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;

Decidability in Propositional Logic

- In propositional logic, we saw that some formulae were tautologies – true under all interpretations
- We also saw that there was a procedure which could be used to tell whether any formula was a tautology - this procedure was the truth table method
- A formula of FOL that is true under all interpretations is said to be *valid*
- So we could try to 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:

  $$\forall n \cdot \ Even(n) \Rightarrow \neg Odd(n)$$

  and the domain Natural Numbers, i.e. {1, 2, 3, 4, . . .}
- 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*
- FOL is often called *semi-decidable* since although there are procedures that will terminate for valid formulas, given a formula that is not valid, the procedures may not terminate

Recap: Resolution Method

The method involves:

- 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

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

  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

```
cat(Kitty)                          } ⊨ mammal(Kitty)
cat(Kitty) ⇒ mammal(Kitty)          }
```

i.e. show

```
( cat(Kitty)
      ∧
  (cat(Kitty) ⇒ mammal(Kitty)) )  ∧ ¬mammal(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

```
cat(Kitty)                          } ⊨ mammal(Kitty)
∀x·cat(x) ⇒ mammal(x)               }
```

i.e. show the following is unsatisfiable

```
( cat(Kitty)
      ∧
  (∀x·cat(x) ⇒ mammal(x)) )  ∧ ¬mammal(Kitty)
```

To Normal Form

- In conjunctive normal form:

```
cat(Kitty)
¬cat(x) ∨ mammal(x)
¬mammal(Kitty)
```


Resolution

- Now to resolve
  `cat(Kitty)` and `¬cat(x) ∨ mammal(x)`

  we must look for a way to replace `x` in `¬cat(x)` in clause 2 so that it matches with `cat(Kitty)` in clause 1
- We do this by applying the *substitution* {`x` ↦ `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 `¬cat(Kitty) ∨ mammal(Kitty)` and now the proof continues as before

Exercise

Theoretical Considerations

- The transformation to normal form is satisfiability preserving. That is, if there is a model for A then there is a model for the transformation of A into CNF.
- **Soundness.** If **false** is derived from applying the resolution method 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.** As already mentioned, first-order logic is undecidable. Resolution is *semi-decidable*, i.e. given an unsatisfiable set of formulae it is guaranteed to derive false, however given a satisfiable set, it may never terminate.

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. q(y) ∨ ¬q(g(y))
2. ¬q(x) ∨ ¬p(x)
```

The proof continues as follows.

```
3. ¬q(g(x)) ∨ ¬p(x)       [1,2,{y ↦ x}]
4. ¬q(g(g(x))) ∨ ¬p(x)   [1,3,{y ↦ g(x)}]
5. ¬q(g(g(g(x)))) ∨ ¬p(x)[1,4,{y ↦ 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. ∀x·has_hair(x) ⇒ mammal(x)
L5. ∀x·eats(x,meat) ⇒ carnivore(x)
L9. ∀x·(mammal(x) ∧ carnivore(x) ∧
colour(x, tawney) ∧ dark_spots(x)) ⇒
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. ¬has_hair(x) ∨ mammal(x)
2. ¬eats(y,meat) ∨ carnivore(y)
3. ¬mammal(z) ∨ ¬carnivore(z) ∨ ¬colour(z,tawney) ∨ ¬dark_spots(z) ∨
   cheetah(z)
4. has_hair(c)
5. eats(c,meat)
6. colour(c,tawney)
7. dark_spots(c)
8. ¬cheetah(c)
9. ¬mammal(c) ∨ ¬carnivore(c) ∨ ¬colour(c,tawney) ∨ ¬dark_spots(c)

   [3,8,{z ↦ c}]
10. ¬mammal(c) ∨ ¬carnivore(c) ∨ ¬colour(c,tawney)    [7,9]
11. ¬mammal(c) ∨ ¬carnivore(c)   [6,10]
12. ¬mammal(c) ∨ ¬eats(c,meat)   [2,11,{y ↦ c}]
13. ¬mammal(c)    [5,12]
14. ¬has_hair(c)  [1,13,{x ↦ c}]
15. false [4,14]
```


Exercise

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 ¬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:

```
(parent(cathy,ian) ∧
 parent(pete,ian) ∧
 female(cathy) ∧
 male(pete) ∧
∀x∀y·(parent(x,y) ∧ female(x)) ⇒ 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.

  `∀x,∀y,∀z· q(x) ∧ r(y,z) ⇒ 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. ¬parent(x, y) ∨ ¬female(x) ∨ 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*
- That is, asking the query `mother(cathy,ian)` of the Prolog program given earlier is similar to applying resolution to the FOL formula of the program conjoined with `¬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
- Although resolution is sound and complete, it is semi-decidable, i.e. applying resolution to 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