COSC 290 Discrete Structures

Lecture 16: Structural induction on trees and SatSolver

Prof. Michael Hay
Wednesday, Mar. 7, 2018
Colgate University

Plan for today

1. SatSolver
2. Practice with structural induction

SatSolver

The task

Given a formula (aka proposition) in conjunctive normal form, determine whether or not there is an assignment of the variables that makes the proposition true.

This is a hugely important problem.

A massive number of computer science problems can be expressed as satisfiability problems. (See CS Connection on p. 326.)

Terminology review

- Recall what it means for a proposition to be written in conjunctive normal form (CNF).
  Example: proposition φ is in CNF:

$$\varphi := (p \vee q \vee r) \wedge (\neg p) \wedge (\neg q \vee \neg r)$$

- A model maintains a mapping between *variables* and *truth values* (basically, a dictionary mapping variables to T or F)
  Example: model $M := \{ p \rightarrow F, q \rightarrow F, r \rightarrow T \}$
- We can evaluate the truth value of a proposition under a model.
  Example: φ evaluates to True under model M.

How to solve the problem

To see if a formula is satisfiable, try *all possible truth assignments*. If one of them evaluates to true, then the formula is satisfiable.

How many truth assignments are there?

If there are n variables, there are 2^n truth assignments. (Why?)

Algorithm idea: use recursion to enumerate all possible truth assignments:

- use a model to keep track of current truth assignment
- initially, model is empty
- each recursive call: assign one more variable to a truth value
- base case: all variables are assigned a truth value

Pseudocode for simplest version of algorithm

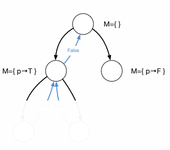
```
def isSat(formula, model):
  # base case
  if all variables have been assigned:
    if formula evaluates to true:
      return True
    else:
      return False
  # recursive case
  var = choose any unassigned variable
  for val in [True, False]:
    update model, assigning var to val
    if isSat(formula, model): # found sat assignment!
      return True
    update model, unassigning var
  return False  # formula is unsatisfiable
```


Example

Consider this CNF formula: $\varphi := (p \vee q \vee r) \wedge (\neg p) \wedge (\neg q \vee \neg r)$ Let's imagine calling `isSat` on formula φ.

Tree diagrams to visualize recursive calls

We will use a tree to diagram the execution of the algorithm: each function call is a node. The state of the model M is shown next to each node.



Making simple algorithm more efficient

A smarter base case: We don't necessarily need to wait until *every* variable has been assigned a truth value.

Examples:

- Fail fast: Suppose $\varphi := (p \vee q \vee r) \wedge (\neg p) \wedge (\neg q \vee \neg r)$ and $M = \{\, p \to T \,\}$. No need to check q and r; proposition cannot be satisfied when p is True.
- Recognize a win quickly: Suppose $\varphi := (p \vee q) \wedge (p \vee r) \wedge (p \vee \neg s)$ and $M = \{\, p \to T \,\}$. No need to check q and r; proposition is satisfiable when p is True.

Your task for lab 2: implement a smarter base case that uses methods Model.isTrue and Model.isFalse to stop recursion early!

Science!

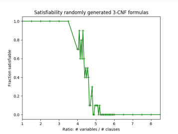


Figure 1: Discovery of a "phase transition" in satisfiability of CNF propositions.

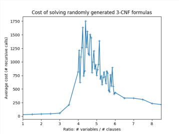


Figure 2: The "hard" problems appear to lie at this phase transition.

Practice with structural induction

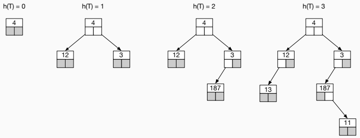
Recall: structural induction

If you have a recursively defined structure – a structure defined in terms of one or more *base* cases and one or more *inductive* cases – you can prove properties about it using structural induction.

With structural induction, proof components should align with components of recursive definition.

Height of a tree

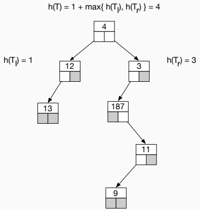
The level of a node in T is the length of the path from it to the root of T. The height of a tree is the max level of any (leaf) node in T. (If a tree has zero nodes, we say the height is -1.)



Height of a tree, defined recursively

We can also define height *recursively*: let $h(T)$ denote the height of tree T.

- Base case: tree T is empty, $h(T) = -1$.
- Inductive case: T is non-empty, thus it consists a root node x, a left subtree T_ℓ, and a right subtree T_r. Then, $h(T) = 1 + \max\{ h(T_\ell), h(T_r) \}$.



Exercise: prove claim

Claim: $nodes(T) \leq 2^{h(T)+1} - 1$

Please work in small groups at the white boards.

For reference: let $h(T)$ denote the height of tree T.

- Base case: tree T is empty, $h(T) = -1$.
- Inductive case: T is non-empty, thus it consists a root node x, a left subtree T_ℓ, and a right subtree T_r. Then, $h(T) = 1 + \max\{ h(T_\ell), h(T_r) \}$.

Poll: Lower bound?

- **False Claim**: $nodes(T) \geq 2^{h(T)+1} - 1$
- **Faulty proof by structural induction**:
  - **Base cases:** T is empty, height is -1 and $nodes(T) \geq 2^{-1+1} - 1 = 0$.
  - **Inductive case:** T is a non-empty tree of height h, consisting of node x and left and right subtrees T_ℓ and T_r.

$$\begin{aligned} nodes(T) &= 1 + nodes(T_\ell) + nodes(T_r) && \text{(b. +1 for root)} \\ &\geq 1 + (2^{h(T_\ell)+1} - 1) + (2^{h(T_r)+1} - 1) && \text{(c. ind. hypothesis)} \\ &\geq 1 + (2^{(h-1)+1} - 1) + (2^{(h-1)+1} - 1) && \text{(d. subtree heights)} \\ &= 2^{h+1} - 1 = 2^{h(T)+1} - 1 && \text{(e. algebra)} \end{aligned}$$

Where's the flaw? A) first sentence of inductive case; B) line b; C) line c; D) line d; E) line e.

Poll: number of leaves

We just showed an upper bound on the number of nodes in T: $nodes(T) \leq 2^{h(T)+1} - 1$. What can we say about *leaves*(T), the number of leaves?

Give the *smallest* upper bound you can. (Hint: try some examples... then start sketching out a proof!)

Claim: *leaves*$(T) \leq$ what goes here?

A) 0
B) $2^{h(T)-1}$
C) $2^{h(T)}$
D) $2^{h(T)+1}$
E) $2^{h(T)+1} - 1$