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 $\text{KB} \models \alpha$ by contradiction (i.e. show that $\text{KB} \land \neg\alpha$ is unsatisfiable)

function PL-Resolution($\text{KB}, \alpha$) returns true or false
inputs: $\text{KB}$, the knowledge base, a sentence in propositional logic
$\alpha$, the query, a sentence in propositional logic
$\text{clauses} \leftarrow$ the set of clauses in the CNF representation of $\text{KB} \land \neg\alpha$
$new \leftarrow \{\}$
loop do
  for each pair of clauses $c_i, c_j$ in $\text{clauses}$ do
    $\text{resolvents} \leftarrow$ PL-Resolve($c_i, c_j$)
    if $\text{resolvents}$ contains the empty clause then return true //< contradiction
    $new \leftarrow new \cup \text{resolvents}$
    if $new \subseteq \text{clauses}$ then return false //< no new clauses; no contradiction
  $\text{clauses} \leftarrow \text{clauses} \cup new$
3
4

Again: soundness, completeness

• Resolution is sound (i.e., correct):
  – if we derive an empty clause (i.e. false) from a set of clauses $\rightarrow$ the set of clauses is unsatisfiable
  – (and it returns true)

• Resolution is complete:
  – if given an unsatisfiable set of clauses $\rightarrow$ procedure is guaranteed to produce derive an empty clause (and return true).

• The resolution method terminates.
  $\rightarrow$ it decides the entailment question
Decidability in Propositional Logic

- Could decide entailment: $KB \models \alpha$
  - if and only if $(KB \land \neg \alpha)$ 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, \ldots\}
- 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

\[ \neg p(x) \lor q(x,a) \]

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

\[
\begin{align*}
\text{cat(Kitty),} \\
\text{cat(Kitty)} & \Rightarrow \text{mammal(Kitty)} \quad \Rightarrow \quad \text{mammal(Kitty)}
\end{align*}
\]

i.e. show

\[
\begin{align*}
\text{cat(Kitty)} \\
(\text{cat(Kitty)} & \Rightarrow \text{mammal(Kitty)}) \quad \Rightarrow \quad \neg \text{mammal(Kitty)}
\end{align*}
\]

is unsatisfiable

**To Normal Form**

- In conjunctive normal form:

\[
\begin{align*}
\text{cat(Kitty),} \\
\neg \text{cat(Kitty)} & \lor \text{mammal(Kitty)} \\
\neg \text{mammal(Kitty)}
\end{align*}
\]

Resolution

- Applying the resolution rule
  1. cat(Kitty) [given]
  2. \neg cat(Kitty) \lor mammal(Kitty) [given]
  3. \neg 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) \Rightarrow mammal(Kitty)

To Normal Form

- In conjunctive normal form:
  cat(Kitty)
  \neg cat(x) \lor mammal(x)
  \neg mammal(Kitty)

Resolution

- Now to resolve
  1. cat(Kitty)
  2. \neg cat(x) \lor mammal(x)
  need to replace x in \neg cat(x)
  so that it matches with cat(Kitty)

- We do this by applying the substitution \{x \mapsto 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 cat(Kitty) \lor mammal(Kitty)
  - the proof continues as before

Resolution with Variables

- Show
  cat(Kitty)
  \forall x \cdot cat(x) \Rightarrow mammal(x)

i.e. show the following is unsatisfiable

\left\{ \begin{array}{l}
  cat(Kitty) \\
  \land \\
  (\forall x \cdot cat(x) \Rightarrow mammal(x))
\end{array} \right. \land \neg mammal(Kitty)
Exercise

• Determine whether

\[ \text{has\_backbone}(\text{ali}) \land \forall x \cdot \neg \text{has\_backbone}(x) \Rightarrow \text{invertebrate}(x) \]

\[ \models \text{invertebrate}(\text{ali}) \]

Answer

• In conjunctive normal form:

1. \( \text{has\_backbone}(\text{ali}) \)
2. \( \neg \neg \text{has\_backbone}(x) \lor \text{invertebrate}(x) \)
3. \( \neg \text{invertebrate}(\text{ali}) \)

• Which further transforms to:

1. \( \text{has\_backbone}(\text{ali}) \)
2. \( \text{has\_backbone}(x) \lor \text{invertebrate}(x) \)
3. \( \neg \text{invertebrate}(\text{ali}) \)

• Then apply a substitution:

\( \{ x \mapsto \text{ali} \} \)

Exercise

• Determine whether

\[ \text{has\_backbone}(\text{ali}) \land \forall x \cdot \neg \text{has\_backbone}(x) \Rightarrow \text{invertebrate}(x) \]

\[ \models \text{invertebrate}(\text{ali}) \]

i.e. determine whether the following is unsatisfiable

\[ \text{has\_backbone}(\text{ali}) \land \forall x \cdot \neg \text{has\_backbone}(x) \Rightarrow \neg \text{invertebrate}(\text{ali}) \]

Answer

• Then applying the resolution rule

1. \( \text{has\_backbone}(\text{ali}) \) [given]
2. \( \text{has\_backbone}(\text{ali}) \lor \text{invertebrate}(\text{ali}) \) [given]
3. \( \neg \text{invertebrate}(\text{ali}) \) [given]
4. \( \text{has\_backbone}(\text{ali}) \) [2, 3]

• derived a new clause that is a subset of the existing clauses, can not derive further resolvents

→ no contradiction

→ \( \text{invertebrate}(\text{ali}) \) is NOT a logical consequence of the KB

→ algorithm returns \text{false}.  

“Invertebrates are animals that neither possess nor develop a vertebral column (commonly known as a backbone or spine), derived from the notochord.”
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. $q(y) \lor \neg q(g(y))$
  2. $\neg q(x) \lor \neg p(x)$

  The proof continues as follows.

  3. $\neg q(g(x)) \lor \neg p(x)$  
     \[1,2,\{y \mapsto x}\]
  4. $\neg q(g(g(x))) \lor \neg p(x)$  
     \[1,3,\{y \mapsto g(x)\}\]
  5. $\neg q(g(g(g(x)))) \lor \neg p(x)$  
     \[1,4,\{y \mapsto g(g(x))\}\]

  ... 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. $\forall x \cdot (\text{mammal}(x) \land \text{carnivore}(x) \land \text{colour}(x, \text{tawney}) \land \text{dark\_spots}(x)) \Rightarrow \text{cheetah}(x)$

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

• Given the following KB:

\[ \forall x \cdot \text{has_feathers}(x) \Rightarrow \text{bird}(x) \]
\[ \forall x \cdot (\text{bird}(x) \land \text{red_breast}(x)) \Rightarrow \text{robin}(x) \]
\[ \text{has_feathers}(\text{bob}) \]
\[ \text{red_breast}(\text{bob}) \]

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

Answer

1. \( \neg \text{has_feathers}(x) \lor \text{bird}(x) \)
2. \( \neg \text{bird}(y) \lor \neg \text{red_breast}(y) \lor \text{robin}(y) \)
3. \( \text{has_feathers}(\text{bob}) \)
4. \( \text{red_breast}(\text{bob}) \)
5. \( \neg \text{robin}(\text{bob}) \)
6. \( \text{bird}(\text{bob}) \)
7. \( \neg \text{red_breast}(\text{bob}) \lor \text{robin}(\text{bob}) \)
8. \( \text{robin}(\text{bob}) \)
9. \( \text{false} \)

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

\[ D \land \neg p \]

• If we resolve first with clauses derived from \( \neg p \), and then the newly derived clauses, we have a \textit{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:
  \[
  \begin{align*}
  &\text{parent(cathy, ian)} \land \\
  &\text{parent(pete, ian)} \land \\
  &\text{female(cathy)} \land \\
  &\text{male(pete)} \land \\
  &\forall x \forall y \cdot (\text{parent}(x, y) \land \text{female}(x)) \Rightarrow \text{mother}(x, y)
  \end{align*}
  \]

**Facts, Rules and Queries**

- Facts (e.g. \text{male}(\text{pete})\]) in Prolog programs are atomic sentences in FOL
- Rules in Prolog programs such as
  \[
  p(X, Y, Z) :\neg q(X), r(Y, Z)
  \]
  are universally quantified FOL formulae.
- Queries in Prolog such as \text{mother}(\text{cathy}, \text{ian}) are dealt with by testing whether \text{mother}(\text{cathy}, \text{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. \text{parent}(\text{cathy}, \text{ian})
2. \text{parent}(\text{pete}, \text{ian})
3. \text{female}(\text{cathy})
4. \text{male}(\text{pete})
5. \neg \text{parent}(x, y) \lor \neg \text{female}(x) \lor \text{mother}(x, y)

- Here the clauses 1-4 contain only one positive predicate and clause 5 contains two negative predicates and one positive

**Inference**

- Prolog answers queries by using a special form of resolution known as SLD resolution
  - asking the query \text{mother}(\text{cathy}, \text{ian}) creates the goal clause \neg \text{mother}(\text{cathy}, \text{ian})
  - the goal clause is maintained: \neg L_1 \lor \cdots \lor \neg L_i \lor \cdots \lor \neg L_n
  - and matched to program clauses: \text{L} \lor \neg K_1 \lor \cdots \lor \neg K_m
  - using resolution to derive: \text{L}_i \lor \cdots \lor \neg K_1 \lor \cdots \lor \neg K_m \lor \cdots \lor \neg L_n\theta
- Similar to applying resolution to the FOL formula of the program conjoined with \neg \text{mother}(\text{cathy}, \text{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 (KB ∧ ¬α 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