Unification

A first-order inference rule

The key advantage of lifted inference rules over propositionalization is that they make only those substitutions which are required to allow particular inferences to proceed.

Generalized Modus Ponens (GMP)

$$\frac{(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q),\ p_1',\ p_2',\ \ldots,\ p_n'}{\text{such that } \mathrm{SUBST}(\theta, p_i) = \mathrm{SUBST}(\theta, p_i') \text{ for all } i}$$

$$\mathrm{SUBST}(\theta, q)$$

- All variables assumed universally quantified
- **Example:**

  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

Generalized Modus Ponens (GMP)

$$\frac{(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q),\ p_1',\ p_2',\ \ldots,\ p_n'}{}$$

such that **SUBST(θ, p_i)= SUBST(θ, p_i')** for all i

SUBST(θ,q)

- All variables assumed universally quantified
- **Example:**

  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

  King(John) Greedy(John) Brother(Richard,John)

Generalized Modus Ponens (GMP)

$$\frac{(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q),\ p_1',\ p_2',\ \ldots,\ p_n'}{}$$

such that $\mathbf{SUBST}(\theta, p_i) = \mathbf{SUBST}(\theta, p_i')$ for all i

$$\mathbf{SUBST}(\theta, q)$$

- **Example:**

  $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

  King(John) Greedy(John) Brother(Richard,John)

  p_1 is King(x), p_2 is Greedy(x), q is Evil(x)

Generalized Modus Ponens (GMP)

$$\frac{(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q),\ p_1',\ p_2',\ \ldots,\ p_n'}{}$$

such that **SUBST(θ, p_i)= SUBST(θ, p_i')** for all i

SUBST(θ,q)

- **Example:**

  $\forall$x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

  King(John) Greedy(John) Brother(Richard,John)

  p_1 is King(x), p_2 is Greedy(x), q is Evil(x)

  p_1' is King(John), p_2' is Greedy(y), θ is {x/John,y/John}

  SUBST(θ,q) is Evil(John)

Unification

It is the process used to find substitutions that make different logical expressions look identical. Unification is a key component of all first-order Inference algorithms.

$\text{UNIFY}(\alpha,\beta) = \theta$ means that $\text{SUBST}(\theta, \alpha) = \text{SUBST}(\theta, \beta)$

θ is our unifier value (if one exists).

Ex: "Who does John know?"

p	q	θ
Knows(John,x)	Knows(John,Jane)	

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	**q**	**θ**
Knows(John,x)	Knows(John,Jane)	{x/Jane}

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	{x/Mary, y/John}

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	**q**	**θ**
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	{x/Mary, y/John}
Knows(John,x)	Knows(y,Mother(y))	

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	{x/Mary, y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John, x/Mother(John)}

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	{x/Mary, y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John, x/Mother(John)}
Knows(John,x)	Knows(x,Mary)	

Unification

UNIFY(α,β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	{x/Mary, y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John, x/Mother(John)}
Knows(John,x)	Knows(x,Mary)	fails, because both use the same variable

$\{x_1/John,\ x_2/Mary\}$

Standardizing apart eliminates overlap of variables

- sometimes it is possible for more than one unifier returned:
  UNIFY (Knows (John, x), Knows(y, z)) =???
- This can return two possible unifications:

  {y/ John, x/ z} which means Knows (John, z)

  OR {y/ John, x/ John, z/ John}.

For each unifiable pair of expressions there is a single most general unifier (MGU), In this case it is {y/ John, x/z).

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

   if 0 = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, 0)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, 0)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
   else return failure
```

```
function UNIFY-VAR(var, x, 0) returns a substitution
   inputs: var, a variable
           x, any expression
           0, the substitution built up so far

   if {var/val} ∈ θ then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {varlx} to θ
```