H250: Honors Colloquium – Introduction to Computation

Conjunctive Normal Form. Tseitin Transform

Marius Minea
marius@cs.umass.edu

Review: Propositional Formulas

Boolean operators: $\neg, \wedge, \vee$.
We'll rewrite $\rightarrow, \leftrightarrow, \oplus$ in these terms.

Interesting Questions:

Are two formulas A and B equivalent? $A \leftrightarrow B$

Is a formula a *tautology* ? (true in *all truth assignments*)

Is a formula *satisfiable*? (in *at least one* truth assignment)

Is a formula a *contradiction*? (has *no* satisfying assignment)

Representation Questions

Can we represent a propositional formula in a "systematic" way?

Can we represent propositional formulas uniquely ?

Ideally, a representation should be

simple and *compact* (easy to implement and store)

easy to process (simple, efficient algorithms)

canonical (a formula is represented in a single way)
two formulas are equal precisely if they have same representation

Conjunctive Normal Form

formula = *conjunction* $\wedge$ of *clauses*
clause = *disjunction* $\vee$ of *literals*
literal = proposition or its negation

$$\begin{aligned} &(a \vee \neg b \vee \neg d)\\ \wedge\ &(\neg a \vee \neg b)\\ \wedge\ &(\neg a \vee c \vee \neg d)\\ \wedge\ &(\neg a \vee b \vee c) \end{aligned}$$

In other words: three levels of operators:
AND $\wedge$: top level – links clauses
OR $\vee$: middle – joins literals within clause
NOT $\neg$: bottom – only applied to propositions

Natural representation because in practice, many formulas arise from multiple constraints that must hold *simultaneously* (AND).

Simple to process, but not *canonical*
(one formula may still be written in many ways)

Useful Transformations (Review)

De Morgan's laws:

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

Distributivity:

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

Distributivity works both ways.
From algebra and circuit logic, we are used to sum of products (OR of ANDs). For CNF, we do the opposite!

Transforming to Conjunctive Normal Form

Think of formula as expression tree. Need to move operator nodes up/down in tree (bring $\wedge$ to top level, $\neg$ to bottom).

1. Push negation inward, until applied to atomic propositions (*de Morgan laws*)

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

2. Push $\vee$ inside $\wedge$ (*distribute* $\vee$ *over* $\wedge$)

$$(A \wedge B) \vee C = (A \vee C) \wedge (B \vee C)$$

Example:

$$\begin{aligned}
& \neg((a \wedge b) \vee ((a \rightarrow (b \wedge c)) \rightarrow c)) \\
= \; & \neg(a \wedge b) \wedge \neg((a \rightarrow (b \wedge c)) \rightarrow c)) \\
= \; & (\neg a \vee \neg b) \wedge ((a \rightarrow (b \wedge c)) \wedge \neg c) \\
= \; & (\neg a \vee \neg b) \wedge (\neg a \vee (b \wedge c)) \wedge \neg c \\
= \; & (\neg a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee c) \wedge \neg c
\end{aligned}$$

Rather straightforward. Could there be problems?

Potential Problem: Size Blowup

Distributivity will duplicate entire subformulas

Can happen repeatedly: $(p_1 \wedge p_2 \wedge p_3) \vee (q_1 \wedge q_2 \wedge q_3) =$
$(p_1 \vee (q_1 \wedge q_2 \wedge q_3)) \wedge (p_2 \vee (q_1 \wedge q_2 \wedge q_3)) \wedge (p_3 \vee (q_1 \wedge q_2 \wedge q_3))$
$= (p_1 \vee q_1) \wedge (p_1 \vee q_2) \wedge (p_1 \vee q_3)$
$\wedge (p_2 \vee q_1) \wedge (p_2 \vee q_2) \wedge (p_2 \vee q_3)$
$\wedge (p_3 \vee q_1) \wedge (p_3 \vee q_2) \wedge (p_3 \vee q_3)$

Worst-case blowup? : *exponential!*

Can't use this transformation for subsequent algorithms (e.g., satisfiability checking) if resulting formula is inefficiently large (possibly too large to represent/process).
Recall our practical requirements for a normal form.

Tseitin Transformation

Idea: rather than duplicate subformula:

introduce *new proposition* to represent it

add constraint: *equivalence* of subformula with new proposition

write this equivalence in CNF

Transformation rules for three basic operators

formula	$p \leftrightarrow$ formula	rewritten in CNF
$\neg A$	$(\neg A \rightarrow p) \wedge (p \rightarrow \neg A)$	$(A \vee p) \wedge (\neg A \vee \neg p)$
$A \wedge B$	$(A \wedge B \rightarrow p) \wedge (p \rightarrow A \wedge B)$	$(\neg A \vee \neg B \vee p) \wedge (A \vee \neg p) \wedge (B \vee \neg p)$
$A \vee B$	$(p \rightarrow A \vee B) \wedge (A \vee B \rightarrow p)$	$(A \vee B \vee \neg p) \wedge (\neg A \vee p) \wedge (\neg B \vee p)$

Tseitin Transformation: Example

Add numbered proposition for each operator:

$(a \overset{1}{\wedge} \neg b) \vee \neg(c \overset{2}{\wedge} d)$

no need to number negations

nor top-level operator $(...) \vee (...)$

New propositions: $p_1 \leftrightarrow a \overset{1}{\wedge} \neg b, \quad p_2 \leftrightarrow c \overset{2}{\wedge} d$.

Rewrite equivalences for new propositions in CNF,
conjunct with top-level operator of formula:

$(p_1 \vee \neg p_2)$	overall formula
$\wedge (\neg a \vee b \vee p_1) \wedge (a \vee \neg p_1) \wedge (\neg b \vee \neg p_1)$	$p_1 \leftrightarrow a \wedge \neg b$
$\wedge (\neg c \vee \neg d \vee p_2) \wedge (c \vee \neg p_2) \wedge (d \vee \neg p_2)$	$p_2 \leftrightarrow c \wedge d$

Can direcly transform multi-argument conjunctions/disjunctions

AND $(\neg A_1 \vee \neg A_2 \vee \neg A_3 \vee p) \wedge (A_1 \vee \neg p) \wedge (A_2 \vee \neg p) \wedge (A_3 \vee \neg p)$

OR $(A_1 \vee A_2 \vee A_3 \vee \neg p) \wedge (\neg A_1 \vee p) \wedge (\neg A_2 \vee p) \wedge (\neg A_3 \vee p)$

Tseitin Transformation: Circuit View

$(a \lor \neg b) \land \neg(c \overset{1}{\lor} d)$

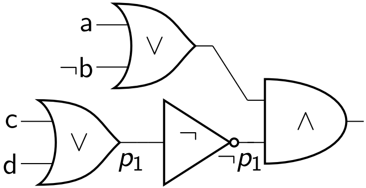
Each gate input: one new proposition

  not needed for $\lor$ as input to $\land$

  not needed for top level $\land$

Example: a single new proposition

$(a \lor \neg b) \land \neg p_1$	our formula
$\land(p_1 \leftrightarrow c \lor d)$	meaning of p_1



Convert each equivalence to CNF (by the above rules)

combine them with $\land$

$(a \lor \neg b) \land p_1$	top level (result)
$\land(c \lor d \lor \neg p_1) \land (\neg c \lor p_1) \land (\neg d \lor p_1)$	

What Do We Get?

A new formula with more propositions than the original one
NOT an equivalent formula

New formula is *satisfiable iff the original is satisfiable*
we call it *equisatisfiable*)

Size of resulting formula: *linear* in original size
good for use in satisfiability checking