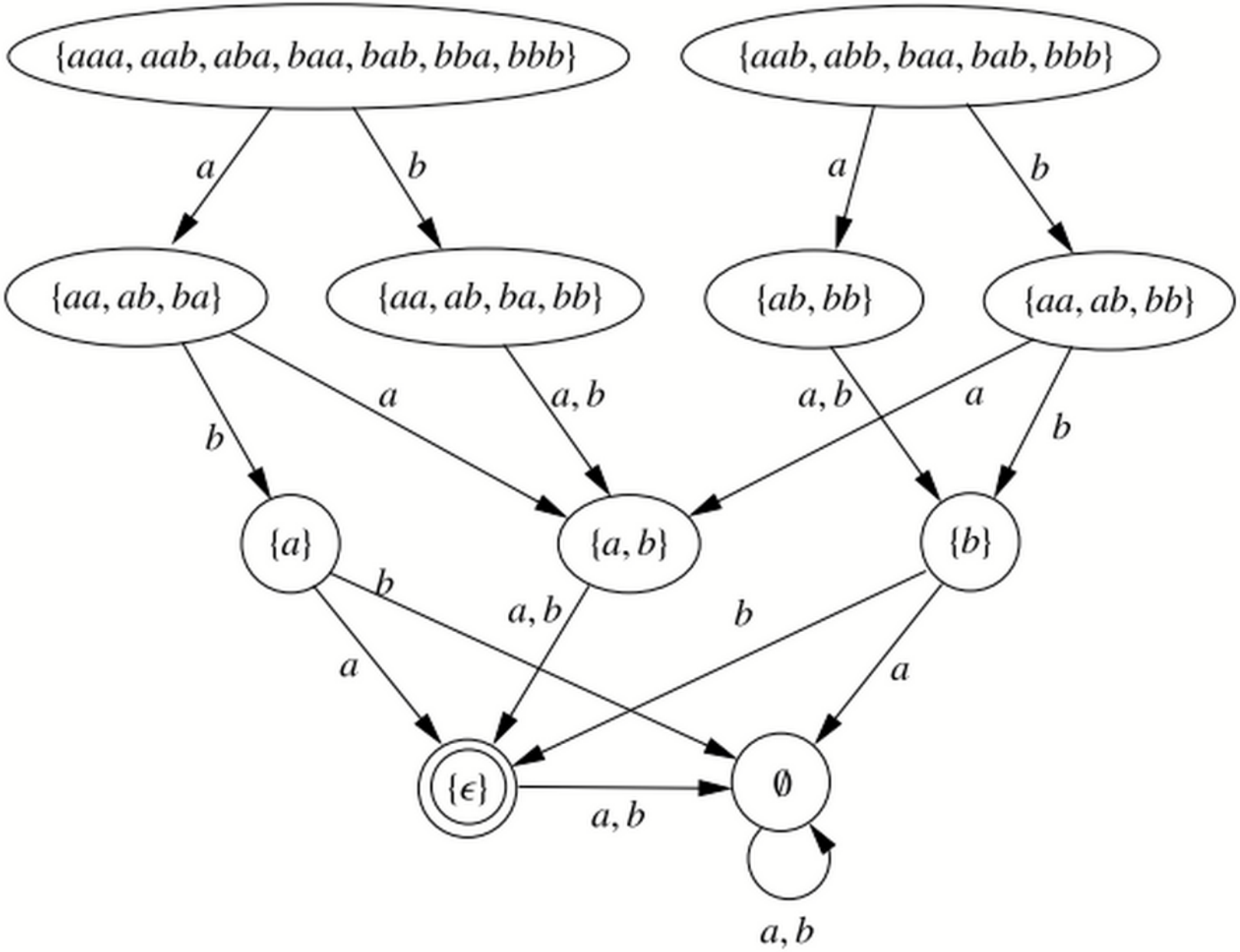
Finite Universes

- 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
  - $L = \emptyset$, or
  - 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.
  - $\emptyset$ is a language of any length!

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;
  - δ_M: $Q_M \times \Sigma \to Q_M$ 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.

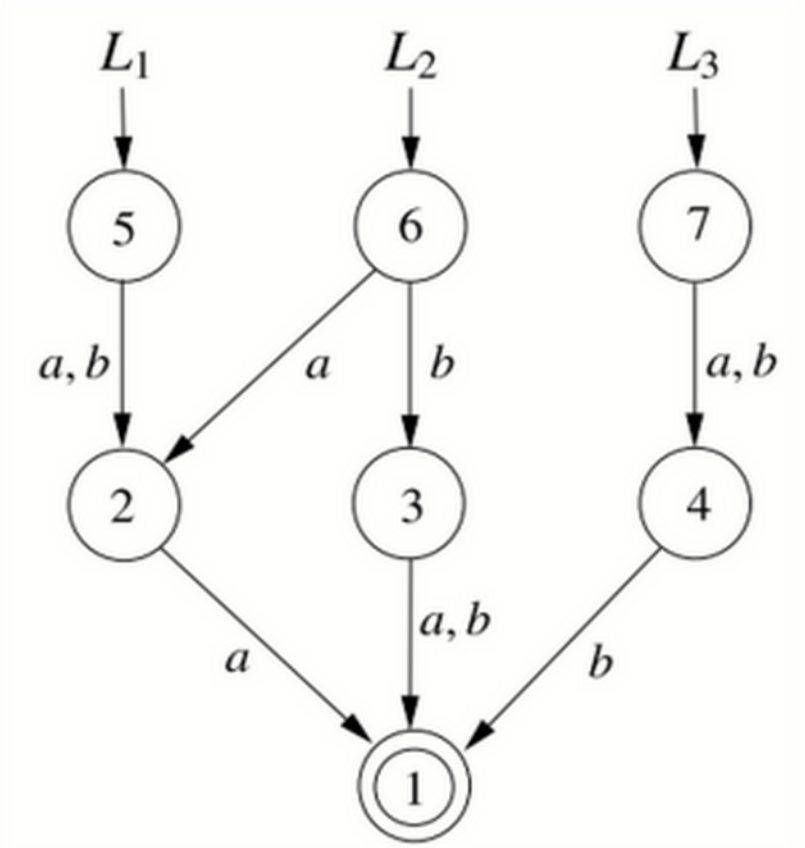
- We denote the „fragment“ of the master automaton reachable from state L by A_L :
  - Initial state is L.
  - States and transitions are those reachable from L.

- Prop: A_L is the minimal DFA recognizing L.

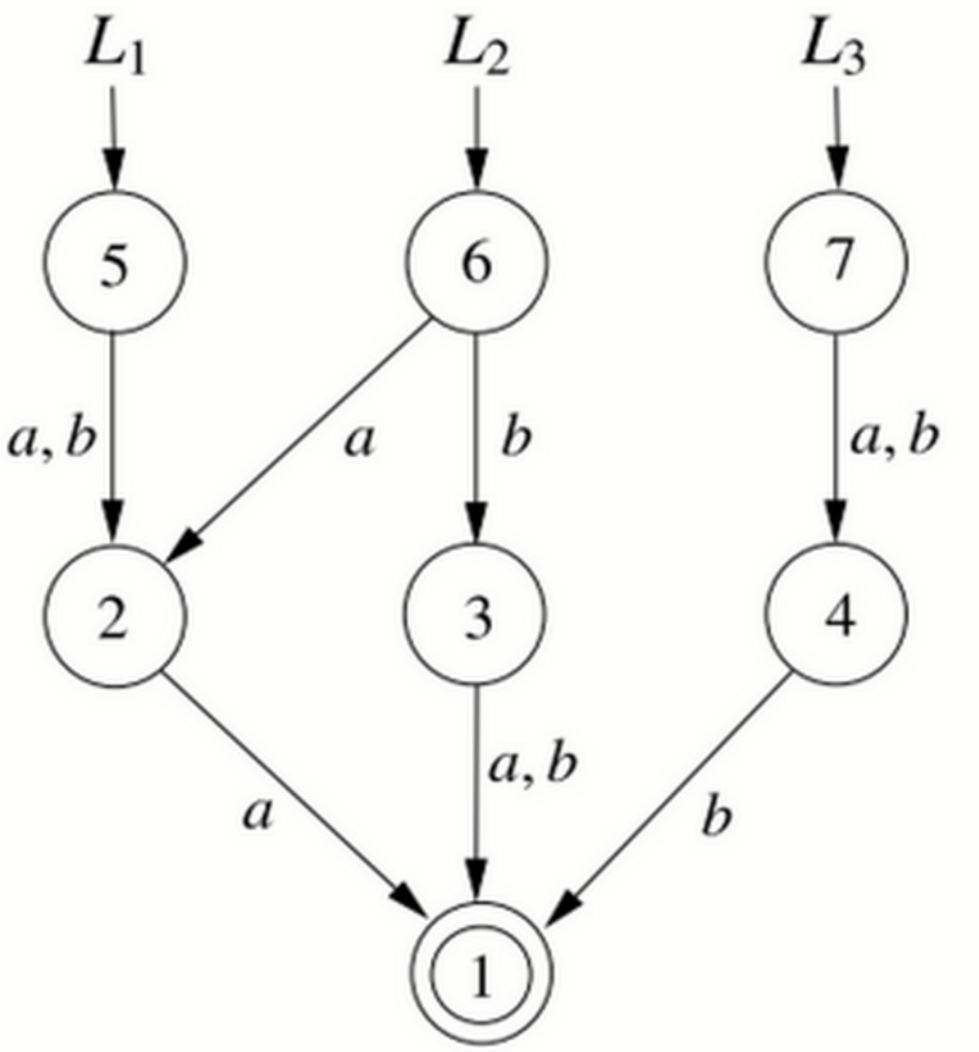
  Proof: By definition, all states of A_L are reachable from its initial state. Since every state of the master automaton recognizes its „own“ language, distinct states of A_L recognize distinct languages.

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.



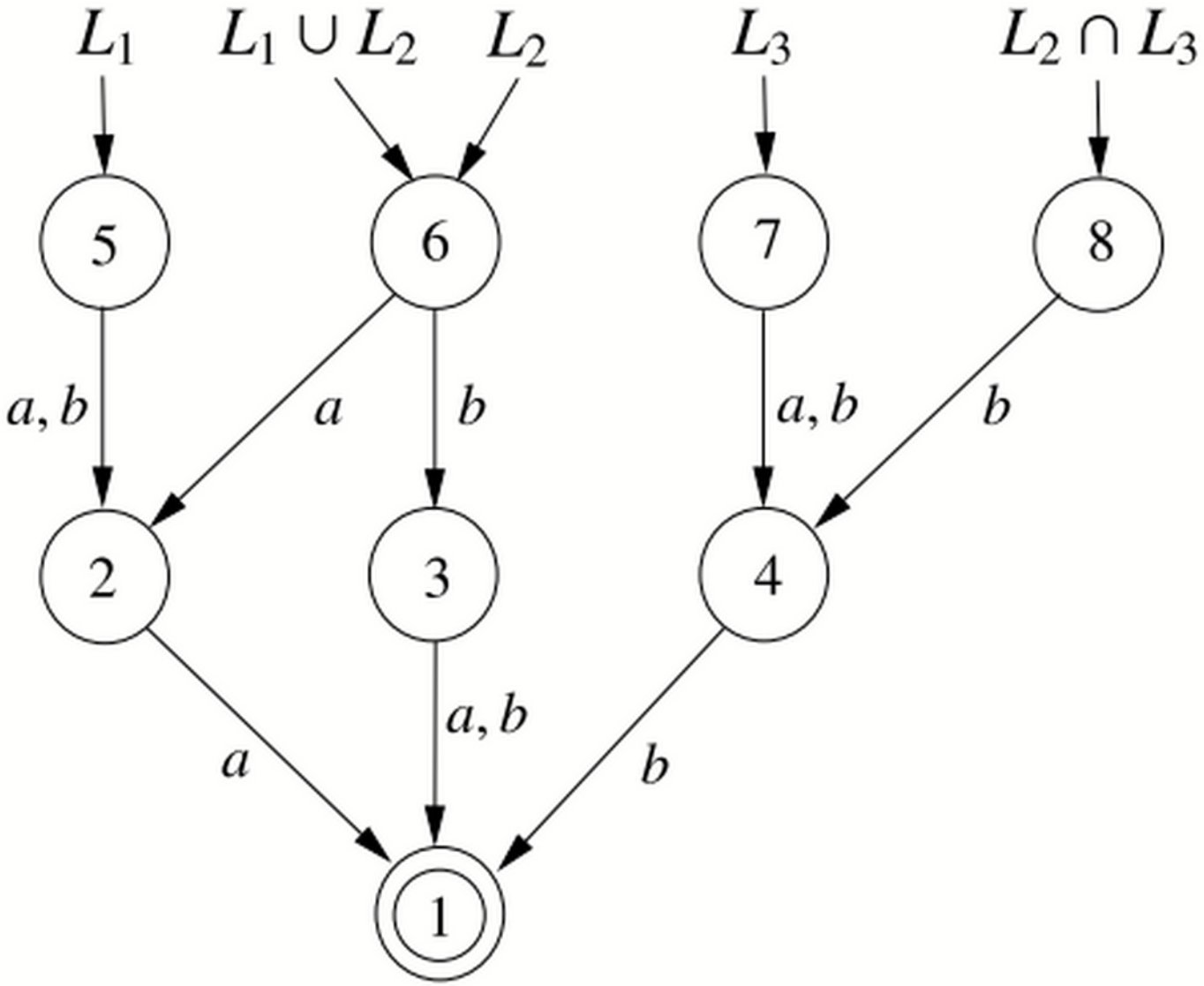
- We represent multi-DFAs as tables of nodes .
- A node is a pair $\langle q, s \rangle$ where
  - q is a state identifier, and
  - $s = (q_1, \ldots, q_m)$ is a successor tuple.
- The table for a multi-DFA contains a node for each state but the state for the empty language.



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

- 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.

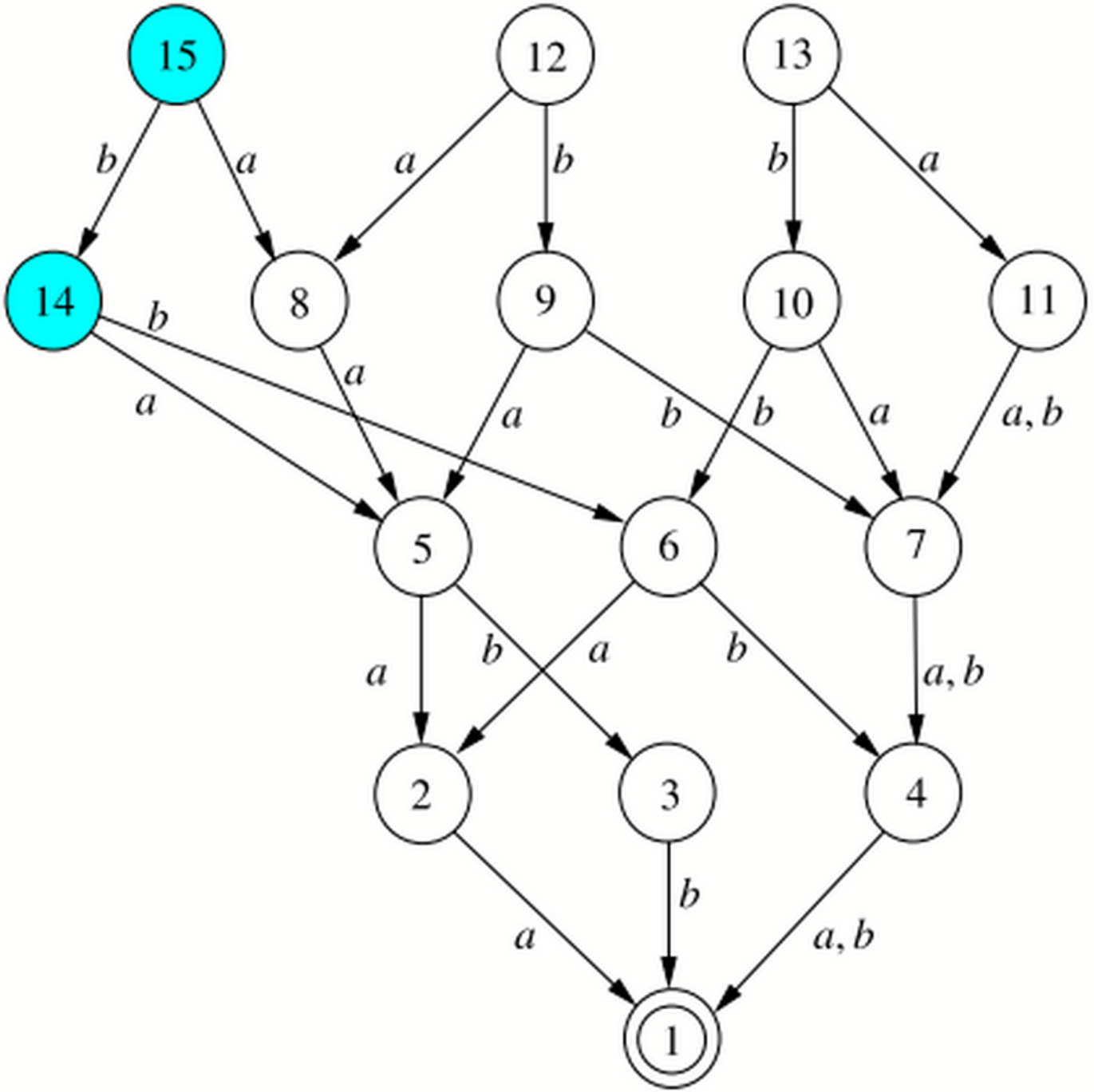
Implementing union and intersection

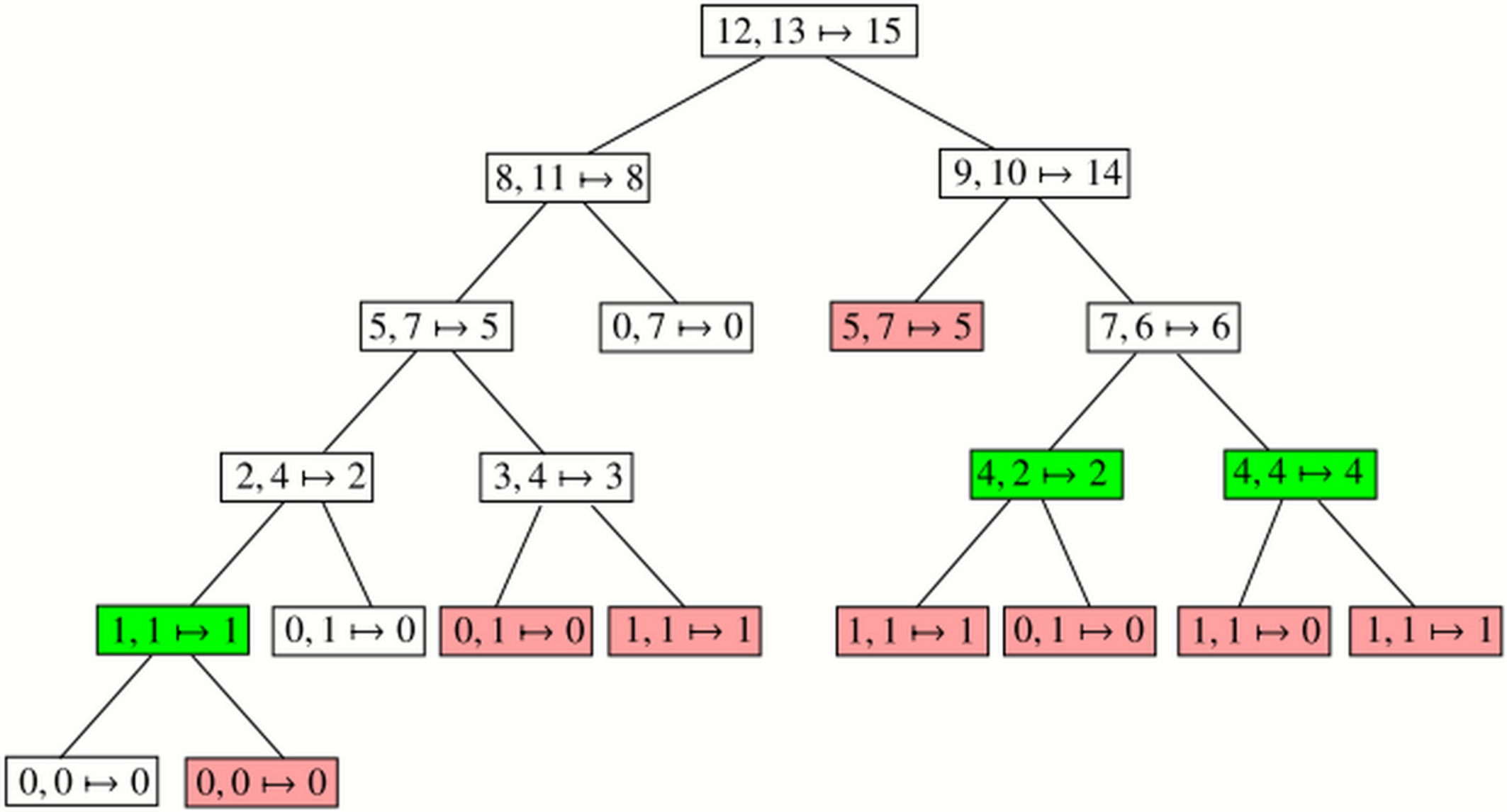


- We give a recursive algorithm $inter[T](q_1, q_2)$:
  - Input: state identifiers q_1, q_2 from table T.
  - Output: identifier of the state recognizing $L(q_1) \cap L(q_2)$ in the multi-DFA for T.
  - Side-effect: if the identifier is not in T, then the algorithm adds new nodes to T, i.e., after termination the table T may have been extended.
- The algorithm follows immediately from the following properties
  (1) if $L_1 = \emptyset$, then $L_1 \cap L_2 = \emptyset$;
  (2) if $L_2 = \emptyset$, then $L_1 \cap L_2 = \emptyset$;
  (3) If $L_1 \neq \emptyset$ and $L_2 \neq \emptyset$, then $(L_1 \cap L_2)^a = L_1^a \cap L_2^a$ for every $a \in \Sigma$.

$inter[T](q_1, q_2)$
Input: table T, states q_1, q_2 of T
Output: state recognizing $\mathcal{L}(q_1) \cap \mathcal{L}(q_2)$

```
1  if G(q1, q2) is not empty  then return G(q1, q2)
2  if q1 = q∅ ∨ q2 = q∅ then return q∅
3  if q1 ≠ q∅ ∧ q2 ≠ q∅ then
4     for all i = 1, ..., m do r_i ← inter[T](q1^{a_i}, q2^{a_i})
5     G(q1, q2) ← make[T](r1, ..., rm)
6     return G(q1, q2)
```


15
12
13
14
8
9
10
11
5
6
7
2
3
4
1
b
a
a
b
b
a
b
a
a
b
a
b
b
a
a, b
a
b
a
b
a, b
a
b
a, b


12, 13 ↦ 15
8, 11 ↦ 8
9, 10 ↦ 14
5, 7 ↦ 5
0, 7 ↦ 0
5, 7 ↦ 5
7, 6 ↦ 6
2, 4 ↦ 2
3, 4 ↦ 3
4, 2 ↦ 2
4, 4 ↦ 4
1, 1 ↦ 1
0, 1 ↦ 0
0, 1 ↦ 0
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
1, 1 ↦ 0
1, 1 ↦ 1
0, 0 ↦ 0
0, 0 ↦ 0

Fixed-length complement

In principle ill-defined, because the complement of a fixed-length language is not fixed-length.

We implement the fixed-length complement instead.

Can't we just swap the states for the empty language and the language containing the empty word?

Yes and no ...

Fixed-length complement

Equations:

- if $L = \emptyset$, then $\overline{L} = \Sigma^n$, where n is the length of L;
- if $L = \{\epsilon\}$, then $\overline{L} = \emptyset$; and
- if $\emptyset \neq L \neq \{\epsilon\}$, then $\left(\overline{L}\right)^a = \overline{L^a}$.
  (Observe that $w \in \left(\overline{L}\right)^a$ iff $aw \notin L$ iff $w \notin L^a$ iff $w \in \overline{L^a}$.)

$comp[T, n](q)$

Input: table T, length n, state q of T of length n

Output: state recognizing the fixed-length complement of $L(q)$

1 **if** $G(q)$ is not empty **then return** $G(q)$

2 **if** $n = 0$ **and** $q = q_\emptyset$ **then return** q_ϵ

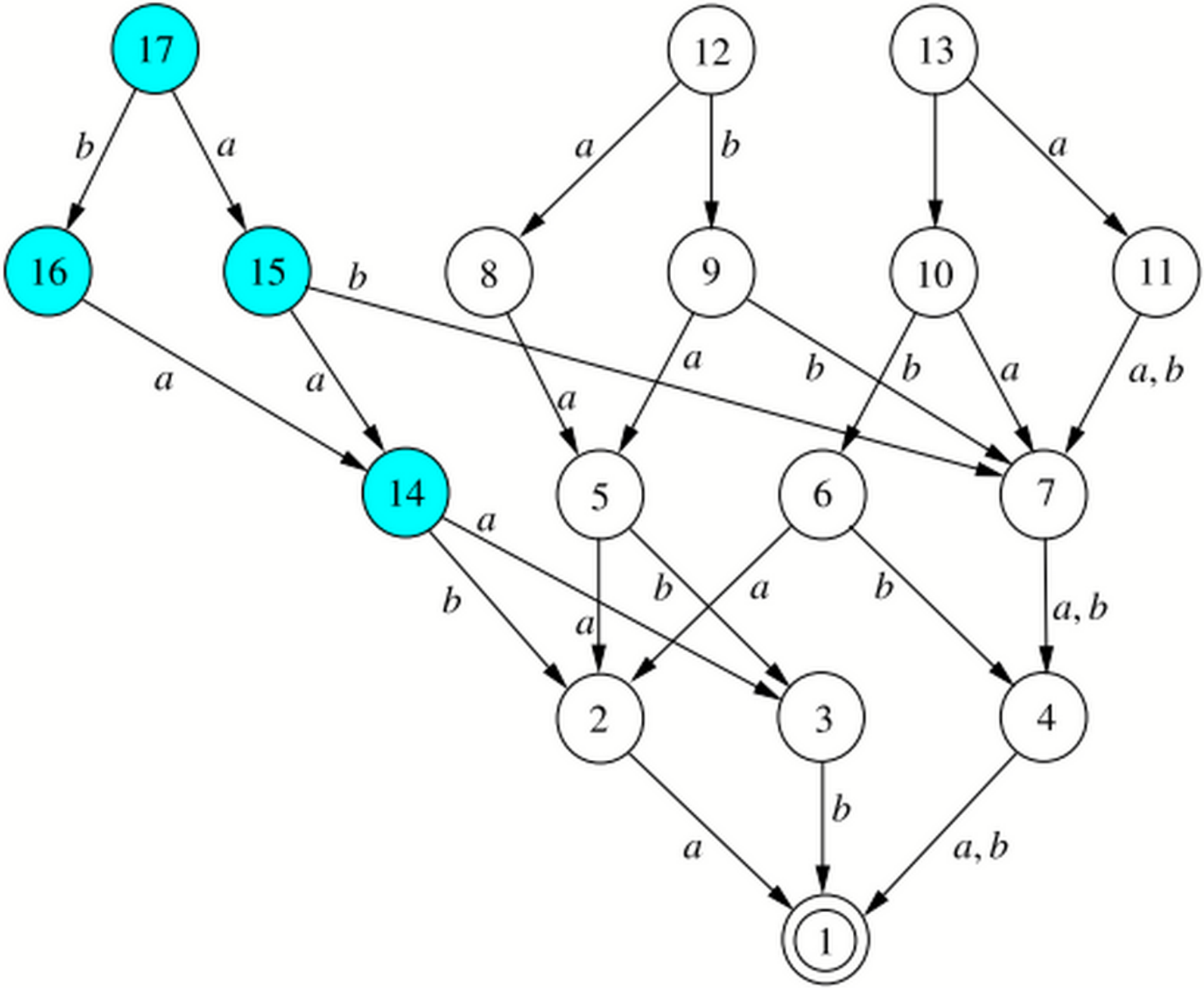
3 **else if** $n = 0$ **and** $q = q_\epsilon$ **then return** $q_\emptyset$

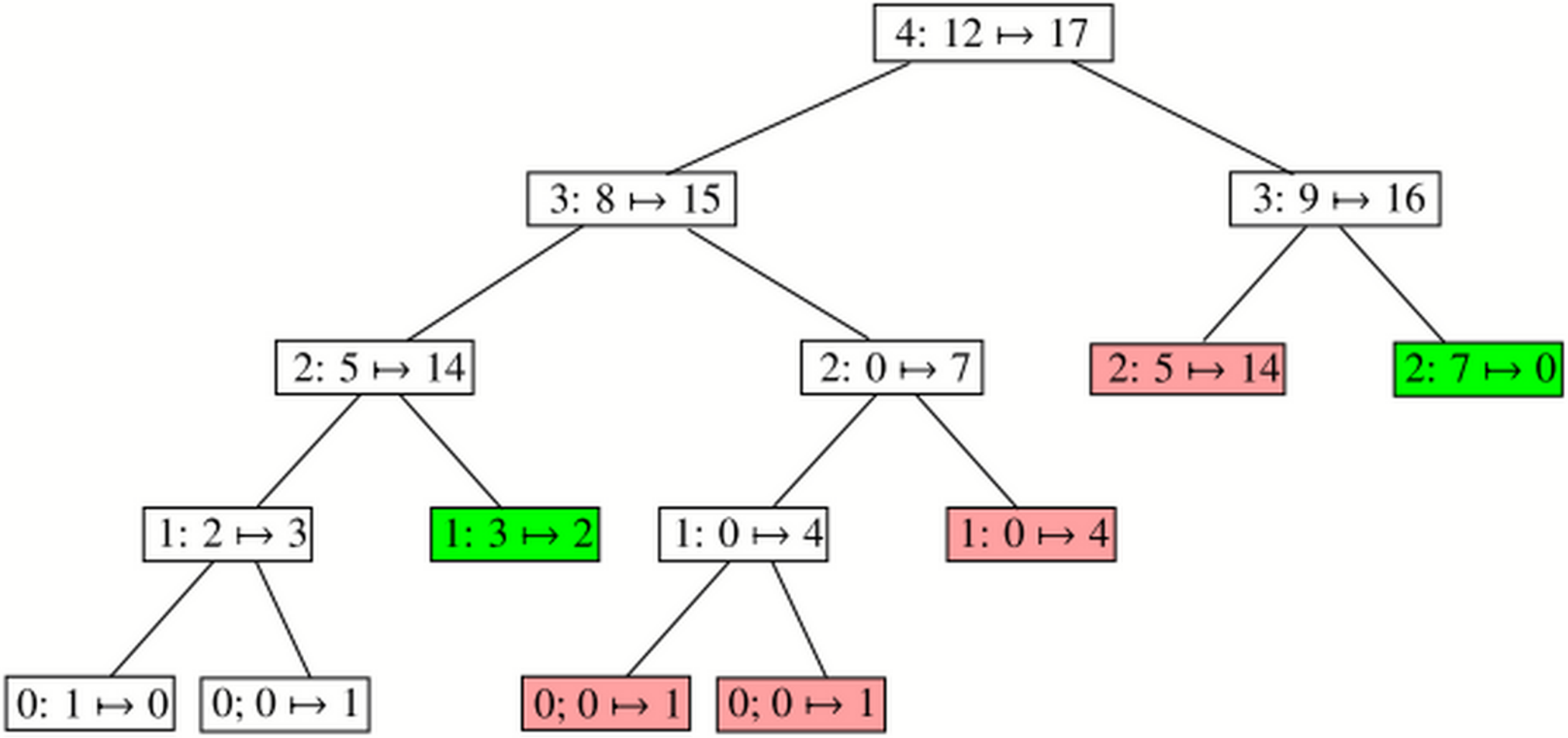
4 **else** $/* \ n \geq 1 \ */$

5 **for all** $i = 1, \ldots, m$ **do** $r_i \leftarrow comp[T, n-1](q^{a_i})$

6 $G(q) \leftarrow \texttt{make}[T](r_1, \ldots, r_m)$

7 **return** $G(q)$


17
12
13
b
a
a
b
a
16
15
8
9
10
11
b
a
b
b
a
a, b
a
a
a
14
5
6
7
a
b
a
b
a, b
b
a
2
3
4
b
a
a, b
1


4: 12 ↦ 17
3: 8 ↦ 15
3: 9 ↦ 16
2: 5 ↦ 14
2: 0 ↦ 7
2: 5 ↦ 14
2: 7 ↦ 0
1: 2 ↦ 3
1: 3 ↦ 2
1: 0 ↦ 4
1: 0 ↦ 4
0: 1 ↦ 0
0; 0 ↦ 1
0; 0 ↦ 1
0; 0 ↦ 1

Emptiness

empty$[T](q)$
Input: table T, state q of T
Output: **true** if $\mathcal{L}(q) = \emptyset$, **false** otherwise

1 **return** $q = q_\emptyset$

Universality

- if $L = \emptyset$, then L is not universal;
- if $L = \{\epsilon\}$, then L is universal;
- if $\emptyset \neq L \neq \{\epsilon\}$, then L is universal iff L^a is universal for every $a \in \Sigma$.

$univ[T](q)$
Input: table T, state q of T
Output: **true** if $\mathcal{L}(q)$ is fixed-length universal,
false otherwise

```
1  if G(q) is not empty then return G(q)
2  if q = q_∅ then return false
3  else if q = q_ε then return true
4  else  /* q ≠ q_∅ and q ≠ q_ε */
5     for all i = 1, ..., m do r_i ← comp[T](q^{a_i})
6     G(q) ← and(univ[T](r_1), ..., univ[T](r_m))
7     return G(q)
```

Inclusion and Equality

Inclusion. Given two languages $L_1, L_2 \subseteq \Sigma^n$, in order to check $L_1 \subseteq L_2$ we compute $L_1 \cap L_2$ and check whether it is equal to L_1 using the equality check shown next. The complexity is dominated by the complexity of computing the intersection.

$eq[T](q_1, q_2)$
Input: table T, states q_1, q_2 of T
Output: **true** if $\mathcal{L}(q_1) = \mathcal{L}(q_2)$, **false** otherwise
1 **return** $q_1 = q_2$

$eq[T_1, T_2](q_1, q_2)$
Input: tables T_1, T_2, states q_1 of T_1, q_2 of T_2
Output: **true** if $\mathcal{L}(q_1) = \mathcal{L}(q_2)$, **false** otherwise

1 **if** $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
2 **if** $q_1 = q_{\emptyset 1}$ and $q_2 = q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow$ `true`
3 **else if** $q_1 = q_{\emptyset 1}$ and $q_2 \neq q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow$ `false`
4 **else if** $q_1 \neq q_{\emptyset 1}$ and $q_2 = q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow$ `false`
5 **else** $/* \; q_1 \neq q_{\emptyset 1}$ and $q_2 \neq q_{\emptyset 2} \; */$
6 $G(q_1, q_2) \leftarrow$ **and**$(\mathsf{eq}(q_1^{a_1}, q_2^{a_1}), \ldots, \mathsf{eq}(q_1^{a_m}, q_2^{a_m}))$
7 **return** $G(q_1, q_2)$

What if the starting point is an NFA?

- Given: NFA A accepting a fixed-length language and containing no cycles.

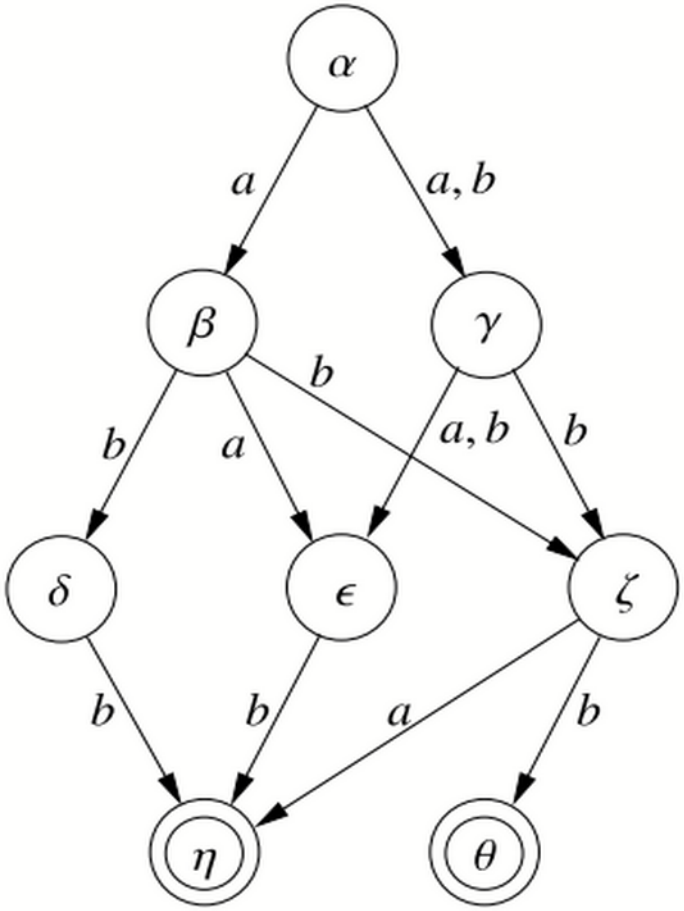
  Goal: simultaneously determinize and minimize A
- Each state of A accepts a fixed-length language.
- We give an algorithm *state*(S):
  - Input: a subset S of states of A accepting languages of the same length.
  - Output: the state of the master automaton accepting $\bigcup_{q \in S} L(q)$.
- Goal is achieved by calling *state*($\{q_0\}$)

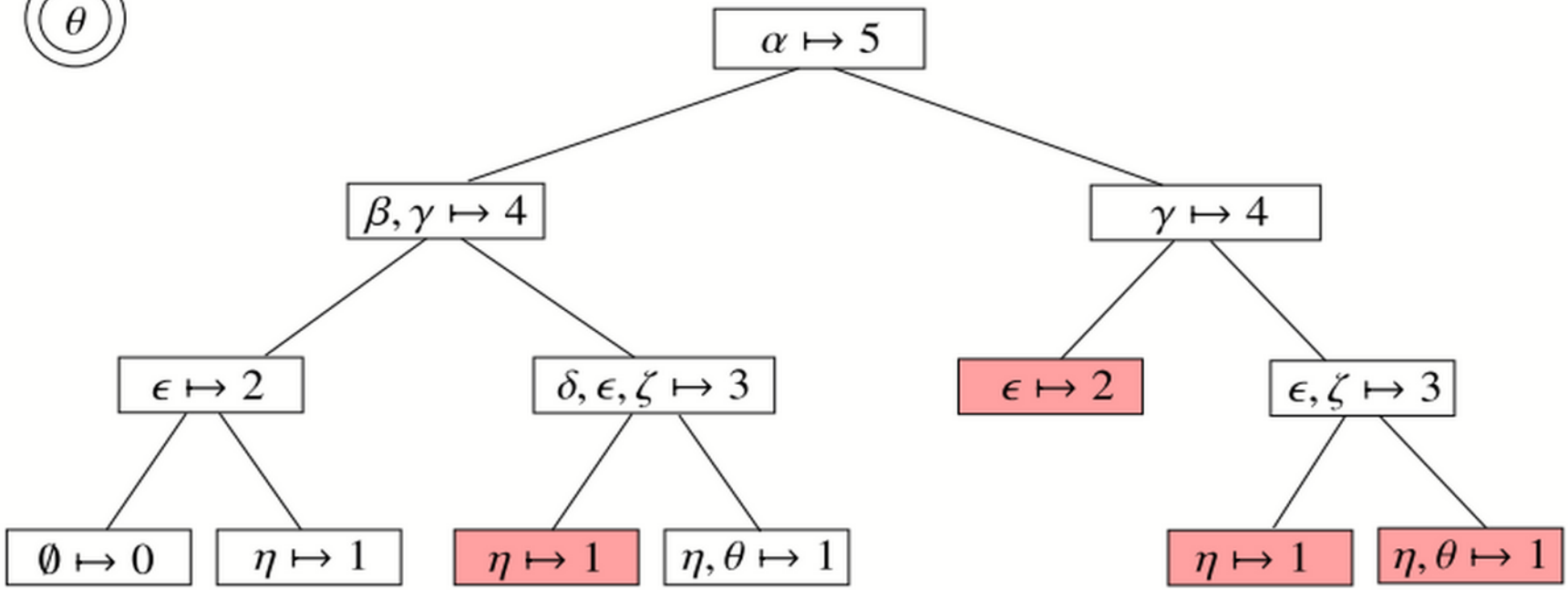
Equations:

- if $S = \emptyset$ then $\mathcal{L}(S) = \emptyset$;
- if $S \cap F \neq \emptyset$ then $\mathcal{L}(S) = \{\epsilon\}$
- if $S \neq \emptyset$ and $S \cap F = \emptyset$, then $\mathcal{L}(S) = \bigcup_{i=1}^{n} a_i \cdot \mathcal{L}(S_i)$, where $S_i = \delta(S, a_i)$.

$state[A](S)$
Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$, set $S \subseteq Q$
Output: master state recognizing $\mathcal{L}(S)$

```
1   if G(S) is not empty then return G(S)
2   else if S = ∅ then return q_∅
3   else if S ∩ F ≠ ∅ then return q_ε
4   else  /* S ≠ ∅ and S ∩ F = ∅ */
5       for all i = 1, ..., m do S_i ← δ(S, a_i)
6       G(S) ← make(state[A](S_1), ..., state[A](S_m));
7       return G(S)
```


α
a
a, b
β
γ
b
b
a
a, b
b
δ
ϵ
ζ
b
b
a
b
η
θ


α ↦ 5
β, γ ↦ 4
γ ↦ 4
ϵ ↦ 2
δ, ϵ, ζ ↦ 3
ϵ ↦ 2
ϵ, ζ ↦ 3
∅ ↦ 0
η ↦ 1
η ↦ 1
η, θ ↦ 1
η ↦ 1
η, θ ↦ 1

Operations on relations

Definition 6.10 *A word relation $R \subseteq \Sigma^* \times \Sigma^*$ has* length $n \geq 0$ *if it is empty and $n = 0$, or if it is nonempty and for all pairs (w_1, w_2) of R the words w_1 and w_2 have length n. If R has length n for some $n \geq 0$, then we say that R is a* fixed-length *word relation, or that R* has fixed-length.

Definition 6.12 *The* master transducer *over the alphabet Σ is the tuple $MT = (Q_M, \Sigma \times \Sigma, \delta_M, F_M)$, where*

- *Q_M is is the set of all fixed-length relations;*
- *$\delta_M : Q_M \times (\Sigma \times \Sigma) \to Q_M$ is given by $\delta_M(R, [a, b]) = R^{[a,b]}$ for every $q \in Q_M$ and $a, b \in \Sigma$;*
- *$F_M = \{(\varepsilon, \varepsilon)\}$.*

With T_R as the "fragment" of MT with R as root we get:

Proposition 6.13 *For every fixed-length word relation R, the transducer T_R is the minimal deterministic transducer recognizing R.*

Storing minimal transducers

Like minimal DFA, minimal deterministic transducers are represented as tables of nodes. However, a remark is in order: since a state of a deterministic transducer has $|\Sigma|^2$ successors, one for each letter of $\Sigma \times \Sigma$, a row of the table has $|\Sigma|^2$ entries, too large when the table is only sparsely filled. Sparse transducers over $\Sigma \times \Sigma$ are better encoded as NFAs over Σ by introducing auxiliary states: a transition $q \xrightarrow{[a,b]} q'$ of the transducer is "simulated" by two transitions $q \xrightarrow{a} r \xrightarrow{b} q'$, where r is an auxiliary state with exactly one input and one output transition.

Computing joins

Equations:

- $\emptyset \circ R = R \circ \emptyset = \emptyset;$
- $\{(\varepsilon, \varepsilon)\} \circ \{(\varepsilon, \varepsilon)\} = \{(\varepsilon, \varepsilon)\};$
- $R_1 \circ R_2 = \bigcup_{a,b,c \in \Sigma} [a, b] \cdot \left(R_1^{[a,c]} \circ R_2^{[c,b]} \right).$

Input: transducer table T, states q_1, q_2 of T
Output: state recognizing $\mathcal{L}(q_1) \circ \mathcal{L}(q_2)$

1 $join[T](q_1, q_2)$
2 **if** $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
3 **if** $q_1 = q_\emptyset$ **or** $q_2 = q_\emptyset$ **then return** $q_\emptyset$
4 **else if** $q_1 = q_\epsilon$ **and** $q_2 = q_\epsilon$ **then return** q_ϵ
5 **else** $/* \, q_\emptyset \neq q_1 \neq q_\epsilon, q_\emptyset \neq q_2 \neq q_\epsilon \, */$
6 **for all** $(a_i, a_j) \in \Sigma \times \Sigma$ **do**
7 $q_{a_i,a_j} \leftarrow union[T]\left(join\left(q_1^{[a_i,a_1]}, q_2^{[a_1,a_j]}\right), \ldots, join\left(q_1^{[a_i,a_m]}, q_2^{[a_m,a_j]}\right)\right)$
8 $G(q_1, q_2) = make(q_{a_1,a_1}, \ldots, q_{a_1,a_m}, \ldots, q_{a_m,a_m})$
9 **return** $G(q_1, q_2)$

Pre and Post

Pre and Post can be reduced to intersection and projection. Define:

$$emb(L) = \{[v_1, v_2] \in (\Sigma \times \Sigma)^n \mid v_2 \in L\}$$

$$pre_S(L) = \{w_1 \in \Sigma^n \mid \exists [v_1, v_2] \in S : v_1 = w_1 \text{ and } v_2 \in L\}$$

Then we have:

$$pre_S(L) \quad = \quad proj_1(S \cap emb(L))$$

We use this to derive equations.

Equations:

$$\text{if } S = \emptyset \text{ or } L = \emptyset, \text{ then } pre_S(L) = \emptyset;$$

$$\text{if } S \neq \emptyset \neq L \text{ then } pre_S(L) = \bigcup_{a,b \in \Sigma} a \cdot pre_{S^{[a,b]}}(L^b),$$

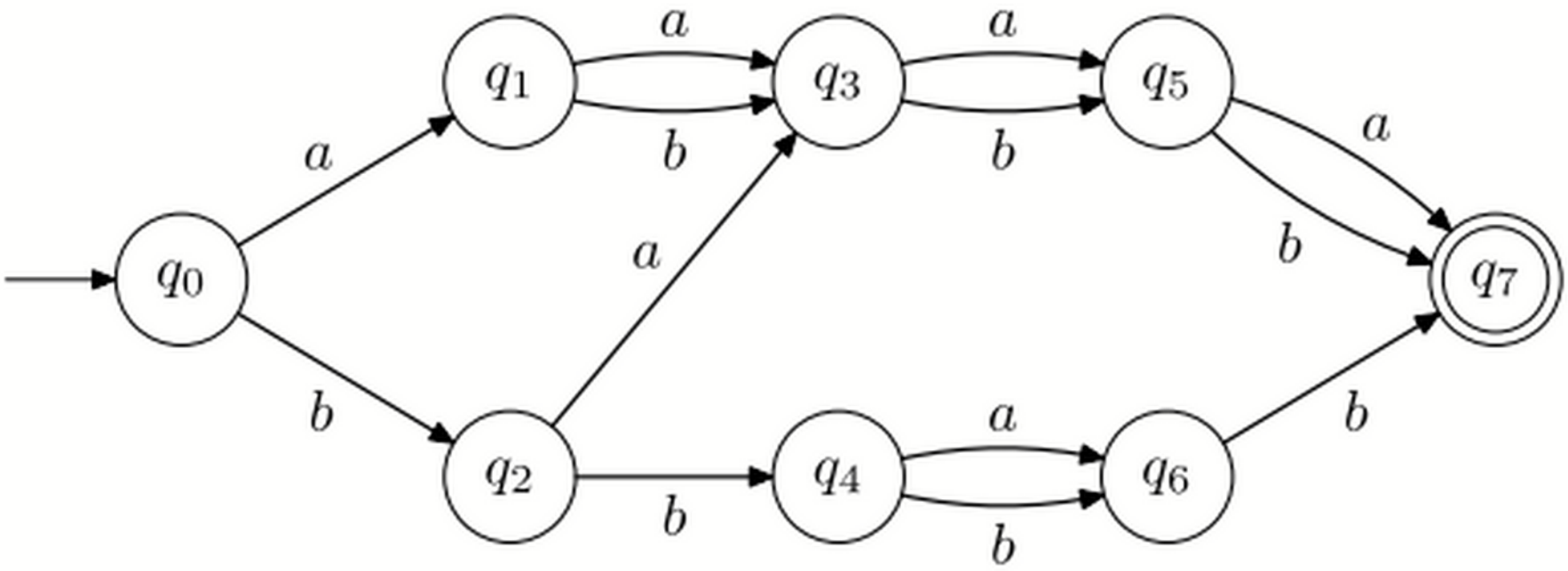
$$\text{where } S^{[a,b]} = \{w \in (\Sigma \times \Sigma)^* \mid [a,b]w \in S\}.$$

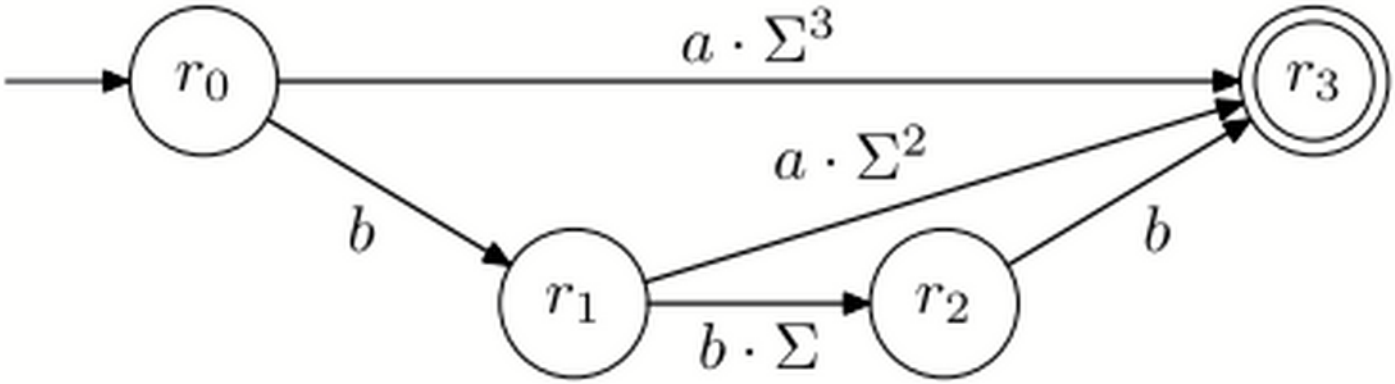
$$
\begin{aligned}
(pre_S(L))^a &= (proj_1(S \cap emb(L)))^a \\
&= \left(proj_1 \left(\bigcup_{b \in \Sigma} [a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b \in \Sigma} proj_1 \left([a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b \in \Sigma} a \cdot proj_1 \left((S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \bigcup_{b \in \Sigma} proj_1 \left((S \cap emb(L))^{[a,b]} \right) \\
&= \bigcup_{b \in \Sigma} proj_1 \left(S^{[a,b]} \cap emb\left(L^b\right) \right) \\
&= \bigcup_{b \in \Sigma} pre_S[a,b](L^b)
\end{aligned}
$$

Input: transducer table TT, table T, state r of TT, state q of T
Output: state of T recognizing $pre_{\mathcal{L}(r)}(\mathcal{L}(q))$

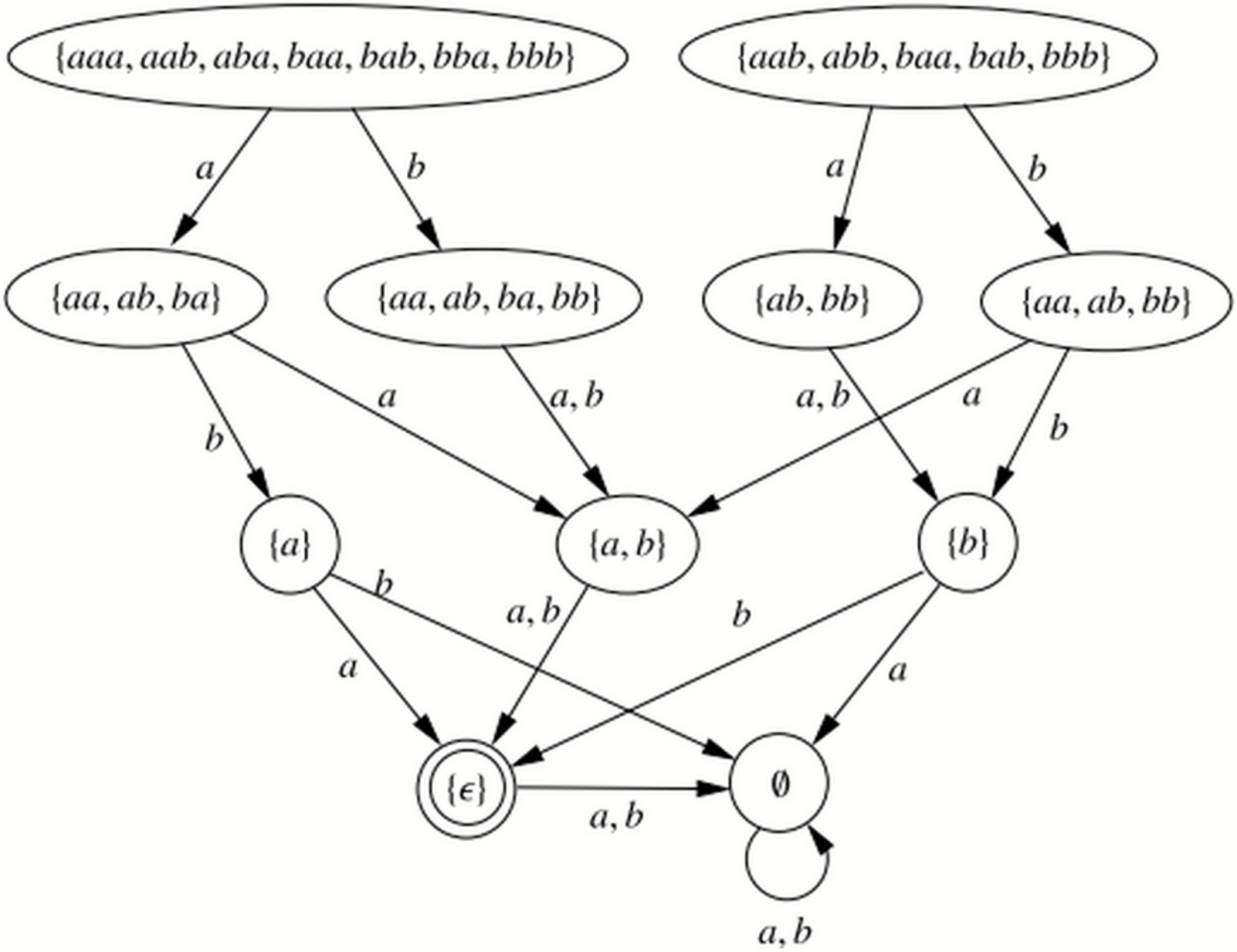
```
1  pre[TT, T](r, q)
2      if G(r, q) is not empty then return G(r, q)
3      if r = r_∅ or q = q_∅ then return q_∅
4      else if r = r_ε and q = q_ε then return q_ε
5      else
6          for all a_i ∈ Σ do
7              q_{a_i} ← union(pre[TT, T](q^{[a_i, a_1]}, r^{a_1}), ..., pre[TT, T](q^{[a_i, a_m]}, r^{a_m}))
8          G(r, q) ← make(q_{a_1}, ..., q_{a_m});
9          return G(r, q)
```

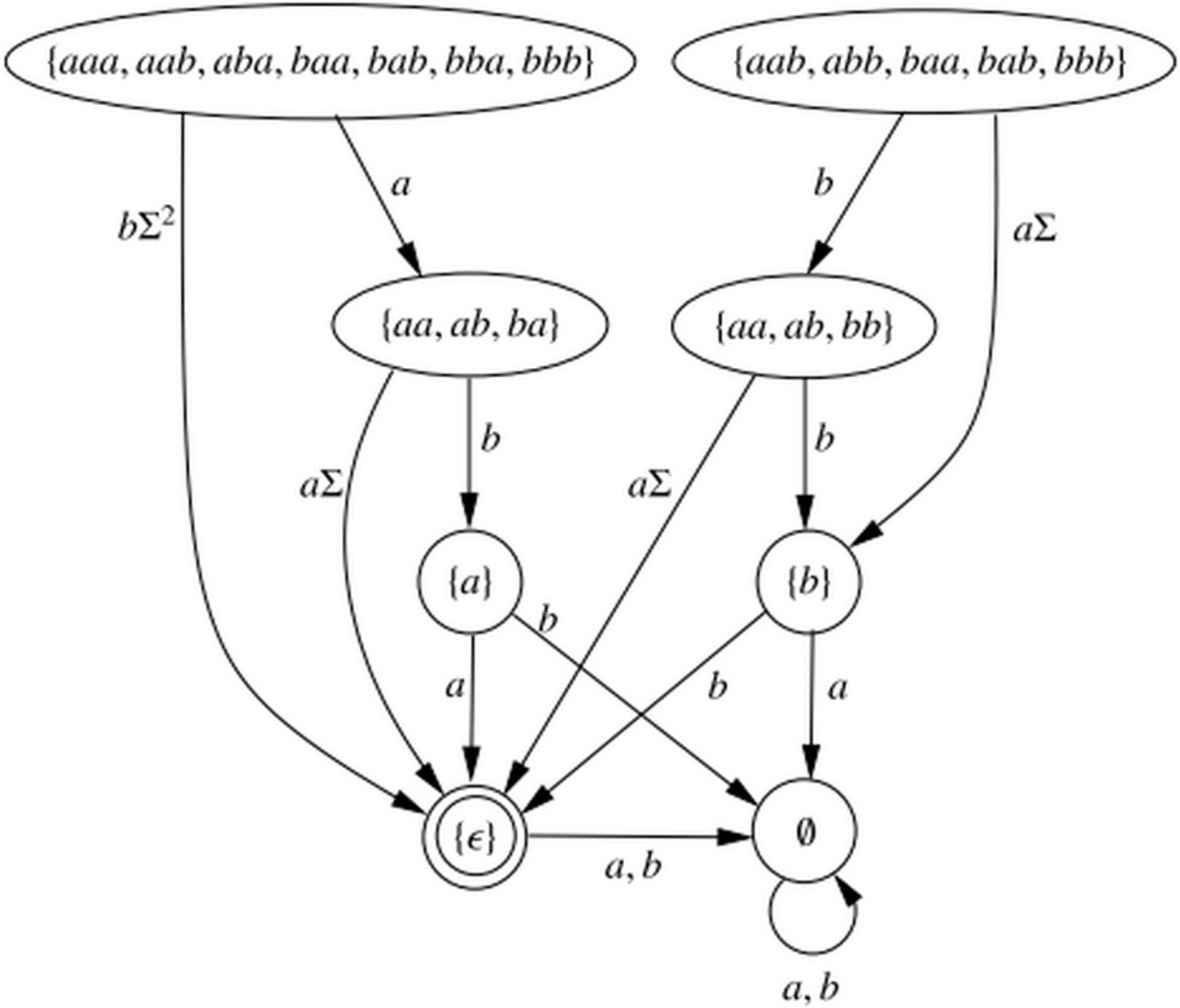
Binary Decision Diagrams



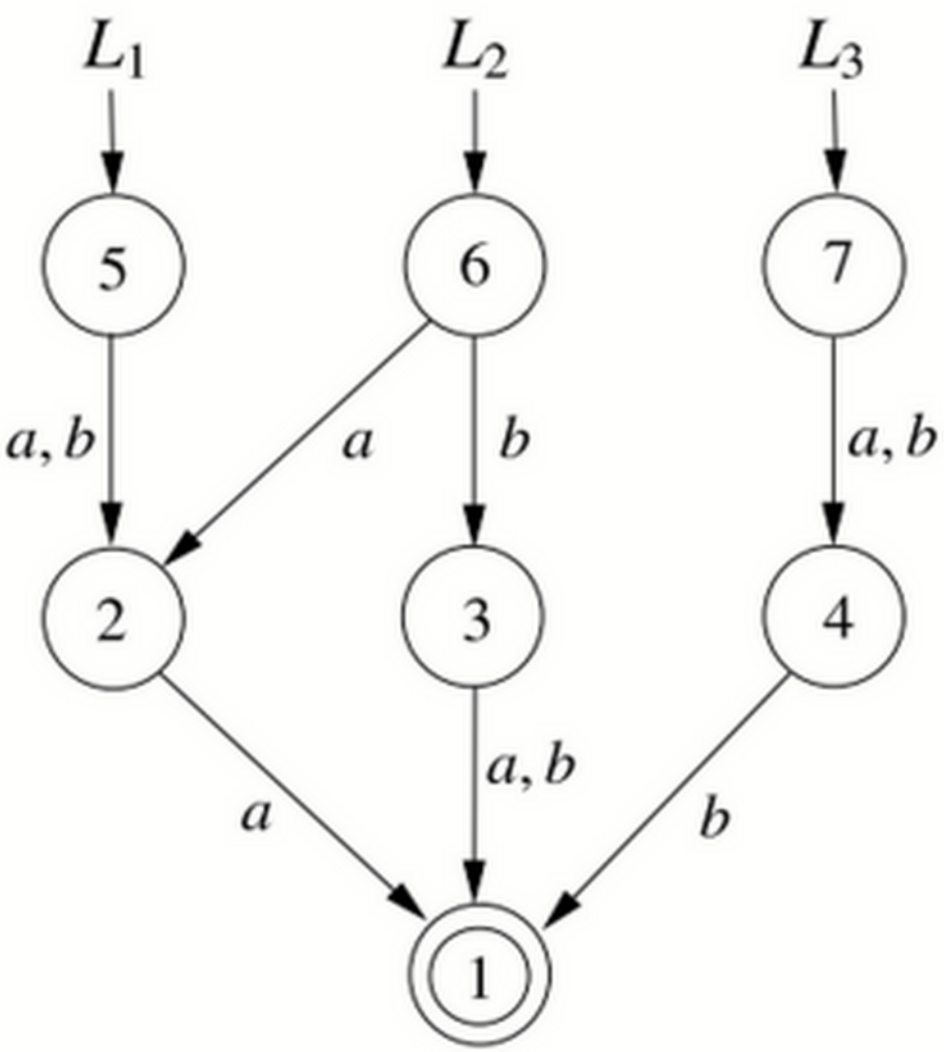


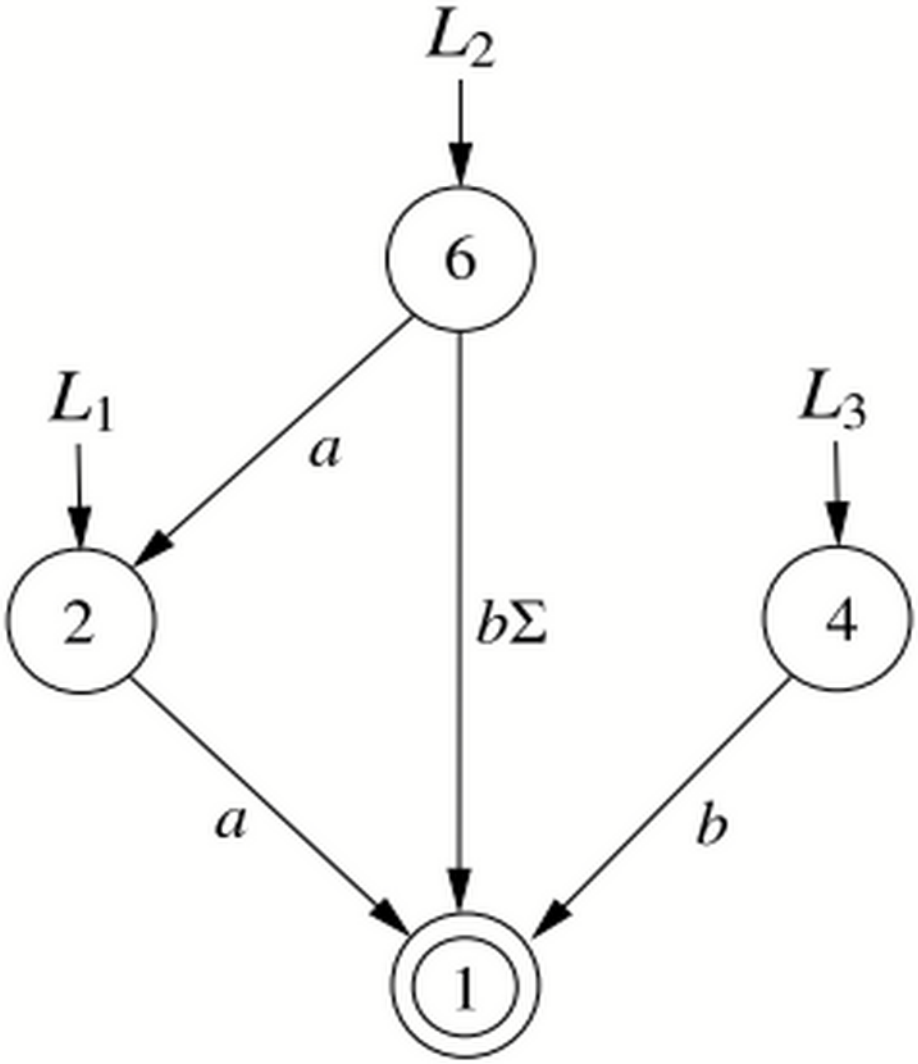
The master z-automaton



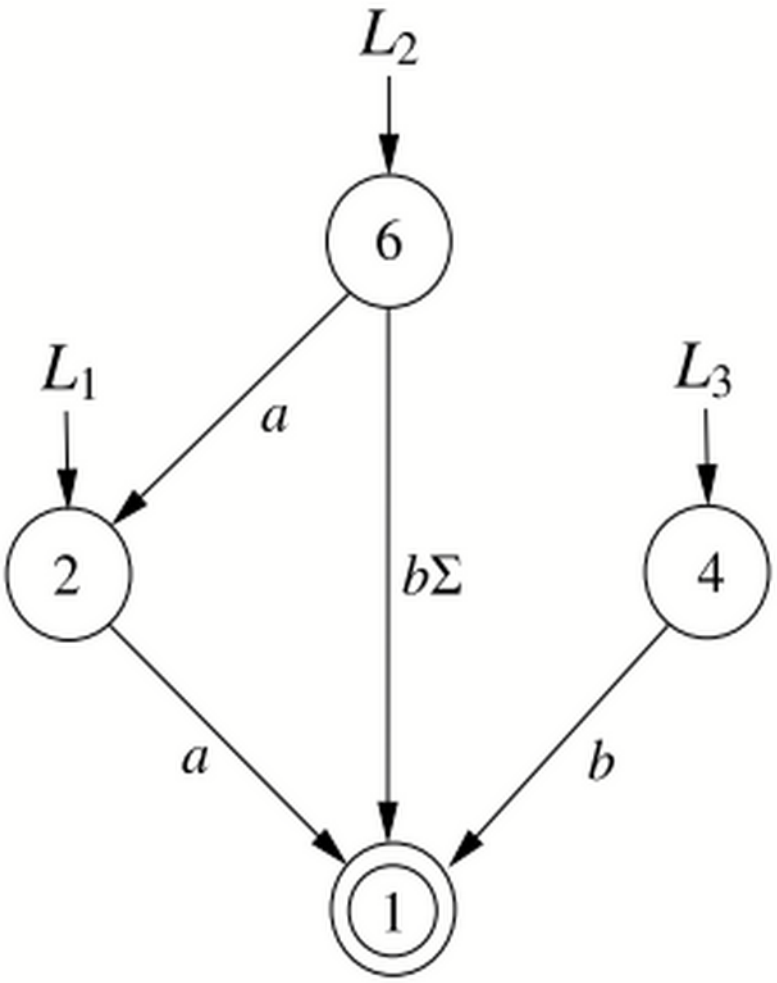

{aaa, aab, aba, baa, bab, bba, bbb}
{aab, abb, baa, bab, bbb}
a
b
bΣ²
aΣ
{aa, ab, ba}
{aa, ab, bb}
b
b
aΣ
aΣ
{a}
{b}
b
a
b
a
{ϵ}
∅
a, b
a, b

Length: 2





Data structure for z-automata



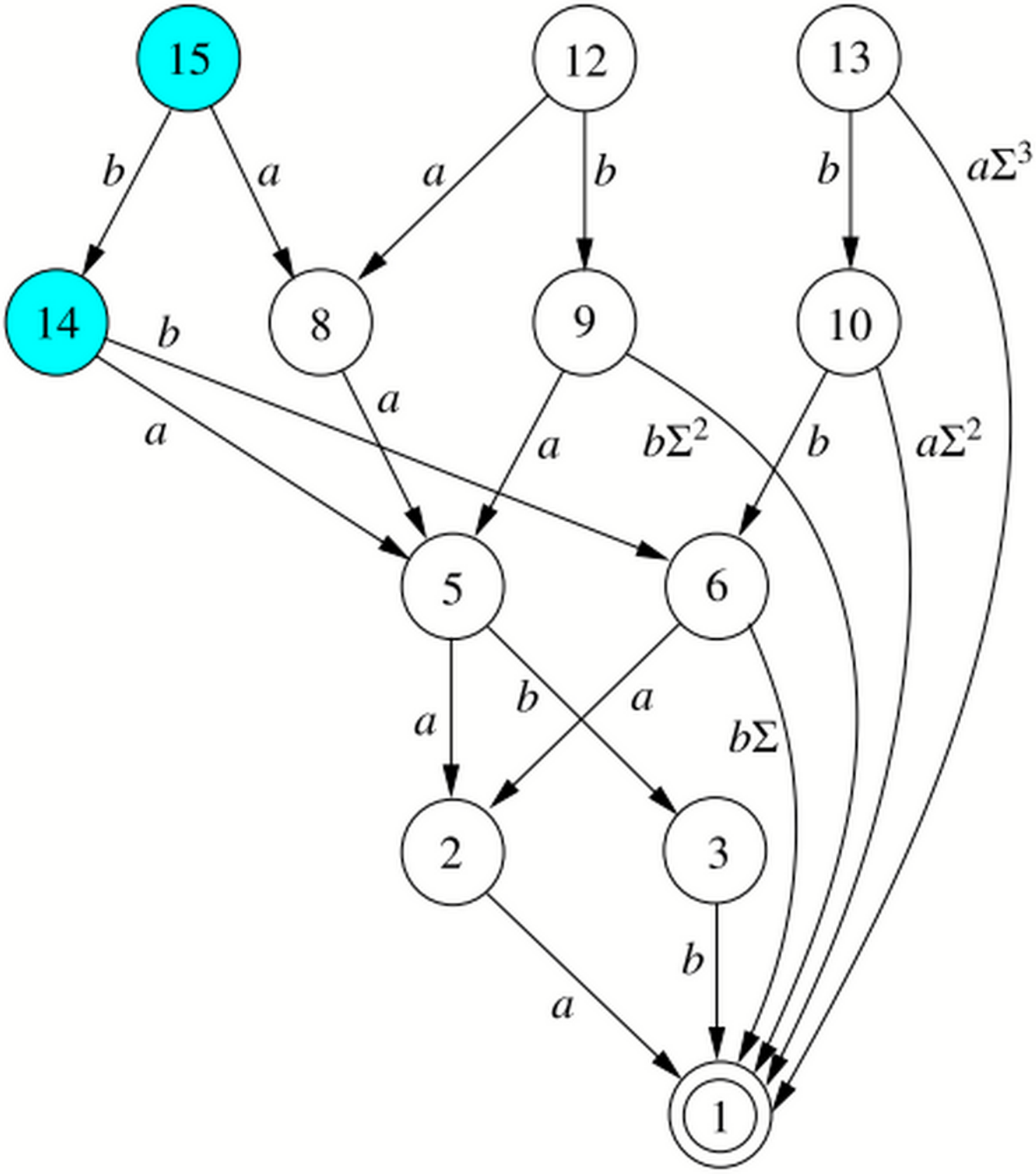
Ident.	Length	a-succ	b-succ
1	0	0	0
2	1	1	0
4	1	0	1
6	2	2	1

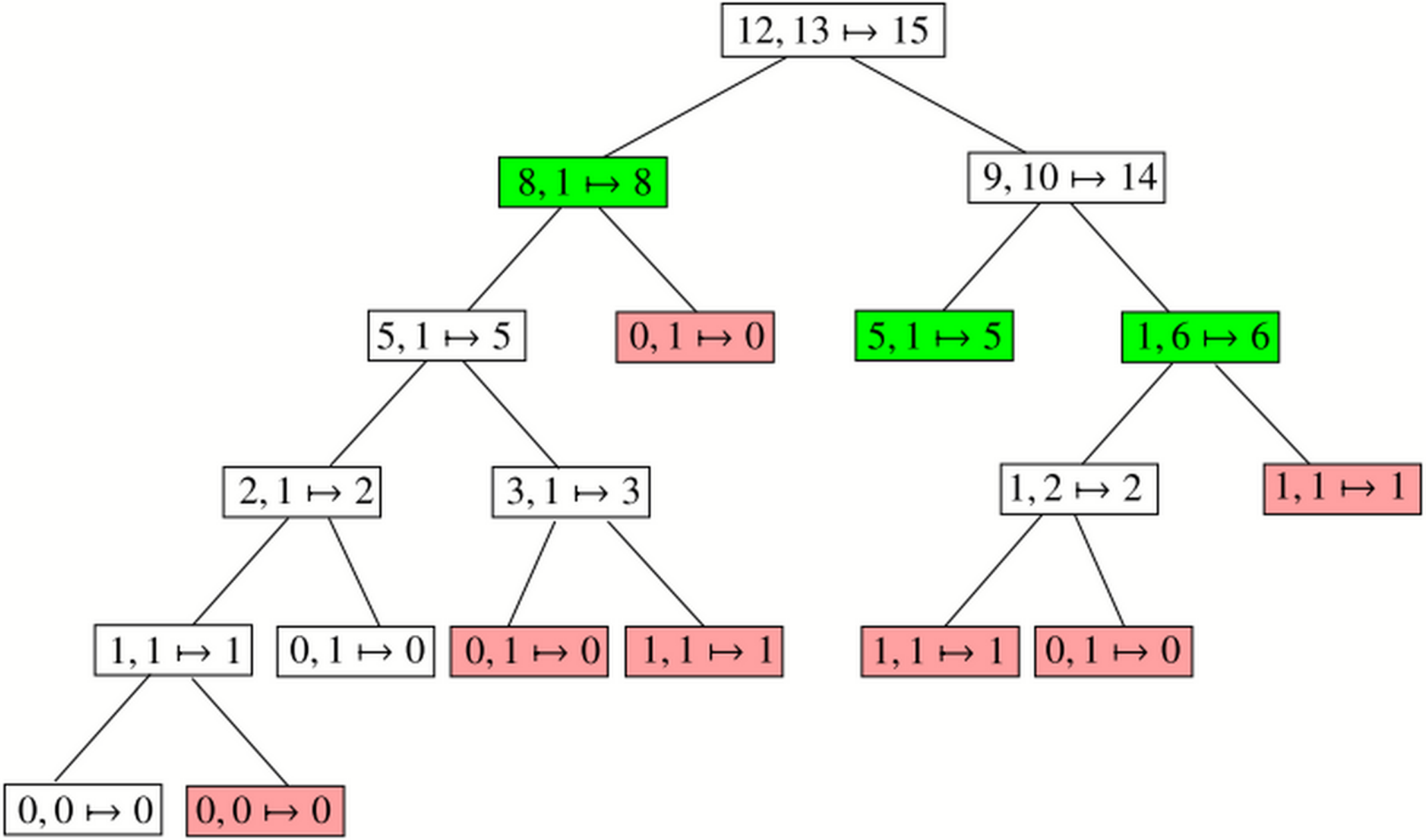
kinter(q_1, q_2)

Input: states q_1, q_2 recognizing $\langle L_1 \rangle, \langle L_2 \rangle$

Output: state recognizing $\langle L_1 \cap L_2 \rangle$

```
1   if G(q1, q2) is not empty  then return G(q1, q2)
2   if q1 = q∅ or q2 = q∅ then return q∅
3   if q1 ≠ q∅ and q2 ≠ q∅ then
4      if l1 < l2  /* lengths of the kernodes for q1, q2 */ then
5         for all i = 1, ..., m do ri ← kinter(q1, q2^ai)
6         G(q1, q2) ← kmake(l2, r1, ..., rm)
7      else if l1    l2 then
8         for all i = 1, ..., m do ri ← kinter(q1^ai, q2)
9         G(q1, q2) ← kmake(l1, r1, ..., rm)
10     else /* l1 = l2 */
11        for all i = 1, ..., m do ri ← kinter(q1^ai, q2^ai)
12        G(q1, q2) ← kmake(l1, r1, ..., rm)
13  return G(q1, q2)
```


15
12
13
14
8
9
10
5
6
2
3
1
b
a
a
b
b
aΣ3
b
a
a
bΣ2
b
aΣ2
a
b
a
bΣ
a
b
a


12, 13 ↦ 15
8, 1 ↦ 8
9, 10 ↦ 14
5, 1 ↦ 5
0, 1 ↦ 0
5, 1 ↦ 5
1, 6 ↦ 6
2, 1 ↦ 2
3, 1 ↦ 3
1, 2 ↦ 2
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
0, 1 ↦ 0
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
0, 0 ↦ 0
0, 0 ↦ 0