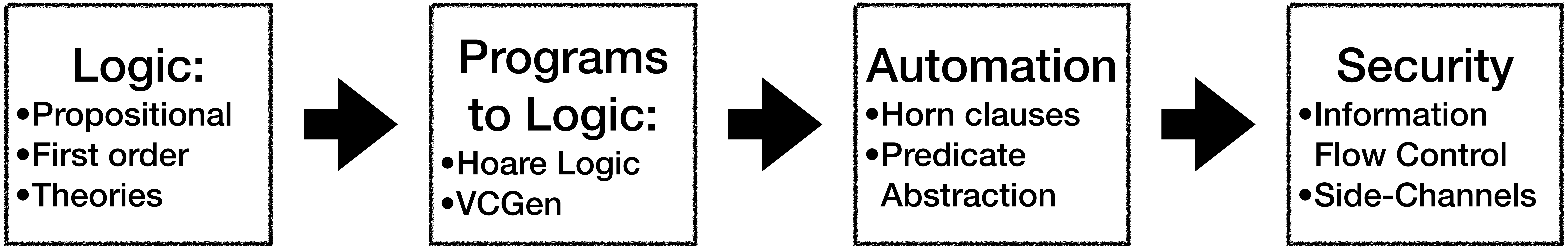
Normal Forms & DPLL

Klaus v. Gleissenthall

VRIJE UNIVERSITEIT AMSTERDAM


Logic:
•Propositional
•First order
•Theories
Programs to Logic:
•Hoare Logic
•VCGen
Automation
•Horn clauses
•Predicate Abstraction
Security
•Information Flow Control
•Side-Channels

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. $\bot \equiv \bot$
2. $\top \equiv \top$
3. $\neg\top \equiv \neg\bot$
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 formulas 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**:

$$\neg(F1 \land F2) \equiv \neg F1 \lor \neg F2$$

$$\neg(F1 \lor F2) \equiv \neg F1 \land \neg F2$$

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:

$$(p \land q \land \neg r) \lor (\neg p \land q \land \neg r) \lor \ldots \lor \ldots$$

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:

$(F_1 \lor F_2) \land (F_3 \lor F_4)$ turns into $(F_1 \land F_3) \lor (F_1 \land F_4) \lor (F_2 \land F_3) \lor (F_2 \land F_4)$

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:

$$(p \lor q \lor r) \land (\neg p \lor q \lor r) \land \ldots \land \ldots$$

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 ∨ over ∧ 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: $I \triangleq \{p \mapsto \bot, q \mapsto \top\}$

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 satifiable 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 def f(x, y, z) { return x + (2*x + 3);}

$\downarrow$

```
def f '(x, y, z) {
   t₁ = 2*y;
   t₂ = t₁ +3;
   t₃ = x + t₂; return t₃;
}
```

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

Tseitin's Transformation

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

Tseitin's Transformation

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

$F_1' \triangleq t_1 \leftrightarrow (\neg r \land s)$

$F_2' \triangleq t_2 \leftrightarrow (p \land t_1)$

$F_3' \triangleq t_3 \leftrightarrow (p \land q)$

$F_4' \triangleq t_4 \leftrightarrow (t_3 \lor t_2)$

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

- Transform each F_i' using standard conversion

Tseitin's Transformation

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

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

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

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

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

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

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

Tseitin's Transformation

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

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

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

- We can deduce a new clause C_1 called the resolvent.

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

- Why is this correct?
    - Suppose p is assigned $\top$, then $\neg p$ is $\bot$, and therefore $l_1' \vee \ldots \vee l_k'$ must be true
    - Suppose p is assigned $\bot$, then p is $\bot$, and therefore $l_1 \vee \ldots \vee l_k$ 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)

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

- We get the resolvent:

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

- 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:

$$p \land (\neg p \lor q) \land (r \lor \neg q \lor s)$$

DPLL

```
bool DPLL($\varphi$) {
    1. $\varphi' = BCP(\varphi)$
    2. **if**($\varphi' = \top$) then **return** SAT;
    3. **else if**($\varphi' = \bot$) then **return** UNSAT;
    4. p = choose var($\varphi'$);
    5. **if**( DPLL($\varphi'$ [p→⊤]) ) **then return** SAT;
    6. **else return** (DPLL($\varphi'$ [p→ ⊥]));
    7. }
```

- Notation: φ [p→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 $\bot$
- Why is this correct?
- This is known as Pure Literal Propagation (PLP)

DPLL with PLP

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

DPLL with PLP

Example:

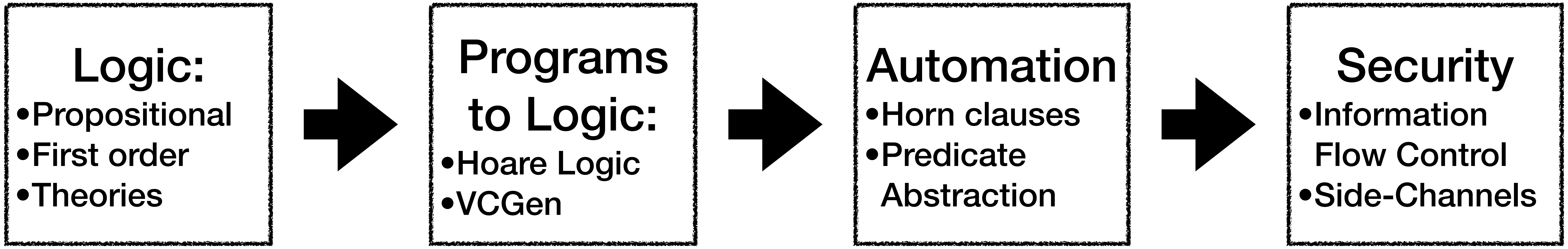
- Apply DPLL with PLP to the following formula

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

```
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→ ⊥]));
    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


Logic:
•Propositional
•First order
•Theories
Programs to Logic:
•Hoare Logic
•VCGen
Automation
•Horn clauses
•Predicate Abstraction
Security
•Information Flow Control
•Side-Channels

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