

COMP219: Artificial Intelligence

Lecture 22: First-Order Resolution

Overview

- Last time
 - Overview of resolution in propositional logic; recap of first-order logic
- Today
 - Resolution in first-order logic
 - How knowledge representation and deduction can be carried out in first-order logic
 - The connection between Prolog, logic and resolution

- Learning outcomes covered today:

Distinguish the characteristics, and advantages and disadvantages, of the major knowledge representation paradigms that have been used in AI, such as production rules, semantic networks, propositional logic and first-order logic;

Solve simple knowledge-based problems using the AI representations studied;

Recap: Resolution Algorithm

- Proof if $KB \models \alpha$ by contradiction (i.e. show that $KB \wedge \neg\alpha$ is unsatisfiable)

function PL-Resolution(KB, α) returns *true* or *false*

inputs: KB , the knowledge base, a sentence in propositional logic

α , the query, a sentence in propositional logic

$clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$

$new \leftarrow \{\}$

loop do

for each pair of clauses C_i, C_j in $clauses$ do

$resolvents \leftarrow$ PL-Resolve(C_i, C_j)

if $resolvents$ contains the empty clause then return *true* //<- contradiction

$new \leftarrow new \cup resolvents$

if $new \subseteq clauses$ then return *false* //<- no new clauses; no contradiction

$clauses \leftarrow clauses \cup new$

Again: soundness, completeness

- Resolution is *sound* (i.e., correct):
 - if we derive an **empty clause** (i.e. **false**) from a set of clauses → the set of clauses is unsatisfiable
 - (and it returns true)
- Resolution is **complete**:
 - if given an unsatisfiable set of clauses → procedure is guaranteed to produce derive an **empty clause** (and return true).
- The resolution method *terminates*.
→ it **decides** the entailment question

Decidability in Propositional Logic

- Could decide entailment: $KB \models \alpha$
 - if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable
 - if and only if $(KB \Rightarrow \alpha)$ is valid (or tautology)
- Procedures which could be used to tell whether $KB \models \alpha$:
 - check negation is a contradiction, e.g., resolution
 - truth table method:
 - check for validity by writing down all the possible interpretations and looking to see whether the formula is true or not

First-Order Example

- Unfortunately in general we can't use this method
- Consider the formula:

$$\forall n \cdot \text{Even}(n) \Rightarrow \neg \text{Odd}(n)$$

and the domain Natural Numbers, i.e. $\{1, 2, 3, 4, \dots\}$

- There are an **infinite number of interpretations**

- Is there any other procedure that we can use, that will be guaranteed to tell us, in a finite amount of time, whether a FOL formula is, or is not, valid?

Proof in FOL Decidable?

- The answer is *no*
- For this reason FOL is said to be *undecidable*
- Entailment in FOL is often called *semi-decidable*:
 - there are procedures that *will* terminate for entailed sentences
 - given non-entailed sentences, these procedures may not terminate

Resolution Method for FOL

Propositional resolution:

- Translation to a normal form (CNF);
- At each step, a new clause is derived from two clauses you already have;
- Proof steps all use the same *resolution* rule;
- Repeat until false is derived or no new formulas can be derived.

We will now consider **how propositional resolution can be extended to first-order logic**

- Begin by translating to normal form...

Normal Form for Predicate Logic

- To write into normal form we must be able to deal with the removal of quantifiers:
 - uses a technique known as **Skolemisation**
- This is quite complex; we will just see some examples here

Dealing with Quantifiers

- Existential quantifiers

$\exists x \cdot b(x)$ is rewritten as $b(a)$

- Informally:

– *somebody is the burglar - call this person a.*

– *a is a “Skolem constant”*

- Note, any remaining variables are taken to be universally quantified

$$\exists y \forall x \cdot p(x) \Rightarrow q(x, y)$$

is rewritten as

$$\neg p(x) \vee q(x, a)$$

where a is a Skolem constant

Variable Free Resolution

- If a set of clauses contain no variables, resolution can be applied similarly to the propositional case

Example: show

$\text{cat}(\text{Kitty}),$
 $\text{cat}(\text{Kitty}) \Rightarrow \text{mammal}(\text{Kitty})$ } $\models \text{mammal}(\text{Kitty})$

i.e. show

$\left(\begin{array}{l} \text{cat}(\text{Kitty}) \\ \wedge \\ (\text{cat}(\text{Kitty}) \Rightarrow \text{mammal}(\text{Kitty})) \end{array} \right) \wedge \neg \text{mammal}(\text{Kitty})$

is unsatisfiable

To Normal Form

- In conjunctive normal form:

`cat(Kitty)`

`¬cat(Kitty) ∨ mammal(Kitty)`

`¬mammal(Kitty)`

Resolution

- Applying the resolution rule

1. `cat(Kitty)` [given]
2. `¬cat(Kitty) ∨ mammal(Kitty)` [given]
3. `¬mammal(Kitty)` [given]
4. `mammal(Kitty)` [1, 2]
5. **false** [3, 4]

- Thus `mammal(Kitty)` is a logical conclusion of
`cat(Kitty)` and
`cat(Kitty) ⇒ mammal(Kitty)`

Resolution with Variables

- Show

$$\left. \begin{array}{l} \text{cat}(\text{Kitty}) \\ \forall x \cdot \text{cat}(x) \Rightarrow \text{mammal}(x) \end{array} \right\} \models \text{mammal}(\text{Kitty})$$

i.e. show the following is unsatisfiable

$$\left(\begin{array}{l} \text{cat}(\text{Kitty}) \\ \wedge \\ (\forall x \cdot \text{cat}(x) \Rightarrow \text{mammal}(x)) \end{array} \right) \wedge \neg \text{mammal}(\text{Kitty})$$

To Normal Form

- In conjunctive normal form:

$\text{cat}(\text{Kitty})$

$\neg \text{cat}(x) \vee \text{mammal}(x)$

$\neg \text{mammal}(\text{Kitty})$

Resolution

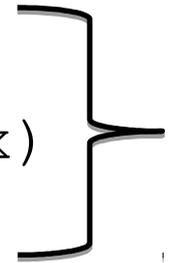
- Now to resolve
 1. $\text{cat}(\text{Kitty})$
 2. $\neg \text{cat}(x) \vee \text{mammal}(x)$need to replace x in $\neg \text{cat}(x)$
so that it matches with $\text{cat}(\text{Kitty})$
- We do this by applying the **substitution** $\{x \mapsto \text{Kitty}\}$
- The process of generating these substitutions is known as **unification**.
 - we substitute the **Most General Unifier**: i.e. make the fewest commitments needed to give a match
- Clause 2 becomes $\neg \text{cat}(\text{Kitty}) \vee \text{mammal}(\text{Kitty})$
 - the proof continues as before

Exercise

- Determine whether

`has_backbone(ali)`

`$\forall x. \neg \text{has_backbone}(x) \Rightarrow \text{invertebrate}(x)$`



`$\models \text{invertebrate}(ali)$`

Exercise

"Invertebrates are animals that neither possess nor develop a vertebral column (commonly known as a backbone or spine), derived from the notochord."

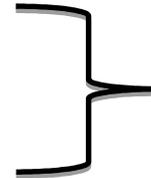


- Determine whether

`has_backbone(ali)`

`∀x. ¬has_backbone(x) ⇒ invertebrate(x)`

`⊢ invertebrate(ali)`



i.e. determine whether the following is unsatisfiable

$\left[\begin{array}{l} \text{has_backbone(ali)} \\ \forall x. \neg \text{has_backbone}(x) \Rightarrow \text{invertebrate}(x) \end{array} \right]$

`∧ ¬invertebrate(ali)`

Answer

- In conjunctive normal form:
 1. `has_backbone (ali)`
 2. `¬¬has_backbone (x) ∨ invertebrate (x)`
 3. `¬invertebrate (ali)`
- Which further transforms to:
 1. `has_backbone (ali)`
 2. `has_backbone (x) ∨ invertebrate (x)`
 3. `¬invertebrate (ali)`
- Then apply a substitution:
$$\{x \mapsto \text{ali}\}$$

Answer

- Then applying the resolution rule

```
1. has_backbone(ali) [given]
2. has_backbone(ali)  $\vee$  invertebrate(ali) [given]
3.  $\neg$ invertebrate(ali) [given]
4. has_backbone(ali) [2, 3]
```

- derived a new clause that is a subset of the existing clauses,
can not derive further resolvents
→ no contradiction
→ `invertebrate(ali)` is NOT a logical consequence of the KB
→ algorithm returns **false**.

Theoretical Considerations

- The transformation to normal form is **satisfiability preserving**:
 - if there is a model for A then there is a model for the transformation of A into CNF.
- **Soundness.** If contradiction (empty clause, false) is derived by applying resolution to a set of clauses S , then S is unsatisfiable.
- **Completeness.** If S is an unsatisfiable set of clauses, then a contradiction can be derived by applying the resolution method.
- **Decidability.** When resolution is given...
 - ... an unsatisfiable set of clauses it is guaranteed to derive contradiction and will terminate. (see completeness).
 - ... a satisfiable set of clauses, it may never terminate.
 - Entailment for FOL is *semi-decidable*.

Example of Non-Termination

- Assume we have the following pair of clauses derived from a formula that is satisfiable. We try to show them unsatisfiable (but they are in fact satisfiable).

$$1. \quad q(y) \vee \neg q(g(y))$$

$$2. \quad \neg q(x) \vee \neg p(x)$$

The proof continues as follows.

$$3. \quad \neg q(g(x)) \vee \neg p(x) \quad [1, 2, \{y \mapsto x\}]$$

$$4. \quad \neg q(g(g(x))) \vee \neg p(x) \quad [1, 3, \{y \mapsto g(x)\}]$$

$$5. \quad \neg q(g(g(g(x)))) \vee \neg p(x) \quad [1, 4, \{y \mapsto g(g(x))\}]$$

...

etc

Rule Base Example

R1: IF animal has hair
THEN animal is a mammal

R5: IF animal eats meat
THEN animal is carnivore

R9: IF animal is mammal
AND animal is carnivore
AND animal has tawney colour
AND animal has dark spots
THEN animal is cheetah

In FO Logic

- We can write the above rules in first-order logic as follows (there are other ways)

L1. $\forall x \cdot \text{has_hair}(x) \Rightarrow \text{mammal}(x)$

L5. $\forall x \cdot \text{eats}(x, \text{meat}) \Rightarrow \text{carnivore}(x)$

L9. $\forall x \cdot (\text{mammal}(x) \wedge \text{carnivore}(x) \wedge \text{colour}(x, \text{tawney}) \wedge \text{dark_spots}(x)) \Rightarrow \text{cheetah}(x)$

- Similarly for the other rules we have seen previously

Working Memory

- Assume that we have the following information in working memory

`cyril has hair,`

`cyril eats meat,`

`cyril has tawney colour,`

`cyril has dark spots`

- This can be written in first-order logic as follows

`F1. has_hair(cyril)`

`F2. eats(cyril,meat)`

`F3. colour(cyril,tawney)`

`F4. dark_spots(cyril)`

Goal

- Assume we want to show that
cyril is a cheetah
- This can be written in first-order logic as
cheetah(cyril)

Reasoning

- To show that

`cheetah(cyril)`

follows from the above first-order formula we must show

`L1, L5, L9, F1, F2, F3, F4 ⊢ cheetah(cyril)`

- We show

`L1 ∧ L5 ∧ L9 ∧ F1 ∧ F2 ∧ F3 ∧ F4 ∧
¬cheetah(cyril)`

is unsatisfiable. We abbreviate `cyril` to `c`

Proof

1. $\neg \text{has_hair}(x) \vee \text{mammal}(x)$
2. $\neg \text{eats}(y, \text{meat}) \vee \text{carnivore}(y)$
3. $\neg \text{mammal}(z) \vee \neg \text{carnivore}(z) \vee \neg \text{colour}(z, \text{tawney}) \vee \neg \text{dark_spots}(z) \vee \text{cheetah}(z)$
4. $\text{has_hair}(c)$
5. $\text{eats}(c, \text{meat})$
6. $\text{colour}(c, \text{tawney})$
7. $\text{dark_spots}(c)$
8. $\neg \text{cheetah}(c)$
9. $\neg \text{mammal}(c) \vee \neg \text{carnivore}(c) \vee \neg \text{colour}(c, \text{tawney}) \vee \neg \text{dark_spots}(c)$

- [3, 8, {z ↦ c}]
10. $\neg \text{mammal}(c) \vee \neg \text{carnivore}(c) \vee \neg \text{colour}(c, \text{tawney})$ [7, 9]
11. $\neg \text{mammal}(c) \vee \neg \text{carnivore}(c)$ [6, 10]
12. $\neg \text{mammal}(c) \vee \neg \text{eats}(c, \text{meat})$ [2, 11, {y ↦ c}]
13. $\neg \text{mammal}(c)$ [5, 12]
14. $\neg \text{has_hair}(c)$ [1, 13, {x ↦ c}]
15. **false** [4, 14]

Exercise

- Given the following KB:

$\forall x \cdot \text{has_feathers}(x) \Rightarrow \text{bird}(x)$

$\forall x \cdot (\text{bird}(x) \wedge \text{red_breast}(x)) \Rightarrow \text{robin}(x)$

$\text{has_feathers}(\text{bob})$

$\text{red_breast}(\text{bob})$

using resolution, show that $\text{KB} \models \text{robin}(\text{bob})$

Answer

1. $\neg \text{has_feathers}(x) \vee \text{bird}(x)$
2. $\neg \text{bird}(y) \vee \neg \text{red_breast}(y) \vee \text{robin}(y)$
3. $\text{has_feathers}(\text{bob})$
4. $\text{red_breast}(\text{bob})$
5. $\neg \text{robin}(\text{bob})$
6. $\text{bird}(\text{bob})$ [1, 3, {x ↦ bob}]
7. $\neg \text{red_breast}(\text{bob}) \vee \text{robin}(\text{bob})$ [2, 6, {y ↦ bob}]
8. $\text{robin}(\text{bob})$ [4, 7]
9. **false** [5, 8]

Search

- Deciding which clauses to resolve together to obtain a proof is similar to the search problems we looked at earlier in the module
- To show p follows from some database D , i.e.

$$D \models p$$

- we apply resolution to

$$D \wedge \neg p$$

- If we resolve first with clauses derived from $\neg p$, and then the newly derived clauses, we have a **backward chaining** system
- (Remember that resolution can be refined, e.g. to restrict which clauses can be resolved, but such restrictions may affect completeness...)

Prolog and First-Order Logic

- Prolog programs are really first-order logic formulae where variables are assumed to be universally quantified
- Consider the Prolog family tree program studied earlier in the module:

```
parent (cathy, ian) .
```

```
parent (pete, ian) .
```

```
female (cathy) .
```

```
male (pete) .
```

```
mother (X, Y) :- parent (X, Y) , female (X) .
```

In FO Logic

- Writing this in FOL we obtain the following:

$(\text{parent}(\text{cathy}, \text{ian}) \wedge$

$\text{parent}(\text{pete}, \text{ian}) \wedge$

$\text{female}(\text{cathy}) \wedge$

$\text{male}(\text{pete}) \wedge$

$\forall x \forall y \cdot (\text{parent}(x, y) \wedge \text{female}(x)) \Rightarrow \text{mother}(x, y)$)

Facts, Rules and Queries

- Facts (e.g. `male (pete)`) in Prolog programs are atomic sentences in FOL

- Rules in Prolog programs such as

`p (X, Y, Z) :- q (X) , r (Y, Z)`

are universally quantified FOL formulae.

$\forall x, \forall y, \forall z \cdot q(x) \wedge r(y, z) \Rightarrow p(x, y, z)$

- Queries in Prolog such as `mother (cathy, ian)` are dealt with by testing whether `mother (cathy, ian)` follows from the FOL formula representing facts and rules of the Prolog program

Horn Clauses

Here is our example written into clausal form

1. `parent(cathy, ian)`
2. `parent(pete, ian)`
3. `female(cathy)`
4. `male(pete)`
5. $\neg\text{parent}(x, y) \vee \neg\text{female}(x) \vee \text{mother}(x, y)$

- Here the clauses 1-4 contain only one positive predicate and clause 5 contains two negative predicates and one positive
 - *Horn Clauses*
- Dealing with Horn Clauses can be very efficient

Inference

- Prolog answers queries by using a special form of resolution known as *SLD resolution*
 - https://en.wikipedia.org/wiki/SLD_resolution
 - asking the query `mother(cathy, ian)` creates the goal clause $\neg\text{mother}(cathy, ian)$
 - the goal clause is maintained: $\neg L_1 \vee \dots \vee \neg L_i \vee \dots \vee \neg L_n$
 - and matched to program clauses: $L \vee \neg K_1 \vee \dots \vee \neg K_m$
 - using resolution to derive: $(\neg L_1 \vee \dots \vee \neg K_1 \vee \dots \vee \neg K_m \vee \dots \vee \neg L_n)\theta$
- Similar to applying resolution to the FOL formula of the program conjoined with $\neg\text{mother}(cathy, ian)$
- Matching in Prolog corresponds to *unification* in resolution

Summary (I)

- We have looked at how first-order formulae can be transformed into a normal form to enable resolution to be applied
- We have seen how resolution can be applied in first-order logic and how Prolog uses resolution
- If a rule-based system is written in FOL we can use resolution to show whether a particular fact follows from the facts (in working memory) and the rule base
- Entailment in FOL is **semi-decidable**:
No algorithm exists that says no to every non-entailed sentence.
- Resolution is sound and complete,
 - but applying resolution to a non-entailed sentence ($\text{KB} \wedge \neg\alpha$ is a satisfiable formula) may lead to non-termination

Summary (II)

- Logic is useful for knowledge representation as it has clear syntax, well-defined semantics (we know what formulae mean), and proof methods e.g. resolution allowing us to show a formula is a logical consequence of others
- Prolog is known as a logic programming language. The language of Prolog is a restricted version of first-order logic (Horn Clauses) and inference is by a form of resolution
- This concludes our study of knowledge representation
- Next time
 - Planning in AI