CPSC 511/611 — Solutions for Assignment #2

Encodings

Suppose that instances of *PUZZLE* are encoded using strings over the alphabet

$$\Sigma_P = \{\mathtt{1}, \mathtt{L}, \mathtt{R}, \mathtt{C}, \mathtt{U}\}$$

where a valid encoding of an instance of this problem

- begins with a string of n $\mathtt{1}$'s, where n is the number of columns (in each row) for each card included in this instance;
- continues with an encoding of each card — which begins with $\mathtt{L}$, continues with n $\mathtt{C}$'s and $\mathtt{U}$'s — indicating whether positions in the left column would be "covered" or "uncovered" if this card was placed "face up" (or in a ***forward*** orientation) — and continues with $\mathtt{R}$ and continues with another n $\mathtt{C}$'s and $\mathtt{U}$'s — indicating wither positions in the *right* column would be "covered" or "uncovered" if this card was placed "face down" (or in a ***backward*** orientation).

It should not be too hard to see that if instances of this problem are encoded in this way, then

- it is easy to decide whether a given string in $\Sigma_P^\star$ really *does* encode an instance of this problem, deterministically and in polynomial time: All you need to do is confirm that the string begins with some number (n) of $\mathtt{1}$'s — and that the rest of the input string has the form
$$\omega_1\omega_2\dots\omega_k$$
where each string $\omega_i \in \Sigma_P^\star$ consists of the letter $\mathtt{L}$, n symbols that are each either $\mathtt{C}$ or $\mathtt{U}$, the letter $\mathtt{R}$, and another n symbols that are each either $\mathtt{C}$ or $\mathtt{U}$.
- The length of an encoding of any instance of this problem that includes k cards, that each have columns with n rows of holes, is linear in $n \times k$. Indeed, the length of the encoding is exactly $2nk + 2k + n$.

Membership in $\mathcal{NP}$

A ***certificate*** for a Yes-instance of this problem, that includes k cards that each has n rows of (possible) holes in each column, will be a string with length k over the alphabet

$$\Sigma_C = \{\mathtt{F}, \mathtt{B}\}$$

— where, here, one might think of $\mathtt{F}$ as standing for "forward" and $\mathtt{B}$ as standing for "backward."

A polynomial-time verification for this problem, using this certificate, would simply check whether all $2n$ positions would be covered if the cards were set down (forwards or backwards) as described by the certificate.

Pseudocode for an algorithm that does this — which also validates that the input string is syntactically correct, the beginning encodes a valid instance of the problem, and that the end encodes a possible certificate for an instance with the expected number of cards — is given in Figure 1 on page 3.

If the above encoding scheme has been read and understood then it should be reasonably clear that this algorithm really *is* using the certificate to verify that the encoded instance is a Yes-instance of the *PUZZLE* problem. Since the most expensive part of the algorithm (after validation of the input string) is just a doubly nested `for`-loop it should be clear that this algorithm can be implemented as a deterministic algorithm using a number of steps that is polynomial in the length of the encoding of an instance of *PUZZLE*, included in its input, in the worst case.

Thus this is a "polynomial-time verification algorithm" for this problem – implying that this problem is in $\mathcal{NP}$.

$\mathcal{NP}$-Hardness

In order to prove that the *PUZZLE* problem is $\mathcal{NP}$-hard it will be shown that

$$L_{\textit{3CNF-SAT}} \preceq L_{\textit{PUZZLE}}$$

— that is, the language of encodings of satisfiable Boolean formulas in 3-conjunctive normal form is polynomial time mapping reducible to the language of encodings of Yes-instances of the *PUZZLE* problem (as these are defined above).

Let Σ_F be the language used to encode Boolean formulas in 3-conjunctive normal form, as defined in class. Recall that it is necessary — and sufficient — to describe a total function

$$f : \Sigma_F^\star \to \Sigma_P^\star$$

such that the following properties are satisfied.

Given a string $\omega \in (\Sigma_P \cup \Sigma_C \cup \{\#\})^\star$:

```
 1. if ($\omega = \mu\#\nu$ for a string $\mu \in \Sigma_P^\star$ and a string $\nu \in \Sigma_C^\star$) then go to step 2; otherwise
    reject.
 2. if ($\mu$ is a valid encoding of an instance of PUZZLE) then go to step 3; otherwise
    reject.
 3. Let $k$ be the number of cards in this instance and $n$ the number of (possible) holes of
    each column of each card.
 4. if ($\nu$ is a string with length $k$ in $\Sigma_C^\star$) then to to step 5; otherwise reject.
 5. for ($1 \le i \le n$) {
 6.   $c_L$ := false;  $c_R$ := false;
 7.   for ($1 \le j \le k$) {
 8.     Suppose that the encoding of the $j^{\text{th}}$ card in $\mu$ — that is, the $j^{\text{th}}$ substring with
        length $2n+2$ in $\mu$ beginning with L — is

            $\mathtt{L}\,\sigma_1\sigma_2\ldots\sigma_n\,\mathtt{R}\,\tau_1\tau_2\ldots\tau_n$

        where $\sigma_1, \sigma_2, \ldots, \sigma_n, \tau_1, \tau_2, \ldots \tau_n \in \{\mathtt{C}, \mathtt{U}\}$.
 9.     if (the $j^{\text{th}}$ symbol in $\nu$ is F) then
10.       if ($\sigma_i$ == C) then $c_L$ := true };
11.       if ($\tau_i$ == C) then $c_R$ := true };
        } else {
12.       if ($\sigma_i$ == C) then $c_R$ := true };
13.       if ($\tau_i$ == C) then $c_L$ := true };
        }
      }
14.   if (($c_L$ == false) or ($c_R$ == false)) { reject }
    }
15. accept
```

Figure 1: A Verification Algorithm for the *PUZZLE* Problem

(a) For every string $\omega \in \Sigma_F^\star$, $\omega \in L_{\mathit{3CNF\text{-}SAT}}$ if and only if $f(\omega) \in L_{\mathit{PUZZLE}}$.

(b) The function f is computable deterministically in (worst case) polynomial time.

The definition of the function f — and corresponding parts of an algorithm to compute it — are as follows.

- If $\omega \in \Sigma_F^\star$ but ω does not encode a Boolean formula in 3-conjunctive normal form, at all, then ω should be mapped to a simple string in $\Sigma_P^\star$ — such as

  ```
  RL
  ```

  — that is not a valid encoding of an instance of *PUZZLE*.

  Note that if ω does not encode a Boolean formula in 3-conjunctive normal form then $\omega \notin L_{\mathit{3CNF\text{-}SAT}}$ — and $f(\omega) \notin L_{\mathit{PUZZLE}}$, as well, if $f(\omega)$ is as defined above.

  As previously noted in class it is possible to decide whether a string $\omega \in L_F$ encodes a Boolean formula in 3-conjunctive normal form deterministically, using a polynomial number of steps in the worst case. It is, therefore, possible, to include the above test (and mapping to a string $f(\omega)$) at the beginning of a deterministic algorithm that uses a polynomial number of steps in the worst case.

- Suppose, now, that ω *does* encode a Boolean formula in 3-conjunctive normal form. A ***pre-processing*** step — that maps strings $\omega \in \Sigma_F^\star$ that encode such formulas to other strings $\widehat{\omega} \in \Sigma_F^\star$ that *also* encode such formulas —- will be useful.

  In particular, this pre-processing step will ensure the following:

  – $\widehat{\omega} \in L_{\mathit{3CNF\text{-}SAT}}$ if and only if $\omega \in L_{\mathit{3CNF\text{-}SAT}}$.

  – $\widehat{\omega}$ encodes a Boolean formula that includes each one of the Boolean variables

  $$x_0, x_1, x_2, \dots, x_{k-1}$$

  for some positive integer k, and that does not include any *other* Boolean variables. This will ensure that that the length of $\widehat{\omega}$ is at most polynomial in the product of the number of clauses in the Boolean formula it encodes and the highest index of a Boolean variable included in that formula — but also that the length of $\widehat{\omega}$ is *at least* linear in each of these values.

  – $\widehat{\omega}$ can be computed deterministically from ω using a number of steps that is at most polynomial in the length of ω in the worst case.

  Pseudocode for an algorithm that transforms ω into $\widehat{\omega}$ is as follows.

```
1. int i := 0;
2. if (the formula encoded by $\omega$ includes the variable $x_i$) then
3.   i = i + 1;
   } else {
4.   if (the formula encoded by $\omega$ includes a variable $x_j$, where $j > i$) then
5.     Modify $\omega$ so it encodes the same formula, except that $x_j$ has been renamed $x_i$
```

```
6.      i := i + 1
      } else {
7.      return $\omega$
      }
    }
```

Since this transformation simply renames variables in the Boolean formula being encoded, it ensures that $\omega \in L_{\textit{3CNF-SAT}}$ if and only if $\widehat{\omega} \in L_{\textit{3CNF-SAT}}$.

It also ensures that if this process terminates at all then the string that is returned encodes a Boolean formula that includes all of the variables

$$x_0, x_1, \ldots, x_{k-1}$$

but no other variables, for some positive integer k.

Notice that the number of times that the integer variable `i`'s value can be increased is at most one more than the number of *distinct* variables included in the formula that was originally encoded by ω, so that this is at most linear in the length of this string.

With a bit of work one can confirm that the number of steps needed for each execution of the body of the inner loop is at most linear in the length of the string ω — and that this length either stays the same or is decreased every time this body of the loop is executed. Thus the number of steps needed for the entire algorithm is at most quadratic in the length of ω — as needed to confirm that this part of the algorithm achieves all of the conditions that are given above.

- Suppose now that the processed string $\widehat{\omega}$ encodes a Boolean formula $\mathcal{F}$ with n clauses and depending on k Boolean variables

  $$x_0, x_1, \ldots, x_{k-1}.$$

  This will be mapped to a corresponding instance P of *PUZZLE* that includes $k+1$ cards, each with two columns having n positions of possible holes (one per clause) each.

  – For $0 \leq i \leq k-1$ the i^{th} card will correspond to the variable x_i. In particular, if $1 \leq j \leq n$ and the j^{th} clause of $\mathcal{F}$ is

  $$(\ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3})$$

  then the j^{th} position on the ***left*** column of the card for x_i is covered if x_i is one (or more) of the literals $\ell_{j,1}$, $\ell_{j,2}$ or $\ell_{j,3}$ appearing in the above clause, and the j^{th} position on the ***right*** column of the card is covered if $\neg x_i$ is one (or more) of these literals.

Thus the card for the variable x_i is encoded by a string

$$\mathrm{L}\,\sigma_1\sigma_2\ldots\sigma_n\,\mathrm{R}\,\tau_1\tau_2\ldots\tau_n$$

where, for $1 \leq j \leq n$,

$$\sigma_j = \begin{cases} \mathtt{C} & \text{if } x_i \text{ is one or more of the literals } \ell_{j,1}, \ell_{j,2}, \text{ and } \ell_{j,3} \\ \mathtt{U} & \text{otherwise,} \end{cases}$$

and

$$\tau_j = \begin{cases} \mathtt{C} & \text{if } \neg x_i \text{ is one or more of the literals } \ell_{j,1}, \ell_{j,2} \text{ or } \ell_{j,3} \\ \mathtt{U} & \text{otherwise.} \end{cases}$$

– There is one more ***control card***. All the positions for holes on the ***left*** are uncovered, and all the positions on the ***right*** are covered. Thus this card is encoded by a string

$$\mathtt{L\,UU\ldots U\,R\,CC\ldots C}$$

that includes n copies of $\mathtt{U}$ and n copies of $\mathtt{C}$.

Note that, since each variable x_i such that $0 \leq i \leq k-1$ appears in the Boolean formula $\mathcal{F}$ at least once, the length of the string $\widehat{\omega}$ encoding this string is at least linear in both n and k. The length of the string encoding the above instance P of *PUZZLE* is at most polynomial in the length of $\widehat{\omega}$ — and at most polynomial in the length of the original string ω as well.

Indeed, it should be clear that the function $f : \Sigma_F^\star \to \Sigma_P^\star$, that has now been described, can be computed deterministically using polynomial time (in the length of the input string) in the worst case.

It remains only to show that $\omega \in L_{\textit{3CNF-SAT}}$ if and only if $f(\omega) \in L_{\textit{PUZZLE}}$ for every string $\omega \in \Sigma_F^\star$.

Suppose first that $\omega \in L_{\textit{3CNF-SAT}}$, so that both ω and the string $\widehat{\omega}$ (obtained from ω using the "pre-processing step described above) encode satisfiable Boolean formulas in 3-conjunctive normal form. Suppose, in particular, that $\widehat{\omega}$ encodes a satisfiable Boolean formula $\mathcal{F}$ that includes each of the variables

$$x_0, x_1, \ldots, x_{k-1}$$

(and no others) and has n clauses, as described above. Let

$$t : \{x_0, x_1, \ldots, x_{k-1}\} \to \{\mathtt{true}, \mathtt{false}\}$$

be a truth assignment such that $\mathcal{F}$ is satisfied under this truth assignment — since $\mathcal{F}$ is satisfiable, at least one such truth assignment exists.

Recall that $f(\omega)$ encodes an instance P of *PUZZLE* that includes $k+1$ cards, each with two columns that each has n rows of positions where holes may exist. Suppose we place the cards in the box as follows:

- For $0 \le i \le k-1$, the card for x_i should be placed in its ***forward*** position (corresponding to the letter F in a certificate) if $t(x_i) = \texttt{true}$, and it should placed in the ***backward*** position (corresponding to the letter B in a certificate) if $t(x_i) = \texttt{false}$.
- The ***control card*** should be placed in its ***forward*** position.

Now let j be an integer such that $1 \le j \le n$. Since the formula $\mathcal{F}$ is satisfied under the truth assignment t, at least one of the three literals $\ell_{j,1}$, $\ell_{j,2}$ or $\ell_{j,3}$ is satisfied under the truth assignment t.

- If this satisfied literal is x_i, for some integer i such that $0 \le i \le k-1$, then the position at row j in the left column in the box is covered (that is, there is no hole visible at this position in the filled box) because the left column for this row in the card for x_i was covered (i.e., filled in) and this card was placed into the box in its "forward" position, since $t(x_i) = \texttt{true}$.
- On the other hand, if this satisfied literal is $\neg x_i$, for some integer i such that $0 \le i \le k-1$, then the position at row j in the left column in the box is covered because the *right* column for this row in the card for x_i was covered, and this card was placed in the box in its "backward" position, instead, since $t(x_i) = \texttt{false}$.
- Thus every position in the left column in the box is covered. Every position in the *right* column in the box is covered, as well, because the "control card" was placed in the box in its "forward" position, and every position in the right column of this card is covered (i.e., filled in).

It follows that if $\omega \in L_{\textit{3CNF-SAT}}$ then $f(\omega) \in L_{\textit{PUZZLE}}$.

Suppose, conversely, that $f(\omega) \in L_{\textit{PUZZLE}}$. Then ω *does* encode a Boolean formula in 3-conjunctive normal form, since $f(\omega) = \texttt{RL} \notin L_{\textit{PUZZLE}}$ otherwise. Since the Boolean formula encoded by ω is satisfiable if and only if the Boolean formula $\mathcal{F}$ encoded by $\widehat{\omega}$ is satisfiable, where $\widehat{\omega}$ is the string obtained from ω using the above "pre-processing" step, it is necessary and sufficient to show that $\mathcal{F}$ is satisfiable in order to prove that $\omega \in L_{\textit{3CNF-SAT}}$ as desired.

Recall that if the cards in the instance of *PUZZLE* each include columns with n positions for holes then there are also n clauses in the Boolean formula $\mathcal{F}$. Furthermore, this instance of *PUZZLE* includes $k+1 \ge 2$ cards for a positive integer k, where $\mathcal{F}$ includes each of the Boolean variables

$$x_0, x_1, \ldots, x_{k-1}$$

— and no others. There is a card for each of these variables along with one more "control card."

Now, since $f(\omega) \in L_{\textit{PUZZLE}}$ there is a way to place the cards in the box, for the instance P of *PUZZLE* that $f(\omega)$ encodes, so that no holes are visible in the box.

Notice that if we switch the orientation of each card in the box (from "forward" to "backward," and vice-versa) then this provides *another* way to cover all the positions in the box. We may therefore assume, without loss of generality, that the "control card" is in its ***forward*** position, in some placement of cards that covers all the positions in the box.

Thus the control card covers all the positions in the ***right*** column in the box but it does not cover any position in the ***left*** column in the box: One (or more) of the cards corresponding to Boolean variables must cover these positions instead.

Consider a truth assignment

$$t : \{x_0, x_1, \dots, x_{k-1}\} \to \{\texttt{true}, \texttt{false}\}.$$

that is defined as follows, using the above placement of cards in the box: For $0 \le i \le k-1$,

$$t(x_i) = \begin{cases} \texttt{true} & \text{if the card for } x_i \text{ has been placed in its } \textbf{\textit{forward}} \text{ position} \\ \texttt{false} & \text{if the card for } x_i \text{ has been placed in its } \textbf{\textit{backward}} \text{ position} \end{cases}$$

Finally, let j be an integer such that $1 \le j \le n$ and consider the truth value assigned to the j^{th} clause in $\mathcal{F}$ using the above truth assignment. Note that, since the position in row j of the left column of the box is covered, it has been covered using the card for some Boolean variable x_i such that $0 \le i \le k-1$. Suppose that

$$\mathtt{L}\,\sigma_1\sigma_2\dots\sigma_n\,\mathtt{R}\,\tau_1\tau_2\dots\tau_n$$

is the encoding of the card for this variable, so that $\sigma_1, \sigma_2, \dots, \sigma_n, \tau_1, \tau_2, \dots, \tau_n \in \{\mathtt{C}, \mathtt{U}\}$.

– If the card for x_i was placed in the box in its ***forward*** position then $t(x_i) = \texttt{true}$, as noted above. Now $\sigma_j = \mathtt{C}$, since this card could be used to cover the position in the left column in row j when the card was placed in the box in its forward position. It follows by the definition of the function f, given above, that x_i must be one of the literals included in the j^{th} clause and, since $t(x_i) = \texttt{true}$, the j^{th} clause is satisfied under this truth assignment.

– On the other hand, if the card for x_i was placed in the box in its ***backward*** position then $t(x_i) = \texttt{false}$, as noted above. Now $\tau_j = \mathtt{C}$, since this card could be used to cover the position in the left column in row j when the card was placed in the box in its backward position. It follows by the definition of the function f, given above, that $\neg x_i$ must be one of the literals included in the j^{th} clause and, since $t(x_i) = \texttt{false}$, the j^{th} clause under this truth assignment in this case, as well.

Since this is true for each integer j such that $1 \leq j \leq n$ every clause of $\mathcal{F}$ is satisfied under t, so that the Boolean formula $\mathcal{F}$ is satisfied under this truth assignment. Thus $\mathcal{F}$ is satisfiable, so that $\widehat{\omega} \in L_{\textit{3CNF-SAT}}$ and $\omega \in L_{\textit{3CNF-SAT}}$ as well, as required.

Thus $L_{\textit{3CNF-SAT}} \preceq L_{\textit{PUZZLE}}$, so that $L_{\textit{PUZZLE}}$ is $\mathcal{NP}$-hard, since $L_{\textit{3CNF-SAT}}$ is.

It now follows that $L_{\textit{PUZZLE}}$ (and the *PUZZLE* problem) is $\mathcal{NP}$-complete.

Note About Encodings and Details of Algorithms

There are, undoubtedly, *lots* of acceptable ways to encode instances of the *PUZZLE* problem as well as certificates. It was definitely ***not*** expected that you described the same encodings as are used in this solution, if you described some.

Indeed, acceptable answers for this problem might have been considerably shorter because they did not include as many details about encodings or, perhaps, a description of the verification algorithm that was quite as detailed as is found here.

On the other hand it ***was*** important that

- For the proof of membership in $\mathcal{NP}$, certificates and a verification algorithm *were* described in enough detail for it to be reasonably clear that the verification algorithm was correct and furthermore, used a number of steps that is at most polynomial in the length of the *included instance of PUZZLE* (and not some exponentially longer certificate as well);
- For the proof of $\mathcal{NP}$-hardness, some function $f : \Sigma^\star \to \Sigma_P^\star$, where Σ is the alphabet used to provide an $\mathcal{NP}$-hard language, was given in enough detail — along with an algorithm to compute it — for it to be clear that
  - f is a ***total*** function from $\Sigma^\star$ to $\Sigma_P^\star$;
  - f is correct; that is, if $L \subseteq \Sigma^\star$ was the $\mathcal{NP}$-hard language used to provide the reduction then, for all $\omega \in \Sigma^\star$, $\omega \in L$ if and only if $f(\omega) \in L_{\textit{PUZZLE}}$;
  - there is a deterministic algorithm to compute f using time that is at most polynomial in the length of the input string in the worst case.