



Tseytin transformation

The **Tseytin transformation**, alternatively written **Tseitin transformation**, takes as input an arbitrary combinatorial logic circuit and produces an equisatisfiable boolean formula in conjunctive normal form (CNF). The length of the formula is linear in the size of the circuit. Input vectors that make the circuit output "true" are in 1-to-1 correspondence with assignments that satisfy the formula. This reduces the problem of circuit satisfiability on any circuit (including any formula) to the satisfiability problem on 3-CNF formulas. It was discovered by the Russian scientist Grigori Tseitin.

Motivation

The naive approach is to write the circuit as a Boolean expression, and use De Morgan's law and the distributive property to convert it to CNF. However, this can result in an exponential increase in equation size. The Tseytin transformation outputs a formula whose size grows linearly relative to the input circuit's.

Approach

The output equation is the constant 1 set equal to an expression. This expression is a conjunction of sub-expressions, where the satisfaction of each sub-expression enforces the proper operation of a single gate in the input circuit. The satisfaction of the entire output expression thus enforces that the entire input circuit is operating properly.

For each gate, a new variable representing its output is introduced. A small pre-calculated CNF expression that relates the inputs and outputs is appended (via the "and" operation) to the output expression. Note that inputs to these gates can be either the original literals or the introduced variables representing outputs of sub-gates.

Though the output expression contains more variables than the input, it remains *equisatisfiable*, meaning that it is satisfiable if, and only if, the original input equation is satisfiable. When a satisfying assignment of variables is found, those assignments for the introduced variables can simply be discarded.

A final clause is appended with a single literal: the final gate's output variable. If this literal is complemented, then the satisfaction of this clause enforces the output expression's to false; otherwise the expression is forced true.

Examples

Consider the following formula ϕ .

$$\phi := ((p \vee q) \wedge r) \rightarrow (\neg s)$$

Consider all subformulas (excluding simple variables):

$$\begin{aligned}& \neg s \\& p \vee q \\& (p \vee q) \wedge r \\& ((p \vee q) \wedge r) \rightarrow (\neg s)\end{aligned}$$

Introduce a new variable for each subformula:

$$\begin{aligned}x_1 &\leftrightarrow \neg s \\x_2 &\leftrightarrow p \vee q \\x_3 &\leftrightarrow x_2 \wedge r \\x_4 &\leftrightarrow x_3 \rightarrow x_1\end{aligned}$$

Conjunct all substitutions and the substitution for ϕ :








$$T(\phi) := x_4 \wedge (x_4 \leftrightarrow x_3 \rightarrow x_1) \wedge (x_3 \leftrightarrow x_2 \wedge r) \wedge (x_2 \leftrightarrow p \vee q) \wedge (x_1 \leftrightarrow \neg s)$$

All substitutions can be transformed into CNF, e.g.

$$\begin{aligned}x_2 \leftrightarrow p \vee q &\equiv (x_2 \rightarrow (p \vee q)) \wedge ((p \vee q) \rightarrow x_2) \\&\equiv (\neg x_2 \vee p \vee q) \wedge (\neg(p \vee q) \vee x_2) \\&\equiv (\neg x_2 \vee p \vee q) \wedge ((\neg p \wedge \neg q) \vee x_2) \\&\equiv (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge (\neg q \vee x_2)\end{aligned}$$

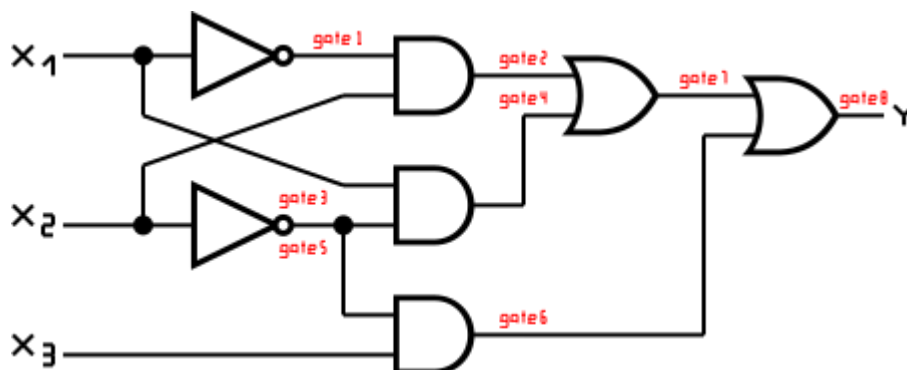
Gate sub-expressions

Listed are some of the possible sub-expressions that can be created for various logic gates. In an operation expression, C acts as an output; in a CNF sub-expression, C acts as a new Boolean variable. For each operation, the CNF sub-expression is true if and only if C adheres to the contract of the Boolean operation for all possible input values.

Type	Operation	CNF sub-expression
 AND	$C = A \cdot B$	$(\bar{A} \vee \bar{B} \vee C) \wedge (A \vee \bar{C}) \wedge (B \vee \bar{C})$
 NAND	$C = \overline{A \cdot B}$	$(\bar{A} \vee \bar{B} \vee \bar{C}) \wedge (A \vee C) \wedge (B \vee C)$
 OR	$C = A + B$	$(A \vee B \vee \bar{C}) \wedge (\bar{A} \vee C) \wedge (\bar{B} \vee C)$
 NOR	$C = \overline{A + B}$	$(A \vee B \vee C) \wedge (\bar{A} \vee \bar{C}) \wedge (\bar{B} \vee \bar{C})$
 NOT	$C = \bar{A}$	$(\bar{A} \vee \bar{C}) \wedge (A \vee C)$
 XOR	$C = A \oplus B$	$(\bar{A} \vee \bar{B} \vee \bar{C}) \wedge (A \vee B \vee \bar{C}) \wedge (A \vee \bar{B} \vee C) \wedge (\bar{A} \vee B \vee C)$
 XNOR	$C = \overline{A \oplus B}$	$(\bar{A} \vee \bar{B} \vee C) \wedge (A \vee B \vee C) \wedge (A \vee \bar{B} \vee \bar{C}) \wedge (\bar{A} \vee B \vee \bar{C})$

Simple combinatorial logic

The following circuit returns true when at least some of its inputs are true, but not more than two at a time. It implements the equation $y = \overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2} + \overline{x_2} \cdot x_3$. A variable is introduced for each gate's output; here each is marked in red:



Notice that the output of the inverter with x_2 as an input has two variables introduced. While this is redundant, it does not affect the equisatisfiability of the resulting equation. Now substitute each gate with its appropriate CNF sub-expression:

gate	CNF sub-expression
gate1	$(\text{gate1} \vee x1) \wedge (\overline{\text{gate1}} \vee \overline{x1})$
gate2	$(\overline{\text{gate2}} \vee \text{gate1}) \wedge (\overline{\text{gate2}} \vee x2) \wedge (\overline{x2} \vee \text{gate2} \vee \overline{\text{gate1}})$
gate3	$(\text{gate3} \vee x2) \wedge (\overline{\text{gate3}} \vee \overline{x2})$
gate4	$(\overline{\text{gate4}} \vee x1) \wedge (\overline{\text{gate4}} \vee \text{gate3}) \wedge (\overline{\text{gate3}} \vee \text{gate4} \vee \overline{x1})$
gate5	$(\text{gate5} \vee x2) \wedge (\overline{\text{gate5}} \vee \overline{x2})$
gate6	$(\overline{\text{gate6}} \vee \text{gate5}) \wedge (\overline{\text{gate6}} \vee x3) \wedge (\overline{x3} \vee \text{gate6} \vee \overline{\text{gate5}})$
gate7	$(\text{gate7} \vee \overline{\text{gate2}}) \wedge (\text{gate7} \vee \overline{\text{gate4}}) \wedge (\text{gate2} \vee \overline{\text{gate7}} \vee \text{gate4})$
gate8	$(\text{gate8} \vee \overline{\text{gate6}}) \wedge (\text{gate8} \vee \overline{\text{gate7}}) \wedge (\text{gate6} \vee \overline{\text{gate8}} \vee \text{gate7})$

The final output variable is gate8 so to enforce that the output of this circuit be true, one final simple clause is appended: (gate8). Combining these equations results in the final instance of SAT:

$$\begin{aligned}
& (\text{gate1} \vee x1) \wedge (\overline{\text{gate1}} \vee \overline{x1}) \wedge (\overline{\text{gate2}} \vee \text{gate1}) \wedge (\overline{\text{gate2}} \vee x2) \wedge \\
& (\overline{x2} \vee \text{gate2} \vee \overline{\text{gate1}}) \wedge (\text{gate3} \vee x2) \wedge (\overline{\text{gate3}} \vee \overline{x2}) \wedge (\text{gate4} \vee x1) \wedge \\
& (\overline{\text{gate4}} \vee \text{gate3}) \wedge (\overline{\text{gate3}} \vee \text{gate4} \vee \overline{x1}) \wedge (\text{gate5} \vee x2) \wedge \\
& (\overline{\text{gate5}} \vee \overline{x2}) \wedge (\overline{\text{gate6}} \vee \text{gate5}) \wedge (\overline{\text{gate6}} \vee x3) \wedge \\
& (\overline{x3} \vee \text{gate6} \vee \overline{\text{gate5}}) \wedge (\text{gate7} \vee \overline{\text{gate2}}) \wedge (\text{gate7} \vee \overline{\text{gate4}}) \wedge \\
& (\text{gate2} \vee \overline{\text{gate7}} \vee \text{gate4}) \wedge (\text{gate8} \vee \overline{\text{gate6}}) \wedge (\text{gate8} \vee \overline{\text{gate7}}) \wedge \\
& (\text{gate6} \vee \overline{\text{gate8}} \vee \text{gate7}) \wedge (\text{gate8}) = 1
\end{aligned}$$

One possible satisfying assignment of these variables is:

variable	value
gate2	0
gate3	1
gate1	1
gate6	1
gate7	0
gate4	0
gate5	1
gate8	1
x2	0
x3	1
x1	0

The values of the introduced variables are usually discarded, but they can be used to trace the logic path in the original circuit. Here, $(x1, x2, x3) = (0, 0, 1)$ indeed meets the criteria for the original circuit to output true. To find a different answer, the clause $(x1 \vee x2 \vee \overline{x3})$ can be appended and the SAT solver executed again.

Derivation

Presented is one possible derivation of the CNF sub-expression for some chosen gates:

OR Gate

An OR gate with two inputs A and B and one output C satisfies the following conditions:

1. if the output C is true, then at least one of its inputs A or B is true,
2. if the output C is false, then both its inputs A and B are false.

We can express these two conditions as the conjunction of two implications:

$$(C \rightarrow (A \vee B)) \wedge (\overline{C} \rightarrow (\overline{A} \wedge \overline{B}))$$

Replacing the implications with equivalent expressions involving only conjunctions, disjunctions, and negations yields

$$(\overline{C} \vee (A \vee B)) \wedge (C \vee (\overline{A} \wedge \overline{B})),$$

which is nearly in conjunctive normal form already. Distributing the rightmost clause twice yields

$$(\overline{C} \vee A \vee B) \wedge ((C \vee \overline{A}) \wedge (C \vee \overline{B})),$$

and applying the associativity of conjunction gives the CNF formula

$$(\overline{C} \vee A \vee B) \wedge (C \vee \overline{A}) \wedge (C \vee \overline{B})$$

NOT Gate

The NOT gate is operating properly when its input and output oppose each other. That is:

1. if the output C is true, the input A is false
2. if the output C is false, the input A is true

express these conditions as an expression that must be satisfied:

$$(C \rightarrow \overline{A}) \wedge (\overline{C} \rightarrow A), (\overline{C} \vee \overline{A}) \wedge (C \vee A)$$

NOR Gate

The NOR gate is operating properly when the following conditions hold:

1. if the output C is true, then neither A or B are true
2. if the output C is false, then at least one of A and B were true

express these conditions as an expression that must be satisfied:

$$\frac{(C \rightarrow \overline{(A \vee B)}) \wedge (\overline{C} \rightarrow (A \vee B)), \overline{\overline{(\overline{C} \vee (\overline{A} \wedge \overline{B}) \wedge (C \vee A \vee B))}}}{\overline{(C \wedge (A \vee B)) \wedge (C \vee A \vee B)}, \overline{(A \wedge C) \vee (B \wedge C) \wedge (C \vee A \vee B)}, (\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C}) \wedge (C \vee A \vee B)}$$

References

- G.S. Tseytin: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, pp. 115–125. Steklov Mathematical Institute (1970). Translated from Russian: *Zapiski Nauchnykh Seminarov LOMI* 8 (1968), pp. 234–259.
 - G.S. Tseytin: On the complexity of derivation in propositional calculus. Presented at the Leningrad Seminar on Mathematical Logic held in September 1966. (<http://www.decision-procedures.org/handouts/Tseit70.pdf>)
-

Retrieved from "https://en.wikipedia.org/w/index.php?title=Tseytin_transformation&oldid=1225425428"