COMP36111: Advanced Algorithms I
Lecture 7: Try to be Logical . . .

Ian Pratt-Hartmann

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

2019–20
Outline

Propositional logic

Clauses

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
• 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 $\theta : \mathbf{P} \rightarrow \{ T, F \}$.

we extend $\theta$ to formulas by setting

$$
\theta(\neg \varphi_1) = T \iff \varphi_1 = F
$$

$$
\theta(\varphi_1 \lor \varphi_2) = T \iff \theta(\varphi_1) = T \text{ or } \theta(\varphi_2) = T
$$

$$
\theta(\varphi_1 \land \varphi_2) = T \iff \theta(\varphi_1) = T \text{ and } \theta(\varphi_2) = T
$$

$$
\theta(\varphi_1 \rightarrow \varphi_2) = T \iff \theta(\varphi_1) = F \text{ or } \theta(\varphi_2) = T
$$

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.

• We mentioned earlier that this problem is decidable. What is its complexity?
Outline

Propositional logic

Clauses

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
• 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

\[
\begin{align*}
  p_1 \lor \neg p_2 \lor p_3 \\
  \neg p_1 \lor \neg p_4 \lor \neg p_7 \lor p_8 \\
  \neg p_{14} \\
  p_1 \\
  \bot
\end{align*}
\]

• If $\ell$ is a literal, denote the opposite literal by $\bar{\ell}$. 
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

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

is clearly satisfiable.

By contrast, 

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

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

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
The *Davis-Putnam (-Logemann-Loveland)* algorithm

\[
\begin{align*}
\text{begin } & \text{resolve}(\Gamma, \ell) \\
& \text{for each } \gamma \in \Gamma \\
& \quad \text{if } \gamma \text{ contains } \ell, \text{ remove } \gamma \text{ from } \Gamma \\
& \quad \text{if } \gamma \text{ contains } \overline{\ell}, \text{ remove } \overline{\ell} \text{ from } \gamma \\
\end{align*}
\]

\[
\begin{align*}
\text{begin DPLL}(\Gamma) \\
& \text{if } \Gamma \text{ is empty then return Yes} \\
& \text{if } \Gamma \text{ contains the empty clause then return No} \\
& \text{while } \Gamma \text{ contains any unit clause } \ell \\
& \quad \text{remove } \ell \text{ from } \Gamma \\
& \quad \Gamma = \text{resolve}(\Gamma, \ell) \\
& \quad \text{if } \Gamma \text{ is empty then return Yes} \\
& \quad \text{if } \Gamma \text{ contains the empty clause then return No} \\
& \text{let } \ell \text{ be the first literal of the first clause of } \Gamma \\
& \text{if DPLL}(\Gamma \cup \{\ell\}) \text{ then return Yes} \\
& \text{if DPLL}(\Gamma \cup \{\overline{\ell}\}) \text{ then return Yes} \\
& \text{return No}
\end{align*}
\]
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

begin NdSatTest(Γ)
for each proposition letter \( p \) occurring in \( Γ \)
Either
    Delete every clause containing the literal \( p \)
Or
    Delete every clause containing the literal \( \neg p \)
if \( Γ = \emptyset \) then return Yes
return No

Hence, SAT is in \( \text{NPTime} \).
There is an easier way to think about this, and indeed about non-deterministic complexity classes in general.

Suppose we are given a formula \( \varphi \) of propositional logic (possibly, but not necessarily a clause).

Find proposition letters it contains and and guess an assignment \( \theta \) for them.

The guess can be made non-deterministically.

One can check, in polynomial time, whether \( \theta \) makes \( \varphi \) true.

Hence, PROPOSITIONAL SAT is in \( \text{NPTIME} \).
Outline

Propositional logic

Clauses

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
• A clause $\ell_1 \lor \cdots \lor \ell_k$ is *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

$$(p_1 \land \cdots \land p_{k-1}) \to p_k.$$
• The problem *Horn-SAT* may now be defined as follows:
  Given: A set of Horn clauses *Γ*
  Return: Yes if *Γ* is satisfiable, and No otherwise.
• The following modification of DPLL decides *Horn-SAT*.

  begin Horn-DPLL(*Γ*)
  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* 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 (or Krom-SAT) just asks for the satisfiability of Krom clauses.
• Let us write the opposite of any literal $\ell$ as $\bar{\ell}$.

• Note that (non-unit) Krom clauses may be regarded as implications:

\[ \ell \lor m \equiv \bar{\ell} \rightarrow m. \]

• We have the problem

KROM-SAT
Given: A set $\Gamma$ of Krom clauses.
Return: Yes if $\Gamma$ is satisfiable, and No otherwise.

• and its complement

KROM-UNSAT
Given: A set $\Gamma$ of Krom clauses.
Return: Yes if $\Gamma$ is unsatisfiable, and No otherwise.
Theorem

The problem KROM-UNSAT is in \textbf{NLogSpace}.

Proof.

Let a set of clauses $\Gamma$ be given. We may assume there are no unit clauses, since these can be eliminated by unit propagation. Also, we may assume $\bot \notin \Gamma$ and $\Gamma \neq \emptyset$. So the clauses in $\Gamma$ are all of the form $\ell \rightarrow m$.

Define a relation $\succeq$ on the literals in $\Gamma$ by $\ell \succeq m$ iff there is a sequence of literals $\ell = \ell_0, \ldots, \ell_k = m$ ($k \geq 1$) such that $\ell_i \rightarrow \ell_{i+1} \in \Gamma$ for each $i$ ($0 \leq i < k$). Thus, $\succeq$ is a pre-order (reflexive and transitive). Write $\ell \sim m$ if $\ell \succeq m$ and $m \succeq \ell$.

It suffices to prove that $\Gamma$ is satisfiable iff there exists no literal $\ell$ such that $\ell \sim \bar{\ell}$. Note that determining $\ell \succeq m$ is essentially a ‘directed graph search’.
Proof.
It is obvious that \( \Gamma \) is unsatisfiable if there exists a literal \( \ell \) such that \( \ell \sim \bar{\ell} \).

To prove the converse, consider the partial order (reflexive and transitive and anti-symmetric) induced by \( \succeq \) on the equivalence classes of \( \sim \)

Note that if \( \ell \) and \( m \) are in the same equivalence class, then so are \( \bar{\ell} \) and \( \bar{m} \). So equivalence classes come in ‘opposite pairs’.
**Proof.**

Suppose that $\ell$ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class $C$ lowest in the partial order, and make all its literals true (no contradictions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contradictions).

Make all literals false in any $D$ such that $\bar{C}$ is reachable from $D$ in the partial order (no contradictions). Continue until all literals have been given a truth value. Easy to see that $\Gamma$ is satisfied.
Proof.
Suppose that $\ell$ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class $C$ lowest in the partial order, and make all its literals true (no contradictions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contradictions).

Make all literals false in any $D$ such that $\bar{C}$ is reachable from $D$ in the partial order (no contradictions). Continue until all literals have been given a truth value. Easy to see that $\Gamma$ is satisfied.
Proof.

Suppose that $\ell$ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class $C$ lowest in the partial order, and make all its literals true (no contradictions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contradictions).

Make all literals false in any $D$ such that $\bar{C}$ is reachable from $D$ in the partial order (no contradictions). Continue until all literals have been given a truth value. Easy to see that $\Gamma$ is satisfied.
Proof.
Suppose that \( \ell \) is never equivalent to \( \bar{\ell} \).

Start with some (undecided) equivalence class \( C \) lowest in the partial order, and make all its literals true (no contradictions). Make all its literals in the opposite equivalence class, say \( \bar{C} \), false. (no contradictions).

Make all literals false in any \( D \) such that \( \bar{C} \) is reachable from \( D \) in the partial order (no contradictions). Continue until all literals have been given a truth value. Easy to see that \( \Gamma \) is satisfied.
Proof.
Suppose that \( \ell \) is never equivalent to \( \bar{\ell} \).

Start with some (undecided) equivalence class \( C \) lowest in the partial order, and make all its literals true (no contradictions). Make all its literals in the opposite equivalence class, say \( \bar{C} \), false. (no contradictions).

Make all literals false in any \( D \) such that \( \bar{C} \) is reachable from \( D \) in the partial order (no contradictions). Continue until all literals have been given a truth value. Easy to see that \( \Gamma \) is satisfied.
Outline

Propositional logic

Clauses

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
• A quantified Boolean formula is an expression

\[ Q_1 p_1 \cdots Q_m p_m \psi, \]

where \( \psi \) is a formula of propositional logic with proposition letters from some set \( P \), and, for all \( i \) \((1 \leq i \leq m)\), \( p_i \in P \) and \( Q_i \in \{\forall, \exists\} \).

• Examples of quantified Boolean formulas:

\[ \exists p_1 \forall p_2 (p_1 \lor p_2) \]
\[ \forall p_1 \exists p_2 (p_1 \leftrightarrow p_2) \]
\[ \forall p_1 \exists p_2 \forall p_3 (p_3 \rightarrow (p_1 \leftrightarrow \neg p_2)) \]
\[ \forall p_1 \exists p_2 (p_1 \lor p_2) \]
\[ \forall p_1 \exists p_2 (p_1 \land p_2) \]

• Read \( \forall p \) as “for all propositions \( p \)” and \( \exists p \) as “for some propositions \( p \)”.


• The semantics of quantified Boolean formulas is as expected:
  • An assignment is a function $\theta : P \rightarrow \{T, F\}$.
  • Write $\theta \approx \theta'$ if for all $q$ except possibly $q = p$, $\theta(q) = \theta'(q)$.
  • We extend $\theta$ to quantified Boolean formulas by setting
    
    $\theta(\neg \varphi_1) = T$ iff $\varphi_1 = F$
    $\theta(\varphi_1 \lor \varphi_2) = T$ iff $\theta(\varphi_1) = T$ or $\theta(\varphi_2) = T$

    $\ldots$

    $\theta(\exists p. \varphi_1) = T$ iff $\theta'(\varphi_1) = T$ for some $\theta'$ s.t. $\theta' \approx_p \theta$
    $\theta(\forall p. \varphi_1) = T$ iff $\theta'(\varphi_1) = T$ for all $\theta'$ s.t. $\theta' \approx_p \theta$. 
• A quantified Boolean formula with no free variables is called a **quantified Boolean sentence**.

• Notice that quantified Boolean sentences are simply true or false (regardless of any assignment).

• We have the problem

  **QBF-SAT**
  Given: A quantified Boolean sentence $\varphi$.
  Return: Yes if $\varphi$ is true, and No otherwise.
• The following algorithm tests whether a given quantified Boolean sentence \( \varphi \) is true.

• For \( 0 \leq k \leq m \), we take \( \alpha = (v_1, \ldots, v_k) \) to be an assignment of \( v_i \) (either \( T \) or \( F \)) to \( p_i \) (\( 1 \leq i \leq k \)).

begin QBF-test\((Q_1p_1 \ldots Q_mp_m.\psi)\)
    return QBF-rec\((Q_1p_1 \ldots Q_mp_m.\psi, ( ))\)

begin QBF-rec\((Q_1p_1 \ldots Q_mp_m.\psi, \alpha)\)
    if \( |\alpha| = m \)
        return \( \alpha(\psi) \)
    if \( Q_{|\alpha|+1} = \forall \) and QBF-rec\((Q_1p_1 \ldots Q_mp_m.\psi, (\alpha, F)) = F \)
        return \( F \)
    if \( Q_{|\alpha|+1} = \exists \) and QBF-rec\((Q_1p_1 \ldots Q_mp_m.\psi, (\alpha, F)) = T \)
        return \( T \)
    return QBF-rec\((Q_1p_1 \ldots Q_mp_m.\psi, (\alpha, T))\)
• What’s not so clear is how much storage this recursive algorithm takes.

• Consider the tree of assignments for a three-variable sentence
  \( \varphi := Q_1 p_1 Q_2 p_2 Q_3 p_3.\psi \)

• We claim this algorithm runs in polynomial space. This shows that QBF-SAT is in \( \text{PSPACE} \).
• The following non-recursive variant makes this a bit clearer.
• $\alpha$ is the current assignment; $d$ is $+1$ (up) or $-1$ (down).

begin QBF-test-it($Q_1p_1 \ldots Q_mp_m\psi$)  
  $\alpha \leftarrow (F); d \leftarrow -1$
  until $|\alpha| = 0$
    if $|\alpha| = m$
      $v \leftarrow \alpha(\varphi); d \leftarrow +1$
    else if $d = -1$
      $\alpha \leftarrow (\alpha, F)$
    else if last element of $\alpha$ is $F$ and
      (($Q_{|\alpha|} = \forall$ and $v = T$) or ($Q_{|\alpha|} = \exists$ and $v = F$))
      $\alpha \leftarrow (\alpha, T); d \leftarrow -1$
    if $d = +1$
      remove last element of $\alpha$
    return $v$

• The only thing you need to store is $\alpha$, $d$ and $v$. 
Outline

Propositional logic

Clauses

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
• We defined the problems PROPOSITIONAL SAT, and the restrictions SAT, \( k \)-SAT (\( k \geq 1 \)) and HORN-SAT.

• We also defined the extension QBF-SAT of PROPOSITIONAL SAT.

• We showed:
  • KROM-SAT (= 2-SAT) is in \( \text{Co-NLogSpace} \).
  • HORN-SAT is in \( \text{PTime} \).
  • (SAT and) PROPOSITIONAL SAT are in \( \text{NPTime} \).
  • QBF-SAT is in \( \text{PSpace} \).

• Spoiler: later on, we will show that \( \text{NLogSpace} \) and \( \text{Co-NLogSpace} \) are the same! Hence, it will follow:
  • KROM-SAT (= 2-SAT) is in \( \text{NLogSpace} \).

• Reading for this lecture:
  • Sipser, Ch. 8.3 (QBF).
  • Unfortunately, Sipser seems not to cover the material on KROM-SAT.
• We defined the problems PROPOSITIONAL SAT, and the restrictions SAT, $k$-SAT ($k \geq 1$) and HORN-SAT.

• We also defined the extension QBF-SAT of PROPOSITIONAL SAT.

• We showed:
  - KROM-SAT (= 2-SAT) is in \textit{Co-NLogSpace}.
  - HORN-SAT is in \textit{PTime}.
  - (SAT and) PROPOSITIONAL SAT are in \textit{NPTIME}.
  - QBF-SAT is in \textit{PSPACE}.

• Spoiler: later on, we will show that \textit{NLogSpace} and \textit{Co-NLogSpace} are the same! Hence, it will follow:
  - KROM-SAT (= 2-SAT) is in \textit{NLogSpace}.

• Reading for this lecture:
  - Sipser, Ch. 8.3 (QBF).
  - Unfortunately, Sipser seems not to cover the material on KROM-SAT.
Outline

Propositional logic

Clauses

SAT

Special cases of SAT

QBF SAT

Summary

Before you dash off
• 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.