Decision Problems

The **decision problem** $D_{\mathcal{L}}$ for a formal language $\mathcal{L} \subseteq \Sigma^*$ is the computational task:

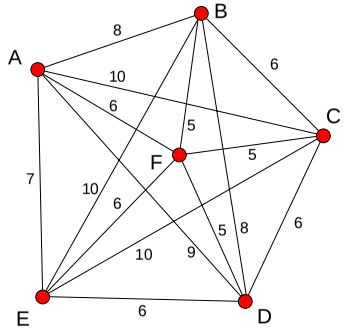
> Given an arbitrary string $I \in \Sigma^*$, to determine whether or not $I \in \mathcal{L}$.

The input string I is called an **instance** of the problem $D_{\mathcal{L}}$.

It is a **positive** or "yes" instance if $I \in \mathcal{L}$, otherwise it is a **negative** or "no" instance.

Any computational decision problem can be represented as the decision problem for some formal language $\mathcal{L}$.

Travelling Salesman Problem



Is there a round tour with distance ≤ 37?

A string to represent this instance of TSP:

a,b,8;a,c,10;a,d,9;a,e,7;a,f,6;b,c,6;b,d,8;b,e,10;
b,f,5;c,d,6;c,e,10;c,f,5;d,e,6;d,f,5;e,f,6;37.

NP decision problems

A decision problem is **non-deterministically polynomial** (NP) if there is a language C and a relation $cert \in C \times \Sigma^*$ such that, for each $I \in \Sigma^*$,

1. I is a positive instance of the problem if and only if there exists $c \in C$ such that $cert(c, I)$, and

2. for any $c \in C$, it can be determined whether or not $cert(c, I)$ in time bounded by a polynomial function of the length of I.

For $I \in \mathcal{L}$, a element $c \in C$ such that $cert(c, I)$ is called a **certificate** for I. It certifies that I is a positive instance of the problem. It can be checked to be a certificate in polynomial time.

Propositional Clauses

A **clause** in the Propositional Calculus is a formula written in **disjunctive form** as

$$\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_m \vee B_1 \vee B_2 \vee \cdots \vee B_n,$$

where $m \geq 0, n \geq 0$, and each A_i and B_j is a single schematic letter.

Schematic letters and their negations are collectively known as **literals**; a clause is a *disjunction of literals*.

The clause above can also be written in **conditional form** as

$$A_1 \wedge A_2 \wedge \ldots A_m \rightarrow B_1 \vee B_2 \vee \cdots \vee B_n.$$

If $m = 0$ this is $B_1 \vee B_2 \vee \cdots \vee B_n$, while if $n = 0$ it may be written as $\neg(A_1 \wedge A_2 \wedge \ldots A_m)$.

Any propositional formula is equivalent to the conjunction of one or more clauses.

The Satisfiability Problem

The decision problem SAT is:

Given a finite set $\{C_1, C_2, \ldots, C_n\}$ of clauses, determine whether there is an assignment of truth-values to the schematic letters appearing in the clauses which makes all the clauses true.

Example: is the following set satisfiable?

$$\{A \vee B,\ A \rightarrow C \vee D,\ \neg(B \wedge C \wedge E),\ C \rightarrow B \vee E,\ \neg(A \wedge E)\}.$$

The size of an instance of SAT is the total number of schematic letters appearing in all the clauses (13 in this example).

SAT is NP: given a candidate certificate, e.g., 'A and D true, the rest false', we can check whether it really is a certificate in *linear* time.

Problem Reduction

Decision problem D_1 can be **reduced** to decision problem D_2 if there is an algorithm which takes as input an arbitrary instance I_1 of D_1 and delivers as output an instance I_2 of D_2 such that I_2 is a positive instance of D_2 if and only if I_1 is a positive instance of D_1.

If D_1 can be reduced to D_2, and we have an algorithm which solves D_2, then we thereby have an algorithm which solves D_1.

In this case, the computational complexity of D_1 is at most the sum of the computational complexities of D_2 and the reduction algorithm. If the reduction algorithm has polynomial complexity, then D_1 is **at most polynomially harder** than D_2.

Cook's Theorem

Any NP problem can be reduced to SAT in polynomial time.

This was proved by Stephen Cook in 1971, by a method analogous to Turing's proof that the *Entscheidungsproblem* was equivalent to the Halting Problem.

Cook encoded as propositional clauses the relevant facts about both the problem instance and the Turing machine which checks candidate certificates.

The resulting set of clauses is satisfiable if and only if the original problem instance is positive.

This reduces the problem of deciding the latter to the problem of deciding the former.

Preparing for the Proof

Given any NP decision problem D, there is a Turing machine M which can determine in at most $P(n)$ steps whether $cert(c, I)$, where I is an instance of D of size n, c is a candidate certificate, and P is a polynomial function.

1. M has states $0, 1, 2, \ldots, q-1$, and tape alphabet $a_1, a_2, \ldots, a_s$.

2. The operation of M is given by functions T, U, and D.

3. Initially, I is inscribed on squares $1, 2, \ldots, n$, and c on squares $-m, \ldots, -2, -1$.

4. M halts scanning square 0, which then contains symbol a_1 if and only if $cert(c, I)$.

Note that $m \leq P(n)$. This is because the computation is completed in at most $P(n)$ steps; during this, the head cannot move more than $P(n)$ steps left of its starting point.

The propositional atoms

We define atomic propositions with intended interpretations as follows:

1. For $i = 0,\ldots,P(n)$ and $j = 0,\ldots,q-1$, proposition Q_{ij} says that after i steps, M is in state j.

2. For $i = 0,\ldots,P(n)$, $j = -P(n),\ldots,P(n)$, and $k = 1,2,\ldots,s$, proposition S_{ijk} says that after i steps, square j of the tape contains symbol a_k.

3. $i = 0,\ldots,P(n)$ and $j = -P(n),\ldots,P(n)$, proposition T_{ij} says that after i steps, M is scanning square j of the tape.

The clauses I

1. *At each step, M is in at least one state.*
For $i = 0,\ldots,P(n)$,

$$\boxed{Q_{i0} \vee Q_{i1} \vee \cdots \vee Q_{i(q-1)}}$$

[altogether $O(P(n))$ literals]

2. *At each step, M is in at most one state.*
For $i = 0,\ldots,P(n)$ and $0 \le j < k \le q-1$,

$$\boxed{\neg(Q_{ij} \wedge Q_{ik})}$$

[altogether $O(P(n))$ literals]

3. *At each step, each tape square contains at least one alphabet symbol.*
For $i = 0,\ldots,P(n)$ and $-P(n) \le j \le P(n)$,

$$\boxed{S_{ij1} \vee S_{ij2} \vee \cdots \vee S_{ijs}}$$

[altogether $O(P(n)^2)$ literals]

The clauses II

4. *At each step, each tape square contains at most one alphabet symbol.*
For $i = 0,\ldots,P(n)$, $-P(n) \le j \le P(n)$, and $1 \le k < l \le s$,

$$\boxed{\neg(S_{ijk} \wedge S_{ijl})}$$

[altogether $O(P(n)^2)$ literals]

5. *At each step, the tape is scanning at least one square.*
For $i = 0,\ldots,P(n)$,

$$\boxed{T_{i(-P(n))} \vee T_{i(1-P(n))} \vee \cdots \vee T_{i(P(n)-1)} \vee T_{iP(n)}}$$

[altogether $O(P(n)^2)$ literals]

6. *At each step, the tape is scanning at most one square.*
For $i = 0,\ldots,P(n)$, $-P(n) \le j < k \le P(n)$,

$$\boxed{\neg(T_{ij} \wedge T_{ik})}$$

[altogether $O(P(n)^3)$ literals]

The clauses III

7. *Initially, the machine is in state 1 scanning square 1.*

$$\boxed{Q_{01},\ T_{01}}$$

[Two literals]

8. *The configuration at each step after the first is determined from the configuration at the previous step by the functions T, U, and D defining the machine M.*
For $i = 0,\ldots,P(n)$, $-P(n) \le j \le P(n)$, $k = 0,\ldots,q-1$, and $l = 1,\ldots,s$,

$$\boxed{\begin{array}{rcl} T_{ij} \wedge Q_{ik} \wedge S_{ijl} & \rightarrow & Q_{(i+1)T(k,l)} \\ T_{ij} \wedge Q_{ik} \wedge S_{ijl} & \rightarrow & S_{(i+1)jU(k,l)} \\ T_{ij} \wedge Q_{ik} \wedge S_{ijl} & \rightarrow & T_{(i+1)(j+D(k,l))} \\ S_{ijk} & \rightarrow & T_{ij} \vee S_{(i+1)jk} \end{array}}$$

[altogether $O(P(n)^2)$ literals]

The clauses IV

9. *Initially, the string $a_{i_1}a_{i_2}\ldots a_{i_n}$ defining the problem instance I is inscribed on squares $1,2,\ldots,n$ of the tape.*

$$\boxed{S_{01i_1},\ S_{02i_2},\ \ldots,\ S_{0ni_n}}$$

[altogether n literals]

10. *By the $P(n)$th step, the machine has reached the halt state, and is then scanning square 0, which contains the symbol a_1.*

$$\boxed{Q_{P(n)0},\ S_{P(n)01},\ T_{P(n)0}}$$

[3 literals]

The total number of literals used is $O(P(n))^3$. Thus the reduction can be accomplished in polynomial time.

D is reduced to SAT

If I is a positive instance of D then there is a certificate c such that M, when run with inputs c, I, halts scanning symbol a_1 on square 0. Hence some sequence of symbols, placed initially on squares $-P(n),\ldots,-1$, will allow the clauses above to be satisfied.

Conversely, if I is a negative instance of D, there is *no* certificate for I, so *whatever* symbols we place on squares $-P(n),\ldots,-1$, when M halts it is not scanning a_1 on square 0, so the set of clauses above are not satisfiable.

From instance I of problem D we have constructed, in polynomial time, a set of clauses which constitute a positive instance of SAT if and only I is a positive instance of D. Thus, we have reduced D to SAT in polynomial time.

Since D was an arbitrary NP problem it follows that *any* NP problem can be converted to SAT in polynomial time.

NP-completeness

Cook's Theorem implies that *any NP problem is at most polynomially harder than SAT*. If we can solve SAT in polynomial time, we can solve *any* NP problem in polynomial time.

A decision problem is **NP-complete** if it is NP, and any NP problem can be reduced to it in polynomial time. Thus SAT is NP-complete.

To prove that an NP problem is NP-complete, it suffices to show that SAT can be reduced to it in polynomial time.

Suppose SAT can be reduced to problem D in polynomial time. Any NP problem D' can be reduced into SAT in polynomial time, and SAT can be reduced into D in polynomial time. The result is a polynomial-time reduction of D' into D. Since D' was an arbitrary NP problem, it follows that D is NP-complete.

Reduction of SAT to 3SAT

3SAT is similar to SAT, but restricts the clauses to at most three schematic letters each.

To reduce an instance of SAT to an instance of 3SAT, replace each clause $C \equiv L_1 \vee L_2 \vee \ldots \vee L_n$, where $n > 3$, by $n-2$ new clauses, using $n-3$ new schematic letters $X_1,\ldots,X_{n-3}$, as follows:

$$\begin{array}{l} L_1 \vee L_2 \vee X_1 \\ X_1 \rightarrow L_3 \vee X_2 \\ X_2 \rightarrow L_4 \vee X_3 \\ \qquad\vdots \\ X_{n-4} \rightarrow L_{n-2} \vee X_{n-3} \\ X_{n-3} \rightarrow L_{n-1} \vee L_n \end{array}$$

Call the new set of clauses $\mathcal{C}$.

We shall show that any truth-assignment to the schematic letters appearing in the L_i which satisfies C can be extended to the X_i so that $\mathcal{C}$ is satisfied, and conversely any truth assignment which satisfies $\mathcal{C}$ also satisfies C.

Extending the truth assignment

A truth-assignment satisfying C must satisfy at least one literal appearing in C, say L_k.

Now assign $true$ to $X_1, X_2, \ldots, X_{k-2}$ and $false$ to $X_{k-1}, \ldots, X_{n-3}$.

Then all the clauses in $\mathcal{C}$ are satisfied:
for $i = 1, 2, \ldots, k-2$, the ith clause is satisfied because X_i is true;
the $(k-1)$th clause is satisfied because L_k is true;
for $j = k, k+1, \ldots, n-2$ the jth clause is satisfied because X_{j-1} is false.

Conversely, suppose each clause in $\mathcal{C}$ is satisfied. If $L_1, \ldots, L_{n-2}$ are all false, all the X_i must be true; in particular X_{n-3} is true, so either L_{n-1} or L_n is true. Thus at least one L_i is true, and hence C is true.

P=NP?

A problem is NP-complete if and only if it is NP and any NP problem can be reduced to it in polynomial time.

All NP-complete problems are interconvertible in polynomial time. They agree in complexity to within a polynomial amount of difference.

If any NP-complete problem has polynomial complexity, they all do. If any of them cannot be solved in polynomial time, then none can. They stand or fall together.

No faster-than-exponential algorithm is known for any NP-complete problem. But no-one has proved that such algorithms do not exist.

If we could solve NP-complete problems in polynomial time, then the class NP would collapse into the class P of problems solvable in polynomial time, i.e., P=NP. We do not know whether this is the case.