CS4124
Answers to Selected Problems of HW #4

Problem 2.5.4

(a) If there are two distinct $p, p' \in K$ satisfying $(q, e, w) \vdash_M^* (p, w, e)$ and $(q, e, w) \vdash_M^* (p', w, e)$, then there must be two distinct computations of M leading to these states. Since it cannot be the case that $(p, w, e) \vdash_M^* (p', w, e)$, or vice versa (because neither is related under $\vdash_M$ to any configurations at all), unless $p = p'$, they must actually diverge from each other at some point. Let $(r, x, \sigma y)$ be the last configuration on which these computations agree – since they agree on at least one and there are only finitely many configurations in a computation, such a configuration must exist, and the last element in the configuration triple cannot be e or no configurations, certainly not two distinct configurations, could follow this one. There are now two possibilities. Either $\delta(r, \sigma) = (s, \rightarrow)$ or $\delta(r, \sigma) = (s, \leftarrow)$ for some state s. In the former case, $(r, x, \sigma y) \vdash_M (s, x\sigma, y)$. In the latter, $(r, x, \sigma y) \vdash (s, x', \tau\sigma y)$ (provided that $|x| > 0$. If this assumption is false, though, $(r, x, \sigma y)$ is related to no other configurations by $\vdash_M$, and the computation halts here, already a violation of our assumptions). In each case, however, the next configuration is uniquely determined, violating our claim that the two computations diverged immediately after $(r, x, \sigma y)$. Thus our initial assumption that there were such distinct p and p' was false.

(b) It will suffice to show that if M accepts wu then M accepts vu. If M accepts wu, then by definition $(s, e, wu) \vdash_M^* (q, wu, e)$ for some $q \in F$. During this computation, M must pass through a configuration of the form (p, w, u) at least once, and possibly more than once. Let us therefore break up its computation into

$$(s, e, wu) \vdash_M^* (p_0, w, u) \vdash_M^+ (p_1, w, u) \vdash_M^+ \ldots \vdash_M^+ (p_k, w, u) \vdash_M^* (q, wu, e),$$

where $k \geq 0$ and the p_i are all the (not necessarily distinct) states of M for which the computation is in a configuration reading the first symbol of u. Because p_0 is the first such state, all the computation before (p_0, w, u) is independent of u, so that $(s, e, w) \vdash_M^* (p_0, w, e)$. This tell us that $\chi_w(s) = p_0$. Therefore, $\chi_v(s) = p$, so $(s, e, v) \vdash_M^* (p, v, e)$. If $u = e$, we are done, so that for the remainder of the proof, we will be able to safely assume that $u = \sigma u'$ for some symbol σ and string u'.

We now prove by induction on k that $(p_0, v, u) \vdash_M^* (p_k, v, u)$. For $k = 0$ this fact is trivial, since they are the same configuration. Assume this holds for $k \leq n$, so that $(p_0, v, u) \vdash_M^* (p_n, v, u)$. It will suffice to prove that $(p_n, v, u) \vdash_M^+ (p_{n+1}, v, u)$, and the desired result will hold by the transitivity of $\vdash_M^*$. There are two possibilities for $\delta(p_n, \sigma)$: either $(r, \rightarrow)$ or $(r, \leftarrow)$ for some state r.

If a left move is indicated, between the configuration $(p_n, w, \sigma u')$ and $(p_{n+1}, w, \sigma u')$ in its computation on wu, M is at no time scanning any symbol of u (since it must first be scanning the σ, which it can first do again when it reaches $(p_{n+1}, w, \sigma u')$). Thus, for our purposes, only σ, the first symbol of u, matters – and that only when it indicates this first, leftward move. That is, since $(p_{n+1}, w, \sigma u')$ is the next time M scans σ, $\theta_w(p_n, \sigma) = p_{n+1}$, so therefore $\theta_v(p_n, \sigma) = p_{n+1}$, so that $(p_n, v, \sigma) \vdash_M^+ (p_{n+1}, v, \sigma)$. By entirely analagous reasoning to that above, the portion of u after σ cannot affect this portion of the computation, so $(p_n, v, \sigma u') \vdash (p_{n+1}, v, \sigma u')$.

If a right move is indicated, between the configurations $(p_n, w, \sigma u')$ and $(p_{n+1}, w, \sigma u')$ in its computation on wu, M is at no time scanning any symbol of w (since it must first scan the σ again, which it can only do again when it reaches $(p_{n+1}, w, \sigma u')$). Thus, this phase of the computation is independent of w, and could equally well take place with v in w's place, so that $(p_n, v, \sigma u') \vdash_M^+ (p_{n+1}, v, \sigma u')$. By entirely similar reasoning, the computation from (p_{n+1}, w, u) to (q, wu, e) depends only on u and not on w (in fact, it depends only on χ_u), so we can in the same fashion replace w with v to obtain $(p_{n+1}, v, u) \vdash_M^* (q, vu, e)$. Putting everything together gives us $(s, e, vu) \vdash_M^* (q, vu, e)$, so M accepts v.

(c) Given a two-way deterministic finite automaton M, define a binary relation on strings in Σ^* by $R(x,y)$ if $\chi_x = \chi_y$ and $\theta_x = \theta_y$. R is clearly an equivalance relation, and therefore can be used to partition Σ^*. By part (b), for any string u, if $R(x,y)$ then $xu \in L(M)$ if and only if $yu \in L(M)$. Note that the converse need not be true – we could have x and y equivalent with resepect to M but with different θ functions. What matters is that the classes under R are a refinement of the classes under $\approx_{L(M)}$, so that $L(M)$ has at most as many classes as there are equivalence classes under R. However, note that χ is a function from a set of $|K|$ elements to a set of $|K|+1$, and that there are therefore at most $(|K|+1)^{|K|}$ different possible such functions. On the other hand, θ is a function from a st of $|K||\Sigma|$ elements to a set of $|K|+1$ elements, and so there can be at most $(|K+1|)^{|K||\Sigma|}$ such functions. Taken together, this allows only $(|K+1|)^{|K|(|\Sigma|+1)}$ different equivalence classes for R, which is finite, so by the Myhill-Nerode Theorem, $L(M)$ is accepted by some deterministic finite automaton.

(d) Given $M = (K, \Sigma, \delta, s, F)$, a deterministic two-way finite automaton. we construct a one-way deterministic finite automaton $M' = (K', \Sigma', \delta', s', F')$ whose states are equivalence classes (χ_w, θ_w), as in c above. The invariant this automaton will maintain is that, after consuming input x, the current state's pair will be (χ_x, θ_x). To compute one entry in a χ or θ function requires time at most $|w||K|$, since a configuration will be determined by state and position within the string only, and since any duplication means entry into an infinite loop, thus forcing the value being computed to be t. Thus, given a string w, the entire χ table for w can be computed in time $|w||K|^2$ and the entire θ table in time $|w||K|^2|\Sigma|$. Since we can do our computations using representative w, the longest such representives we will need will be of length $|K|$, by a pumping argument, making these times $|K|^3$ and $|K|^3|\Sigma|$. In the worst possible case, every possible pair of χ and θ will need to be used as a state. By the combinatorics of the previous problem, this yields $\mathcal{O}(|K|^{|K||\Sigma|})$ states, each of which takes time $|\Sigma||K^3|$ to calculate, for a total time of $\mathcal{O}(|\Sigma||K|^{3+|K||\Sigma|})$.

(e) Let $K = \{q_0\} \cup \{y_1, y_2 \ldots y_{n-1}, y_n\} \cup \{n_1, n_2, \ldots n_{n-1}, n_n\} \cup \{b_2, \ldots b_{n-1}\}$/ $\Sigma = \{a, b\}$, $s = q_0$, and $F = q_n$. We let δ be given as in this table:

q	σ	$\delta(q,\sigma)$
q_0	a	$(y_1, \rightarrow)$
q_0	b	$(n_1, \rightarrow)$
$y_i : 0 < i < n$	a	$(y_{i+1}, \rightarrow)$
$y_i : 0 < i < n$	b	$(y_{i+1}, \rightarrow)$
$n_i : 0 < i < n$	a	$(n_{i+1}, \rightarrow)$
$n_i : 0 < i < n$	b	$(n_{i+1}, \rightarrow)$
y_n	a	$(b_{n-1}, \leftarrow)$
y_n	b	$(b_{n-1}, \leftarrow)$
n_n	a	$(b_{n-1}, \leftarrow)$
m_n	b	$(b_{n-1}, \leftarrow)$
$b_i : 2 < i < n$	a	$(b_{i-1}, \leftarrow)$
$b_i : 2 < i < n$	b	$(b_{i-1}, \leftarrow)$
b_2	a	$(q_0, \leftarrow)$
b_2	b	$(q_0, \leftarrow)$

M repeatedly scans a symbol and then move right n symbols. If this brings it to the end of the input, it must halt – and it will be in the accepting state y_n if and only if the symbol n symbols to the left was an a. Otherwise, it returns $n-1$ symbols to the left and tries again.

(f) The argument can indeed be so extended – χ and θ need to be modified to work with sets of states, rather than individual states, but at that point the the proof can be carried ahead as before.

Problem 2.6.3

Throughout this problem, we will use the terminology of string matching to say that there is a *match* of x in w *at* a given symbol σ of w if x is a substring of w and the occurrence of σ we have designated is the first symbol of x as a substring of w.

(a) Suppose $w \in L(M_x)$. Then there is some computation $(q_0, w) \vdash_M^* (q_n, e)$ of M which leads to acceptance. Since there is only one transition out of q_0 which does not lead to q_0, M must take this transition at some point, so that $(q_0, ua_1v) \vdash_M^* (q_0, a_1v) \vdash_M (q_1, v) \vdash_M^* (q_n, e)$. But there is only one transition out of q_1, so M must take it, which means the first symbol of v is a_2, and $(q_1, a_2v') \vdash_M (q_2, v')$. This reasoning can be repeated inductively, until we have forced a transition into q_n on the symbol a_n, and we have that $w = ua_1a_2 \ldots a_nz$ for some $u, z \in \Sigma^*$, so $x = a_1a_2 \ldots a_n$ is a substring of w.

On the other hand, suppose $w = ua_1a_2 \ldots a_nz$ for some $u, z \in \Sigma^*$. Then

$$(q_0, ua_1a_2 \ldots a_nz) \vdash_M^* (q_0, a_1a_2 \ldots a_nz) \vdash_M (q_1, a_2 \ldots a_nz) \vdash_M \ldots \vdash_M (q_n, z) \vdash_M^* (q_n, e)$$

so that $w \in L(M_n)$.

(b) Let us consider the equivalence classes under $\approx_L$ for this language. Assign to each string w the string $p(w)$ which is the longest prefix of x which is a suffix of w. Let $p(w) = e$ for strings $w \in L$. Our claim is that there is one class per possible prefix. For strings in L, this claim is trivial, as if $w \in L$, then $wz \in L$ for any string z.

Suppose $w \notin L$, $p(w) = p(y)$, and that $wz \in L$. Then $w = w'p(w)$ for some $w' \in \Sigma^*$. There cannot be a match of x in wz in w', because that match would need to extend into z, and thus $p(w)$ would not be the longest possible prefix. Thus the match must be contained entirely in $p(w)z$, which is the same string as $p(y)z$, ao $yz \in L$ also. The same reasoning shows that if $y \in L$ then $w \in L$.

Suppose on the other hand that $p(w) \neq p(y)$, and suppose without loss of generality that $|p(w)| > |p(y)|$. If u is the string such that $x = p(w)u$, then $wu \in L$. However, $yu \notin L$. By the reasoning above, there can be no match of x in yu which starts before $p(y)$. But then $|p(y)u| < |x|$, so there cannot possibly be a match starting any later.

By raw brute force, M' can be constructed in time $\mathcal{O}(n^3|\Sigma|)$. We work with a representative string for each state – that string is just the appropriate prefix of x. For each transition, we add the appropriate symbol, and then scan the string backwards to see whether it matches each appropriate prefix. Since $|\delta| = n|\Sigma|$, the overall construction takes time $\mathcal{O}(n^3|\Sigma|)$. But see below for techniques to improve this algorithm.

(c) Let M'' be the same as M with a few modifications. We add one e-transition from each state as follows: from state q_i, add a transition to the state whose number is $|p(a_2a_3 \ldots a_i)|$, that is, to the state whose string equivalence class is that of the longest prefix of x that is a proper suffix of the string $a_1 \ldots a_i$ (the i-symbol prefix of x).

Suppose $w \in L(M)$. Then $(q_0, w) \vdash_M^* (q_n, e)$. But M'' retains every transition of M, so that $\vdash_M \subseteq \vdash_{M''}$, and $\vdash_M^* \subseteq \vdash_{M''}^*$. Thus $(q_0, w) \vdash_{M''}^* (q_n, e)$ and $w \in L(M'')$.

On the other hand, suppose $w \in L(M'')$. Then $(q_0, w) \vdash_{M''}^* (q_n, e)$. We claim that any e-transitions made during this computation can be removed by an inductive process. Suppose that $(q_0, yz) \vdash_{M''}^* (q_i, z) \vdash_{M''}$

(q_j, z). where the first portion includes no e-transitions. Then, if $y = y'p(y)$, we can decompose this computaton into $(q_0, y'p(y)z) \vdash^*_{M''} (q_0, p(y)z) \vdash^*_{M''} (q_i, z)$. We can, however, break up $p(y)$ into uv, where v is the longest prefix of x that is a proper sufffix of $p(y)$. By our definitions above, the e-transition from q_i must be such that q_j is the state corresponding to v. Thus, we can write $(q_0, y'uvz) \vdash^*_M (q_0, uvz)$, because this looping involves no e-transitions; we can write $(q_0, vz) \vdash^*_M (q_0, vz)$ because this is further looping in q_0 that is always permitted; we can write $(q_0, vz) \vdash^*_M (q_j, z)$ because this just involves the forward transitions on $a_1, a_2, \ldots, a_j$ that are permissible because $v = a_1 a_2 \ldots a_j$. Putting it all together, we have $(q_0, yz) \vdash^*_M (q_j, z)$ (since $y = y'uv$). We can thus repeatedly apply this process to remove any e-transitions om an accepting computation, so that $(q_0, w) \vdash^*_M (q_n, e)$ and $w \in L(M)$.

(d) Consider the following algorithm for simulating M'' on input w. If the next symbol σ of w matches the forward transition from the current state, take that transition, otherwise take the e-transition and *do not consume* the σ. Note that each decision point requires only a single binary comparison, and that the entire automaton requires only $\mathcal{O}(n)$ space to describe.

The key observation is that in order to take an e-transition backwards, we must have made some forward transition previously. Thus, there cannot be more e-transitions made than forward transitions. Since each forward transition requires consuming a symbol, there can be at most $|w|$ of them, so there are at most $2|w|$ transitions total in processing w. Thus, the simulation takes time $\mathcal{O}(|w|)$ and space $\mathcal{O}(n)$, even as Σ is allowed to vary.

(e) M'' can easily be computed by a brute-force method. The forward transitions and those of q_0 and q_n are trivial to determine. For the backward transitions, we need to know, for each prefix p of x (of which there are at most n), the longest prefix of x (n possibilities) which is a proper prefix of p (brute-force string comparison taking at most $|p| < n$ steps). By just trying all possibilities, the overall time for this algorithm is $\mathcal{O}(n^3)$.

(f) As noted in the problem, we are trying to compute $f(j)$, for $1 \leq j \leq n-1$, given the values of $f(i)$ for all $i < j$. $f(1) = 0$ as a base case. Let us think of this problem in the following manner:

A prefix of x that is a suffix of $a_1 a_2 \ldots a_j$ can thought of as some prefix of x that consists of some suffix of $a_1 a_2 \ldots a_{j-1}$ followed by a_j. Thus, if $a_1 a_2 \ldots a_i$ is that prefix of $a_1 a_2 \ldots a_{j-1}$, then we have $a_j = a_{i+1}$. Thus, we should look at the various possible such prefixes $a_1 \ldots a_i$ and search for the largest i such that both $a_1 a_2 \ldots a_i$ is a suffix of $a_1 a_2 \ldots a_{j-1}$ and $a_{i+1} = a_j$. If we do this in order from largest to smallest, we will be able to stop searching as soon as some value of i passes both tests.

We have, however, an effective way to search through the set of all proper prefixes of w that are suffixes of $a_1 a_2 \ldots a_{j-1}$, in order from longest to shortest. $f(j-1)$ designates the longest. If this fails, $f(f(j-1))$ names the next longest, and so on until we arrive at $f^k(j-1) = 0$, at whcih point we can try e and will then have exhausted our possible prefixes. Thus, if we call our "longest prefix" variable l, then we start by setting $l = f(j-1)$. We then repeatedly see whether $a_{l+1} = a_j$. If yes, we stop, and set $f(j) = l+1$. If not, we replace l with $f(l)$ and check again. If at any point we try to take $f(l)$, then the longest prefix of x which is a suffix of $a_1 a_2 \ldots a_j$ is e, and we set $f(j) = 0$.

At the end of each stage, either $f(j) = l$ or $f(j) = l+1$. On the other hand, each stage, after incrementing j, opens by setting $l = f(j-1)$, so that l in effect carries over from each loop to the next. Note that each comparison operation results in either the termination of a stage (and thus an increment) or in some decrement of l. We cannot decrement l more times than it is incremented, and it can be incremented at most n times. Thus, there are at most $2n$ comparisons made, so that this algorithm runs in time $\mathcal{O}(n)$.

(g) M is given by $(K, \Sigma, \Delta, s, F)$, where

$$\begin{aligned} K &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \\ \Sigma &= \{a, b\} \\ s &= q_0 \\ F &= \{q_7\}, \end{aligned}$$

and Δ is as given in the following table:

(q_i, σ, q_f)
(q_0, a, q_0)
(q_0, a, q_1)
(q_0, b, q_0)
(q_1, a, q_2)
(q_2, b, q_3)
(q_3, b, q_4)
(q_4, a, q_5)
(q_5, a, q_6)
(q_6, b, q_7)
(q_7, a, q_7)
(q_7, b, q_7)

On input *aababbaaabbaaabbaabb*, M can compute as follows:

$$
\begin{aligned}
(q_0, aababbaaabbaaabbaabb) &\vdash_M (q_0, ababbaabbaaabbaabb) \\
&\vdash_M (q_0, babbaabbaaabbaabb) \\
&\vdash_M (q_0, abbaabbaaabbaabb) \\
&\vdash_M (q_0, bbaabbaaabbaabb) \\
&\vdash_M (q_0, baabbaaabbaabb) \\
&\vdash_M (q_0, aabbaaabbaabb) \\
&\vdash_M (q_0, abbaaabbaabb) \\
&\vdash_M (q_0, bbaaabbaabb) \\
&\vdash_M (q_0, baaabbaabb) \\
&\vdash_M (q_0, aaabbaabb) \\
&\vdash_M (q_0, aabbaabb) \\
&\vdash_M (q_1, abbaabb) \\
&\vdash_M (q_2, bbaabb) \\
&\vdash_M (q_3, baabb) \\
&\vdash_M (q_4, aabb) \\
&\vdash_M (q_5, abb) \\
&\vdash_M (q_6, bb) \\
&\vdash_M (q_7, b) \\
&\vdash_M (q_7, e)
\end{aligned}
$$

The equivalent deterministic finite automaton M' is given by $(K, \Sigma, \delta', s, F)$ (only δ has changed from M) , where δ is given by the following table:

q	σ	$\delta(q,\sigma)$
q_0	a	q_1
q_0	b	q_0
q_1	a	q_2
q_1	b	q_0
q_2	a	q_2
q_2	b	q_3
q_3	a	q_1
q_3	b	q_4
q_4	a	q_5
q_4	b	q_0
q_5	a	q_6
q_5	b	q_0
q_6	a	q_2
q_6	b	q_7
q_7	a	q_7
q_7	b	q_7

On input *aababbaaabbaaabbaabb*, M can compute as follows:

$$
\begin{aligned}
(q_0, aababbaaabbaaabbaabb) &\vdash_{M'} (q_1, ababbaabbaaabbaabb)\\
&\vdash_{M'} (q_2, babbaabbaaabbaabb)\\
&\vdash_{M'} (q_3, abbaabbaaabbaabb)\\
&\vdash_{M'} (q_1, bbaabbaaabbaabb)\\
&\vdash_{M'} (q_0, baabbaaabbaabb)\\
&\vdash_{M'} (q_0, aabbaaabbaabb)\\
&\vdash_{M'} (q_1, abbaaabbaabb)\\
&\vdash_{M'} (q_2, bbaaabbaabb)\\
&\vdash_{M'} (q_3, baaabbaabb)\\
&\vdash_{M'} (q_4, aaabbaabb)\\
&\vdash_{M'} (q_5, aabbaabb)\\
&\vdash_{M'} (q_6, abbaabb)\\
&\vdash_{M'} (q_2, bbaabb)\\
&\vdash_{M'} (q_3, baabb)\\
&\vdash_{M'} (q_4, aabb)\\
&\vdash_{M'} (q_5, abb)\\
&\vdash_{M'} (q_6, bb)\\
&\vdash_{M'} (q_7, b)\\
&\vdash_{M'} (q_7, e)
\end{aligned}
$$

M'', the useful equivalent nondeterministic finite automaton, is given by $(K, \Sigma, \Delta'', s, F)$ (all components are the same as in M except for Δ''), where Δ'' is given by the following table:

(q_i, σ, q_f)
(q_0, a, q_1)
(q_0, b, q_0)
(q_1, a, q_2)
(q_1, e, q_0)
(q_2, e, q_1)
(q_2, b, q_3)
(q_3, e, q_0)
(q_3, b, q_4)
(q_4, a, q_5)
(q_4, e, q_0)
(q_5, a, q_6)
(q_5, e, q_1)
(q_6, e, q_2)
(q_6, b, q_7)
(q_7, a, q_7)
(q_7, b, q_7)

On input ***aababbaaabbaaabbaabb***, M can compute as follows:

$$
\begin{aligned}
(q_0, aababbaaabbaaabbaabb) &\vdash_{M''} (q_1, ababbaabbaaabbaabb) \\
&\vdash_{M''} (q_2, babbaabbaaabbaabb) \\
&\vdash_{M''} (q_3, abbaabbaaabbaabb) \\
&\vdash_{M''} (q_0, abbaabbaaabbaabb) \\
&\vdash_{M''} (q_1, bbaabbaaabbaabb) \\
&\vdash_{M''} (q_0, bbaabbaaabbaabb) \\
&\vdash_{M''} (q_0, baabbaaabbaabb) \\
&\vdash_{M''} (q_0, aabbaaabbaabb) \\
&\vdash_{M''} (q_1, abbaaabbaabb) \\
&\vdash_{M''} (q_2, bbaaabbaabb) \\
&\vdash_{M''} (q_3, baaabbaabb) \\
&\vdash_{M''} (q_4, aaabbaabb) \\
&\vdash_{M''} (q_5, aabbaabb) \\
&\vdash_{M''} (q_6, abbaabb) \\
&\vdash_{M''} (q_2, abbaabb) \\
&\vdash_{M''} (q_1, abbaabb) \\
&\vdash_{M''} (q_2, bbaabb) \\
&\vdash_{M''} (q_3, baabb) \\
&\vdash_{M''} (q_4, aabb) \\
&\vdash_{M''} (q_5, abb) \\
&\vdash_{M''} (q_6, bb) \\
&\vdash_{M''} (q_7, b) \\
&\vdash_{M''} (q_7, e)
\end{aligned}
$$

Problem 3.1.3

(a) $G = (V, \Sigma, R, S)$, where:

$$\begin{aligned} V &= \{a, b, S\} \\ \Sigma &= \{a, b\} \\ R &= \{S \to aSa, \\ &\quad S \to bSb, \\ &\quad S \to c\}. \end{aligned}$$

(b) $G = (V, \Sigma, R, S)$, where:

$$\begin{aligned} V &= \{a, b, S\} \\ \Sigma &= \{a, b\} \\ R &= \{S \to aSa, \\ &\quad S \to bSb, \\ &\quad S \to e\}. \end{aligned}$$

(c) $G = (V, \Sigma, R, S)$, where:

$$\begin{aligned} V &= \{a, b, S\} \\ \Sigma &= \{a, b\} \\ R &= \{S \to aSa, \\ &\quad S \to bSb, \\ &\quad S \to a, \\ &\quad S \to b, \\ &\quad S \to e\}. \end{aligned}$$

Problem 3.1.4

$G = (V, \Sigma, R, S)$, where:

$$\begin{aligned} V &= \{S, a, b, (,), \cup, ^*, \emptyset\} \\ \Sigma &= \{a, b, (,), \cup, ^*, \emptyset\} \\ R &= \{S \to a, \\ &\quad S \to b, \\ &\quad S \to \emptyset, \\ &\quad S \to (SS), \\ &\quad S \to (S \cup S), \\ &\quad S \to S^*\}. \end{aligned}$$

Problem 3.1.10

(a)

$$\begin{aligned} K &= \{S, abA, bA, A, baB, aB, B, aS, bS, b, f\} \\ \Sigma &= \{a, b\} \\ s &= S \\ F &= \{f\} \\ \Delta &= \{\text{(S,e,abA), (abA,a,bA), (bA,b,A), (S,e,B), (S,e,baB) (baB,b,aB),} \\ &\quad \text{(aB,a,B), (S,e,f), (A,e,bS), (bS,b,S) (B,e,aS), (aS,a,S), (A,e,b),} \\ &\quad \text{(b,b,e)}\} \end{aligned}$$

$$S \Rightarrow abA \Rightarrow abbS \Rightarrow abbB \Rightarrow abbaS \Rightarrow abba$$

$(S, abba) \vdash_M (abA, abba) \vdash_M (bA, bba) \vdash_M (A, ba) \vdash_M (bS, ba) \vdash_M$
$(S, a) \vdash_M (B, a) \vdash_M (aS, a) \vdash_M (S, e) \vdash_M (f, e)$

(b) Let $G = (V, \Sigma, R, S)$. Without loss of generality, assume that no rule of G produces more than one terminal; that is, that $R \subseteq (V - \Sigma) \times (\Sigma \cup \{e\})(V - \Sigma \cup \{e\})$. Any right-linear grammar can easily be converted to this form by introducing new, "dummy" nonterminals (in the fashion of the removal of long rules in the conversion to Chomsky Normal Form in chapter 3.6). Given such a grammar, create a nondeterministic finite automaton $M = (V - \Sigma \cup \{f\}, \Sigma, \Delta, S, \{f\})$, where

$$\begin{aligned} \Delta &= \{(A, \sigma, B) : (A, \sigma B) \in R, \sigma \in \Sigma \cup e\} \\ &\quad \cup \{(A, \sigma, f) : (A, \sigma) \in R, \sigma \in \Sigma \cup e\} \end{aligned}$$

On the other hand, suppose we have a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$. Define a context-free grammar $G = (K \cup \Sigma, \Sigma, R, s)$, where

$$R = \{p \rightarrow \sigma q : \delta(p, \sigma) = q\} \cup \{q \rightarrow e : q \in F\}$$

These constructions express the idea that there is exactly as much information in a single non-terminal which produces terminals from left to right as there is in a single state of a machine which consumes input from left to right. They can be proven correct by straightforward induction on the length of a derivation/computation.

(c) This problem is functionally identical to part (b), except that the strings are generated from right to left. Since regular and context-free languages are closed under reversal, given a left-linear grammar, we reverse it into a right-linear one, then carry out the above construction, and finally reverse the resulting automaton. Given an automaton, reverse it, generate a right-linear grammar, and then reverse that grammar to yield a left-linear one.

(d) $L(G)$ is not necessarily regular. Consider the grammar $G = (V, \Sigma, R, S)$ given by

$$\begin{aligned} V &= \{a, b, S, T\} \\ \Sigma &= \{a, b\} \\ R &= \{S \rightarrow aT, \\ &\quad S \rightarrow e, \\ &\quad T \rightarrow Sb\} \end{aligned}$$

G generates the language $\{a^n b^n : n \in \mathbf{N}\}$, which we already know not to be regular.