Inferences and Proofs, Resolution

CS156

Chris Pollett

Oct 20, 2014

Outline

- Search and Proofs
- Quiz
- Conjunctive Normal Form
- Resolution

Introduction

- Before the midterm we were talking about Logical Agents.
- We were focusing initially on propositional logic as a framework for representing our knowledge base as this is about the simplest logic in town.
- We were starting to talk about algorithms by which a logical agent could infer things about its environments (which in turn would hopefully inform the choice of its actions).
- We were looked at examples of how we could see if $KB \models \alpha$ holds in the Wumpus World setting using model checking. i.e., we check that in every model in which KB holds α holds as well.
- Checking all models of a collection of propositional logic statements involves checking all truth assignments to the variables. If one has n variables, one gets 2^n meaning this algorithm is slow.
- So we said we wanted to determine entailment via theorem proving -- applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.
- To do this we need three notions: logical equivalence, validity, and satisfiability. We write $a \equiv b$ (a is **logically equivalent to** b) if $a \models b$ and $b \models a$. A statement is **valid** if it is true in all models. This is related to entailment by the deduction theorem). A statement is **satisfiable** if there is some model in which it is true.
- We gave a list of common logical equivalences for propositional logic. Things like $(\neg(A \wedge B) \equiv (\neg A \vee \neg B)$).

- We used logical equivalence together with two rules of inferences: Modus Ponens and AND-elimination to show how a Wumpus World logical agent could deduce the contents of an as yet to be seen square.
- To start today, let's briefly spend a moment reviewing that proof, and then move on to a second theorem proving system known as resolution.

An Example

- Recall from last day we derived $\neg P_{1,2}$ from the following knowledge base statements R_i using model checking.
  1. *There is no pit in [1,1]:*
     $R_1 : \neg P_{1,1}$
  2. *A square is breezy iff there is a pit in a neighboring square. The has to be stated for each square: for now we include just the relevant squares:*
     $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
     $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
  3. *The preceding sentences are true in all wumpus worlds. Now we include the breeze percept for the first two squares visited in the specific world the agent is in:*
     $R_4 : \neg B_{1,1}$
     $R_5 : B_{2,1}$
- Alternatively, we can come up with a (propositional) **proof** of $\neg P_{1,2}$ from the above knowledge base using rules of inferences.
- A proof consists of a sequence of lines $R_1, ...R_m$ where each R_i is either from the knowledge base, a valid propositional statement, or follows from some R_j and R_k where $j < i$ and $k < i$ by some rule of (propositional) inference.

The proof of $\neg P_{1,2}$

- Applying Biconditional-elimination to R_2 gives:
  $R_6 : \big(B_{1,1} \Rightarrow \big(P_{1,2} \vee P_{2,1}\big)\big) \wedge \big(\big(P_{1,2} \vee P_{2,1}\big) \Rightarrow B_{1,1}\big)$
- Applying And-Elimination to R_6 gives:
  $R_7 : \big(\big(P_{1,2} \vee P_{2,1}\big) \Rightarrow B_{1,1}\big)$
- Logical equivalence for contraposition gives:
  $R_8 : \big(\neg B_{1,1} \Rightarrow \neg\big(P_{1,2} \vee P_{2,1}\big)\big)$
- Modus Ponens of R_4 and R_8 give:
  $R_9 : \neg\big(P_{1,2} \vee P_{2,1}\big)$
- Applying DeMorgan's rule to R_9 yields
  $R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$
- The desired statement follows from R_{10} by And-Elimination

Search and Proofs

- Each line in the proof we just gave can be checked mechanically (say by a program) to determine it is either from the KB or follows from preceding lines by one of our rules of inference.
- We found this proof by looking in our textbook, but we could imagine framing proof search as a problem and using one of our algorithms from earlier in the semester to find proofs:
  - *Initial State: the initial knowledge base*
  - *Actions: The set of actions consist of all inference rules applied to all the sentences that match the top half of an inference rule*
  - *Result: The result of an action is to add the bottom half of the inference rule to our KB*
  - *Goal: The goal is a KB the contains the statement we are trying to prove*
- This gives an alternative to model-checking which is often faster as the proof can ignore irrelevant propositions no matter how many there are.
- Notice this algorithm and propositional logic illustrate a property of knowledge bases known as **monotonicity**: that the set of entailed sentences can only increase as information is added to the database.
- There are actually logics known as **nonmonotonic logics** (extensively studied in AI as connected with human reasoning) for which this property fails.

Quiz

Which of the following is true?

1. Our KB-Agent pseudo-code only returned actions, it didn't tell the database about them in any way.
2. DeMorgan's Laws are examples of logical equivalence.
3. Another name for proof by contradiction is Modus Ponens.

Proof by Resolution

- So far we have looked at making inference algorithms which are sound; we have not considered the completeness of our algorithms.
- We next consider a proof system that has a single inference rule, **resolution**, which is known to be sound and complete.
- In its simplest form, the rule says that if we known $(A \vee B)$ and we know $\neg B$ then we can conclude A.
- We can generalize this to:
$$\frac{L_1 \vee ... \vee L_m,\ \ \neg L_m \vee L_1\,' \vee ... \vee L_n\,'}{L_1 \vee ... \vee L_{m-1} \vee L_1\,' \vee ... \vee L_n\,'}$$
In the above, L_i and $L_j\,'$ are **literals**: either variables or their negation.
- For a variable X, we often write $\overline{X}$ for $\neg X$.
- That the top of the inference logically implies the bottom can be check using model checking so the rule above is **sound**.
- The bottom line of this inference is called the **resolvent**.
- If we don't have any L_j"s then it is called **unit resolution**. That is, the right hand clause consists of just $\overline{L_m}$. (There are algorithms that work better for this case).
- You often see clauses written as $\{L_1, ..., L_m\}$ rather than use $\vee$.
- Set notation is used because $\{L, L\}$ is logically equivalent to $\{L\}$.

Conjunctive Normal Form

- Notice resolution works on an AND of OR's of literals.
- Any propositional formula can be rewritten into one which is equivalent to an AND of ORs or literals. This equivalent form is called **Conjunctive Normal Form**
- To see this just write down the truth table for the formula, and consider the rows for which the formula is false. That one of these rows doesn't happen can be written as an OR of literals. That none of these rows happens, is just an AND of these ORs.
- Another way to see this is via formula rewriting: One uses implication and biconditional elimination, followed by DeMorgan laws

Example of Converting to CNF

- Consider the statement $(A \wedge B) \vee \left(\overline{A} \wedge \overline{B}\right) \vee (A \wedge \neg B)$. If we view True as 1 and False as 0. This expresses $A \geq B$.
- We could express $A = \max(A, B, C)$ as $A \geq B \wedge A \geq C$.
- This is an AND of ORs of ANDs as written, so not CNF.
- Its truth table looks:

A	B	C	A = max(A, B, C)
True	True	True	True
True	True	False	True
True	False	True	True
True	False	False	True
False	True	True	False
False	True	False	False
False	False	True	False
False	False	False	True

- To make a CNF, we look at the three false rows. If the variable X for a column has a True in it we take that variable, if it has a false we take take $\overline{X}$. So the row False, True, False, becomes $A \vee \overline{B} \vee C$.
- This formula asserts that row didn't happen.
- So the CNF for the whole formula is: $\left(A \vee \overline{B} \vee \overline{C}\right) \wedge \left(A \vee \overline{B} \vee C\right) \wedge \left(A \vee B \vee \overline{C}\right)$
- As a collection of clauses we would write: $\left\{A, \overline{B}, \overline{C}\right\}, \left\{A, \overline{B}, C\right\}, \left\{A, B, \overline{C}\right\}$.

An Algorithm For Resolution

- Inference procedures based on resolution work by trying to produce a refutation
- We take $KB \wedge \neg\alpha$ (that is, the negation of $KB \Rightarrow \alpha$) and convert it to a CNF formula.
- We then cycle over pairs of clauses and for each pair that contains a resolvable literal we resolve the clauses.
- We keep repeating until no new clauses can be derived.
- If we ever resolve and get the empty clause (equivalent to false), then we know KB entails α.

Example

- Consider the statement about natural numbers: $K \leq M$ or $M \leq K$.
- To use propositional logic and resolution to show such a statement, we might consider n-bit variants of the statement, and show each n-bit variant is true.
- If we know, K and M are n-bit numbers, we can use propositional variables $K_1, ...K_n$ and $M_1, ..., M_n$ to represent these numbers. So $K \leq M$ can be expressed as the OR over j of statements saying:
  $(K_n \Leftrightarrow M_n) \wedge ... \wedge \left(K_{j+1} \Leftrightarrow M_{j+1}\right) \wedge \left(K_j \wedge \neg M_j\right)$.
  I.e., the high order bits are the same until j where K_j must be True and M_j must be False.
- Let's give a resolution refutation of the 1-bit version of $K \leq M$ or $M \leq K$.
- This is just the statement:
  $(K_1 \Leftrightarrow M_1) \vee (M_1 \wedge \neg K_1) \vee (K_1 \Leftrightarrow M_1) \vee (K_1 \wedge \neg M_1)$
  which is the same as
  $(K_1 \Leftrightarrow M_1) \vee (M_1 \wedge \neg K_1) \vee (K_1 \wedge \neg M_1)$
- As a first step, we convert the negation of this formula to CNF using DeMorgan's Laws:
  $\{K_1, M_1\}, \{\overline{K_1}, \overline{M_1}\}, \{\overline{K_1}, M_1\}, \{K_1, \overline{M_1}\}$
- Resolving the first and third clause gives $\{M_1, M_1\} = \{M_1\}$. Resolving the second and fourth clause gives $\{\overline{M_1}, \overline{M_1}\} = \{\overline{M_1}\}$. Then resolving these two gives the empty clause, completing the refutation.

Completeness

- To show resolution is complete we need to show that if a set of clauses is unsatisfiable then the resolution closure contains the empty clause.
- To prove this we actually show the contrapositive: If the resolution closure (RC) does not contain the empty clause then there is a satisfying assignment.
- The following procedure give us the satisfying assignment: For i from 1 to k (where i cycles over variable indexes):
  - *If there is a clause in RC(S) containing the literal* $\neg P_i$ *and all its other literals are false under the assignment chosen for* $P_1, ...P_{i-1}$ *then assign false to* P_i.
  - *Otherwise, assign true to* P_i.

Example No Refutation

- Suppose we only had the following two clauses from the example a couple of slides back:
  $\{\overline{K_1}, M_1\}, \{K_1, \overline{M_1}\}$
- Then the resolution closure of these two clauses is:
  $\{\overline{K_1}, M_1\}, \{K_1, \overline{M_1}\}, \{M_1, \overline{M_1}\}, \{K_1, \overline{K_1}\}$.
- Letting $P_1 = K_1$, and $P_2 = M_1$. This gives:
  $\{\overline{P_1}, P_2\}, \{P_1, \overline{P_2}\}, \{P_2, \overline{P_2}\}, \{P_1, \overline{P_1}\}$.
- When $i = 1$, in the for loop of the last slide, no assignments have been made so there is no clause in which all the other variables have been assigned False containing $\overline{P_1}$. So we set P_1 to True in our model.
- When $i = 2$, again we don't find such a clause, so P_2 is assigned true.
- Substituting back for K_1 and M_1 we get both of these would be assigned true by this algorithm. Notice this assignment does satisfy both clauses in:
  $\{\overline{K_1}, M_1\}, \{K_1, \overline{M_1}\}$
- Suppose we had started with the three clauses:
  $\{\overline{P_1}\}, \{\overline{P_1}, P_2\}, \{P_1, \overline{P_2}\}$
- In this case, the resolution closure would be:
  $\{\overline{P_1}\}, \{\overline{P_2}\}, \{\overline{P_1}, P_2\}, \{P_1, \overline{P_2}\}, \{P_2, \overline{P_2}\}, \{P_1, \overline{P_1}\}$.
- In the for loop when $i = 1$, the first clause will force P_1 to be false; when $i = 2$, the second clause, will force P_2 to be false. One can verify the $P_1 \rightarrow$False, $P_2 \rightarrow$False assignment in fact makes all of the clauses we started with true.