Artificial Intelligence

Lecture 6 – Propositional Calculus II

Outline

- Propositional Calculus Recap
  - Truth Tables
  - Entailment
  - Inference
- Proof Systems
  - Sound & Complete Inference
- Resolution
  - Clausal Form
  - The Resolution Rule
  - Soundness & Completeness
- Proof by Contradiction
- Resolution & Search

Propositional Calculus

- *Atomic propositions* are simple declarative sentences that can be *true* or *false*
- A *model* is an assignment of *true* or *false* to each atomic proposition *p*
- We can construct more complex sentences using the *logical connectives*: ¬ "not", ∧ "and", ∨ "or", and → "implies"

$$a \to b \equiv \neg a \lor b$$

$$a \leftrightarrow b \equiv a \to b \land b \to a$$

- The meaning of the logical connectives is given in terms of *truth tables*

Truth Tables

- The truth tables for the four basic connectives are:

a	*b*	¬*a*	*a* ∧ *b*	*a* ∨ *b*	*a* → *b*
true	*true*	*false*	*true*	*true*	*true*
true	*false*	*false*	*false*	*true*	*false*
false	*true*	*true*	*false*	*true*	*true*
false	*false*	*true*	*false*	*false*	*true*

- Using truth tables we can determine the truth or falsity of *any* complex sentence in a given model

Constructing Truth Tables

- We need to enumerate all possible assignments of *true* or *false* to each atomic proposition
- Recall that with n propositional variables, there are 2^n cases - therefore 2^n rows in the truth table, e.g., for $p \land q \rightarrow r$ there are 3 variables, and $2^3 = 8$ rows
- Each atomic proposition will be *true* in exactly half of the models, and *false* in the others
- Start with p, assign *true* to first half of rows, and *false* to second half
- The second variable, q, should be *true* in half the cases where p is *true* and half the cases where p is *false*
- And so on. . .

Example
(Constructing a Truth Table)

$a \land b \rightarrow c$ (2^3 = 8 models)

a	b	c	$a \land b$	$a \land b \rightarrow c$
true	*true*	*true*	*true*	*true*
true	*true*	*fasle*	*true*	*fasle*
true	*fasle*	*true*	*false*	*true*
true	*false*	*false*	*false*	*true*
false	*true*	*true*	*false*	*true*
false	*true*	*false*	*false*	*true*
false	*false*	*true*	*false*	*true*
false	*false*	*false*	*false*	*true*

- We can view this also as binary counting, where 0 = *true* and 1 = *false*, 000, 001, 010, 011, . . .

Entailment

- Given a notion of truth, we can say what it means for the truth of one statement to follow necessarily from the truth (or falsity) of others
- Definition (Entailment)
  - A set of sentences $\{\alpha_1, \alpha_2, ...\alpha_n\}$ *entails* a sentence β, written $\{\alpha_1, \alpha_2, ...\alpha_n\} \models \beta$ if in all models where $\{\alpha_1, \alpha_2, ...\alpha_n\}$ are *true*, β is also *true*
- Example (Entailment)
  - For example, $a \vee b, \neg a \models b$ since in all models where $a \vee b$ and $\neg a$ are *true*, b is also *true*

Inference

- *Entailment* can be used to derive conclusions - i.e., to carry out *logical inference*
- By enumerating all possible models we can determine if a sentence β follows logically from sentences $\{ \alpha_1, \alpha_2, \ldots \alpha_n \}$
- This gives us a reasoning process whose conclusions are guaranteed to be *true* if the premises are *true*
- Recall that with n propositions, there are 2^n possible models (rows in the truth table)
- Computing entailment in this way is therefore exponential in the number of propositional variables
- Time complexity is $O(2^n)$, space complexity is $O(n)$

Proof Systems

- Instead of trying to show semantically that $\alpha \models \beta$, a *proof system* instead uses *rules of inference* to derive valid formulas from other formulas *syntactically*
- For example, the rule of *modus ponens* mentioned in the last lecture:

$$\frac{\alpha, \alpha \rightarrow \beta}{\beta}$$

(α_n and β can be arbitrary formulas)

- A *proof* consists of a sequence of inference steps - applications of inference rules - that lead from the initial formulas to the formula to be derived, written $\alpha_1, \ldots, \alpha_n \vdash \beta$
- Proofs may still be exponential in the worst case, but they can be much shorter

Sound & Complete Inference

- Rules of inference are chosen to give a *sound* and *complete* inference procedure
- A *sound* inference procedure is one which derives only entailed sentences, i.e., derives *true* conclusions given *true* premises, $\alpha \vdash \beta$ only if $\alpha \models \beta$
- A *complete* inference procedure is one that can derive any sentence that is entailed, i.e., derives all *true* conclusions from a set of premises, $\alpha \vdash \beta$ if $\alpha \models \beta$
- Ideally, an inference procedure should be both sound and complete: $\alpha \vdash \beta$ if and only if $\alpha \models \beta$

Resolution

- Many proof systems for propositional calculus, e.g., natural deduction, tableaux methods, etc. - we shall focus on *resolution*
- Resolution has a single rule of inference: the *resolution rule*
- Sound and (refutation) complete
- Widely used in AI theorem proving and problem solving systems
- Requires that the logical description of the problem is formulated in terms of *clauses*

Clausal Form

literal	a literal is an atomic proposition or its negation, e.g., p, $\neg p$ are literals
clause	a clause is a disjunction of literals, e.g., $a \vee b$, $\neg a \vee c$, and $a \vee b \vee c$ are all clauses
CNF	a sentence expressed as a conjunction of clauses is said to be in *conjunctive normal form* (CNF), e.g., $(a \vee b) \wedge (\neg a \vee c) \wedge (a \vee b \vee c)$ is in CNF

- Any complex sentence in propositional calculus can be re-expressed in conjunctive normal form (although this might result in an exponentially larger formula in the worst case)

Converting to CNF

1. Eliminate → using:
   - $(\alpha \rightarrow \beta) \equiv (\neg\alpha \vee \beta)$
2. Convert ∧ to ∨ using distributivity:
   - $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
3. Move ¬ inward so that it appears only in front of a propositional variable, using double negation elimination and De Morgan's laws:
   - $\neg\neg\alpha \equiv \alpha$
   - $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$
   - $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$
4. Collect terms:
   - $(\alpha \vee \alpha) \equiv \alpha$
   - $(\alpha \wedge \alpha) \equiv \alpha$

The Resolution Rule

- Definition (Unit Resolution)

$$\frac{\alpha \lor \beta, \neg\beta}{\alpha}$$

- Definition (Resolution)

$$\frac{\alpha \lor \beta, \neg\beta \lor \gamma}{\alpha \lor \gamma}$$

- β, and $\neg\beta$ are *complementary literals*, i.e., one is the negation of the other, α and γ are clauses
- Resolution takes two clauses and produces a new clause containing all the literals of the original clauses *except* the two complementary literals
- The derived clause is called the *resolvent*

Example (Resolution)

- From the clauses $\neg a \lor b$ and $a \lor b$ we can derive b by resolution:

$$\frac{\neg a \lor b,\ a \lor b}{b}$$

- This is clearly a valid inference, as $\neg a \lor b,\ a \lor b \models b$:

a	*b*	¬*a*	¬*a* ∨ *b*	*a* ∨ *b*
true	***true***	*false*	***true***	***true***
true	*fasle*	*false*	*false*	*true*
false	***true***	*true*	***true***	***true***
false	*false*	*true*	*true*	*false*

Soundness of Resolution

$$\frac{\alpha \lor \beta, \neg\beta \lor \gamma}{\alpha \lor \gamma}$$

- Resolution is *sound*
- We can see this by considering the literal β:
  - if β is *true* then $\neg\beta$ is *false* and hence γ must be *true*, since $\neg\beta \lor \gamma$ is *true*
  - conversely, if β is *false*, then α must be *true* since $\alpha \lor \beta$ is *true*
- Hence, $\alpha \lor \gamma$ is *true*

Completeness of Resolution

- While resolution is sound, it is not complete for arbitrary formulas
- For example, we cannot derive $b \lor \neg b$ from a using resolution, even though $a \models b \lor \neg b$
- However, resolution is *refutation complete* - if a set of clauses is inconsistent, it is possible to derive a contradiction

Proof by Contradiction

- Recall that $\alpha \models \beta$ if and only if $\alpha \land \neg\beta$ is unsatisfiable (is t*rue* in no model)
- Proving β from α by checking the unsatisfiability of α ^ $\neg\beta$ is called *proof by refutation* or *proof by contradiction*
- We assume the sentence β to be false and show that this leads to a contradiction with the known formulas α
- For resolution, we add the negation of the clause we wish to derive to the premises and show that this leads to an empty clause (i.e., a contradiction)

Example (Proof by Contradiction)

- Show that $(a \lor b) \land (a \rightarrow b) \land (b \rightarrow a) \models a \land b$
- First, convert to clausal form using

$$a \rightarrow b \equiv \neg a \lor b$$

$$b \rightarrow a \equiv a \lor \neg b$$

- This gives: $(a \lor b) \land (\neg a \lor b) \land (a \lor \neg b) \models a \land b$
- Assume $a \land b$ to be *false* and convert to clausal form using

$$\neg(a \land b) \equiv \neg a \lor \neg b$$

- which gives $(a \lor b) \land (\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b) \models \varnothing$

Proof by Contradiction

- Clauses
  - *(1) a* ∨ *b*
  - (2) ¬*a* ∨ *b*
  - *(3) a* ∨ ¬*b*
  - (4) ¬*a* ∨ ¬*b*

- Proof

1	*a* ∨ *b*	(1)
2	¬*a* ∨ *b*	(2)
3	b	1, 2, by resolution
4	*a* ∨ ¬*b*	(3)
5	a	3, 4, by resolution
6	¬*a* ∨ ¬*b*	(4)
7	¬*b*	5, 6, by resolution
8	Ø	3, 7, by resolution

Murder Mystery

- There has been a murder! The police are not releasing many details
- Suspects are Prof. Purple, General Horseradish, or Reverend Fields
- The murder either took place in the study or the hall
- The murder weapon was either a heavy candlestick or a revolver
- The Reverend is too old and frail to wield the candlestick
- We know that the revolver was not taken out of the study
- Only the General and the Professor had access to the study
- Prove that the reverend couldn’t have committed the murder

Murder Mystery

- There has been a murder! The police are not releasing many details
- Suspects are Prof. Purple, General Horseradish, or Reverend Fields:
  - `Prof` $\vee$ `General` $\vee$ `Reverend`
- The murder either took place in the study or the hall:
  - `Study` $\vee$ `Hall`
- The murder weapon was either a heavy candlestick or a revolver:
  - `Candlestick` $\vee$ `Revolver`
- The Reverend is too old and frail to wield the candlestick:
  - `Candlestick` $\rightarrow \neg$`Reverend`
- We know that the revolver was not taken out of the study:
  - `Revolver` $\rightarrow$ `Study`
- Only the General and the Professor had access to the study:
  - `Study` $\rightarrow \neg$`Reverend`
- Prove that the reverend couldn't have committed the murder

Murder Mystery

- Goal: prove that the reverend couldn't have committed the murder
- Clauses:

(1) `Prof ∨ General ∨ Reverend`

(2) `Study ∨ Hall`

(3) `Candlestick ∨ Revolver`

(4) `¬Candlestick ∨ ¬Reverend`

(5) `¬Revolver ∨ Study`

(6) `¬Study ∨ ¬Reverend`

- To Prove: `¬Reverend`, so add `Reverend` as clause 7 and derive a contradiction (empty clause)

(7) `Reverend`

Murder Mystery

- Proof

1	`¬Candlestick ∨ ¬Reverend`	(4)
2	`Reverend`	(7)
3	`¬Candlestick`	1, 2, by resolution
4	`Candlestick ∨ Revolver`	(3)
5	`Revolver`	3, 4, by resolution
6	`¬Revolver ∨ Study`	(5)
7	`Study`	5, 6, by resolution
8	`¬Study ∨ ¬Reverend`	(6)
9	`¬Reverend`	7, 8, by resolution
10	$\varnothing$	2, 9, by resolution

Resolution & Search

- Finding a resolution proof can be viewed as a *search problem*:
  - *states* are sets of clauses;
  - *initial state* is the initial set of clauses plus the negation of the clause to be derived;
  - *goal states* are a set of clauses containing the empty clause;
  - a single *operator* - resolution rule - which may be applicable to many pairs of clauses in a state
- The *state space* is all possible sets of clauses that can be derived from the initial state by resolution

Searching for a Proof

- Any of the systematic (uninformed or informed) search techniques can be used to search for a resolution proof
- Branching factor of the search is determined by the number of resolvable clauses/literals (possible resolvents) in a given state
- High branching factor means that theorem provers often use *depth-first* search techniques (low memory requirements)
- For example, problems where the set of clauses contains all possible combinations of n positive and negative literals have a $O(2^n)$ worst-case branching factor

Proofs and Solutions

- So far we have considered *entailment* - if α is *true* then β must be *true*
- In many cases we also want to know what the proof is
  - e.g., planning can be formulated as a theorem proving problem - steps in the proof correspond to steps in the plan; is the goal reachable from the initial state, and if so how?
- In other cases we simply want to know if a given set of formulas is *satisfiable*
  - e.g., checking whether a student can take two modules which might clash, the steps in the proof are not of interest
- Local search techniques can be used in these cases

Local Search & Satisfiability

- Local search algorithms such as hill-climbing and simulated annealing can be applied directly to satisfiability problems
- States are complete assignments of *true* or *false* to each atomic proposition (i.e., states are models)
- Operators simply flip the truth value of each atomic proposition (so there are *n* applicable operators in each state)
- Evaluation function counts the number of unsatisfied clauses
- State space landscape usually contains lots of local minima

Example: 8 Queens

- We can formulate the eight queens problem as a satisfiability problem
- The queen in each column must be in one of eight rows

$$(q_{1,1} \lor q_{1,2} \lor \ldots \lor q_{1,8}) \land \ldots \land (q_{8,1} \lor \ldots \lor q_{8,8})$$

- Where $q_{1,1}$ is a proposition that states that the queen in column 1 is placed in row 1, and so on
- We also need clauses which rule out particular assignments of queens to rows, e.g.,:

$$(\neg q_{1,1} \lor \neg q_{2,2}) \land \ldots \land (\neg q_{1,1} \lor \neg q_{8,8})$$

WALKSAT

- WALKSAT is a simple and effective local search algorithm for satisfiability problems
- On each iteration, the algorithm picks an unsatisfied clause and changes the truth value of one of the atomic propositions in the clause
- To pick the atomic proposition whose truth value will be flipped, it either:
  - flips whichever atomic proposition maximises the number of satisfied clauses
  - chooses an atomic proposition at random
- Halts when a satisfying assignment is found or when the number of iterations reaches a pre-specified limit

WALKSAT properties

- If WALKSAT returns a model then the input clauses are satisfiable
- If no model is found within the specified number of iterations, it is likely that the clauses are not satisfiable, but this is not a proof
- Given an infinite number of iterations, WALKSAT will eventually return a model (if one exists) - however if the clauses are unsatisfiable the algorithm will never terminate
- Local search algorithms like WALKSAT are most useful when we believe that there is a satisfying assignment
- However they can’t detect unsatisfiability, which is what we need to show entailment