Question 1.

We show that $PAL_{LIN}$ is undecidable by a reduction from Modified Post Correspondence Problem, which is known to be undecidable. Assume to the contrary that $PAL_{LIN}$ is decidable. We show how to convert an instance of MPCP to an instance of $PAL_{LIN}$ so that we can use the decider for $PAL_{LIN}$ to decide the instance of MPCP. Let $S$ be the start symbol of the linear grammar, $G$. For the first tile, $[t_1|b_1]$, construct the rule $S \rightarrow t_1Tb_1^{rev}$. For every tile $[t_i|b_i] 1 < i \leq \ell$, construct the rule $T \rightarrow t_iTb_i^{rev}|$ where $|$ is a special terminal. This grammar generates strings that look like $t_1t_2...t_\ell b_\ell^{rev}...b_2^{rev}b_1^{rev}$ where the left prefix before the $|$ is the string obtained by reading the tops of the tiles in a forward direction and the right suffix after the $|$ is the string obtained by reading the bottom of the tiles in a backward direction. Then, $t_1t_2...t_\ell b_\ell^{rev}...b_2^{rev}b_1^{rev}$ is a palindrome if and only if $t_1t_2...t_\ell = b_1b_2...b_\ell$ which is a solution to MPCP. Then, when we pass $G$ to a decider for $PAL_{LIN}$, if it accepts, there is some palindrome in $L(G)$ and there is a solution to the instance of MPCP; if it rejects, there is no palindrome and no solution.

We can also do the reduction from PCP. This grammar has the set of rules $S \rightarrow t_iSb_i^{rev}$, $S \rightarrow t_i|$ for $1 \leq i \leq \ell$. Note the lack of a rule $S \rightarrow |$ without other terminals present. Here, we remove the restriction that it must begin with a specified tile, but we need to ensure that at least one tile is used, as the empty string is a palindrome, but a solution to the PCP cannot be to use no tiles!

Question 2.

It seems like it’s possible to prove this using the following argument (sketch): “Construct an NFA, $N$, from $R$. Construct a DFA, $D$, from $N$. Produce $D^{rev}$, a DFA that is the reversal of $D$ so that $L(D^{rev}) = rev(L(D))$. Construct $D \cap D^{rev}$ and use $E_{DFA}$ to determine if the language is empty; if it is not, accept, otherwise, reject.” This implies that if the language and its reverse contain any strings, then the language contains a palindrome, which is not true. The problem is that this does not check to see if a string that is the reverse of itself is in the language, that is $\{w \in L(R) : w = w^{rev}\}$; rather, it checks for an instance $\{w \in L(R) \text{ and } w^{rev} \in L(R)\}$. These are not equivalent. As a concrete example, take $R = 0(00)^*11(11)^*\cup11(11)^*0(00)^*$; in this case, the language contains no palindromes, but the language is its own reverse.

Instead, we do the following. In a finite automata, we could keep track of paths strings could take from the start state to a final state. If a string can make it from $q_0$ to some other state, $q_i$ and then the reverse of that string (or one symbol and then the reverse in case of an odd length string) can make it from $q_i$ to $q_f \in F$, then the language contains
a string that is a palindrome. Then a TM that is a decider for $PAL_{REX}$ would do the following (sketch).

Convert $R$ to an NFA, $D$. For each string $w \in \Sigma^*$, we determine two sets: $S_w$ is the set of states that can be reached by having $D$ process $w$, and $F_w$ is the set of states that can reach a final state by having $D$ process $w$ starting in that state. Now $S_\epsilon$ is the $\epsilon$-closure of $\{q_0\}$, and $F_\epsilon$ is the set of states containing $F$ and all states that have a path of $\epsilon$-transitions to a state of $F$. Enter $(\epsilon, S_\epsilon, F_\epsilon)$ in a table. If $S_\epsilon \cap F_\epsilon \neq \emptyset$, accept (the empty string is in the language and is a palindrome). Now we process the strings of $\Sigma^*$ in nondecreasing order by length, starting with length 1. To process a string $w$ of length $n$, write $w = xa$ where $x$ has length $n - 1$ and $a \in \Sigma$. If the table has no entry for $x$, do not do anything more for $w$. If the table has an entry $(x, S_x, F_x)$, compute $S_w$ to be the $\epsilon$-closure of $\cup_{q \in S_x} \delta(q, a)$ and $F_w$ to be the set of all states $q$ so that the $\epsilon$-closure of $\delta(q, a)$ contains at least one state in $F_x$. Do the following checks. If $S_w \cap F_w \neq \emptyset$, accept (we have found an even length palindrome). If $S_w \cap F_x \neq \emptyset$, accept (we have found an odd length palindrome). Now check whether there is any string $z$ in the table with $S_w = S_z$ and $F_w = F_z$. If there is, do nothing further with $w$. Otherwise add entry $(w, S_w, F_w)$ to the table. If we process all strings of length $n$ without adding anything to the table, reject.

The algorithm will terminate after processing up to length at most $2^{2|Q|}$ because there are at most $2^{|Q|}$ ways to choose $S_w$ and $F_w$, and it terminates if no string of a certain length adds a new table entry. If the language contains a palindrome, this method will find it after processing the first half of the string. (If $S_w = S_z$ and $F_w = F_z$, then $w$ is a prefix of a palindrome if and only if $z$ is; indeed if $wyw^{rev}$ is a palindrome, so is $zyz^{rev}$.) So if a palindrome is in the language, our algorithm finds one.

Because this is a decider for $PAL_{REX}$, it is decidable.

**Question 3.**

We first note the relationships between $P_{TM}, \overline{P_{TM}},$ and $\neg P_{TM}$. Let $OK_{TM} = \{\langle M \rangle : M$ is a proper encoding of a TM$\}$. We have not been specific about the exact method by which Turing machines are encoded, but every reasonable encoding has the property that we can decide whether a string encodes a TM or not; in other words, $OK_{TM}$ is decidable (with decider $B$). Now

- $\Sigma^* \setminus P_{TM} = \overline{P_{TM}}$ (from the definition of complement)
- $\overline{P_{TM}} \cap OK_{TM} = \neg P_{TM}$

We can show that if a decider or recognizer exists for $\overline{P_{TM}}$, we can construct a decider or recognizer, respectively, for $\neg P_{TM}$.

Let $D$ be a decider for $\overline{P_{TM}}$. Construct the decider, $D'$, for $\neg P_{TM}$ as follows.

$D' = \text{"On input } w:\$

1. Run $D$ on $w$.
   a. If $D$ rejects, then $w \in P_{TM}$, so reject.
   b. If $D$ accepts, run $B$ on $w$ and accept if it accepts, reject if it rejects.”

$D'$ accepts only when $w \notin P_{TM}$ and $w = \langle M \rangle$ is an encoding of a TM, which is precisely the definition of $\neg P_{TM}$. A construction for a recognizer, $R'$, for $\neg P_{TM}$ given a recognizer,
R, for $\overline{P_{TM}}$, is almost identical. 

$R' = "On input w:\n\quad 1. Run B on w; if it rejects, reject.\n\quad 2. If B accepts, run R on w.\n\quad \quad a. If R rejects, reject. (w \in P_{TM})\n\quad \quad b. If R accepts, accept.\n\"$ 

(R may run forever, so $R'$ may also run forever.) $R'$ accepts only when $w \in \overline{P_{TM}}$ and $w = \langle M \rangle$ is an encoding of a TM, that is when $w \in \neg P_{TM}$.

**Question 4.**

Construct each of the infinite unhappy Turing machines as follows.

$M_i = "On input w:\n\quad 1. Obtain own description, \langle M_i \rangle via the recursion theorem.\n\quad 2. Move the tape head right $i$ positions and then left $i$ positions.\n\quad 3. Compare $w$ to \langle $M_i$ \rangle. If $w = \langle M_i \rangle$, reject; otherwise, accept."

It’s clear that for any particular $M_i$, $M_i$ is an unhappy TM; $L(M_i) = \Sigma^* \setminus \langle M_i \rangle$. The purpose of step 2 is to create a correspondence between $\mathbb{N}$ and the set of unhappy TMs so that we produce an infinite number of unhappy TMs by ensuring that if $i \neq j$ then $\langle M_i \rangle \neq \langle M_j \rangle$. Any nonsense computation that causes differences in the description of one unhappy TM from another would suffice.

**Question 5.**

Assume to the contrary that $MIN_{TM}$ has a recognizable, infinite subset, $N \subset MIN_{TM}$, and let $E$ be an enumerator for $N$. Then, we construct a TM, $C$, as follows.

$C = "On input w:\n\quad 1. Obtain own description, \langle C \rangle, via the recursion theorem.\n\quad 2. Run $E$ as a subroutine until $E$ prints some $\langle D \rangle$ with length longer than $\langle C \rangle$.\n\quad 3. Simulate $D$ on $w$ and answer however $D$ answers."

$N$ is an infinite subset of $MIN_{TM}$, but $\langle C \rangle$ is finite, so $E$ is guaranteed to eventually print some $\langle D \rangle$ with the property that $\langle D \rangle$ is longer than $\langle C \rangle$, but $\langle D \rangle$ is minimal. $C$ recognizes the same language as $D$, and $C$ is a TM that does exactly what $D$ does, yet its description is shorter than $D$’s. Thus, we have a contradiction, and $E$ cannot exist.