COMP36111: Advanced Algorithms I
Lecture 7: Propositional logic satisfiability

Ian Pratt-Hartmann

Room KB2.38: email: ipratt@cs.man.ac.uk

2016–17
Outline

Propositional logic

Clauses

The Davis-Putnam algorithm

The probability of satisfiability

Special cases

Summary
• Let \( P = \{p_1, p_2, \ldots\} \) be a countably infinite set. We call the elements of \( P \) *proposition letters*.

• The set of formulas of propositional logic is defined recursively as follows:
  
  • every element of \( P \) is a formula
  
  • if \( \varphi_1 \) and \( \varphi_2 \) are formulas, then so are

\[
(\neg \varphi_1), \ (\varphi_1 \lor \varphi_2), \ (\varphi_1 \land \varphi_2), \ (\varphi_1 \rightarrow \varphi_2)
\]

• For example:

\[
(\neg (p_1 \rightarrow ((\neg p_2) \lor p_3)))
\]

\[
((p_1 \rightarrow \neg p_1)) \land ((\neg p_1) \rightarrow p_1)
\]

are formulas

• We omit parentheses for clarity, using standard conventions:

\[
\neg (p_1 \rightarrow (\neg p_2 \lor p_3))
\]

\[
(p_1 \rightarrow \neg p_1) \land (\neg p_1 \rightarrow p_1)
\]
• An assignment is a function $\theta : \mathcal{P} \rightarrow \{T, F\}$.

• We extend $\theta$ to formulas by setting

\[
\theta(\neg \varphi_1) = T \text{ iff } \varphi_1 = F
\]
\[
\theta(\varphi_1 \lor \varphi_2) = T \text{ iff } \theta(\varphi_1) = T \text{ or } \theta(\varphi_2) = T
\]
\[
\theta(\varphi_1 \land \varphi_2) = T \text{ iff } \theta(\varphi_1) = T \text{ and } \theta(\varphi_2) = T
\]
\[
\theta(\varphi_1 \rightarrow \varphi_2) = T \text{ iff } \theta(\varphi_1) = F \text{ or } \theta(\varphi_2) = T
\]

• A formula $\varphi$ is satisfiable if there exists an assignment $\theta$ such that $\theta(\varphi) = T$.

• For example,

\[
\neg(p_1 \rightarrow (\neg p_2 \lor p_3))
\]

is satisfiable, but

\[
(p_1 \rightarrow \neg p_1) \land (\neg p_1 \rightarrow p_1)
\]

is not
• We then have the problem:

PROPOSITIONAL SAT
Given: a propositional logic formula $\varphi$;
Return: Yes if $\varphi$ is satisfiable, and No otherwise.

• Later on, we shall be interested in the complexity of the satisfiability problem.
Outline

Propositional logic

Clauses

The Davis-Putnam algorithm

The probability of satisfiability

Special cases

Summary
• It turns out that the following special case is as general as we need.

• A literal is an expression $p$ or $\neg p$, where $p$ is a proposition letter.

• A clause is an expression $\ell_1 \lor \cdots \lor \ell_k$, where the $\ell_i$ are literals. (We allow the empty disjunction, denoted $\bot$, which contains no literals.)

• Examples of clauses

\begin{align*}
& p_1 \lor \neg p_2 \lor p_3 \\
& \neg p_1 \lor \neg p_4 \lor \neg p_7 \lor p_8 \\
& \neg p_{14} \\
& p_1 \\
& \bot
\end{align*}
• We extend any assignment $\theta$ to literals by setting

$$\theta(\neg p) = \begin{cases} F & \text{if } \theta(p) = T \\ T & \text{otherwise} \end{cases}$$

and to to clauses by setting

$$\theta(\ell_1 \lor \cdots \lor \ell_k) = \begin{cases} T & \text{if } \theta(\ell_i) = T \text{ for some } i \\ F & \text{otherwise} \end{cases}$$

A set of clauses is *satisfiable* if there exists an assignment $\theta$ such that $\theta(\gamma) = T$ for all $\gamma \in \Gamma$. 
• Thus, the set of clauses

\[
\{(p_1 \lor -p_2 \lor p_3), (-p_1 \lor -p_4 \lor -p_7 \lor p_8), -p_{14}\}
\]

is clearly satisfiable.

• By contrast,

\[
\{(p_1 \lor p_2), (p_1 \lor -p_2), (-p_1 \lor p_2), (-p_1 \lor -p_2)\}
\]

is clearly unsatisfiable.
• We then have the problem:

**SAT**
Given: a set of clauses \( \Gamma \);
Return: Yes if \( \Gamma \) is satisfiable, and No otherwise.

• We are also interested in the special case where there is a fixed bound on the length of each clause.

• For \( k \geq 2 \), we have the problem

**\( k \)-SAT**
Given: a set of clauses \( \Gamma \), each with at most \( k \) literals;

Return: Yes if \( \Gamma \) is satisfiable, and No otherwise.
Outline

Propositional logic

Clauses

The Davis-Putnam algorithm

The probability of satisfiability

Special cases

Summary
• The *Davis-Putnam* (-Logemann-Loveland) algorithm

begin resolve(Γ, ℓ)
    for each γ ∈ Γ
        if γ contains ℓ, remove γ from Γ
        if γ contains ¬ℓ, remove ¬ℓ from γ

begin DPLL(Γ)
    if Γ is empty then return Yes
    if Γ contains the empty clause then return No
    while Γ contains any unit clause ℓ
        remove ℓ from Γ
        Γ = resolve(Γ, ℓ)
        if Γ is empty then return Yes
        if Γ contains the empty clause then return No
    let ℓ be the first literal of the first clause of Γ
    if DPLL(Γ ∪ {ℓ}) then return Yes
    if DPLL(Γ ∪ {¬ℓ}) then return Yes
return No
The DLLP algorithm (which is deterministic) can be seen to run in time bounded by $2^{p(n)}$, where $p$ is some fixed polynomial, and $n$ is the total size of $\Gamma$.

It follows that SAT is in $\text{ExpTime}$.

In fact, this algorithm is (close to) the best way of determining propositional clause satisfiability in practice.

Nevertheless, from the point of view of the complexity classes seen in the last lecture, we can do ‘better’ . . .
Consider the following **non-deterministic** algorithm for SAT

\[
\text{begin NdSatTest}(\Gamma) \\
\quad \text{if } \Gamma \text{ contains } \bot \text{ then return No} \\
\quad \text{while } \Gamma \text{ is non-empty} \\
\quad \quad \text{Select some proposition letter } p \text{ occurring in } \Gamma \\
\quad \quad \text{Either} \\
\quad \quad \quad \text{Delete every clause containing the literal } p \\
\quad \quad \quad \text{Delete } \neg p \text{ from all remaining clauses} \\
\quad \quad \text{Or} \\
\quad \quad \quad \text{Delete every clause containing the literal } \neg p \\
\quad \quad \quad \text{Delete } p \text{ from all remaining clauses} \\
\quad \quad \text{if } \Gamma \text{ contains } \bot \text{ then return No} \\
\quad \text{return Yes}
\]

Hence, SAT is in \text{NP\text{-}TIME}. 
Outline

Propositional logic

Clauses

The Davis-Putnam algorithm

The probability of satisfiability

Special cases

Summary
• Suppose we fix integers $m > 0$, $n > 0$ and $k > 1$.  
• There is a finite number of (multi-) sets of $m$ $k$-literal clauses over $n$ proposition letters.  
• Some of these will be satisfiable, others not. So how many are satisfiable (as a function of $m$, $k$ and $n$)?  
• Immediately, we see that, for fixed $k$:  
  • if $m/n$ is small, then the probability of satisfiability is high;  
  • if $m/n$ is large, then the probability of satisfiability is low.  
• But what does the relationship look like in detail?  
• In practice, we must solve this problem by generating a sample of sets of clauses at random, and then running an algorithm such as DPLL.
• Here is a graph I obtained by running my own implementation on large, randomly generated sets of 3-literal clauses.

• Probability of satisfiability is plotted against $m/n$ where $m$ is number of clauses and $n$ is number or proposition letters.

• Graphs are given for $n = 20$, $n = 30$, $n = 40$, $n = 50$. 
• The 50% satisfiability point seems to be achieved at around $m/n = 4.3$

• As $n \to \infty$, the 50% threshold value seems to approach a limit; moreover, the transition seems to get steeper with increasing $n$.

• This phenomenon is known as a *phase transition*: it still has the status of a conjecture.
• A clause $\ell_1 \lor \cdots \lor \ell_k$ is \textit{Horn} if all but at most one of the literals are negative.

• For example,

$$\neg p_1 \lor p_2, \quad \neg p_1 \lor \neg p_2 \lor p_3, \quad p_1, \quad \neg p_1$$

are all Horn, while

$$p_1 \lor p_2, \quad p_1 \lor \neg p_2 \lor p_3$$

are not.

• Note that a Horn clause

$$\neg p_1 \lor \cdots \lor \neg p_{k-1} \lor p_k$$

can be written as an implication

$$(p_1 \land \cdots \land p_{k-1}) \rightarrow p_k.$$
The problem *Horn-SAT* may now be defined as follows:

Given: A set of Horn clauses $\Gamma$
Return: Yes if $\Gamma$ is satisfiable, and No otherwise.

The following modification of DPLL decides *Horn-SAT*.

begin Horn-DPLL($\Gamma$)
    if $\Gamma$ contains the empty clause then return No
    while $\Gamma$ contains any unit clause $\ell$
        remove $\ell$ from $\Gamma$
        $\Gamma = \text{resolve}(\Gamma, \ell)$
        if $\Gamma$ contains the empty clause then return No
    return Yes
end Horn-DPLL

Horn-DPLL is easily seen to run in time $O(n^2)$. 
• Another special case is 2-SAT
• Terminology: a clause is *Krom* if it contains at most two literals.
• For example,

\[ \neg p_1 \lor p_2, \quad \neg p_1 \lor \neg p_2, \quad p_1, \quad \neg p_1 \]

are all Krom, while \( \neg p_1 \lor \neg p_2 \lor p_3 \) is not.
• The problem 2-SAT just asks for the satisfiability of Krom clauses.
• To think about:

How well does DPLL work for 2-SAT?
Devise an algorithm for deciding 2-Sat in time \( O(n^2) \)
Outline

Propositional logic

Clauses

The Davis-Putnam algorithm

The probability of satisfiability

Special cases

Summary
• We defined the problems PROPOSITIONAL SAT, SAT, $k$-SAT, HORN-SAT.

• We presented the DPLL algorithm for SAT, and saw that it runs in exponential time—even in the case 3-SAT. (We also showed that SAT is in $\text{NPTime}$.)

• We presented a modified version of this algorithm for Horn-SAT, and saw that it runs in polynomial time.

• I asked you to think about 2-SAT!