A proof that HORNSAT is P-complete

A clause $l_1 \lor l_2 \lor \cdots \lor l_n$, 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 $\mathbf{P}$ (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* $\mathbf{P}$*-complete under log-space reductions.*

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

Let $C = (V = \{v_1, \ldots, v_n\}, E)$ be a monotone, variable-free Boolean circuit. Thus each gate of C is of sort $\land$, $\lor$, $\mathbf{true}$ or **false**, and the answer to MONOTONE CIRCUIT VALUE is "yes" if and only if the output gate of the circuit C evaluates to $\mathbf{true}$. 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 if and only if the output gate of the circuit C evaluates to $\mathbf{true}$. We first convert C into a Horn CNF Boolean expression $\tilde{f}(C) = \bigwedge_{v \in V} C_v$ 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. $\tilde{f}(C)$ is defined as follows:

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

Note that the construction of $\tilde{f}(C)$ is NOT exact. For instance, if v is an $\land$-gate with its children v' and v'' assigned to true (i.e., $\hat{v}'$ and $\hat{v}''$ are assigned to be false), then the variable $\hat{v}$ corresponding to the gate v can have either of the values true or false since the clause $C_v = \neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}$ is already satisfied. BUT, if $\hat{v}'$ and $\hat{v}''$ are assigned to be true (i.e., 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 a gate v of C evaluates to* **false** *in C, then all satisfying truth assignments for $\tilde{f}(C)$ must assign the variable $\hat{v}$ to* $\mathbf{true}$.

Proof. By induction on the structure of C.

Induction base: For a constant input gate with $s(v) = \mathbf{true}$ the lemma holds trivially because v cannot evaluate to **false**. If v is an a constant input gate with $s(v) =$ **false**, then the clause $\hat{v}$ is in $\tilde{f}(C)$ and the lemma clearly holds.

Induction hypothesis: Assume a sub-sircuit C' of C for which the lemma holds. C' is assumed to be closed under the children-relation, i.e. if a gate v is in C', then its children are in C', too.

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 $\tilde{f}(C'')$ 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 **false** in C'', then both v' and v'' must evaluate to **false** in C'' and in C'. By the induction hypothesis, it must be that $\hat{v}'$ and $\hat{v}''$ are assigned to $\mathbf{true}$ in any satisfying truth assignment for $\tilde{f}(C')$. Thus $\hat{v}$ must be assigned to $\mathbf{true}$ in any satisfying truth assignment for $\tilde{f}(C'') = (\neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}) \wedge \tilde{f}(C')$.

2. Assume that v is a $\wedge$-gate. If v evaluates to **false** in C'', then v' or v'' evaluates **false** in C'' and in C'. By the induction hypothesis, it must be that $\hat{v}'$ or $\hat{v}''$ is assigned to $\mathbf{true}$ in any satisfying truth assignment for $\tilde{f}(C')$. Thus $\hat{v}$ must be assigned to $\mathbf{true}$ in any satisfying truth assignment for $\tilde{f}(C'') = (\neg\hat{v}' \vee \hat{v}) \wedge (\neg\hat{v}'' \vee \hat{v}) \wedge \tilde{f}(C')$.

□

Lemma 3 *The truth assignment for $\hat{v}_1, \ldots, \hat{v}_n$ that assigns $\hat{v}_i$ to* $\mathbf{true}$ *(***false***) if and only if v evaluates to* **false** *(*$\mathbf{true}$*) in C, satisfies $\tilde{f}(C)$.*

Proof. By inspecting all the clauses in $\tilde{f}(C)$.

1. Case $s(v) = \vee$ and v', v'' are the children of v.

   If either v' or v'' evaluates to **true**, then (i) v evaluates to **true**, (ii) either $\hat{v}'$ or $\hat{v}''$ is assigned to **false**, (iii) $\hat{v}$ is assigned to **false**, and (iv) the clause $C_v = \neg\hat{v}' \vee \neg\hat{v}'' \vee \hat{v}$ is satisfied.

   If both v' and v'' evaluate to **false**, then (i) v evaluates to **false**, (ii) $\hat{v}$, $\hat{v}'$ and $\hat{v}''$ are assigned to **true**, and (iii) the clause $C_v = (\neg\hat{v}' \vee \hat{v}) \wedge (\neg\hat{v}'' \vee \hat{v})$ is satisfied.

2. Case $s(v) = \wedge$ and v', v'' are the children of v.

   If both v' and v'' evaluate to **true**, then (i) v evaluates to **true**, (ii) $\hat{v}$, $\hat{v}'$ and $\hat{v}''$ are assigned to **false**, and (iii) the clauses in $C_v = (\neg\hat{v}' \vee \hat{v}) \wedge (\neg\hat{v}'' \vee \hat{v})$ are satisfied.

   If either v' or v'' evaluates to **false**, then (i) v evaluates to **false**, (ii) either $\hat{v}'$ or $\hat{v}''$ is assigned to **true**, (iii) $\hat{v}$ is assigned to **true**, and (iv) the clauses in $C_v = (\neg\hat{v}' \vee \hat{v}) \wedge (\neg\hat{v}'' \vee \hat{v})$ are satisfied.

3. Case $s(v) =$ **true**. Now v evaluates to **true**, $\hat{v}$ is assigned to **false**, and the clause $C_v = \neg\hat{v}$ is satisfied.

4. Case $s(v) =$ **false**. Now v evaluates to **false**, $\hat{v}$ is assigned to **true**, and the clause $C_v = \hat{v}$ is satisfied.

□

As a consequence of the two lemmas above, we have the following.

Theorem 4 *The output gate v_1 of C evaluates to* **true** *if and only if there is a satisfying truth assignment for $\tilde{f}(C)$ assigning the variable $\hat{v}_1$ to* **false**.

Now we form $f(C)$ from $\tilde{f}(C)$ by adding an extra Horn clause $\neg\hat{v}_1$ for the output gate v_1 of C. It is a clear consequence of the theorem above that the output gate v_1 of C evaluates to **true** if and only if $f(C)$ 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 $vi = vj \vee vk$, $vi = vj \wedge vk$, $vi = \top$ (for **true** gates) or $vi = \bot$ (for **false** gates), and the numbers $1 \leq i, j, k \leq n = |V|$ are coded in binary by using symbols 0 and 1. Now the reduction can be carried out by using three "registers" (tapes or tape positions) 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 ($\vee$, $\wedge$, $\mathtt{true}$ 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. $\square$

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 $x = x_1x_2 \cdots x_n \in \Sigma^*$, define $\kappa(x) = \kappa(x_1)\kappa(x_2) \cdots \kappa(x_n)$. 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 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 $x' = x'_1x'_2 \ldots x'_n$, M' first non-deterministically guesses a string $x = x_1x_2 \ldots x_n$ and then deterministically checks that $\kappa(x) = x'$ (remember that the coding is given). The machine M' then executes M on x as a subroutine and says "yes" ("no") 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 Non-Proof

To learn a lesson, we show a construction that does *not* work. 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 $\langle q, \sigma, q', \rho, D\rangle \in \Delta \Leftrightarrow \langle q, \kappa(\sigma), q', \kappa(\rho), D\rangle \in \Delta'$. That is, we choose M' to be the "coding" of the machine M.

First, the following can be shown by a simple inductive argument on the length of the computation: If the input for M is x and the input for M' is $\kappa(x)$, then if M can perform a transition sequence

$$\langle s, \sigma, q_1, \rho_1, d_1\rangle\langle q_1, \rho_1, q_2, \rho_2, d_2\rangle \cdots \langle q_{n-1}, \rho_{n-1}, q_n, \rho_n, d_n\rangle$$

ending in configuration $\langle q_n, w, u\rangle$, then M' can perform the sequence

$$\langle s, \kappa(\sigma), q_1, \kappa(\rho_1), d_1\rangle\langle q_1, \kappa(\rho_1), q_2, \kappa(\rho_2), d_2\rangle \cdots \langle q_{n-1}, \kappa(\rho_{n-1}), q_n, \kappa(\rho_n), d_n\rangle$$

of transitions ending in configuration $\langle q_n, \kappa(w), \kappa(u)\rangle$. That is, if for an input x there is a computation of M ending in "yes" state, then for the input $\kappa(x)$ there is a computation of M' ending in "yes" state. Therefore, $\mathcal{L}(M') \supseteq \kappa(L)$.

However, the converse does *not* necessarily hold. It may be possible that for an input $\tilde{x} = \tilde{x}_1 \ldots \tilde{x}_n$ the machine M' can perform a sequence

$$\langle s, \tilde{\sigma}, q_1, \tilde{\rho}_1, d_1\rangle \cdots \langle q_{n-1}, \tilde{\rho}_{n-1}, q_n, \tilde{\rho}_n, d_n\rangle$$

of transitions ending in configuration $\langle q_n, \tilde{w}, \tilde{u}\rangle$, while there is no input $x = x_1 \ldots x_n$ such that $\kappa(x) = \tilde{x}$ for which the machine M can perform 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 $1 \le i \le n$, $\tilde{w} = \kappa(w)$ and $\tilde{u} = \kappa(u)$. For instance, let

$$M = \langle K = \{p, q\}, \Sigma = \{a, b\}, \Delta = \{(p, a, q, b, -), (q, a, \text{“yes”}, a, -)\}, p\rangle$$

and let $\kappa = \{a \mapsto a, b \mapsto a\}$. Now

$$M' = \langle K = \{p, q\}, \Sigma = \{a, b\}, \Delta' = \{(p, a, q, a, -), (q, a, \text{“yes”}, a, -)\}, p\rangle.$$

It is quite clear that M cannot enter the "yes" state on any input and thus $L = \mathcal{L}(M) = \emptyset$. However, with any input beginning with a, M' enters its "yes" state after two steps. Thus $\mathcal{L}(M') = a^*$, which does not equal to $\kappa(L) = \emptyset$. To sum up, although $\mathcal{L}(M') \supseteq \kappa(L)$, sometimes it possible that $\mathcal{L}(M') \supset \kappa(L)$ and therefore $\mathcal{L}(M') \neq \kappa(L)$.

(b)

Show that $\mathbf{P}$ is closed under codings if and only if $\mathbf{P} = \mathbf{NP}$.

If $\mathbf{P} = \mathbf{NP}$, then by the previous result $\mathbf{P}$ is closed under codings.

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

1. a Boolean expression represented as in page 77 of 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 occur 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 in L (numbers are presented in binary of *same length*, e.g. $3' \equiv 1'0'$ and $1 \equiv 01$). 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 $\mathbf{P}$, 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 $v \times (1 + \lceil \log v \rceil + 2)$ pseudo blanks where v is the number of variables in the expression), i.e., $\kappa(L)$ is the SAT problem. If $\mathbf{P}$ were closed under coding, then there would be a deterministic polynomial time Turing machine deciding $\kappa(L)$ and thus SAT would be in $\mathbf{P}$. Since all the problems in $\mathbf{NP}$ are reducible in polynomial time (by using log-space reductions) to SAT (SAT is $\mathbf{NP}$-complete), this would imply that all the problems in $\mathbf{NP}$ are solvable in polynomial time and $\mathbf{P} = \mathbf{NP}$.

We conclude that $\mathbf{P}$ is closed under codings if and only if $\mathbf{P} = \mathbf{NP}$.