Resolution

In this lecture you will see how to convert the “natural” proof system of previous lectures into one with fewer operators and only one proof rule.

You will see how this proof system can be used to solve the important Satisfiability problem.

Along the way we discuss equivalences between expressions and normal forms.

Problem: Satisfiability

- Given a set of Boolean constraints, can they all be satisfied at once?
- This kind of problem arises, for example, in timetable scheduling
- Formalization
  - Input: An expression in propositional logic
  - Question: Is there an assignment of Boolean values to the variables for which the expression evaluates to true?

Examples

- Let φ = (x or y) and (not(x) or y)
- Setting x = true, y = true satisfies φ
- Let φ’ = (x or y) and (not(x) or y) and (not(y))
- In this case, *no* assignment of truth values to variables makes formula true
- One way of seeing this is to use the truth table method
- But is there a more efficient algorithm for Satisfiability?

Some Equivalence Rules

De Morgan's Laws	not (x or y) = (not(x) and not(y)) not (x and y) = (not(x) or not(y))
Associativity Laws	x or (y or z) = (x or y) or z x and (y and z) = (x and y) and z
Commutativity Laws	x or y = y or x x and y = y and x
Idempotent laws	x or x = x x and x = x
Absorptive laws	x or (x and y) = x = x and (x or y)
Identity laws	x or false = x x and true = x
(Left) Distributivity laws	x or (y and z) = (x or y) and (x or z) x and (y or z) = (x and y) or (x and z)
(Right) Distributivity laws	(x and y) or z = (x or z) and (y or z) (x or y) and z = (x and z) or (y and z)

Normal Forms

It is not strictly necessary to use all the logical operators. For example:

$A \rightarrow B$	is equivalent to	not(A) or B
$A \rightarrow B$	is equivalent to	not(A and not(B))
A or B	is equivalent to	not(not(A) and not(B))

and so on….

We can use this to convert any expression into
An equivalent one with fewer operator types.

Conjunctive Normal Form (CNF)

Any expression in our logic can be rewritten as a set of sets of propositions or their negations:

[[E1,E2,…], … , [En,…]]

Equivalent to ((E1 or E2 or …) and … (En or …))

For example:

(a and not(b) $\rightarrow$ c) and a and not(c)

becomes

[[not(a),b,c], [a], [not(c)]]

What’s So Special About This?

The order of expressions (like E1) in an “or” set doesn’t matter

[[E1,E2,…], … , [En,…]]

The order of sub-sets (like [E1,E2,…]) in the “and” set doesn’t matter

Conversion to CNF

(a and not(b) → c) and a and not(c)

$\boxed{\text{P → Q equivalent to not(P) or Q}}$

(not(a and not(b)) or c) and a and not(c)

$\boxed{\text{not(P and Q) equivalent to not(P) or not(Q)}}$

(not(a) or not(not(b)) or c) and a and not(c)

$\boxed{\text{not(not(P)) equivalent to P}}$

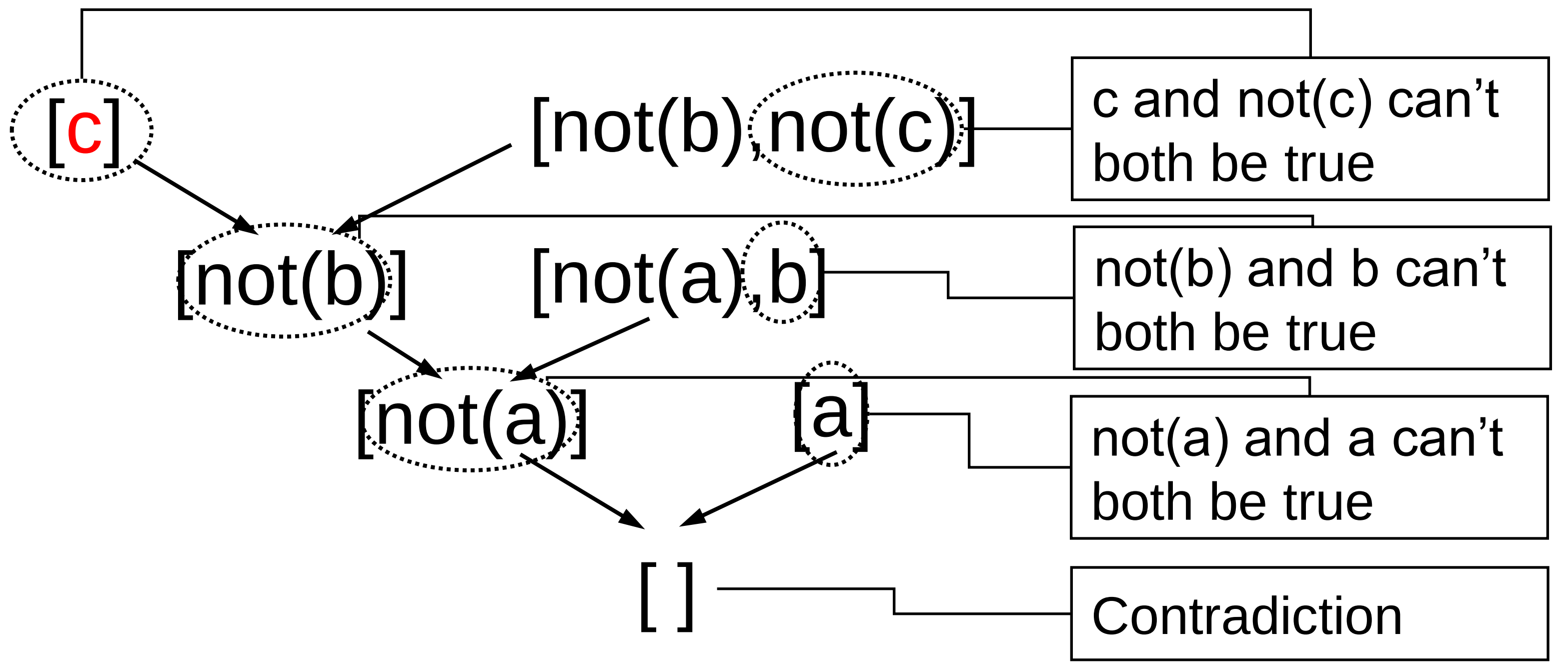
(not(a) or b or c) and a and not(c)

$\boxed{\text{(P or …) and … to [[P,…],…]}}$

[[not(a),b,c], [a], [not(c)]]

Showing Inconsistency

Suppose I have [[a], [not(a),b], [not(b),not(c)]] and I want to know if this is inconsistent with c


[c]
[not(b),not(c)]
c and not(c) can't both be true
[not(b)]
[not(a),b]
not(b) and b can't both be true
[not(a)]
[a]
not(a) and a can't both be true
[ ]
Contradiction

Proof by Inconsistency

To prove P from   [[E1,E2,…], … , [En,…]]

show that [[not(P)], [E1,E2,…], … , [En,…]] is inconsistent

A Resolution Proof

To prove b from [[not(a),b,c], [a], [not(c)]]

show that [[not(b)], [not(a),b,c], [a], [not(c)]] is inconsistent

$$\begin{tikzcd}
\text{[not(b)]} \arrow[dr] & & & \text{[not(a),b,c]} \arrow[dll] \\
& \text{[not(a), c]} \arrow[dr] & & \text{[a]} \arrow[dl] \\
& & \text{[c]} \arrow[dr] & \text{[not(c)]} \arrow[d] \\
& & & \text{[ ]}
\end{tikzcd}$$

One Proof Rule to Rule Them All

- Suppose we have already derived clauses C_1 and C_2, where C_1 contains the literal x and C_2 contains the literal not(x)
- Then we can derive the clause C_1 – {x} U C_2 – {not(x)}
- Goal: To derive the empty clause, i.e., the clause with no literals
- Clauses are inconsistent if and only if there is a way to derive empty clause

Problem: Satisfiability

- Is there a more efficient algorithm for Satisfiability than the truth table method?
- Truth table method requires building a truth table of size 2^n, where n is number of variables
- Idea: Use Resolution to try to *prove* that the input formula is inconsistent (unsatisfiable)

The Non-determinism Issue

- Resolution, as we have defined it, is not an *algorithm*
- There can be many options for which clauses to resolve at any point. Which to choose?
- Example: [[not(a),b], [not(b), c], [not(c),a]]
- In this case, every pair of clauses can be resolved

Davis-Putnam Algorithm for Resolution

- Choose an ordering of the variables (we will see how to do this with an example)
- For i = 1 to n do
  - Stage i: Resolve all pairs of clauses such that one contains the i'th variable negated, and the other contains the i'th variable un-negated
- Formula is unsatisfiable precisely when the empty clause occurs at the end

An Example

[[x,y], [x,z], [not(x), y,z], [not(z), y], [not[y]]

An Example

[[x,y], [x,z], [not(x), y,z], [not(z), y], [not[y]]

$$\downarrow$$

[[x,y], [x,y], [not(x), y], [not(y)]]

(Resolving on z)

An Example

[[x,y], [x,z], [not(x), y,z], [not(z), y], [not[y]]

$$\downarrow$$

[[x,y], [not(x), y], [not(y]]

(Resolving on z)

An Example

[[x,y], [x,z], [not(x), y,z], [not(z), y], [not[y]]

$\downarrow$

[[x,y], [not(x), y], [not(y]]

$\downarrow$

[[y], [not(y]]

(Resolving on x)

An Example

$$\begin{tikzcd}
\text{[[x,y], [x,z], [not(x), y,z], [not(z), y], [not[y]]} \arrow[d] \\
\text{[[x,y], [not(x), y], [not(y)]]} \arrow[d] \\
\text{[[y], [not(y)]]} \arrow[d] \\
\text{[[]]}
\end{tikzcd}$$

Unsatisfiable!

Another Example

[[x,y], [x,z], [not(z)], [not[y]]

Another Example

$$\begin{tikzcd}
\text{[[x,y], [x,z], [not(z)], [not[y]]} \arrow[d] \\
\text{[[x], [x,z], [not(z)]]}
\end{tikzcd}$$

(Resolving on y)

Another Example

$$\begin{tikzcd}
{[[x,y], [x,z], [not(z)], [not[y]]} \arrow[d] \\
{[[x], [x,z], [not(z)]]} \arrow[d] \\
{[[x]]}
\end{tikzcd}$$

(Resolving on z)

Another Example

$$\begin{tikzcd} {[[x,y], [x,z], [not(z)], [not[y]]]} \arrow[d] \\ {[[x], [x,z], [not(z)]]} \arrow[d] \\ {[[x]]} \end{tikzcd}$$

Satisfiable!

Things to Practice

- Convert some expressions involving all the connectives into CNF
- Try some resolution proofs, and some runs of the Davis-Putnam algorithm