# Berman's theorem

*ho boon suan*

*june 14, 2022*

> I learned this proof of Berman's theorem from Yuval Filmus: `https://math.stackexchange.com/a/1416150/`. Here I'm just writing it up to my satisfaction, as usual.

**Theorem** (Piotr Berman, 1978)**.** *If there exists an **NP**-hard unary language $U \subset \{1\}^*$, then $\mathbf{P} = \mathbf{NP}$.*

*Proof.* Using the **NP**-hard unary language $U$, we define a polynomial-time algorithm for SAT that makes use of downward self-reducibility. The idea is that, since there exists a polynomial-time reduction $f \colon \mathsf{SAT} \leq_p U$ such that $\varphi \in \mathsf{SAT}$ if and only if $f(\varphi) \in U$, we have $f(|\varphi|) \leq |\varphi|^c$ for some constant $c$. Since $U$ is unary and therefore sparse (that is, it has at most one string of any length), it follows that there are only $m^c$ possible values for $f(\psi)$ whenever $\psi$ is a satisfiable CNF of size at most $m$.

We now describe our algorithm for SAT. Given a CNF $\varphi$ of size $m$ on variables $x_1, \ldots, x_n$, we initialize a set with the single element $(\varphi, f(\varphi))$. If $f(\varphi)$ is not of the form $1^j$, we immediately reject $\varphi$. Our algorithm will maintain the following invariants: *after each step, the set has at most $m^c$ elements, and the original formula $\varphi$ is satisfiable if and only if our set contains some element $(\psi, f(\psi))$ for which $\psi$ is satisfiable.* For each element $(\psi, f(\psi))$ in the set, we replace it with two new elements $(\psi|_{x_1 = \top}, f(\psi|_{x_1 = \top}))$ and $(\psi|_{x_1 = \bot}, f(\psi|_{x_1 = \bot}))$. Finally, we remove all elements whose second entry is not of the form $1^j$, and if there are multiple elements who both have second entry $1^j$, we keep only one of them. Thus, after each step, we see that there are at most $m^c$ elements in our set. After $n$ steps, the formula $\varphi$ will be satisfiable if and only if our set contains the element $(\top, f(\top))$. Since the reduction $f$ is polynomial-time and each of the $n$ steps makes at most $2m^c$ calls to $f$, we conclude that our algorithm for SAT runs in polynomial-time, which implies that $\mathbf{P} = \mathbf{NP}$ as needed. $\qquad\square$

**Some thoughts.** Many statements of this result require the unary language in question to be *NP-complete*, but we only need it to be **NP**-hard. In fact, we only need a polynomial-time computable function $f \colon \{0, 1\}^* \to \{0, 1\}^*$ such that $f(\psi) \in \{1\}^*$ whenever $\psi$ is a satisfiable CNF.

*References*

Berman, Piotr. *Relationship Between Density and Deterministic Complexity of **NP**-complete Languages*. Fifth International Colloquium on Automata, Languages and Programming, Italy (July 1978), Springer-Verlag Lecture Notes in Computer Science **62**, 63–71.
  `doi:10.1007/3-540-08860-1_6`