A. [Although you are not required to justify answers, I justfiy mine, for educational value.]

A.A. TRUE. [Using the product construction, you can build a PDA that simulates a PDA and a DFA at the same time, with final states defined so that the product PDA accepts the union of their two languages.]

A.B. FALSE. [Neither EQ_{TM} nor its complement is recognizable.]

A.C. TRUE. [Any CFL is an element of P. Hence $CFL \subseteq P \subseteq PSPACE$.]

A.D. FALSE. [We can conclude that B is NP-hard, but we do not know that $B \in NP$.]

A.E. FALSE. [We don't know that the TM in question halts at all. It could run forever, bouncing back and forth between two tape cells. If the question specified that the TM was a decider, then the answer would switch to TRUE.]

A.F. TRUE. [We know that $NP \subseteq PSPACE$, and any language in $PSPACE$ can be reduced to TQBF.]

A.G. FALSE. [Actually, this might be true, but we do not know it. What we do know is that there is an equivalent TM of time $2^{\mathcal{O}(f(n))}$.]

A.H. FALSE. [Actually, we do not know this, but the common belief is that NP is not closed under complementation. So $\bar{A}$ might not even be in NP, in which case it cannot be NP-complete.]

B.A. First, the statement is scanned into a list of tokens. This task is roughly at the level of DFAs; in our assignment, we accomplished it using regular expressions. Second, the list of tokens is parsed into a tree. This task is roughly at the level of PDAs (although for most real-world programming languages PDAs do not quite suffice). Third, the parse tree is evaluated. This task requires Turing machines, because PDAs cannot perform addition, for example.

B.B. Even if the syntax of the language is not truly context-free, it is always decidable, so interpreters and compilers can detect syntax errors reliably. Semantic errors are much harder to detect. One issue is that we would have to agree on a way of specifying the semantics of a program, separate from the program itself. But even for the easily specified semantics of "this program should halt", Turing machines cannot detect semantic errors reliably, because the halting problem is undecidable.

C. [This is a problem from Chapter 3, and from a previous exam.] Assume, for the sake of contradiction, that A is context-free. Then there exists a pumping length p for A. Let $w = 0^p1^p\#0^p1^p$. Then $w \in A$ and $|w| \geq p$, so the pumping lemma applies. There exist u, v, x, y, z such that $w = uvxyz$, $|vxy| \leq p$, $|vy| \geq 1$, and $uv^ixy^iz \in A$ for all $i \geq 0$. There are three to five cases:

- If vxy is contained in the first 0^p1^p, then we can pump v and y up, to make the left side of the string longer than the right side of the string.

- Symmetrically, if vxy is contained in the second 0^p1^p, then we can pump v and y down, to make the right side of the string shorter than the left side of the string.
- If vxy is contained in $1^p\#0^p$:
  - If # is contained in vy, then pumping up v and y produces a string with multiple #s.
  - If # is contained in x and y is non-empty, then pumping down v and y produces a string with fewer 0s on the right side than on the left side.
  - If # is contained in x and y is empty, then v is non-empty, and pumping up v and y produces a string with more 1s on the left side than on the right side.

In each of the cases, we see that w can be pumped to leave A, which contradicts the pumping lemma. We conclude that A is not context-free.

D. [This is a problem from Chapter 5.] Assume, for the sake of contradiction, that A is decidable by a Turing machine D. We will build a decider C for $EMPTY_{TM}$. This C, on input $\langle M \rangle$, does these steps:

1. Modify $M = (Q, \Sigma, \Gamma, q_0, q_{\text{acc}}, q_{\text{rej}}, \delta)$ to a new TM N, by adding a new tape symbol. More precisely, for some symbol $a \notin \Gamma$, add a to Σ and to Γ.
2. Add an $(a \to a, R)$-arrow to each state in N, as follows. On q_{acc}, make the arrow loop back to q_{acc}. For all of the other $|Q| - 1$ states, put them into any order such that q_0 is first and q_{rej} is last. On each of these states, make the arrow go to the next state in the order. The arrow on q_{rej} is not important, but make it go back to q_0. Thus the $(a \to a, R)$-arrows, except for the one on q_{acc}, form a loop of length $|Q| - 1$.
3. Run D on $\langle N \rangle$, and output whatever D outputs.

Suppose that $\langle M \rangle \in EMPTY_{TM}$. Then q_{acc} is useless in M and in N, so D accepts $\langle N \rangle$, so C accepts $\langle M \rangle$. Conversely, suppose that $\langle M \rangle \notin EMPTY_{TM}$. Then q_{acc} is not useless in M or N. Also, no other state is useless in N: When N is run on the input $a^{|Q|-2}$, it enters every state $q \neq q_{\text{acc}}$ before rejecting. Thus D rejects $\langle N \rangle$ and C rejects $\langle M \rangle$. Thus C decides $EMPTY_{TM}$, which is not decidable. This contradiction implies that A is also not decidable.

E. [I will omit the drawing.] My DFA consists of a start state followed by 10 "layers" of states, corresponding to the 10 digits of a valid ISBN. Each layer consists of 11 states. Thus there are $1 + 10 \cdot 11 = 111$ states in the whole DFA (although later we will optimize out some of them). Intuitively, the 11 states in a layer are labeled with $i = 0, 1, 2, \ldots, 10$, with state i meaning "the current remainder modulo 11 is i". From the start state, here are the transitions to the states in the first layer:

- On input symbol 0, the current remainder modulo 11 is $(10 \cdot 0)\%11 = 0$, so transition from the start state to state 0.
- On input 1, the current remainder is $(10 \cdot 1)\%11 = 10$, so transition to 10.
- On input 2, the current remainder is $(10 \cdot 2)\%11 = 9$, so transition to 9.
- On input 3, the current remainder is $(10 \cdot 3)\%11 = 8$, so transition to 8.

The pattern should be clear. Now, to give an idea of what happens in the rest of the DFA, I'll explain the transitions from state 3 of the first layer to the states in the second layer:

- On input 0, the current remainder is $(3 + 9 \cdot 0)\%11 = 3$, so transition to state 3.
- On input 1, the current remainder is $(3 + 9 \cdot 1)\%11 = 1$, so transition to 1.
- On input 2, the current remainder is $(3 + 9 \cdot 2)\%11 = 10$, so transition to 10.
- On input 3, the current remainder is $(3 + 9 \cdot 3)\%11 = 8$, so transition to 8.

...and so on. Similar structure can be imposed on every layer leading up to the last layer. In the last layer, make the 0-state the DFA's lone accept state. For all states in the last layer, all outgoing transitions go to the 10-state in the last layer, which is our "preferred reject state".

We have not yet described the X-transitions. In most layers, the X-transitions simply go to the preferred reject state, reflecting the fact that most of the digits are not allowed to be X. However, there are nontrivial X-transitions from the second-to-last layer to the last layer, to reflect the fact that the final digit is allowed to be X.

This DFA rejects all strings of length other than 10. It also rejects strings with X occurring before the final digit. It also rejects strings in which the weighted sum of the digits is not 0 modulo 11. The strings that it accepts are exactly the valid ISBNs.

We can optimize out some of the states. In the first layer, state 1 is not needed. In the final layer, only state 0 and state 10 are needed. (States $1, 2, \ldots, 9$ can all be collapsed into state 10.) So the total number of states could be reduced from 111 to 101.

[Some students realized that the set of ISBNs is finite, and thus a DFA can be hard-coded with a path for each one. This is correct, but the resulting DFA has roughly 10 billion states, which is space-inefficient by a factor of 100 million. Such responses got most, but not all, of the possible points.]