Normal Forms of Propositional Logic

Bow-Yaw Wang

Institute of Information Science
Academia Sinica, Taiwan

September 22, 2021

Outline

1. Normal Forms
   - Semantic equivalence, satisfiability, and validity
   - Conjunctive normals forms and validity
   - Horn clauses and satisfiability

Semantically Equivalence and Validity

- Consider two formulae $\phi_1 \land \phi_2$ and $\phi_2 \land \phi_1$.
- Intuitively, $\phi_1 \land \phi_2$ and $\phi_2 \land \phi_1$ should have the same "meaning."
- More formally, two formulae ϕ and ψ have the same meaning if their truth tables coincide.

Definition

Let ϕ and ψ be propositional logic formulae. ϕ and ψ are <u>semantically equivalent</u> (written $\phi \equiv \psi$) if both $\phi \vDash \psi$ and $\psi \vDash \phi$ hold.

- Examples

$$
\begin{array}{rcl}
p \implies q & \equiv & \neg q \implies \neg p \\
p \land q \implies p & \equiv & r \lor \neg r
\end{array}
\qquad
\begin{array}{rcl}
p \implies q & \equiv & \neg p \lor q \\
p \land q \implies r & \equiv & p \implies (q \implies r)
\end{array}
$$

- A formula ϕ is valid if it is a tautology.

Definition

Let ϕ be a propositional logic formula. ϕ is <u>valid</u> if $\vDash \phi$.

Semantic Entailment and Validity

Lemma

Let $\phi_1, \phi_2, \ldots, \phi_n, \psi$ be propositional logic formulae. $\phi_1, \phi_2, \ldots, \phi_n \vDash \psi$ iff $\vDash \phi_1 \implies (\phi_2 \implies \cdots \implies (\phi_n \implies \psi))$.

Proof.

Suppose $\vDash \phi_1 \implies (\phi_2 \implies \cdots \implies (\phi_n \implies \psi))$ Consider any valuation. If $\phi_1, \phi_2, \ldots, \phi_n$ evaluate to T under the valuation, ψ must evaluate to T since $\vDash \phi_1 \implies (\phi_2 \implies \cdots \implies (\phi_n \implies \psi))$. Hence $\phi_1, \phi_2, \ldots, \phi_n \vDash \psi$.

The other direction is proved in Step 1 of the completeness theorem. □

Conjunctive Normal Form (CNF)

Definition

A literal L is either an atom p or its negation $\neg p$. A clause D is a disjunction of literals. A formula C is in conjunctive normal form (CNF) if it is a conjunction of clauses.

$$\begin{array}{rcl} L & ::= & p \mid \neg p \\ D & ::= & L \mid L \vee D \\ C & ::= & D \mid D \wedge C \end{array}$$

- Examples: $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$, $(p \vee r) \wedge (\neg p \vee r) \wedge (p \vee \neg r)$

Validity of CNF Formulae

Lemma

A clause $L_1 \vee L_2 \vee \cdots \vee L_m$ is valid iff there is a propositional atom p such that L_i is p and L_j is $\neg p$ for some $1 \leq i, j \leq m$.

Proof.

Without loss of generality, assume $L_1 = p$ and $L_2 = \neg p$. Then $p \vee \neg p \vee L_3 \vee \cdots \vee L_m$ evaluates to T for any valuation. The clause is valid. Conversely, consider the valuation where all literals evaluate to F. This is possible since every literal L_i has no negation in the clause. The clause evaluates to F under the valuation. □

- Examples:
  - $p \vee q \vee q \vee \neg p \vee r$ is valid;
  - $p \vee \neg q \vee r \vee \neg q$ is not valid (consider $\{p \mapsto \mathrm{F}, q \mapsto \mathrm{T}, r \mapsto \mathrm{F}\}$).
- For any propositional logic formula ϕ in CNF, the validity of ϕ can be checked in linear time.

From Truth Tables to Conjunctive Normal Form

- Suppose we have the truth table for a formula ϕ with propositional atoms $p_1, p_2, \ldots, p_n$.
- For each line l where ϕ evaluates to F, construct a clause ψ_l as follows.
  - $\psi_l = L_{l,1} \vee L_{l,2} \vee \cdots \vee L_{l,n}$ where $L_{l,j} = \neg p_j$ if p_j is T at line l; otherwise $L_{l,j} = p_j$.
- Then $\phi \equiv \psi_1 \wedge \psi_2 \wedge \cdots \psi_m$ where ψ_l's are contructed for every line evaluating ϕ to F.
- Observe that $\psi_1 \wedge \psi_2 \wedge \cdots \psi_m$ is F iff ψ_l is F for some $1 \leq l \leq m$. $\psi_l = L_{l,1} \vee L_{l,2} \vee \cdots \vee L_{l,n}$ is F iff $L_{l,j}$ is F for every $1 \leq j \leq n$. $L_{l,j}$ is F iff p_j has its truth value at line l.
- In other words, $\psi_1 \wedge \psi_2 \wedge \cdots \psi_m$ is F under a valuation iff the valuation evaluates ϕ to F in ϕ's truth table.

From Truth Tables to Conjunctive Normal Form

Example

Translate $p \vee q \implies q \wedge \neg r$ into CNF.

Proof.

p	q	r	$p \vee q \implies q \wedge \neg r$	p	q	r	$p \vee q \implies q \wedge \neg r$
F	F	F	T	T	F	F	F
F	F	T	T	T	F	T	F
F	T	F	T	T	T	F	T
F	T	T	F	T	T	T	F

p	q	r	ψ_I	p	q	r	ψ_I
F	T	T	$p \vee \neg q \vee \neg r$	T	F	F	$\neg p \vee q \vee r$
T	F	T	$\neg p \vee q \vee \neg r$	T	T	T	$\neg p \vee \neg q \vee \neg r$

$$p \vee q \implies q \wedge \neg r \equiv (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r). \qquad \square$$

Outline

1. Normal Forms
   - Semantic equivalence, satisfiability, and validity
   - Conjunctive normals forms and validity
   - Horn clauses and satisfiability

Validity Checking

- Given a propositional logic formula in conjunctive normal form, we can check the validity of the formula in linear time.
- Recall that a formula is valid iff it is a theorem.
- If we can translate any propositional logic formula into conjunctive normal form, we can check the validity of the formula!
- We know how to translate any logic formula to conjunctive normal form by its truth table.
  - This is not satisfactory. If we have to construct its truth table, we can check validity already.
- We will give an algorithm $\texttt{CNF}(\phi)$ to convert any propositional logic formula into conjunctive normal form without building its truth table.

From Formula to Conjunctive Normal Form

- Any propositional logic formula can be transformed to conjunctive normal form by the following equivalences:

$$\phi \implies \psi \equiv \neg\phi \vee \psi$$

$$\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi \qquad \neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$$

$$\phi \wedge (\psi_1 \vee \psi_2) \equiv (\phi \wedge \psi_1) \vee (\phi \wedge \psi_2)$$

$$\phi \vee (\psi_1 \wedge \psi_2) \equiv (\phi \vee \psi_1) \wedge (\phi \vee \psi_2)$$

- The algorithm $\texttt{CNF}(\phi)$ hence consists of three steps:
  - Remove every implication ($\implies$) from ϕ (Algorithm $\texttt{IMPL\_FREE}(\phi)$);
  - Push every negation ($\neg$) to literals (Algorithm $\texttt{NNF}(\phi)$);
  - Apply law of distribution (Algorithm $\texttt{PUSHDISJ}(\phi)$).

Algorithm IMPL_FREE(ϕ)

Input: ϕ : a logic formula
Output: ϕ' : all implications ($\implies$) in ϕ' are removed and $\phi' \equiv \phi$
switch ϕ **do**

- **case** ϕ is a literal: **do return** ϕ;
- **case** ϕ is $\neg\phi_1$: **do return** $\neg\texttt{IMPL\_FREE}(\phi_1)$;
- **case** ϕ is $\phi_1 \wedge \phi_2$: **do return** $\texttt{IMPL\_FREE}(\phi_1) \wedge \texttt{IMPL\_FREE}(\phi_2)$;
- **case** ϕ is $\phi_1 \vee \phi_2$: **do return** $\texttt{IMPL\_FREE}(\phi_1) \vee \texttt{IMPL\_FREE}(\phi_2)$;
- **case** ϕ is $\phi_1 \implies \phi_2$: **do return** $\texttt{IMPL\_FREE}(\neg\phi_1 \vee \phi_2)$;
- **otherwise do** `assert(0)`;

Algorithm 1: IMPL_FREE(ϕ)

Algorithm $\texttt{NNF}(\phi)$

Input: ϕ : a logic formula without implication ($\Longrightarrow$)
Output: ϕ' : only propositional atoms in ϕ' are negated and $\phi' \equiv \phi$
switch ϕ **do**
- **case** ϕ is a literal: **do return** ϕ;
- **case** ϕ is $\neg\neg\phi_1$: **do return** $\texttt{NNF}(\phi_1)$;
- **case** ϕ is $\phi_1 \land \phi_2$: **do return** $\texttt{NNF}(\phi_1) \land \texttt{NNF}(\phi_2)$;
- **case** ϕ is $\phi_1 \lor \phi_2$: **do return** $\texttt{NNF}(\phi_1) \lor \texttt{NNF}(\phi_2)$;
- **case** ϕ is $\neg(\phi_1 \land \phi_2)$: **do return** $\texttt{NNF}(\neg\phi_1 \lor \neg\phi_2)$;
- **case** ϕ is $\neg(\phi_1 \lor \phi_2)$: **do return** $\texttt{NNF}(\neg\phi_1 \land \neg\phi_2)$;
- **otherwise do** $\texttt{assert}(0)$;

Algorithm 2: $\texttt{NNF}(\phi)$

Definition

Let ϕ be a propositional logic formula. If only propositional atoms in ϕ are negated, ϕ is in negation normal form.

Algorithm $\mathtt{PUSHDISJ}(\phi)$

Input: ϕ : an NNF formula without implication ($\implies$)
Output: ϕ' : ϕ' is in CNF and $\phi' \equiv \phi$
switch $\underline{\phi}$ **do**
  case $\underline{\phi \text{ is a literal}}$: **do return** $\underline{\phi}$;
  case $\underline{\phi \text{ is } \phi_1 \wedge \phi_2}$: **do return** $\underline{\mathtt{PUSHDISJ}(\phi_1) \wedge \mathtt{PUSHDISJ}(\phi_2)}$;
  case $\underline{\phi \text{ is } \phi_1 \vee \phi_2}$: **do return** $\underline{\mathtt{DISTR}(\mathtt{PUSHDISJ}(\phi_1), \mathtt{PUSHDISJ}(\phi_2))}$;

Algorithm 3: $\mathtt{PUSHDISJ}(\phi)$

Input: η_1, η_2 : η_1, η_2 are in CNF
Output: ϕ' : ϕ' is in CNF and $\phi' \equiv \eta_1 \vee \eta_2$
if $\underline{\eta_1 \text{ is } \eta_{11} \wedge \eta_{12}}$ **then** **return** $\underline{\mathtt{DISTR}(\eta_{11}, \eta_2) \wedge \mathtt{DISTR}(\eta_{12}, \eta_2)}$;
else if $\underline{\eta_2 \text{ is } \eta_{21} \wedge \eta_{22}}$ **then** **return** $\underline{\mathtt{DISTR}(\eta_1, \eta_{21}) \wedge \mathtt{DISTR}(\eta_1, \eta_{22})}$;
else **return** $\underline{\eta_1 \vee \eta_2}$;

Algorithm 4: $\mathtt{DISTR}(\eta_1, \eta_2)$

Checking Validity Revisited

- Let ϕ be a propositional logic formula. Consider the following algorithm for checking its validity.
  1. Compute a CNF formula $\psi = \texttt{CNF}(\phi)$.
  2. Check the validity of ψ.

Validity and Satisfiability

Definition

Let ϕ be a propositional logic formula. ϕ is satisfiable if it evaluates to T under some valuation.

- Example: $p \lor q \implies p$ is satisfiable (consider $\{p \mapsto \mathrm{T}, q \mapsto \mathrm{T}\}$); it is not valid (consider $\{p \mapsto \mathrm{F}, q \mapsto \mathrm{T}\}$).

Proposition

Let ϕ be a propositional logic formula. ϕ is satisfiable iff $\neg\phi$ is not valid.

Proof.

Suppose ϕ evaluates to T under a valuation. Then $\neg\phi$ evaluates to F under the valuation. $\neg\phi$ is not valid.
Conversely, suppose $\neg\phi$ is not valid. Hence $\neg\phi$ evaluates to F under a valuation. Thus ϕ evaluates to T under the valuation. ϕ is satisfiable. $\square$

Satisfiability of Propositional Logic Formulae

- Let ϕ be a propositional logic formula. Consider the following algorithm for checking its satisfiability.
  1. Compute a CNF formula $\psi = \texttt{CNF}(\neg\phi)$.
  2. Check the validity of ψ.
  3. Return "ϕ is satisfiable" if ψ is not valid; Return "ϕ is not satisfiable" if ψ is valid.
- Recall that satisfiability of propositional logic formulae is an NP-complete problem.
- Is the above algorithm in polynomial time? Why?

Outline

1. Normal Forms
   - Semantic equivalence, satisfiability, and validity
   - Conjunctive normals forms and validity
   - Horn clauses and satisfiability

Horn Clauses

- Given a propositional logic formula in CNF, it is easy to check its validity; it is "hard" to check its satisfiability.
- We will consider a subclass of CNF formulae whose satisfiability can be checked efficiently.

Definition

A Horn formula is a propositional logic formula ϕ of the following form:

$$\begin{array}{rcl} P & ::= & \bot \mid \top \mid p \\ A & ::= & P \mid P \wedge A \\ C & ::= & A \implies P \\ H & ::= & C \mid C \wedge H. \end{array}$$

A clause of the form C is called a Horn clause.

- Example: $(p \wedge q \wedge s \implies \bot) \wedge (q \wedge r \implies p) \wedge (\top \implies s)$
- Nonexample: $(p \wedge \neg q \wedge s \implies \bot) \wedge (q \wedge r \implies p \wedge s) \wedge (p \vee r \implies s)$

Satisfiability of Horn Formulae

- Consider a Horn clause $P_1 \land P_2 \land \cdots \land P_n \implies Q$.
- If $P_1, P_2, \ldots, P_n$ are assigned to T, then Q must be T; otherwise, Q can be an arbitrary truth value.
- We hence have the following (informal) algorithm:
  1. Mark $\top$ if it occurs in the Horn formula ϕ;
  2. If there is a Horn clause $P_1 \land P_2 \land \cdots \land P_n \implies Q$ in ϕ such that all P_j for $1 \leq j \leq n$ are marked, mark Q;
  3. If $\bot$ is marked, print "The Horn formula ϕ is unsatisfiable."
  4. Print "The Horn formula ϕ is satisfiable."

Algorithm $\texttt{Horn}(\phi)$

Input: ϕ: ϕ is a Horn formula
Output: "unsatisfiable" if ϕ is unsatisfiable; otherwise "satisfiable."
mark all occurrences of $\top$ in ϕ;
while there is a Horn clause $P_1 \land P_2 \land \cdots \land P_n \implies Q$ in ϕ such that P_j are all marked but Q is not **do**
    mark Q;
if $\perp$ is marked **then** **return** "unsatisfiable" ;
else **return** "satisfiable" ;

Algorithm 5: $\texttt{Horn}(\phi)$

Satisfiability of Horn Formulae

Theorem

Let ϕ be a Horn formula with n propositional atoms. $\texttt{Horn}(\phi)$ *runs at most $n+1$ iterations and decides the satisfiability of ϕ correctly.*

Proof.

At each iteration, an unmarked atom will be marked. Since there are n atoms, there are at most $n+1$ iterations.

By induction on the number of iterations, we show that "all marked P are true for all valuations where ϕ evaluates to T." At iteration 0 (before entering the loop), only $\top$ are marked. Clearly, $\top$ must be true for any valuation. At iteration $k+1$, consider a Horn clause $P_1 \wedge P_2 \wedge \cdots \wedge P_n \implies Q$ where $P_1, P_2, \ldots, P_n$ are marked but not Q. For a valuation ν where ϕ evaluates to T, $P_1 \wedge P_2 \wedge \cdots \wedge P_n \implies Q$ must evaluate to T. Since $P_1, P_2, \ldots, P_n$ are true in ν (by IH), Q must be true in ν.

Satisfiability of Horn Formulae

Proof (cont'd).

We now prove `Horn`(ϕ) answers correctly. When `Horn`(ϕ) returns "unsatisfiable," there is a Horn clause $P_1 \land P_2 \land \cdots \land P_n \implies \bot$ where $P_1, P_2, \ldots, P_n$ are all marked. Suppose ν is a valuation where ϕ evaluates to T. Then $P_1, P_2, \ldots, P_n$ must be true in ν. Hence $P_1 \land P_2 \land \cdots \land P_n \implies \bot$ evaluates to F. ϕ cannot evaluate to T under ν. A contradiction.

When `Horn`(ϕ) returns "satisfiable," define a valuation ν where all marked propositional atoms are assigned to T and all unmarked atoms are F. We claim ϕ evaluates to T in ν. Suppose not. There is a Horn clause $P_1 \land P_2 \land \cdots P_n \implies Q$ in ϕ which evaluates to F under ν. That is, $P_1, P_2, \ldots, P_n$ are T but Q is F under ν. By the definition of ν, $P_1, P_2, \ldots, P_n$ are marked by the algorithm. Hence Q must also be marked by the algorithm. Q cannot be F in ν. A contradiction. □