CS 228 : Logic in Computer Science

Krishna. S

Completeness

$\varphi_1, \ldots, \varphi_n \models \psi \Rightarrow \varphi_1, \ldots, \varphi_n \vdash \psi$

Whenever $\varphi_1, \ldots, \varphi_n$ semantically entail ψ, then ψ can be proved from $\varphi_1, \ldots, \varphi_n$. The proof rules are complete

Completeness : 3 steps

- Given $\varphi_1, \ldots, \varphi_n \models \psi$
- Step 1: Show that $\models \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$
- Step 2: Show that $\vdash \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$
- Step 3: Show that $\varphi_1, \ldots, \varphi_n \vdash \psi$

Completeness : Step 1

- Assume $\varphi_1, \ldots, \varphi_n \models \psi$. Whenever all of $\varphi_1, \ldots, \varphi_n$ evaluate to true, so does ψ.
- If $\not\models \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$, then ψ evaluates to false when all of $\varphi_1, \ldots, \varphi_n$ evaluate to true, a contradiction.
- Hence, $\models \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$.

Completeness : Step 2

- Given $\models \psi$, show that $\vdash \psi$
- Assume $p_1, \ldots, p_n$ are the propositional variables in ψ. We know that all the 2^n assignments of values to $p_1, \ldots, p_n$ make ψ true.
- Using this insight, we have to give a proof of ψ.

Completeness : Step 2

Truth Table to Proof

Let φ be a formula with variables $p_1, \ldots, p_n$. Let $\mathcal{T}$ be the truth table of φ, and let l be a line number in $\mathcal{T}$. Let $\hat{p}_i$ represent p_i if p_i is assigned true in line l, and let it denote $\neg p_i$ if p_i is assigned false in line l. Then

1. $\hat{p}_1, \ldots, \hat{p}_n \vdash \varphi$ if φ evaluates to true in line l
2. $\hat{p}_1, \ldots, \hat{p}_n \vdash \neg\varphi$ if φ evaluates to false in line l

Truth Table to Proof

- Structural Induction on φ.

Truth Table to Proof

- Structural Induction on φ.
- Base : If $\varphi = p$, a proposition, then we have $p \vdash p$ and $\neg p \vdash \neg p$.

Truth Table to Proof

- Structural Induction on φ.
- Base : If $\varphi = p$, a proposition, then we have $p \vdash p$ and $\neg p \vdash \neg p$.
- Assume for formulae of size $\leqslant k - 1$ (size=height of the parse tree). What is a parse tree?

Truth Table to Proof

- Structural Induction on φ.
- Base : If $\varphi = p$, a proposition, then we have $p \vdash p$ and $\neg p \vdash \neg p$.
- Assume for formulae of size $\leqslant k - 1$ (size=height of the parse tree). What is a parse tree?
- Case Negation : $\varphi = \neg\varphi_1$

Truth Table to Proof

- Structural Induction on φ.
- Base : If $\varphi = p$, a proposition, then we have $p \vdash p$ and $\neg p \vdash \neg p$.
- Assume for formulae of size $\leqslant k - 1$ (size=height of the parse tree). What is a parse tree?
- Case Negation : $\varphi = \neg\varphi_1$
  - Assume φ evaluates to true in line l of $\mathcal{T}$. Then φ_1 evaluates to false in line l. By inductive hypothesis, $\hat{p}_1, \ldots, \hat{p}_n \vdash \neg\varphi_1$.

Truth Table to Proof

- Structural Induction on φ.
- Base : If $\varphi = p$, a proposition, then we have $p \vdash p$ and $\neg p \vdash \neg p$.
- Assume for formulae of size $\leqslant k - 1$ (size=height of the parse tree). What is a parse tree?
- Case Negation : $\varphi = \neg\varphi_1$
  - Assume φ evaluates to true in line l of $\mathcal{T}$. Then φ_1 evaluates to false in line l. By inductive hypothesis, $\hat{p}_1, \ldots, \hat{p}_n \vdash \neg\varphi_1$.
  - Assume φ evaluates to false in line l of $\mathcal{T}$. Then φ_1 evaluates to true in line l. By inductive hypothesis, $\hat{p}_1, \ldots, \hat{p}_n \vdash \varphi_1$. Use the $\neg\neg i$ rule to obtain a proof of $\hat{p}_1, \ldots, \hat{p}_n \vdash \neg\neg\varphi_1$.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.
  - If φ evaluates to false in line l, then φ_1 evaluates to true and φ_2 to false. Let $\{q_1, \ldots, q_k\}$ be the variables of φ_1 and let $\{r_1, \ldots, r_j\}$ be the variables in φ_2. $\{q_1, \ldots, q_k\} \cup \{r_1, \ldots, r_j\} = \{p_1, \ldots, p_n\}$.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.
  - If φ evaluates to false in line l, then φ_1 evaluates to true and φ_2 to false. Let $\{q_1, \ldots, q_k\}$ be the variables of φ_1 and let $\{r_1, \ldots, r_j\}$ be the variables in φ_2. $\{q_1, \ldots, q_k\} \cup \{r_1, \ldots, r_j\} = \{p_1, \ldots, p_n\}$.
  - By inductive hypothesis, $\hat{q}_1, \ldots, \hat{q}_k \models \varphi_1$ and $\hat{r}_1, \ldots, \hat{r}_j \models \neg\varphi_2$. Then, $\hat{p}_1, \ldots, \hat{p}_n \models \varphi_1 \wedge \neg\varphi_2$.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.
  - If φ evaluates to false in line l, then φ_1 evaluates to true and φ_2 to false. Let $\{q_1, \ldots, q_k\}$ be the variables of φ_1 and let $\{r_1, \ldots, r_j\}$ be the variables in φ_2. $\{q_1, \ldots, q_k\} \cup \{r_1, \ldots, r_j\} = \{p_1, \ldots, p_n\}$.
  - By inductive hypothesis, $\hat{q}_1, \ldots, \hat{q}_k \models \varphi_1$ and $\hat{r}_1, \ldots, \hat{r}_j \models \neg\varphi_2$. Then, $\hat{p}_1, \ldots, \hat{p}_n \models \varphi_1 \wedge \neg\varphi_2$.
  - Prove that $\varphi_1 \wedge \neg\varphi_2 \vdash \neg(\varphi_1 \rightarrow \varphi_2)$.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.

Truth Table to Proof

- Case $\to$: $\varphi = \varphi_1 \to \varphi_2$.
  - If φ evaluates to true in line l, then there are 3 possibilities. If both φ_1, φ_2 evaluate to true, then we have $\hat{p}_1, \ldots, \hat{p}_n \models \varphi_1 \land \varphi_2$. Proving $\varphi_1 \land \varphi_2 \vdash \varphi_1 \to \varphi_2$, we are done.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.
  - If φ evaluates to true in line l, then there are 3 possibilities. If both φ_1, φ_2 evaluate to true, then we have $\hat{p}_1, \ldots, \hat{p}_n \models \varphi_1 \wedge \varphi_2$. Proving $\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$, we are done.
  - If both φ_1, φ_2 evaluate to false, then we have $\hat{p}_1, \ldots, \hat{p}_n \models \neg\varphi_1 \wedge \neg\varphi_2$. Proving $\neg\varphi_1 \wedge \neg\varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$, we are done.

Truth Table to Proof

- Case $\rightarrow$: $\varphi = \varphi_1 \rightarrow \varphi_2$.
  - If φ evaluates to true in line l, then there are 3 possibilities. If both φ_1, φ_2 evaluate to true, then we have $\hat{p}_1, \ldots, \hat{p}_n \models \varphi_1 \wedge \varphi_2$. Proving $\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$, we are done.
  - If both φ_1, φ_2 evaluate to false, then we have $\hat{p}_1, \ldots, \hat{p}_n \models \neg\varphi_1 \wedge \neg\varphi_2$. Proving $\neg\varphi_1 \wedge \neg\varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$, we are done.
  - Last, if φ_1 evaluates to false and φ_2 evaluates to true, then we have $\hat{p}_1, \ldots, \hat{p}_n \models \neg\varphi_1 \wedge \varphi_2$. Proving $\neg\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$, we are done.

Truth Table to Proof

- Prove the cases $\wedge, \vee$.

On An Example

We know $\models (p \land q) \to p$. Using this fact, show that $\vdash (p \land q) \to p$.

- $p, q \vdash (p \land q) \to p$
- $\neg p, q \vdash (p \land q) \to p$
- $p, \neg q \vdash (p \land q) \to p$
- $\neg p, \neg q \vdash (p \land q) \to p$

Now, combine the 4 proofs above to give a single proof for
$\vdash (p \land q) \to p$.

Completeness : Steps 2, 3

- Step 2: From $\models \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$, use LEM on all the propositional variables of $\varphi_1, \ldots, \varphi_n, \psi$ to obtain $\vdash \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$.

Completeness : Steps 2, 3

- Step 2: From $\models \varphi_1 \to (\varphi_2 \to (\dots(\varphi_n \to \psi)\dots))$, use LEM on all the propositional variables of $\varphi_1, \dots, \varphi_n, \psi$ to obtain $\vdash \varphi_1 \to (\varphi_2 \to (\dots(\varphi_n \to \psi)\dots))$.
- Step 3: Take the proof $\vdash \varphi_1 \to (\varphi_2 \to (\dots(\varphi_n \to \psi)\dots))$. This proof has *n* nested boxes, the *i*th box opening with the assumption φ_i. The last box closes with the last line ψ. Hence, the line immediately after the last box is $\varphi_n \to \psi$.

Completeness : Steps 2, 3

- Step 2: From $\models \varphi_1 \to (\varphi_2 \to (\dots(\varphi_n \to \psi)\dots))$, use LEM on all the propositional variables of $\varphi_1, \dots, \varphi_n, \psi$ to obtain $\vdash \varphi_1 \to (\varphi_2 \to (\dots(\varphi_n \to \psi)\dots))$.
- Step 3: Take the proof $\vdash \varphi_1 \to (\varphi_2 \to (\dots(\varphi_n \to \psi)\dots))$. This proof has n nested boxes, the ith box opening with the assumption φ_i. The last box closes with the last line ψ. Hence, the line immediately after the last box is $\varphi_n \to \psi$.
- In a similar way, the $(n-1)$th box has as its last line $\varphi_n \to \psi$, and hence, the line immediately after this box is $\varphi_{n-1} \to (\varphi_n \to \psi)$ and so on.

Completeness : Steps 2, 3

- Step 2: From $\models \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$, use LEM on all the propositional variables of $\varphi_1, \ldots, \varphi_n, \psi$ to obtain $\vdash \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$.
- Step 3: Take the proof $\vdash \varphi_1 \to (\varphi_2 \to (\ldots(\varphi_n \to \psi)\ldots))$. This proof has n nested boxes, the ith box opening with the assumption φ_i. The last box closes with the last line ψ. Hence, the line immediately after the last box is $\varphi_n \to \psi$.
- In a similar way, the $(n-1)$th box has as its last line $\varphi_n \to \psi$, and hence, the line immediately after this box is $\varphi_{n-1} \to (\varphi_n \to \psi)$ and so on.
- Add premises $\varphi_1, \ldots, \varphi_n$ on the top. Use MP on the premises, and the lines after boxes 1 to n in order to obtain ψ.

Summary

Propositional Logic is sound and complete.

Normal Forms

- A literal is a propositional variable p or its negation $\neg p$. These are referred to as positive and negative literals respectively.

Normal Forms

- A literal is a propositional variable p or its negation $\neg p$. These are referred to as positive and negative literals respectively.
- A formula F is in CNF if it is a conjunction of a disjunction of literals.

$$F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} L_{i,j}$$

each $L_{i,j}$ is a literal.

Normal Forms

- A literal is a propositional variable p or its negation $\neg p$. These are referred to as positive and negative literals respectively.
- A formula F is in CNF if it is a conjunction of a disjunction of literals.

$$F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} L_{i,j}$$

  each $L_{i,j}$ is a literal.
- A formula F is in DNF if it is a disjunction of a conjunction of literals.

$$F = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} L_{i,j}$$

  each $L_{i,j}$ is a literal.

Normal Forms

In the following, equivalent stands for semantically equivalent

Let F be a formula in CNF and let G be a formula in DNF. Then $\neg F$ is equivalent to a formula in DNF and $\neg G$ is equivalent to a formula in CNF.

Normal Forms

In the following, equivalent stands for semantically equivalent

Let F be a formula in CNF and let G be a formula in DNF. Then $\neg F$ is equivalent to a formula in DNF and $\neg G$ is equivalent to a formula in CNF.

Every formula F is equivalent to some formula F_1 in CNF and some formula F_2 in DNF.

CNF Algorithm

Given a formula F, $(x \to [\neg(y \vee z) \wedge \neg(y \to x)])$

- Replace all subformulae of the form $F \to G$ with $\neg F \vee G$, and all subformulae of the form $F \leftrightarrow G$ with $(\neg F \vee G) \wedge (\neg G \vee F)$. When there are no more occurrences of $\to, \leftrightarrow$, proceed to the next step.

CNF Algorithm

Given a formula F, $(x \to [\neg(y \vee z) \wedge \neg(y \to x)])$

- Replace all subformulae of the form $F \to G$ with $\neg F \vee G$, and all subformulae of the form $F \leftrightarrow G$ with $(\neg F \vee G) \wedge (\neg G \vee F)$. When there are no more occurrences of $\to, \leftrightarrow$, proceed to the next step.
- Get rid of all double negations : Replace all subformulae
  - $\neg\neg G$ with G,
  - $\neg(G \wedge H)$ with $\neg G \vee \neg H$
  - $\neg(G \vee H)$ with $\neg G \wedge \neg H$

  When there are no more such subformulae, proceed to the next step.

CNF Algorithm

Given a formula F, $(x \to [\neg(y \vee z) \wedge \neg(y \to x)])$

- Replace all subformulae of the form $F \to G$ with $\neg F \vee G$, and all subformulae of the form $F \leftrightarrow G$ with $(\neg F \vee G) \wedge (\neg G \vee F)$. When there are no more occurrences of $\to, \leftrightarrow$, proceed to the next step.
- Get rid of all double negations : Replace all subformulae
  - $\neg\neg G$ with G,
  - $\neg(G \wedge H)$ with $\neg G \vee \neg H$
  - $\neg(G \vee H)$ with $\neg G \wedge \neg H$

  When there are no more such subformulae, proceed to the next step.
- Distribute $\vee$ wherever possible.

The resultant formula F_1 is in CNF and is provably equivalent to F.
$[(\neg x \vee \neg y) \wedge (\neg x \vee \neg z)] \wedge [(\neg x \vee y) \wedge (\neg x \vee \neg x)]$

The Hardness of SAT

- Given a formula φ how to check if φ is satisfiable?
- Given a formula φ how to check if φ is unsatisfiable?
- SAT is NP-complete

Polynomial Time Formula Classes

Horn Formulae

- A Horn Formula is a particularly nice kind of CNF formula, which can be quickly checked for satisfiability.
- Programming languages Prolog and Datalog are based on Horn clauses in first order logic

Horn Formulae

- A Horn Formula is a particularly nice kind of CNF formula, which can be quickly checked for satisfiability.
- Programming languages Prolog and Datalog are based on Horn clauses in first order logic
- A formula F is a Horn formula if it is in CNF and every disjunction contains atmost one positive literal.

Horn Formulae

- A Horn Formula is a particularly nice kind of CNF formula, which can be quickly checked for satisfiability.
- Programming languages Prolog and Datalog are based on Horn clauses in first order logic
- A formula F is a Horn formula if it is in CNF and every disjunction contains atmost one positive literal.
- $p \wedge (\neg p \vee \neg q \vee r) \wedge (\neg a \vee \neg b)$ is Horn, but $a \vee b$ is not Horn.

Horn Formulae

- A Horn Formula is a particularly nice kind of CNF formula, which can be quickly checked for satisfiability.
- Programming languages Prolog and Datalog are based on Horn clauses in first order logic
- A formula F is a Horn formula if it is in CNF and every disjunction contains atmost one positive literal.
- $p \wedge (\neg p \vee \neg q \vee r) \wedge (\neg a \vee \neg b)$ is Horn, but $a \vee b$ is not Horn.
- A basic Horn formula is one which has no $\wedge$. Every Horn formula is a conjunction of basic Horn formulae.

Horn Formulae

- Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.

Horn Formulae

- Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- Basic Horn with both positive and negative literals are written as an implication $p \wedge q \wedge \cdots \wedge r \rightarrow s$ involving only positive literals.

Horn Formulae

- Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- Basic Horn with both positive and negative literals are written as an implication $p \wedge q \wedge \cdots \wedge r \rightarrow s$ involving only positive literals.
- Basic Horn with no negative literals are of the form p and are written as $\top \rightarrow p$.

Horn Formulae

- Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- Basic Horn with both positive and negative literals are written as an implication $p \wedge q \wedge \cdots \wedge r \rightarrow s$ involving only positive literals.
- Basic Horn with no negative literals are of the form p and are written as $\top \rightarrow p$.
- Basic Horn with no positive literals are written as $p \wedge q \wedge \cdots \wedge r \rightarrow \bot$.

Horn Formulae

- Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- Basic Horn with both positive and negative literals are written as an implication $p \wedge q \wedge \cdots \wedge r \to s$ involving only positive literals.
- Basic Horn with no negative literals are of the form p and are written as $\top \to p$.
- Basic Horn with no positive literals are written as $p \wedge q \wedge \cdots \wedge r \to \bot$.
- Thus, a Horn formula is written as a conjunction of implications.