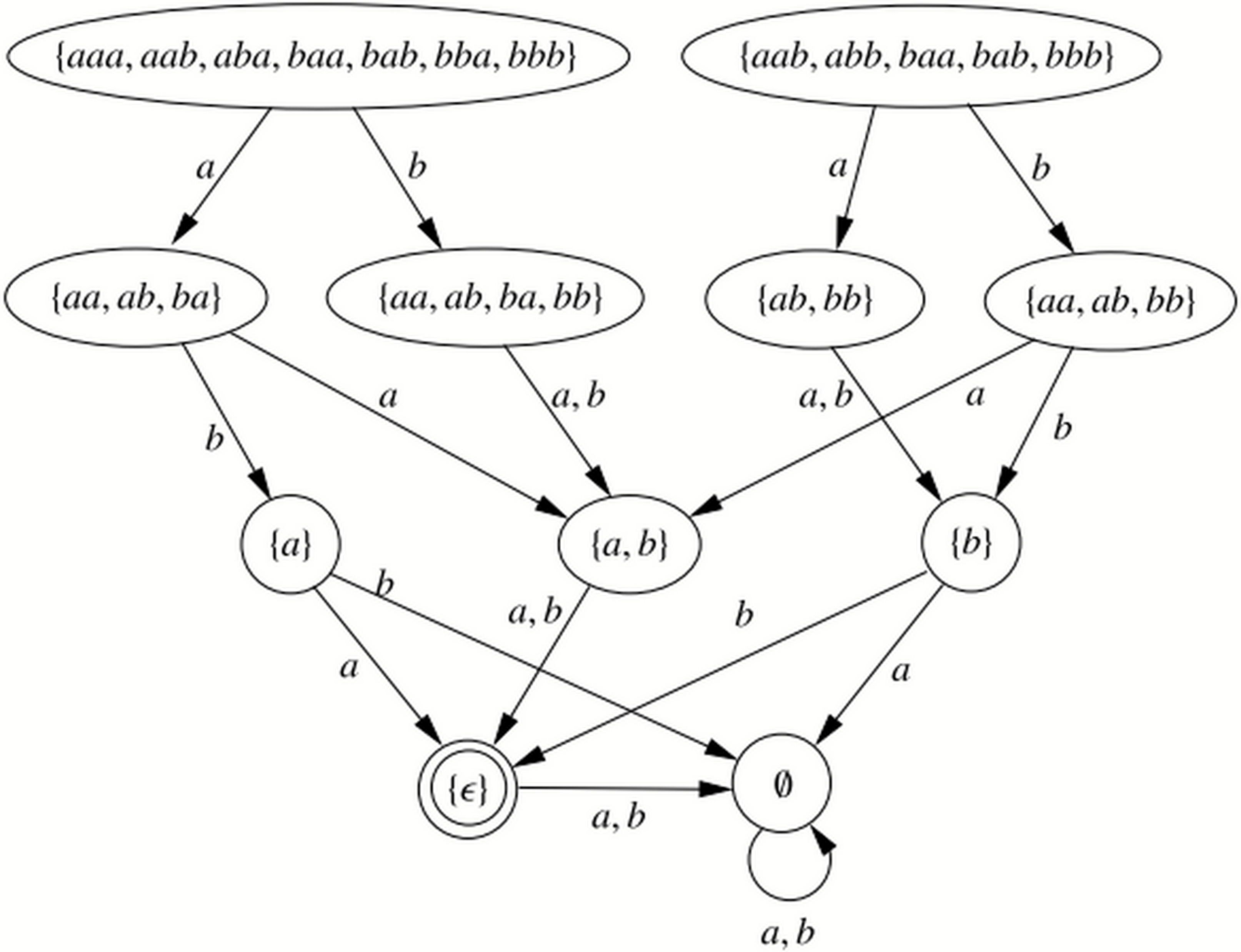
Finite Universes

When the universe is finite (e.g., the interval [0, 2^32 -1]), then all objects can be encoded by words of the same length.

Definition 6.1 *A language $L \subseteq \Sigma^*$ has* length $n \geq 0$ *if it is empty and $n = 0$, or if it is nonempty and all words of L have length n. If L has length n for some $n \geq 0$, then we say that L is a* fixed-length *language, or that L* has fixed-length.

Notice that every fixed-length language contains only finitely many words, and so it is automatically regular. Observe also that there are exactly two languages of length 0: the empty language $\emptyset$, and the language $\{\varepsilon\}$ containing only the empty word.

The Master Automaton



Definition 6.2 *The* master automaton *over the alphabet* Σ *is the tuple* $M = (Q_M, \Sigma, \delta_M, F_M)$, *where*

- Q_M *is is the set of all fixed-length languages;*
- $\delta \colon Q_M \times \Sigma \to Q_M$ *is given by* $\delta(L, a) = L^a$ *for every* $q \in Q_M$ *and* $a \in \Sigma$*;*
- F_M *is the singleton set containing the language* $\{\epsilon\}$ *as only element.*

Proposition 6.4 *Let L be a fixed-length language. The language recognized from the state L of the master automaton is L.*

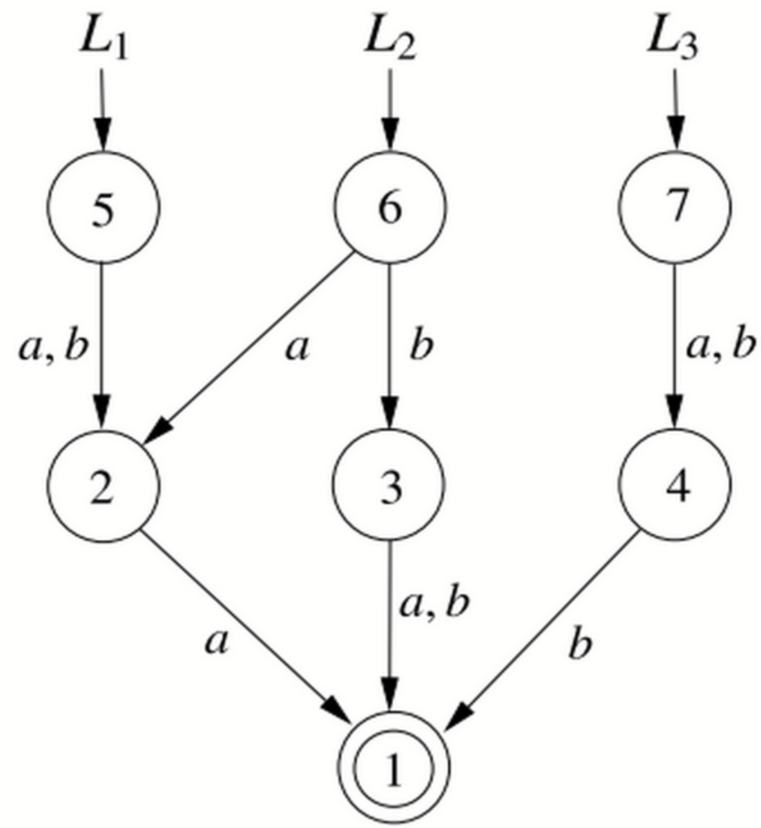
Proof: By induction on the length n of L. If $n = 0$, then $L = \{\epsilon\}$ or $L = \emptyset$, and the result is proved by direct inspection of the master automaton. For $n > 0$ we observe that the successors of the initial state L are the languages L^a for every $a \in \Sigma$. Since, by induction hypothesis, the state L^a recognizes the language L^a, the state L recognizes the language L. $\square$

Proposition 6.5 *For every fixed-language L, the automaton* A_L *is the minimal DFA recognizing L.*

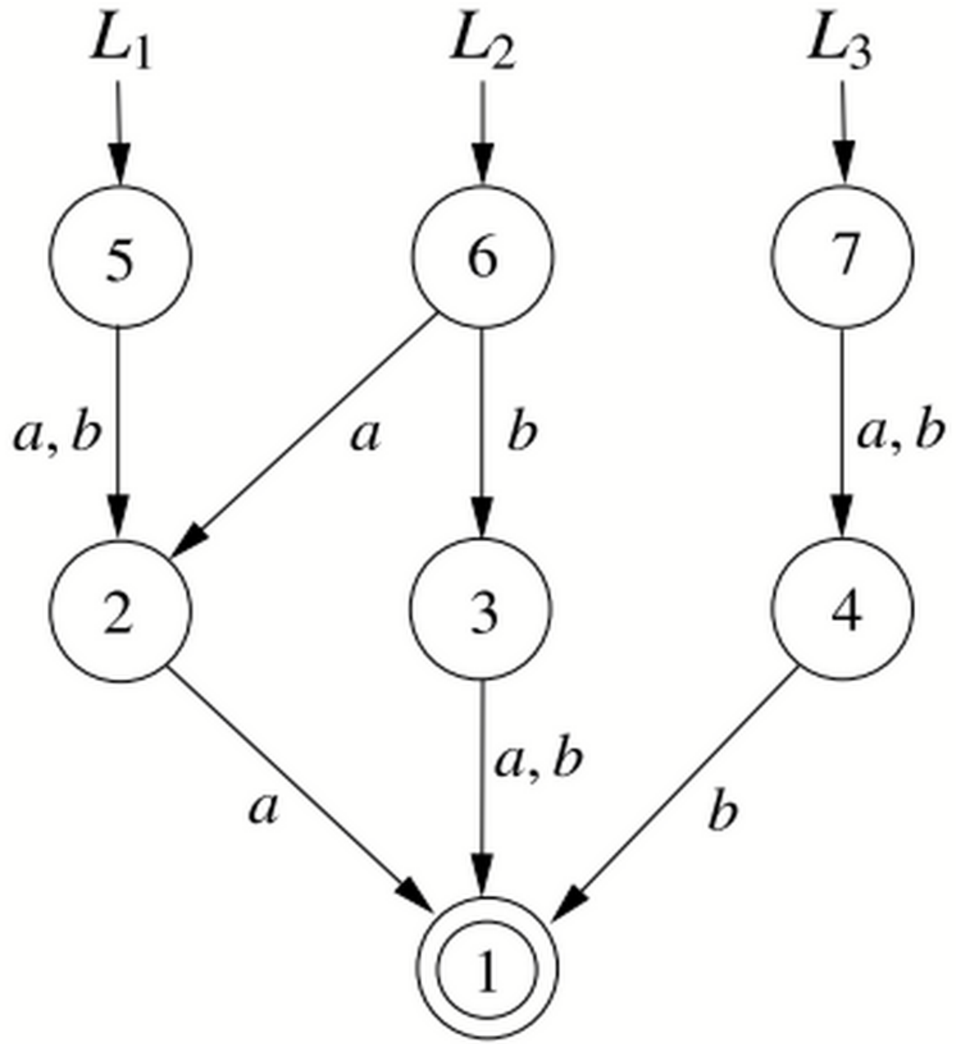
Proof: By Proposition 6.4, distinct states of recognize distinct languages. By definition of A_L, all states are reachable from the initial state. So A_L is minimal. $\square$

Data structure for fixed-length languages

Proposition 6.5 allows to define a data structure for representing *finite sets of fixed-length languages*, all of them *of the same length*. Loosely speaking, the structure representing the languages $\mathcal{L} = \{L_1, \ldots, L_m\}$ is the fragment of the master automaton containing the states recognizing $L_1, \ldots, L_n$ and their descendants. It is a DFA with multiple initial states, and for this reason we call it *the multi-DFA for* $\mathcal{L}$.



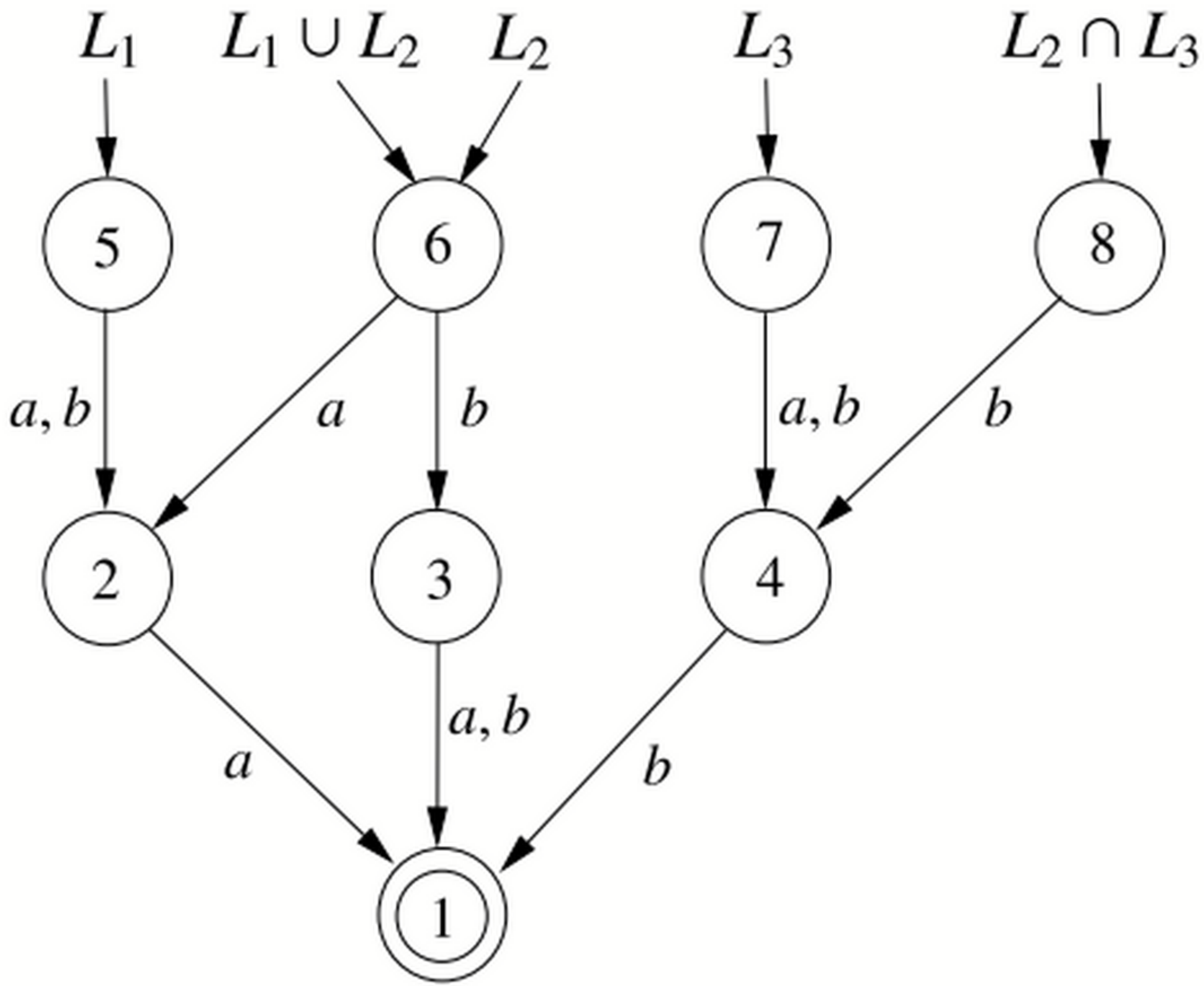
In order to manipulate multi-DFAs we represent them as a *table of nodes*. Assume $\Sigma = \{a_1, \ldots, a_m\}$. A *node* is a pair $\langle q, s \rangle$, where q is a *state identifier* and $s = (q_1, \ldots, q_m)$ is the *successor tuple* of the node. The multi-DFA is represented by a table containing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



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

The procedure $make[T](s)$**.** The algorithms on multi-DFAs use a procedure $make[T](s)$ that returns the state of T having s as successor tuple, if such a state exists; otherwise, it adds a new node $\langle q, s\rangle$ to T, where q is a fresh state identifier (different from all other state identifiers in T) and returns q. The procedure assumes that all the states of the tuple s appear in T.[2] For instance, if T is the table above, then $make[T](2, 0)$ returns 5, but $make[T](3, 1)$ adds a new row, say 8, 3, 1, and returns 8.

Implementing union and intersection



Given two fixed-length languages L_1, L_2 *of the same length* and a boolean operation $\odot$, there is a generic algorithm that returns the state of the master automaton recognizing $L_1 \widehat{\odot} L_2$. Let us consider the case of intersection, the other cases being similar. The 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 $L_1 \neq \emptyset \neq L_2$, then $(L_1 \cap L_2)^a = L_1^a \cap L_2^a$ for every $a \in \Sigma$;

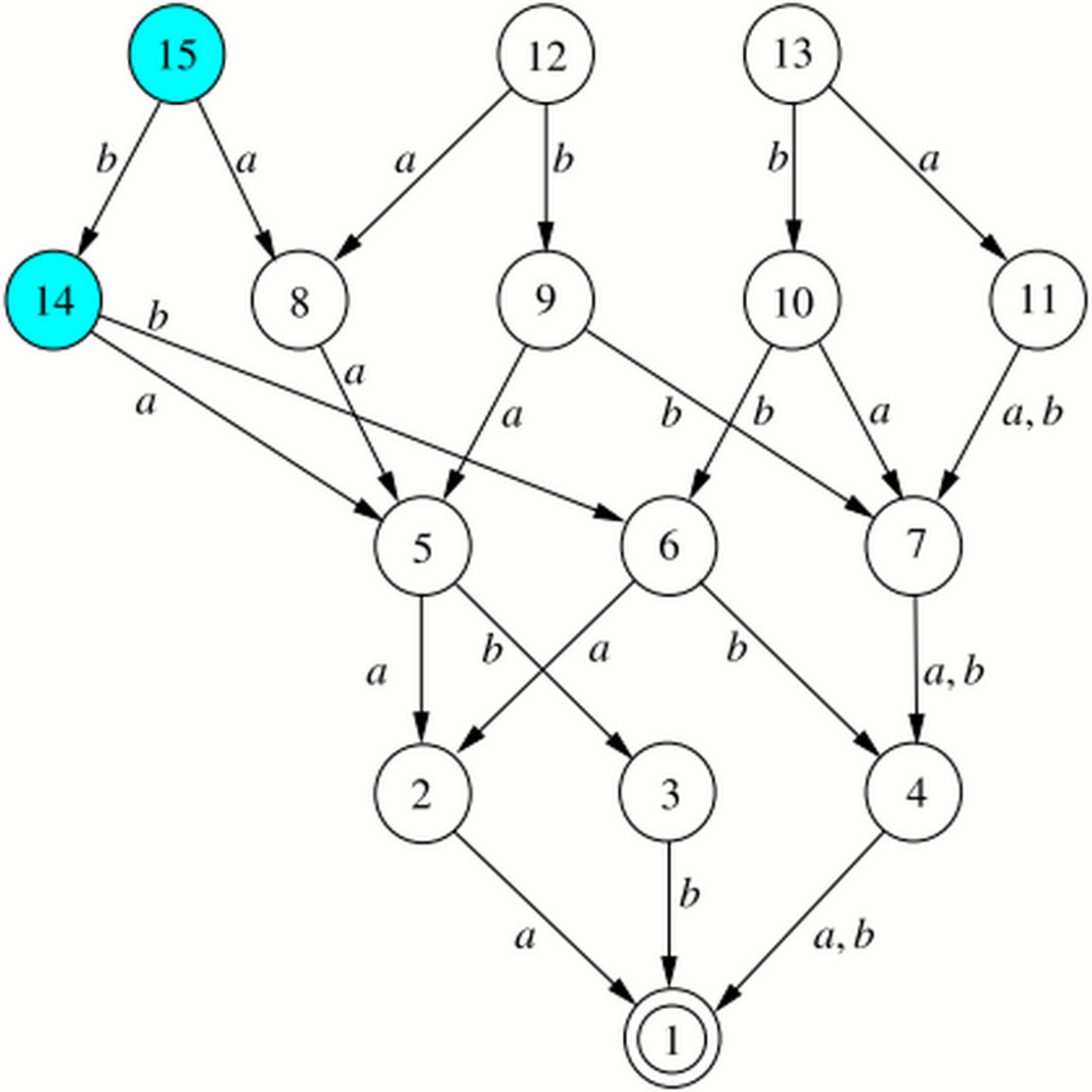
lead to the recursive algorithm *inter*$[T](q_1, q_2)$ shown in Table 6.1. Assume the states q_1, q_2 recognize the languages L_1, L_2. The algorithm returns the state identifier $q_{L_1 \cup L_2}$.

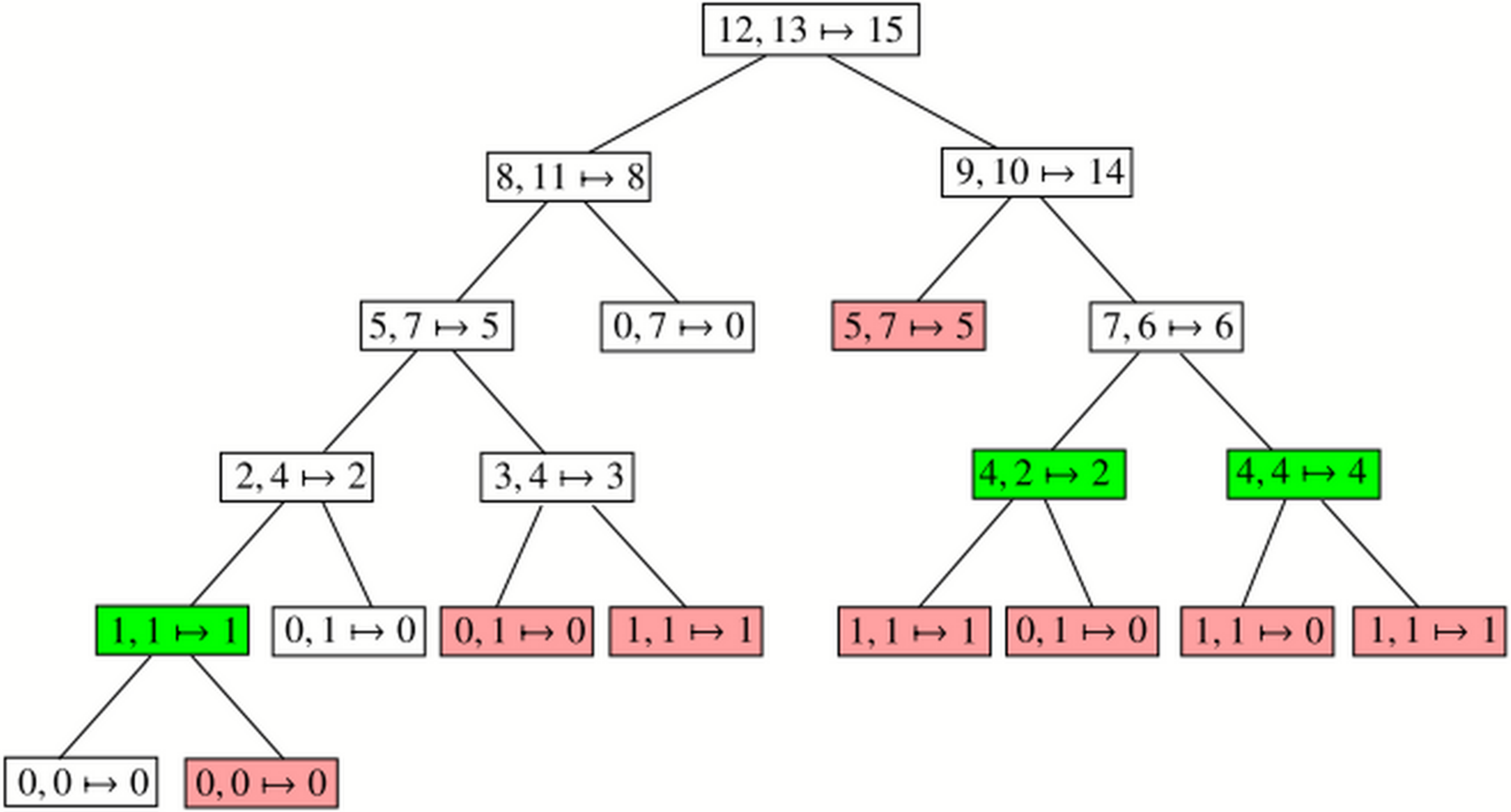
$inter[T](q_1, q_2)$

Input: table T, states q_1, q_2 of T

Output: state recognizing $\mathcal{L}(q_1) \cap \mathcal{L}(q_2)$

if $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
if $q_1 = q_\emptyset \vee q_2 = q_\emptyset$ **then return** $q_\emptyset$
if $q_1 \neq q_\emptyset \wedge q_2 \neq q_\emptyset$ **then**
    for all $i = 1, \ldots, m$ **do** $r_i \leftarrow inter[T](q_1^{a_i}, q_2^{a_i})$
    $G(q_1, q_2) \leftarrow \mathtt{make}[T](r_1, \ldots, r_m)$
    return $G(q_1, q_2)$


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


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 $L(q)$

if $G(q)$ is not empty **then return** $G(q)$
if $n = 0$ **and** $q = q_\emptyset$ **then return** q_ϵ
else if $n = 0$ **and** $q = q_\epsilon$ **then return** $q_\emptyset$
else $/* \ n \geq 1 \ */$
    for all $i = 1, \ldots, m$ **do** $r_i \leftarrow comp[T, n-1](q^{a_i})$
    $G(q) \leftarrow \texttt{make}[T](r_1, \ldots, r_m)$
    return $G(q)$

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

if $G(q)$ is not empty **then return** $G(q)$
if $q = q_\emptyset$ **then return false**
else if $q = q_\epsilon$ **then return true**
else $/ * \; q \neq q_\emptyset$ and $q \neq q_\epsilon \; * /$
    for all $i = 1, \ldots, m$ **do** $r_i \leftarrow comp[T](q^{a_i})$
    $G(q) \leftarrow$ **and**$(univ[T](r_1), \ldots, univ[T](r_m))$
    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

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

if $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
if $q_1 = q_{\emptyset 1}$ and $q_2 = q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow \texttt{true}$
else if $q_1 = q_{\emptyset 1}$ and $q_2 \neq q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow \texttt{false}$
else if $q_1 \neq q_{\emptyset 1}$ and $q_2 = q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow \texttt{false}$
else $/ * \; q_1 \neq q_{\emptyset 1}$ and $q_2 \neq q_{\emptyset 2} \; * /$
    $G(q_1, q_2) \leftarrow$ **and**$(\mathtt{eq}(q_1^{a_1}, q_2^{a_1}), \ldots, \mathtt{eq}(q_1^{a_m}, q_2^{a_m}))$
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

We determinize and minimize A in one single pass

Observe: each state of A accepts a fixed-length language

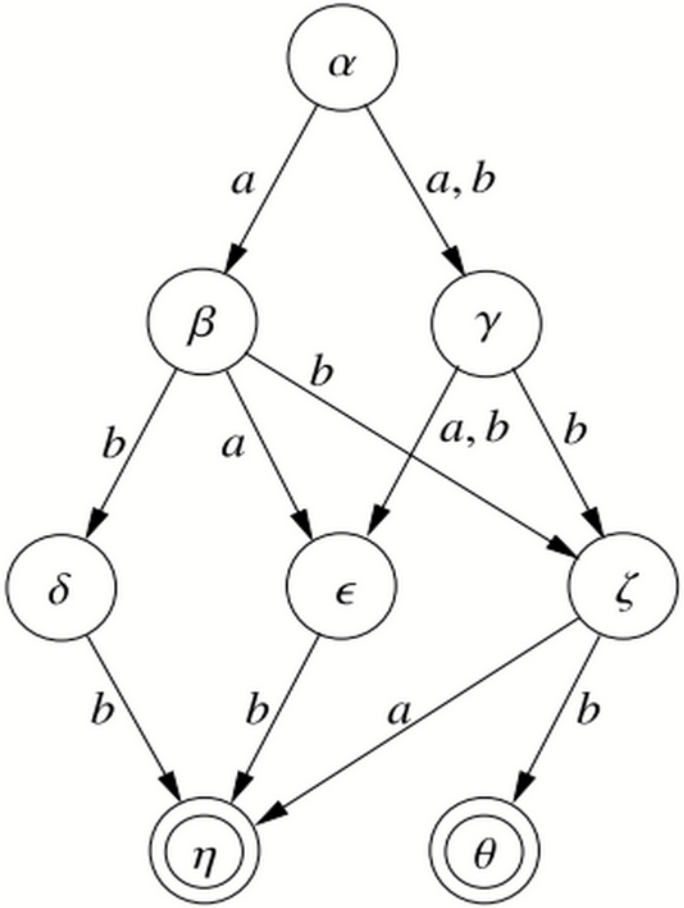
We give an algorithm state(S) :
input: a subset S of states of the NFA accepting languages of the same length
output: the state of the master automaton accepting the union of the languages accepted from the states of S

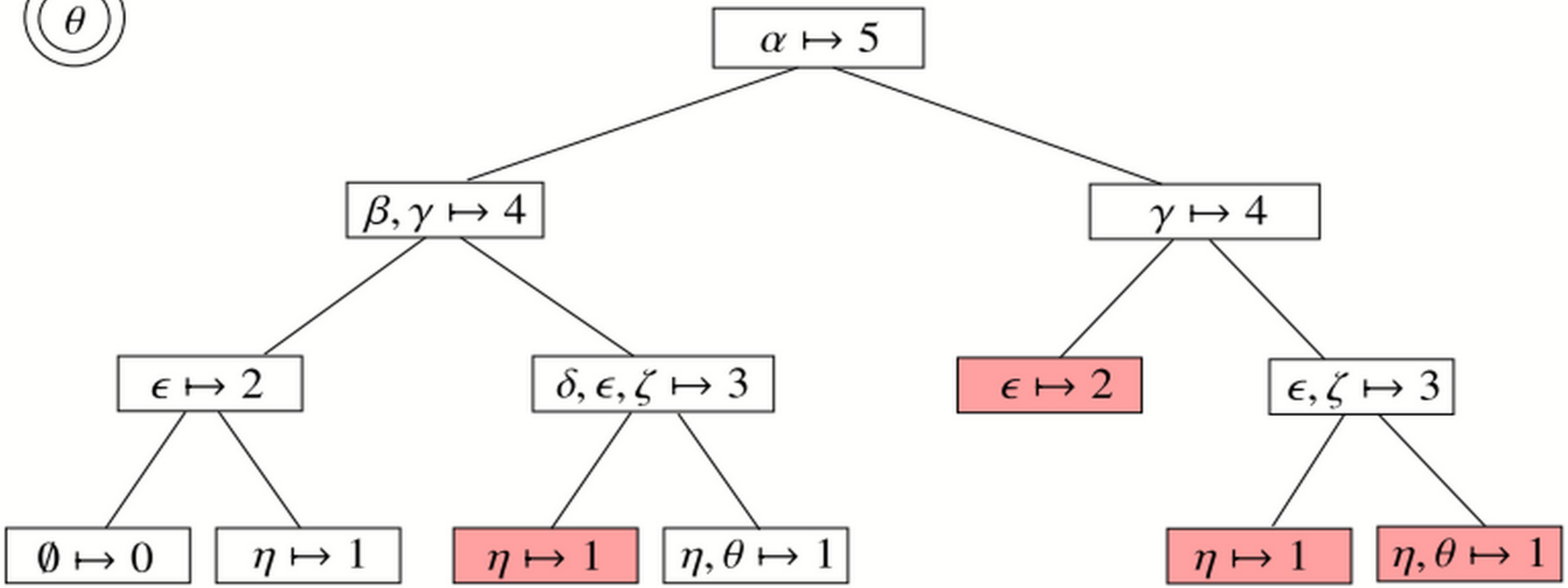
Equations:

- 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 $\mathcal{L}(S)$

if $G(S)$ is not empty **then return** $G(S)$
else if $S = \emptyset$ **then return** $q_\emptyset$
else if $S \cap F \neq \emptyset$ **then return** q_ϵ
else $/ * S \neq \emptyset$ and $S \cap F = \emptyset * /$
  for all $i = 1, \ldots, m$ **do** $S_i \leftarrow \delta(S, a_i)$
  $G(S) \leftarrow make(state[A](S_1), \ldots, state[A](S_m))$;
  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 $\mathcal{L}(q_1) \circ \mathcal{L}(q_2)$

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

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

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

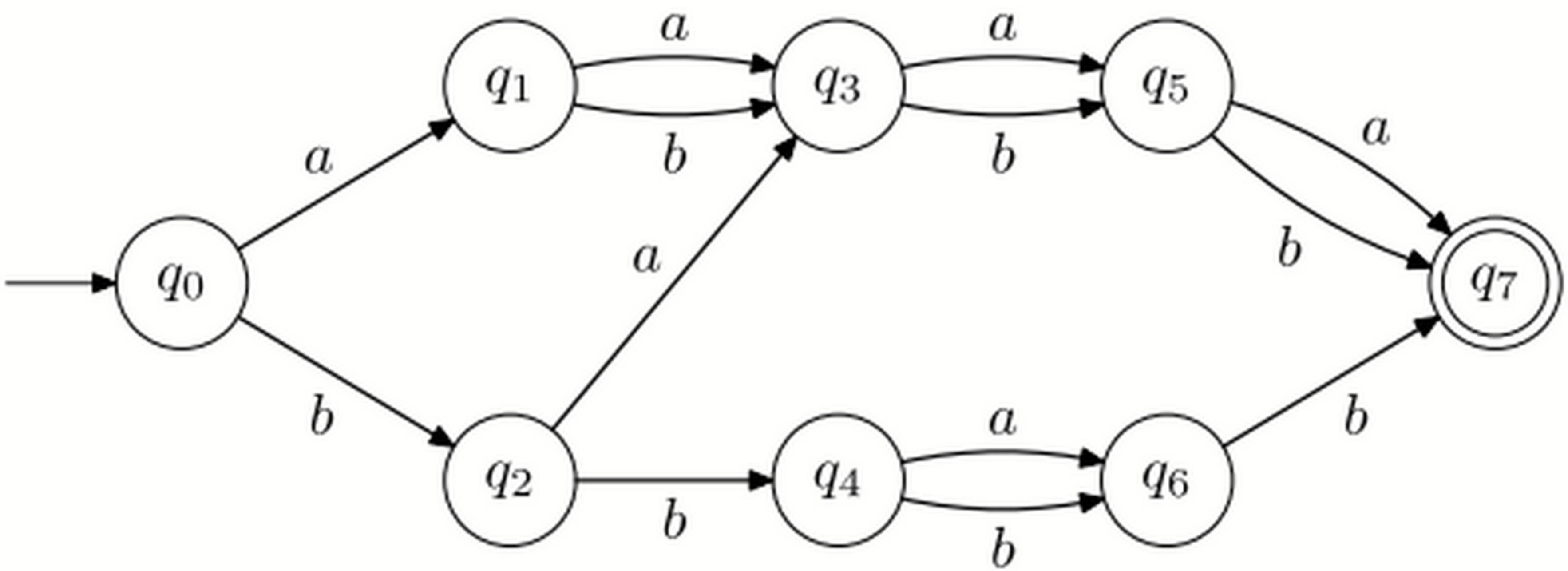
$$\textit{where } S^{[a,b]} = \{w \in (\Sigma \times \Sigma)^* \mid [a,b]w \in S\}.$$

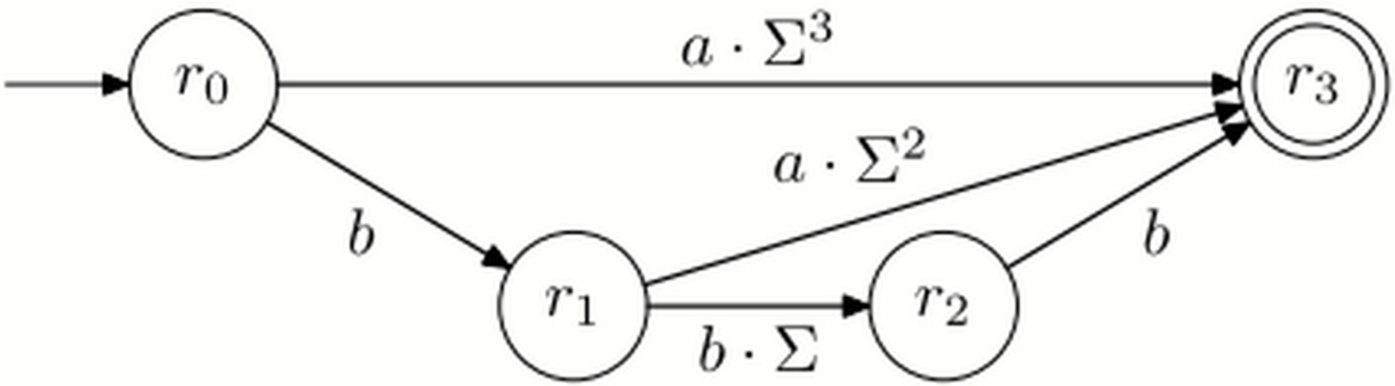
$$
\begin{aligned}
(pre_S(L))^a &= (proj_1(S \cap emb(L)))^a \\
&= \left(proj_1 \left(\bigcup_{b\in\Sigma} [a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b\in\Sigma} proj_1 \left([a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b\in\Sigma} a \cdot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

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

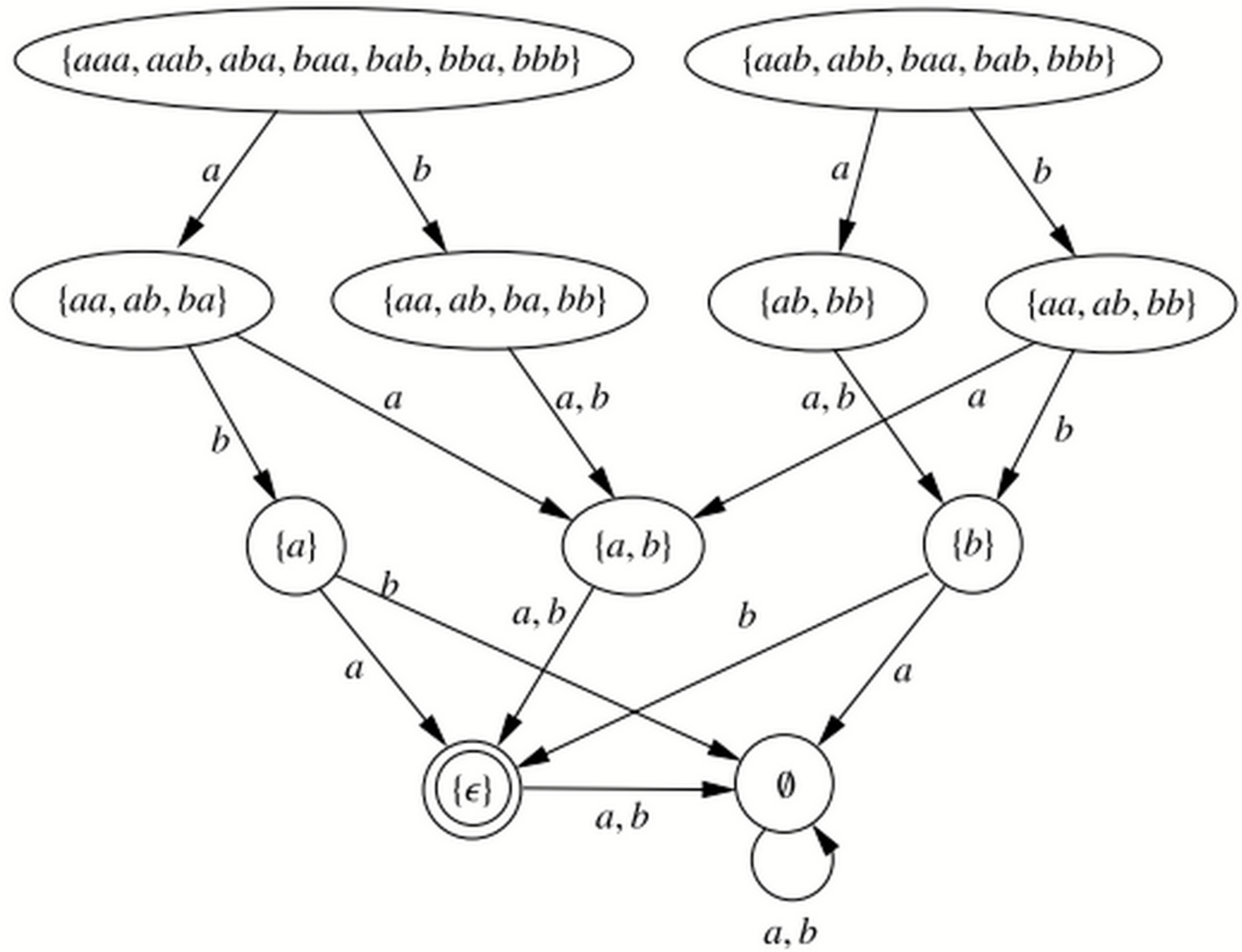
$pre[TT, T](r, q)$
  if $G(r, q)$ is not empty **then return** $G(r, q)$
  if $r = r_\emptyset$ **or** $q = q_\emptyset$ **then return** $q_\emptyset$
  else if $r = r_\epsilon$ **and** $q = q_\epsilon$ **then return** q_ϵ
  else
    for all $a_i \in \Sigma$ **do**
      $q_{a_i} \leftarrow union\left(pre[TT, T]\left(q^{[a_i, a_1]}, r^{a_1}\right), \ldots, pre[TT, T]\left(q^{[a_i, a_m]}, r^{a_m}\right)\right)$
    $G(r, q) \leftarrow make(q_{a_1}, \ldots, q_{a_m})$;
    return $G(r, q)$

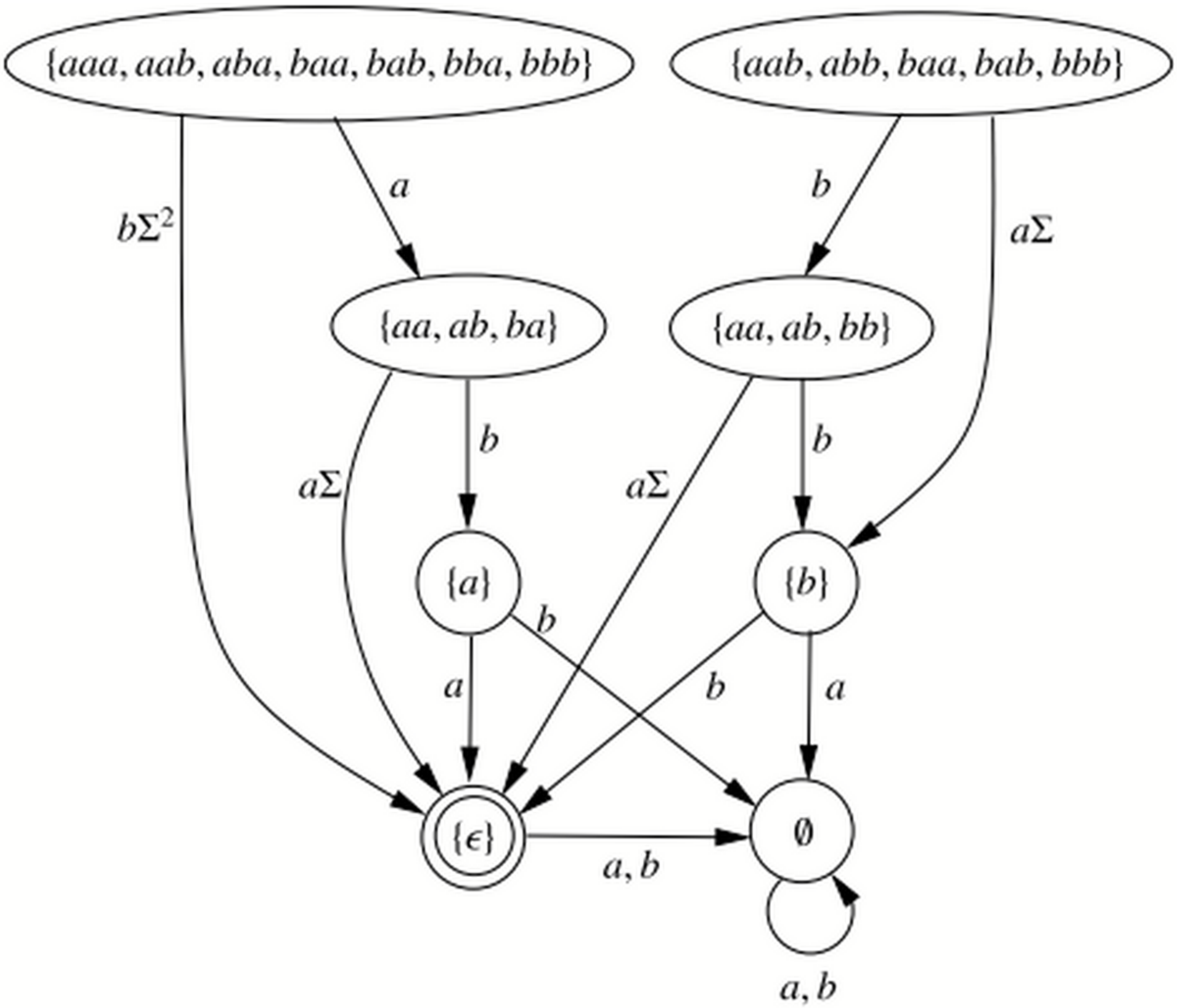
Binary Decision





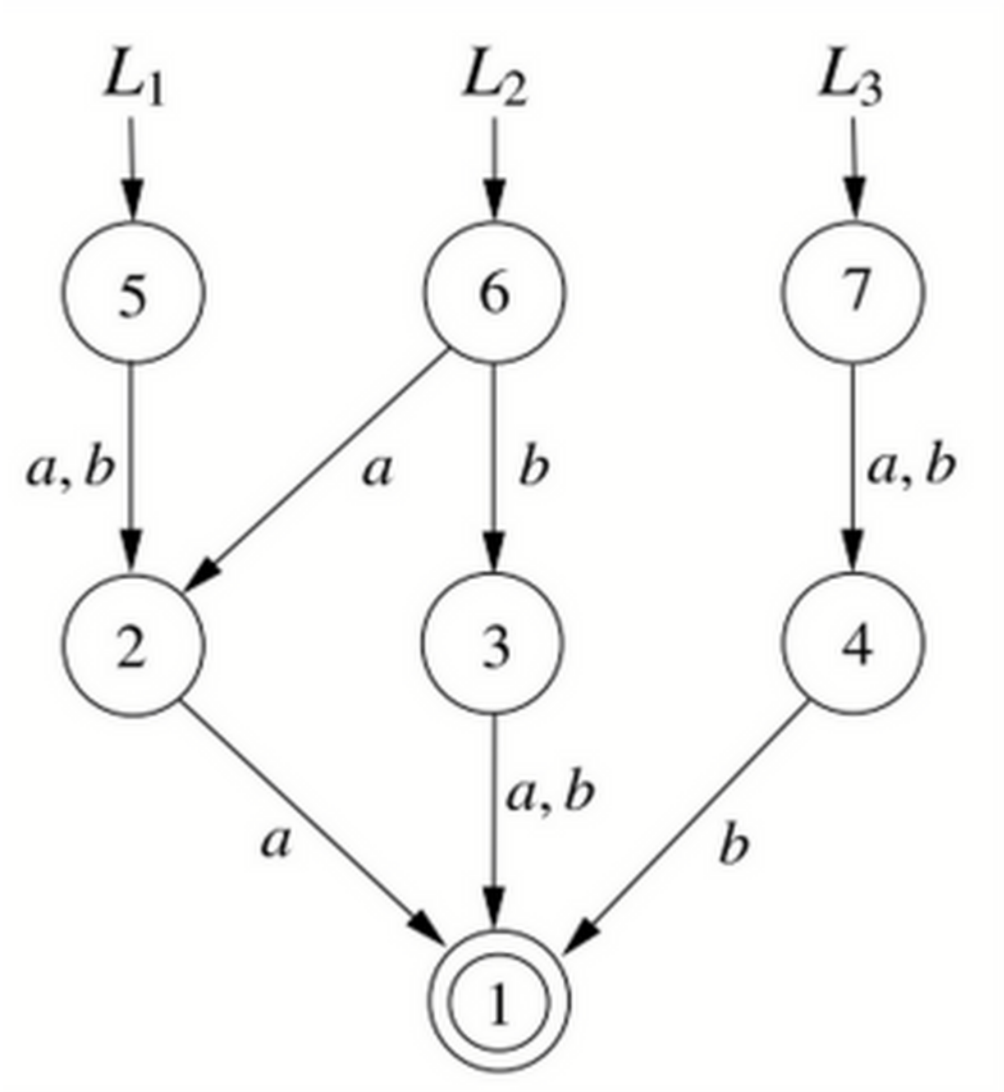
The master z-automaton

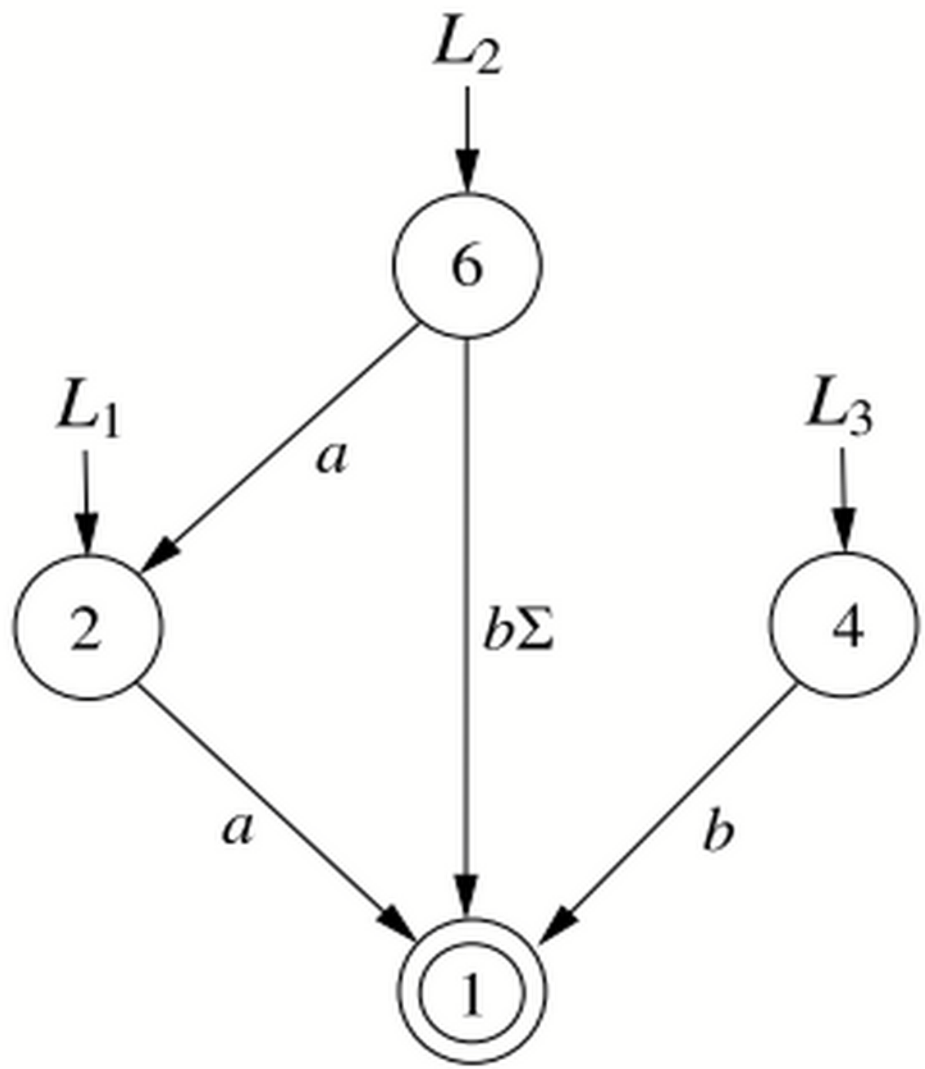




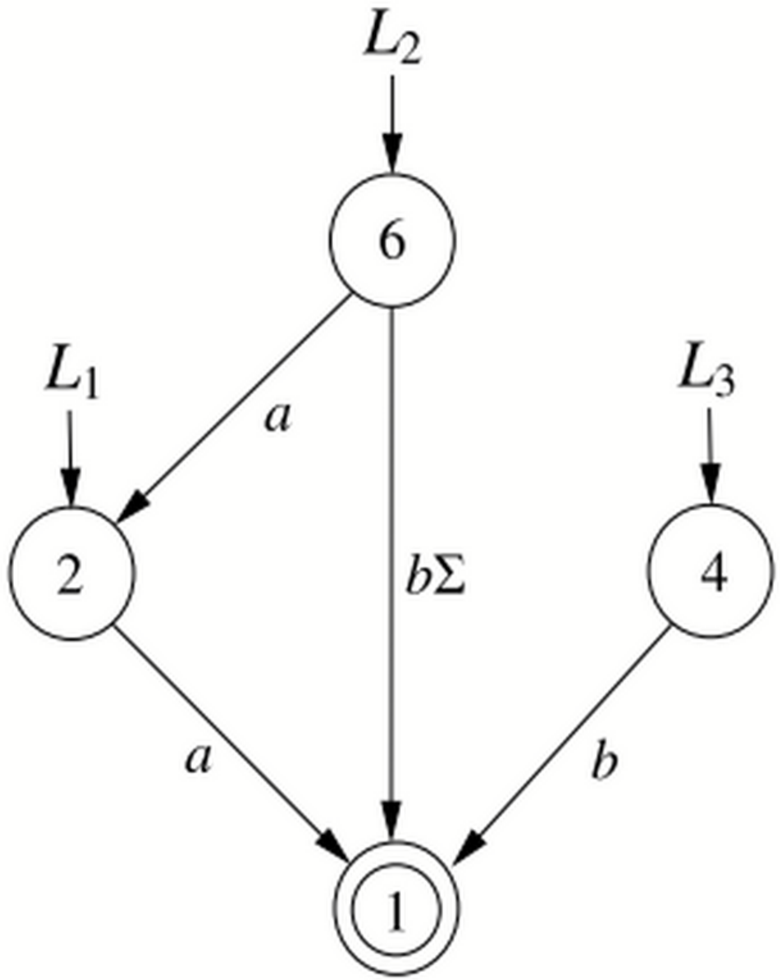
$\{aaa, aab, aba, baa, bab, bba, bbb\}$
$\{aab, abb, baa, bab, bbb\}$
$b\Sigma^2$
a
b
$a\Sigma$
$\{aa, ab, ba\}$
$\{aa, ab, bb\}$
$a\Sigma$
b
$a\Sigma$
b
$\{a\}$
$\{b\}$
b
a
b
a
$\{\epsilon\}$
$\emptyset$
a, b
a, b

Length: 2

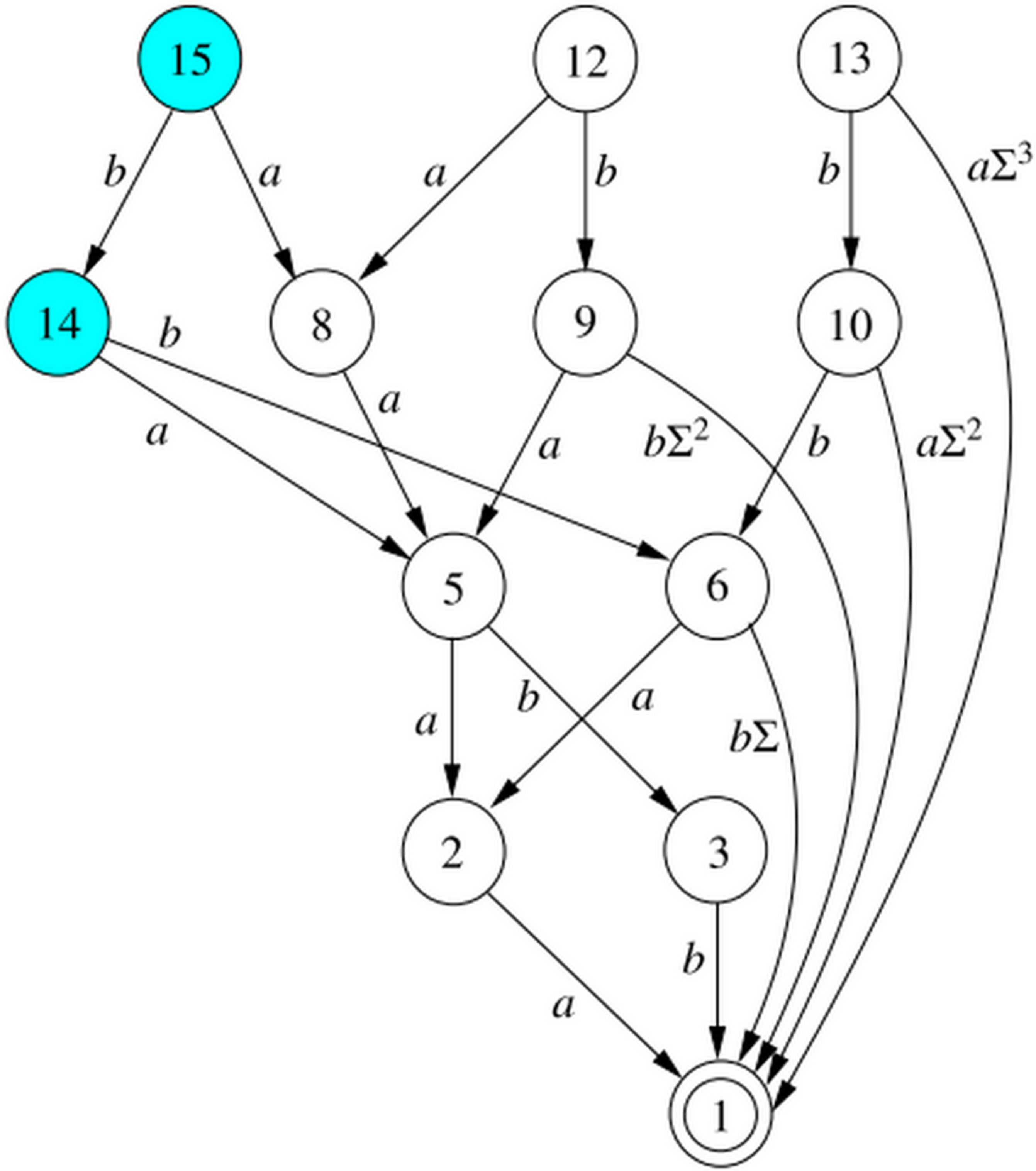




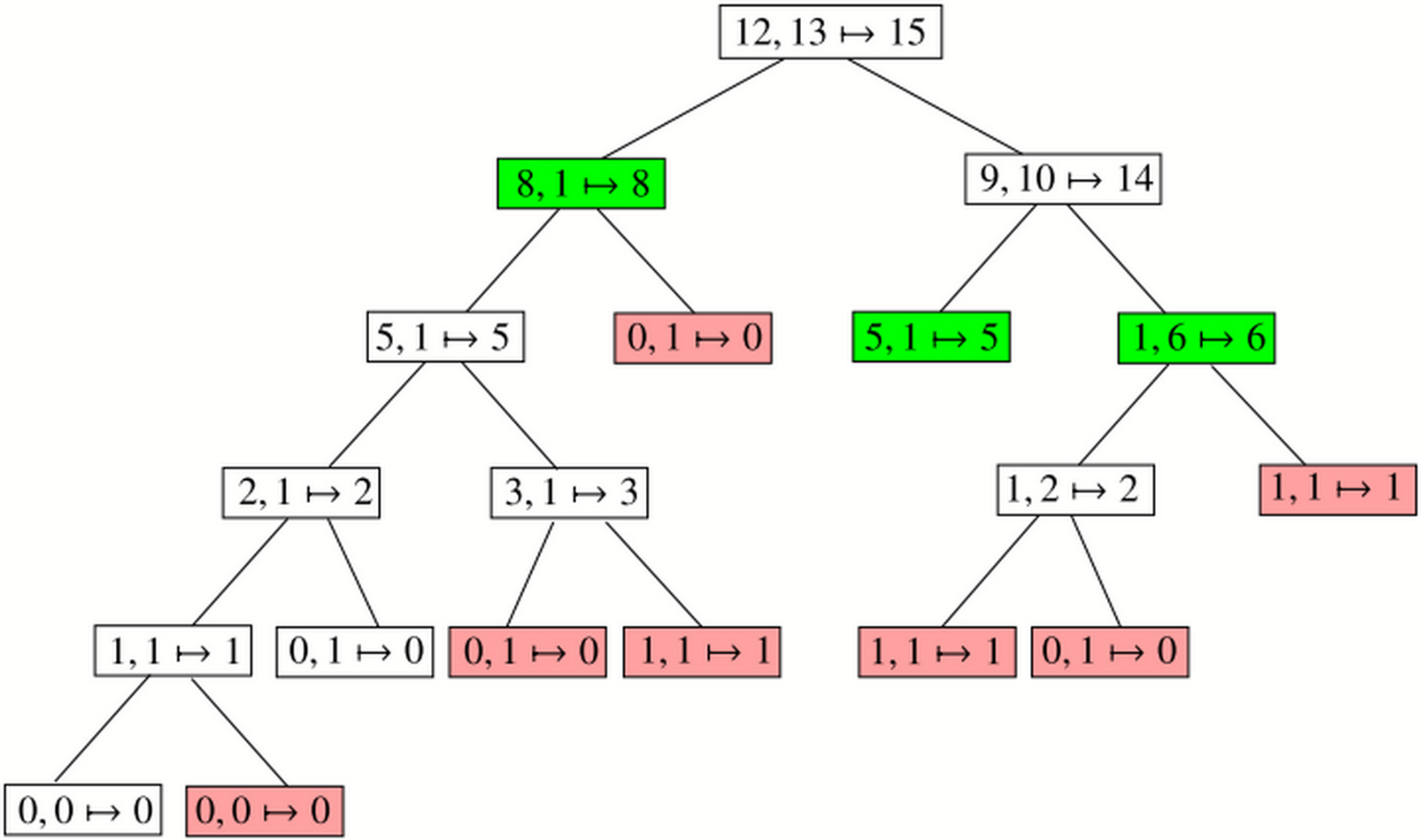
Data structure for z-automata



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



15
12
13
b
a
a
b
b
$a\Sigma^3$
14
8
9
10
b
a
a
$b\Sigma^2$
b
$a\Sigma^2$
5
6
a
b
a
$b\Sigma$
2
3
a
b
1



$12, 13 \mapsto 15$
$8, 1 \mapsto 8$
$9, 10 \mapsto 14$
$5, 1 \mapsto 5$
$0, 1 \mapsto 0$
$5, 1 \mapsto 5$
$1, 6 \mapsto 6$
$2, 1 \mapsto 2$
$3, 1 \mapsto 3$
$1, 2 \mapsto 2$
$1, 1 \mapsto 1$
$1, 1 \mapsto 1$
$0, 1 \mapsto 0$
$0, 1 \mapsto 0$
$1, 1 \mapsto 1$
$1, 1 \mapsto 1$
$0, 1 \mapsto 0$
$0, 0 \mapsto 0$
$0, 0 \mapsto 0$