Problem 5.3.2

Let $L_1$ and $L_2$ be semidecided by Turing machines $M_1$ and $M_2$. Construct a new two-tape Turing machine $M'$ which does the following on input $w$. First, it copies $w$ onto the second tape. It then runs $M_1$ on its first tape. If $M_1$ halts, $M'$ runs $M_2$ on the second tape, halting if $M_2$ halts. Then $L(M') = L(M_1) \cap L(M_2)$. For let $w \in L(M_1) \cap L(M_2)$. In this case, both $M_1$ and $M_2$ will halt on input $w$, so that $M'$ on input $w$ will see both simulations terminate and will itself halt, so that $w \in L(M')$. On the other hand, let $w \in L(M')$. In this case, $M'$ must have completed its algorithm on input $w$, in which it first saw $M_1$ halt on $w$ and then saw $M_2$ halt on $w$. Thus, since both $M_1$ and $M_2$ halt on input $w$, $w \in L(M_1) \cap L(M_2)$.

See the solution to problem 4.5.3 for closure under union.

Problem 5.4.1

(a) We note that given any Turing machine $M = (K, \Sigma, \delta, s, H)$, there are only $|K|(|k - 1)||\Sigma|^k - 1$ distinct configurations of $M$ that $M$ can be in that use fewer than $k$ tape squares. Thus, if we simulate $M$ on $w$ for $|K|(|k - 1)||\Sigma|^k - 1 + 1$ steps, either it must halt before finishing this many steps, repeat some configuration, or enter a configuration using $k$ or more squares. In the either of the first two cases, we know that the computation of $M$ on $w$ never uses $k$ tape squares, because it either halts or enters an infinite loop without having done so, so there can certainly be no future configuration in which it does so. In the third case, $M$ has manifestly used $k$ tape squares in its computation on $w$. Either way, after the end of this simulation of a finite number of steps, we are guaranteed an answer.

(b) If $f$ is recursive, then there is some Turing machine $N$ which computes $f$. Let us consider the two-tape Turing machine $M'$ which operates as follows: on input $w$ it lets $N$ calculate $f(|w|)$ on its second tape. $M'$ then runs the simulator Turing machine $S$ described in part (a) with input $(M, w, f(|w|))$ on its first tape, returning the same answer that $S$ returns.

(c) Suppose this problem were solvable. Then there would be some Turing machine $F$ (for “finite-checking”) which could solve it. Consider the Turing machine $G$ which executes the following algorithm when given “$M$” “$w$” as input:

1. Run $F$ on “$M$” and “$w$”. If $F$ says yes (so that $M$ uses an infinite amount of tape on input $w$), reject.

2. Otherwise, simulate $M$ on $w$ until it either halts or enters a configuration for the second time. If it halts, accept. Otherwise, reject.

If $F$ correctly solves the problem of whether $M$ uses a finite amount of tape on $w$, then $G$ correctly solves the halting problem. Since the halting problem is unsolvable, no such $G$, and hence no such $F$, can exist.

To see why $G$ works, suppose $M$ halts on $w$. Then it cannot use an infinite amount of tape, and so $F$ will reject “$M” “w”, so that $G$ will go on to step 2. In the course of its simulation, it will eventually see $M$ halt, so it will accept.

On the other hand, suppose $M$ fails to halt on $w$. If $M$ fails to halt and uses an infinite quantity of tape, $F$ will accept, leading $G$ to reject “$M” “w” in step 1. However, if $M$ fails to halt but uses only finitely many tape squares, as argued in part (a), $M$ must enter an infinite loop, and thus repeat a configuration, so that after a finite amount of simulation time, $G$ will reject “$M” “w”.


Problem 5.4.2

(a) This problem is undecidable. Suppose it were solvable; then some machine \( G \) would solve it. But given \( M \) and \( w \), we could feed \((M, w, h)\) to \( G \) (where \( h \) is the halting state of \( M \); if more than one, we can simply repeat our query several times), and return \( G \)'s answer, and this would constitute an effective procedure for deciding the halting problem.

(b) This problem is undecidable. Suppose it were solvable; then some machine \( G \) would solve it. But given \( M \) and \( w \), we could create a new machine \( M' \) as follows: \( M' \) is identical to \( M \) except that we add a new special state \( p \) such that if \( M' \) ever enters state \( p \), it proceeds to move to the left of its tape and simulate the action of \( M \) on the string \( w \) (\( M' \) will lay down a right endmarker as part of this simulation, moving the endmarker to the right as needed, erasing whatever formerly lay in the square to the right of the endmarker), and if \( M \) would ever halt, entering state \( q \). We note that there will be a configuration of \( M' \) with state \( p \) that yields a configuration with state \( q \) if and only if \( M \) halts on \( w \). If we then feed \((M', p, q)\) to \( G \), \( G \) will constitute an effective procedure for determining whether \( M \) halts on \( w \) for arbitrary \( M \) and \( w \), thus solving the halting problem.

(c) This problem is solvable. Examine \( \delta \) to see whether there are any rules of the form \( \delta(p, r) = (q, \tau) \). If so, then there is such a configuration, indeed an infinite number of them. \((p, \rightarrow \downarrow q)\) is one example. If not, then no configuration exists from which \( M \) can enter a configuration with state \( q \).

(d) This problem is undecidable. We reduce from the problem of determining whether an arbitrary Turing machine halts on the empty tape. Given the description of a machine \( M \), create the string \( "M^*" \cdot a \), where \( a \) is a symbol not in the alphabet of \( M \) and \( M^* \) is a Turing machine that is identical to \( M \) except that whenever it halts it also writes an \( a \). Clearly \( M^* \) writes an \( a \) when started on the empty tape if and only if \( M \) halts when started on the empty tape.

(e) This problem is solvable. Start by simulating \( M \) on the empty tape and stop when \( M \) either writes a nonblank symbol, enters a configuration for the second time, or moves more than \(|K|\) squares from the left end of the tape. If it has not written a nonblank symbol by this time, it never will. (If \( M \) has entered some configuration twice, it is clearly caught in an infinite loop. If it has moved more than \(|K|\) squares to the right, we must have \((q, \rightarrow \downarrow j \downarrow \uparrow) \cdot (q, \rightarrow \downarrow i \downarrow \uparrow) \) for some \( i < j \) by the pigeonhole principle, so that \( M \) is caught in an infinite loop moving inexorably to the right.

(f) This problem is solvable. Start by simulating \( M \) on input \( w \) and stop when either (1) \( M \) moves its head left, (2) \( M \) repeats a configuration, or (3) \( M \) moves its head to a position to the right of the end of the input and subsequently enters two distinct configurations \( C \) and \( C' \) with the same state and same symbol being currently scanned. If \( M \) has not moved left at this point, it never will. If (2) applies, then \( M \) is caught in an infinite stationary loop. If (3) applies but not (2), then \( M \) must periodically move to the right, but then from configuration \( C' \), \( M \) will follow the same sequence of transitions that led it from \( C \) to \( C' \), and has thus entered a rightwards moving infinite loop.
(g) This problem is undecidable. Suppose that there were a Turing machine $G$ which solved this problem. Given the description of a Turing machine $M$, give to $G$ the input $"M"^*"M_{\Sigma^*}$, where $M_{\Sigma^*}$ is a Turing machine which immediately halts on any input. $G$ would then accept this input if and only if $L(M) = \emptyset$. We have thus reduced the undecidable problem of determining whether an arbitrary Turing machine accepts any strings at all to the given problem, showing it to be undecidable as well.

(h) This problem is undecidable. Suppose that there were a Turing machine $G$ which solved this problem. Given the description of a Turing machine $M$, give to $G$ the input $"M"^*"M"$. Then $G$ will accept this input if and only if there is a string on which $M$ halts, so that we have reduced the undecidable problem of determining whether an arbitrary Turing machine accepts any strings at all to the given problem, showing it to be undecidable as well.

(i) This problem is undecidable. Suppose that there were a Turing machine $G$ which solved this problem. Given $"M"^*$, give to $G$ the input $"M_{\star}"$, where $M_{\star}$ is the Turing machine which erases its input and then simulates the action of $M$ on the empty tape, halting if and when $M$ would have. If $e \in L(M)$, then $M_{\star}$ halts on all strings and thus semidecides an infinite language. If $e \not\in L(M)$, then $M_{\star}$ halts on no input and thus semidecides the empty language, which is definitely finite. By reversing the answer that $G$ gives us, then, we have reduced the undecidable problem of determining whether an arbitrary Turing machine $M$ halts on the empty string to the given problem, showing it to be undecidable as well.

Problem 5.5.1

First, we show that if $M$ is a pushdown automaton with one state, $M$ accepts all strings if and only if it accepts all strings of length one. Suppose that $M$ accepts all strings of length one. Because $M$ has only one state, we know that $F = \{s\}$, so that $(s, \sigma, e) \vdash_M^* (s, e, e)$ for each $\sigma$ in $\Sigma$. Then, given a string $w \in \Sigma^*$, we can easily show that $(s, w, e) \vdash_M^* (s, e, e)$ by induction on $|w|$. If $|w| = 0$, then $w = e$ and the result follows from the reflexivity of $\vdash_M^*$. If $|w| = n + 1$, let $w = \sigma x$, for $|x| = n$. Then $(s, \sigma x, e) \vdash_M^* (s, x, e)$ by assumption and $(s, x, e) \vdash_M^* (s, e, e)$ by the inductive hypothesis. Thus $M$ accepts $w$. On the other hand, if $M$ does not accept some given string of length one, then it certainly does not accept all strings.

The problem of determining whether $M$ accepts all strings of length one, however, is decidable. By converting $M$ to a context-free grammar $G$, converting $G$ to an equivalent grammar $G'$ in Chomsky Normal Form, and then running the dynamic programming algorithm on $G'$ and each string of length one.

Problem 5.7.2

Suppose $L$ is enumerated by some Turing machine $M$. Then we construct a three-tape Turing machine $M'$ which enumerates $L$ without repetitions as follows. $M'$ simulates $M$ on its second tape and keeps a list of all strings that $M$ has generated on its third tape. Whenever $M$ enumerates a string that does not appear in the master list on the third tape, $M'$ adds that string to the list and enumerates the string on the first tape. Thus $M'$ produces every string that $M$ produces, and without duplication.
Problem 5.7.4

(a) In trying to reconstruct the derivation of a string \( w \) in a context-sensitive grammar, consider the set of strings of length \(|w|\) or less over \( V^* \) as the vertices in a directed graph where there is an edge from \( x \) to \( y \) if and only if \( x \Rightarrow y \). This is a finite graph and each edge can be computed (there are only finitely many rules which could be applied at any of finitely many places), so we can compute reachability in a finite amount of time and thus completely determine \( \Rightarrow^* \). We then just need to look and see whether \( S \Rightarrow^* \). It suffices to consider only strings of length \(|w|\) or less since if \(|x| > |y|\) then no series of applications of rules to \( x \) can ever result in a string of length \(|y|\) or less.

(b) The condition that \( w \neq e \) automatically means that \(|uAv| \leq |uwv|\), so that if a grammar contains only rules of the form \( uAv \rightarrow uwv \) for \( w \neq e \), it is context-sensitive.

In the other direction, suppose we have a context-sensitive grammar with \( r \) rules, each of the form \( u \Rightarrow v \) for \(|u| \leq |v|\). Consider the \( m \)-th rule. Let it have the form \( \sigma_1 \sigma_2 \ldots \sigma_n \Rightarrow \tau_1 \tau_2 \ldots \tau_n \tau_{n+1} \ldots \tau_{n+k} \) (for some \( k \geq 0 \)). Introduce new nonterminals \( A_1^m, \ldots, A_n^m, B_1^m, \ldots, B_k^m \) and replace the rule with the following rules:

- \( A_1^m \ldots A_{n-1}^m \sigma_i \sigma_{i+1} \ldots \sigma_n \rightarrow A_1^m \ldots A_{i-1}^m \sigma_i \sigma_{i+1} \ldots \sigma_n \) for each \( 1 \leq i \leq n \).

- \( \tau_1 \ldots \tau_{i-1} A_i^m A_{i+1}^m \ldots A_n^m \rightarrow \tau_1 \ldots \tau_{i-1} \tau_i A_{i+1}^m \ldots A_n^m \) for each \( 1 \leq i < n \).

- \( \tau_1 \ldots \tau_{n-1} A_n^m \rightarrow \tau_1 \ldots \tau_{n-1} \tau_n B_1 \)

- \( B_i^m \rightarrow \tau_i B_{i+1}^m \) for each \( 1 \leq i < k \).

- \( B_k^m \rightarrow \tau_k \).

Each of these rules obeys the criterion for being in the correct form, and it can easily be checked that they must be applied strictly in order in which they are listed in order for the derivation not to block. As for the condition that each \( A \) in such a rule must be a nonterminal, this can be enforced by making every symbol of the original grammar a nonterminal, adding a new terminal for each old terminal, and adding the appropriate rule \( A \rightarrow a \) for each pair of old and new terminal symbols.

(c) Given a context-sensitive grammar \( G \), we can construct a nondeterministic TM which will semidecide \( L(G) \) by having \( M \) carry out backwards derivations in \( G \). That is, given an input string \( w \), \( M \) will repeatedly undo rule applications of \( G \) on \( w \), hoping eventually to arrive at the start symbol by itself. If \( M \) succeeds, then it will halt, indicating that there existed some derivation \( S \Rightarrow^* w \).

More precisely, \( M \) has the entire ruleset of \( G \) built into its transition function. It then iterates through the following loop indefinitely (each iteration starts and ends with a configuration of the form \((s,v\upharpoonright x)\) for some string \( x \):

1. If \( x = S \), halt.

2. Nondeterministically pick a rule of \( G \) to “unapply.”

3. Nondeterministically pick a location in \( x \) at which to “unapply” this rule.

4. If the portion of \( x \) starting at the location picked in step 3 does not match the right-hand-side of the rule picked in step 2, enter an infinite loop.

5. If the selected rule does match \( x \) at the selected location, replace the matched portion of \( x \) by the left-hand-side of the rule.

6. If the left-hand-side of the rule was shorter than the right-hand-side, shift the trailing portion of \( x \) left to remove any internal spaces. [Because of the phrasing of the problem, a lot of people were careful to use a symbol other than the blank for these internal temporary “blanks”, so that the head never “visited” internal blanks.]

7. Return the tape head to the leftmost blank.
Suppose there exists some derivation $S \Rightarrow_\ast^* w$ of $w$ in $G$. Then every intermediate string in this derivation has length at most $|w|$. This fact follows (by an easy induction on the length of a derivation) from the property that $|u| \leq |v|$ for any rule in a context-sensitive grammar. $M$ will have a computation in which it moves through the strings of this derivation, in reverse order, so it will have a halting computation. Contrariwise, any halting computation of $M$ must have halted in step 1 after some finite number of loop iterations; we can string together the value of $x$ after each iteration to reconstruct a derivation of $w$ in $G$. 