The regular expression describing the strings accepted is
\[ R_{13}^3 = 1 + 0(0+10)^* 11 + 1 + 0(0+10)^* 11 \left[ (0+1+00)(0+10)^* 11 \right] (0+1+00)(0+10)^* 11 \]
Let $L_1 = \{0^{n}10^{n+1} | n \geq 1\}$ and let $L_2 = \{0^{n}10^{2n}1 | n \geq 1\}$. Express the set $L$ consisting of strings of the form $010010001000001...$ where the number of zeros in successive blocks increases as $1, 2, 4, 5, 10, 11, 22, 23,...$ i.e., alternate adding one and multiplying by two. The number of blocks of zeros can be even or odd.

A correct solution is:

$$(L_1)^* (0^*1^*) \cap 01(L_2)^* (0^*1^*)$$

Both sides need to be followed by $(0^*1^*)$ in order to allow strings to have an even or odd number of zeros.
4.2.1

a) aabbaa

b) baababbaa

c) The language of the regular expression a(ab)*ba

d) a + abba

e) Each b must come from either 1 or 2. However, if the first b comes from 2 and the second comes from 1, then they will both need the a between them as part of h(2) and h(1), respectively. Thus, the inverse homomorphism consists of the strings {110, 102, 022}

f) The language of the regular expression 02* + 1*0. As in part (e), each b can come from either 1 or 2.
We can prove that $L/a$ is a regular language as well by starting with the homomorphism:

$$h_1(\hat{a}) = a \quad h_1(\hat{b}) = b$$
$$h_1(a) = a \quad h_1(b) = b$$

We can then apply $h_1^{-1}(L)$, the inverse homomorphism of $h_1$ to the language. From this, we get the original string with possible hats over every letter in the alphabet. We want to remove the last letter so we take the intersection of the result of the inverse homomorphism with the regular expression:

$$R = (a + b)^*\hat{a}$$

to yield:

$$L' = h_1^{-1}(L) \cap R$$

This new expression, $L'$, is the original $L$ with the last letter having a hat. From here, we can define a second homomorphism to get rid of the hat:

$$h_2(\hat{a}) = \emptyset$$
$$h_2(a) = a \quad h_2(b) = b$$

We can then morph $L'$ into $L/a$ by removing the last letter. We do this by removing all the letters with hats (since only the last letter has a hat):

$$L/a = h_2(L')$$
Problem 4.2.6 by machine construction

a) Let \( L \) be a regular language, and \( M_L \) a DFA for \( L \). We will construct a DFA \( M' \) for \( \text{min}(L) \).

Observe that if \( w \in \text{min}(L) \), then \( M_L \) on input \( w \) ends on some final state. Furthermore, no previous state along the path taken by \( w \) through \( M_L \) is allowed to be a final state. So to construct \( M' \), we simply take \( M_L \), and for each final state, we remove all outgoing transitions (including self-loops), and replace them with a transition to a “trap” state for each possible input symbol. This gives us the desired \( M' \).

b) Let \( L \) be a regular language, and \( M_L \) a DFA for \( L \). We will construct a DFA \( M' \) for \( \text{max}(L) \).

Observe that if \( w \in \text{max}(L) \), then \( M_L \) on input \( w \) ends on some final state. Furthermore, from this state, no final state is reachable via one or more transitions. So to construct \( M' \), we apply the following algorithm to \( M_L \): Let \( M' = M_L \) initially. For each final state \( q \) in \( M_L \), we find the set of all states reachable from \( q \) via one or more transitions, using depth-first search/breadth-first search/your favorite search algorithm. If there are any final states in this set (including \( q \) itself), we make \( q \) non-final in \( M' \).

c) Similar to part (b), we can construct \( M' \) as follows: Let \( M' = M_L \) initially. For every non-final state \( q \) in \( M_L \), we find the set of all states reachable from \( q \) via one or more transitions. If there are any final states in this set, we make \( q \) final in \( M' \).

Alternatively, we can also use a “reverse marking” algorithm: Starting with all of the final states in \( M_L \), we look at the set of all states that can go to one of these final states on one transition. If any of them are non-final, we mark them as final and repeat the process. When no more new states are marked as final in an iteration, we can stop. The resulting DFA is the \( M' \) we want.