}(M_1)\]

• ...all of its missiles were sold to it by Col. West
  \[\forall x \ \text{Missle}(x) \land \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})\]

• Missiles are weapons
  \[\text{Missle}(x) \implies \text{Weapon}(x)\]
FC: Example Knowledge Base

• An enemy of America counts as “hostile”
  \[ \text{Enemy}(x, \text{America}) \implies \text{Hostile}(x) \]

• Col. West who is an American
  \[ \text{American}(\text{Col. West}) \]

• The country Nono, an enemy of America
  \[ \text{Enemy}(\text{Nono, America}) \]
FC: Example Knowledge Base

American(West)  Missile(MI)  Owns(Nono,M1)  Enemy(Nono, America)
FC: Example Knowledge Base
FC: Example Knowledge Base

- Criminal(West)
  - Weapon(M1)
  - Sells(West,M1,Nono)
  - Hostile(Nono)
  - American(West)
  - Missile(M1)
  - Owns(Nono,M1)
  - Enemy(Nono,America)
Forward Chaining Algorithm

function FOL-FC-Ask(KB, α) returns a substitution or false

    repeat until new is empty
        new ← {}  
        for each sentence r in KB do
            (p₁ ∧ ... ∧ pₙ ⇒ q) ← STANDARDIZE-APART(r)
            for each θ such that (p₁ ∧ ... ∧ pₙ)θ = (p₁' ∧ ... ∧ pₙ')θ
                for some p₁', ..., pₙ' in KB
                    q' ← SUBST(θ, q)
                    if q' is not a renaming of a sentence already in KB or new then do
                        add q' to new
                        φ ← UNIFY(q', α)
                        if φ is not fail then return φ
                    add new to KB
    return false
Backward Chaining (more déjà vu)

- Consider the item to be proven a goal
- Find a rule whose head is the goal (and bindings)
- Apply bindings to the body, and prove these (subgoals) in turn
- If you prove all the subgoals, increasing the binding set as you go, you will prove the item.
- Typical in Prolog
Backward Chaining Example

Criminal(West)

\{x/West, y/M1, z/Nono\}

American(West)

\{\}

Weapon(y)

\{z/Nono\}

Sells(West,M1,z)

\{\}

Hostile(Nono)

Missile(y)

\{y/M1\}

Missile(M1)

\{\}

Owns(Nono,M1)

\{\}

Enemy(Nono,America)

\{\}
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, the current substitution, initially the empty substitution { }
    local variables: ans, a set of substitutions, initially empty

if goals is empty then return {θ}
q' ← SUBST(θ, FIRST(goals))
for each r in KB where STANDARDIZE-APART(r) = (p₁ ∧ ... ∧ pₙ ⇒ q)
    and θ' ← UNIFY(q, q') succeeds
    ans ← FOL-BC-Ask(KB, [p₁, ..., pₙ|REST(goals)], COMPOSE(θ', θ)) ∪ ans
return ans
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 with every subgoal on the stack
• Inefficient due to repeated subgoals (both success and failure)
  • Fix using caching of previous results (extra space)
• Widely used without improvements for logic programming