CS389L: Automated Logical Reasoning

Lecture 2: Normal Forms and DPLL

Işıl Dillig

Overview

- An algorithm called DPLL for determining satisfiability
- Many SAT solvers used today based on DPLL
- However, requires converting formulas to a respresentation called normal forms

Normal Forms

- A normal form of a formula F is another formula F' such that F is equivalent to F', but F' obeys certain syntactic restrictions.
- There are three kinds of normal forms that are interesting in propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)

Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg, \wedge, \vee$ (i.e., no $\rightarrow$, $\leftrightarrow$)
- Negations appear only in literals
- i.e., negations not allowed inside $\wedge$, $\vee$, or any other $\neg$
- Is formula $p \vee (\neg q \wedge (r \vee \neg s))$ in NNF?
- What about $p \vee (\neg q \wedge \neg(\neg r \wedge s))$?
- What about $p \vee (\neg q \wedge (\neg\neg r \vee \neg s))$?

Conversion to NNF I

- To make sure the only logical connectives are $\neg, \wedge, \vee$, need to eliminate $\rightarrow$ and $\leftrightarrow$
- How do we express $F_1 \rightarrow F_2$ using $\vee, \wedge, \neg$?
- How do we express $F_1 \leftrightarrow F_2$ using only $\neg, \wedge. \vee$?

Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in
- Use DeMorgan's laws to distribute $\neg$ over $\wedge$ and $\vee$:

$$\neg(F_1 \wedge F_2) \Leftrightarrow \neg F_1 \vee \neg F_2$$

$$\neg(F_1 \vee F_2) \Leftrightarrow \neg F_1 \wedge \neg F_2$$

- We also disallow double negations:

$$\neg\neg F \Leftrightarrow F$$

NNF Example

Convert $F: \neg(p \rightarrow (p \wedge q))$ to NNF

Disjunctive Normal Form (DNF)

- A formula in disjunctive normal form is a disjunction of conjunction of literals.

$$\bigvee_i \bigwedge_j \ell_{i,j} \quad \text{for literals } \ell_{i,j}$$

- i.e., $\vee$ can never appear inside $\wedge$ or $\neg$
- Called disjunctive normal form because disjuncts are at the outer level
- Each inner conjunction is called a clause
- Question: If a formula is in DNF, is it also in NNF?

Conversion to DNF

- To convert formula to DNF, first convert it to NNF.
- Then, distribute $\wedge$ over $\vee$:

$$\begin{array}{lcl} (F_1 \vee F_2) \wedge F_3 & \Leftrightarrow & (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\ F_1 \wedge (F_2 \vee F_3) & \Leftrightarrow & (F_1 \wedge F_2) \vee (F_1 \wedge F_3) \end{array}$$

Example

Convert $F: (q_1 \vee \neg\neg q_2) \wedge (\neg r_1 \rightarrow r_2)$ into DNF

DNF and Satisfiability

- Claim: If formula is in DNF, trivial to determine satisfiability. How?
-
-
- Idea: To determine satisfiability, convert formula to DNF and just do a syntactic check.

DNF and Blow-up in formula size

- This idea is completely impractical. Why?
- Consider formula: $(F_1 \vee F_2) \wedge (F_3 \vee F_4)$
- In DNF:

$$(F_1 \wedge F_3) \vee (F_1 \wedge F_4) \vee (F_2 \wedge F_3) \vee (F_2 \wedge F_4)$$

- Every time we distribute, formula size doubles!
- Moral: DNF conversion causes exponential blow-up in size!
- Checking satisfiability by converting to DNF is almost as bad as truth tables!

Conjunctive Normal Form (CNF)

- A formula in conjuctive normal form is a conjunction of disjunction of literals.

$$\bigwedge_i \bigvee_j \ell_{i,j} \quad \text{for literals } \ell_{i,j}$$

- i.e., $\wedge$ not allowed inside $\vee, \neg$.
- Called conjunctive normal form because conjucts are at the outer level
- Each inner disjunction is called a clause
- Is formula in CNF also in NNF?

Conversion to CNF

- To convert formula to CNF, first convert it to NNF.
- Then, distribute $\vee$ over $\wedge$:

$$(F_1 \wedge F_2) \vee F_3 \quad \Leftrightarrow \quad (F_1 \vee F_3) \wedge (F_2 \vee F_3)$$
$$F_1 \vee (F_2 \wedge F_3) \quad \Leftrightarrow \quad (F_1 \vee F_2) \wedge (F_1 \vee F_3)$$

CNF Conversion Example

Convert $F: (p \leftrightarrow (q \rightarrow r))$ into CNF

DNF vs. CNF

- Fact: Unlike DNF, it is not trivial to determine satisfiability of formula in CNF.
- Does CNF conversion cause exponential blow-up in size?
- News: But almost all SAT solvers first convert formula to CNF before solving!

Why CNF?

- Interesting Question: If it is just as expensive to convert formula to CNF as to DNF, why do solvers convert to CNF although it is much easier to determine satisfiability in DNF?
-

Equisatisfiability

- Two formulas F and F' are equisatisfiable iff:

$\boxed{F \text{ is satisfiable if and only if } F' \text{ is satisfiable}}$

- If two formulas are equisatisfiable, are they equivalent?
- Example:
-
- Equisatisfiability is a much weaker notion than equivalence.
- But useful if all we want to do is determine satisfiability.

The Plan

- To determine satisfiability of F, convert formula to equisatisfiable formula F' in CNF
- Use an algorithm (DPLL) to decide satisfiability of F'
- Since F' is equisatisfiable to F, F is satifiable iff algorithm decides F' is satisfiable
- Big question: How do we convert formula to equisatisfiable formula without causing exponential blow-up in size?

Tseitin's Transformation

Tseitin's transformation converts formula F to equisatisfiable formula F' in CNF with only a linear increase in size.

Tseitin's Transformation I

- Step 1: Introduce a new variable p_G for every subformula G of F (unless G is already an atom).
- For instance, if $F = G_1 \wedge G_2$, introduce two variables p_{G_1} and p_{G_2} representing G_1 and G_2 respectively.
- p_{G_1} is said to be representative of G_1 and p_{G_2} is representative of G_2.

Tseitin's Transformation II

- Step 2: Consider each subformula

$$G : G_1 \circ G_2 \quad (\circ \text{ arbitrary boolean connective})$$

- Stipulate representative of G is equivalent to representative of $G_1 \circ G_2$

$$p_G \leftrightarrow p_{G_1} \circ p_{G_2}$$

- Step 3: Convert $p_G \leftrightarrow p_{G_1} \circ p_{G_2}$ to equivalent CNF (by converting to NNF and distributing $\vee$'s over $\wedge$'s).
- Observe: Since $p_G \leftrightarrow p_{G_1} \circ p_{G_2}$ contains at most three propositional variables and exactly two connectives, size of this formula in CNF is bound by a constant.

Tseitin's Transformation II

- Given original formula F, let p_F be its representative and let S_F be the set of all subformulas of F (including F itself).
- Then, introduce the formula

$$p_F \wedge \bigwedge_{G=(G_1 \circ G_2) \in S_F} CNF(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

- Claim: This formula is equisatisfiable to F.
- The proof is by structural induction
- Formula is also in CNF because conjunction of CNF formulas is in CNF.

Tseitin's Transformation and Size

- Using this transformation, we converted F to an equisatisfiable CNF formula F'.
- What about the size of F'?

$$p_F \wedge \bigwedge_{G=(G_1 \circ G_2) \in S_F} CNF(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

- $|S_F|$ is bound by the number of connectives in F.
- Each formula $CNF(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$ has constant size.
- Thus, trasformation causes only linear increase in formula size.
- More precisely, the size of resulting formula is bound by $30n + 2$ where n is size of original formula

Tseitin's Transformation Example

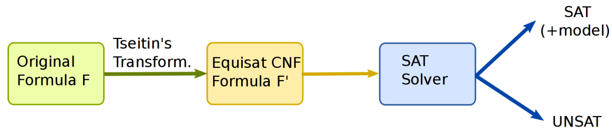
Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1.

2.

3.

SAT Solvers



- Almost all SAT solvers today are based on an algorithm called DPLL (Davis-Putnam-Logemann-Loveland)

DPLL: Historical Perspective

- 1962: the original algorithm known as DP (Davis-Putnam) $\Rightarrow$ "simple" procedure for automated theorem proving

- Davis and Putnam hired two programmers, George Logemann and David Loveland, to implement their ideas on the IBM 704.
- Not all of their ideas worked out as planned $\Rightarrow$ refined algorithm to what is known today as DPLL

DPLL insight

- There are two distinct ways to approach the boolean satisfiability problem:
- Search
  - Find satisfying assignment in by searching through all possible assignments $\Rightarrow$ most basic incarnation: truth table!
- Deduction
  - Deduce new facts from set of known facts $\Rightarrow$ application of proof rules, semantic argument method
- DPLL combines search and deduction in a very effective way!

Deduction in DPLL

- Deductive principle underlying DPLL is propositional resolution
- Resolution can only be applied to formulas in CNF
- SAT solvers convert formulas to CNF to be able to perform resolution

Propositional Resolution

- Consider two clauses in CNF:

$$C_1 : \quad (l_1 \lor \ldots p \ldots \lor l_k) \qquad C_2 : \quad (l'_1 \lor \ldots \neg p \ldots \lor l'_n)$$

- From these, we can deduce a new clause C_3, called resolvent:

$$C_3 : \quad (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots\ldots \lor l'_n)$$

- Correctness:
  - Suppose p is assigned $\top$: Since C_2 must be satisfied and since $\neg p$ is $\bot$, $(l'_1 \lor \ldots\ldots \lor l'_n)$ must be true.
  - Suppose p is assigned $\bot$: Since C_1 must be satisfied and since p is $\bot$, $(l_1 \lor \ldots\ldots \lor l_k)$ must be true.
  - Thus, C_3 must be true.

Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.
- Unit resolution is propositional resolution, but one of the clauses must be a unit clause (i.e., contains only one literal)
- $C_1 : p \qquad C_2 : \quad (l_1 \vee \ldots \neg p \ldots \vee l_n)$
- Resolvent: $(l_1 \vee \ldots \vee l_n)$
- Performing unit resolution on C_1 and C_2 is same as replacing p with true in the original clauses.
- In DPLL, all possible applications of unit resolution called Boolean Constraint Propagation (BCP).

Boolean Constraint Propagation (BCP) Example

- Apply BCP to CNF formula:

$$(p) \wedge (\neg p \vee q) \wedge (r \vee \neg q \vee s)$$

- Resolvent of first and second clause:
- New formula:
- Apply unit resolution again:
- No more unit resolution possible, so this is the result of BCP.

Basic DPLL

```
bool DPLL(φ)
{
  1. φ' = BCP(φ)
  2. if(φ' = ⊤) then return SAT;
  3. else if(φ' = ⊥) then return UNSAT;
  4. p = choose_var(φ');
  5. if(DPLL(φ'[p ↦ ⊤])) then return SAT;
  6. else return (DPLL(φ'[p ↦ ⊥]));
}
```

- Recursive procedure; input is formula in CNF
- Formula is $\top$ if no more clauses left
- Formula becomes $\bot$ if we derive $\bot$ due to unit resolution

An Optimization: Pure Literal Propagation

- If variable p occurs only positively in the formula (i.e., no $\neg p$), p must be set to $\top$
- Similarly, if p occurs only negatively (i.e., only appears as $\neg p$), p must be set to $\bot$
- This is known as Pure Literal Propagation (PLP).

DPLL with Pure Literal Propagation

```
bool DPLL(φ)
{
  1. φ' = BCP(φ)
  2. φ'' = PLP(φ')
  3. if(φ'' = ⊤) then return SAT;
  4. else if(φ'' = ⊥) then return UNSAT;
  5. p = choose_var(φ'');
  6. if(DPLL(φ''[p ↦ ⊤])) then return SAT;
  7. else return (DPLL(φ''[p ↦ ⊥]));
}
```


Example

$$F : (\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg q \vee \neg r) \wedge (p \vee \neg q \vee \neg r)$$

- No BCP possible because no unit clause
- No PLP possible because there are no pure literals
- Choose variable q to branch on:

$$F[q \mapsto \top] : (r) \wedge (\neg r) \wedge (p \vee \neg r)$$

- Unit resolution using (r) and $(\neg r)$ deduces $\bot \Rightarrow$ backtrack

Example Cont.

$$F: (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r)$$

- Now, try $q = \bot$

$$F[q \mapsto \bot]: (\neg p \lor r)$$

- By PLP, set p to $\bot$ and r to $\top$
- $F[q \mapsto \bot, p \mapsto \bot, r \mapsto \top] : \top$
- Thus, F is satisfiable and the assignment $[q \mapsto \bot, p \mapsto \bot, r \mapsto \top]$ is a model (i.e., a satisfying interpretation) of F.

Summary

- Normals forms: NNF, DNF, CNF (will come up again)
- For every formula, there exists an equivalent formula in normal form
- But equivalence-preserving transformation to DNF and CNF causes exponential blowup
- However, Tseitin's transformation gives an equisatisfiable formula in CNF with only linear increase in size
- Almost all SAT solvers work on CNF formulas to perform BCP
- DPLL basis of most state-of-the-art SAT solvers

Next Lecture

- Substantial improvements over basic DPLL used by modern SAT solvers: non-chronological backtracking and learning
- Implementation tricks used to perform BCP very efficiently
- Useful heuristics for choosing variable to branch on