CDM

Propositional Logic

Klaus Sutner

Carnegie Mellon University

10-prop-logic 2017/12/15 23:21

1. Propositional Statements

- Semantics
- Normal Forms

- We would like our logic to be powerful so we can express complicated properties.
- We would like to have a general decision algorithm that checks whether a formula is valid.

These desiderata clash badly. For example, thanks to Gödel, we cannot hope to deal with general arithmetic over $\mathbb{N}$.

So let's start with a very weak logic (which is insufficient in and of itself, but is an important building block for fancier systems): propositional logic.

$$p \qquad \neg\phi \qquad \phi \vee \psi \qquad \phi \wedge \psi \qquad \phi \Rightarrow \psi$$

More precisely, we will focus on the part having to do with logical connectives such as "and", "or", "implies" and "not".

We will discuss a simple framework that allows us to represent these connectives in a very formal manner.

The next step is to build a deduction system that captures our intuitive way of reasoning. For example, everyone would agree that if we have somehow established an assertion "φ and ψ" then we have in particular also established ψ.

Or if we manage to establish both "ϕ" and "ϕ implies ψ" then it is safe to conclude that we also know "ψ".

On the other hand, if we only know "ϕ" so far it would be an error to conclude "ϕ and ψ".

And so on and so forth.

Isn't this all blindingly obvious?

Propositional reasoning on small, natural examples is indeed more or less obvious – simply because we are trying to capture precisely the fundamental laws of thought that everyone understands and agrees upon.

So, for simple cases of propositional reasoning the human mind is perfectly adequate, but when we have to deal with huge assertions involving thousands of connectives efficient algorithms are necessary. Many practical problems such as checking a train schedule for correctness naturally produces huge propositional expressions.

Another important issue is that we are trying to design algorithms that are capable of propositional reasoning. To this end we need rather fastidious definitions and a very careful description of the deduction system, no appeals to intuition are admissible.

Lastly, when we further extend our system to full predicate logic things become much, much more complicated – you can think of propositional logic as an excellent (and directly useful) warm-up exercise.

We start with atomic assertions, that are either true or false, but cannot be broken apart (at least not in the current framework).

- "stack S is empty"
- "18 is a prime number"
- "the GCD of 100 and 35 is 5"
- "polynomial p has no integer roots"
- "there are more reals than rationals"
- "the algorithm terminates on all inputs"

Then we arrange these atomic assertions into more complicated, compound ones using logical connectives.

Complicated assertions can be built up from atomic assertions by logical connectives such as "not", "and", "or", and "implies".

Example

- "p and q implies p"
- "p implies q, implies q"
- "p implies q, implies not p implies not q"
- "p or not p"

We want to understand how these connectives work.

In particular, we want to be able to determine the truth (or falsehood) of a compound statement from its atomic pieces and the connectives.

We fix a language for propositional formulae, much like a programming language.

$\bot$, $\top$	constants false, true
p, q, r, $\ldots$	propositional variables
$\neg$	not
$\wedge$	and, conjunction
$\vee$	or, disjunction
$\Rightarrow$	conditional (implies)
$\Leftrightarrow$	biconditional (equivalent)

Negation is unary, all the others a binary.

We assume a countably infinite supply of variables, sometimes indexed as in $p_0, p_1, \ldots$

Definition

The set of propositional formulae is defined by:

- The constants $\bot$ and $\top$ are formulae.
- Every variable is a formula.
- If φ and ψ are formulae, so are
  - $\neg\varphi$
  - $\varphi \wedge \psi$
  - $\varphi \vee \psi$
  - $\varphi \Rightarrow \psi$
  - $\varphi \Leftrightarrow \psi$

Since we use infix notation, we also need parentheses, otherwise an expression like $p \wedge q \vee r$ is not uniquely parsable. Our rules should loke like $(\varphi \wedge \psi)$ and so on.

To avoid impenetrable forests of parentheses, one uses precedence:

$$\neg > \wedge > \vee > \Rightarrow > \Leftrightarrow$$

and declare $\Rightarrow$ to be right-associative, so we can write for example

$$(p \wedge q \Rightarrow r) \Rightarrow \neg p \vee q \Rightarrow p \Rightarrow r$$

Alternatively, we could use prefix or postfix notation.

Exercise

Write a context-free grammar and corresponding compiler for these various options.

$$\bot \Rightarrow p$$

$$(p \Rightarrow \bot) \Rightarrow \neg p$$

$$p \Rightarrow q \Rightarrow p$$

$$\neg(p \wedge q) \Rightarrow \neg p \vee \neg q$$

$$p \wedge (p \Rightarrow q) \Rightarrow q$$

$$(p \wedge q \Rightarrow r) \Rightarrow \neg p \vee q \Rightarrow p \Rightarrow r$$

Exercise

For each of these, rewrite them in fully parenthesized form. Then determine if they are intuitively valid.

- Propositional Statements

2. Semantics

- Normal Forms

We have used familiar notation to express propositional formulae, so everybody knows that $A \wedge B$ is supposed to express a conjunction.

However, strictly speaking all we have so far is syntax, a grammar for a class of expressions – actually, we didn't even bother to write down a precise grammar, but it would not be difficult to do so, and we are not interested in parsers at this point.

However, we do need to be more careful in pinning down the semantics, the meaning of our formulae.

For propositional formulae this is not very difficult, but it's a good warm-up exercise for more complicated systems later (e.g., the semantics of an expression in predicate logic is quite a bit more difficult to describe).

x	y	$H_{\mathsf{and}}(x,y)$	$H_{\mathsf{or}}(x,y)$	$H_{\mathsf{imp}}(x,y)$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	1

x	y	$H_{\mathsf{equ}}(x,y)$	$H_{\mathsf{xor}}(x,y)$	$H_{\mathsf{nand}}(x,y)$
0	0	1	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

Definition

A truth assignment is a map that associates a truth value with each variable.

Given a formula φ and a truth assignment σ for all the variables, we can determine the truth value of φ under α, written $[\![\varphi]\!]_\sigma$.

$$
\begin{aligned}
[\![\bot]\!]_\sigma &= 0 \\
[\![\top]\!]_\sigma &= 1 \\
[\![\neg\varphi]\!]_\sigma &= H_{\mathsf{not}}([\![\varphi]\!]_\sigma) \\
[\![\varphi \vee \psi]\!]_\sigma &= H_{\mathsf{or}}([\![\varphi]\!]_\sigma, [\![\psi]\!]_\sigma) \\
[\![\varphi \wedge \psi]\!]_\sigma &= H_{\mathsf{and}}([\![\varphi]\!]_\sigma, [\![\psi]\!]_\sigma) \\
[\![\varphi \Rightarrow \psi]\!]_\sigma &= H_{\mathsf{imp}}([\![\varphi]\!]_\sigma, [\![\psi]\!]_\sigma) \\
[\![\varphi \Leftrightarrow \psi]\!]_\sigma &= H_{\mathsf{equ}}([\![\varphi]\!]_\sigma, [\![\psi]\!]_\sigma)
\end{aligned}
$$

Of course, $[\![p]\!]_\sigma = \sigma(p)$ is directly given for propositional variables p.

Definition

Let σ be a truth assignment and φ a propositional formula. σ satisfies or models φ if φ is true under σ: $[\![\varphi]\!]_\sigma = 1$.

A truth assignment satisfies a set of formulae Φ if it satisfies each formula in the set.

In symbols

$$\sigma \models \varphi \qquad \text{and} \qquad \sigma \models \Phi$$

Incidentally, this notation is lifted from Frege's Begriffsschrift.

Strictly speaking, a truth assignment need only be defined on the variables appearing in the formula and could be defined as a partial map from the collection of all variables but we will not quibble.

Definition

Let Φ be a set of formulae and φ a propositional formula. φ is a semantic consequence of Φ if for every truth assignment σ that satisfies Φ also satisfies φ: $\sigma \models \Phi$ implies $\sigma \models \varphi$

In symbols,

$$\Phi \models \varphi$$

Example

The famous modus ponens, a classical rule of inference.

$$\varphi, \varphi \Rightarrow \psi \models \psi.$$

Example

From $\Phi, \varphi \models \bot$ it follows that $\Phi \models \neg\varphi$.

Example

From $\Phi, \varphi \models \psi$ it follows that $\Phi \models \varphi \Rightarrow \psi$.

Definition

A formula φ is a tautology if it is true under all truth assignments. In symbols,

$$\models \varphi$$

A formula φ is a contradiction if no assignment satisfies φ.

A formula φ is a contingency (or satisfiable) if there is some assignment that satisfiable φ.

Tautologies are what corresponds to the informal notion of a "true" statement or a "valid" statement (at least in propositional logic).

Example

$p \wedge \neg p$ is a contradiction, $p \vee \neg p$ is a tautology.

Exercise

Show that all the implications from above are tautologies.

Naturally, we can associate decision problems with all these definitions. Finding good algorithms to solve these problems computationally will turn out to be quite a challenge.

Problem:	**Tautology**
Instance:	A propositional formula φ.
Question:	Is φ a tautology?

Problem:	**Satisfiability**
Instance:	A propositional formula φ.
Question:	Is φ a contingency?

Problem:	**Satisfiability (search)**
Instance:	A propositional formula φ.
Solution:	A satisfying truth assignment.

Likewise we could introduce a decision problem **Contradiction**.

Note that these problems are all closely related, though:

- φ is a tautology iff $\neg\varphi$ is not satisfiable.
- φ is a contradiction iff φ is not satisfiable.

In practice, Satisfiability testing algorithms are particularly important.

Hence fast algorithms for one problem could be used to solve the others (alas, there is a glitch: some algorithms require input to be in a special form).

Here is a brute-force attack: we construct a truth table, an idea apparently due to Lewis Carroll. The table is a brute-force list of the truth values for all possible assignments. For example the 3-variable formula

$$\varphi = (((p \wedge q) \Rightarrow r) \wedge (p \Rightarrow q)) \Rightarrow (p \Rightarrow r)$$

produces the table

p	q	r	$(p \wedge q) \Rightarrow r$	$p \Rightarrow q$	$p \Rightarrow r$	φ
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	1
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	1	1	1	1	1

So, φ is a tautology. This is easy to implement in principle: just one big loop ranging over all truth assignments, plus a little evaluation routine.

Regrettably, this method falls apart if we have many variables: the table has 2^n rows for n variables. In applications, n may be several thousand.

Let's suppose we have 500 variables. Let's further assume every stable particle in the universe (about 10^{80}) is a computer that can check 10^9 assignments per second. Lastly assume that the universe is 15 billion years old and that our "universal computer" has been running non-stop ever since the Big Bang. There are

$$3.27339 \cdot 10^{150}$$

truth assignments to check, but so far we would have handled only

$$4.7304 \cdot 10^{106}$$

of these assignments, a minuscule fraction.

Perhaps one could somehow simplify formulae so that it's easy to check whether they are tautologies or contingencies?

Definition

Two formulae φ and ψ are (semantically) equivalent if for any truth assignment σ: $[\![\varphi]\!]_\sigma = [\![\psi]\!]_\sigma$.

In symbols:

$$\varphi \equiv \psi$$

Claim

φ and ψ are equivalent if, and only if, $\varphi \Leftrightarrow \psi$ is a tautology.

Proof. To see this note

$$\begin{aligned} [\![\varphi \Leftrightarrow \psi]\!]_\sigma &= H_{\mathsf{equ}}([\![\varphi]\!]_\sigma, [\![\psi]\!]_\sigma) \\ &= H_{\mathsf{equ}}([\![\varphi]\!]_\sigma, [\![\varphi]\!]_\sigma) = 1 \end{aligned}$$

□

$$\varphi \wedge \bot \equiv \bot$$
$$\varphi \wedge \top \equiv \varphi$$
$$\varphi \wedge \varphi \equiv \varphi$$
$$\varphi \wedge \psi \equiv \psi \wedge \varphi$$
$$\varphi \wedge (\psi \wedge r) \equiv (\varphi \wedge \psi) \wedge r$$
$$\varphi \wedge (\psi \vee r) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge r)$$
$$\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$$
$$\varphi \Rightarrow \bot \equiv \neg\varphi$$

Exercise

Establish all these equivalences.

In the formula $\varphi \Rightarrow \psi$

- φ is the antecedent, hypothesis or premiss
- ψ is the consequent or conclusion

No one has any problems understanding the meaning of conjunction, disjunction and negation, but some feel uneasy about implications: if the hypothesis is false, the whole implication is true regardless of the conclusion.

So $\bot \Rightarrow \varphi$ is a tautology, regard less of φ.

Some people find this counterintuitive.

Claim

$\varphi \Rightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$, and to $\neg(\varphi \wedge \neg\psi)$.

Any implication can be associated with 3 natural variants.

Implication	$\varphi \Rightarrow \psi$
Converse	$\psi \Rightarrow \varphi$
Inverse	$\neg\varphi \Rightarrow \neg\psi$
Contrapositive	$\neg\psi \Rightarrow \neg\varphi$

Claim

An implication and its contrapositive are equivalent.

However, in general an implication is not equivalent to its converse, nor it inverse.

It is immediate from the definitions that

- φ is a tautology iff $\varphi \equiv \top$
- φ is a contradiction iff $\varphi \equiv \bot$
- φ is satisfiable iff not $\varphi \equiv \bot$

Hence testing whether two formulae are equivalent is at least as hard as our decision problems.

It's no worse, though: recall that two formulae are equivalent iff $\varphi \iff \psi$ is a tautology.

Thus Equivalence Testing and Tautology are of the same complexity.

These equivalences can be used to simplify complicated formulae.

Note that this is very similar to our analysis of digital circuits in terms of Boolean algebra.

- Instead of circuits we have formulae,
- instead of inputs we have truth assignments,
- instead of equations we have equivalences.

But other than that, we have the same laws.

Needless to say, this cannot be coincidence. Before we explore this idea, let's pin down a formal system for propositional logic.

- Propositional Statements
- Semantics

3. Normal Forms

There are two important and drastically different ways to represent Boolean formulae in connection with our decision algorithms.

- Special syntactic forms, in particular NNF, CNF, DNF.
- A custom designed data structure, a so-called binary decision diagram.

As we will see, the representation is critical for complexity issues.

Let's start with normal forms.

It seems reasonable to preprocess the input a bit: we can try to bring the input into some particularly useful syntactic form, a form that can then be exploited by the algorithm.

There are two issues:

- The transformation must produce an equivalent formula (but see below about possible additions to the set of variables).
- The transformation should be fast.

Here are some standard methods to transform propositional formulae.

We will focus on formulae using $\neg, \wedge, \vee$, a harmless assumptions since we can easily eliminate all implications and biconditionals.

Definition

A formula is in Negation Normal Form (NNF) if it only contains negations applied directly to variables.

A literal is a variable or a negated variable.

Theorem

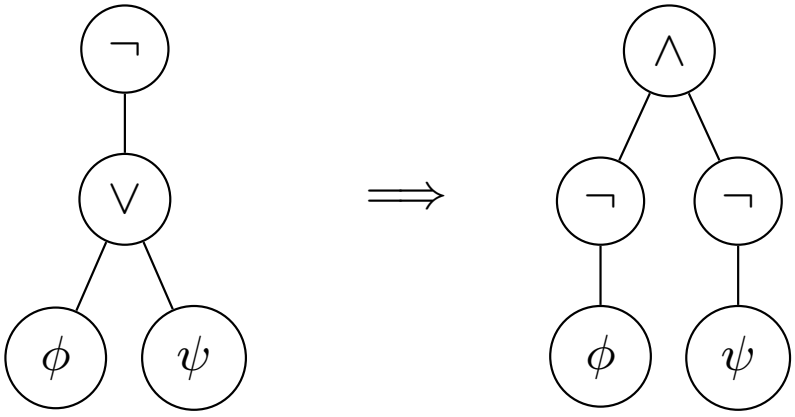
For every formula, there is an equivalent formula in NNF.

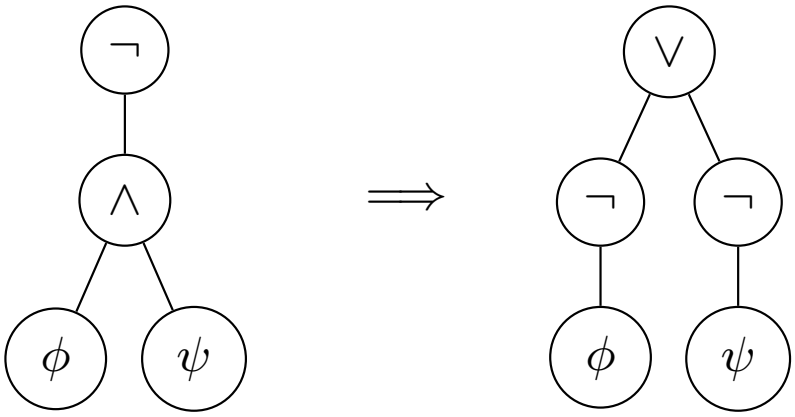
The algorithm pushes all negation signs inward, until they occur only next to a variable.

Rules for this transformation:

$$\begin{aligned} \neg(\varphi \wedge \psi) &\Rightarrow \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) &\Rightarrow \neg\varphi \wedge \neg\psi \\ \neg\neg\varphi &\Rightarrow \varphi \end{aligned}$$

Push negations down to the leaves.





This is an example of a rewrite system: replace the LHS by the RHS of a substitution rule, over and over, until a fixed point is reached.

Note that this requires pattern matching: we have to find the handles in the given expression.

$$\neg(p \wedge (q \vee \neg r) \wedge s \wedge \neg t)$$
$$\neg p \vee \neg(q \vee \neg r) \vee \neg s \vee t$$
$$\neg p \vee (\neg q \wedge r) \vee \neg s \vee t$$

Also note that the answer is not unique, here are some other NNFs for the same formula:

$$\neg p \vee \neg s \vee t \vee (\neg q \wedge r)$$
$$(\neg p \vee \neg s \vee t \vee \neg q) \wedge (\neg p \vee \neg s \vee t \vee r)$$

Conversion to NNF is a search problem, not a function problem.

So we are left with a formula built from literals using connectives $\wedge$ and $\vee$. The most elementary such formulae have special names.

Definition

A minterm is a conjunction of literals.
A maxterm is a disjunction of literals.

Note that a truth table is essentially a listing of all possible 2^n full minterms over some fixed variables $x_1, x_2, \ldots, x_n$ combined with the corresponding truth values of the formula $\varphi(x_1, \ldots, x_n)$.

By forming a disjunction of the minterms for which the formula is true we get a (rather clumsy) normal form representation of the formula.

The reason we are referring to the normal form as clumsy is that it contains many redundancies in general. Still, the idea is very important and warrants a definition.

Definition

A formula is in Disjunctive Normal Form (DNF) if it is a disjunction of minterms (conjunctions of literals).

In other words, a DNF formula is a "sum of products" and looks like so:

$$(x_{11} \wedge x_{12} \wedge \ldots \wedge x_{1n_1}) \vee (x_{21} \wedge \ldots \wedge x_{2n_2}) \vee \ldots \vee (x_{m1} \wedge \ldots \wedge x_{mn_m})$$

where each x_{ij} is a literal.

In short: $\bigvee_i \bigwedge_j x_{ij}$.

If you think of the formula as a circuit DNF means that there are two layers: an OR gate on top, AND gates below. Note that this only works if we assume unbounded fan-in and disregard negation.

Theorem

For every formula, there is an equivalent formula in DNF.

Step 1: First bring the formula into NNF.

Step 2: Then use the rewrite rules

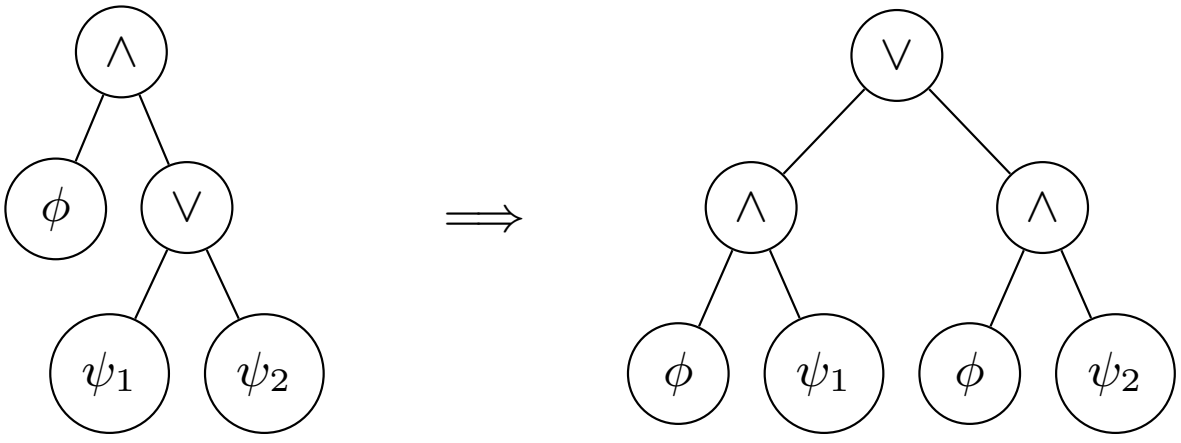
$$\begin{aligned} \varphi \wedge (\psi_1 \vee \psi_2) &\Rightarrow (\varphi \wedge \psi_1) \vee (\varphi \wedge \psi_2) \\ (\psi_1 \vee \psi_2) \wedge \varphi &\Rightarrow (\psi_1 \wedge \varphi) \vee (\psi_2 \wedge \varphi) \end{aligned}$$

□

Exercise

Prove that these rules really produce a DNF. What is the complexity of this algorithm?

Here is the corresponding operation in terms of the parse tree.



Note: we have created a second copy of ϕ in the new tree.

First conversion to NNF.

$$\neg(p \vee (q \wedge \neg r) \vee s \vee \neg t$$
$$\neg p \wedge \neg(q \wedge \neg r) \wedge \neg s \wedge t$$
$$\neg p \wedge (\neg q \vee r) \wedge \neg s \wedge t$$

Reorder, and then distribute $\wedge$ over $\vee$.

$$\neg p \wedge \neg s \wedge t \wedge (\neg q \vee r)$$
$$(\neg p \wedge \neg s \wedge t \wedge \neg q) \vee (\neg p \wedge \neg s \wedge t \wedge r)$$

Computationally, there is one crucial difference between conversion to NNF and conversion to DNF:

- The size of the formula in NNF is linear in the size of the input.
- The size of the formula in DNF is not polynomially bounded by the size of the input.

Thus, even if we have a perfect linear time implementation of the rewrite process, we still wind up with an exponential algorithm.

Exercise

Construct an example where conversion to DNF causes exponential blow-up.

One reason DNF is natural is that one can easily read off a canonical DNF for a formula if we have a truth table for it.

For n variables, the first n columns determine 2^n full minterms (containing each variable either straight or negated).

$$10011 \qquad \Rightarrow \qquad x_1\overline{x}_2\overline{x}_3x_4x_5$$

Select those rows where the formula is true, and collect all the corresponding minterms into a big disjunction.

Done!

Note the resulting formula has $O(2^n)$ conjunctions of n literals each.

Unfortunately, this approach may not produce optimal results: for $p \vee q$ we get

p	q	$(p \vee q)$
0	0	0
0	1	1
1	0	1
1	1	1

So brute force application of this method yields 3 full minterms:

$$(p \wedge \neg q) \vee (\neg p \wedge q) \vee (p \wedge q)$$

Clearly, we need some simplification process. More about this later in the discussion of resolution.

This is essentially the same method we used to get the expressions for sum and carry in the 2-bit adder.

In a Boolean algebra, one talks about sums of products of literals instead of DNF.

Hence, any Boolean expression can be written in sum-of-products form.

Is there also a product-of-sums representation?

Sure . . .

Unlike with ordinary arithmetic, in Boolean algebra there is complete symmetry between meet and join.

Definition

A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of maxterms (disjunctions of literals).

The maxterms are often referred to as clauses in this context. So, a formula in CNF looks like

$$\bigwedge_i \bigvee_j x_{ij}.$$

Theorem

For every formula, there is an equivalent formula in CNF.

Again start with NNF, but now use the rules

$$\begin{aligned} \varphi \vee (\psi_1 \wedge \psi_2) &\Rightarrow (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2) \\ (\psi_1 \wedge \psi_2) \vee \varphi &\Rightarrow (\psi_1 \vee \varphi) \wedge (\psi_2 \vee \varphi) \end{aligned}$$

The formula

$$\varphi = (p_{10} \wedge p_{11}) \vee (p_{20} \wedge p_{21}) \vee \ldots \wedge (p_{n0} \wedge p_{n1})$$

is in DNF, but conversion to CNF using our rewrite rule produces the exponentially larger formula

$$\varphi \equiv \bigwedge_{f:[n]\to\mathbf{2}} \bigvee_{i\in[n]} p_{if(i)}$$

Exercise

Show that there is no small CNF for φ: the 2^n disjunctions of length n must all appear.

Are CNF-based (or DNF-based) algorithms then useless in practice?

The key idea here is to give up on transformations that strictly preserve logical equivalence. Instead focus on equisatisfiability: for two sets of formulae Γ and Γ': Γ' is satisfiable iff Γ is satisfiable.

This is a better framework since it allows for more transformations. It is also used heavily in the whole Gilmore/Davis/Putnam approach below.

In our case, the new set Γ' may well contain additional variables.

This is a nice example of thinking outside of the box.

No, but we have to be careful with the conversion process to control blow-up. Instead of preserving the underlying set of propositional variables, we extend it by a new variable q_ψ for each subformula ψ of ϕ.

For a propositional variable p we let $q_p = p$ and introduce a clause $\{p\}$. Otherwise we introduce clauses as follows:

$$q_{\neg\psi} : \{q_\psi, q_{\neg\psi}\}, \{\neg q_\psi, \neg q_{\neg\psi}\}$$

$$q_{\psi\vee\varphi} : \{\neg q_\psi, q_{\psi\vee\varphi}\}, \{\neg q_\varphi, q_{\psi\vee\varphi}\}, \{\neg q_{\psi\vee\varphi}, q_\varphi, q_\psi\}$$

$$q_{\psi\wedge\varphi} : \{q_\psi, \neg q_{\psi\wedge\varphi}\}, \{q_\varphi, \neg q_{\psi\wedge\varphi}\}, \{\neg q_\psi, \neg q_\varphi, q_{\psi\wedge\varphi}\}$$

The intended meaning of q_ψ is pinned down by these clauses, e.g.

$$q_{\psi\vee\varphi} \equiv q_\psi \vee q_\varphi$$

Consider again the formula

$$\varphi = (p_{10} \wedge p_{11}) \vee (p_{20} \wedge p_{21}) \vee \ldots \vee (p_{n0} \wedge p_{n1})$$

Set $B_k = (p_{k0} \wedge p_{k1})$ and $A_k = B_k \vee B_{k+1} \vee \ldots \vee B_n$ for $k = 1, \ldots, n$. Thus, $\varphi = A_1$ and all the subformulae other than variables are of the form A_k or B_k.

The clauses in the Tseitin form of φ are as follows (we ignore the variables):

- q_{A_k}:
  $\{q_{B_k}, \neg q_{B_k \wedge A_{k-1}}\}, \{q_{A_{k-1}}, \neg q_{B_k \wedge A_{k-1}}\}, \{\neg q_{B_k}, \neg q_{A_{k-1}}, q_{B_k \wedge A_{k-1}}\}$

- q_{B_k}: $\{\neg p_{k0}, q_{B_k}\}, \{\neg p_{k1}, q_{B_k}\}, \{\neg q_{B_k}, p_{k1}, p_{k0}\}$

Exercise

Make sure you understand in the example how any satisfying assignment to φ extends to a satisfying assignment of the Tseitin CNF, and conversely.

Theorem

Let Γ be the set of clauses in Tseitin CNF for formula ϕ. Then Γ and ϕ are equisatisfiable. Moreover, C can be constructed in time linear in the size of ϕ.

Proof.

$\Rightarrow$: Suppose that $\sigma \models \Gamma$.

An easy induction shows that for any subformula ψ we have $[\![\psi]\!]_\sigma = [\![q_\psi]\!]_\sigma$. Hence $[\![\phi]\!]_\sigma = [\![q_\phi]\!]_\sigma = 1$ since $\{q_\phi\}$ is a clause in C.

$\Leftarrow$: Assume that $\sigma \models \phi$.

Define a new valuation τ by $\tau(q_\psi) = [\![\psi]\!]_\sigma$ for all subformulae ψ. It is easy to check that $\tau \models \Gamma$.

□

It should be noted that CNF and DNF are not particularly useful for a human being when it comes to understanding the meaning of a formula (NNF is not quite as bad). But that's not their purpose: they provide a handle for specialized algorithms to test validity and satisfiability. We'll focus on the latter.

First note that one can perform various cleanup operations without affecting satisfiability in CNF.

- We can delete any clause that contains a literal and its negation.
- We can delete any clause that contains another clause (as a subset).

The last step is justified by the equivalence

$$\varphi \wedge (\varphi \vee \psi) \equiv \varphi$$

Here is a very small example. We verify that Peirce's Law

$$((A \to B) \to A) \to A$$

is a tautology. Rewriting the implications we get

$$\neg(\neg(\neg A \vee B) \vee A) \vee A$$

which turns into

$$(\neg A \vee B \vee A) \wedge (\neg A \vee A)$$

By the first simplification rule we are done.

Exercise

Construct a formula Φ_n that is satisfiable if, and only if, the $n \times n$ chessboard has the property that a knight can reach all squares by a sequence of admissible moves.

What would your formula look like in CNF and DNF?

Exercise

Construct a formula Φ_n that is satisfiable if, and only if, the $n \times n$ chessboard admits a knight's tour: a sequence of admissible moves that touches each square exactly once.

Again, what would your formula look like in CNF and DNF?

Exercise

How hard is it to convert a formula to CNF?

Exercise

Show how to convert directly between DNF and CNF.

Exercise

Show: In CNF, if a clause contains x and $\overline{x}$, then we can remove the whole clause and obtain an equivalent formula.

Exercise

Suppose a formula is in CNF.
How hard is it to check if the formula is a tautology?

Exercise

Suppose a formula is in DNF.
How hard is it to check if the formula is a tautology?

Exercise

How about checking whether a formula in DNF (or CNF) is a contradiction?