CS761 Artificial Intelligence

First-order Logic Inference: Basic Tools

First-order Inference Problem

Definition

A sentence φ is called a logical consequence of a first-order knowledge base KB, written as

$$\mathrm{KB} \models \varphi$$

if φ is true in every model of KB.

The inference problem $\mathit{Ask}(\mathrm{KB}, \varphi)$ takes as input a first-order knowledge base KB and a formula φ, and outputs whether $\mathrm{KB} \models \varphi$.

Example. Consider signature that contains unary relations: $King, Greedy, Evil$, and function $father$.

The knowledge base KB contains:

$$\begin{aligned}
&\forall x\colon King(x) \land Greedy(x) \to Evil(x)\\
&King(John)\\
&\forall y\colon Greedy(y)\\
&Brother(Richard, John)
\end{aligned}$$

The inference problems: $Ask(\mathrm{KB}, Evil(John))$ should return true.

We now try to describe basic techniques used for first-order inferencing.

Two basic tools for first-order logic inference:

1. Grounding:
    - Instantiate variables with constants
    - This turns a sentence into a proposition
    - Then use methods for propositional logic to carry out inference
2. Lifting:
    - Inference is directly applied to first-order formulas
    - Generalise propositional methods to first-order logic
    - Unification is an important task here

Method 1: Grounding

- **Aim:** Reduce first-order logic inference task to propositional logic inference task.
- **Method:** Turn first-order logic sentences into "equivalent" propositions.
- **Terminology:**
  - A substitution is of the form x/t where x is a variable and t is a term.
  - For a set S of substitutions and formula φ, we use $Subst(S, \varphi)$ to denote the formula obtained by applying substitutions in S to φ. **E.g.,** $Subst(\{x/c\}, R(x, 5)) = R(c, 5)$.
- **Observations:**
  - A sentence without any quantifier is essentially a proposition. **E.g.,** $King(John)$ and $Greedy(John)$.
  - We need inference rules on existential sentences of the form $\exists x\colon \varphi(x)$ and universal sentences of the form $\forall x\colon \varphi(x)$

Grounding Rule 1: Existential Instantiation

Existential instantiation

From $\exists x\colon \varphi(x)$, derive $\varphi(c)$ where c is a new constant symbol, called the Skolem constant, i.e.,

$$\frac{\exists x\colon \varphi(x)}{Subst(\{x/c\}, \varphi(x))}$$

where c is not in the original signature.

Example. From $\exists y\colon Brother(Richard, y) \wedge Evil(y)$, we can infer

$$Brother(Richard, c) \wedge Evil(c)$$

Note:

- We only need to apply existential instantiation once for every existential quantifier.
- In this way we may eliminate all existential quantifiers from the KB.

Grounding 2: Universal Instantiation

Universal instantiation.

From $\forall x\colon \varphi(x)$, derive $\varphi(t)$ where t is any ground term, i.e.,

$$\frac{\forall x\colon \varphi(x)}{Subst(\{x/t\}, \varphi(x))}$$

Example. From $\forall x\colon King(x) \land Greedy(x) \rightarrow Evil(x)$, we can infer

$$King(John) \land Greedy(John) \rightarrow Evil(John)$$

Note:

- Applying universal instantiation produces a sentences that is only an instance of the universal sentence.
- Universal instantiation may be applied for multiple times to infer infinitely many sentences.

- The repeated applications of existential and universal instantiation on KB produces a new knowledge base KB′ that contains sentences without quantifiers.
- **Fact.** For any sentence φ, KB $\models \varphi$ if and only if KB′ $\models \varphi$.
- **Question.** How large should KB′ be?
- Since universal instantiation may be applied infinitely many times, KB′ could be infinite. But fortunately, we have ...

Herbrand's Theorem (1930)

If a sentence φ is entailed by a first-order KB, then it is entailed by a finite subset of KB.

We may thus perform inference by propositionalisation:

1. Apply grounding rules to introduce propositions to KB.
2. Apply proposition logic inference.
3. If a proof to φ is not found, repeat to generate more propositions.

Example of applying propositionalisation.

KB:

$$\begin{aligned}
&\forall x\colon King(x) \land Greedy(x) \rightarrow Evil(x)\\
&King(John)\\
&\forall y\colon Greedy(y)\\
&Brother(Richard, John)
\end{aligned}$$

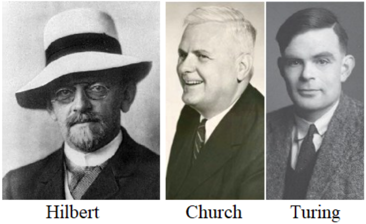
The inference problems: $Ask(\mathrm{KB}, Evil(John))$.

Proof.

1. $King(John) \land Greedy(John) \rightarrow Evil(John)$ (universal inst.)
2. $King(Richard) \land Greedy(Richard) \rightarrow Evil(Richard)$ (universal inst.)
3. $Greedy(John)$ (universal inst.)
4. $Greedy(Richard)$ (universal inst.)
5. $Evil(John)$ (modus ponens)

Remark on Decidability.

- **1928 David Hilbert's Entscheidungs problem (decision problem):** *Can one design an algorithm that given as input a sentence and a KB, tells whether the sentence is entailed by this KB?*
- Suppose we implement an inference engine that applies the method above. The method runs a search strategy to find proofs.
- Since universal instantiation can be used infinitely many times, the generated path may have arbitrary length.
- How long should we wait for the search procedure to end? Can we tell that the search will stop and find a proof?
- **1936 Alan Turing, Alonzo Church's undecidability theorem:** First-order inference is undecidable.
  - If the search finds a proof, then the sentence φ is indeed true.
  - But no algorithm exists that decides whether φ is true or not.



Method 2: Lifting

- **Shortcomings of propositionalisation**: generates too many sentences, resulting in a very large propositional knowledge base for inference.
- We need to generate only "relevant" sentences.
- This will allow us to do inference directly on first-order sentences.

Example. Matching premises.

- $\forall x \colon King(x) \wedge Greedy(x) \rightarrow Evil(x)$
- $\forall x \colon Greedy(x)$
- $King(John)$

Inference requires unification of variables.

Lifting: Unification

A unification algorithm takes 2 sentences φ_1 and φ_2 and returns a substitution S such that

$$Subst(S, \varphi_1) = Subst(S, \varphi_2)$$

if it exists. In this case, we call S a unifier.

Examples. The following table list some possible sentences and unifiers

φ_1	φ_2	S
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	$\{y/John, x/mother(John)\}$
$Knows(John, x)$	$Knows(x, OJ)$	fail

Note. There can be multiple unifiers of two sentences.

Example. $Knows(John, x)$ and $Knows(y, z)$ can be unified by

- $S_1 = \{y/John, x/z\}$
- $S_2 = \{y/John, x/Richard, z/Richard\}$

S_2 is more specific than S_1.

Definition

- Unifier S_1 is more general than unifier S_2 if S_2 can be obtained by adding substitutions to S_1.
- We let Unify(φ_1, φ_2) denote the most general unifier (mgu).

We now describe a unification algorithm:

- Recursively explore the two expressions "side by side", building up a unifier along the way
- Declare failure if the two corresponding points in the structures do not match.
- **Note.** When matching a variable against a complex term, we must check if the variable itself occurs inside the term.

Example.

$$\begin{aligned}
&\mathrm{Unify}(Knows(John, mother(x)), Knows(y, mother(Jane)), \emptyset)\\
=&\mathrm{Unify}([John, mother(x)], [y, mother(Jane)], \emptyset)\\
=&\mathrm{Unify}(mother(x), mother(Jane), \mathrm{Unify}(John, y, \emptyset))\\
=&\mathrm{Unify}(mother(x), mother(Jane), \{John/y\})\\
=&\mathrm{Unify}(x, Jane, \{John/y\})\\
=&\{John/y, x/Jane\}
\end{aligned}$$

Example.

$$
\begin{aligned}
&\mathrm{Unify}(Knows(John, x), Knows(y, mother(x)), \emptyset)\\
=&\mathrm{Unify}([John, x], [y, mother(x)], \emptyset)\\
=&\mathrm{Unify}(x, mother(x), \mathrm{Unify}(John, y, \emptyset))\\
=&\mathrm{Unify}(x, mother(x), \{John/y\})\\
=&\mathrm{fail}
\end{aligned}
$$

Unification algorithm **Unify**(x, y, S)

INPUT: x, y variables, constants, lists, or compound expressions
S the substitution built up so far
OUTPUT: mgu S' such that $Subst(S, x) = Subst(S, y)$.

if S = failure **then**
  return failure
else if $x = y$ **then**
  return S
else if $IsVariable(x)$ **then**
  return UnifyVar(x, y, S)
else if $IsVariable(y)$ **then**
  return UnifyVar(y, x, S)
else if $IsCompound(x)$ and $IsCompound(y)$ **then**
  return Unify$(x.args, y.args,$ Unify$(x.op, y.op, S))$
else if $IsList(x)$ and $IsList(y)$ **then**
  return Unifty$(x.rest, y.rest,$ Unify$(x.first, y.first, S))$
else
  return failure
end if

Algorithm **UnifyVar**(x, y, S)

INPUT: x variable
y variables, constants, lists, or compound expressions
S the substitution built up so far
OUTPUT: mgu S' such that $Subst(S, x) = Subst(S, y)$.

```
if {x/v} ∈ S for some v then
    return Unify(v, y, S)
else if {y/v} ∈ S for some v then
    return Unify(x, v, S)
else if OccurCheck(x, y) then          ▷ Check if variable x appears in y
    return failure
else
    return add {x/y} to S
end if
```

Lifting: Generalised Modus Ponens (GMP)

Generalised modus ponens (GMP) inference rule

For atomic sentences p_i, p_i' and q, where there is a substitution S such that $Subst(S, p_i') = Subst(S, p_i)$ for all $1 \le i \le n$, we can apply the following inference rule:

$$\frac{p_1', p_2', \ldots, p_n', \ (p_1 \land p_2 \land \cdots \land p_n \to q)}{Subst(S, q)}$$

Example.

$$\frac{King(John), Greedy(John), \ \forall x\colon King(x) \land Greedy(x) \to Evil(x)}{Evil(John)}$$

Here we have

- p_1' is $King(John)$ and p_1 is $King(x)$
- p_2' is $Greedy(John)$ and p_2 is $Greedy(x)$
- S is $\{x/John\}$, q is $Evil(x)$
- $Subst(S, q)$ is $Evil(John)$

First-order definite clauses. We now demonstrate the use of lifting in the inference task of a special type of FO knowledge bases.

- A first order definite clause is of the form

$$H \leftarrow A_1 \wedge \cdots \wedge A_m$$

  where $m \geq 0$, H and $A_1, \ldots, A_m$ are atomic sentences.

  Each $H \leftarrow A_1 \wedge \cdots \wedge A_m$ is interpreted as a sentence $\forall x, y, \ldots : (A_1 \wedge \cdots \wedge A_m) \rightarrow H$.

- A definite clause knowledge base is a knowledge base that contains only definite clauses.

- The definite clause inference engine would need to handle queries of the form

$$\mathit{Ask}(\mathrm{KB}, \varphi)$$

  where φ is an atomic sentence.

Example. *"The law says that it is a crime for a New Zealander to sell alcohol to minors. The girl Lucy is 15 years old and has some beers. All of the beers were sold to her by David, who is a New Zealander."* Prove that David is guilty.

- *"it is a crime for a New Zealander to sell alcohol to minors"*

$$(1)\ Crime(x) \leftarrow NZ(x) \wedge Alcohol(y) \wedge Sells(x, y, z) \wedge Minor(z)$$

- *"Lucy is 15 years old and has some beers"*

$$(2)\ Owns(Lucy, B), (3)\ Beers(B), (4)\ Under17(Lucy)$$

- *"All of the beers were sold to her by David"*

$$(5)\ Sells(David, x, Lucy) \leftarrow Beers(x) \wedge Owns(Lucy, x)$$

- Beers are alcohol

$$(6)\ Alcohol(x) \leftarrow Beers(x)$$

- Anyone younger than 17 years old is a minor.

$$(7)\ Minor(x) \leftarrow Under17(x)$$

- "David, who is a New Zealander"

$$(8)\ NZ(David)$$

- Query: $Ask(\mathrm{KB}, Crime(David))$

Forward chaining. Start with the atomic sentences in the KB and apply GMP in the forward direction, adding new atomic sentences, until no further inferences can be made.

(9) $Alcohol(B)$	$GMP, (3), (6)$
(10) $Sells(David, B, Lucy)$	$GMP, (3), (2), (5)$
(11) $Minor(Lucy)$	$GMP, (4), (7)$
(12) $Crime(David)$	$GMP, (8), (9), (10), (11), (1)$

Summary of The Topic

The following are the main knowledge points covered:

- FO Inference problem: KB $\models \varphi$
- The two basic ideas for FO inference:
  - Grounding: Turning a first order KB into a propositional KB Propositionalisation
  - Lifting: Unification to infer directly on FO sentences Unification
- Two grounding inference rules:
  - Existential instantiation
  - Universal instantiation
- Generalised modus ponens inference rule, and forward chaining.