Lecture #12: Classical Reductions
Proofs of Claims

Once again, this document is primarily "for interest only," that is, for students introduced in more details about the material included in the lecture notes.

1 Completing a Proof that $L_{\textbf{FSAT}} \preceq_{\textbf{P, M}} L_{\textbf{CNF-SAT}}$

Consider the reduction from L_{FSAT} to $L_{\text{CNF-SAT}}$ described, at a high level, in the notes for Lecture #12. The goal of this section is to describe, in more detail, an algorithm that can be used to compute the function $f : \Sigma_F^\star \to \Sigma_F^\star$ described in these notes, and to sketch a proof that this can be implemented using a deterministic multi-tape Turing machine, using a number of moves that is at most polynomial in the length of the input string.

During the description of the algorithm being presented, a trace of execution will be provided on an input formula

$$((\neg(x_1 \vee \neg x_4) \wedge x_5) \vee x_2).$$

1.1 Notation: Renaming Variables

Consider an input string $\omega \in \Sigma_F^\star$. As described in the previous lecture $L_{\mathsf{F}} \in \mathcal{P}$, so it is possible to decide whether $\omega \in L_{\mathsf{F}}$ using time at most polynomial in $|\omega|$. It this is not the case then the empty string λ should be returned as output (by having this as the non-blank string on an output tape).

Suppose, instead, that $\omega \in L_{\mathsf{F}}$. Let $\mathcal{F}$ be the formula, defined over the set of Boolean variables $\mathcal{V} = \{x_0, x_1, x_2, \dots\}$, that is encoded by ω.

In this construction a new Boolean variable $y_{\mathcal{G}}$ will be used for each subformula $\mathcal{G}$ of $\mathcal{F}$, and a formula $\widehat{\mathcal{F}}$ will be obtained over the set of variables

$$\mathcal{W} = \mathcal{V} \cup \{y_{\mathcal{G}} \mid \mathcal{G} \text{ is a subformula of } \mathcal{F}\}.$$

Suppose that $\mathcal{F}$ has ℓ subformulas $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_\ell$, so that $\mathcal{W} = \mathcal{V} \cup \{y_{\mathcal{G}_i} \mid 1 \leq i \leq \ell\}$.

Since only finitely many variables appear in $\mathcal{F}$, there exists an integer $k \geq 0$ such that x_k appears in $\mathcal{F}$ but x_h does not appear in $\mathcal{F}$ for any integer h such that $h > k$. Thus the truth value of $\mathcal{F}$, under a truth assignment φ, only depends on the truth values that φ assigns to the variables in the set

$$\mathcal{V}_k = \{x_0, x_1, x_2, \ldots x_k\}$$

— and the only variables that appear in the formula $\mathcal{F}$ in conjunctive normal form being considered, here, belong to the finite set

$$\widehat{\mathcal{W}} = \mathcal{V}_k \cup \{y_{\mathcal{G}} \mid \mathcal{G} \text{ is a subformula of } \mathcal{F}\} = \{x_0, x_1, x_2, \ldots, x_k\} \cup \{y_{\mathcal{G}_1}, y_{\mathcal{G}_2}, \ldots, y_{\mathcal{G}_\ell}\}.$$

Now, in order to obtain a reduction to L_{CNFSAT}, we must map $\mathcal{F}$ to a Boolean formula in conjunctive normal form that only depends on the variables in $\mathcal{V}$, instead. In order to obtain this, consider a map $\psi : \widehat{\mathcal{W}} \to \mathcal{V}$ such that

- $\psi(x_i) = x_i$ for every integer i such that $0 \leq i \leq k$, and
- $\psi(y_{\mathcal{G}_j}) = x_{k+j}$ for every integer j such that $1 \leq j \leq \ell$.

Then ψ is a well-defined mapping from $\widehat{\mathcal{W}}$ to $\mathcal{V}$ such that distinct variables in $\widehat{\mathcal{W}}$ are mapped to distinct variables in $\mathcal{V}$ — this is an "injective function".

This renaming is completed by extending ψ to be a function defined over Boolean formulas, as well: For any Boolean formula $\widehat{\mathcal{G}}$ defined on the variables in $\widehat{W}$, let $\psi(\widehat{\mathcal{G}})$ be the corresponding Boolean formula that is obtained by replacing each occurrence of a variable $z \in \widehat{\mathcal{W}}$ with the corresponding variable $\psi(z) \in \mathcal{V}$ — so that $\psi(\widehat{\mathcal{G}})$ is defined over the variables in $\mathcal{V}$. Then $\psi(\widehat{\mathcal{G}})$ is satisfiable if only if $\widehat{\mathcal{G}}$ is.

Rather than generating the formula $\widehat{\mathcal{F}}$ described in the lecture notes, the algorithm presented here will rename variables "on the fly" so that it is only working with variables in $\mathcal{V}$ and is generating the corresponding formula $\psi(\widehat{\mathcal{F}})$, instead.

1.2 Invariant to be Maintained

As described in the lecture notes, a set $\mathcal{S}_{\mathcal{G}}$ of clauses (that are defined over the set of Boolean variables in $\mathcal{W}$) is defined for each subformula $\mathcal{G}$ of $\mathcal{F}$. These clauses are defined so that a truth assignment $\varphi : \mathcal{V} \to \{\mathtt{T}, \mathtt{F}\}$ can only be extended, in such a way that the clauses in this set can all be satisfied, by requiring $y_{\mathcal{G}}$ to receive the truth value that the subformula $\mathcal{G}$ has. In particular, the following definitions were given.

- If $\mathcal{G}$ is the Boolean variable x_i then

$$\mathcal{S}_{\mathcal{G}} = \{(y_{\mathcal{G}} \vee \neg x_i), (\neg y_{\mathcal{G}} \vee x_i)\}. \tag{1}$$

- If $\mathcal{G}$ is $\neg\mathcal{G}'$ for another subformula $\mathcal{G}'$ then

$$\mathcal{S}_{\mathcal{G}} = \{(y_{\mathcal{G}} \vee y_{\mathcal{G}'}), (\neg y_{\mathcal{G}} \vee \neg y_{\mathcal{G}'})\} \cup \mathcal{S}_{\mathcal{G}'}. \tag{2}$$

- If $\mathcal{G} = (\mathcal{G}_1 \wedge \mathcal{G}_2 \wedge \ldots \wedge \mathcal{G}_k)$ for $k \geq 1$, then

$$\mathcal{S}_{\mathcal{G}} = \{(y_{\mathcal{G}} \vee \neg y_{\mathcal{G}_1} \vee \neg y_{\mathcal{G}_2} \vee \cdots \vee \neg y_{\mathcal{G}_k})\} \cup \{(\neg y_{\mathcal{G}} \vee y_{\mathcal{G}_i}) \mid 1 \leq i \leq k\} \cup \bigcup_{1 \leq i \leq k} \mathcal{S}_{\mathcal{G}_i}. \tag{3}$$

- If $\mathcal{G} = (\mathcal{G}_1 \vee \mathcal{G}_2 \vee \cdots \vee \mathcal{G}_k)$ for $k \geq 2$, then[1]

$$\mathcal{S}_{\mathcal{G}} = \{(\neg y_{\mathcal{G}} \vee y_{\mathcal{G}_1} \vee y_{\mathcal{G}_2} \vee \cdots \vee y_{\mathcal{G}_k})\} \cup \{(y_{\mathcal{G}} \vee \neg y_{\mathcal{G}_i}) \mid 1 \leq i \leq k\} \cup \bigcup_{1 \leq i \leq k} \mathcal{S}_{\mathcal{G}_i}. \tag{4}$$

For each set $\mathcal{S}$ of clauses defined over Boolean variables in $\mathcal{W}$, let us extend the function ψ again (so that it is now defined for sets of Boolean formulas) by setting

$$\varphi(\mathcal{S}) = \{\psi(\widehat{\mathcal{C}}) \mid \widehat{\mathcal{C}} \in \mathcal{S}\}$$

so that $\psi(\mathcal{S})$ includes the corresponding clauses defined over variables in $\mathcal{V}$, as defined above. One can see by the above definitions that if $\mathcal{H}$ is a subformula of $\mathcal{G}$ then $\mathcal{S}_{\mathcal{H}} \subseteq \mathcal{S}_{\mathcal{G}}$, and $\psi(\mathcal{S}_{\mathcal{H}}) \subseteq \psi(\mathcal{S}_{\mathcal{G}})$.

When given a string $\omega \in \Sigma_F^\star$ encoding a Boolean formula $\mathcal{F}$ as input, the algorithm to be described will — effectively[2] — maintain

- a Boolean formula $\widehat{\mathcal{G}}$, defined on the set of Boolean variables in $\mathcal{W}$, and
- a set $\mathcal{S}$ of clauses, each defined over the variables in $\mathcal{W}$, and
- a positive integer h

such that the following property is maintained:

Invariant:

(a) The formula $\mathcal{F}$, encoded by ω, is satisfiable if and only if the formula

$$\widehat{\mathcal{G}} \wedge \bigwedge_{\mathcal{C} \in \mathcal{S}} \mathcal{C} \tag{5}$$

is satisfiable.

(b) If g is the smallest positive integer such that $y_{\mathcal{G}_g}$ does not appear in $\widehat{\mathcal{G}}$ or in any of the clauses in $\mathcal{S}$, then $\psi(y_{\mathcal{G}_g}) = x_h$.

[1] One could also apply this rule when $k = 1$: The definition of $\mathcal{S}_{\mathcal{G}}$ would be the same as when the rule above it (as shown at line (3)) is applied in this case.

[2] Since this is working with encodings, when implemented using a Turing machine, the algorithm will maintain the corresponding Boolean function $\psi(\widehat{\mathcal{G}})$ and set of clauses $\psi(\mathcal{S})$ instead.

On input $\omega \in \Sigma_F^\star \ldots$

1. `if` $(\omega \in L_{\mathsf{F}})$ `{`
2. Set $\widetilde{\mathcal{G}}$ to be the Boolean formula $\mathcal{F}$ encoded by ω
3. $\widetilde{\mathcal{S}} := \emptyset$
4. Set k be the largest integer such that the variable x_k appears in $\mathcal{F}$
5. $h := k + 1$
6. `while` (a variable x_i is a subformula of $\widetilde{\mathcal{G}}$, for an integer i such that $0 \leq i \leq k$) `{`
7. Let i be an integer such that $0 \leq i \leq k$ and x_i is a subformula of $\widetilde{\mathcal{G}}$
8. Replace the the first occurrence of x_i in $\widetilde{\mathcal{G}}$ with x_h
9. $\widetilde{\mathcal{S}} := \widetilde{\mathcal{S}} \cup \{(x_h \vee \neg x_i), (\neg x_h \vee x_i)\}$
10. $h := h + 1$

`}`

Figure 1: Polynomial-Time Many One Reduction — Beginning of Algorithm

1.3 Algorithm — and its Correctness

Consider an execution of the algorithm that begins in Figure 1 above, continues in Figure 2 on page 6, and ends in Figure 3 on page 7, on an input string $\omega \in \Sigma_F^\star$. Since the steps at lines 1 and 18 are executed if $\omega \notin L_{\mathsf{F}}$, the string $f(\omega) = \lambda$ is returned in this case — so that $f(\omega) \notin L_{\text{CNF-SAT}}$, as required. It remains only to consider the case that $\omega \in L_{\mathsf{F}}$ — in which case the steps at lines 1–17 are executed instead. As above, let $\mathcal{F}$ be the Boolean formula encoded by ω in this case.

Since $\widetilde{\mathcal{G}}$ and $\widetilde{S}$ are initialized to have values $\mathcal{F} = \psi(\mathcal{F})$ and $\emptyset$, respectively, at lines 2 and 3, the Boolean formula at line (5), above, is equal to $\mathcal{F}$ at this point — and it is certainly satisfiable if and only $\mathcal{F}$ is. The variable h is defined to have value $k + 1$, for k as described above, so that $\psi(y_{\mathcal{G}_1}) = x_h$. Since no variables $y_{\mathcal{G}_g}$ appear in $\widehat{\mathcal{G}}$ or in any clause in the set $\mathcal{S}$ at this point, this establishes the *invariant* given above (where, again, each Boolean formula $\mathcal{G}$ is being replaced, on the fly, by $\psi(\mathcal{G})$).

Consider the loop at lines 6–10. One can see by an examination of these lines of the algorithm that each variable in $\mathcal{V}$ (which would be a variable x_i such that $0 \leq i \leq k$) is labelled as a subclause $\mathcal{G}_g$, so that this is replaced by a corresponding variable $\psi(y_{\mathcal{G}_g})$ — a variable x_{k+g}, for a positive integer g — with clauses added to $\widetilde{\mathcal{S}} = \psi(\mathcal{S})$ according to the rule at line (1), above.

Suppose, for example, that ω encodes the following Boolean formula $\mathcal{F}$:

$$((\neg(x_1 \vee \neg x_4) \wedge x_5) \vee x_2).$$

Then $\widetilde{\mathcal{G}}$ would be initialized to be this formula at line 2, $\widetilde{\mathcal{S}}$ would be initialized to be the empty set at line 3, k would be set to be 5 at line 4, and h would be initialized with value 5 at line 6. If variables are detected and replaced in order from left to right in $\mathcal{F}$, then x_1 would be replaced with $\psi(y_{\mathcal{G}_1}) = x_6$, x_4 would be replaced with $\psi(y_{\mathcal{G}_2}) = x_7$, x_5 would be replaced with $\psi(y_{\mathcal{G}_3}) = x_8$, and x_2 would be replaced with $\psi(y_{\mathcal{G}_4}) = x_8$. Clauses would be added to the set $\mathcal{S}$ each time the step at line 9 is executed and h would be incremented at each execution of the step at line 10. The test at line 4 would fail the fifth time it is checked, at which point

- $\widetilde{\mathcal{G}}$ is the formula
$$((\neg(x_6 \vee \neg x_7) \wedge x_8) \vee x_9),$$
- $\widetilde{\mathcal{S}}$ is the set of clauses
$$\{(x_6 \vee \neg x_1), (\neg x_6 \vee x_1), (x_7 \vee \neg x_4), (\neg x_7 \vee x_4), \\ (x_8 \vee \neg x_5), (\neg x_8 \vee x_5), (x_9 \vee \neg x_2), (\neg x_9 \vee x_2)\}$$
- and $h = 10$.

The above ***invariant*** is satisfied, for this example —- and it can be proved that it holds in general, after each execution of the body of the loop at lines 6–10, by induction on the number of times that the loop body has been executed so far — considering the clauses added to $\widetilde{\mathcal{S}}$ during the execution of the step at line 9. Since the number of Boolean variables appearing in $\mathcal{F}$ is finite — indeed, it is bounded by the length $|\omega|$ of the string encoding this formula — the execution of this loop eventually ends, and the execution of the algorithm continues with the step at line 11. At each execution of the body of the loop at lines 11–16, a subformula of $\widetilde{\mathcal{G}}$ with one of the forms

- $\neg v$, for a variable v,
- $(x_{g_1} \wedge x_{g_2} \wedge \cdots \wedge x_{g_m})$ for a positive integers m and non-negative integers $g_1, g_2, \ldots, g_m$, or
- $(x_{g_1} \vee x_{g_2} \vee \cdots \vee x_{g_m})$ for an integer $m \geq 2$ and non-negative integers $g_1, g_2, \ldots, g_m$.

is processed. In each case the subformula is replaced in $\widetilde{\mathcal{G}}$ with the (currently unused) variable x_h, h is incremented, and clauses are added to $\widetilde{\mathcal{S}}$, following the rules at lines (2)–(4), in order to ensure that a truth assignment can only be extended, to satisfy all of the clauses in the set, by setting the truth value of the new variable to be the truth value of the subformula it replaces in $\widetilde{\mathcal{G}}$.

11. `while` ($\widetilde{\mathcal{G}}$ is not a variable) `{`

12. `if` ($\widetilde{\mathcal{G}}$ has a subformula $\neg v$ for some variable v) `{`

13. Let $\neg x_i$ be subformula of $\widetilde{\mathcal{G}}$ for some positive integer i. Replace the subformula $\neg x_i$ with the variable in x_h in $\widetilde{\mathcal{G}}$, adding the clauses

$$(x_h \vee x_i), (\neg x_h \vee \neg x_i)$$

to the set $\widetilde{\mathcal{S}}$, and adding one to the value of h.

14. `}` `else if` ($\widetilde{\mathcal{G}}$ has a subformula $(x_{g_1} \wedge x_{g_2} \wedge \cdots \wedge x_{g_m})$ for a positive integer m and integers $g_1, g_2, \ldots g_m$) `{`

15. Let $(x_{g_1} \wedge x_{g_2} \wedge \cdots \wedge x_{g_m})$ be a subformula of $\widetilde{\mathcal{G}}$, as above. Replace this subformula with the variable x_h in $\widetilde{\mathcal{G}}$, adding to $\widetilde{\mathcal{S}}$ the clause

$$(x_h \vee \neg x_{g_1} \vee \neg x_{g_2} \vee \cdots \vee \neg x_{g_m})$$

and each of the clauses

$$(\neg x_h \vee x_{g_i})$$

such that $1 \leq i \leq m$. Increase the value of h by one.

`}` `else {`

16. Let $(x_{g_1} \vee x_{g_2} \vee \cdots \vee x_{g_m})$ be a subformula of $\widetilde{\mathcal{G}}$ for an integer $m \geq 2$ and positive integers $i_1, i_2, \ldots, i_m$. Replace this subformula with x_h in $\widetilde{\mathcal{G}}$, adding to $\widetilde{\mathcal{S}}$ the clause

$$(\neg x_h \vee x_{g_1} \vee x_{g_2} \vee \cdots \vee x_{g_m})$$

and each of the clauses

$$(x_h \vee \neg x_{g_i})$$

such that $1 \leq i \leq m$. Increase the value of h by one.

`}`

`}`

Figure 2: Polynomial-Time Many-One Reduction — Continuation of Algorithm

Consider the ongoing example. Immediately before the execution of the loop at lines 11–16, $\widetilde{\mathcal{G}}$ is the formula

$$((\neg(x_6 \vee \neg x_7) \wedge x_8) \vee x_9)$$

and $h = 10$. During the first execution of the body of the loop, the subformula $\neg x_7$ is replaced

17. Return the encoding of the Boolean formula

$$(\widetilde{\mathcal{G}}) \wedge \bigwedge_{\mathcal{C} \in \widetilde{\mathcal{S}}} \mathcal{C}$$

as output.

} else {

18. Return the empty string, λ, as output.

}

Figure 3: Polynomial-Time Many-One Reduction — Conclusion of Algorithm

by the variable x_{10} to obtain the formula

$$((\neg(x_6 \vee x_{10}) \wedge x_8) \vee x_9)$$

with the value of h increased to 11, and with new clauses added so that $\widetilde{\mathcal{S}}$ is the set

$$\{(x_6 \vee \neg x_1), (\neg x_6 \vee x_1), (x_7 \vee \neg x_4), (\neg x_7 \vee x_4), (x_8 \vee \neg x_5), (\neg x_8 \vee x_5), \\ (x_9 \vee \neg x_2), (\neg x_9 \vee x_2), (x_{10} \vee x_7), (\neg x_{10} \vee \neg x_7)\}.$$

During the second execution of the body of the loop, the subformula $(x_6 \vee x_{10})$ is replaced by the variable x_{11} to obtain the formula

$$((\neg x_{11} \wedge x_8) \vee x_9)$$

with the value of h increased to 12, and with new clauses added so that $\widetilde{\mathcal{S}}$ is the set

$$\{(x_6 \vee \neg x_1), (\neg x_6 \vee x_1), (x_7 \vee \neg x_4), (\neg x_7 \vee x_4), (x_8 \vee \neg x_5), (\neg x_8 \vee x_5), \\ (x_9 \vee \neg x_2), (\neg x_9 \vee x_2), (x_{10} \vee x_7), (\neg x_{10} \vee \neg x_7), (\neg x_{11} \vee x_6 \vee x_{10}), \\ (x_{11} \vee \neg x_6), (x_{11} \vee \neg x_{10})\}.$$

During the third execution of the body of the loop, the subformula $\neg x_{11}$ is replaced by the variable x_{12} to obtain the formula

$$((x_{12} \wedge x_8) \vee x_9)$$

with the value of h increased to 13, and with new clauses added so that $\widetilde{\mathcal{S}}$ is the set

$$\{(x_6 \vee \neg x_1), (\neg x_6 \vee x_1), (x_7 \vee \neg x_4), (\neg x_7 \vee x_4), (x_8 \vee \neg x_5), (\neg x_8 \vee x_5), \\ (x_9 \vee \neg x_2), (\neg x_9 \vee x_2), (x_{10} \vee x_7), (\neg x_{10} \vee \neg x_7), (\neg x_{11} \vee x_6 \vee x_{10}), \\ (x_{11} \vee \neg x_6), (x_{11} \vee \neg x_{10})\}.$$

During the fourth execution of the body of the loop, the subformula $(x_{12} \wedge x_8)$ is replaced by the formula x_{13} to obtain the formula

$$(x_{13} \vee x_9)$$

with the value of h increased to 14, and with new clauses added so that $\widetilde{\mathcal{S}}$ is the set

$$\{(x_6 \vee \neg x_1), (\neg x_6 \vee x_1), (x_7 \vee \neg x_4), (\neg x_7 \vee x_4), (x_8 \vee \neg x_5), (\neg x_8 \vee x_5),$$
$$(x_9 \vee \neg x_2), (\neg x_9 \vee x_2), (x_{10} \vee x_7), (\neg x_{10} \vee \neg x_7), (\neg x_{11} \vee x_6 \vee x_{10}),$$
$$(x_{11} \vee \neg x_6), (x_{11} \vee \neg x_{10}), (x_{13} \vee \neg x_{12} \vee \neg x_8), (\neg x_{13} \vee x_{12}), (\neg x_{13} \vee x_8)\}.$$

During the fifth (and final) execution of the body of the loop, the subformula $(x_{13} \vee x_9)$ is replaced by the formula x_{14} to obtain the formula

$$x_{14}$$

with the value of h increased to 15, and with new clauses added so that $\widetilde{\mathcal{S}}$ is the set

$$\{(x_6 \vee \neg x_1), (\neg x_6 \vee x_1), (x_7 \vee \neg x_4), (\neg x_7 \vee x_4), (x_8 \vee \neg x_5), (\neg x_8 \vee x_5),$$
$$(x_9 \vee \neg x_2), (\neg x_9 \vee x_2), (x_{10} \vee x_7), (\neg x_{10} \vee \neg x_7), (\neg x_{11} \vee x_6 \vee x_{10}),$$
$$(x_{11} \vee \neg x_6), (x_{11} \vee \neg x_{10}), (x_{13} \vee \neg x_{12} \vee \neg x_8), (\neg x_{13} \vee x_{12}), (\neg x_{13} \vee x_8)$$
$$(\neg x_{14} \vee x_{13} \vee x_9), (x_{14} \vee \neg x_{13}), (x_{14} \vee \neg x_9)\}.$$

At this point the loop test fails, the step at line 17 is executed, and an encoding of the formula

$$((x_{14}) \wedge (x_6 \vee \neg x_1) \wedge (\neg x_6 \vee x_1) \wedge (x_7 \vee \neg x_4) \wedge (\neg x_7 \vee x_4), (x_8 \vee \neg x_5) \wedge (\neg x_8 \vee x_5)$$
$$\wedge (x_9 \vee \neg x_2) \wedge (\neg x_9 \vee x_2) \wedge (x_{10} \vee x_7) \wedge (\neg x_{10} \vee \neg x_7) \wedge (\neg x_{11} \vee x_6 \vee x_{10})$$
$$\wedge (x_{11} \vee \neg x_6) \wedge (x_{11} \vee \neg x_{10}) \wedge (x_{13} \vee \neg x_{12} \vee \neg x_8), \wedge (\neg x_{13} \vee x_{12}) \wedge (\neg x_{13} \vee x_8)$$
$$\wedge (\neg x_{14} \vee x_{13} \vee x_9) \wedge (x_{14} \vee \neg x_{13}) \wedge (x_{14} \vee \neg x_9))$$

is returned as output.

In the general case one can prove, by induction on the number of executions of the loop body, that the ***Invariant*** is satisfied at the beginning of each execution of the body of this loop. Thus it is satisfied when the step at line 17 is reached and executed, so that the string returned as output is the encoding $f(\omega)$ of a Boolean formula in conjunctive normal form that is satisfiable if and only if $\mathcal{F}$ is satisfiable, as required.

Indeed, this algorithm computes the function $f : \Sigma_F^\star \to \Sigma_F^\star$ described in the lecture notes, so it remains only to show that the number of steps used by a multi-tape Turing machine, implementing this algorithm, is at most polynomial in the length of the input string ω.

1.4 Completeness of the Analysis

Consider a multi-tape Turing machine with five or more tapes with tapes as follows. During the first step of the execution of the Turing machine, the leftmost cell of each tape can be marked in order to make it easy to find when needed.

- The first tape is the input tape. This can be used to store a copy of the encoding of the Boolean formula $\widetilde{\mathcal{G}}$.
- The second tape can be used to maintain encodings of the clauses in the set $\widetilde{\mathcal{S}}$. Maintenance and use of this tape is simplified if no symbols are used to separate these encodings — brackets can be matched, instead, to see when one encoding ends and the next begins.
- A third tape can be used to maintain a decimal encoding of of the integer h.
- A small number of additional work tapes can be used to carry out various operations.
- A final tape will be used as the output tape.

Each of the major stages of the algorithm can now be implemented as follows.

- Since $L_{\mathsf{F}} \in \mathcal{P}$ the test at line 1 can certainly be carried out using a number of steps that is polynomial in the length of the input string.

  If the output tape has not been modified during this[3], then it suffices to end the execution immediately after this, if $\omega \notin L_F$ — because the step at line 18 should be reached and executed and the empty string returned. It therefore suffices to consider the case that $\omega \in L_{\mathsf{F}}$, so that ω encodes a Boolean formula $\mathcal{F}$.

- Assuming again that the input string was not changed, apart from marking the leftmost cell, during the above step, the steps at lines 2 and 3 can be carried out simply by ensuring that the tape heads for the second and third tapes rest at the leftmost cells of the tapes. The cost for this is dominated by the cost to check whether $\omega \in L_{\mathsf{F}}$.

  A decimal representation of k can be obtained by initializing the third tape to store the decimal representation of 0 and then sweeping over the input, comparing the index of each Boolean variable found as a subformula in $\mathcal{F}$ to the integer whose decimal representation is on this tape, and replacing this integer whenever a larger one is found. The total number of steps used can be shown to be linear in $|\omega|$. Once this has been done, the integer can be incremented, to obtain a decimal representation of $k = h + 1$ (executing the step at line 5) using $O(|\omega|)$ additional steps. The length of the decimal representation of k is at most $|\omega| + 1$.

[3] This can be arranged by coping the input onto another tape and having that other tape used, for the first step, instead of the input tape.

- If a work tape is used to store the encoding of a formula that is being constructed, and this is copied back to the first tape at the end, then the execution of the loop at lines 6–10 can be carried out by making a single sweep from the left to right over the the encoding of $\mathcal{F}$, replacing variables and adding clauses to the set $\widetilde{\mathcal{S}}$ as one goes. Each encoding of a variable, that is a substring of ω, is replaced by encoding of a new variable whose encoding certainly has length at most $2|\omega|$, so it can be argued that the length of the encoding of the formula $\widetilde{\mathcal{G}}$, resulting from this process, is at most $2|\omega|^2$. Similarly, a consideration of the pair of clauses added to $\widetilde{\mathcal{S}}$, each time a variable is replaced,, have encodings whose total length is at most the sum of $2|\omega|$, twice the length of the encoding of the variable to be replaced, and a small constant. It can therefore be argued that, when $|\omega|$ is sufficiently large, the length of the non-blank string on the second tape is at most $3|\omega|^2$ when the execution of this loop ends.

- Each subformula of $\mathcal{F}$ must begin with a different symbol in $\mathcal{F}$, and this can be used to argue that there are at most $|\omega|$ executions of the body of the loop at lines 11–16. The length of the encoding of $\widetilde{\mathcal{G}}$ is either unchanged or reduced every time the body of this loop is executed, and clauses whose encodings have total length at linear in the length of the replaced subformula — thus, in $O(|\omega|^2)$ — are added to $\widetilde{\mathcal{S}}$ at each execution of the body of this loop.

  This can be used to argue that the execution of the loop can be carried out using $O(|\omega|^3)$ moves of the Turing machine and that the length of the non-blank string on the second tape is in $O(|\omega|^3)$ when the execution of this loop ends.

- The generation of the output at line 17 is easily carried out using a sweep over the non-blank string on the second tape, writing output symbols (including leading and ending brackets, and logical operator as needed), followed by a sweep of the output tape head back to the left. The number of moves used for this is certainly in $O(|\omega|^3)$ too.

Thus the number of moves used by the Turing machine is in $O(|\omega|^3)$ — completing a proof that $L_{\text{FSAT}} \preceq_{\text{P, M}} L_{\text{CNF-SAT}}$.

2 Completing a Proof that $L_{\text{CNF-SAT}} \preceq_{\text{P, M}} L_{\text{3CNF-SAT}}$

Consider the polynomial-time many-one reduction from $L_{\text{CNF-SAT}}$ to $L_{\text{3CNF-SAT}}$ described in the lecture notes. Like the reduction from L_{FSAT} to $L_{\text{CNF-SAT}}$ considered above, this involves a transformation from a Boolean formula $\mathcal{F}$ to a Boolean formula $\widehat{\mathcal{F}}$ in simpler form — such that $\widehat{\mathcal{F}}$ depends on Boolean variables which $\mathcal{F}$ does not, but that such that $\widehat{\mathcal{F}}$ is satisfiable if and only if $\mathcal{F}$ is.

As described in the lecture notes, each clause of $\mathcal{F}$ is translated into a set of one or more clauses:

- A clause (ℓ_1) of $\mathcal{F}$ including a single literal is used to produce a set $\{(\ell_1 \vee \ell_1 \vee \ell_1)$ including a single clause, namely, the "or" of three copies of the literal in the clause from $\mathcal{F}$.
- A clause $(\ell_1 \vee \ell_2)$ of $\mathcal{F}$ including two literals is used to produce a set $\{(\ell_1 \vee \ell_2 \vee \ell_2)\}$ that also includes a single clause, namely one with three literals, including two copies of the second literal in the clause from $\mathcal{F}$.
- A clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ of $\mathcal{F}$ including three literals is used to produce a set of size one, containing this clause.
- For $k \geq 4$ a clause $(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$ including k literals is used to produce a set including $k-2$ clauses, including $k-2$ new variables that are not used anywhere else. Additional details are in the lecture notes.

An algorithm computing a function $f : \Sigma_F^\star \to \Sigma_F^\star$, making use of this idea, can be obtained by adapting — and considerably simplifying — the algorithm shown in Figures 1–3 on pages 4–7.

- The test at line 1, checking whether the input string ω belongs to L_F, should be replaced with a test whether this string belongs to L_{CNF}. Since $L_{\text{CNF}} \in \mathcal{P}$, as described in the lecture notes, this test can be implemented by a deterministic Turing machine, using a number of steps that is polynomial in the length of the input string.
- Neither an extra copy of a Boolean formula $\widetilde{\mathcal{G}}$ nor a set $\widetilde{\mathcal{S}}$ will be needed, so that the steps at lines 2 and 3 are not needed (and these can be removed). Instead, the algorithm continues (if the test at line 1 passed) with the steps at lines 4 and 5.
- Both of the loops at lines 6–10 and 11–16 can be replaced by a simple process in which the machine sweeps from left to right over the input string, processing each clause that is seen and processing it using the rules given above — writing the "and" of clauses for the corresponding set (as given above) onto the output tape — renaming the new variables on the fly, so the the variables used are $x_{k+1}, x_{k+2}, x_{k+3}$, and so on. The decimal representation of h will be incremented as new variables are used, as in the original algorithm.

A Turing machine computed the function, that uses time in $O(|\omega|^2)$ is easily described, as needed to establish that $L_{\text{CNF-SAT}} \preceq_{\text{P, M}} L_{\text{3CNF-SAT}}$.

A Bit of History

Richard Manning Karp is an American computer scientist and a Professor Emeritus at the University of California, Berkeley. In his paper "Reducibility Among Combinatorial Problems" [4], Professor Karp noted that the problem, proved by Stephen Cook to be complete for $\mathcal{NP}$ with respect to polynomial-time oracle reductions [1], was complete for $\mathcal{NP}$ with respect to polynomial-time many-one reductions as well. Professor Karp also proved numerous other problems to be "$\mathcal{NP}$-complete," in this sense, in that paper. Consequently, ***polynomial-time many-one reductions*** are often called ***Karp reductions*** in *his* honour.

Professor Karp won the Turing award in 1985, in recognition of this, and various other contributions.

Sources of Additional Information

Chapter 34 of the third edition of *Introduction to Algorithms* [2] includes an introduction to several other "classical" $\mathcal{NP}$-complete problems and sketches of proofs that they are $\mathcal{NP}$-complete.

Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness [3] is an excellent older reference that includes information about how one can prove that a language is $\mathcal{NP}$-complete, as well as ***many*** more $\mathcal{NP}$-complete problems.

References

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, third edition, 2019.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[4] Richard M. Karp. Reducibility among combinarorial problems. *The Journal of Symbolic Logic*, 40:618–619, 1975.