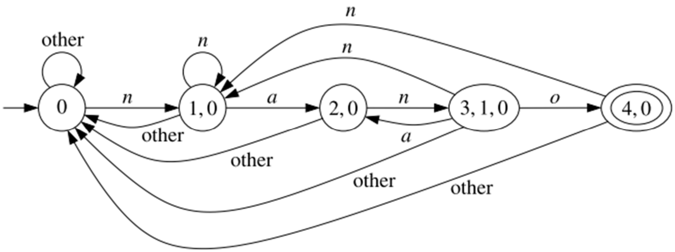
Lazy DFAs

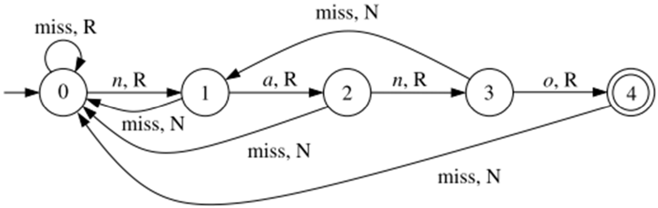
- We introduce a new data structure: lazy DFAs. We construct a lazy DFA for $\Sigma^* p$ with m states and $2m$ transitions.
- Lazy DFAs: automata that read the input from a tape by means of a reading head that can move one cell to the right or stay put
- DFA=Lazy DFA whose head never stays put

Lazy DFA for $\Sigma^* p$

- By the fundamental property, the DFA B_p for $\Sigma^* p$ behaves from state S_k as follows:
  - If a is a hit, then $\delta_B(S_k, a) = S_{k+1}$, i.e., the DFA moves to the next state in the spine.
  - If a is a miss, then $\delta_B(S_k, a) = \delta_B(t(S_k), a)$, i.e., the DFA moves to the same state it would move to if it were in state $t(S_k)$.
- When a is a miss for S_k, the lazy automaton moves to state $t(S_k)$ without advancing the head. In other words, it „delegates" doing the move to $t(S_k)$
- So the lazyDFA behaves the same for all misses.



other
n
n
n
0
n
1, 0
a
2, 0
n
3, 1, 0
o
4, 0
other
a
other
other
other



miss, N
miss, R
0
n, R
1
a, R
2
n, R
3
o, R
4
miss, N
miss, N
miss, N

- Formally,
  - $\delta_C(S_k, a) = (S_{k+1}, R)$ if a is a hit
  - $\delta_C(S_k, a) = (t(S_k), N)$ if a is a miss
- So the lazy DFA has $m + 1$ states and $2m$ transitions, and can be constructed in $O(m)$ space.

- Running the lazy DFA on the text takes $O(n+m)$ time:
  - For every text letter we have a sequence of „stay put" steps followed by a „right" step. Call it a macrostep.
  - Let S_{j_i} be the state after the i-th macrostep. The number of steps of the i-th macrostep is at most $j_{i-1} - j_i + 2$.

  So the total number of steps is at most

$$\sum_{i=1}^{n} (j_{i-1} - j_i + 2) = j_0 - j_n + 2n \leq m + 2n$$

Computing $Miss$

- For the $O(m+n)$ bound it remains to show that the lazy DFA can be constructed in $O(m)$ time.
- Let $Miss(k)$ be the head of the state reached from S_k by a miss.
- It is easy to compute each of $Miss(0), \dots, Miss(m)$ in $O(m)$ time, leading to a $O(n+m^2)$ time algorithm.
- Already good enough for almost all purposes. But, can we compute all of $Miss(0), \dots, Miss(m)$ together in time $O(m)$? Looks impossible!
- It isn‘t though ...

$$
miss(S_i) = \begin{cases} S_0 & \text{if } i = 0 \text{ or } i = 1 \\ \delta_B(miss(S_{i-1}), b_i) & \text{if } i > 1 \end{cases}
$$

$$
\delta_B(S_j, b) = \begin{cases} S_{j+1} & \text{if } b = b_{j+1} \text{ (hit)} \\ S_0 & \text{if } b \neq b_{j+1} \text{ (miss) and } j = 0 \\ \delta_B(miss(S_j), b) & \text{if } b \neq b_{j+1} \text{ (miss) and } j \neq 0 \end{cases}
$$

Miss(p)
Input: word pattern $p = b_1 \cdots b_m$.
Output: heads of targets of miss transitions.

```
Miss(0) ← 0; Miss(1) ← 0
for i ← 2, ..., m do
    Miss(i) ← DeltaB( Miss(i − 1) , b_i )
```

DeltaB(j, b)
Input: number $j \in \{0, \ldots, m\}$, letter b.
Output: head of the state $\delta_B(S_j, b)$.

```
while b ≠ b_{j+1} and j ≠ 0 do  j ← Miss(j)
if b = b_{j+1} then return j + 1
else return 0
```

Miss(p)
Input: word pattern $p = b_1 \cdots b_m$.
Output: heads of targets of miss transitions.

$Miss(0) \leftarrow 0$; $Miss(1) \leftarrow 0$
for $i \leftarrow 2, \ldots, m$ **do**
    $Miss(i) \leftarrow DeltaB(\, Miss(i-1)\, ,\, b_i\,)$

DeltaB(j, b)
Input: number $j \in \{0, \ldots, m\}$, letter b.
Output: head of the state $\delta_B(S_j, b)$.

while $b \neq b_{j+1}$ **and** $j \neq 0$ **do** $j \leftarrow Miss(j)$
if $b = b_{j+1}$ **then return** $j+1$
else return 0

- All calls to *DeltaB* lead together to $O(m)$ iterations of the while loop.
- The call $DeltaB(Miss(i-1), b\_i)$ executes at most $Miss(i-1) - (Miss(i) - 1)$ iterations.

- Total number of iterations:

$$\sum_{i=2}^{m}(Miss(i-1) - Miss(i) + 1)$$

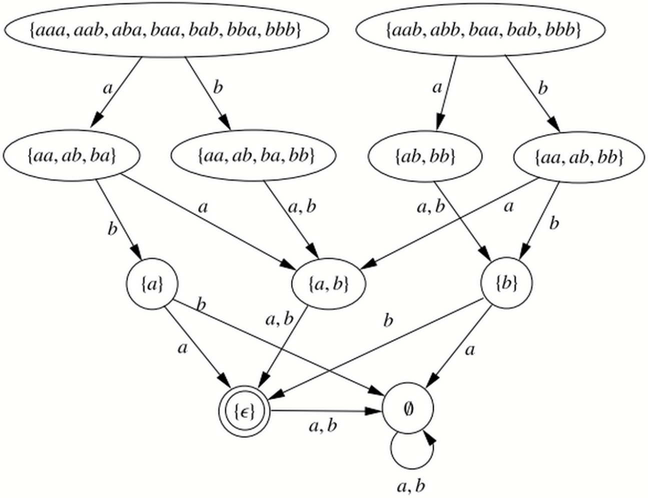
$$\leq \; Miss(1) - Miss(m) + m$$

$$\leq m$$

7. Finite Universes

- When the universe is finite (*e.g.*, the interval $[0, 2^{32} - 1]$), all objects can be encoded by words of the same length.
- A language L has length $n \geq 0$ if
  - $L = \emptyset$ and $n = 0$, or
  - $L \neq \emptyset$ and every word of L has length n.
- L is a fixed-length language if it has length n for some $n \geq 0$.
- Observe:
  - Fixed-length languages contain finitely many words.
  - $\emptyset$ and $\{\varepsilon\}$ are the only two languages of length 0.

The Master Automaton



- The master automaton over Σ is the tuple $\mathrm{M} = (Q_M, \Sigma, \delta_M, F_M)$, where
  - Q_M is the set of all fixed-length languages;
  - $\delta_M\colon Q_M \times \Sigma \to Q_M$ is given by $\delta_M(L, a) = L^a$;
  - F_M is the set $\{\{\varepsilon\}\}$.

- Prop: The language recognized from state L of the master automaton is L.

  Proof: By induction on the length n of L.

  $n = 0$. Then either $L = \emptyset$ or $L = \{\varepsilon\}$, and result follows by inspection.

  $n > 0$. Then $\delta_M(L, a) = L^a$ for every $a \in \Sigma$, and L^a has smaller length than L. By induction hypothesis the state L^a recognizes the language L^a, and so the state L recognizes the language L.

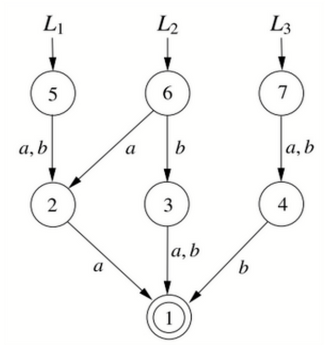
- We denote the „fragment“ of the master automaton reachable from state L by A_L :
  - Initial state is L.
  - States and transitions are those reachable from L.

- Prop: A_L is the minimal DFA recognizing L.

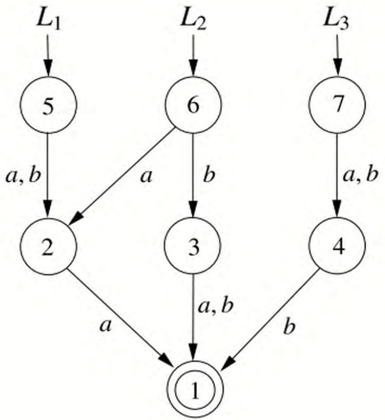
  Proof: By definition, all states of A_L are reachable from its initial state. Since every state of the master automaton recognizes its „own“ language, distinct states of A_L recognize distinct languages.

Data structure for fixed-length languages

- The structure representing the set of languages $\mathcal{L} = \{L_1, \ldots, L_m\}$ is the fragment of the master automaton containing states $L_1, \ldots, L_m$ and their descendants.
- It is a multi-DFA , i.e., a DFA with multiple initial states.

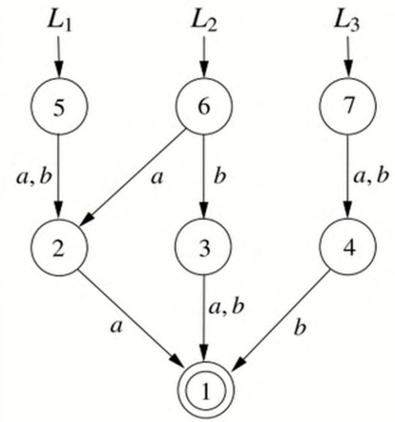


In order to manipulate multi-DFAs we represent them as a *table of nodes*. Assume $\Sigma = \{a_1, \ldots, a_m\}$. A *node* is a pair $\langle q, s\rangle$, where q is a *state identifier* and $s = (q_1, \ldots, q_m)$ is the *successor tuple* of the node. The multi-DFA is represented by a table containing a node for each state, but the state corresponding to the empty language[1].



Ident.	a-succ	b-succ
2	1	0
3	1	1
4	0	1
5	2	2
6	2	3
7	4	4

- We represent multi-DFAs as tables of nodes .
- A node is a pair $\langle q, s \rangle$ where
  - q is a state identifier, and
  - $s = (q_1, \ldots, q_m)$ is a successor tuple.
- The table for a multi-DFA contains a node for each state but the state for the empty language.



Ident.	a-succ	b-succ
2	1	0
3	1	1
4	0	1
5	2	2
6	2	3
7	4	4

- The procedure $make[T](s)$
  - returns the state identifier of the node of table T having s as successor tuple, if such a node exists;
  - otherwise it adds a new node $\langle q, s \rangle$ to T, where q is a fresh identifier , and returns q.
- $make[T](s)$ assumes that T contains a node for every identifier in s.