Technische Universität München **Winter term 2020/21**
I7
Prof. J. Esparza / M. Lazić / C. Weil-Kennedy

Automata and Formal Languages — Exercise Sheet 5

Exercise 5.1

Let $L_1 = \{baa, aaa, bab\}$ and $L_2 = \{baa, aab\}$.

(a) Give an algorithm for the following operation:

INPUT: A fixed-length language $L \subseteq \Sigma^k$ described explicitly as a set of words.
OUTPUT: State q of the master automaton over Σ such that $L(q) = L$.

(b) Use the previous algorithm to build the states of the master automaton for L_1 and L_2.

(c) Compute the state of the master automaton representing $L_1 \cup L_2$.

(d) Identify the kernels $\langle L_1 \rangle$, $\langle L_2 \rangle$, and $\langle L_1 \cup L_2 \rangle$.

Exercise 5.2

(a) Give an recursive algorithm for the following operation:

INPUT: States p and q of the master automaton.
OUTPUT: State r of the master automaton such that $L(r) = L(p) \cdot L(q)$.

Observe that the languages $L(p)$ and $L(q)$ can have different lengths. Try to reduce the problem for p, q to the problem for p^a, q.

(b) Give an recursive algorithm for the following operation:

INPUT: A state q of the master automaton.
OUTPUT: State r of the master automaton such that $L(r) = L(q)^R$

where R is the reverse operator.

(c) A *coding* over an alphabet Σ is a function $h \colon \Sigma \mapsto \Sigma$. A coding h can naturally be extended to a morphism over words, i.e. $h(\varepsilon) = \varepsilon$ and $h(w) = h(w_1)h(w_2)\cdots h(w_n)$ for every $w \in \Sigma^n$. Give an algorithm for the following operation:

INPUT: A state q of the master automaton and a coding h.
OUTPUT: State r of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

Can you make your algorithm more efficient when h is a permutation?

Exercise 5.3

Let $k \in \mathbb{N}_{>0}$. Let $\text{flip} : \{0,1\}^k \to \{0,1\}^k$ be the function that inverts the bits of its input, e.g. $\text{flip}(010) = 101$. Let $\text{val} : \{0,1\}^k \to \mathbb{N}$ be such that $\text{val}(w)$ is the number represented by w in the *least significant bit first* encoding.

(a) Describe the minimal transducer that accepts

$$L_k = \left\{[x,y] \in (\{0,1\} \times \{0,1\})^k \mid \text{val}(y) = \text{val}(\text{flip}(x)) + 1 \bmod 2^k \right\}.$$

(b) Build the state r of the master transducer for L_3, and the state q of the master automaton for $\{010, 110\}$.

(c) Adapt the algorithm *pre* seen in class to compute *post* and compute using this algorithm *post*(r, q).

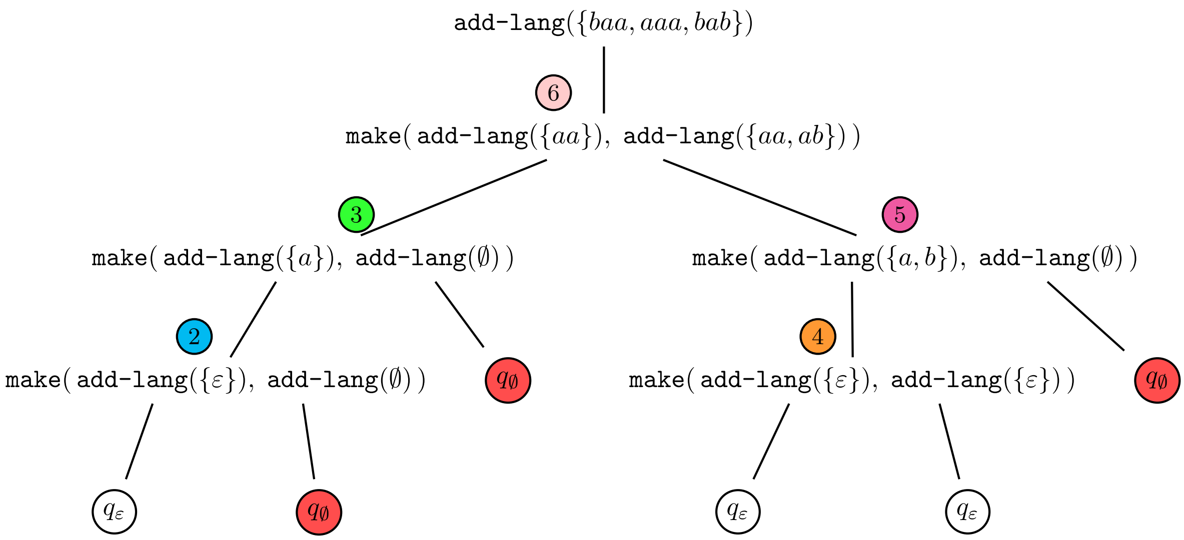
Solution 5.1

(a)

Input: A fixed-length language $L \subseteq \Sigma^k$ described explicitely by a set of words.
Output: State q of the master automaton over Σ such that $L(q) = L$.

```
add-lang(L):
    if L = ∅ then
        return q_∅
    else if L = {ε} then
        return q_ε
    else
        for a_i ∈ Σ do
            L^{a_i} ← {u | a_i u ∈ L}
            s_i ← add-lang(L^{a_i})
        return make(s_1, s_2, ..., s_n)
```

(b) Executing add-lang(L_1) yields the following computation tree:



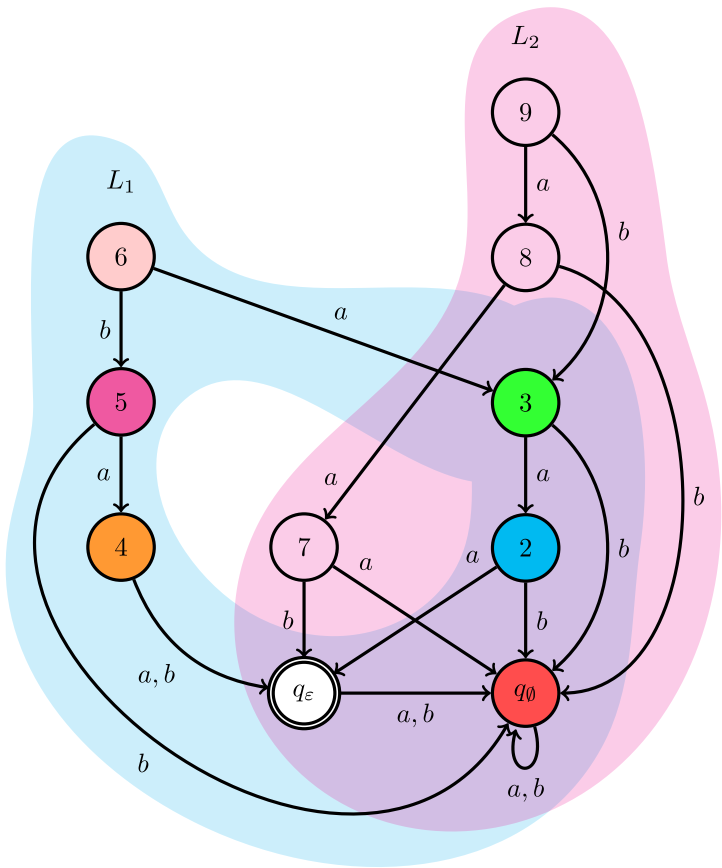
The table obtained after the execution is as follows:

Ident.	a-succ	b-succ
2	q_ε	$q_\emptyset$
3	2	$q_\emptyset$
4	q_ε	q_ε
5	4	$q_\emptyset$
6	3	5

Calling add-lang(L_2) adds the following rows to the table and returns 9:

Ident.	a-succ	b-succ
7	$q_\emptyset$	q_ε
8	7	$q_\emptyset$
9	8	3

The resulting master automaton fragment is:

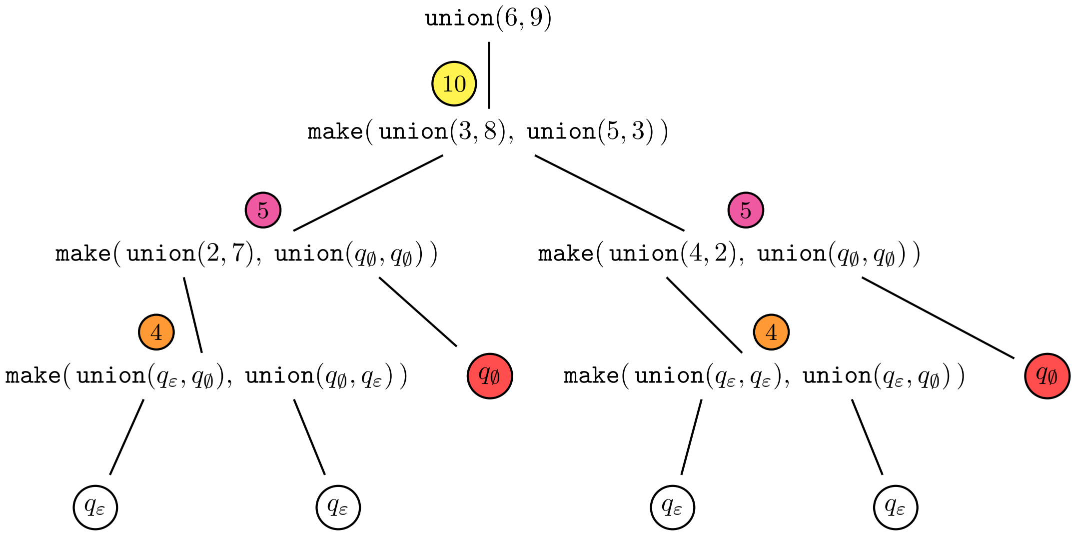


(c) Let us first adapt the algorithm for intersection to obtain an algorithm for union:

Input: States p and q of same length of the master automaton.
Output: State r of the master automaton such that $L(r) = L(p) \cup L(q)$.

`union`(p, q) *:*
    if $G(p, q)$ is not empty **then**
        return $G(p, q)$
    else if $p = q_\emptyset$ and $q = q_\emptyset$ **then**
        return $q_\emptyset$
    else if $p = q_\varepsilon$ or $q = q_\varepsilon$ **then**
        return q_ε
    else
        for $a_i \in \Sigma$ **do**
            $s_i \leftarrow$ `union`(p^{a_i}, q^{a_i})
        $G(p, q) \leftarrow$ `make`$(s_1, s_2, \ldots, s_n)$
        return $G(p, q)$

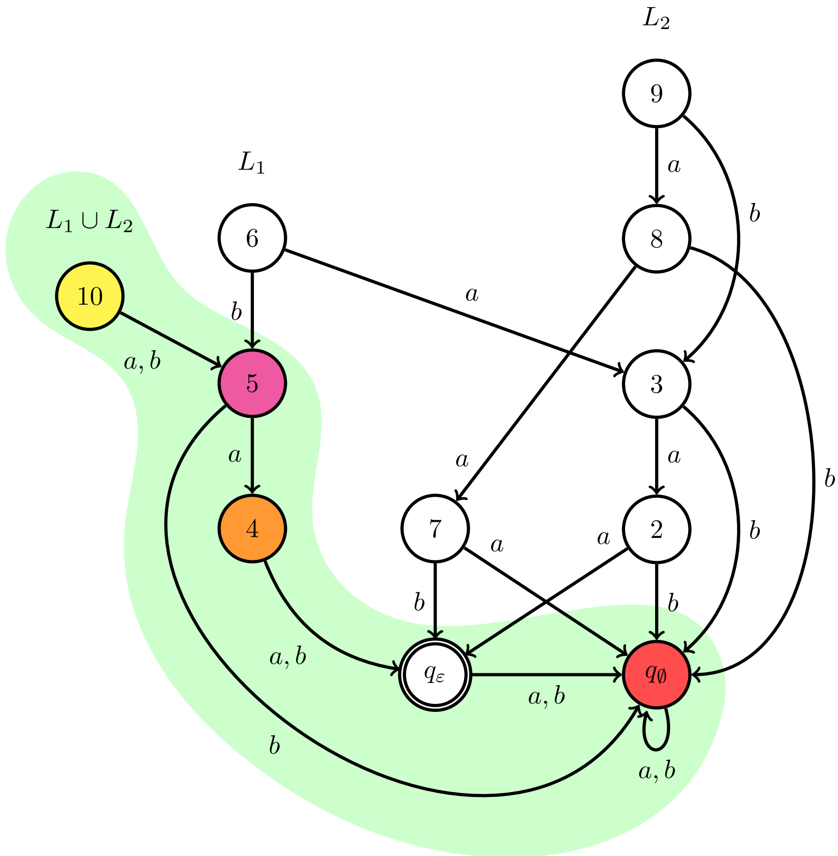
Executing `union`$(6, 9)$ yields the following computation tree:



Calling `union`$(6, 9)$ adds the following row to the table and returns 10:

Ident.	a-succ	b-succ
10	5	5

The new fragment of the master automaton is:



★ Note that `union` could be slightly improved by returning q whenever $p = q$, and by updating $G(q, p)$ at the same time as $G(p, q)$.

(d) The kernels are:

$$\langle L_1 \rangle = L_1,$$
$$\langle L_2 \rangle = L_2,$$
$$\langle L_1 \cup L_2 \rangle = \{aa, ab\}.$$

Solution 5.2

(a) Let L and L' be fixed-length languages. The following holds:

$$L \cdot L' = \begin{cases} \emptyset & \text{if } L = \emptyset, \\ L' & \text{if } L = \{\varepsilon\}, \\ \bigcup_{a \in \Sigma} \{a\} \cdot L^a \cdot L' & \text{otherwise.} \end{cases}$$

These identities give rise to the following algorithm:

Input: States p and q of the master automaton.
Output: State r of the master automaton such that $L(r) = L(p) \cdot L(q)$.

`concat`(p, q) :
    if $G(p, q)$ is not empty **then**
        return $G(p, q)$
    else if $p = q_\emptyset$ **then**
        return $q_\emptyset$
    else if $p = q_\varepsilon$ **then**
        return q
    else
        for $a_i \in \Sigma$ **do**
            $s_i \leftarrow$ `concat`(p^{a_i}, q)
        $G(p, q) \leftarrow$ `make`$(s_1, s_2, \ldots, s_n)$
        return $G(p, q)$

(b) Let L be a fixed-length language. The following holds:

$$L^R = \begin{cases} \emptyset & \text{if } L = \emptyset, \\ \{\varepsilon\} & \text{if } L = \{\varepsilon\}, \\ \bigcup_{a \in \Sigma} (L^a)^R \cdot \{a\} & \text{otherwise.} \end{cases}$$

These identities give rise to the following algorithm:

★ Note that Lines 11 and 12 are introduced in order to represent the language $\{a_i\}$ in Line 13 as a state `make`$(s_1, s_2, \ldots, s_n)$ of the master automaton. This can be avoided by using the algorithm from Exercise 8.1, namely the state that represents $\{a_i\}$ is `add-lang`$(\{a_i\})$. Thus, Lines 11-13 can be replaced just by $r \leftarrow$ `concat`(`reverse`(q^{a_i}), `add-lang`$(\{a_i\})$)

Input: A state q of the master automaton.
Output: State r of the master automaton such that $L(r) = L(q)^R$.

```
reverse(q):
    if G(q) is not empty then
        return G(q)
    else if q = q_∅ then
        return q_∅
    else if q = q_ε then
        return q_ε
    else
        p ← q_∅
        for a_i ∈ Σ do
            s_i ← q_ε
            s_j ← q_∅ for every i ≠ j
            r ← concat(reverse(q^{a_i}), make(s_1, s_2, ..., s_n))
            p ← union(p, r)
        G(q) ← p
        return G(q)
```

(c) Let L be a fixed-length language and let h be a coding. The following holds:

$$h(L) = \begin{cases} \emptyset & \text{if } L = \emptyset, \\ \{\varepsilon\} & \text{if } L = \{\varepsilon\}, \\ \bigcup_{a \in \Sigma} h(a) \cdot h(L^a) & \text{otherwise.} \end{cases}$$

These identities give rise to the following algorithm:

Input: A state q of the master automaton and a coding h.
Output: State r of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

```
coding(q, h):
    if G(q) is not empty then
        return G(q)
    else if q = q_∅ then
        return q_∅
    else if q = q_ε then
        return q_ε
    else
        p ← q_∅
        for a ∈ Σ do
            r ← coding(q^a, h)
            s_{h(a)} ← r
            s_b ← q_∅ for every b ≠ h(a)
            p ← union(p, make(s))
        G(q) ← p
        return G(q)
```

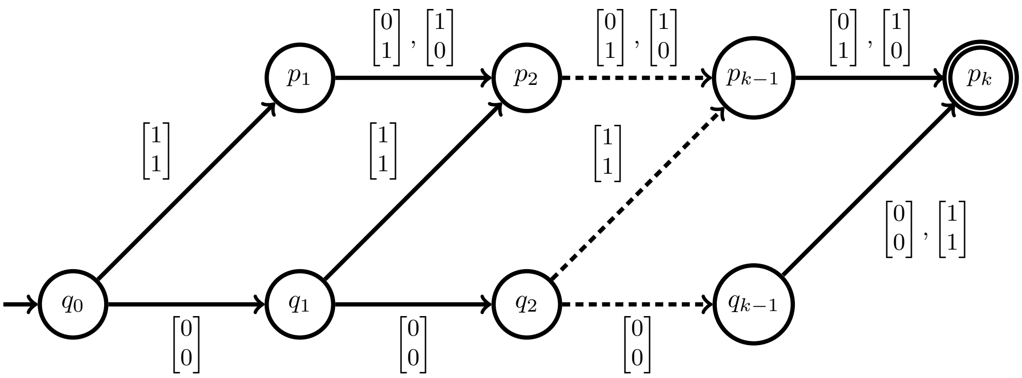
The above algorithm makes use of `union` because the coding may be the same for distinct letters, i.e. $h(a) = h(b)$ for $a \neq b$ is possible. However, if the coding is a permutation, then this is not possible, and thus each letter maps to a unique residual. Therefore, the algorithm can be adapted as follows:

Input: A state q of the master automaton and a coding h which is a permutation.
Output: State r of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

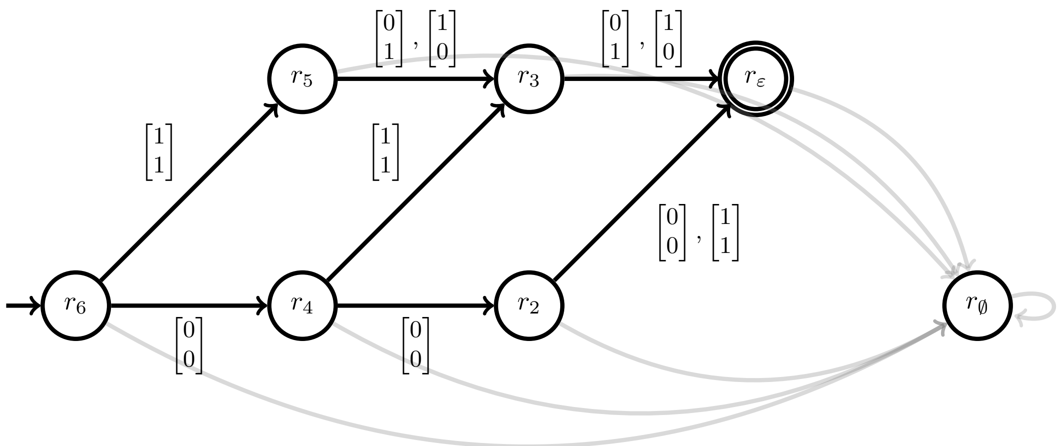
```
coding-permutation(q, h):
    if G(q) is not empty then
        return G(q)
    else if q = q_∅ then
        return q_∅
    else if q = q_ε then
        return q_ε
    else
        for a ∈ Σ do
            s_h(a) ← coding-permutation(q^a, h)
        G(q) ← make(s)
        return G(q)
```

Solution 5.3

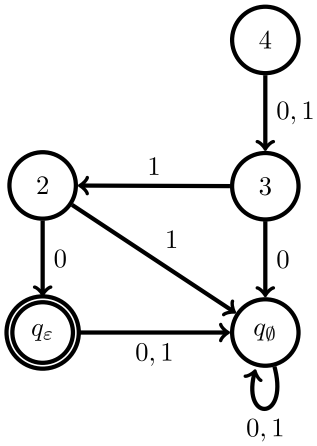
(a) Let $[x, y] \in L_k$. We may flip the bits of x at the same time as adding 1. If $x_1 = 1$, then $\neg x_1 = 0$, and hence adding 1 to val(flip(x)) results in $y_1 = 1$. Thus, for every $1 < i \leq k$, we have $y_i = \neg x_i$. If $x_1 = 0$, then $\neg x_1 = 1$. Adding 1 yields $y_1 = 0$ with a carry. This carry is propagated as long as $\neg x_i = 1$, and thus as long as $x_i = 0$. If some position j with $x_j = 1$ is encountered, the carry is "consumed", and we flip the remaining bits of x. These observations give rise to the following minimal transducer for L_k:



(b) The minimal transducer accepting L_3 is



State 4 of the following master automaton fragment accepts $\{010, 110\}$:



(c) We can establish the following identities similar to those obtained for *pre*:

$$post_R(L) = \begin{cases} \emptyset & \text{if } R = \emptyset \text{ or } L = \emptyset, \\ \{\varepsilon\} & \text{if } R = \{[\varepsilon, \varepsilon]\} \text{ and } L = \{\varepsilon\}, \\ \bigcup_{a,b \in \Sigma} b \cdot post_{R^{[a,b]}}(L^a) & \text{otherwise.} \end{cases}$$

To see that these identities hold, let $b \in \Sigma$ and $v \in \Sigma^k$ for some $k \in \mathbb{N}$. We have,

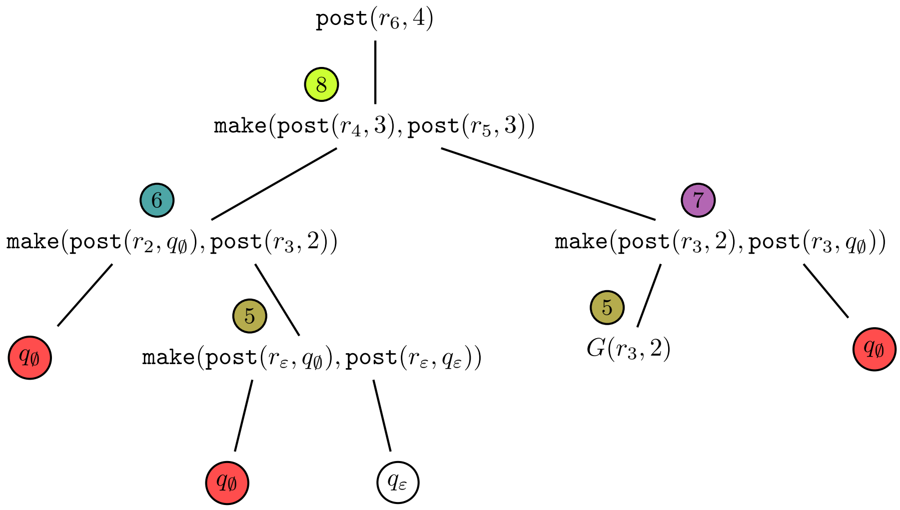
$$\begin{aligned} bv \in post_R(L) &\iff \exists a \in \Sigma, u \in \Sigma^k \text{ s.t. } au \in L \text{ and } [au, bv] \in R \\ &\iff \exists a \in \Sigma, u \in L^a \text{ s.t. } [au, bv] \in R \\ &\iff \exists a \in \Sigma, u \in L^a \text{ s.t. } [u, v] \in R^{[a,b]} \\ &\iff \exists a \in \Sigma \text{ s.t. } v \in Post_{R^{[a,b]}}(L^a) \\ &\iff v \in \bigcup_{a \in \Sigma} Post_{R^{[a,b]}}(L^a) \\ &\iff bv \in \bigcup_{a \in \Sigma} b \cdot Post_{R^{[a,b]}}(L^a). \end{aligned}$$

We obtain the following algorithm:

Input: A state r of the master transducer and a state q of the master automaton.
Output: State p of the master automaton such that $L(p) = Post_R(L)$ where $R = L(r)$ and $L = L(q)$.

`post`(r, q) :
    if $G(r, q)$ is not empty **then**
        return $G(r, q)$
    else if $r = r_\emptyset$ or $q = q_\emptyset$ **then**
        return $q_\emptyset$
    else if $r = r_\varepsilon$ and $q = q_\varepsilon$ **then**
        return q_ε
    else
        for $b_i \in \Sigma$ **do**
            $p \leftarrow q_\emptyset$
            for $a \in \Sigma$ **do**
                $p \leftarrow$ `union`$(p,$ `post`$(r^{[a,b_i]}, q^a))$
            $s_i \leftarrow p$
        $G(q, r) \leftarrow$ `make`$(s_1, s_2, \ldots, s_n)$
        return $G(q, r)$

Note that the transducer for L_3 has some "strong" deterministic property. Indeed, for every state r and $b \in \{0,1\}$, if $r^{[a,b]} \neq r_\emptyset$ then $r^{[\neg a,b]} = r_\emptyset$. Hence, for a fixed $b \in \{0,1\}$, at most one term of the form "$\mathtt{post}(r^{[a,b]}, q^a)$" can differ from $q_\emptyset$ at line 12 of the algorithm. Thus, unions made by the algorithm on this transducer are trivial, and executing $\mathtt{post}(6,4)$ yields the following computation tree:



Calling $\mathtt{post}(6,4)$ adds the following rows to the master automaton table and returns 8:

Ident.	0-succ	1-succ
5	$q_\emptyset$	q_ε
6	$q_\emptyset$	5
7	5	$q_\emptyset$
8	6	7

The resulting master automaton fragment: