A. [I don't draw the diagram in these solutions, but I give you information that can easily be turned into the diagram. I also give justifications.] All of these sets are subsets or supersets of each other — rather than having interesting intersections, say. From smallest to largest:

context-free languages	[contained in P by the algorithm used in Problem B]
P	[contained in NP because every DTM is trivially an NTM]
NP	[contained in NPSPACE by space-time lemma]
PSPACE = NPSPACE	[$\subseteq$ because every DTM is an NTM; $\supseteq$ by Savitch]
EXPTIME	[contains PSPACE by counting configurations]
EXPSPACE	[contains EXPTIME by space-time lemma]
decidable languages	[definitions of time and space complexity assume deciders]

B. The algorithm produces the following table. Because the start variable S appears in the upper right cell, we conclude that w is derivable from the grammar.

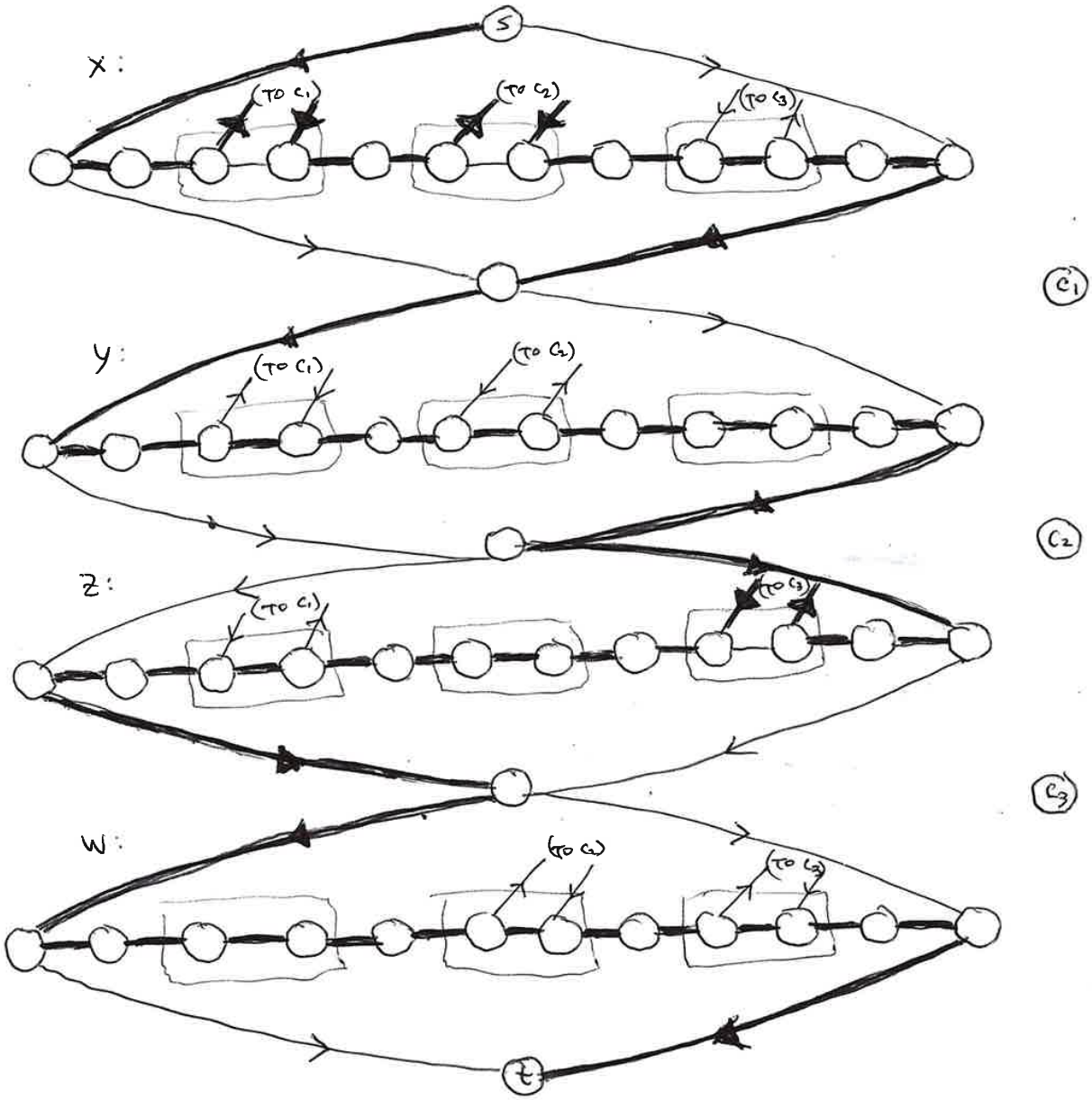
	$j=1$	2	3	4	5
$i=1$	S, T, A	U	[none]	[none]	S, T
2		S, T, A	V	T, S	U
3			S, T, B	U	[none]
4				S, T, A	U
5					S, T, A

C. The upper bound in question is $t(n) \leq 2^{\mathcal{O}(s(n))}$, which is the number of configurations that an $s(n)$-space Turing machine has. If the deterministic Turing machine is a decider, then it can't repeat a configuration, because then it would repeat indefinitely and therefore hang.

So the question is: Does nondeterminism allow a nondeterministic decider to repeat a configuration? Intuitively, you might think so, because the machine can choose to repeat once or twice and then not repeat any more. But recall that we defined a nondeterministic Turing machine to be a decider if, on every input, every branch of computation halts. So, if it is able to repeat a configuration, then some branch will indefinitely repeat that configuration, and the nondeterministic Turing machine will not be a decider.

So yes, the argument works; $t(n) \leq 2^{\mathcal{O}(s(n))}$ for nondeterministic Turing machines as well.

D.A. [Here I can't avoid drawing a picture.] To keep the picture clean, I have adopted an unusual notation: The six edges marked "to c_1" attach to node c_1, and similarly for c_2 and c_3. Also, edges without arrows are meant to be bi-directional.



D.B. Yes, the formula can be satisfied, and so there is a Hamiltonian path through G from s to t. Here's an example, also shown in bold in the diagram above.

A satisfying assignment is x = TRUE, y = TRUE, z = FALSE, w = TRUE. The corresponding path traverses the x diamond left-to-right, the y diamond left-to-right, the z diamond right-to-left, and the w diamond left-to-right.

The detours to c_1, c_2, c_3 are not unique. The fact that x makes the first clause TRUE allows the path to detour to c_1 from the x diamond. The fact that x makes the second clause TRUE allows the path to detour to c_2 from the x diamond. The fact that $\bar{z}$ makes the third clause true allows the path to detour to c_3 from the z diamond.

E.A. Let $s = 0^p 1 1 0^p$.

E.B. Let $s = 0^p 1 0^p$ (which uses $w = \epsilon$).

E.C. Let $s = 0^p 1^p 2 0^p 1^p$.

E.D. Let $s = 0^p 1 0^{2p} 1 0^{3p}$.

F. Yes, $\neq$SAT is NP-complete, as we now check.

First, $\neq$SAT is an element of NP, for approximately the same reason that SAT and 3SAT are: A polynomial-time verifier could take a proposed unequal-satisfying assignment as a certificate and check whether it works.

Second, $\neq$SAT is NP-hard, because any $A \in NP$ satisfies $A \leq_p$ 3SAT (the Cook-Levin theorem, from class) and 3SAT $\leq_p \neq$SAT (from homework).