Universität Bielefeld

Artificial Intelligence Methods

Marc Erich Latoschik
AI & VR Lab
Artificial Intelligence Group
University of Bielefeld

Universität Bielefeld

Inference in first-order logic

*see (Russel & Norvig, 2004) Chapter 9

Outline

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

Universal instantiation (UI)

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

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

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

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

  $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

Existential instantiation (EI)

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields:

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

provided C_1 is a new constant symbol, called a **Skolem constant.**

UI vs. EI

- UI can be applied multiple times to generate multiple different consequences.

- EI plays a somewhat different role in inference:
  - EI is applied once since it says that *there must exist one individual for which the consequence is true*.
  - EI denotes that individual using a name…
  - …but this name must be unique:

    - Imagine a number which is somewhat larger than 2,71828 and which satisfies $x = \lim_{n \to \infty} \left(1 + 1/n\right)^n$ for x.
    - We can give this number the name e but it would be a mistake to denote it π.

  - After that, the EI sentence can be removed, e.g., after asserting Kill(Murder,Victim) we can remove $\exists x$ Kill(*x*, *Victim*).
  - Strictly speaking: The new knowledge base is not logically equivalent but it is inferential equivalent in the sense that *it is satisfiable exactly when the original knowledge base is satisfiable*.

Reduction to propositional inference

Suppose the KB contains just the following:

$\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
King(John)
Greedy(John)
Brother(Richard,John)

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

- The new KB is **propositionalized**: proposition symbols are

  King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction contd.

- 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*)))

- **Theorem:** Herbrand (1930):
  If a sentence α is entailed by an FOL KB,
  it is entailed by a finite subset of the propositionalized KB

- **Idea:** For n = 0 to ∞ do
  1. create a propositional KB by instantiating with depth-n terms
  2. see if α is entailed by this KB
  - does this remind you of something?
  - ➢ Remember "Iterative Deepening Search"

- **Problem:** works if α is entailed, loops if α is not entailed

Reduction contd.

Theorem: Turing (1936), Church (1936):
Entailment for FOL is **semidecidable**
(algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)

Similar problem:

```
function stoptest(Prog, Input) {
   if (Prog(Input) terminates) then return YES;
   else return NO;}

function test(Prog){
   retval = YES;
   while (retval == YES) {
        retval = stoptest(Prog,Prog);
        // Does not terminate if Prog fead
        // with its own code terminates
   }
   return retval;}

print(test(&test));    // Terminates iff it does not terminate
                       // (contradiction)
```


Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
- E.g., from:
  $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
  King(John)
  $\forall y$ Greedy(y)
  Brother(Richard,John)
- it seems obvious that *Evil*(*John*)…
  …but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant
- With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations.

Generalized Modus Ponens (GMP)

- Find some x such that x is a king and x is greedy and then infer that x is evil.
- If there is a substitution θ that makes the premise of the implication identical with already existing sentences then we can assert the conclusion after applying θ.
- For the example $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$, King(John), Greedy(John) that would be a substitution {x/John}.
- Let the inference do even more, let's say we know $\forall y\ Greedy(y)$…
- …for Evil(John) we need a θ which is applied to the variables of the implication part as well as for the sentences to be matched, here {x/John,y,John}.

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

where $p_i'\theta = p_i\theta$ or $SUBST(\theta, p_i') = SUBST(\theta, p_i)$ for all *i*

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

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

- GMP used with KB of **definite clauses** (exactly one positive literal)
- All variables assumed universally quantified
- GMP is the **lifted version** of MP known from PL.

Soundness of GMP

- Need to show that

$$p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$$

provided that $p_i'\theta = p_i\theta$ for all *I*

- Lemma: For any sentence *p*, we have $p \models p\theta$ by UI
  - Lemma holds in particular for a θ which satisfies GMP conditions.

1. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
2. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$

- From $p_i'\theta = p_i\theta$, 1 and 2 qθ follows by ordinary Modus Ponens:

$$\frac{\begin{array}{c} p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta \\ p_1'\theta \wedge \ldots \wedge p_n'\theta \end{array}}{q\theta}$$

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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

➢ **Standardizing apart** eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,
  θ = {y/John, x/z } or 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 }

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
           y, a variable, constant, list, or compound
           θ, the substitution built up so far

   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(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
```


The unification algorithm

```
function UNIFY-VAR(var, x, θ) returns a substitution
   inputs: var, a variable
           x, any expression
           θ, 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 {var/x} to θ
```

Example knowledge base

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

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)

The country Nono, an enemy of America …

Enemy(Nono,America)

- *“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 Col. West is a criminal

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 = (p'1 ∧ ... ∧ p'n)θ
                      for some p'1, ..., p'n 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
```

Forward chaining proof

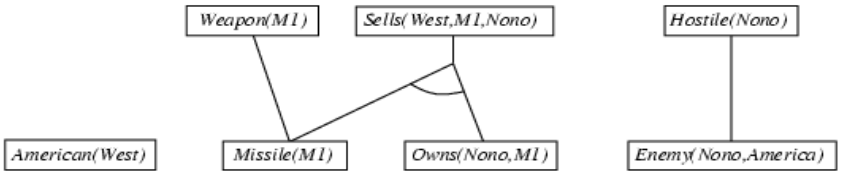
American(West)

Missile(M1)

Owns(Nono,M1)

Enemy(Nono,America)

Forward chaining proof



Forward chaining proof