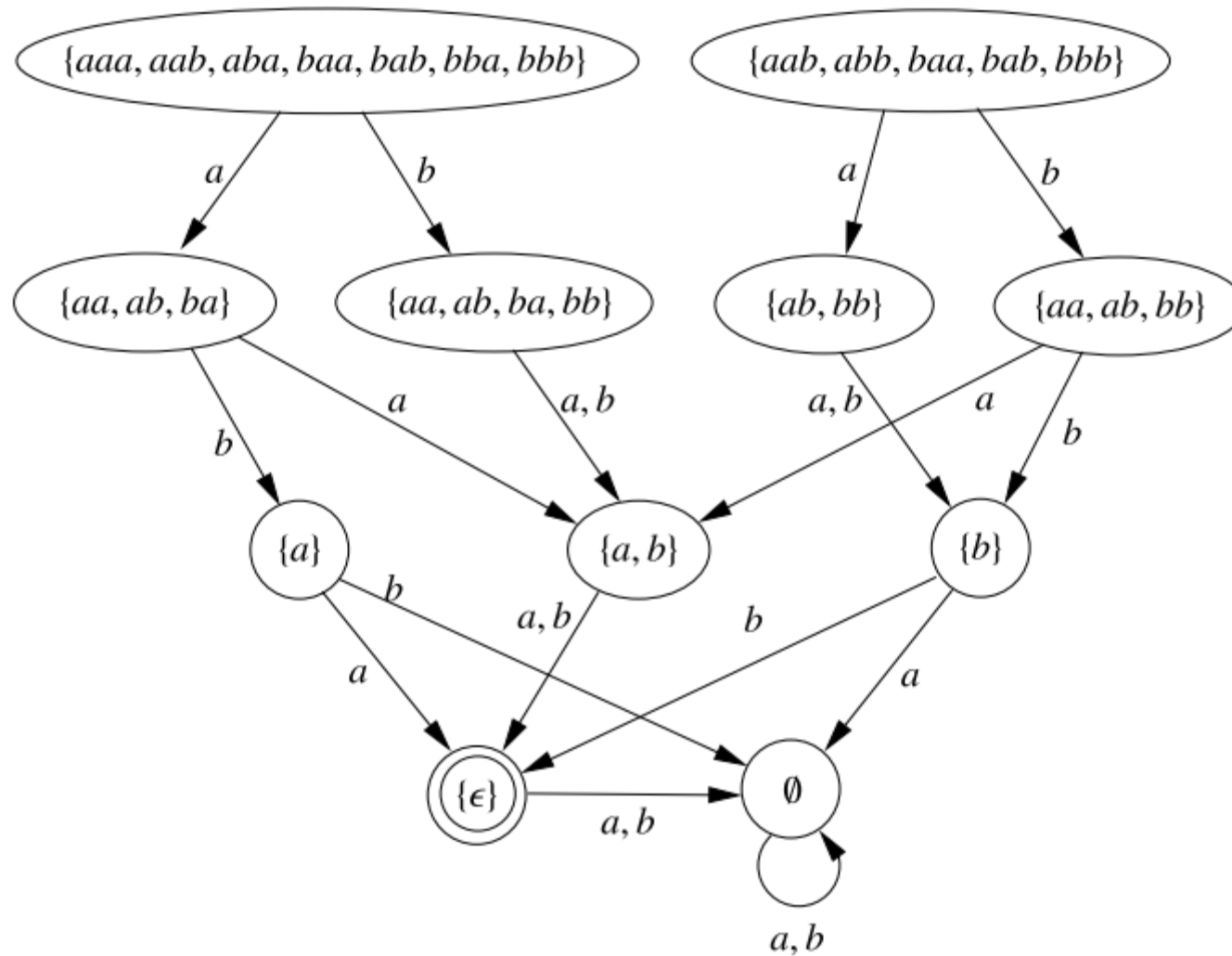


Finite Universes

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

- The **master automaton** over Σ is the tuple $M = (Q_M, \Sigma, \delta_M, F_M)$, where
 - Q_M is the set of all fixed-length languages;
 - $\delta_M: Q_M \times \Sigma \rightarrow Q_M$ is given by $\delta_M(L, a) = L^a$;
 - F_M is the set $\{ \{\varepsilon\} \}$ (the only final state is the language $\{\varepsilon\}$).
- **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.

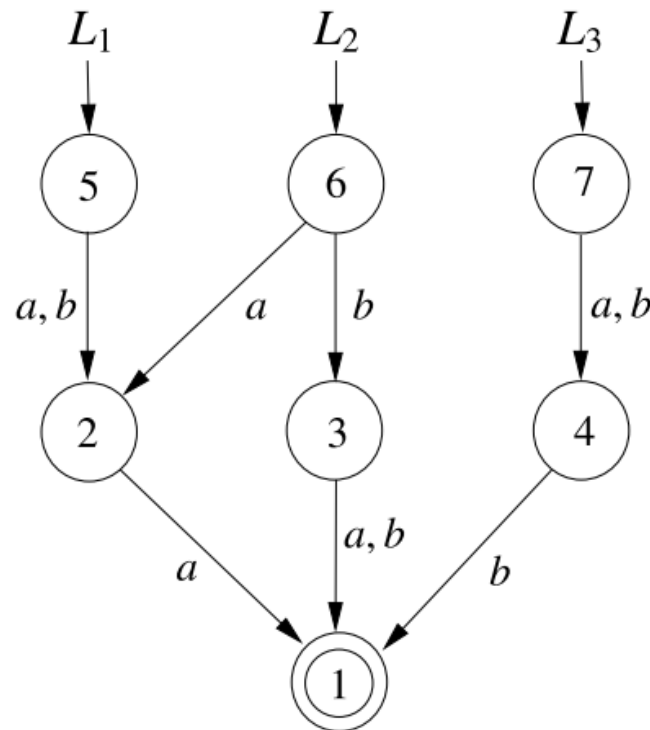
$n > 0$. Then $\delta_M(L, a) = L^a$ for every $a \in \Sigma$, and L^a has smaller length than L . By induction hypothesis the state L^a recognizes the language L^a , and so the state L recognizes the language L .

The master automaton

- 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.

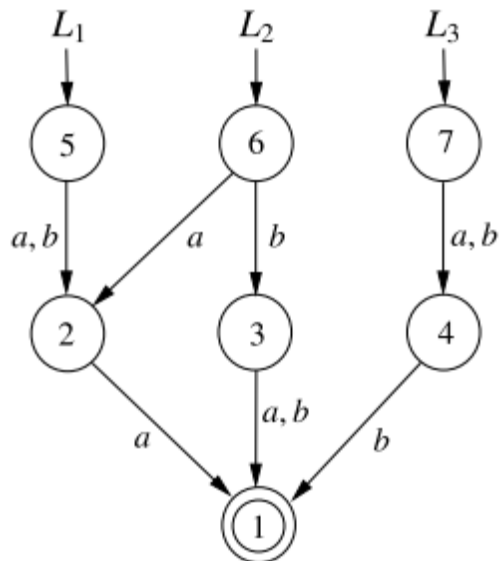
Data structure for fixed-length languages

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



Data structure for fixed-length languages

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



Ident.	<i>a</i> -succ	<i>b</i> -succ
2	1	0
3	1	1
4	0	1
5	2	2
6	2	3
7	4	4

Data structure for fixed-length languages

- 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 of the same length.
 - **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 $L_1 = \emptyset$ or $L_2 = \emptyset$ then $L_1 \cap L_2 = \emptyset$;
 - (2) if $L_1 = \{\varepsilon\} = L_2$ then $L_1 \cap L_2 = \{\varepsilon\}$;
 - (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 $a \in \Sigma$.

Implementing union and intersection

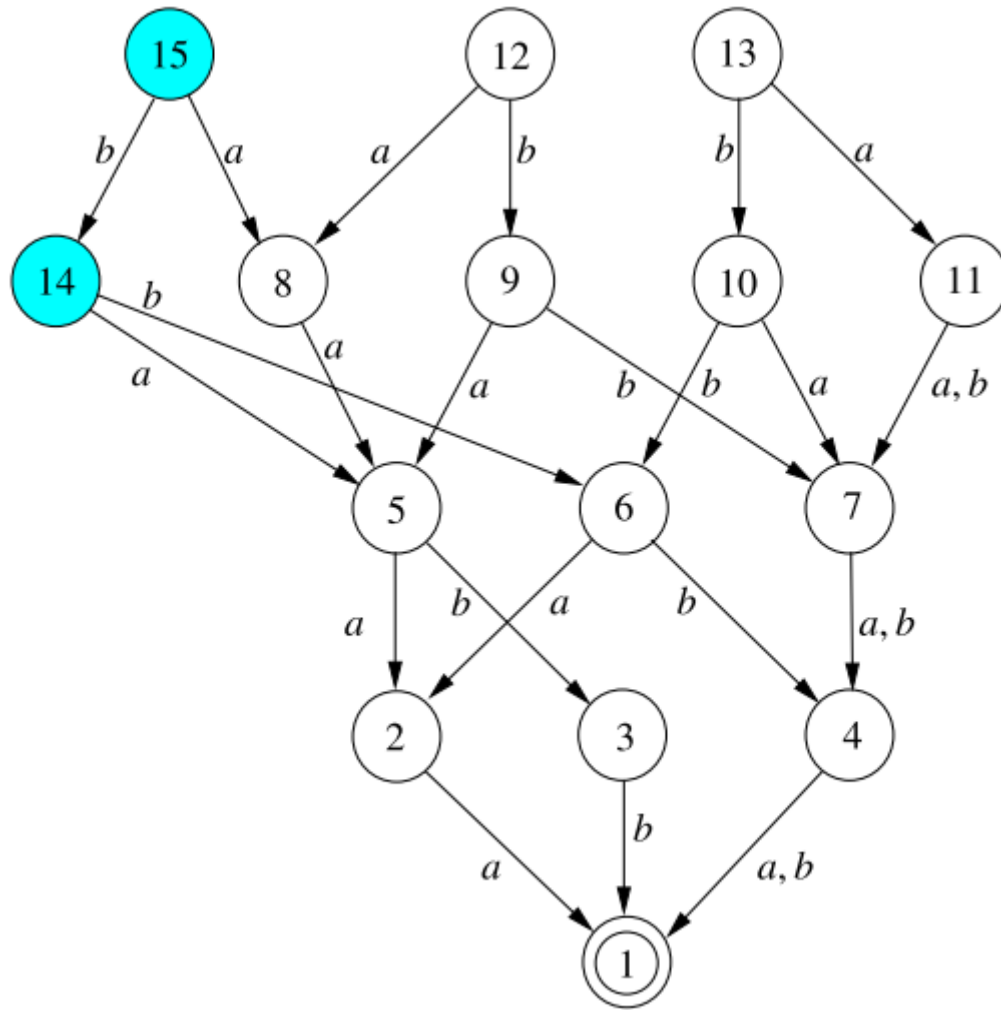
inter(q_1, q_2)

Input: states q_1, q_2 recognizing languages of the same length

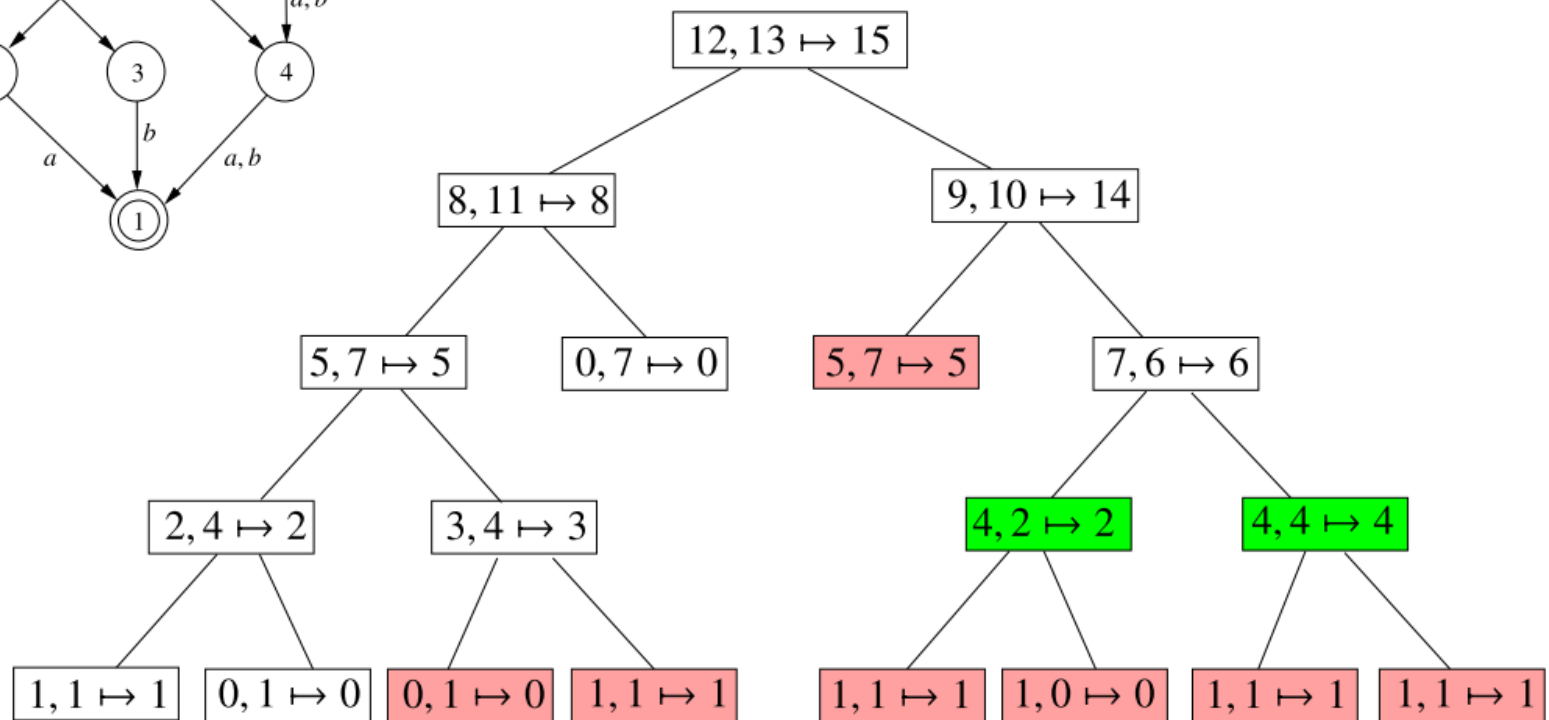
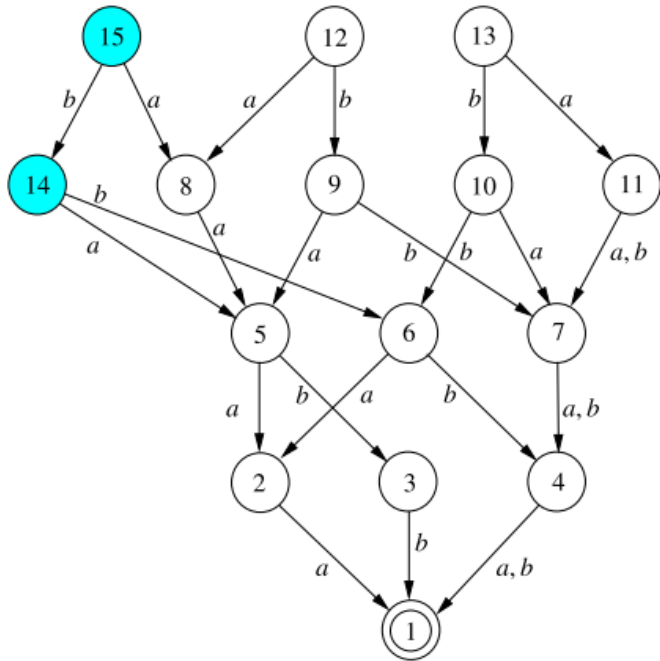
Output: state recognizing $L(q_1) \cap L(q_2)$

- 1 **if** $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
- 2 **if** $q_1 = q_0$ **or** $q_2 = q_0$ **then return** q_0
- 3 **else if** $q_1 = q_\epsilon$ **and** $q_2 = q_\epsilon$ **then return** q_ϵ
- 4 **else** /* $q_1, q_2 \notin \{q_0, q_\epsilon\}$ */
- 5 **for all** $i = 1, \dots, m$ **do** $r_i \leftarrow \text{inter}(q_1^{a_i}, q_2^{a_i})$
- 6 $G(q_1, q_2) \leftarrow \text{make}(r_1, \dots, r_m)$
- 7 **return** $G(q_1, q_2)$

Implementing union and intersection



Implementing union and intersection



Implementing fixed-length complement

- If a set $X \subseteq U$ is encoded by a language L of length n , then the set $U \setminus X$ is encoded by the fixed-length complement $\Sigma^n \setminus L$, denoted by \bar{L}^n . This is **different** from \bar{L} !
- Since the empty language has all lengths, we have $\bar{\emptyset}^n = \Sigma^n$ for every $n \geq 0$, in particular $\bar{\emptyset}^0 = \Sigma^0 = \{\epsilon\}$,
- The algorithm follows immediately from the following properties
 1. If L has length 0 and $L = \emptyset$ then $\bar{L}^0 = \{\epsilon\}$.
 2. If L has length 0 and $L = \{\epsilon\}$ then $\bar{L}^0 = \emptyset$.
 3. If L has length $n \geq 1$, then $(\bar{L}^n)^a = \bar{L}^{a^{n-1}}$.

Implementing fixed-length complement

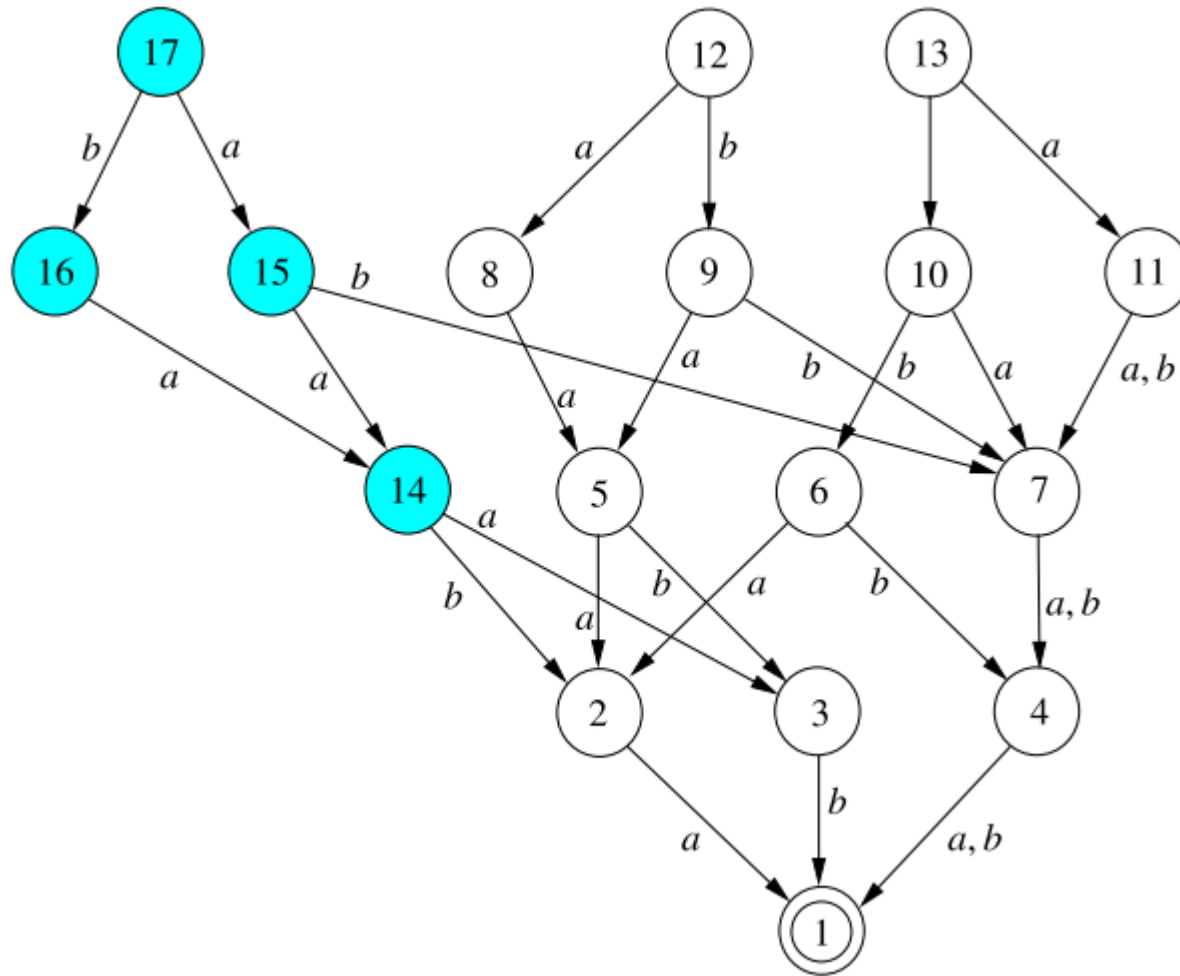
comp(n, q)

Input: length n , state q of length n

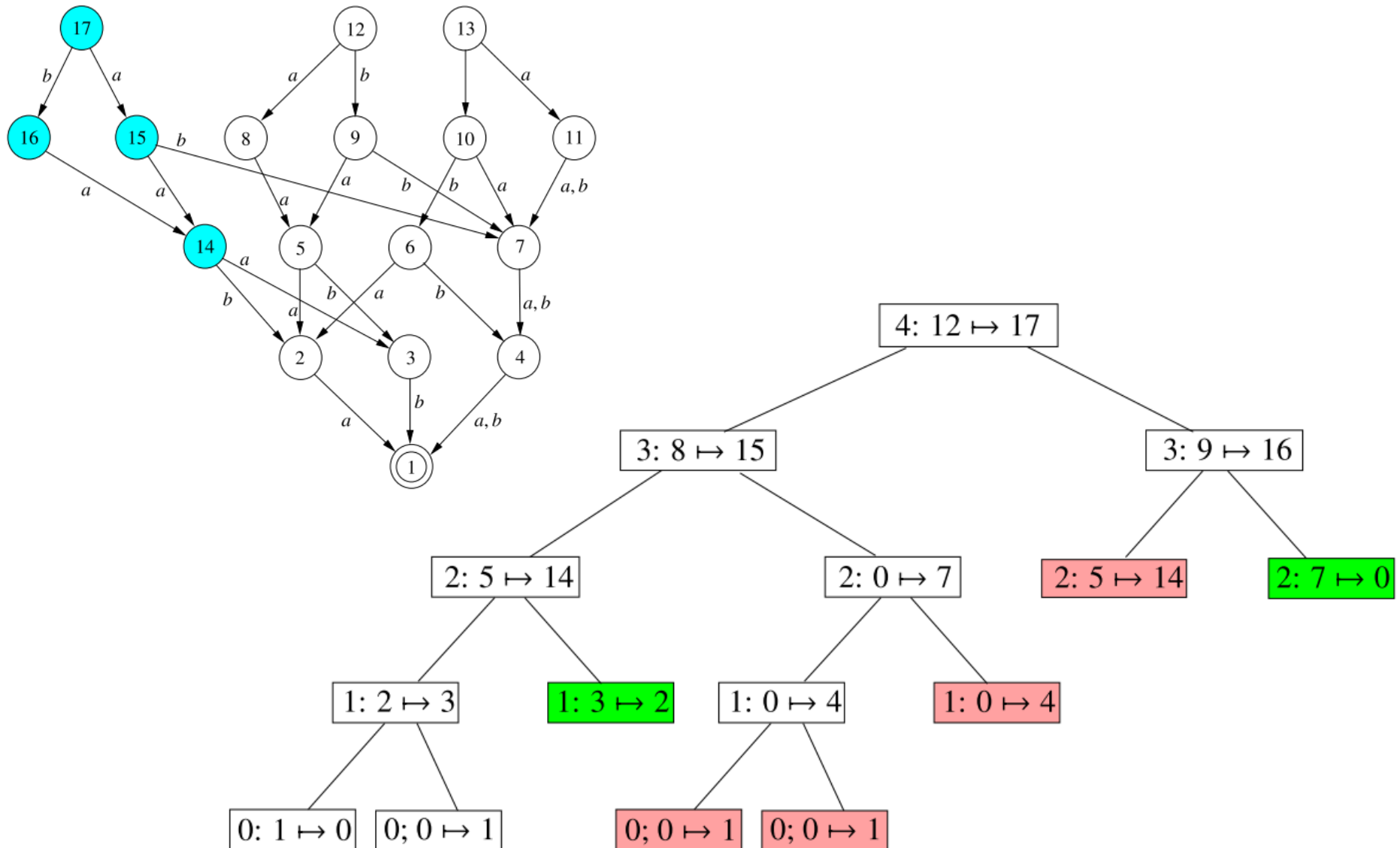
Output: state recognizing $\overline{L(q)}^n$

- 1 **if** $G(n, q)$ is not empty **then return** $G(n, q)$
- 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** $i = 1, \dots, m$ **do** $r_i \leftarrow \text{comp}(n - 1, q^{a_i})$
- 6 $G(n, q) \leftarrow \text{make}(r_1, \dots, r_m)$
- 7 **return** $G(n, q)$

Implementing fixed-length complement



Implementing fixed-length complement



Implementing fixed-length universality

- A language L of length n is **fixed-length universal** if $L = \Sigma^n$.
- The algorithm for universality follows immediately from the following properties
 - (1) If $L = \emptyset$ then L is not universal.
 - (2) If $L = \{\varepsilon\}$ then L is universal.
 - (3) If $\emptyset \neq L \neq \{\varepsilon\}$ then L is universal iff L^a is universal for every $a \in \Sigma$.

Implementing fixed-length universality

univ(q)

Input: state q

Output: **true** if $L(q)$ is fixed-length universal,
false otherwise

- 1 **if** $G(q)$ is not empty **then return** $G(q)$
- 2 **if** $q = q_0$ **then return false**
- 3 **else if** $q = q_\epsilon$ **then return true**
- 4 **else** / * $q \neq q_0$ and $q \neq q_\epsilon$ * /
- 5 $G(q) \leftarrow$ **and**($univ(q^{a_1}), \dots, univ(q^{a_m})$)
- 6 **return** $G(q)$

Implementing fixed-length equality

- If two languages L_1, L_2 of the same length are represented by nodes q_1, q_2 of the same table then we have $L_1 = L_2$ iff $q_1 = q_2$, and so equality can be checked in constant time.
- If the languages are represented by nodes from different tables, then equality amounts to isomorphism of the DFAs rooted at the nodes.

eq2(q_1, q_2)

Input: states q_1, q_2 of different tables

Output: true if $L(q_1) = L(q_2)$, false otherwise

```
1   if  $G(q_1, q_2)$  is not empty then return  $G(q_1, q_2)$ 
2   if  $q_1 = q_{01}$  and  $q_2 = q_{02}$  then  $G(q_1, q_2) \leftarrow$  true
3   else if  $q_1 = q_{01}$  and  $q_2 \neq q_{02}$  then  $G(q_1, q_2) \leftarrow$  false
4   else if  $q_1 \neq q_{01}$  and  $q_2 = q_{02}$  then  $G(q_1, q_2) \leftarrow$  false
5   else /*  $q_1 \neq q_{01}$  and  $q_2 \neq q_{02}$  */
6      $G(q_1, q_2) \leftarrow$  and( $\text{eq}(q_1^{a_1}, q_2^{a_1}), \dots, \text{eq}(q_1^{a_m}, q_2^{a_m})$ )
7   return  $G(q_1, q_2)$ 
```

NFAs as starting point

- **Given:** Acyclic NFA A accepting a fixed-length language.
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 $state(Q_0)$

NFAs as starting point

- The algorithm follows from the following observations:
 - 1) If $S = \emptyset$ then $L(S) = \emptyset$.
 - 2) If $S \cap F \neq \emptyset$ then $L(S) = \{\epsilon\}$.
 - 3) If $S \neq \emptyset$ and $S \cap F \neq \emptyset$ then $L(S) = \bigcup_{\{i=1\}}^n a_i \cdot L(S_i)$,
where $L(S_i) = \delta(S, a_i)$.
- This leads directly to a recursive algorithm:

NFAs as starting point

det&min(A)

Input: NFA $A = (Q, \Sigma, \delta, Q_0, F)$

Output: master state recognizing $L(A)$

1 **return** *state*(Q_0)

state(S)

Input: set $S \subseteq Q$ recognizing languages of the same length

Output: state recognizing $L(S)$

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

2 **else if** $S = \emptyset$ **then return** q_0

3 **else if** $S \cap F \neq \emptyset$ **then return** q_ϵ

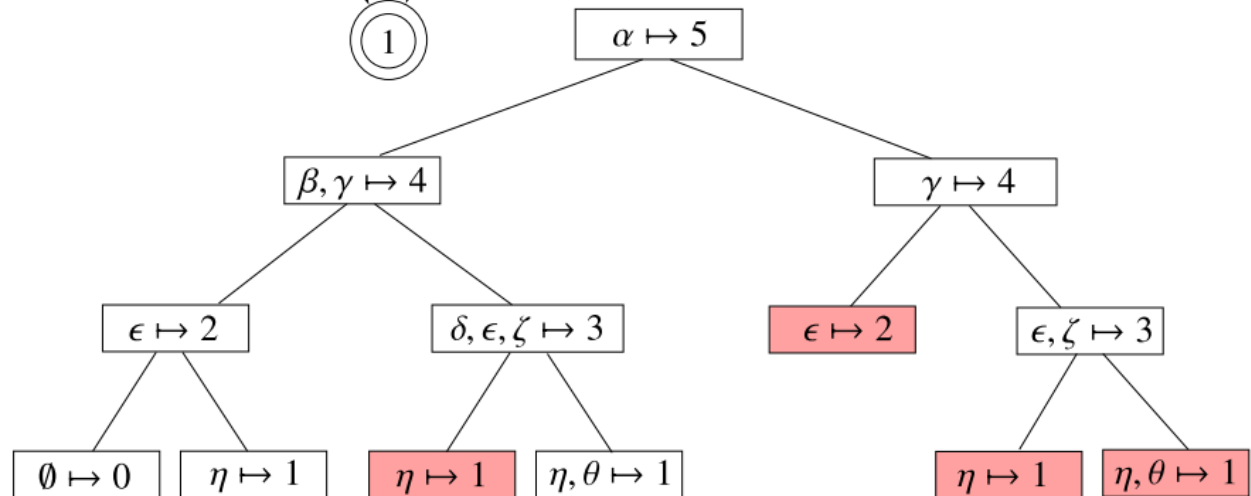
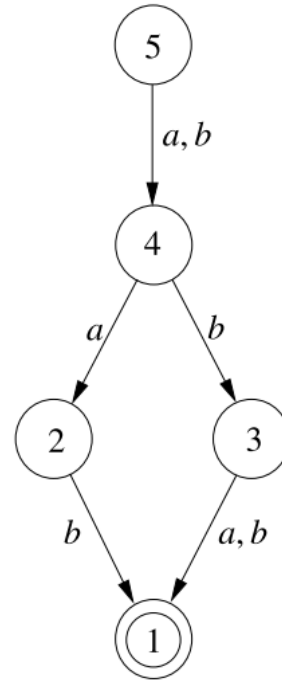
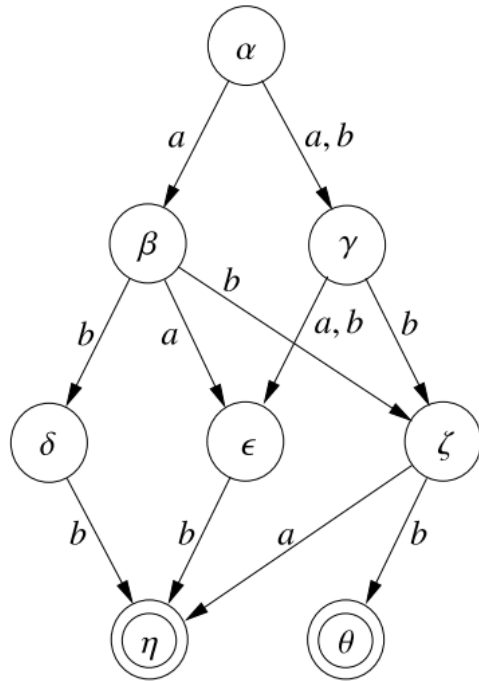
4 **else** /* $S \neq \emptyset$ and $S \cap F = \emptyset$ */

5 **for all** $i = 1, \dots, m$ **do** $S_i \leftarrow \delta(S, a_i)$

6 $G(S) \leftarrow \text{make}(\text{state}(S_1), \dots, \text{state}(S_m));$

7 **return** $G(S)$

NFAs as starting point



Implementing operations on relations

- Assumptions:
 - Objects are encoded as words of Σ^n (one word for each object)
 - Pairs of objects are encoded as words of $(\Sigma \times \Sigma)^n$.
Recall: $\Sigma^n \times \Sigma^n$ and $(\Sigma \times \Sigma)^n$ are isomorphic.
 - Observe: objects and pairs of objects are both encoded as words of length n , but over different alphabets.

- Notation: Given $R \subseteq \Sigma^n \times \Sigma^n$, we denote

$$R^{[a,b]} = \{ (w_1, w_2) \in \Sigma^{n-1} \times \Sigma^{n-1} \mid (aw_1, bw_2) \in R \}.$$

- **Master transducer:** Master automaton over the alphabet $\Sigma \times \Sigma$.

Implementing fixed-length join

- The algorithm follows from:

1) $\emptyset \circ R = R \circ \emptyset = \emptyset$

2) $\{[\epsilon, \epsilon]\} \circ \{[\epsilon, \epsilon]\} = \{[\epsilon, \epsilon]\}$

- 3) If R_1, R_2 have length at least 1, then

$$R_1 \circ R_2 = \bigcup_{a,b,c \in \Sigma} [a, b] \cdot \left(R_1^{[a,c]} \circ R_2^{[c,b]} \right)$$

Implementing fixed-length join

join(r_1, r_2)

Input: states r_1, r_2 of transducer table

Output: state recognizing $L(r_1) \circ L(r_2)$

```
1   if  $G(r_1, r_2)$  is not empty then return  $G(r_1, r_2)$ 
2   if  $r_1 = q_0$  or  $r_2 = q_0$  then return  $q_0$ 
3   else if  $r_1 = q_\epsilon$  and  $r_2 = q_\epsilon$  then return  $q_\epsilon$ 
4   else /*  $q_0 \neq r_1 \neq q_\epsilon$  and  $q_0 \neq r_2 \neq q_\epsilon$  */
5     for all  $(a_i, a_j) \in \Sigma \times \Sigma$  do
6        $r_{i,j} \leftarrow \text{union} \left( \text{join} \left( r_1^{[a_i, a_1]}, r_2^{[a_1, a_j]} \right), \dots, \text{join} \left( r_1^{[a_i, a_m]}, r_2^{[a_m, a_j]} \right) \right)$ 
7      $G(r_1, r_2) = \text{make}(r_{1,1}, \dots, \dots, r_{m,m})$ 
8     return  $G(r_1, r_2)$ 
```

Implementing fixed-length pre and post

- The algorithm for *pre* (*post* is analogous) follows from:

1) If $R = \emptyset$ or $L = \emptyset$ then $pre_{R(L)} = \emptyset$

2) If $R = \{[\epsilon, \epsilon]\}$ and $L = \{\epsilon\}$ then $pre_{R(L)} = \{\epsilon\}$

3) If $\emptyset \neq R \neq \{[\epsilon, \epsilon]\}$ and $\emptyset \neq L \neq \{\epsilon\}$ then

$$pre_R(L) = \bigcup_{a,b \in \Sigma} a \cdot pre_{R[a,b]}(L^b)$$

Proof of 3):

$$\begin{aligned} & aw_1 \in pre_R(L) \\ \Leftrightarrow & \exists bw_2 \in L: [aw_1, bw_2] \in R \\ \Leftrightarrow & \exists b \in \Sigma \exists w_2 \in L^b: [w_1, w_2] \in R^{[a,b]} \\ \Leftrightarrow & \exists b \in \Sigma: w_1 \in pre_{R[a,b]}(L^b) \\ \Leftrightarrow & aw_1 \in \bigcup_{b \in \Sigma} a \cdot pre_{R[a,b]}(L^b) \end{aligned}$$

Implementing fixed-length pre and post

pre(r, q)

Input: state r of transducer table, state q of automaton table

Output: state recognizing $pre_{L(r)}(L(q))$

```
1   if  $G(r, q)$  is not empty then return  $G(r, q)$ 
2   if  $r = r_\emptyset$  or  $q = q_\emptyset$  then return  $q_\emptyset$ 
3   else if  $r = r_\epsilon$  and  $q = q_\epsilon$  then return  $q_\epsilon$ 
4   else
5     for all  $a_i \in \Sigma$  do
6        $q'_i \leftarrow \text{union} \left( \text{pre} \left( r^{[a_i, a_1]}, q^{a_1} \right), \dots, \text{pre} \left( r^{[a_i, a_m]}, q^{a_m} \right) \right)$ 
7        $G(q, r) \leftarrow \text{make}(q'_1, \dots, q'_m)$ 
8   return  $G(q, r)$ 
```


Implementing projection

- We reduce projection to *pre*.
- The projection of a language $R \subseteq \Sigma^n \times \Sigma^n$ onto the first component is the language $pre_R(\Sigma^n)$.
- Specializing the algorithm for *pre* we obtain:

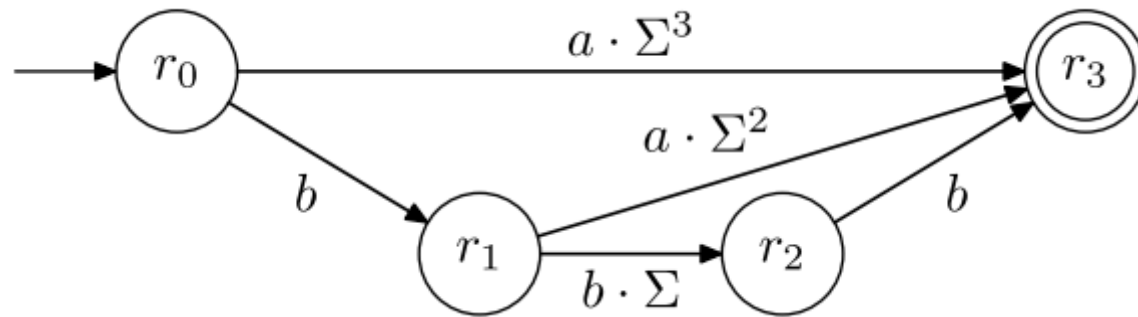
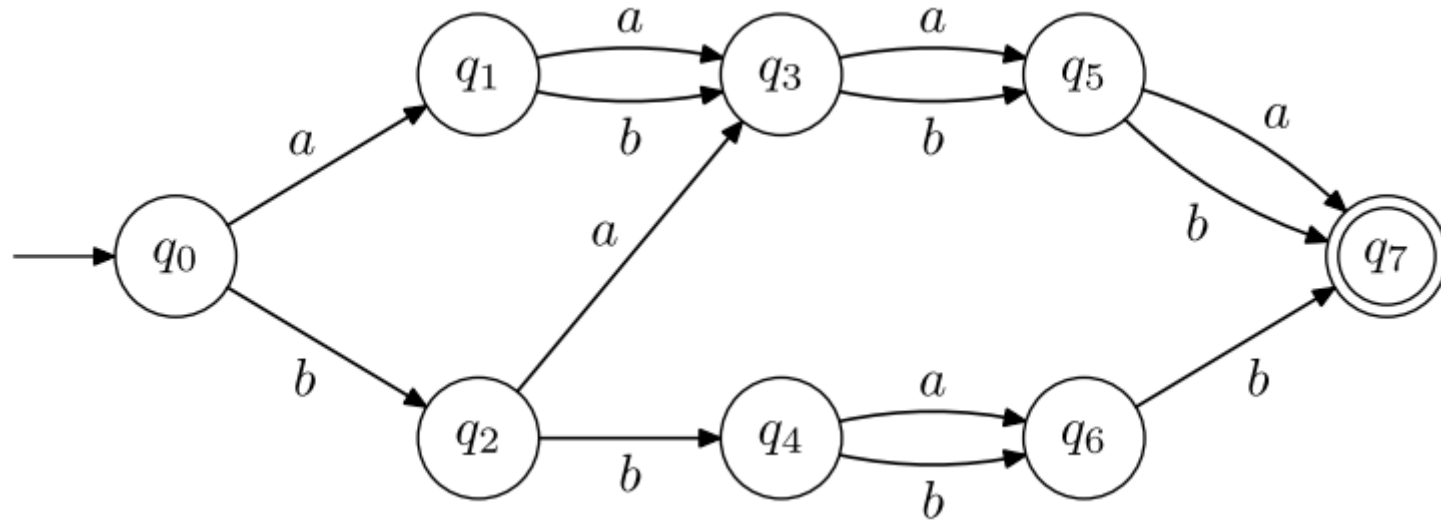
*pro*₁(*r*)

Input: state *r* of transducer table

Output: state recognizing *proj*₁(*L*(*r*))

```
1  if G(r) is not empty then return G(r)
2  if r = r0 then return q0
3  else if r = rε then return qε
4  else
5    for all ai ∈ Σ do
6      q'i ← union(pro1(r[ai,a1]), ..., pro1(r[ai,am]))
7      G(r) ← make(q'1, ..., q'm)
8  return G(r)
```

Decision Diagrams (DDs)

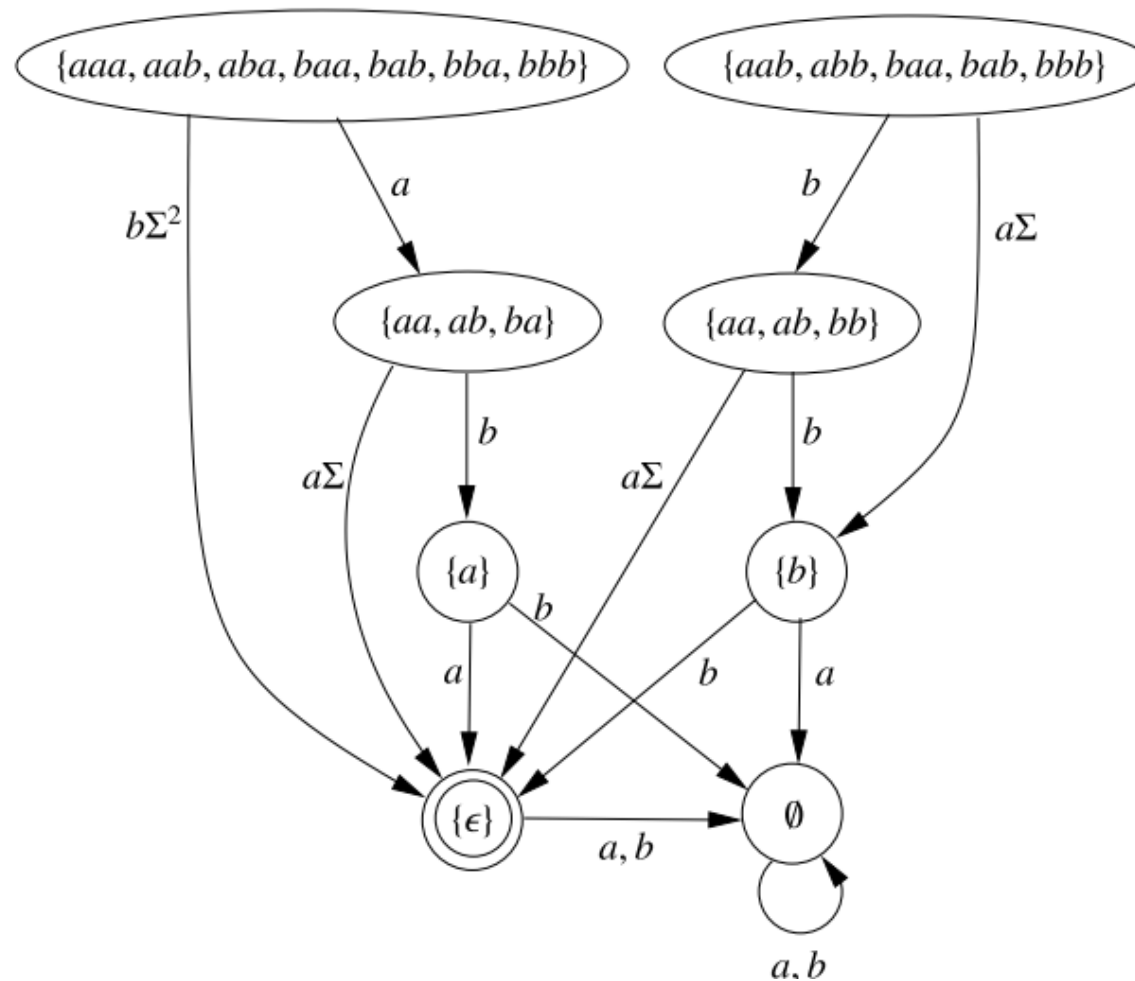


Decision Diagrams (DDs)

- A **decision diagram** is an automaton
 - whose transitions are labeled by regular expressions of the form $a\Sigma^n$, $n \geq 0$, and
 - satisfies the following **determinacy condition** for every state q and letter a : there is exactly one $k \geq 0$ such that $\delta(q, a\Sigma^k) \neq \emptyset$, and for this k there is a state q' such that $\delta(q, a\Sigma^k) = \{q'\}$.
- Observe: Every DFA is a DD.
- A fixed-length language L is a **kernel** if $L = \emptyset$, $L = \{\epsilon\}$, or there are $a, b \in \Sigma$ such that $L^a \neq L^b$.
- The kernel $\langle L \rangle$ of a fixed-length language L is the unique kernel satisfying $L = \Sigma^k \langle L \rangle$ for some $k \geq 0$. Observe: k and $\langle L \rangle$ uniquely determine L for every $L \neq \emptyset$.

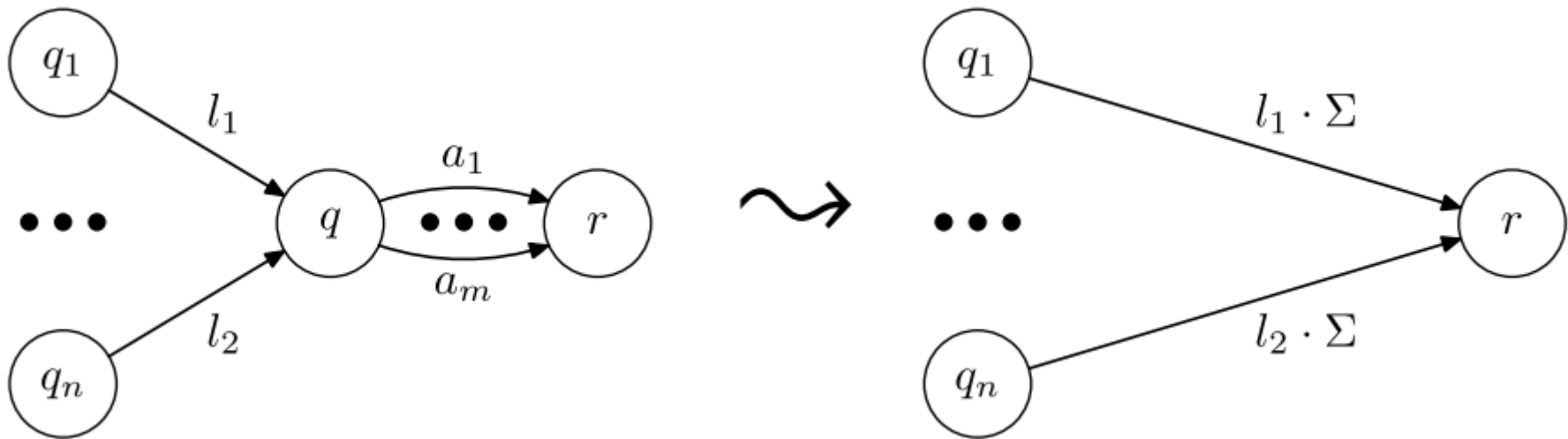
The master decision diagram

- All kernels as states, $\{\epsilon\}$ as final state, transitions $(K, a\Sigma^k, \langle K^a \rangle)$



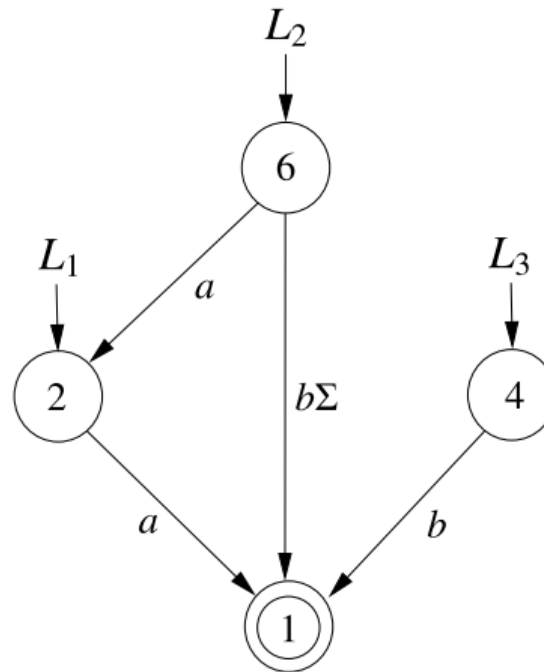
Reduction rule

- **Proposition:** The unique minimal DD for a kernel is the fragment of the master DD rooted at the kernel (modulo labels of transitions leaving the states \emptyset and $\{\epsilon\}$).
- **Proposition:** The minimal DD for a kernel is obtained from its minimal DFA by exhaustively applying the following „reduction rule“:



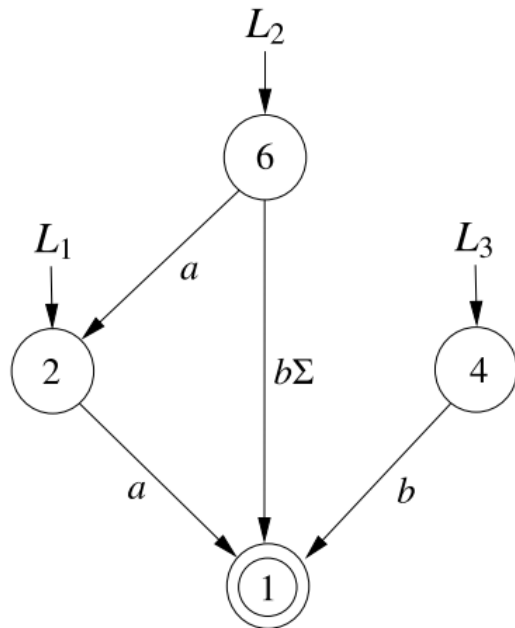
Data structure for kernels

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



Data structure for kernels

- We represent multi-DDs as **tables of kernodes**.
- A **kernode** is a triple $\langle q, l, s \rangle$ where
 - q is a **state identifier**,
 - l is a **length**, and
 - $s = (q_1, \dots, q_m)$ is a **successor tuple**.
- The table for a multi-DD contains a node for each state **but** the states for \emptyset and ϵ .



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

Implementing intersection

- Given kernels K_1, K_2 of languages L_1, L_2 , we wish to compute $K_1 \sqcap K_2 = \langle L_1 \cap L_2 \rangle$.

- We have

1. If $K_1 = \emptyset$ or $K_2 = \emptyset$ then $K_1 \sqcap K_2 = \emptyset$.

2. If $K_1 \neq \emptyset \neq K_2$ then

$$K_1 \sqcap K_2 = \begin{cases} \langle \Sigma^{l_2-l_1} K_1 \cap K_2 \rangle & \text{if } l_1 < l_2 \\ \langle K_1 \cap \Sigma^{l_1-l_2} K_2 \rangle & \text{if } l_2 < l_1 \\ \langle K_1 \cap K_2 \rangle & \text{if } l_1 = l_2 \end{cases}$$

3. If $l_1 < l_2$ then $\langle (\Sigma^{l_2-l_1} K_1 \cap K_2)^a \rangle = K_1 \sqcap \langle K_2^a \rangle$

4. If $l_2 < l_1$ then $\langle (K_1 \cap \Sigma^{l_1-l_2} K_2)^a \rangle = \langle K_1^a \rangle \sqcap K_2$

5. If $l_1 = l_2$ then $\langle (K_1 \cap K_2)^a \rangle = \langle K_1^a \rangle \sqcap \langle K_2^a \rangle$

- 3.-5. lead to a recursive algorithm

Implementing intersection

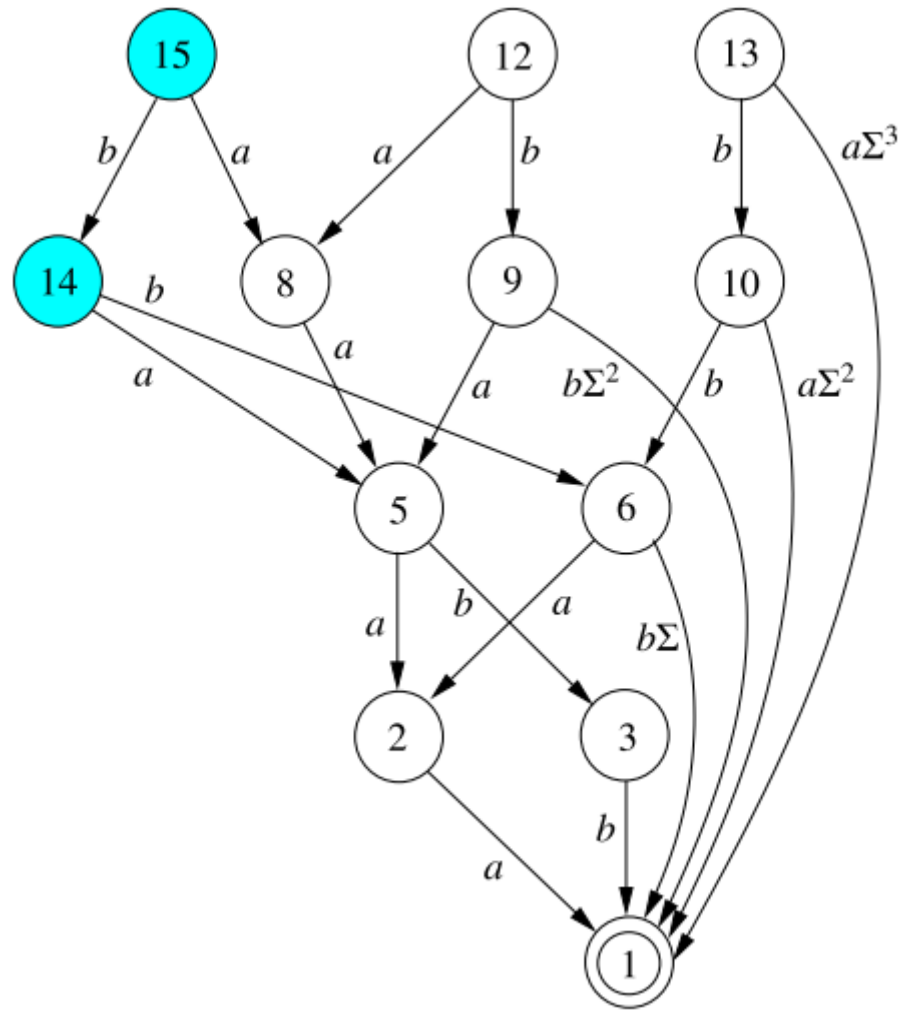
kinter(q_1, q_2)

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  $G(q_1, q_2)$  is not empty then return  $G(q_1, q_2)$ 
2  if  $q_1 = q_0$  or  $q_2 = q_0$  then return  $q_0$ 
3  if  $q_1 \neq q_0$  and  $q_2 \neq q_0$  then
4    if  $l_1 < l_2$  /* lengths of the kernodes for  $q_1, q_2$  */ then
5      for all  $i = 1, \dots, m$  do  $r_i \leftarrow kinter(q_1, q_2^{a_i})$ 
6       $G(q_1, q_2) \leftarrow kmake(l_2, r_1, \dots, r_m)$ 
7    else if  $l_1 = l_2$  then
8      for all  $i = 1, \dots, m$  do  $r_i \leftarrow kinter(q_1^{a_i}, q_2)$ 
9       $G(q_1, q_2) \leftarrow kmake(l_1, r_1, \dots, r_m)$ 
10   else /*  $l_1 > l_2$  */
11     for all  $i = 1, \dots, m$  do  $r_i \leftarrow kinter(q_1^{a_i}, q_2^{a_i})$ 
12      $G(q_1, q_2) \leftarrow kmake(l_1, r_1, \dots, r_m)$ 
13  return  $G(q_1, q_2)$ 
```

Implementing intersection



Implementing intersection

