Algorithms for Computational Logic

Introduction

Emmanuel Hebrard (adapted from João Marques Silva, Inês Lynce and Vasco Manquinho)

Outline

1. The Complexity of SAT
2. The Tractability of SAT Fragments
1. **The Complexity of SAT**
   - P and NP
   - Cook-Levin Theorem

2. **The Tractability of SAT Fragments**
   - Tractable Fragments

---

**Cook-Levin Theorem**

SAT is NP-complete

- SAT is “at least as hard” as any problem in NP
  - If there exists a polynomial algorithm for SAT then there exists one for every problem in NP
  - If SAT ∈ P then NP = P
Recall:

**P**
Set of problems that are solved by a polynomial *Turing Machine* (running in $O(n^c)$ time for a constant $c$)

**NP**
Set of problems that are solved by a polynomial *Non-determinist* Turing Machine (running in $O(n^c)$ time for a constant $c$)

**NP-hardness**

**NP-hard problem**
A problem $Q$ is NP-hard if it is “at least as hard as the hardest problem in NP”: if $Q$ can be solved in $O(T)$ time then any problem in NP can be solved in $O(Tn^c)$ time for some constant $c$.

- If an NP-hard problem can be solved in polynomial time, then $P = NP$

**NP-complete problem**
A problem $Q$ is NP-complete if it is NP-hard and is in NP
Turing Machines

- An infinite tape, where we can read/write the symbols 0 and 1 and a head

- A “program”
  - A finite set of states with an initial state \( q_0 \) and a final state \( q_f \).
  - A transition table associating a triplet \( \langle \text{state, symbol, \{←, →\} } \rangle \) to every pair \( \langle \text{state, symbol} \rangle \)

- Meaning: “if reading symbol \( x \) in state \( q \) then write \( x' \), change to state \( q' \) and move right/left”

### Turing Machines, example

<table>
<thead>
<tr>
<th>état</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_f, 0, * )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_2, 0, \to )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_3, 1, \leftarrow )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, 0, \leftarrow )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, 1, \to )</td>
</tr>
</tbody>
</table>
Non-determinist Turing Machines

- A non-determinist Turing Machine can have several transitions in the same configuration
- We assume that it makes the right choice (or explore all possible choices in parallel)
- It is sufficient to have up two transitions for any one configuration

<table>
<thead>
<tr>
<th>état</th>
<th>symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_f, 0, *$</td>
<td>$q_1, 0, \rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2, 0, \rightarrow$ ou $q_4, 1, \leftarrow$</td>
<td>$q_1, 1, \rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3, 1, \leftarrow$</td>
<td>$q_2, 1, \rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_4, 0, \leftarrow$</td>
<td>$q_3, 1, \leftarrow$</td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td>$q_0, 1, \rightarrow$</td>
<td>$q_4, 1, \leftarrow$</td>
<td></td>
</tr>
</tbody>
</table>

The Complexity of SAT

Proof of the Cook-Levin theorem (1)

- Consider a problem $Q$ and a Turing machine that solves it in polynomial time: $O(n^c)$
- This machine executes $O(n^c)$ instructions and therefore requires a tape of length $O(n^c)$
- We build the propositional logic formula with the following variables:
  - A variable $R_{i,t}$ for every cell $i$ of the tape, every symbol $k$ and every time step $t$: true iff the symbol written on cell $i$ at time $t$ is $k$ ($O(1)$ symbols, hence $O(n^c)$ variables)
  - A variable $L_{i,t}$ for every cell $i$ of the tape and every time step $t$: true iff the head is at position $i$ at time $t$ ($O(n^c)$ variables)
  - A variable $Q_{j,t}$ for every state $q_j$ of the program and every time step $t$: true iff the machine is in state $q_j$ at time $t$ ($O(1)$ states, hence $O(n^c)$ variables)
- For a transition $(q_2, 0) \rightarrow (q_3, 1, \leftarrow)$, we add the following clauses, for all $i$ and all $t$:
  - $Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow Q_{3,t+1}$
  - $Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow L_{i-1,t+1}$
  - $Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow R_{i,1,t+1}$
- $O(n^c)$ other clauses
Proof of the Cook-Levin theorem (2)

- Consider a problem \( Q \in \mathbf{P} \)
- \( Q \) admits a Turing machine that runs in \( O(|x|^c) \) time
- For any input \( x \), there exists a Horn Formula \( \phi(Q, x) \) such that:
  - \( \phi(Q, x) \) is satisfiable if and only if \( Q(x) = \text{true} \)
  - \( |\phi(Q, x)| \in O(|x|^c) \)
- An algorithm for Horn-SAT can solve any problem in \( \mathbf{P} \) in polynomial time
  - Not so useful in itself (though Horn-SAT is \( \mathbf{P} \)-complete for log space reductions)

Proof of the Cook-Levin theorem (3)

- Can we come up with a similar encoding for non-deterministic machines?
- There are \( O(1) \) non-deterministic transitions (in the program)
- We add a variable \( X_{i,t} \) for every non-deterministic transition \( i \) and for every time \( t \)
- The transition clauses become:
  - \( X_{i,t} \land Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow Q_{3,t+1} \)
  - \( X_{i,t} \land Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow L_{i-1,t+1} \)
  - \( X_{i,t} \land Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow R_{i,1,t+1} \)
  - \( \neg X_{i,t} \land Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow Q_{3,t+1} \)
  - \( \neg X_{i,t} \land Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow L_{i-1,t+1} \)
  - \( \neg X_{i,t} \land Q_{2,t} \land L_{i,t} \land R_{i,0,t} \Rightarrow R_{i,1,t+1} \)
- They are not Horn anymore
  - Otherwise we would have shown \( \mathbf{P} = \mathbf{NP}! \)
Proof of the Cook-Levin theorem (conclusion)

Preuve

Consider a problem \( Q \in P \)

\( Q \) admits a non-determinist Turing machine that runs in \( O(|x|^{c_1}) \) time

For any input \( x \) there exists a Boolean formula \( \phi(Q, x) \) such that:

\[ \phi(Q, x) \text{ is satisfiable if and only if } x \in \text{true}(Q) \text{ et } |\phi(Q, x)| \in O(|x|^{c_2}) \]

All problems in \( NP \) reduce to \( SAT \)

- If \( SAT \) is in \( P \), then all problems in \( NP \) can be solved in polynomial time and therefore \( P = NP \)
- If \( SAT \) is not in \( P \), then \( P \neq NP \)

Si \( SAT \in P \) alors on peut trouver une interprétation de \( \phi(Q, x) \) en temps polynomial, et donc résoudre \( Q \) en temps polynomial, quel que soit \( Q \in NP \)

Donc \( SAT \in P \) implique \( P = NP \)!
SAT is NP-complete (Cook’s theorem)

3-SAT is hard: Exercise
  - Encoding:
    \[(p_1 \lor p_2 \lor x) \land (\neg x \lor p_3 \lor \ldots \lor p_k) \iff (p_1 \lor p_2 \lor \ldots \lor p_k)\]

2-SAT is easy (Resolution)

Horn-SAT is easy (Unit propagation)

Intermediate Problems

Ladner’s Theorem
If P = NP, then there are problems in NP that are neither in P nor NP-complete.

For instance GraphIsomorphism may be such problem; or Factorisation

What about fragments of SAT?
  - We know some are easy (2-SAT, Horn-SAT), are there others?
  - How do we know which ones are hard and which ones are easy?
  - Are there some in the intermediate class?
**Constraint Satisfaction Problem (CSP)**

**Data:** a triplet \((X, D, C)\) where:
- \(X\) is a ordered set of variables
- \(D\) is a domain
- \(C\) is a set of constraints, where for \(c \in C\):
  - its scope \(S(c)\) is a list of variables
  - its relation \(R(c)\) is a subset of \(D^{\mid S(c)\mid}\)

**Question:** does there exist a solution \(\sigma \in D^{\mid X\mid}\) such that for every \(c \in C\), \(\sigma(S(c)) \in R(c)\)?

---

**Projection**

The projection \(\sigma(X)\) of a tuple \(\sigma\) on a set of variables \(X = (x_{i_1}, \ldots, x_{i_k}) \subseteq X\) as the tuple \((\sigma(x_{i_1}), \ldots, \sigma(x_{i_k}))\)

- Example: the constraint \(x + y = z\) (on the Boolean ring)

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>(z)</th>
<th>(S(x + y = z))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**CNF and Generalized Relations**

- A relation \(R(c)\) over some variables can easily be expressed in clausal form
- Each clause excludes exactly one tuple, example: \(x + y + z \neq 2\)

<table>
<thead>
<tr>
<th>(x + y + z \neq 2)</th>
<th>(x + y + z = 2)</th>
<th>(\iff)</th>
<th>CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x) (y) (z)</td>
<td>(x) (y) (z)</td>
<td>(\iff)</td>
<td>(\land)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- A clause is a particular case of relation on the Boolean domain
We can define fragments of CSP via restrictions on the **domain**, the **structure** or on the **language**

- **Domain**: Boolean CSPs: $D = \{0, 1\}$, Three-valued CSPs, CSP on $\mathbb{Z}$, etc.
- **Structure**: e.g., the incidence graph (bipartite graph variables / constraints) is a *tree* or has a bounded *treewidth*
- **Language**: the library of relations is restricted to a given set $\Gamma$

---

**Language fragment**

CSP($\Gamma$) is the problem of deciding the satisfiability of a CSP whose constraints all have relations in $\Gamma$.

- For instance, Three-valued CSP($\{\neq\}$) is NP-hard since 3-Coloration is NP-hard

---

**Definability**

**pp-definability**

A relation $R$ over $x_1, \ldots, x_k$ on domain $D$ is *(pp-)definable* from a set of relation $\Gamma$ if and only if there exists a CSP $\mathcal{N} = (\mathcal{X}, D, \mathcal{C})$ such that:

- $\{x_1, \ldots, x_k\} \subseteq \mathcal{X}$
- $c \in \mathcal{C} \implies R(c) \in \Gamma \cup \{=\}$
- $R(x_1, \ldots, x_k) \iff (x_1, \ldots, x_k)$ can be extended to a solution of $\mathcal{N}$

- i.e., the relation $R$ can be encoded using relations in $\Gamma$

  - $<$ is definable from $\{\leq, \neq\}$
  - A $k$-clause ($p_1 \lor \ldots \lor p_k$) is definable from 3-clauses
  - All $k$-ary relations are definable from $k$-clauses
Closure

$\ll \Gamma \gg$ is the set of relations that are definable from $\Gamma$

- $\text{CSP}(\Gamma)$ and $\text{CSP}(\ll \Gamma \gg)$ have the same complexity
- Boolean CSPs whose incidence graph is such that constraints vertices have degree 2 (constraints are on at most 2 variables) is in $\mathbf{P}$
  - Any binary relation is definable by binary clauses
  - If $\Gamma$ is the languages composed of 2-clauses, $\{(x \lor y), (\bar{x} \lor y), (\bar{x} \lor \bar{y})\}$, then:
    - $\text{CSP}(\Gamma)$ is 2-\text{-SAT}
    - $\text{CSP}(\ll \Gamma \gg)$ is “Boolean binary CSP”

Schaefer’s Dichotomy Theorem

Boolean CSP($\ll \Gamma \gg$) is in $\mathbf{P}$ if:
- $\Gamma$ are 2-clauses
- $\Gamma$ are Horn-clauses
- $\Gamma$ are dual Horn-clauses
- $\Gamma = \{\oplus\}$ (i.e., XOR. Also known as “Affine-SAT”)
- Every relation in $\Gamma$ accepts the tuple with only 0
- Every relation in $\Gamma$ accepts the tuple with only 1
and is NP-hard otherwise

- Dichotomy: we know the complexity of all the language-based fragments of SAT, and none of them is an intermediate problem