COMP219: Artificial Intelligence

Lecture 21: Propositional Resolution

Overview

- Last time
  - Satisfiability as a search problem; Conjunctive Normal Form; DPLL algorithm

- Today
  - Propositional resolution
    - Characterisation
    - Algorithm
    - Automated reasoning
  - Recap of first-order logic

- 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;

Resolution

- Computer methods are needed to deal with huge knowledge bases
- Enumeration of models is not feasible in propositional logic
- Natural deduction contains too many rules; hard to implement search

- Resolution is a proof method for classical propositional and first-order logic; requires formulae to be in CNF
- Given a formula φ resolution will decide whether the formula is *unsatisfiable or not*
- Resolution was suggested by John Robinson in the 1960s and he claimed it to be *machine oriented* as it had only one rule of inference

Validity, Satisfiability and Entailment

- Implications for Knowledge Representation
- *Deduction Theorem:*

  $\mathrm{KB} \models \alpha$ if and only if ($\mathrm{KB} \Rightarrow \alpha$) is valid
- Or,. . .

  $\mathrm{KB} \models \alpha$ if and only if ($\mathrm{KB} \land \neg\alpha$) is unsatisfiable

  reductio ad absurdum

- For propositional, predicate and many other logics

Resolution

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 rule
  - resolution rule
- Repeat until false is derived (i.e. the formula contains a literal **and** its negation) or no new formulae can be derived
- We first introduce the method for propositional logic and then (next lecture) extend it to first-order predicate logic

Resolution Rule

- Each A_i is known as a *clause* and we consider the set of clauses $\{A_1, A_2 \ . \ . \ . \ A_k\}$
- The (propositional) resolution rule is as follows

$$\frac{\begin{array}{c} A \lor p \\ B \lor \neg p \end{array}}{A \lor B}$$

- $A \lor B$ is called the *resolvent*
- $A \lor p$ and $B \lor \neg p$ are called *parents of the resolvent*
- p and $\neg p$ are called *complementary literals*
- Note in the above A or B can be empty

Resolution applied to sets of clauses

- Show by resolution that the following set of clauses is unsatisfiable

$$\{p \lor q,\ p \lor \neg q,\ \neg p \lor q,\ \neg p \lor \neg q\}$$

1. $p \lor q$
2. $p \lor \neg q$
3. $\neg p \lor q$
4. $\neg p \lor \neg q$
5. p [1, 2]

Resolution applied to sets of clauses

- Show by resolution that the following set of clauses is unsatisfiable

$$\{p \lor q,\ p \lor \neg q,\ \neg p \lor q,\ \neg p \lor \neg q\}$$

1. $p \lor q$
2. $p \lor \neg q$
3. $\neg p \lor q$
4. $\neg p \lor \neg q$
5. p [1, 2]
6. $\neg p$ [3, 4]

Resolution applied to sets of clauses

- Show by resolution that the following set of clauses is unsatisfiable

$$\{p \lor q,\ p \lor \neg q,\ \neg p \lor q,\ \neg p \lor \neg q\}$$

1. $p \lor q$
2. $p \lor \neg q$
3. $\neg p \lor q$
4. $\neg p \lor \neg q$
5. p [1, 2]
6. $\neg p$ [3, 4]
7. **false** [5, 6]

Exercise

- use resolution: is the following set of clauses satisfiable?

$$\{\neg p \lor q \lor r,\ p,\ \neg q,\ \neg r\}$$

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 ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  loop do
   for each pair of clauses Ci, Cj in clauses do
     resolvents ← PL-Resolve(Ci, Cj)
     if resolvents contains the empty clause then return true        //<- contradiction
     new ← new ∪ resolvents
   if new ⊆ clauses then return false          //<- no new clauses; no contradiction
   clauses ← clauses ∪ new
```

Full Circle Example

- Using resolution show

$$((q \land p) \Rightarrow r) \models (\neg p \lor \neg q \lor r)$$

- show that

$$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$$

- is unsatisfiable

- Translate to CNF
- Apply the resolution algorithm

1) Transformation to CNF

$$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$$

1) Transformation to CNF

$$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$$

$$\equiv (\neg(q \land p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$$

1) Transformation to CNF

$$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$$

$$\equiv (\neg(q \land p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$$

$$\equiv ((\neg q \lor \neg p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$$

1) Transformation to CNF

$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv (\neg(q \land p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv ((\neg q \lor \neg p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv (\neg q \lor \neg p \lor r) \land (\neg\neg p \land \neg\neg q \land \neg r)$

1) Transformation to CNF

$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv(\neg(q \land p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv((\neg q \lor \neg p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv(\neg q \lor \neg p \lor r) \land (\neg\neg p \land \neg\neg q \land \neg r)$

$\equiv(\neg q \lor \neg p \lor r) \land (p \land q \land \neg r)$

1) Transformation to CNF

$((q \land p) \Rightarrow r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv (\neg(q \land p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv ((\neg q \lor \neg p) \lor r) \land \neg(\neg p \lor \neg q \lor r)$

$\equiv (\neg q \lor \neg p \lor r) \land (\neg\neg p \land \neg\neg q \land \neg r)$

$\equiv (\neg q \lor \neg p \lor r) \land (p \land q \land \neg r)$

$\equiv (\neg q \lor \neg p \lor r) \land p \land q \land \neg r$

2) Resolution

1. $\neg q \lor \neg p \lor r$
2. p
3. q
4. $\neg r$

- Finally, apply the resolution rule.

2) Resolution

1. $\neg q \lor \neg p \lor r$
2. p
3. q
4. $\neg r$

- Finally, apply the resolution rule.

5. $\neg q \lor r$ [1, 2]

2) Resolution

1. ¬q ∨ ¬p ∨ r
2. p
3. q
4. ¬r

- Finally, apply the resolution rule.

5. ¬q ∨ r [1, 2]
6. r [5, 3]

2) Resolution

1. $\neg q \lor \neg p \lor r$
2. p
3. q
4. $\neg r$

- Finally, apply the resolution rule.

5. $\neg q \lor r$ [1, 2]
6. r [5, 3]
7. **false** [4, 6]

Theoretical Issues

- Resolution is *refutation complete*. That is, if given an unsatisfiable set of clauses the procedure is guaranteed to produce **false**
- Resolution is *sound*. That is, if we derive **false** from a set of clauses then the set of clauses is unsatisfiable
- The resolution method *terminates*. That is, we apply the resolution rule until we derive false or no new clauses can be derived, and it will always stop

Reducing the Search Space (I)

- Although the basic resolution method is complete, it is not very efficient. This is due to the search space that has to be explored
- A lot of effort has been applied in trying to reduce the search space
  - The elimination of tautologies (e.g. clauses such as $p \lor q \lor \neg q$)
  - Subsumption (if a clause set contains the clauses p and $p \lor q$, $p \lor q$ may be discarded); removes useless or redundant rules.

Reducing the Search Space (II)

- Some forms of resolution restrict which clauses may be resolved together e.g. *unit resolution* (always resolve using at least one unit clause) or *set of support* (after the first step, use at most one original clause)
- Heuristics may be applied to guide the proof search e.g. *weighting strategies*
- Applying strategies such as set of support or heuristics may affect completeness

Automated Reasoning

- The resolution proof method may be automated, i.e. carried out by a computer program
- Theorem provers based on resolution have been developed
  - https://en.wikipedia.org/wiki/Automated_theorem_proving
- Prolog also uses resolution, but only for a subset of FOL: Horn Clauses
  - **At most one positive literal** in any clause.
  - `p :- q, r` is equivalent to…

    $(q \land r) \Rightarrow p$

    $p \lor \neg(q \land r)$

    $p \lor \neg q \lor \neg r$
  - This greatly improves efficiency, making Prolog usable as a programming language.

Resolution in Prolog

(1) `p:- q, r.` i.e. $p \lor \neg q \lor \neg r$

(2) `q:- t.` i.e. $q \lor \neg t$

(3) `r:- u.` i.e. $r \lor \neg u$

(4) `t.`

(5) `u.`

We want to show that (1—5) entails p

First add:

(6) $\neg p$.

Now prove (1-6) is unsatisfiable…

Resolve (6) and (1) to get (7) $\neg q \lor \neg r$

Resolve (7) and (2) to get (8) $\neg t \lor \neg r$

Resolve (4) and (8) to get (9) $\neg r$

Resolve (9) and (3) to get (10) $\neg u$

Resolve (10) and (5) to get empty clause.

$\neg p$ is unsatisfiable and hence p is true.

Pros and Cons of Propositional Logic

Benefits over most programming languages, data structures and databases:

- Propositional logic is *declarative*
  - separates knowledge and inference
- Propositional logic allows partial/disjunctive/negated information
  - allows to specify uncertainty about complex cases
- Propositional logic is *compositional*
  - Meaning of $q \land r$ depends (only!) on meaning of q and r
  - i.e., it is *context-independent* (unlike natural language, where meaning depends on context)

But…

- Propositional logic has very limited expressive power
  (unlike natural language)

Example

- Consider

$$\frac{\text{Kitty is a cat} \quad \text{cats are mammals}}{\text{Kitty is a mammal}}$$

- In propositional logic this would be represented as

$$\frac{c, m}{k}$$

  - This derivation is not valid in propositional logic. If it were then from any c and m could derive any k.
  - We **need to capture the connection between c and m**.

- To do this, we will use *first-order (or predicate) logic.*

Recap of First-Order Logic

- Whereas propositional logic assumes the world contains *facts*, first-order logic (like natural language) assumes the world contains
  - Objects (people, houses, numbers, colours...);
  - Relations (part of, after, prime, brother of, . . .);
  - Functions (best friend, one more than, end of . . .)

- Examples:

```
course_lecturer(Frans,COMP219)
male(Frans)
< (3, 4)
< (4, plustwo(1))
mammal(Kitty)
```

  - `Frans, Kitty, COMP219, 3, 4` and `1` are *constants*.
  - `course_lecturer, male, mammal`, and `<` are *predicates*.
    - `male, mammal` have *arity* one and the other predicates have arity two.
  - `Plustwo` is a function (that refers to other objects),
    - e.g. `plustwo(1)` refers to the constant `3`

Quantifiers

- Quantifiers allow us to express properties about collections of objects
- The **quantifiers** are
  $\forall$ universal quantifier *'For all* . . . *'*
  $\exists$ existential quantifier *'There exists* . . . *'*

- If `P(x)` is a predicate then we can write
  $\forall$`x·P(x)`; and
  $\exists$`x·P(x)`;
  where `x` is a *variable* which can stand for any object in the domain

Interpretations

- We need a domain to which we are referring.
  `course_lecturer(Frans,COMP219)`
- The name `Frans` is mapped to the object in the domain we are referring to (me)
- The name `COMP219` is mapped to the object in the domain we are referring to (the course COMP219)
- The predicate name `course_lecturer` will be mapped to a set of pairs of objects where the first in the pair is the (real) person who teaches the second in the pair
- Hence the above evaluates to true

Syntax of Predicate Logic

- The formulas of predicate logic are constructed from the following symbols
  - a set PRED of predicate symbols with arity;
  - a set FUNC of function symbols with arity;
  - a set CONS of constant symbols;
  - a set VAR of variable symbols;
  - the quantifiers $\forall$ and $\exists$;
  - **true, false** and the connectives $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\Leftrightarrow$, $=$.
- Note propositions can be viewed as predicates with arity 0

Terms

- The set of terms, TERM, is constructed by the following rules
  - any constant is in TERM;
  - any variable is in TERM;
  - if $t_1, \ldots t_n$ are in TERM and f is a function symbol of arity n then $f(t_1, \ldots t_n)$ is a term.
    - `f(x,y)`
    - `add(2,4)`
    - `mother_of(Katie)`

'terms' refer to objects

Well-Formed Formulae

- The set of sentences or *well-formed formulae* of predicate logic are:
  - **true**, **false** and propositional formulae are in WFF.
  - if $t_1, ..., t_n$ are in TERM and p is a predicate symbol of arity n then $p(t_1, ... t_n)$ is in WFF.
  - If A and B are in WFF then so is $\neg A$, $A \vee B$, $A \wedge B$, $A \Rightarrow B$ and $A \Leftrightarrow B$.
  - If A is in WFF and x is a variable then $\forall x \cdot A$ and $\exists x \cdot A$ are in WFF.
  - If A is in WFF then so is (A).

Exercise (Nell)

- Which of the following are well-formed formulae of first-order logic?

1) $\forall\exists \cdot p(x)$
2) $\exists x \cdot p(\neg x)$
3) $\forall x \cdot p(x) \wedge \exists y \cdot r(y)$

Domains and Interpretation

- Suppose we have a formula $\forall \mathtt{x} \cdot \mathtt{P(x)}$. What does x *range* over? *Physical objects*, *numbers*, *people*, *times*, . . . ?
- Depends on the **domain** that we intend…
- In each domain we have an **interpretation** that specifies which objects correspond to which constants, etc.
- Often, we "name" a domain to make our intended interpretation clear. E.g.,
  - Suppose our intended interpretation is the "positive integers" and that >,+,×, . . . have "the usual mathematical interpretation".
  - Is this formula *satisfiable* under the above interpretation?

    $\exists \mathtt{n} \cdot \mathtt{n} = (\mathtt{n} \times \mathtt{n})$

Summary

- We have described how to apply the proof method *resolution* in propositional logic
  - First, formulae need to be in conjunctive normal form
  - There is only one rule of inference
- We have had a brief recap of first-order logic
  - We have looked at its syntax but we haven't seen its formal semantics (see good AI and logic books)
  - Informally we've seen we need a domain of interest; constants, predicates, functions have mappings into this domain ('the interpretation')
  - To evaluate quantifiers we must check whether all objects in the domain satisfy the formula (∀) or some object does (∃)

- Next time
  - We will look at extending resolution to FOL