Eliminate state 4:

$\text{In: } (1, c) \quad \text{Out: } (2, b)$

Update $b(3, 1, 2)$, $T(1, 1)$, $T(1, 3)$, $T(2, 1)$, and $T(2, 3)$.

For $K = n$ down to 3:

for $i = 1$ to $K$:

for $j = 1$ to $K$:

$T(i, j) = T(i, k) \cdot T(k, k)^* \cdot T(k, j)$.

Only one acc. state other than $S$.

Hence re-number it (3) and don't need to add an extra final state.

One can skip doing this.

Eliminate state 3:

$\text{In: } (1, f) \quad \text{Out: } (2, a)$

Update $T(1, 2)$, $T(2, 2)$ only.

$b + c^* a = a + c \cdot c^* a = a + c^* b$

$b + c^* b = c + b \cdot c^* \cdot a = b c^* a$

$b + c a = a + c^* a$

$L(11) = (b + c^* b) (b + c)^* a$

$L(12) = T''(1, 1)$
We have completed a cycle of proving Kleene's Theorem:

For every language $A$ over an alphabet $\Sigma$:

1. There is a regular expression $r$ s.t. $A = L(r)$.
2. There is a DFA $M$ s.t. $A = L(M)$.
3. There is an NFA $N$ s.t. $A = L(IN)$.
4. There is a GNFA $N'$ s.t. $A = L(IN')$.

We proved $1 \Rightarrow 3 \Rightarrow 2 \Rightarrow 4$.

We can use these steps algorithmically.

**Theorem:** For every regular expressions $r_1$, $r_2$ we can build a regexp $r_3$ such that $L(r_3) = L(r_1) \cap L(r_2)$.

**Algorithm:**

**Proof:**

- Convert $r_1$ and $r_2$ into equivalent NFAs $N_1$ and $N_2$.
- Convert $N_1$ and $N_2$ into equivalent DFAs $M_1$ and $M_2$.
- Use Cartesian product to build a DFA $M_3$ s.t. $L(M_3) = L(M_1) \cap L(M_2)$.
- Convert $M_3$ into the regexp $r_3$. 

... or $M_3 = L(M_1) \cup L(M_2)$...
An Example to Note: For every $K$, define

$L_K = \{x \in \{0, 1\}^* : \text{the } k\text{th bit from the end is a } 1\}$

Regular $N_K = (0+1)^K \{0 \ldots 0\}$

$NFA\ N_K = \begin{array}{|c|c|}
\hline
& \circ \\vdots \\
\hline
\end{array}$

Fact: The smallest DFA $M_K$ s.t. $L(M_K) = L_K$ has $2^K$ states!

Strategy: One state for every last $K$ bit read.
Accept for $K$ bits begin\text{\text}ing with 1
For $K = 3$, $2^3 = 8$ states

Extra: How do we know that $2^K$ states are needed? A preview:
Consider the two states 110 and 111 at the top, which we got to upon reading $x = 110$ and $y = 111$, respectively. Now suppose the next two chars are $z = 00$. Then $xz = 11000$, which does not belong to $L_3$ because the third char from the right is a 0. But $yz = 11100$, which does belong to $L_3$. We can say $L_3(z) \neq L_3(yz)$ for short, thinking of $L_3(xz)$ as the broken function for $w \in L_3$. This means the DFA needed different states to process $x$ and $y$. If it did not, it would have needed to give the same answer to $xz$ and $yz$. Similar reasoning applies to any two states, taking $z = 00$, $w = 0$, or $z = 1$ depending on where the states' binary labels differ. So the DFA needs all 8 states.