Introduction to Logic in Computer Science: Autumn 2006

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Plan for Today

Now that we have a basic understanding of what complexity classes are and how they relate to each other, we are going to identify some specific decision problems that are complete with respect to interesting complexity classes (mostly **NP**):

- The original **NP**-complete problem: the satisfiability problem for propositional logic
- Variants of the satisfiability problem, some of which are also **NP**-complete, and some of which are not
- **NP**-complete problems in graph theory
- An example for a **PSPACE**-complete problem: satisfiability for quantified boolean formulas

Cook's Theorem

The first decision problem ever to be shown to be **NP**-complete is the satisfiability problem for propositional logic.

SATISFIABILITY (SAT)
Instance: Propositional formula φ
Question: Is φ satisfiable?

The *size* of an instance of SAT is the length of φ. Clearly, SAT can be solved in *exponential* time (by trying all possible models), but no (deterministic) *polynomial* algorithm is known.

Theorem 1 (Cook) SAT *is* **NP***-complete.*

Proof: **NP**-membership is easy to show: if someone guesses a satisfying assignment of truth values to propositional letters (model), then we can check its correctness in polynomial time. ✓

The proof of **NP**-hardness is sketched on the following slides ...

Proof of NP-Hardness (1)

We need to show that *any* problem in **NP** can be reduced to SAT. By definition, for any problem in **NP** there is a nondeterministic algorithm that accepts or rejects all instances of the problem in polynomial time. We now sketch how to encode this algorithm as a propositional formula.

Suppose the algorithm runs in time $\leq n^k$. So it will use at most n^k memory cells. Hence, there *exists* (nondeteminism!) a table of size $n^k \times n^k$ for each possible run of the algorithm, with each table cell holding an element belonging to some (small) alphabet.

Introduce a propositional variable x_{ijs} for each row i, column j, symbol s, saying whether or not cell (i, j) is holding symbol s. The number of these variables is polynomial in n.

If we can fully specify all constraints these x_{ijk} have to meet to represent a well-formed computation table for an accepting run as a formula (of polynomial size), then we are done, as that formula will be satisfiable iff there is an accepting run.

Proof of NP-Hardness (2)

Follows Sipser (1997). Each table cell must hold exactly one symbol:

$$\varphi_{cell} \quad = \quad \bigwedge_{1 \leq i,j \leq n^k} \left[\bigvee_{s \in S} x_{ijs} \ \wedge \ \bigwedge_{s,s' \in S} \neg(x_{ijs} \wedge x_{ijs'}) \right]$$

We can also encode the initial configuration as a conjunction φ_{start}, and the characteristics of an accepting configuration as a formula φ_{accept}.

To encode what is a legal move from one configuration to the next, we need to be more specific about our machine model. As any algorithm can be implemented on a Turing machine, we may assume that the contents of memory cell (i, j) will only depend on the content of its predecessor and that cells neighbours: $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$.

We can write a formula $\varphi_{move(i,j)}$ specifying all possible moves of this sort for (i, j). Then $\varphi_{move} = \bigwedge_{1 \leq i,j \leq n^k} \varphi_{move(i,j)}$ describes *all* legal moves. Finally, $\varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{accept} \wedge \varphi_{move}$ encodes the algorithm. ✓

M. Sipser. *Introduction to the Theory of Computation*. Thomson, 1997.

Variants of Satisfiability

If we restrict the structure of propositional formulas, then there's a chance that the satisfiability problem will become easier.

k-SATISFIABILITY (kSAT)
Instance: Conjunction φ of disjunctions of k literals each
Question: Is φ satisfiable?

Alternative formulation: Check whether a given set S of k-clauses is satisfiable.

By close inspection of the proof for SAT, it is possible to show (see book for details):

Theorem 2 3SAT *is* **NP**-*complete.*

Only once we go down from 3 to 2, we get a positive result ...

2SAT is in P

Theorem 3 2SAT *is in* **P**.

Proof sketch: Recall that REACHABILITY $\in$ **P**. We are going to reduce 2SAT to REACHABILITY.

Let φ be a formula in CNF with clauses of length 2 and let P be the set of propositional variables in φ.

Build a graph $G = (V, E)$ with $V = P \cup \{\neg p \mid p \in P\}$ and $(x, y) \in E$ iff a there is a clause in φ that is equivalent to $x \to y$.

Observe that φ is *unsatisfiable* iff there is a $p \in P$ such that there is both a path from p to $\neg p$ and from $\neg p$ to p in G.

This condition can be tested by running our algorithm for REACHABILITY several times. ✓

Counting Clauses

If not all clauses of a given formula in CNF can be satisfied simultaneously, what is the maximum number of clauses that can?

> MAXIMUM k-SATISFIABILITY (MAXkSAT)
> **Instance:** Set S of k-clauses and $K \in \mathbb{N}$
> **Question:** Is there a satisfiable $S' \subseteq S$ such that $|S'| \geq K$?

For this kind of problem, we cross the border between **P** and **NP** already for $k = 2$ (rather than $k = 3$, as before):

Theorem 4 MAX2SAT *is* **NP***-complete.*

Proof sketch: MAX2SAT is clearly in **NP**: if someone guesses an $S' \subseteq S$ and a model with $|S'| \geq K$, we can check whether S' is true in that model in polynomial time. ✓

Next we show **NP**-hardness by reducing 3SAT to MAX2SAT ...

Reduction from 3SAT to MAX2SAT

Consider the following 10 clauses:

$$(x),\ (y),\ (z),\ (w),$$
$$(\neg x \vee \neg y),\ (\neg y \vee \neg z),\ (\neg z \vee \neg x),$$
$$(x \vee \neg w),\ (y \vee \neg w),\ (z \vee \neg w)$$

Observe: any model satisfying $(x \vee y \vee z)$ can be extended to satisfy (at most) 7 of them; all other models satisfy at most 6 of them.

Given an instance of 3SAT, construct an instance of MAX2SAT: For each clause $C_i = (x_i \vee y_i \vee z_i)$ in φ, write down these 10 clauses with a new w_i. If the input has n clauses, set $K = 7n$.

Then φ is satisfiable iff (at least) K of the 2-clauses in the new problem are satisfiable. ✓

Satisfiability for Horn Clauses

Another restriction of the satisfiability problem:

> HORN SATISFIABILITY (HORNSAT)
> **Instance:** Conjunction φ of Horn clauses
> **Question:** is φ satisfiable?

Recall that Prolog is built around Horn clauses.

Theorem 5 HORNSAT *is in* **P**.

Proof sketch: The following (polynomial) algorithm does the job. Distinguish negative Horn clauses: $(\neg x_1 \vee \cdots \vee \neg x_k)$; and implications (including atoms: $x_1 = \mathit{true}$): $(x_1 \wedge \cdots \wedge x_\ell \rightarrow y)$.

Initially, make all variables *false*. Iterate: if $(x_1 \wedge \cdots \wedge x_\ell \rightarrow y)$ not satisfied, then make y true $\rightsquigarrow$ *minimal* model M for implications.

Full formula can only be satisfiable if M satisfies all negative clauses as well. For if not, we'd get $\{x_1, \ldots, x_k\} \subseteq M$ and we'd have to make one of the x_i *false*, thereby contradicting minimality. ✓

Independent Sets

Many conceptually simple problems that are **NP**-complete can be formulated as problems in graph theory. Example:

Let $G = (V, E)$ be an *undirected* graph. An *independent set* is a set $I \subseteq V$ such that there are no edges between any of the vertices in I.

INDEPENDENT SET
Instance: Undirected graph $G = (V, E)$ and $K \in \mathbb{N}$
Question: Does G have an independent set I with $|I| \geq K$?

Theorem 6 INDEPENDENT SET *is* **NP***-complete.*

Proof sketch: **NP**-membership: easy ✓
NP-hardness: by reduction from 3SAT with n clauses —
Given a conjunction φ of 3-clauses, construct a graph $G = (V, E)$.
V is the set of occurrences of literals in φ. Edges: make a "triangle" for each 3-clause, and connect complementary literals. Set $K = n$.
Then φ is satisfiable iff there is an independent set of size K. ✓

More Graph-theoretic Problems

All of the following are also **NP**-complete problems. They each take an undirected graph $G = (V, E)$ and an integer K as input.

- CLIQUE: Is there a $V' \subseteq V$ with $|V'| \geq K$ such that any two vertices in V' are joined by an edge in E?
- VERTEX COVER: Is there a $V' \subseteq V$ with $|V'| \leq K$ such that for any edge $(x, y) \in E$ either $x \in V'$ or $y \in V'$?
- HAMILTON PATH: Does G have a Hamilton path (that is, a path visiting every vertex)? (no K needed as input)
- TRAVELLING SALESMAN PROBLEM: Is there a path $\leq K$ visiting each vertex once? (additional input: distances)
- MAXCUT: Is there a $V' \subseteq V$ such that the number of edges between nodes in V' and in $V \setminus V'$ exceeds K?

Quantified Boolean Formulas

The canonical example for a **PSPACE**-complete problem is the satisfiability problem for *quantified boolean formulas*.

A quantified boolean formula (QBF) is a propositional formula φ preceded by either $\forall x$ or $\exists x$ for each propositional variable in φ:

$$\forall x.\exists y.\forall z.[(x \to y) \vee z]$$

The semantics is exactly what you would expect it to be ...

This gives rise to a new decision problem:

> Quantified Satisfiability (QSat)
> **Instance:** Quantified boolean formula φ
> **Question:** Is φ satisfiable (true)?

The size of a QSat instance is the length of φ.

PSPACE-Completeness

Theorem 7 QSAT *is* **PSPACE***-complete.*

Proof idea: **PSPACE**-membership: Recursively go through the formula in the obvious manner. The recursion depth is given by the number of propositional variables, and at each state we have to remember what "satisfiability requirements" we still need to meet (polynomial space). ✓

PSPACE-hardness: We need to chow that any problem in **PSPACE** can be encoded as a QBF. Use the same idea as for Cook's Theorem: encode the computation table as a formula. Problem: the number of rows in the table might be exponential. Solution: Use the idea from the proof of Savitch's Theorem and write a QBF expressing that you can go from configuration c_1 to configuration c_2 in t steps if there's another configuration c such that you can go from c_1 to c and from c to c_2 in $\lceil \frac{t}{2} \rceil$ steps each ...

Remark

The satisfiability and the validity problem coincide for QBFs. This matches up nicely with **PSPACE** = **coPSPACE**.

For comparison, in propositional logic these are distinct problems and we don't know whether **NP** $=^{?}$ **coNP**.

References

Again, all of today's material can be found in Papadimitriou's book.

- Cook's Theorem is proved in Chapter 8, and further **NP**-completeness results are established in Chapter 9.
- The **PSPACE**-completeness proof for quantified boolean formulas may be found in Chapter 19.

For large collections of **NP**-complete problems, the books by Garey and Johnson (1979) and Ausiello *et al.* (1999) are useful references.

C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag, 1999.

Summary

We have seen a range of **NP**-complete problems:

- Logic: SAT, 3SAT, MAX2SAT, but *not* 2SAT and HORNSAT
- Graph Theory: INDEPENDENT SET and others

Recall that the **P-NP** borderline is widely considered to represent the move from tractable to intractable problems, so developing a feel for what sort of problems are **NP**-complete is important to understand what can and what cannot be computed in practice.

In logic, in particular, we also encounter problems that are a lot harder: the **PSPACE**-completeness result for QSAT is an example.