Unification and Lifting

Craig Eminger

Propositional approach = not efficient.

Ex.) $\forall\, x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$
$King(John)$
$Greedy(John)$
$Brother(Richard, John)$

The query is $Evil(x)$.

Apply **Universal Instantiation** and we get:

$King(John) \land Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$

By applying a complete propositional algorithm (chapter 7), we can determine $Evil(John)$.

What happens when we add ***Siblings*(*Peter, Sharon*)** to our knowledge base?

$$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$King(John)$
$Greedy(John)$
$Brother(Richard, John)$
$\boldsymbol{Siblings(Peter, Sharon)}$

$$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$King(John)$
$Greedy(John)$
$Brother(Richard, John)$
$\boldsymbol{Siblings(Peter, Sharon)}$

2 new values in our vocabulary = "Peter" and "Sharon".

Apply **Universal Instantiation** and we get:

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
$King(Peter) \wedge Greedy(Peter) \Rightarrow Evil(Peter)$
$King(Sharon) \wedge Greedy(Sharon) \Rightarrow Evil(Sharon)$

These sentences are not all necessary in our KB.
We can teach the computer to make better inferences.

We want an X where X is a King and X is Greedy (then X is evil).

Ideally, we want θ = {substitution set}.

i.e.) $\theta = \{x/\ John\}$

$\forall y$ *Greedy*(y) = "for all values y, y is greedy"

Or basically, "everyone is greedy."

Now our $\theta = \{x/\ John,\ y/\ John\}$, so we can infer *Evil*(x)

The inference rule that encapsulates this process called **Generalized Modus Ponens**.

<u>Generalized Modus Ponens</u>:

For atomic sentences p_i, p_i', and q, where there is a substitution θ such that SU
θ, p_i) = SUBST(θ, p_i'), for all i,

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

N + 1 premises = N atomic sentences + one implication.
Applying SUBST(θ, q) yields the conclusion we seek.

p_1'	=	*King*(*John*)	p_2'	=	*Greedy*(*y*)
p_1	=	*King*(*x*)	p_2	=	*Greedy*(x)
θ	=	{*x* / *John*, *y*/ *John*}	*q*	=	*Evil*(*x*)

SUBST(θ, *q*) is *Evil*(*John*)

Generalized Modus Ponens = **lifted** Modus Ponens

Lifted - transformed from:

Propositional Logic → First-order Logic

(**Note:** Backwards chaining, forwards chaining, and resolution algorithms also have lifted forms you will see later)

How do we determine substitution θ? → **unification**.

Unification = process we use to find substitutions that make different logical expressions look identical

Algorithm: UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q)

θ is our **unifier** value (if one exists).

Ex.) "Who does John know?"

UNIFY(*Knows*(*John*, *x*), *Knows*(*John*, *Jane*)) = {*x*/ *Jane*}.

UNIFY(*Knows*(*John*, *x*), *Knows*(*y*, *Bill*)) = {*x*/*Bill*, *y*/ *John*}.

UNIFY(*Knows*(*John*, *x*), *Knows*(*y*, *Mother*(*y*))) = {*x*/*Bill*, *y*/ *John*}.

Can anyone see the problem with the next example?

UNIFY(*Knows*(*John*, *x*), *Knows*(x, *Elizabeth*)) = FAIL.

What can we do to fix it?

Both use the same variable, X. X can't equal both John and Elizabeth.

The solution: change the variable X to Y (or any other value) in *Knows*(*X*, *Elizabeth*)

Knows(*X*, *Elizabeth*) $\rightarrow$ *Knows*(*Y*, *Elizabeth*)

Still means the same.

This is called **standardizing apart**.

Another problem = sometimes possible for > 1 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*} which means *Knows*(*John*, *John*).

For each unifiable pair of expressions there is a single **most general unifier** (MGU). (algorithm on page 278)

Occur check: makes sure same variable isn't used twice
increases time complexity

Storage and Retrieval:

What we do	Function	Primative
Store info in KB	TELL	STORE
Get info from KB	ASK	FETCH

Easy way to implement:

Store all sentences in a long list, browse list one sentence at a time with UNIFY on an ASK query.

Easy way = inefficient.

Any ideas how to improve this?

Problem: On a FETCH, you would compare your query sentence with sentences that have no chance of unification.

i.e.) *Knows*(*John*, *x*) vs. *Brother*(*Richard*, *John*)

Not compatible.

Solution: categorize sentences w/ **indexing**.

Predicate indexing - one method of such

- store facts in "buckets"
- buckets stored in hash table
- accessed by index keys
- best w/ lots of predicate symbols & few clauses for each

More clauses for symbols $\rightarrow$ Multiple index keys

Ex.) *Employs*(*nameOfCompany*, *nameOfEmployee*)

Given *Employs*(*AIMA.org*, *Richard*), can answer:

Employs(*AIMA.org*, *Richard*)	Does AIMA.org employ Richard?
Employs(*x*, *Richard*)	Who employs Richard?
Employs(*AIMA.org*, *y*)	Whom does AIMA.org employ?
Employs(*x*, *y*)	Who employs whom?

We can arrange this into a **subsumption lattice**.

(the lattice for this example is shown on page 280)

A **subsumption lattice** has the following properties:

•child of any node obtained from its parents by one substitution

•the “highest” common descendant of any two nodes is the result of applying their most general unifier

•predicate with *n* arguments contains $O(2^n)$ nodes (in our example, we have two arguments, so our lattice has four nodes)

•repeated constants = slightly different lattice **[see Lattice b]**

Adding function symbols:

•creates new lattice structures

•lattice gains in complexity (size increases exponentially)

•indexing benefit lost by a > cost for storing/maintaining indices