Propositional and First Order Reasoning

Terminology

- Propositional variable: boolean variable (p)
- Literal: propositional variable or its negation

  p $\neg$p
- Clause: disjunction of literals q $\vee$ $\neg$p $\vee$ $\neg$r
  given by set of literalsL {q, $\neg$p, $\neg$r}
- Conjunctive Normal Form: conjunction of clauses (q $\vee$ $\neg$p $\vee$ $\neg$r) $\wedge$ (p $\vee$ r)
  given by set of sets of literals
  { {q, $\neg$p, $\neg$r}, {p, q} }

Generate Verification Condition

```
if (p)
  q=true;
else
  r=true;
if (!p)
  q=false;
else
  r=false;
assert(q = !r);
```

$$[(p \wedge q_1 \wedge (r_1 \leftrightarrow r)) \vee$$
$$(\neg p \wedge r_1 \wedge (q_1 \leftrightarrow q))] \wedge$$
$$[(\neg p \wedge \neg q_2 \wedge (r_2 \leftrightarrow r_1)) \vee$$
$$(p \wedge \neg r_2 \wedge (q_2 \leftrightarrow q_1))] \wedge$$
$$\neg(q_2 \leftrightarrow \neg r_2)$$

Resolution Rule

$$\frac{C \cup \{p\} \qquad D \cup \{\neg p\}}{C \cup D}$$

Goal: obtain empty clause (contradiction)

Observation: if the above resolution can be made, and if D' is a superset of D, then also this works (but is worse):

$$\frac{C \cup \{p\} \qquad D' \cup \{\neg p\}}{C \cup D'}$$

We keep only D. A subset clause **subsumes** its supersets.

Unit Resolution

unit clause: {p}

$$\frac{\{p\} \qquad D \cup \{\neg p\}}{D}$$

Since p is true, ¬p is false, so it can be removed

New clauses *subsumes* previous one

Boolean Constraint Propagation

def BCP(S : Set[Clause]) : Set[Clause] =
  if for some unit clause U $\in$ S clause C $\in$ S,
    resolve(U,C) $\notin$ S
  then BCP(S $\cup$ resolve(U,C))
  else S

def delSubsumed(S : Set[Clause]) : Set[Clause] =
  if there are C1,C2 $\in$ S such that C1 $\subseteq$ C2
  then delSubsumes(S \ {C2}) **else** S

DPLL Algorithm

```
def isSatDPLL(S : Set[Clause]) : Boolean =
 val S' = delSubsumed(BCP(S))
 if ({} in S') then false
 else if (S' has only unit clauses) then true
 else
   val P = pick a variable from FV(S')
   DPLL(F' union {p})  || DPLL(F' union {Not(p)})
```

How to obtain clauses?

Translate to Conjunctive Normal Form

Generate a set of clauses for a formula

A) Applying: $p \vee (q \wedge r) \rightarrow (p \vee q) \wedge (p \vee r)$

+ simple
+ no new variables introduced in translation
- obtain exponentially size formula, e.g. from
  $(p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_3) \vee \dots \vee (p_{n-1} \wedge \neg p_n)$

B) Introducing fresh variables – due to Tseitin

+ not exponential
+ useful and used in practice

Key idea: give names to subformulas

Apply Transformation to Example

- Without fresh variables
- With fresh variables

Tseitin’s translation

- Translate to negation normal form (optional)
  - push negation to leaves
  - polynomial time, simple transformation
- For each subformula F_i have variable p_i
- For F_i of the form $F_m \bigvee F_n$ introduce into CNF the conjunct

  p_i <-> $(p_m \bigvee p_n)$ i.e.

  $(p_i \rightarrow p_m \bigvee p_n)$, $(p_m \bigvee p_n) \rightarrow p_i$

  $\{\neg p_i , p_m , p_n \}, \{\neg p_m , p_i \}, \{\neg p_n , p_i \}$
- 3 small clauses per node of original formula

Polynomial algorithm for SAT?

- Checking satisfiability of formulas in DNF is polynomial time process
  - DNF is disjunction of conjunctions of literals
  - If a formula is in DNF, it is satisfiable iff one of its disjuncts is satisfiable
  - A conjunction is satisfiable iff it does not list two contradictory literals
- Algorithm:
  - Analogously to CNF, use Tseitin's transformation to generate DNF of a formula
  - test the satisfiability of the resulting formula

Correctness of Tseitin's transformation

- Original formula: F
- Translated formula: [[F]]
- Variables introduced in translation: $p_1, \ldots, p_n$

[[F]] is equivalent to $\exists p_1, \ldots, p_n.\ F$

- A satisfiable assignment for [[F]] is a satisfiable assignment for F.
- If we find satisfiable assignment for F, subformulas give us assignment for p_i

DPLL

Davis–Putnam–Logemann–Loveland

- Algorithm for SAT
- Key ideas
  - use Boolean Constraint Propagation (BCP) exhaustively apply unit resolution
  - otherwise, try to set variable p true/false (add the appropriate unit clause {p}, {¬ p})

DPLL Algorithm

```
def isSatDPLL(S : Set[Clause]) : Boolean =
  val S' = delSubsumed(BCP(S))
  if ({} in S') then false
  else if (S' has only unit clauses) then true
  else
    val P = pick a variable from FV(S')
    DPLL(S' union {p})  || DPLL(S' union {Not(p)})
```

DPLL is complete

- Case analysis on all truth values
- Truth table method, with optimizations

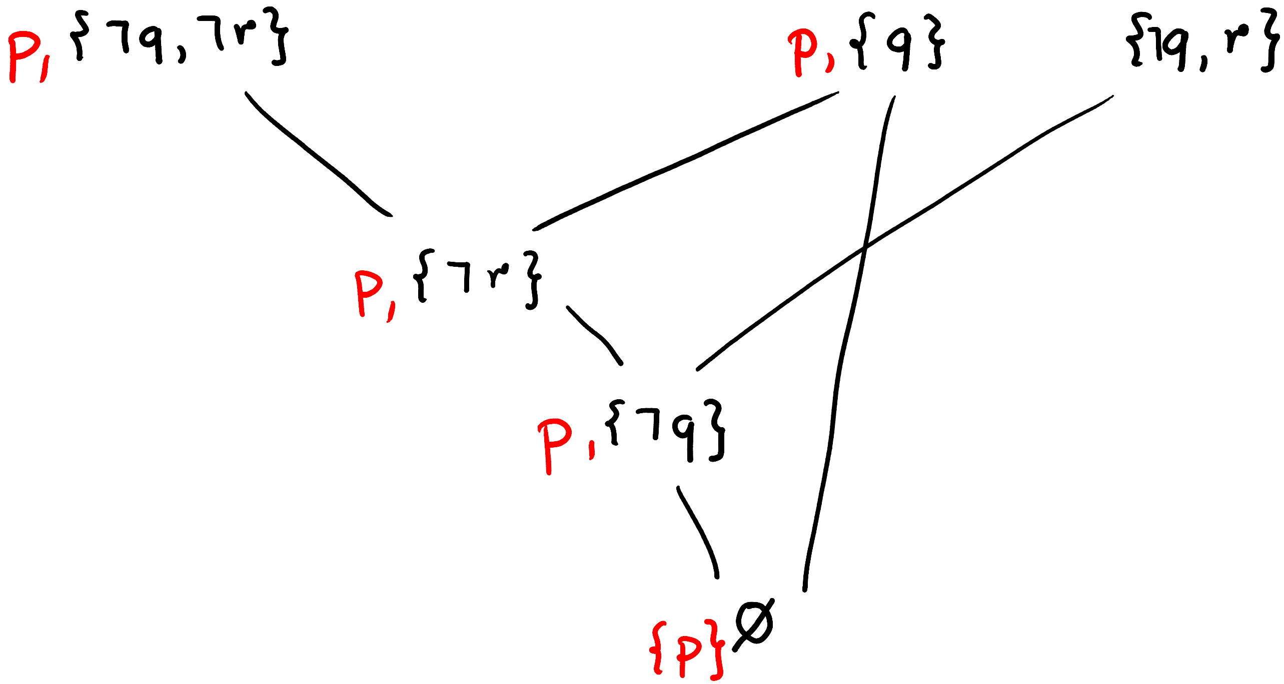
DPLL Proof is Resolution Proof

- Why is each reasoning step resolution
- When DPLL terminates, it can emit a proof
- Claim:
  - it can always emit a **resolution proof**
  - emitting proofs is only polynomial overhead, a natural extension of the algorithm
- What steps does DPLL make:
  - unit resolution is resolution
  - subsumption – does not remove proof existence
  - case analysis on truth values – why is it resolution?

$\{p, \neg q, \neg r\}$ $\{q, \neg p\}$ $\{p, q\}$ $\{\neg q, r\}$

false ∨ ... ∨ true false ∨ ...

decision: p → false



if

$S \cup \{\neg p\} \vdash \emptyset$

then

$S \vdash \{p\}$

Why Case Analysis is Resolution

$C \cup \{\neg p\} \vdash \emptyset$

so, have proof

C

$\{p\}$

$C \cup \{p\} \vdash \emptyset$

so, have proof

C

$\{\neg p\}$

$\emptyset$

so, got proof for

C

$\emptyset$

First-Order Logic Terminology

- **Terms**: built using **function symbols** from
  - variables
  - constants
- **Atomic formulas**: combine terms using relation symbols
  - they are like propositional formulas (but have structure)
  - equality is one binary relation symbol
- **Literal**: atomic formula or its negation
- **Clause**: disjunction of literals
- **Conjunctive Normal Form:** conjunction of clauses
  $\{ \{Q(f(x),x), \neg P(a), \neg R(x,f(x))\}, \{Q(a,b), P(b)\} \}$