1. Let $M_i$ run in time $\alpha_i n^r + \beta_i$ and accept $L_i$, $i = 1, 2$. First notice that our original multitape strategy for accepting $L_1 \cup L_2$ is inadequate here, since moving back to a one-tape TM will square the time requirement. Instead, a suitably quick TM $N$ to accept $L_1 \cup L_2$ runs as follows. The tape symbols will be of the form $[xy]$ where $x, y \in A^*$. We also want the ability to “mark” squares, for which we have extra symbols $[xy\ast]$ where $x, y \in AA^*$.

On input $\sigma$, $N$ first “marks” the initial tape square of the input, by replacing $a = [aa]$ say with $[aa\ast]$. It then acts like $M_1$ using only the first component of each symbol. As it does so, it keeps track in its state of whether the marked square is to the right or left of the head. If $M_1$ accepts then so does $N$. Otherwise, it finds the $\ast$ (which takes at most time $\alpha_1 n^r + \beta_1$ since that’s as far away from the home square as the head can ever get, and it knows which way to go) and then runs like $M_2$ on the second component of the symbols, accepting exactly if $M_2$ does. The total time is $2(\alpha_1 n^r + \beta_1 + 6) + \alpha_2 n^r + \beta_2 + 6$ which is $\leq \alpha n^r + \beta$ for some $\alpha, \beta$ since $r \geq 1$ and clearly $N$ accepts $L_1 \cup L_2$, as required.

The case for $L_1 \cap L_2$ is similar and that for $A^* - L_1$ easier.

2. On input $\sigma \in \{0, 1\}^*$, say $\sigma = a_1 a_2 \ldots a_n$, the TM $M$ first prints 0+ in the two squares to the left of the home square and returns to the home square. Call this the 0’th ‘counting stage’. Now suppose we have reached the $j$’th counting stage where $j \leq |\sigma|$ and written on the tape at this time we have

$$m x a_{j+1} a_{j+2} \ldots a_n$$

where either $x = +$ and the binary number

$$m = \text{number of 1’s in } a_1 a_2 \ldots a_j \text{ minus number of 0’s}$$

or $x = -$ and

$$m = \text{number of 0’s in } a_1 a_2 \ldots a_j \text{ minus number of 1’s}.$$

At this stage if $M$ sees B then it checks if the number to the left of the + or – is 0 and accepts if it is, halts without accepting if it isn’t. If on the other hand $M$ sees 0 or 1 then $M$ remembers and deletes it, moves to the binary number to the left of the +/- and amends it, and if necessary the +/-, as required. Finally it moves this whole binary number and the +/-.
at the right of it one square to the right and sets itself scanning $a_{j+2}$ in the $j+1$'st counting stage.

As far as time goes the binary number to the left of the +/- cannot ever be more than $|\sigma|$ (= $n$ in this case) so requires at most $|n| + 1$ squares and from that it can be seen that the time required to effect the $j$'th counting stage is at most $\alpha |n| + \beta$ for some fixed constants $\alpha, \beta$. Since we have to repeat this stage some $|\sigma| + 1$ times the $M$ runs in time $\alpha' n |n| + \beta'$ for some fixed $\alpha', \beta'$, as required.

3. We examine the computations of $U$ on inputs of the form $\sigma_M 111111 \tau$ where $\sigma_M$ is fixed but $\tau$ varies. We consider first the initialisation process (that is, setting up the tapes ready for the simulation), and then the simulation itself.

Clearly finding the “111111”, testing that $\sigma_M$ really does code a TM, and copying $\sigma_M$ to tape 3 depends only on $\sigma_M$ and not on the length of $\tau$; since we are considering $\sigma_M$ as fixed, this requires only a constant amount of space and time. The next stage involves copying $\tau$ to tape 2, which clearly requires time and space $\leq 2|\tau| + 2$. The process of copying $\tau$ back to tape 1 in its new “coded and padded” form can be done in time and space $(12 + |\sigma_M|)|\tau|$ (where of course $12 + |\sigma_M|$ may be regarded as constant). Similar arguments show that the heads may be returned to the left in time proportional to $|\tau|$, and with no additional space requirement, while writing $\sigma_M$ to tape 2 takes constant time and space. Thus, the initialisation process takes $\leq \alpha |\tau|$ steps and visits $\leq \beta |\tau|$ tape squares for some constants $\alpha$ and $\beta$ depending only on $\sigma$.

Next we consider what $U$ must do when simulating a step of the machine $M$. Considering first the space requirement, we see that each time $M$ uses a new tape square, $U$ will use at most $12 + |\sigma_M|$ new squares on tape 1. The total number of squares visited by $M$ on input $\tau$ is $\leq S(|\tau|)$, so the total number of new squares on tape 1 visited by $U$ during the simulation stage is $\leq (12 + |\sigma_M|)S(|\tau|)$. It is easily seen that the number of squares used on tapes 2 and 3 depend only on $\sigma_M$ and not on $\tau$, so that $U$ uses $\leq \gamma S(|\tau|)$ steps in the simulation stage for some constant $\gamma$.

Turning now to time, each time $U$ simulates a step of $M$, it must travel along tape 3 and back again searching for the correct rule to apply. The number of steps required to do this is at most twice the length of the content of tape 3; however, since the content of tape 3 depends only on $\sigma_M$ this may be regarded as constant. Thus, each step of $M$ is simulated in $\delta$ steps of $U$ for some constant $\delta$. Since the computation of $M$ on input $\tau$ takes $\leq T(|\tau|)$ steps, the simulation stage will take $\leq \delta |\tau|$.

So in total, the computation of $U$ visits $\leq \alpha |\tau| + \gamma S(|\tau|)$ tape squares, and uses $\leq \beta |\tau| + \zeta T(|\tau|)$ tape squares. Since $S(|\tau|), T(|\tau|) \geq |\tau|$ it suffices to set $\delta = \beta + \delta$ and $\lambda = \alpha + \gamma$. 

2
4. Let $n$ be even, $m$ be the integer part of $n/3$ and $C'_r$ be the crossing sequence at the interface

$$1^r0^{m-r}(10)^i|(10)^{n-2m-i}1^{m-r}0^r,$$

Much as in the result for the language of palindromes (Section 2.5) we have,

$$\sum_{i=1}^{n-2m} |C'_r| \leq T(n)$$

so for some $1 \leq i_r \leq \frac{n-2m}{2}$,

$$|C'_{i_r}| \leq \frac{2T(n)}{n-2m}.$$  

Again arguing like in the case of palindromes, we see that for $0 \leq t < r \leq m$ and $1 \leq i, j \leq \frac{n-2m}{2}$ we must have $C'_r \neq C'_j$. Indeed, the words

$$1^r0^{m-r}(10)^i|(10)^{n-2m-i}1^{m-r}0^r \text{ and } 1^t0^{m-t}(10)^j|(10)^{n-2m-j}1^{m-t}0^t$$

are both in $L_e$, and if $C'_r = C'_j$ then interleaving the accepting computations for these words at the given interfaces would yield an accepting computation for

$$1^r0^{m-r}(10)^i|(10)^{n-2m-i}1^{m-r}0^r$$

which has $2(r - t)$ more 1’s than 0’s and so is not in $L_e$.

Hence the map $r \mapsto C'_r$ is 1-1 from a set of $m$ elements into a set of

$$\sum_{i=0}^{\frac{2T(n)}{n-2m}} s^i \leq s^{2T(n)/(n-2m)+1}$$

elements, where $s$ is the number of states of $M$ (we may assume $s > 1$). By the pigeon-hole principle then,

$$m \leq \frac{2T(n)}{n-2m} + 1.$$  

Taking logs and solving for $T(n)$ gives

$$T(n) \geq \left( \frac{n-2m}{2} \right) \left( \frac{\log_2 m}{\log_2 s} - 1 \right).$$

Since $(n-2)/3 \leq m \leq n/3$ this means

$$T(n) \geq \frac{n}{6} \left( \frac{\log_2(n-2)/3}{\log_2 s} - 1 \right) \geq \alpha n \log_2 n$$
for some constant $\alpha$. [If you are a bit rusty on manipulating logs, you may want to satisfy yourself that such an $\alpha$ exists with some more detailed calculations, but in an exam this observation would be fine.]

All that was just for even values of $n$. However, since the function $T(n)$ is monotonically increasing, for sufficiently large odd $n$ we have

$$T(n) \geq T(n-1) \geq \alpha(n-1) \log_2(n-1) \geq \alpha' \log_2 n$$

for some constant $\alpha' < \alpha$, and of course for sufficiently large even $n$ we also have $T(n) \geq \alpha' \log_2 n$.

5. Recall that multitape polynomial time is the same as 1-tape polynomial time, so it will suffice to demonstrate a multitape machine $M$ accepting this language in polynomial time. The machine $M$ first checks that the input really does have the form $\overline{a_1} @ \ldots @ \overline{a_k} @ \overline{b}$ for some natural numbers $k \geq 1, a_1, \ldots, a_k, b$. If not, it rejects. At the same time, it checks whether $k \geq 2$.

If $k = 1$ then $M$ simply has to check whether $\overline{a_1} = \overline{b}$, accepting exactly if it is. If $k \geq 2$ then $M$ acts like the solution to Question 11 on Exercise Sheet 1 (using some extra working tapes), in order to add $a_1$ to $a_2$. Once it has done this, tape 1 now contains $(a_1 + a_2) @ \overline{a_3} @ \ldots @ \overline{a_k} @ \overline{b}$ so it suffices to repeat the process.

Considering now the complexity, the initial check that the input has the right form requires only a pass along the input and back, and so takes linear time. In the case $k = 1$, the check that $\overline{a_1} = \overline{b}$ can also easily be done in (2-tape) linear time. In the case $k \geq 2$, each iteration uses essentially the same number of steps as the adding machine; the one in the model solution to Ex1 Q11 clearly just passes backward and forward along its input, and so takes only linear in the length of the two numbers to be added, which is certainly less than the length of the input to $M$ (perhaps your solution takes longer — if so, try to find a more efficient one!). The number of iterations is clearly less than the length of the input, so the total number of steps is at most quadratic in the input length. Thus, $M$ runs in multitape time $\alpha n^2 + \beta$, so the language is in $\text{Time}(n^4) \subseteq \mathcal{P}$.

6. To show that $\text{PRIMES}$ is in $\mathcal{NP}^c$, we need to show that its complement $L = \{0,1\}^* \setminus \text{PRIMES}$ is in $\mathcal{NP}$. Note that $L$ is not (quite) the language of composite numbers, since it also includes 0, 1 and all those words which don’t code a natural number at all! So our NDTM begins by checking (deterministically) whether the input begins with a 0, or is equal to 1, accepting if it is. It should be clear that this step takes constant time.

Having done this, we know that the tape contains the code $\overline{n}$ for a number $n \geq 2$, so the machine can begin the real business of checking whether $n$ is composite. To do this, it will (non-deterministically) “guess” two arbitrary integers $a, b \geq 2$, and then check (using the deterministic polynomial time algorithm we are allowed to assume) whether $ab = n$, accepting exactly if
it is. Of course $n$ is composite if and only if $ab = n$ for some such $a$ and $b$, that is, if and only if some computation accepts, as required. There is one remaining subtlety: we want our NDTM to be halting, so we cannot guess arbitrarily large $a$ and $b$; but this is okay since it clearly suffices to consider $a, b < n$ or (more conveniently for implementation) $|\pi|, |\tilde{\jmath}| \leq |\pi|$

There are numerous ways to achieve this, but perhaps the easiest is to allow our NDTM a second tape (remember that multitape polynomial time is the same thing as 1-tape polynomial time!). It starts by copying $\pi$ onto tape 2. We keep the head of tape 2 at the right-hand end, but return the head of tape 1 to the left-hand end, write an @ immediately to the left of the input word and move left a further square. We now proceed leftwards one square at a time on both tapes. At each step we write (non-deterministically) either a 0 or a 1. We stop this process (non-deterministically) at any point after the second step (this ensures that we have written at least 2 characters, and hence not the representative for 0 or 1) and we definitely stop if we reach a B on tape 1. We then write another @ on tape 1, return the head of tape 2 to the rightmost end, and repeat the process to write another number of on tape 1. Finally, we apply the polynomial time algorithm to check if the two numbers we have created multiply to give $n$. [Actually some of our computations may run this algorithm on tape contents not of the form $\pi@\tilde{b}@\pi$ (how can this happen?), but these will be rejected by the algorithm and hence don’t matter.]

Considering the time complexity, it should be clear that the initial check for a 0, 1 or a non-code takes constant time, the copying of $\pi$ and writing of numbers $a$ and $b$ takes (multitape) linear time, and of course checking if $ab = n$ takes polynomial time by assumption. Thus, the algorithm takes (non-deterministic) polynomial time, so $L \in \mathcal{NP}$ and $PRIMES \in \mathcal{NP}$.

7. Since any NDTM is a TM, $\mathcal{P} \subseteq \mathcal{NP}$. If $L \in \mathcal{NP}$, say $N$ is a NDTM accepting $L$ and running in time $\alpha n^k + \beta$ ($k \geq 1$) then by remarks in the course notes the equivalent TM runs in space $\leq \alpha' (\alpha n^k + \beta) + \beta'$. Hence $L \in Space(n^k) \subseteq PSpace$. Again by remarks in the course notes if a TM runs in space $\alpha n^k + \beta$ then it runs in time $\gamma^{\alpha n^k + \beta}$ for some $\gamma$ so $PSpace \subseteq \bigcup_{p(x) \in \mathbb{N}[x]} \text{Time}(2^{p(n)})$.

8. First note that for any $p(x) \in \mathbb{N}[x]$, 
$$p(n) \leq \alpha n^k + \beta \leq \alpha |n|^k + \beta + H$$
where $H$ is the maximum of the finite set 
$$\{ \alpha n^k + \beta | |n| \leq k \}$.
Hence $\text{Time}(n^k) \subseteq \text{Time}(n|n|)$ for any $k \in \mathbb{N}$ so
$$\mathcal{P} = \bigcup_{p(x) \in \mathbb{N}[x]} \text{Time}(p(n)) = \bigcup_{k \in \mathbb{N}} \text{Time}(n^k) \subseteq \text{Time}(n|n|) \subseteq \text{Time}(2^n)$$

5
by the Time Hierarchy Theorem since
\[ \lim_{n \to \infty} \frac{\eta[n]}{2^{n/2}} = 0. \]

9. Since \( A^* \setminus L_1, A^* \setminus L_2 \in \mathcal{NP} \), by Proposition 14, \((A^* \setminus L_1) \cup (A^* \setminus L_2) \in \mathcal{NP}\). Hence \( L_1 \cap L_2 = A^* \setminus (A^* \setminus L_1) \cup (A^* \setminus L_2) \in \mathcal{NP}^c \). Similarly for union \( \cap \) and \( \cup \).

10. Pick \( \tau_0 \in A^*_1 \setminus L_2 \) and let TM \( M \) compute \( f : A^*_1 \to A^*_2 \) in polynomial time. Let the TM \( N \) act as follows: on input \( \sigma \in A^*_3 \), \( N \) first checks (in linear time) if \( \sigma \in A^*_1 \). If not then \( N \) outputs as \( g(\sigma) \), \( \tau_0 \). In this case certainly \( \sigma \notin L_1 \) since \( L_1 \subseteq A^*_1 \) so \( g \) has the required property (and works in linear time). On the other hand if \( \sigma \in A^*_1 \) then let \( N \) just run like \( M \) on \( \sigma \) to compute \( f(\sigma) \) (in polynomial time). \( N \) now checks in time linear in \( |f(\sigma)| \) (so still polynomial time since \( f(\sigma) \) cannot be longer than \( |\sigma| \) plus the number of steps taken by \( M \) on input \( \sigma \)) if \( f(\sigma) \in A^*_1 \). If \( f(\sigma) \in A^*_1 \) then set this to be \( g(\sigma) \). Notice that again in this case \( \sigma \in L_1 \) iff \( g(\sigma) \in L_2 \). Finally if \( f(\sigma) \notin A^*_1 \) then certainly \( f(\sigma) \notin L_2 \) (since \( L_2 \subseteq A^*_1 \)) so in this case \( N \) again outputs \( \tau_0 \) as \( g(\sigma) \), and again the required condition on \( g \) is satisfied. Summing the possible times taken by \( N \) shows that \( g \) is polynomial time computable.

11. Let the TM act a follows. On input \( \sigma \) replace the symbols in \( \sigma \) by 1’s so we end up with \( 1^n \) where \( n = |\sigma| \). Now to get \( 1^{kn} \) for fixed \( k \in \mathbb{N} \) just copy \( 1^n \) \( k \) times, onto tape 2 say. Similarly to get \( 1^{n^2} \) copy \( 1^n \) onto tape 2 once for each 1 in \( 1^n \), i.e. \( n \) times. To get \( 1^{n^3} \) copy \( 1^{n^2} \) once for each 1 in \( 1^n \) etc. Clearly in this way we can get \( 1^{p(n)} \) for any \( p(x) \in \mathbb{N}[x] \).

12. Suppose \( f : L_1 \leq_p L_2 \), and let \( T_f \) be a TM computing \( f \) in polynomial time.

For part (i), suppose \( L_2 \in \mathcal{P} \) and let \( M \) be a TM accepting \( L_2 \) in polynomial time. Then we can build a TM \( N \) which on input \( \sigma \in A^*_1 \) operates as follows. First, it runs like \( T_f \) to replace \( \sigma \) on the tape with \( f(\sigma) \). By assumption on \( T_f \), this takes time bounded above by a polynomial function of \( |\sigma| \). Next, the machine runs like \( M \), to check whether \( f(\sigma) \in L_2 \), which by the definition of a reduction is the same as checking if \( \sigma \in L_1 \). By assumption on \( M \), this takes time bounded above by polynomial function in \( |f(\sigma)| \) which (just as in part (ii)) is itself bounded above by a polynomial function in \( |\sigma| \). Thus, \( N \) accepts \( L_1 \) in polynomial time, and so \( L_1 \in \mathcal{P} \).

For part (iii), if \( L_2 \in \mathcal{NP}^c \) then \((A^*_2 \setminus L_2) \in \mathcal{NP}^c \). Moreover for any \( \sigma \in A^*_1 \) we have
\[ \sigma \in (A^*_1 \setminus L_1) \iff \sigma \notin L_1 \iff f(\sigma) \notin L_2 \iff \sigma \in (A^*_2 \setminus L_2) \]
so that \( f : (A^*_1 \setminus L_1) \leq_p (A^*_2 \setminus L_2) \). So by part (ii) we have \((A^*_1 \setminus L_1) \in \mathcal{NP}^c \) and hence \( L_1 \in \mathcal{NP}^c \).
For part (iv), suppose $L_2 \in PSpace$ and let $M$ be a $TM$ accepting $L_2$ in polynomial space. Then we can build a $TM$ $N$ which on input $\sigma \in A_1^*$ operates as follows. First, it runs like $T_f$ to replace $\sigma$ on the tape with $f(\sigma)$. By assumption on $T_f$, this takes time bounded above by a polynomial function of $|\sigma|$. Since the machine cannot visit more than one new tape square for each step it takes, it also takes space bounded above by a polynomial function of $|\sigma|$. Next, the machine runs like $M$, to check whether $f(\sigma) \in L_2$, which by the definition of a reduction is the same as checking if $\sigma \in L_1$. By assumption on $M$, this takes space bounded above by polynomial function in $|f(\sigma)|$ which (just as in part (ii)) is itself bounded above by a polynomial function in $|\sigma|$. Thus, $N$ accepts $L_1$ in polynomial space, and so $L_1 \in PSpace$.

13. Let $f_1, f_2, f_3, \ldots$ list all the polynomial time computable functions from $\{0, 1\}^*$ to $\{0, 1\}^*$. Notice that such a list is possible because each is computed by a TM and (as we argued in Section 1.8) there are only countably essentially different TM’s. Suppose that at stage $n$ we have determined distinct words $\sigma_i, \tau_i \in \{0, 1\}^*$ with $|\sigma_i|, |\tau_i| = i + 1$ for $i < n$.

Pick $\sigma_n \in \{0, 1\}^{n+1} \setminus \{f_i(\tau_i) \mid i \leq n\}$ and furthermore such that $f_n(\sigma_n) \notin \{\sigma_i \mid i < n\}$ if this is possible. Notice that this first choice is certainly possible since

$$|\{0, 1\}^{n+1}| = 2^{n+1} > n + 2 > n \geq \left|\{f_i(\tau_i) \mid i < n\}\right|$$

Next pick $\tau_n \in \{0, 1\}^{n+1} \setminus \{f_i(\sigma_i) \mid i \leq n\}$ and furthermore such that $f_n(\tau_n) \notin \{f(\sigma_i) \mid i \leq n\}$ if this is possible. Notice that again this first choice is certainly possible.

Finally let

$$L_1 = \{\sigma_i \mid i \in \mathbb{N}^+\}, \quad L_2 = \{\tau_i \mid i \in \mathbb{N}^+\}.$$ 

Then neither $L_1 \not\subseteq L_2$ nor $L_2 \not\subseteq L_1$, since suppose $f$ was polynomial time computable and $f : L_1 \not\subseteq L_2$. Then $f = f_n$ for some $n$. In this case if the “furthermore” option worked in defining $\sigma_n \in L_1$ then by the construction, $f_n(\sigma_n) \neq \tau_i$ for $i < n$. Furthermore since $\tau_k$ is chosen to be not equal to $f_n(\sigma_n)$ for $k \geq n$, this means that $f_n(\sigma_n) \notin L_2$, contradiction. On the other hand if the “furthermore” option could not be fulfilled it must be because for any $\sigma \in \{0, 1\}^{n+1} \setminus \{f_i(\tau_i) \mid i \leq n\}$ we have $f_n(\sigma) \in \{\tau_i \mid i < n\}$. But since the set

$$\{0, 1\}^{n+1} \setminus \{f_i(\tau_i) \mid i \leq n\}$$

has at least $2^{n+1} - n > 2$ elements there must be a $\sigma'$ in this set which is not equal to $\sigma_n$, and hence not in $L_1$ since by construction $L_1$ does not have two
elements of the same length. But then by the failure of the “furthermore case” \( f_n(\sigma') \in L_2 \), so again we have a contradiction! The proof that we also cannot have \( L_2 \subseteq L_1 \) is essentially the same.

14. Let \( \sigma \in \{0, 1, \@\}^* \). We can check by a TM if \( \sigma \) codes \( \langle n_1, n_2, \ldots, n_k \rangle \)

i.e. has the form \( n_1@n_2@\ldots@n_k \) with \( n_1, n_2, \ldots, n_k \in \mathbb{N} \), by simply going back and forth across \( \sigma \). Assuming this check is OK go through, on tape 2, in increasing lexicographic order, all words \( i_1i_2\ldots i_k \in \{0, 1\}^k \) and for each successive word check if

\[
\sum_{i_j=0} n_j = \sum_{i_j=1} n_j.
\]

Accept just if one such check proves successful by the time the enumeration of \( \{0, 1\}^k \) is complete. Since the space required to do these additions is at most \( 2 + |\sigma| \) such a TM clearly runs in linear space.

15. For \( L \subseteq \mathcal{A}^* \) and \( \iota \) the identity map on \( \mathcal{A}^* \) (which is clearly polynomial time computable!) \( \iota : L \preceq_p L \) so \( L \equiv_p L \), i.e. \( \equiv_p \) is reflexive. If \( L_1 \equiv_p L_2 \), then \( L_1 \preceq_p L_2 \) and \( L_2 \preceq_p L_1 \) so by trivially reversing these last two, \( L_2 \equiv_p L_1 \), i.e. symmetry. Finally if \( L_1 \equiv_p L_2 \), \( L_2 \equiv_p L_3 \) then \( L_1 \preceq_p L_2 \), \( L_2 \preceq_p L_3 \) so \( L_1 \preceq_p L_3 \) by Proposition 17. Similarly \( L_3 \preceq_p L_1 \) so transitivity follows.

16. Let \( \mathcal{A} = \{a_1, a_2, \ldots, a_k\} \) and as in the section of the notes on coding Turing Machines code \( a_i \) by \( \overline{a}_i = 011\hat{1}i \in \{0, 1\}^* \), \( i = 1, 2, \ldots, k \). Similarly code \( c_1c_2\ldots c_j \in \mathcal{A}^* \) by \( \overline{c}_1\overline{c}_2\ldots \overline{c}_j \in \{0, 1\}^* \). Clearly the function \( g \) mapping \( c_1c_2\ldots c_j \) to \( \overline{c}_1\overline{c}_2\ldots \overline{c}_j \) is polynomial time computable (in fact linear time on a 2TM). Let

\[
L' = \{ \overline{c}_1\overline{c}_2\ldots \overline{c}_j \mid c_1c_2\ldots c_j \in L \} \subseteq \{0, 1\}^*,
\]

so \( g : L \preceq_p L' \). Also \( h : L' \preceq_p L \) where \( h : \{0, 1\}^* \rightarrow \mathcal{A}^* \) is the polynomial time computable defined by \( h(\sigma) = c_1c_2\ldots c_j \) if \( \sigma = \overline{c}_1\overline{c}_2\ldots \overline{c}_j \) for some (necessarily unique) \( c_1c_2\ldots c_j \in \mathcal{A}^* \), and \( h(\sigma) \) equals some fixed \( \tau_0 \in \mathcal{A}^* \setminus L \) otherwise.

17. It is enough to notice that if \( M \) runs in polynomial space and accepts \( L \subseteq \mathcal{A}^* \) then the construction which showed that the recursive sets are closed under complements (Section 1.8) also shows that there is a TM accepting \( \mathcal{A}^* \setminus L \) and running in the same space as \( M \). Hence if \( L \in PSpace \) then \( (\mathcal{A}^* \setminus L) \in PSpace \) and so if \( PSpace = \mathcal{NP}c \) then

\[
L \in \mathcal{NP}c \iff L \in PSpace \iff (\mathcal{A}^* \setminus L) \in PSpace \iff (\mathcal{A}^* \setminus L) \in \mathcal{NP}c \iff L \in \mathcal{NP}c.
\]