COMP6463 Overview of Logic and Computation: Propositional Logic

Alwen Tiu

The Australian National University

July 29, 2011

Outline

- Normal form transformations: NNF, DNF, CNF.
- Definitional CNF.
- Proving unsatisfiability: the resolution method.
- Termination, soundness and completeness of resolution.

Normal forms

In checking validity or satisfiability, certain forms of formulae are easier to work with than others. Three important normal forms that are commonly used:

- Negation normal form (NNF).
- Disjunctive Normal Form (DNF).
- Conjunctive Normal Form (CNF).

We shall work only with formulae which are implication-free. That is, every implication $A \rightarrow B$ is converted into its equivalent $\neg A \vee B$.
We also assume the input formulae do not contain the nullary operators $\bot$ and $\top$.

Some useful tautologies

Units:		$A \wedge \top \equiv A$	$A \vee \top \equiv \top$
		$A \wedge \bot \equiv \bot$	$A \vee \bot \equiv A$
		$A \vee \neg A \equiv \top$	$A \wedge \neg A \equiv \bot$
Idempotency:		$A \wedge A \equiv A$	$A \vee A \equiv A$
Commutativity:		$A \wedge B \equiv B \wedge A$	$A \vee B \equiv B \vee A$
Associativity:		$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$	
		$A \vee (B \vee C) \equiv (A \vee B) \vee C$	
Distributivity:		$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	
		$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$	
Implication:		$A \to B \equiv \neg A \vee B$	
de Morgan:	$\neg\neg A \equiv A$	$\neg(A \wedge B) \equiv \neg A \vee \neg B$	$\neg(A \vee B) \equiv \neg A \wedge \neg B$

Negation normal form

- A formula is in *negation normal form* if the negation operator is applied only to variables.
- Every formula can be transformed into an equivalent NNF using the following tautologies:

$$\begin{aligned} &\text{DM1:} && \neg(A \wedge B) \equiv \neg A \vee \neg B \\ &\text{DM2:} && \neg(A \vee B) \equiv \neg A \wedge \neg B \\ &\text{DM3:} && \neg\neg A \equiv A \end{aligned}$$

NNF Algorithm

function $\mathrm{NNF}(A)$

case A **of**

A is a literal: **return** A

A is $\neg\neg B$: **return** $\mathrm{NNF}(B)$

A is $B \wedge C$: **return** $\mathrm{NNF}(B) \wedge \mathrm{NNF}(C)$

A is $B \vee C$: **return** $\mathrm{NNF}(B) \vee \mathrm{NNF}(C)$

A is $\neg(B \wedge C)$: **return** $\mathrm{NNF}(\neg B) \vee \mathrm{NNF}(\neg C)$

A is $\neg(B \vee C)$: **return** $\mathrm{NNF}(\neg B) \wedge \mathrm{NNF}(\neg C)$

end case

Note: a *literal* is a variable or a negated variable, e.g., $\neg x$.

NNF: an example

$$
\begin{aligned}
&\mathrm{NNF}(\neg(\neg p \wedge q) \vee (p \wedge \neg(r \wedge q))) \\
&\equiv \mathrm{NNF}(\neg(\neg p \wedge q)) \vee \mathrm{NNF}(p \wedge \neg(r \wedge q)) \\
&\equiv (\mathrm{NNF}(\neg\neg p) \vee \mathrm{NNF}(\neg q)) \vee \mathrm{NNF}(p \wedge \neg(r \wedge q)) \\
&\equiv (\mathrm{NNF}(p) \vee \mathrm{NNF}(\neg q)) \vee \mathrm{NNF}(p \wedge \neg(r \wedge q)) \\
&\equiv (p \vee \mathrm{NNF}(\neg q)) \vee \mathrm{NNF}(p \wedge \neg(r \wedge q)) \\
&\equiv (p \vee \neg q) \vee \mathrm{NNF}(p \wedge \neg(r \wedge q)) \\
&\equiv (p \vee \neg q) \vee (\mathrm{NNF}(p) \wedge \mathrm{NNF}(\neg(r \wedge q))) \\
&\equiv (p \vee \neg q) \vee (p \wedge \mathrm{NNF}(\neg(r \wedge q))) \\
&\equiv (p \vee \neg q) \vee (p \wedge (\mathrm{NNF}(\neg r) \vee \mathrm{NNF}(\neg q))) \\
&\equiv (p \vee \neg q) \vee (p \wedge (\neg r \vee \mathrm{NNF}(\neg q))) \\
&\equiv (p \vee \neg q) \vee (p \wedge (\neg r \vee \neg q))
\end{aligned}
$$

Conjunctive Normal Form

- A formula is in *conjunctive normal form* (CNF) if it is of the form

$$(A_{11} \vee \cdots \vee A_{1k_1}) \wedge \cdots \wedge (A_{m1} \vee \cdots \vee A_{mk_m})$$

  where each A_{ij} is a literal.
- In other words, a CNF formula is a conjunction of disjunctions of literals.
- Every NNF formula can be transformed into an equivalent CNF formula using the distributivity laws:

$$\begin{aligned} \text{Dist1:} \quad & A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) \\ \text{Dist2:} \quad & A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C) \end{aligned}$$

  Hence, every formula can be transformed into an equivalent CNF formula.

CNF Algorithm (1)

Input: a formula in NNF.
Output: a CNF formula equivalent to the input.
function $\mathrm{CNF}(A)$

case A **of**
A is a literal: **return** A
A is $B \wedge C$: **return** $\mathrm{CNF}(B) \wedge \mathrm{CNF}(C)$
A is $B \vee C$: **return** $\mathrm{DistCNF}(\mathrm{CNF}(B), \mathrm{CNF}(C))$
end case

$\mathrm{DistCNF}$ distributes a CNF over another CNF, using the distributive law Dist1.

CNF Algorithm (2)

Input: two CNF formulae A and B.
Output: a CNF of $A \vee B$.
function $\mathrm{DistCNF}(A, B)$
  if A is $C \wedge D$ **then**
    return $\mathrm{DistCNF}(C, B) \wedge \mathrm{DistCNF}(D, B)$
  else if B is $E \wedge F$ **then**
    return $\mathrm{DistCNF}(A, E) \wedge \mathrm{DistCNF}(A, F)$
  else
    return $A \vee B$.
  end if

In the worst case, what is the space complexity of CNF transformation?

CNF: an example

$$
\begin{aligned}
&\mathrm{CNF}((p \vee \neg q) \vee (p \wedge (\neg r \vee q))) \\
&\equiv \mathrm{DistCNF}(\mathrm{CNF}(p \vee \neg q), \mathrm{CNF}(p \wedge (\neg r \vee q))) \\
&\equiv \mathrm{DistCNF}(p \vee \neg q, \mathrm{CNF}(p \wedge (\neg r \vee q))) \\
&\equiv \mathrm{DistCNF}(p \vee \neg q, p \wedge (\neg r \vee q)) \\
&\equiv \mathrm{DistCNF}(p \vee \neg q, p) \wedge \mathrm{DistCNF}(p \vee \neg q, \neg r \vee q) \\
&\equiv (p \vee \neg q \vee p) \wedge \mathrm{DistCNF}(p \vee \neg q, \neg r \vee q) \\
&\equiv (p \vee \neg q \vee p) \wedge (p \vee \neg q \vee \neg r \vee q)
\end{aligned}
$$

Note that several computation steps are omitted in the subcalls $\mathrm{CNF}(p \vee \neg q)$ and $\mathrm{CNF}(p \wedge (\neg r \vee q))$.

CNF and validity checking

- Checking validity of a CNF formula can be done in linear time, in the size of the formula.
- To check that a disjunction of literal $(l_1 \vee \ldots \vee l_k)$ is valid, we only need to check that there are two dual literals.
- Therefore, to check that a CNF is valid, check that each disjunction of literals in the CNF is valid.
- So one way of checking validity of a formula is to first transform it into CNF and then use this procedure to check its validity.
- Note that satisfiability checking for CNF formulae is NP-complete.

Disjunctive Normal Form

- A formula is in *disjunctive normal form* (DNF) if it is of the form

$$(A_{11} \wedge \cdots \wedge A_{1k_1}) \vee \cdots \vee (A_{m1} \wedge \cdots \wedge A_{mk_m})$$

  for some m and n. As in CNF, each A_{ij} is a literal.
- Every NNF formula can be transformed into an equivalent DNF formula using the distributivity laws Dist1 and Dist2.
- Exercise: give an algorithm for DNF transformation.

Definitional CNF

- The CNF transformation, in the worse case, can result in a normal form that is exponentially larger than the input formula.
- To avoid this blow up, we can introduce *new variables* in the transformation.
- This idea, originally due to Tseitin (1968), results in CNF which is linear in size of the original input.
- But, the original formula and the resulting CNF are not equivalent. Instead, we have a weaker property, i.e., they are equisatisfiable.
- A formula F is *equisatisfiable* to G if satisfiability of F implies satisfiability of G and vice versa.

Tseitin encoding

- The basic idea is to introduce new propositional variables as abbreviations or 'definitions' for subformulae.
- Assume that the input formula is already in NNF.
- Tseiting encoding works by repeatedly replacing subformulae of the form $l_1 * l_2$, where $*$ is either $\wedge$ or $\vee$, and l_1 and l_2 are literals, with a new variable, say x, and add a definition

$$x \equiv (l_1 * l_2)$$

  in conjunction to the substituted formula.
- When the original formula is reduced to a variable, we then apply the usual CNF transformation to the definitions.

Tseitin encoding: an example

Consider transforming the formula $(p \vee (q \wedge \neg r)) \wedge s$.

$$\begin{aligned}
&\boxed{(p \vee (q \wedge \neg r)) \wedge s} && \rightsquigarrow \\
&(p_1 \equiv (q \wedge \neg r)) \wedge \boxed{(p \vee p_1) \wedge s} && \rightsquigarrow \\
&(p_1 \equiv (q \wedge \neg r)) \wedge (p_2 \equiv (p \vee p_1)) \wedge \boxed{(p_2 \wedge s)} && \rightsquigarrow \\
&(p_1 \equiv (q \wedge \neg r)) \wedge (p_2 \equiv (p \vee p_1)) \wedge (p_3 \equiv (p_2 \wedge s)) \wedge p_3.
\end{aligned}$$

Then apply the usual CNF transformation:

$$\begin{aligned}
&(\neg p_1 \vee q) \wedge (\neg p_1 \vee \neg r) \wedge (p_1 \vee \neg q \vee r) \wedge \\
&(\neg p_2 \vee p \vee p_1) \wedge (p_2 \vee \neg p) \wedge (p_2 \vee \neg p_1) \wedge \\
&(\neg p_3 \vee p_2) \wedge (\neg p_3 \vee s) \wedge (p_3 \vee \neg p_2 \vee \neg s) \wedge \\
&p_3.
\end{aligned}$$

Tseitin encoding: equisatisfiability

To prove equisatisfiability of the Tseitin encoding, first define a notion of a *context*.

A context is a formula with a 'hole' [ ]. We denote a context by $F[\,]$. It is defined inductively as follows:

- [ ] is a context.
- If $F[\,]$ is a context then $F[\,] \vee G$, $G \vee F[\,]$, $F[\,] \wedge G$ and $G \wedge F[\,]$ are also contexts.

If $F[\,]$ is a context and G is a formula, then $F[G]$ is the formula obtained by replacing the hole in $F[\,]$ with G.

Theorem

Let $F[\,]$ be a context and let G be a formula. Suppose x is a variable not appearing in either $F[\,]$ or G. Then $F[G]$ and $(x \equiv G) \wedge F[x]$ are equisatisfiable.

A representation of CNF using sets

- To simplify presentation, we use another notation to represent CNF as sets of *clauses*.
- A clause is a disjunction of literals

$$(l_1 \vee \cdots \vee l_n)$$

  where $n \geq 0$.
- Notational convention: in the case where $n = 0$, the clause is said to be the *empty clause*, and is written as $\bot$.
- Since $\vee$ satisfies commutativity and idempotency, we also assume that:
  - the order of the literals in the clause is immaterial, e.g., we consider both $(x_1 \vee \neg y_1 \vee z)$ and $(z \vee x_1 \vee \neg y_1)$ to be the same clause;
  - there are no duplications of literals in a clause, e.g., $(z \vee z \vee y)$ is represented simply as $(z \vee y)$.

Example

- The CNF

$$(a \vee b \vee c \vee c) \wedge (\neg a \vee \neg b) \wedge \neg c \wedge (\neg b \vee \neg a)$$

will be represented as the following set:

$$\{(a \vee b \vee c), (\neg a \vee \neg b), (\neg c)\}.$$

- We call this representation the clausal form of the original CNF formula.
- Notice that a duplicate literal c in the clause $(a \vee b \vee c \vee c)$ is removed in the clausal form.
- Notice also that, since $(\neg a \vee \neg b)$ and $(\neg b \vee \neg a)$ represent the same clause, only one of them is included in the clausal form.

The resolution proof method

- The resolution proof method is really a *refutation procedure*.
- Given a formula in CNF, it checks whether the formula is unsatisfiable.
- Since validity is dual to unsatisfiability, we can use resolution to check for validity: F is valid if $\neg F$ is unsatisfiable.

The inference rule for resolution

- The basic mechanism of the resolution proof method is an *inference rule* for transforming a CNF (in clausal form) into another CNF, preserving satisfiability.
- The only inference rule is the *resolution rule*: if the input CNF has two clauses of the form:

$$(l_1 \vee \cdots \vee l_m \vee x) \text{ and } (k_1 \vee \cdots \vee k_n \vee \neg x)$$

then add the following clause to the CNF:

$$(l_1 \vee \cdots \vee l_m \vee k_1 \cdots \vee k_n)$$

We call the latter the resolvent of the former two clauses.

- Note: duplicate literals need to be removed in the resolvent.
- The original CNF is unsatisfiable if we are able to derive the empty clause (i.e., 'false') via the resolution rule.

Example: an instance of resolution

Suppose the input CNF is

$$\{(a \vee b \vee \neg c), (c \vee a \vee d), (d \vee e)\}$$

then we can apply the resolution rule to the clauses $(a \vee b \vee \neg c)$ and $(c \vee a \vee d)$, which gives us a new clause $(a \vee b \vee d)$.
The resulting CNF is therefore

$$\{(a \vee b \vee \neg c), (a \vee c \vee d), (d \vee e), (a \vee b \vee d)\}$$

Example: an unsatisfiable CNF

Suppose the input CNF is

$$\{(a \vee b), (a \vee \neg b), (\neg a)\}.$$

Resolving $(a \vee b)$ and $(a \vee \neg b)$ gives us (a), and resolving this further with $(\neg a)$ gives us $\bot$ (the empty clause).
Therefore the input CNF is unsatisfiable.

Example: a wrong instance of resolution

- One can only resolve one pair of literals at a time. Resolving more than one pair results in an unsound proof method.
- For example, suppose the input CNF is:

$$\{(a \vee b), (\neg a \vee \neg b)\}$$

  Note that this input CNF is satisfiable, e.g., by mapping a to 1 and b to 0.
- In resolving $(a \vee b)$ and $(\neg a \vee \neg b)$, we have a choice of removing a and $\neg a$, or b and $\neg b$, but not both pairs simultaneously.
- If we do remove both pairs, then we end up with the empty clause $\perp$, and this would imply that the original CNF is unsatisfiable, contradicting the fact that it is satisfiable.

Saturated sets of clauses

- Given two clauses C_1 and C_2, one containing a literal l and the other containing $\neg l$, we write $res(C_1, C_2, l)$ to denote the resolvent of C_1 and C_2 with l as the literal to be resolved.
- A set of clauses Δ is saturated if for all $C_1 \in \Delta$ and $C_2 \in \Delta$ and literal l, if $res(C_1, C_2, l)$ is defined, then

$$res(C_1, C_2, l) \in \Delta.$$

- In other words, Δ is closed under the resolution rule; applying resolution to any two clauses in Δ results in another clause already in Δ.

Unsatisfiability testing with resolutions

Input: a set of clauses Δ
Output: return true if Δ is unsatisfiable, otherwise return false.
function unsat(Δ):

if $\bot \in \Delta$ **then**
return true
else if Δ is saturated **then**
return false
else
select C_1 and C_2 and l such $res(C_1, C_2, l) \notin \Delta$
return unsat($\Delta \cup \{res(C_1, C_2)\}$)
end if

Example: an unsatisfiable set of clauses

A simple way to represent resolution proofs is by listing the clauses from the input CNF, and extend the list with derived clauses one at a time until one derives $\bot$.
Example: consider this CNF:

$$\{(\neg x \vee y), (\neg y \vee \neg z), (x \vee \neg z), z\}.$$

One of its resolution proofs can be represented as follows:

(1)	$\neg x \vee y$	
(2)	$\neg y \vee \neg z$	
(3)	$x \vee \neg z$	
(4)	z	
(5)	$\neg x \vee \neg z$	resolution of (1) and (2)
(6)	$\neg z$	resolution of (3) and (5)
(7)	$\bot$	resolution of (4) and (6)

Example: clause re-use

A clause can be used in more than one resolution step.

(1)	$\neg b \vee \neg d$	
(2)	$c \vee d$	
(3)	$\neg a \vee \neg d$	
(4)	$a \vee b$	
(5)	$\neg a \vee \neg c$	
(6)	$\neg b \vee \neg c$	
(7)	$\neg b \vee c$	res. of (1) and (2)
(8)	$c \vee \neg a$	res. of (2) and (3)
(9)	$b \vee \neg c$	res. of (4) and (5)
(10)	$c \vee b$	res. of (8) and (4)
(11)	$\neg c$	res. of (9) and (6)
(12)	c	res. of (7) and (2)
(13)	$\bot$	res. of (11) and (12)

Example: proving tautology using resolution

Consider the formula:

$$(a \wedge (b \vee c)) \rightarrow ((a \wedge b) \vee (a \wedge c)).$$

To prove its validity using resolution, we show that its negation is unsatisfiable.
Its negation, in CNF clausal form, is

$$\{a, b \vee c, \neg a \vee \neg b, \neg a \vee \neg c\}$$

(1)	a	
(2)	$b \vee c$	
(3)	$\neg a \vee \neg b$	
(4)	$\neg a \vee \neg c$	
(5)	$\neg b$	res. of (1) and (3)
(6)	$\neg c$	res. of (1) and (4)
(7)	c	res. of (2) and (5)
(8)	$\bot$	res. of (6) and (7)

Termination, soundness, and completeness of resolution

Theorem

Given an input Δ in clausal form, the procedure unsat(Δ) terminates.

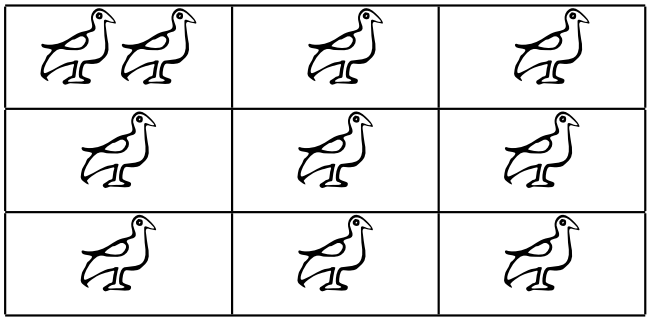
Theorem

The resolution method is sound. That is, given a clausal form Δ, if unsat(Δ) returns true, then Δ is unsatisfiable.

Theorem

The resolution method is complete. That is, if a clausal form Δ is unsatisfiable, then unsat(Δ) returns true.

Example: The Pigeon hole principle



If you put n pigeons into $n-1$ holes (for $n \geq 2$), then there is a hole containing more than one pigeon.

The Pigeon hole principle in propositional logic

- It's important in satisfiability solving (and logic in general) because it is an example of a 'hard' problem.
- There are **no** resolution proofs of the pigeon hole principle of size polynomial in n, where n is the number of pigeons.
  The shortest resolution proof is of length exponential in n.
- It is used as a benchmark to compare relative "efficiency" of various propositional proof system.
- There *are* propositional proof systems that yield shortest proofs of polynomial size, e.g., *extended resolution*.

Encoding the Pigeon hole principle

For each $n \geq 2$, the pigeon hole principle can be formulated as a set of clauses as follows:

- Introduce a variable $p_{i,j}$ for each $1 \leq i \leq n$, $1 \leq j \leq n-1$.
  The intuition is that $p_{i,j}$ says that (pigeon) i gets mapped to (hole) j.
- Add the following set of clauses:

$$\begin{aligned} S_n = \quad & \{p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n-1} \mid 1 \leq i \leq n\} \\ & \cup \{\neg p_{i,k} \vee \neg p_{j,k} \mid 1 \leq i < j \leq n, 1 \leq k \leq n-1\}. \end{aligned}$$

- S_n asserts that there is a one-to-one mapping from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n-1\}$. Hence, S_n must be unsatisfiable by the pigeon hole principle.

Example

Here's an example for $n = 3$ (in clausal form):

$$
\begin{aligned}
&\{\ p_{1,1} \vee p_{1,2},\ \ p_{2,1} \vee p_{2,2},\ \ p_{3,1} \vee p_{3,2},\\
&\neg p_{1,1} \vee \neg p_{2,1},\ \ \neg p_{1,2} \vee \neg p_{2,2},\\
&\neg p_{1,1} \vee \neg p_{3,1},\ \ \neg p_{1,2} \vee \neg p_{3,2},\\
&\neg p_{2,1} \vee \neg p_{3,1},\ \ \neg p_{2,2} \vee \neg p_{3,2}\\
&\}
\end{aligned}
$$

Try proving it using resolution. Try larger n and watch the exponential blow up!