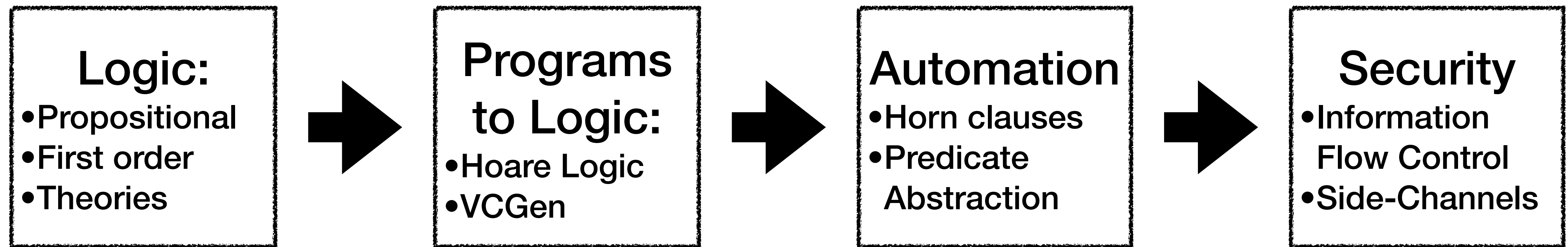


Normal Forms & DPLL

Klaus v. Gleissenthall





Where are we?

- Logic is the language of computation
- Propositional logic syntax & semantics
- Naive ways of checking sat: truth table and deductive proofs
- This lecture: checking sat efficiently via normal forms & DPLL

Normal Forms

- Syntactic restriction on formulas
- Idea: re-writing into equivalent normal form formula
- Engineering trick! Makes processing easier

Equivalence

Two formulas F_1 and F_2 are equivalent, write $F_1 \equiv F_2$ iff for all I : $I \models F_1$ iff $I \models F_2$

$F_1 \equiv F_2$ iff $F_1 \leftrightarrow F_2$ is valid

Quiz:

- Which of these equivalences hold?

1. $\perp \equiv \perp$

2. $\top \equiv \top$

3. $\neg \top \equiv \neg \perp$

4. $\neg(p \vee q) \equiv \neg p \wedge \neg q$

5. $p \equiv p \vee q$

6. $\neg \neg p \equiv p$

Normal Forms

Negation Normal Form

A formula F is in negation normal form (NNF) if:

- F only uses the connectives \wedge, \vee, \neg
- Negation only appears in literals

Quiz:

Are these formulas in negation normal form?

1. $p \rightarrow q$
2. $p \vee (\neg q \wedge (r \vee \neg s))$
3. $p \vee (\neg q \wedge \neg(\neg r \wedge s))$
5. $p \vee (\neg q \wedge (\neg\neg r \vee \neg s))$

Normal Forms

Conversion to Negation Normal Form

How to eliminate $\rightarrow, \leftrightarrow$?

Push negations inside formulas via **DeMorgan's laws**:

$$\neg(F1 \wedge F2) \equiv \neg F1 \vee \neg F2$$

$$\neg(F1 \vee F2) \equiv \neg F1 \wedge \neg F2$$

and eliminate double negation via $\neg\neg F \equiv F$.

Normal Forms

Converting to Negation Normal Form

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

Normal Forms

Disjunctive Normal Form (DNF)

A formula in disjunctive normal form (DNF) looks as follows:

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

That is, it's a disjunction of conjunctions of literals.

Quiz:

1. Is a formula that is in DNF also in NNF?
2. How can we check satisfiability of a formula in DNF?

Normal Forms

Converting into Disjunctive Normal Form

To convert to DNF:

- First convert into NNF
- Then distribute \wedge over \vee using the following equivalences:

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

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

Normal Forms

Converting into Disjunctive Normal Form

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

Normal Forms

Disjunctive Normal Form and Size Blow-up

Claim: For a DNF formula, it is trivial to determine satisfiability. How?

Idea: Convert into DNF, then do syntactic check.

Problem: DNF conversion causes exponential blow-up in size. For example:

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

Quiz:

Is DNF a good normal form for our solving algorithm?

Normal Forms

Conjunctive Normal Form

A formula in conjunctive normal form (CNF) looks as follows:

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

That is, it's a conjunction of disjunctions of literals (clauses).

Quiz:

1. Is a formula that is in CNF also in NNF?

Normal Forms

Converting into Conjunctive Normal Form

To convert to CNF:

- First convert into NNF
- Then distribute \vee over \wedge using the following equivalences:

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

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

Normal Forms

Converting into Conjunctive Normal Form

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

Normal Forms

DNF vs CNF

Unlike DNF, determining Satisfiability of a CNF formula is not trivial

Does CNF conversion cause exponential blow-up in size?

But almost all SAT solvers first convert formula to CNF before solving! *But Why?*

Normal Forms

Equisatisfiability

The answer lies in the notion of **equisatisfiability**

Two formulas F_1 and F_2 are equisatisfiability, iff F_1 is satisfiable iff F_2 is satisfiable

Quiz:

If F_1 and F_2 are equisatisfiability, are they equivalent?

Normal Forms

Equisatisfiability

The answer lies in the notion of **equisatisfiability**

Two formulas F_1 and F_2 are equisatisfiability, iff F_1 is satisfiable iff F_2 is satisfiable

Quiz:

If F_1 and F_2 are equisatisfiability, are they equivalent?

Example: p and $(p \vee q)$

Consider: $I \triangleq \{p \rightarrow \perp, q \rightarrow \top\}$

Normal Forms

The Plan

To determine satisfiability of F , convert formula to equisatisfiable formula F' in CNF

Use algorithm (DPLL) to decide satisfiability of F'

Since F' is equisatisfiable to F , F is satisfiable iff algorithm decides F' is satisfiable

But: How to convert formula to equisatisfiable formula without exponential blow-up in size?

Tseitin's Transformation

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

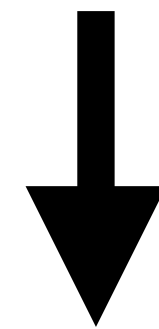
Properties:

- F is unsat iff F' is unsat
- Any model of F' is a model of F , if we disregard additional variables

Tseitin's Transformation

Intuition: Three Address Code

Example: Compute $\text{def } f(x, y, z) \{ \text{return } x + (2 * x + 3); \}$



```
def f '(x, y, z) {  
     $t_1 = 2 * y;$   
     $t_2 = t_1 + 3;$   
     $t_3 = x + t_2;$  return  $t_3;$   
}
```

- Introduce new variables t_1, t_2, t_3 .
- Use to define subexpressions
- Now the same in logic!

Tseitin's Transformation

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

Tseitin's Transformation

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

$$F_1' \triangleq t_1 \Leftrightarrow (\neg r \wedge s)$$

$$F' \triangleq F_1' \wedge F_2' \wedge F_3' \wedge F_4' \wedge t_4$$

$$F_2' \triangleq t_2 \Leftrightarrow (p \wedge t_1)$$

- Transform each F_i' using standard conversion

$$F_3' \triangleq t_3 \Leftrightarrow (p \wedge q)$$

$$F_4' \triangleq t_4 \Leftrightarrow (t_3 \vee t_2)$$

Tseitin's Transformation

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

$$F_1' \triangleq t_1 \Leftrightarrow (\neg r \wedge s)$$

$$F_2' \triangleq t_2 \Leftrightarrow (p \wedge t_1)$$

$$F_3' \triangleq t_3 \Leftrightarrow (p \wedge q)$$

$$F_4' \triangleq t_4 \Leftrightarrow (t_3 \vee t_2)$$

Convert formula F to an equisatisfiability formula F' in CNF with only a linear increase in size.

- equisatisfiability: Proof by structural induction
- size linear:
 - num of sub formulas, bounded by num of connectives
 - each F_i' has constant size

$$F' \triangleq F_1' \wedge F_2' \wedge F_3' \wedge F_4' \wedge t_4$$

Tseitin's Transformation

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

Tseitin's Transformation

- You'll implement the standard conversion as well as Tseitin's transformation in assignment 1.

DPLL

- DPLL (Davis-Putnam-Logemann-Loveland)
- Convert to CNF using Tseitin's Transformation
- How can we decide satisfiability of a CNF formula?
- We've seen:
 - Enumerating Interpretations (Search)
 - Semantic Arguments (Deductive Proofs)
- DPLL uses a mixture of both!

Deduction in DPLL

- Deductive principle underlying DPLL is propositional resolution
- Can only be applied to formulas in CNF
- That's why SAT solvers use Tseitin's transformation!

Propositional Resolution

- Let's look at two clauses in CNF

$$C_1 \triangleq (l_1 \vee \dots \vee p \vee \dots \vee l_k) \qquad C_2 \triangleq (l_1' \vee \dots \vee \neg p \vee \dots \vee l_k')$$

- We can deduce a new clause C_3 called the **resolvent**.

$$C_3 \triangleq (l_1 \vee \dots \vee l_k \vee l_1' \vee \dots \vee l_k')$$

- Why is this correct?
 - Suppose p is assigned \top , then $\neg p$ is \perp , and therefore $l_1' \vee \dots \vee l_k'$ must be true
 - Suppose p is assigned \perp , then p is \perp , and therefore $l_1 \vee \dots \vee l_k$ must be true
 - Thus C_3 must be true

Unit Resolution

- DPLL uses a restricted for called **unit resolution**
- Here, one of the clauses needs to be a unit clause (i.e., contain only one literal)

$$C_1 \triangleq p \qquad C_2 \triangleq (l_1 \vee \dots \vee \neg p \vee \dots \vee l_k)$$

- We get the **resolvent**:

$$C_3 \triangleq (l_1 \vee \dots \vee \dots \vee l_k)$$

- Same as replacing p with \top in C_1 and C_2
- Performing all possible applications of unit resolution is called Boolean Constraint Propagation (BCP).

Boolean Constraint Propagation (BCP)

Example:

- Apply BCP to the following formula:

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

DPLL

bool DPLL(φ) {

1. $\varphi' = \text{BCP}(\varphi)$
2. **if**($\varphi' = \top$) **then return SAT**;
3. **else if**($\varphi' = \perp$) **then return UNSAT**;
4. $p = \text{choose var}(\varphi')$;
5. **if**(DPLL(φ' [$p \rightarrow \top$])) **then return SAT**;
6. **else return** (DPLL(φ' [$p \rightarrow \perp$]));
7. }

- Notation: φ [$p \rightarrow e$] : formula φ , where we substitute e for p .

Optimization: Pure Literal Propagation

- If some variable p occurs **only positively** (i.e., no $\neg p$), set p to \top
- If some variable p occurs **only negatively** (i.e., only $\neg p$), set p to \perp
- Why is this correct?
- This is known as Pure Literal Propagation (PLP)

DPLL with PLP

bool DPLL(φ) {

1. $\varphi' = \text{BCP}(\varphi)$
2. $\varphi'' = \text{PLP}(\varphi)$
3. **if**($\varphi'' = \top$) **then return SAT**;
4. **else if**($\varphi'' = \perp$) **then return UNSAT**;
5. $p = \text{choose var}(\varphi'')$;
6. **if**(DPLL($\varphi'' [p \rightarrow \top]$)) **then return SAT**;
7. **else return** (DPLL($\varphi'' [p \rightarrow \perp]$));
8. }

DPLL with PLP

Example:

- Apply DPLL with PLP to the following formula

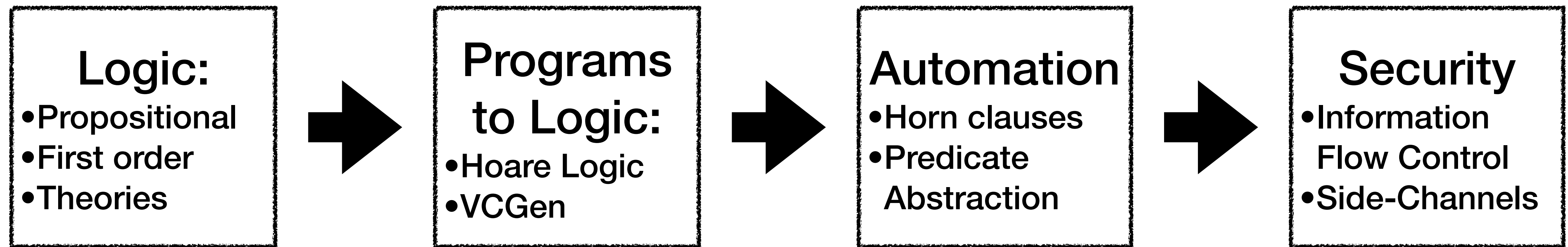
$$F \triangleq (\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg q \vee \neg r) \wedge (p \vee \neg q \vee \neg r)$$

DPLL(φ) {

1. $\varphi' = \text{BCP}(\varphi)$
2. $\varphi'' = \text{PLP}(\varphi)$
3. **if**($\varphi'' = \top$) **then return SAT**;
4. **else if**($\varphi'' = \perp$) **then return UNSAT**;
5. $p = \text{choose var}(\varphi'')$;
6. **if**(DPLL($\varphi'' [p \rightarrow \top]$)) **then return SAT**;
7. **else return** (DPLL($\varphi'' [p \rightarrow \perp]$));
8. }

Summary

- Normal forms: engineering trick to make solvers easier
- DNF and CNF conversions cause blow-up
- Tseitin's transformation avoids this, by a clever trick
- DPLL uses CNF form to perform resolution
- DPLL is the basis for most modern SAT solvers
- Many more optimization that we won't cover



Where are we?

- Logic as the language of computation
- We can now ask and answer questions in propositional logic
- But, it's too restricted to encode many important problems about programs
- Next lecture: more expressive logics:
 - First order logic (too expressive, as it's undecidable)
 - The solution: First order theories