Boolean Logic

CS254

Chris Pollett

Sep 27, 2006.

Outline

- CNF and DNF
- Satisfiability/Validity
- Horn Clauses
- Circuits
- A Counting Argument

Conjunctive/Disjunctive Normal Form

- A **literal** is a variable or the negation of a variable
- A **clause** is a finite ORing of literals. $(l_1 \vee l_2 \vee l_3 \ldots)$
- An **implicant** is a finite ANDing of literals $(l_1 \wedge l_2 \wedge l_3 \ldots)$
- A formula F is said to be in **conjunctive normal form(CNF)** if it is a finite ANDing of clauses.
- A formula F is said to be in **disjunctive normal form(DNF)** if it is a finite ORing of implicants.

Prop. Any boolean expression F is equivalent to a boolean expression in CNF and is equivalent to a boolean expression in DNF.

Sketch of Proof. View F as giving a boolean function of its variables to true and false. Define an implicant by setting what literal for a given variable is used according to a "true row" in F's truth table, then AND together all such implicants for true rows gives an equivalent DNF to F. The equivalent CNF is obtained similarly but using false rows.

Satisfiability and Validity

- A formula F is *satisfiable* if there is a truth assignment v such that v|= F
- A formula F is a *valid* (aka a *tautology*) if for every truth assignment v, v|=F.
- For example, (x ∨ ¬x) is both satisfiable and valid; (x ∧ y) is satisfiable, but not valid; (x ∧ ¬x) is not satisfiable.

Proposition A Boolean expression is unsatisfiable iff its negation is valid.

Proof. If F is unsatisfiable means that for every truth assignment v, v|=/= F. So for every truth assignment, v |= ¬F, i.e., ¬F is valid. The reverse direction is equally easy.

Horn Clauses

- It is unknown whether checking if a formula is satisfiable or not can be done in polynomial time on a deterministic TM.
- It can be done in nondeterministic polynomial time.
- One class of formulas for which we are able to check satisfiability in polynomial time are the Horn formulas.
- A *Horn clause* is a clause with at most one unnegated variable.
- A *Horn formula* is a conjunction of Horn clauses.

Algorithm for Checking Satisfiability of a Horn formula

1. Start with the all false truth assignment v. If it already satisfies all the clauses output "yes."
2. Repeat until all clauses satisfied.
   a) Pick an unsatisfied clause C with a unnegated variable. If no such clause exists go to 3.
   b) Change the value of this unnegated variable x in the truth assignment v to true.
3. If all clauses are satisfied output "yes"; otherwise, output "no".

- For two truth assignments v1, v2, the relationship $v1 \subseteq v2$ means that whenever $v1(x_i) = \text{true}$ then $v2(x_i) = \text{true}$.
- The v´ be a truth assignment satisfying a Horn formula F. Let v be the assignment given by the above algorithm. Then $v \subseteq v'$. To see this, suppose otherwise. Let C be first the clause and x be the corresponding variable in the execution of the algorithm that causes to not be contained in v´. This clause must be true in v´, but as x is positive in C, and C is not satisfied with an assignment contained in v´ where x is not true, either x must be true in v´ or v´ doesn't satisfy this clause.
- So if this algorithm ever makes a clause false after its at most one positive literal has been set, it is impossible for there to be a satisfying assignment.
- If a clause is not satisfied, but has not been disqualified for the above reason, its positive literal could still be set by the algorithm, so it will eventually be satisfied if the Horn formula is satisfiable.

Boolean Function

- An n-ary boolean function is a function

  f:{true,false}^{n} --> {true, false}.
- For example, ∧, ∨, =>, and <=> are examples of 2-ary (aka binary) boolean functions.
- How many n-ary boolean functions are there?
  - There are 2^n many possible inputs to an n-ary function.
  - For each of these we have a choice of setting the value to true or to false.
  - So there are 2^{2^n} many such functions.

Boolean functions and Boolean Expressions

- Notice a Boolean function is completely determined by its truth table, as this says for each input what its value is.
- Recall our proof that every Boolean expression is equivalent to a DNF, really showed every truth table can be written as a DNF. So we have:

Proposition. Any n-ary Boolean function can be expressed as a Boolean expression in n-variables.

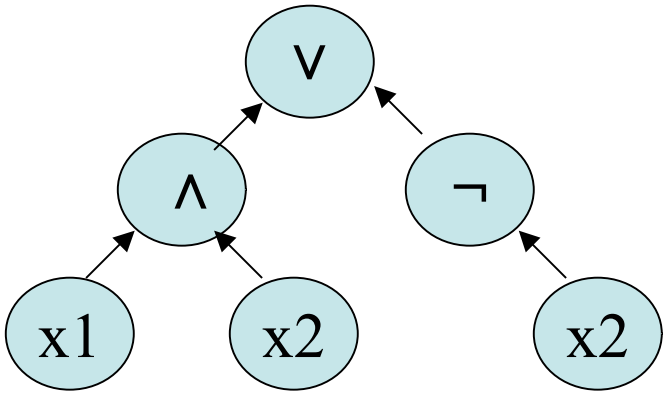
- Notice the size of the expression is roughly:

  (size to write down a variable)*(number of variables in a row)*(# of true rows in truth table). This give an upper bound on the size of $O(\log n * n * 2^n)$.

- It turns out that there are boolean functions which require very large circuits.

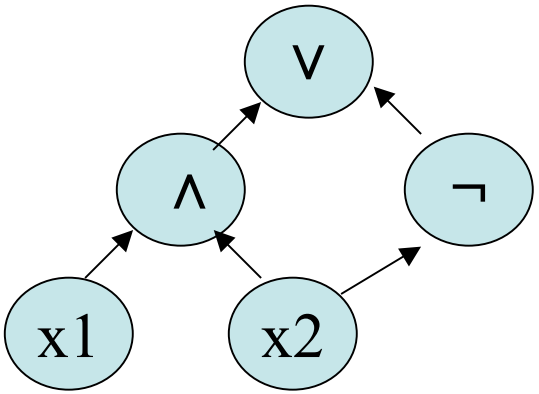
Boolean Circuits

- Boolean circuits are a more succinct was to represent Boolean functions than Boolean expressions
- Boolean circuits allow us to reuse computations we have already done. For example:



Boolean Expression

(viewed as a tree)



Equivalent Boolean Circuit (notice we can reuse x2)

Boolean Circuits Formal Definition

Defn. A *Boolean circuit* is a directed, acyclic, labeled graph (V, E, s), where s:V-->{set of labels}. We assume the vertices are numbered 1,..,n. All vertices in the graph have indegree 0, 1, or 2. Those of indegree 0 are called inputs and are labeled with variables or true or false. Those of indegree 1 are must be labeled with ¬ and those of indegree two can be with ∧ or ∨. We require the inputs j, k to any gate i be such that j<i, k<i. Gate n is called the output and has outdegree 0.

- To figure out the output of a circuit for a given truth assignment to the input variables we can evaluate the circuit from the lower numbered gates to the higher numbered gates using the usual rules for ∧,∨, ¬ at each gate.
- This procedure can be implemented in polynomial time and is called CIRCUIT VALUE.
- In contrast, the problem of determining whether a given circuit is satisfiable (CIRCUIT SAT) is not known to be in polynomial time.

A Counting Argument

Theorem. For any n≥2 there is an n-ary Boolean function f such that no Boolean circuit with $2^n/(2n)$ or fewer gates can compute it.

Proof. We know there are 2^{2^n} n-ary boolean functions. Let m= $2^n/(2n)$. Let's get an upper bound on how many circuits there are with at most this many gates. For each gate in such a circuit we have at most (n+5) choices for the gate type and at most m^2 choices for the gate inputs. As there are at most m gates in the circuit, we get at most $((n+5)m^2)^m$ possible circuits of size m. Let's compare this with 2^{2^n} by taking the log of both functions and recalling m= $2^n/(2n)$. The log of the number of n-ary Boolean functions is 2^n , on the hand the log of the number of circuits is 2^n [1- log[$4n^2/(n+5)$]/(2n)]. So some function must be missed.

More on Complexity Classes