CSE 473 Propositional Logic
SAT Algorithms

Luke Zettlemoyer
(With many slides from Dan Weld, Raj Rao, Mausam, Stuart Russell, Dieter Fox, Henry Kautz, Min-Yen Kan...)

Irrationally held truths may be more harmful than reasoned errors.

- Thomas Huxley (1825-1895)
Propositional Logic

• **Syntax**
  – Atomic sentences: P, Q, ...
  – Connectives: ∧, ∨, ¬, ⇒

• **Semantics**
  – Truth Tables

• **Inference**
  – Modus Ponens
  – Resolution
  – DPLL
  – GSAT

• **Complexity**
Truth tables for connectives

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$Q$</td>
<td>$\neg P$</td>
<td>$P \land Q$</td>
<td>$P \lor Q$</td>
<td>$P \implies Q$</td>
<td>$P \iff Q$</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Types of Reasoning (Inference)

• **Deduction** *(showing entailment, \(|=\))*
  
  \(S = \text{question}\)
  
  Prove that \(\text{KB} \models S\)
  
  Typically use rules to derive new formulas from old (inference)

• **Model Finding** *(showing satisfiability)*
  
  \(S = \text{description of problem}\)
  
  Show \(S\) is satisfiable
Validity and Satisfiability

A sentence is valid if it is true in all models,
  e.g., \( \text{True}, \ A \lor \neg A, \ A \Rightarrow A, \ (A \land (A \Rightarrow B)) \Rightarrow B \)

Validity is connected to inference via the Deduction Theorem:
  \( KB \models \alpha \) if and only if \( (KB \Rightarrow \alpha) \) is valid

A sentence is satisfiable if it is true in some model
  e.g., \( A \lor B, \ C \)

A sentence is unsatisfiable if it is true in no models
  e.g., \( A \land \neg A \)

Satisfiability is connected to inference via the following:
  \( KB \models \alpha \) if and only if \( (KB \land \neg \alpha) \) is unsatisfiable
  i.e., prove \( \alpha \) by reductio ad absurdum
Inference

$KB \vdash_i \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ by procedure } i$

Consequences of $KB$ are a haystack; $\alpha$ is a needle.
Entailment = needle in haystack; inference = finding it

Soundness: $i$ is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: $i$ is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the $KB$. 
Truth Tables for Inference

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>$KB$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Enumerate rows (different assignments to symbols),
if $KB$ is true in row, check that $\alpha$ is too

**Problem:** exponential time and space!
Logical Equivalence

Two sentences are logically equivalent iff true in same models:

\( \alpha \equiv \beta \) if and only if \( \alpha \models \beta \) and \( \beta \models \alpha \)

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg \alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \beta \Rightarrow \neg \alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \iff \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg \alpha \lor \neg \beta) \quad \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg \alpha \land \neg \beta) \quad \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
Proof Methods

Proof methods divide into (roughly) two kinds:

Application of inference rules
- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
- Typically require translation of sentences into a normal form

Model checking
  truth table enumeration (always exponential in $n$)
  improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
  heuristic search in model space (sound but incomplete)
    e.g., min-conflicts-like hill-climbing algorithms
Special Syntactic Forms

• General Form:
  
  \[(q \land \neg r) \rightarrow s) \land \neg (s \land t)\]

• Conjunction Normal Form (CNF)

  \[\neg q \lor r \lor s \land (\neg s \lor \neg t)\]
  
  Set notation: \{ (\neg q, r, s), (\neg s, \neg t) \}
  
  empty clause () = false

• Binary clauses: 1 or 2 literals per clause

  \[\neg q \lor r\] \quad \[\neg s \lor \neg t\]

• Horn clauses: 0 or 1 positive literal per clause

  \[\neg q \lor \neg r \lor s\] \quad \[\neg s \lor \neg t\]
  
  \[(q \land r) \rightarrow s\] \quad \[(s \land t) \rightarrow false\]
Propositional Logic: Inference Algorithms

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. Exhaustive Enumeration
4. DPLL (Davis, Putnam Loveland & Logemann)
5. GSAT

\{ Deduction \}
\{ Model Finding \}
Example

KB with Horn Clauses

\[
P \Rightarrow Q
\]
\[
L \land M \Rightarrow P
\]
\[
B \land L \Rightarrow M
\]
\[
A \land P \Rightarrow L
\]
\[
A \land B \Rightarrow L
\]
\[
A
\]
\[
B
\]

Proof And/Or Graph
Inference Technique II: Forward/Backward Chaining

• Require sentences to be in **Horn Form**:
  
  KB = conjunction of Horn clauses
  
  – Horn clause =
    
    • proposition symbol or
    
    • “(conjunction of symbols) ⇒ symbol”
      
      (i.e. clause with at most 1 positive literal)
  
  – E.g., KB = C ∧ (B ⇒ A) ∧ (C ∧ D ⇒ B)

• F/B chaining based on “Modus Ponens” rule:
  
  \[
  \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta
  \]
  
  B
  
  – Sound and complete for Horn clauses
Forward chaining algorithm

function PL-FC-ENTAILS?(KB, q) returns true or false

local variables: count, a table, indexed by clause, initially the number of premises
inferred, a table, indexed by symbol, each entry initially false
agenda, a list of symbols, initially the symbols known to be true

while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
        inferred[p] ← true
        for each Horn clause c in whose premise p appears do
            decrement count[c]
            if count[c] = 0 then do
                if HEAD[c] = q then return true
                PUSH(HEAD[c], agenda)
    return false
Query = Q
(i.e. “Is Q true?”)
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Backward chaining

Idea: work backwards from the query $q$:

- to prove $q$ by BC,
  - check if $q$ is known already, or
  - prove by BC all premises of some rule concluding $q$

Avoid loops: check if new subgoal is already on goal stack

Avoid repeated work: check if new subgoal
- 1. has already been proved true, or
- 2. has already failed
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Forward vs. backward chaining

• FC is data-driven, automatic, unconscious processing,
  – e.g., object recognition, routine decisions

• FC may do lots of work that is irrelevant to the goal

• BC is goal-driven, appropriate for problem-solving,
  – e.g., How do I get an A in this class?
  – e.g., What is my best exit strategy out of the classroom?
  – e.g., How can I impress my date tonight?

• Complexity of BC can be much less than linear in size of KB
Inference 2: Resolution
[Robinson 1965]

\{ (p \lor \alpha), (\neg p \lor \beta \lor \gamma) \} \vdash_R (\alpha \lor \beta \lor \gamma)

Correctness

If S1 \vdash_R S2 then S1 \models S2

Refutation Completeness:

If S is unsatisfiable then S \vdash_R ()
Conversion to CNF

\[ B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \beta \]

1. Eliminate \( \iff \), replacing \( \alpha \iff \beta \) with \( (\alpha \implies \beta) \land (\beta \implies \alpha) \).
   \[ (B_{1,1} \implies (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1}) \]

2. Eliminate \( \implies \), replacing \( \alpha \implies \beta \) with \( \neg \alpha \lor \beta \).
   \[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1}) \]

3. Move \( \neg \) inwards using de Morgan's rules and double-negation:
   \[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \lor \neg P_{2,1}) \lor B_{1,1}) \]

4. Apply distributivity law (\( \land \) over \( \lor \)) and flatten:
   \[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \]
Resolution algorithm

- To show $\text{KB} \vdash \alpha$, use proof by contradiction, i.e., show $\text{KB} \land \neg \alpha$ unsatisfiable

```plaintext
function PL-RESOLUTION (KB, \alpha) returns true or false

    clauses \leftarrow \text{the set of clauses in the CNF representation of } KB \land \neg \alpha
    new \leftarrow \{ \}
    loop do
        for each $C_i, C_j$ in clauses do
            resolvents \leftarrow PL-RESOLVE (C_i, C_j)
            if resolvents contains the empty clause then return true
            new \leftarrow new \cup resolvents
        end loop
    end loop
    if new \subseteq clauses then return false
    clauses \leftarrow clauses \cup new
```
Resolution

If the unicorn is mythical, then it is immortal, but if it is not mythical, it is a reptile. If the unicorn is either immortal or a reptile, then it is horned.

Prove: the unicorn is horned.

\[
\begin{align*}

&M = \text{mythical} \\
&I = \text{immortal} \\
&R = \text{reptile} \\
&H = \text{horned}
\end{align*}
\]
Resolution as Search

• States?
• Operators
**Model Checking**: Truth tables for inference

\[
\begin{array}{ccccccccc}
B_{1,1} & B_{2,1} & P_{1,1} & P_{1,2} & P_{2,1} & P_{2,2} & P_{3,1} & KB & \alpha_1 \\
false & false & false & false & false & false & false & false & true \\
false & false & false & false & false & false & true & true & true \\
false & true & false & false & false & false & true & true & true \\
false & true & false & true & false & true & true & true & true \\
false & true & true & true & true & true & true & true & true \\
true & true & true & true & true & true & true & false & false
\end{array}
\]

\[
\text{alpha}_1 = \text{not } P_{\{12\}} \text{ ("[1,2] is safe")}
\]
Inference 4: DPLL
(Enumeration of *Partial* Models)
[Davis, Putnam, Loveland & Logemann 1962]
Version 1

dpll_1(pa) {
    if (pa makes F false) return false;
    if (pa makes F true) return true;
    choose P in F;
    if (dpll_1(pa ∪ {P=0})) return true;
    return dpll_1(pa ∪ {P=1});
}

Returns true if F is satisfiable, false otherwise
DPLL Version 1

\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
DPLL Version 1

\[(a \lor b \lor c)\]
\[(a \lor \neg b)\]
\[(a \lor \neg c)\]
\[(\neg a \lor c)\]
DPLL Version 1

\[(F \lor b \lor c)\]
\[(F \lor \neg b)\]
\[(F \lor \neg c)\]
\[(T \lor c)\]
DPLL Version 1

(F v F v c)
(F v T)
(F v ¬c)
(T v c)
DPLL Version 1

(F ∨ F ∨ F)  
(F ∨ T)  
(F ∨ T)  
(T ∨ F)
DPLL Version 1
(a ∨ b ∨ c)
(a ∨ ¬b)
(a ∨ ¬c)
(¬a ∨ c)
DPLL as Search

- Search Space?
- Algorithm?
Improving DPLL

If literal $L_1$ is true, then clause $(L_1 \lor L_2 \lor \ldots)$ is true.

If clause $C_1$ is true, then $C_1 \land C_2 \land C_3 \land \ldots$ has the same value as $C_2 \land C_3 \land \ldots$.

Therefore: Okay to delete clauses containing true literals!
If literal $L_1$ is true, then clause $(L_1 \lor L_2 \lor ...) \text{ is true}$
If clause $C_1$ is true, then $C_1 \land C_2 \land C_3 \land ...$ has the same value as $C_2 \land C_3 \land ...$
Therefore: Okay to delete clauses containing true literals!
If literal $L_1$ is false, then clause $(L_1 \lor L_2 \lor L_3 \lor ...) \text{ has}$ the same value as $(L_2 \lor L_3 \lor ...)$
Therefore: Okay to delete shorten containing false literals!
Improving DPLL

If literal $L_1$ is true, then clause $(L_1 \lor L_2 \lor \ldots)$ is true.

If clause $C_1$ is true, then $C_1 \land C_2 \land C_3 \land \ldots$ has the same value as $C_2 \land C_3 \land \ldots$.

Therefore: Okay to delete clauses containing true literals!

If literal $L_1$ is false, then clause $(L_1 \lor L_2 \lor L_3 \lor \ldots)$ has the same value as $(L_2 \lor L_3 \lor \ldots)$.

Therefore: Okay to delete shortening containing false literals!

If literal $L_1$ is false, then clause $(L_1)$ is false.

Therefore: the empty clause means false!
DPLL version 2

dpl1_2(F, literal) {
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing ¬literal
    if (F contains empty clause) return false;
    choose V in F;
    if (dpl1_2(F, ¬V)) return true;
    return dpl1_2(F, V);
}

Partial assignment corresponding to a node is the set of chosen literals on the path from the root to the node
DPLL Version 2

(a ∨ b ∨ c)
(a ∨ ¬b)
(a ∨ ¬c)
(¬a ∨ c)
DPLL Version 2

(F ∨ b ∨ c)
(F ∨ ¬b)
(F ∨ ¬c)
(T ∨ c)
(\neg b)
(\neg c)

(b \lor c)

DPLL Version 2
DPLL Version 2

(F ∨ c)
(T)
(¬c)
DPLL Version 2

\[(c)\]

\[(\neg c)\]
DPLL Version 2

(F)

(T)

(a)

(b)

(c)
DPLL Version 2

Empty clause!

()
Representing Formulae

• CNF = Conjunctive Normal Form
  – Conjunction ($\land$) of Disjunctions ($\lor$)

• Represent as set of sets
  – $((A, B), (\neg A, C), (\neg C))$
  – $((\neg A), (A))$
  – $(())$
  – $((A))$
  – $(())$
Structure in Clauses

• Unit Literals
  A literal that appears in a singleton clause
  \{\neg b \ c\} \{\neg c\} \{a \neg b \ e\} \{d \ b\} \{e \ a \neg c\}
  *Might as well set it true!  And simplify*
  \{\neg b\} \{a \neg b \ e\} \{d \ b\}

• Pure Literals
  – A symbol that always appears with same sign
  – \{a \neg b \ c\} \{\neg c \ d \neg e\} \{\neg a \neg b \ e\} \{d \ b\} \{e \ a \neg c\}
  *Might as well set it true!  And simplify*
  \{a \neg b \ c\} \{\neg a \neg b \ e\} \{e \ a \neg c\}
In Other Words

Formula $(L) \land C_2 \land C_3 \land \ldots$ is only true when literal $L$ is true.

Therefore: Branch immediately on unit literals!

May view this as adding constraint propagation techniques into play.
In Other Words

Formula \((L) \land C_2 \land C_3 \land \ldots\) is only true when literal \(L\) is true.

Therefore: Branch immediately on unit literals!

If literal \(L\) does not appear negated in formula \(F\), then setting \(L\) true preserves satisfiability of \(F\).

Therefore: Branch immediately on pure literals!

May view this as adding constraint propagation techniques into play.
DPLL (previous version)
Davis – Putnam – Loveland – Logemann

dplll(F, literal) {
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing \(-\)literal
    if (F contains empty clause)

    return dplll(F, L);
    choose V in F;
    if (dplll(F, \(-\)V)) return true;
    return dplll(F, V);
}

DPLL (for real!)
Davis – Putnam – Loveland – Logemann

dpll(F, literal) {
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
    return false;
  if (F contains a unit or pure L)
    return dpll(F, L);
  choose V in F;
  if (dpll(F, ¬V)) return true;
  return dpll(F, V);
}
(a ⊕ b ⊕ c)
(a ⊕ ¬b)
(a ⊕ ¬c)
(¬a ⊕ c)

DPLL (for real)
Compare with DPLL Version 1

\[(a \lor b \lor c)\]

\[(a \lor \neg b)\]

\[(a \lor \neg c)\]

\[(\neg a \lor c)\]
DPLL (for real!)
Davis – Putnam – Loveland – Logemann

dpll(F, literal){
    remove clauses containing literal
    if (F contains no clauses) return true;
    shorten clauses containing ¬literal
    if (F contains empty clause)
        return false;
    if (F contains a unit or pure L)
        return dpll(F, L);
    choose V in F;
    if (dpll(F, ¬V)) return true;
    return dpll(F, V);
}
Heuristic Search in DPLL

• Heuristics are used in DPLL to select a (non-unit, non-pure) proposition for branching.

• Idea: identify a most constrained variable
  – Likely to create many unit clauses

• MOM’s heuristic:
  – Most occurrences in clauses of minimum length
Success of DPLL

• 1962 – DPLL invented
• 1992 – 300 propositions
• 1997 – 600 propositions (satz)
• Additional techniques:
  – Learning conflict clauses at backtrack points
  – Randomized restarts
  – 2002 (zChaff) 1,000,000 propositions – encodings of hardware verification problems
Other Ideas?

• How else could we solve SAT problems?
WalkSat (Take 1)

• *Local* search (Hill Climbing + Random Walk) over space of *complete* truth assignments
  – With prob $p$: flip any variable in any unsatisfied clause
  – With prob $(1-p)$: flip *best* variable in any unsat clause
    • best = one which minimizes #unsatisfied clauses
Refining Greedy Random Walk

• Each flip
  – makes some false clauses become true
  – breaks some true clauses, that become false
• Suppose $s_1 \rightarrow s_2$ by flipping $x$. Then:
  $$\#\text{unsat}(s_2) = \#\text{unsat}(s_1) - \text{make}(s_1,x) + \text{break}(s_1,x)$$
• Idea 1: if a choice breaks nothing, it’s likely good!
• Idea 2: near the solution, only the break count matters
  – the make count is usually 1
Walksat (Take 2)

state = random truth assignment;
while ! GoalTest(state) do
    clause := random member \{ C | C is false in state \};
    for each x in clause do compute break[x];
    if exists x with break[x]=0 then var := x;
    else
        with probability p do
            var := random member \{ x | x is in clause \};
        else
            var := arg x min \{ break[x] | x is in clause \};
        endif
    state[var] := 1 – state[var];
end
return state;

Put everything inside of a restart loop. Parameters: p, max_flips, max_runs
Random 3-SAT

- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
    - $n$ variables, $l$ clauses

- Which are the hard instances?
  - around $l/n = 4.3$
Random 3-SAT

- Varying problem size, $n$
- Complexity peak appears to be largely invariant of algorithm
  - backtracking algorithms like Davis-Putnam
  - local search procedures like GSAT
- *What’s so special about 4.3?*
Random 3-SAT

- Complexity peak coincides with solubility transition
  - \( l/n < 4.3 \) problems under-constrained and SAT
  - \( l/n > 4.3 \) problems over-constrained and UNSAT
  - \( l/n=4.3 \), problems on “knife-edge” between SAT and UNSAT
Prop. Logic Themes

• **Expressiveness**
  Expressive but awkward
  No notion of objects, properties, or relations
  Number of propositions is fixed

• **Tractability**
  NP in general
  Completeness / speed tradeoff
  Horn clauses, binary clauses