A proof that HORNSAT is P-complete

A clause $l_1 \vee l_2 \vee \cdots \vee c_n$, where $l_1, \ldots, l_n$ are literals, is a *Horn clause* if it has at most one positive literal (page 78 in Papadimitriou's book). Furthermore, we know that the problem HORNSAT asking whether a conjunctive normal form (CNF) Boolean expression consisting only of Horn clauses is satisfiable is in **P** (page 79).

A language L in a complexity class $\mathcal{C}$ is $\mathcal{C}$-complete (under log-space reductions) if any language $L' \in \mathcal{C}$ can be reduced (in log-space) to L. And, if L is $\mathcal{C}$-complete and L reduces to a language $L' \in \mathcal{C}$, then L' is also $\mathcal{C}$-complete.

Theorem 1 HORNSAT *is* **P***-complete under log-space reductions.*

Proof. First, as mentioned above, HORNSAT is in **P**. We now reduce MONOTONE CIRCUIT VALUE, which we know to be **P**-complete (page 171 in the book), to HORNSAT and in this way show that HORNSAT is **P**-complete.

Let $C = (V = \{v_1, \ldots, v_n\}, E)$ be a monotone variable-free Boolean circuit (an instance for MONOTONE CIRCUIT VALUE). Thus the gates of C are of sort $\wedge$, $\vee$, **true** or **false** and the answer to MONOTONE CIRCUIT VALUE is "yes" iff the output gate of the circuit C evaluates to **true**. We now construct a conjunction $f(C)$ of Horn clauses (i.e. a Boolean expression $f(C)$ in Horn CNF) such that $f(C)$ is satisfiable iff the output gate of the circuit C evaluates to **true**. We first convert C into a Horn CNF Boolean expression $\hat{f}(C) = \bigwedge_{v \in V} C_v$ over variables $\hat{v}_1, \ldots, \hat{v}_n$ corresponding to the gates $v_1, \ldots, v_n$ of C. The intuitive meaning of a variable $\hat{v}_i$ is that it denotes that a gate v_i evaluates to **false** in C. $\hat{f}(C)$ is defined as follows:

- If the sort of the gate v, denoted by $s(v)$, is $\vee$ and v', v'' are the children of v, then C_v is $(\hat{v}' \wedge \hat{v}'') \Rightarrow \hat{v}$ which equals to the Horn clause $\neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}$. That is, if both of children evaluate to false, then so should the gate itself.
- If $s(v) = \wedge$ and v', v'' are the children of v, then C_v is $(\hat{v}' \Rightarrow \hat{v}) \wedge (\hat{v}'' \Rightarrow \hat{v})$ which is equal to $(\neg\hat{v}' \vee \hat{v}) \wedge (\neg\hat{v}'' \vee \hat{v})$. The idea is that if either of the children is false, then the gate should also be.
- if $s(v) =$ **true**, then C_v is $\neg\hat{v}$ and
- if $s(v) =$ **false**, then C_v is $\hat{v}$.

Note that the construction of $\hat{f}(C)$ is NOT exact in the sense that, for instance, if v is an $\wedge$-gate with children v' and v'' assigned to true, then the variable $\hat{v}$ corresponding to the gate v can have either of the values true or false in a satisfying truth assignment for $\hat{f}(C)$ since $\hat{v}'$ and $\hat{v}''$ are assigned to false. BUT, if $\hat{v}'$ and $\hat{v}''$ assigned to be true (gates v' and v'' evaluate to false), then $\hat{v}$ MUST be true (v evaluates to false). We formalize this in the following.

Lemma 2 *If the output gate v_1 of C evaluates to* **true** *in C, then and only then is there a satisfying truth assignment for $\hat{f}(C)$ assigning the variable $\hat{v}_1$ to* **false***.*

The lemma in turn is established by the following claim:

Claim 3 *Let C' be a sub-circuit of C such that the input gates belong to C' and if v is a gate in C', then the children of v also belong to C'. Then a gate v evaluates to* **true** *in C' iff there is a satisfying truth assignment for $\hat{f}(C')$ such that $\hat{v}$ is assigned to* **false***.*

Proof. By induction on the structure of C.

Induction base: For C' consisting only of input gates, the claim clearly holds.

Induction hypothesis: Let C' be a sub-circuit of C such that the lemma holds.

Induction step: Consider a gate v of C not in C' such that the children of v, v' and v'', belong to C'. Let C'' be the sub-circuit otherwise similar to C' but augmented with v and let $\hat{f}(C'')$ be the corresponding Horn CNF formula. Now there are two cases depending on the sort of v:

1. Assume that v is a $\vee$-gate.

   If v evaluates to **true** in C'', then one or both of v' and v'' evaluate **true** in C''. By the induction hypothesis, there is a satisfying truth assignment for $\hat{f}(C')$ such that $\hat{v}'$ or/and $\hat{v}''$ is assigned to **false**. But then $\neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}$ is satisfied and $\hat{v}$ can have any value, even **false**, in the same truth assignment extended to $\hat{f}(C'')$.

   On the other hand, if there is a satisfying truth assignment for $\hat{f}(C'')$ such that $\hat{v}$ evaluates to **false**, then, in order to $\neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}$ to be satisfied, $\hat{v}'$ or/and $\hat{v}''$ must be assigned to **false**. However, the same truth assignment restricted to all clauses except $\neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}$ is a satisfying truth assignment for $\hat{f}(C')$ and then, by the induction hypothesis, v' or/and v'' evaluates to **true** in C' and therefore v must evaluate to **true** in C''.

2. Assume that v is a $\wedge$-gate.

   If v evaluates to **true** in C'', then both v' and v'' evaluate **true** in C''. By the induction hypothesis, there is a satisfying truth assignment for $\hat{f}(C')$ such that $\hat{v}'$ and $\hat{v}''$ is assigned to **false**. But then $\neg\hat{v}' \vee \hat{v}$ and $\neg\hat{v}'' \vee \hat{v}$ are both satisfied and $\hat{v}$ can have any value, even **false**, in the same truth assignment extended to $\hat{f}(C'')$.

   On the other hand, if there is a satisfying truth assignment for $\hat{f}(C'')$ such that $\hat{v}$ evaluates to **false**, then, in order to $\neg\hat{v}' \vee \hat{v}$ and $\neg\hat{v}'' \vee \hat{v}$ to be satisfied, $\hat{v}'$ and $\hat{v}''$ must be assigned to **false**. However, the same truth assignment restricted to all clauses except $\neg\hat{v}' \vee \hat{v}$ and $\neg\hat{v}'' \vee \hat{v}$ is a satisfying truth assignment for $\hat{f}(C')$ and then, by the induction hypothesis, v' and v'' evaluate to **true** in C' and therefore v must evaluate to **true** in C''.

□

Now we form $f(C)$ from $\hat{f}(C)$ by adding an extra clause $\neg v_1$ for the output gate v_1 of C. It is a clear consequence of the lemma above that the output gate v_1 of C evaluates to **true** iff $f(C)$ has a satisfying truth assignment. We have therefore mapped an object C to an object $f(C)$ such that C is in MONOTONE CIRCUIT VALUE iff $f(C)$ is in HORNSAT.

It remains to show that this reduction from a circuit C to a Horn CNF formula $f(C)$ can be done by using space $\mathcal{O}(\log n)$. Assume that the input format for C is of form $g_1; g_2; \ldots; g_n;$, where each g_i is either $vi = vj \wedge vk$, $vi = vj \wedge vk$, $vi = \top$ (for **true** gates) or $vi = \bot$ (for **false** gates), where $1 \leq i, j, k \leq n = |V|$ are coded in binary by using symbols 0 and 1. Now the reduction can be carried out by using three "registers" i, j, k, each of which is of logarithmic size w.r.t. the representation of C and a constant size register for the sort of the current gate definition g_i scanned. The reduction machine scans the g_is of the input one at the time, saving the indices i, j, k and the sort of the gate ($\wedge, \vee,$ **true** or **false**) to the corresponding registers. It then writes the above mentioned (constant size) Horn clauses corresponding to the scanned g_i into the output tape. Thus the reduction can clearly be performed in logarithmic space. □

8.4.8

Let Σ be a fixed alphabet. A *coding* κ is defined to be a mapping from Σ to Σ (κ is not necessarily one-to-one). For a string $x = x_1 x_2 \dots x_n \in \Sigma^*$, define $\kappa(x) = \kappa(x_1)\kappa(x_2)\dots\kappa(x_n)$. If $L \subseteq \Sigma^*$ is a language, define $\kappa(L) = \{\kappa(x) \mid x \in L\}$. Without loss of generality we also assume that the Turing machine specific symbols $\triangleright$ and $\sqcup$ are mapped to themselves and no other symbol is mapped to them.

(a)

The goal is to show that **NP** is closed under codings, that is, if $L \in \mathbf{NP}$ is a language and κ is a coding, then $\kappa(L) \in \mathbf{NP}$.

A Simple Proof

Let $\kappa : \Sigma \to \Sigma$ be a coding and $L \in \mathbf{NP}$ a language. Now let $M = \langle K, \Sigma, \Delta, s)\rangle$ be a non-deterministic Turing machine deciding L in polynomial time (since $L \in \mathbf{NP}$, one must exist). We now construct a non-deterministic Turing machine M' deciding $\kappa(L)$ as follows. Given an input $x' = x'_1 x'_2 \dots x'_n$, M' first non-deterministically guesses a string $x = x_1 x_2 \dots x_n$ such that $\kappa(x) = x'$ (remember that the coding is given and thus known to M'). M' then executes M on x as a subroutine and says "yes" iff M says so. Clearly, if there is a string $x \in L$ such that $\kappa(x) = x'$, then and only then M' accepts x'. Therefore M' accepts x' iff $x' \in \kappa(L)$. Since M' runs in(non-deterministic) polynomial time, $\kappa(L) \in \mathbf{NP}$. Thus **NP** is closed under codings.

A More Complicated Proof

Let $\kappa : \Sigma \to \Sigma$ be a coding and $L \in \mathbf{NP}$ a language. Now let $M = \langle K, \Sigma, \Delta, s)\rangle$ be a non-deterministic Turing machine deciding L in polynomial time (since $L \in \mathbf{NP}$, one must exist). Define the non-deterministic Turing machine $M' = \langle K, \Sigma, \Delta', s\rangle$ such that $\langle q, \sigma, q', \rho, D\rangle \in \Delta \Leftrightarrow \langle q, \kappa(\sigma), q', \kappa(\rho), D\rangle \in \Delta'$.

Now the following can be shown by a simple inductive argument on the lenght of the computation: If the input for M is x and the input for M' is $\kappa(x)$, then if M permforms a sequence $\langle s, \sigma, q_1, \rho_1, d_1\rangle\langle q_1, \rho_1, q_2, \rho_2, d_2\rangle \dots \langle q_{n-1}, \rho_{n-1}, q_n, \rho_n, d_n\rangle$

of transitions ending in configuration $\langle q_n, w, u\rangle$, then M' can perform the sequence $\langle s, \kappa(\sigma), q_1, \kappa(\rho_1), d_1\rangle\langle q_1, \kappa(\rho_1), q_2, \kappa(\rho_2), d_2\rangle \ldots \langle q_{n-1}, \kappa(\rho_{n-1}), q_n, \kappa(\rho_n), d_n\rangle$ of transitions ending in configuration $\langle q_n, \kappa(w), \kappa(u)\rangle$. And, the other way round, if M' performs a sequance $\langle s, \tilde{\sigma}, q_1, \tilde{\rho}_1, d_1\rangle \ldots \langle q_{n-1}, \tilde{\rho}_{n-1}, q_n, \tilde{\rho}_n, d_n\rangle$ of transitions ending in configuration $\langle q_n, \tilde{w}, \tilde{u}\rangle$, then M can permform a sequence $\langle s, \sigma, q_1, \rho_1, d_1\rangle \ldots \langle q_{n-1}, \rho_{n-1}, q_n, \rho_n, d_n\rangle$ of transitions ending in configuration $\langle q_n, w, u\rangle$ such that $\tilde{\sigma} = \kappa(\sigma)$, $\tilde{\rho}_i = \kappa(\rho_i)$ for all $1 \leq i \leq n$, $\tilde{w} = \kappa(w)$ and $\tilde{u} = \kappa(u)$.

Induction base: if $n = 0$, the initial configurations $\langle s, \triangleright, x\rangle$ and $\langle s, \triangleright, \kappa(x)\rangle$ fulfill the condition. Induction hypothesis: After the corresponding sequences of transitions of length n, the configurations of M and M' are $\langle q, w, u\rangle$ and $\langle q, \kappa(w), \kappa(u)\rangle$, respectively. Induction step: From the construction of M' it is easy to see that if M transforms a configuration $\langle q_n, w, u\rangle$ to $\langle q', w', u'\rangle$ by executing a transition $\langle q, \sigma, q', \rho, d\rangle$, then the corresponding transition $\langle q, \kappa(\sigma), q', \kappa(\rho), d\rangle$ of M' transforms $\langle q, \kappa(w), \kappa(u)\rangle$ to $\langle q', \kappa(w'), \kappa(u')\rangle$. Similarly, if M' can transform $\langle q, \tilde{w}, \tilde{u}\rangle$ to $\langle q', \tilde{w}', \tilde{u}'\rangle$ by executing a transition $\langle q, \tilde{\sigma}, q', \tilde{\rho}, d\rangle$, then there is, by the construction of M', a transition $\langle q, \sigma, q', \rho, d\rangle$ in M such that $\tilde{\sigma} = \kappa(\sigma)$ and $\tilde{\rho} = \kappa(\rho)$ and therefore it is possible to M to transform $\langle q_n, w, u\rangle$, where $\kappa(w) = \tilde{w}$ and $\kappa(u) = \tilde{u}$, to $\langle q', w', u'\rangle$ such that $\kappa(w') = \tilde{w}'$ and $\kappa(u') = \tilde{u}'$.

Now it is clear that M' decides the language $\kappa(L)$ in polynomial time. This is because for each $x = x_1 \ldots x_n \in L$, M has a accepting computation of polynomial length since M decides L. By the above result the corresponding computation visiting the same states in M' accepts $\kappa(x)$. On the other hand, if there is an accepting computation in M' for a string $x' = x'_1 \ldots x'_n$, then by the above result, there is a corresponding computation in M accepting some $x = x_1 \ldots x_n$ such that $\kappa(x) = x'$. Therefore M' accepts x' iff $x' \in \kappa(L)$. Thus **NP** is closed under codings.

(b)

Show that **P** is closed under codings if and only if **P** = **NP**.

If **P** = **NP**, then by the previous result **P** is closed under codings.

Obviously the construction in the more complicated proof of part (a) does not work for **P** since the constructed machine M' may be non-deterministic even when M is deterministic because κ is not necessarily one-to-one. Similarly, the machine used in the simpler proof of (a) is not deterministic, either.

Now let L be the language such that each string in L consists of

1. a Boolean expression represented as on page 77 in the book and

2. a truth valuation for the expression satisfying it. However, the valuation is expressed by using extra symbols $x', 0', 1', ,, =, \mathbf{true}', \mathbf{false}'$ which do not occurr in the representation of the expression.

The parts are separated with ';'. For instance, $(x1 \wedge \neg x2) \vee x3; x'1' = \mathbf{true}', x'2' = \mathbf{false}', x'3' = \mathbf{false}'$ is a string string in L (numbers are presented in binary of same lenght, e.g. $3' \equiv 1'0'$ and $1' \equiv 0'1'$). The corresponding problem SAT CHECKING "Given a Boolean expression and a truth valuation for it, does the truth valuation satisfy the expression" is clearly in **P**, i.e., L can be decided in deterministic polynomial time. Now take a coding κ mapping the symbols $x', 0', 1', ,, =, \mathbf{true}'$ and $\mathbf{false}'$ to a pseudo blank symbol $\sqcup'$, keeping other symbols intact. Now $\kappa(L)$ is the set of all satisfiable Boolean expressions (each augmented with $v \times (1 + \log v + 2)$ pseudo blanks where v is the number of variables in the expression), i.e, deciding $\kappa(L)$ is the SAT problem. If **P** were closed under coding, then there would be a deterministic polynomial time Turing machine deciding $\kappa(L)$ and thus SAT would be in **P**. This would imply that, since all problems in **NP** are reducible in polynomial time (by using log-space reductins) to SAT (since SAT is **NP**-complete), all problems in **NP** would be solvable in polynomial time and **P** would be **NP**.

We conclude that **P** is closed under codings iff $\mathbf{P} = \mathbf{NP}$.