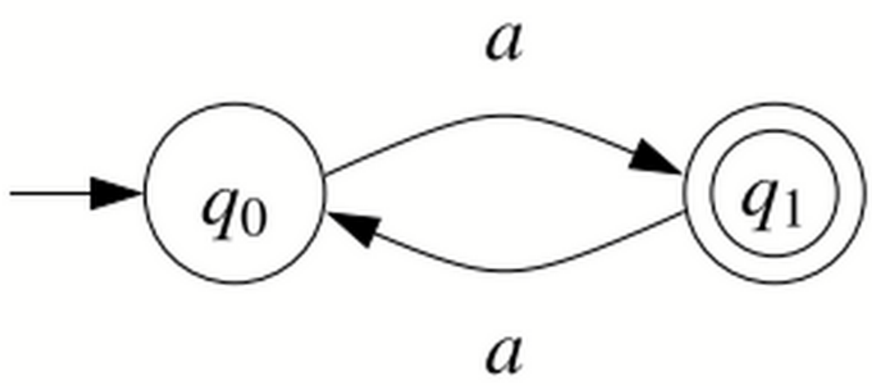
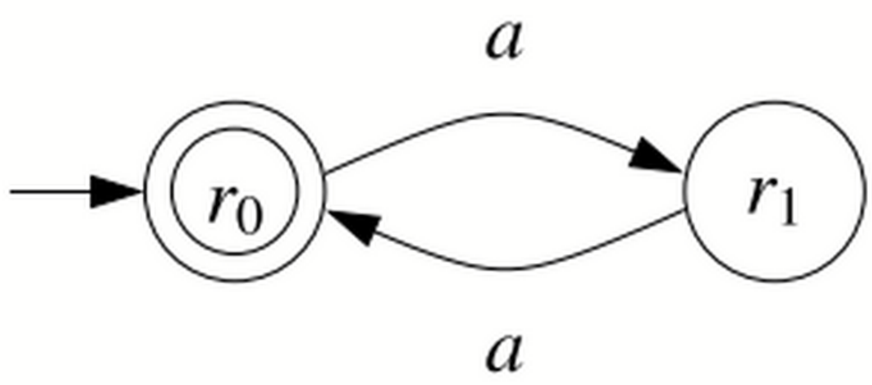
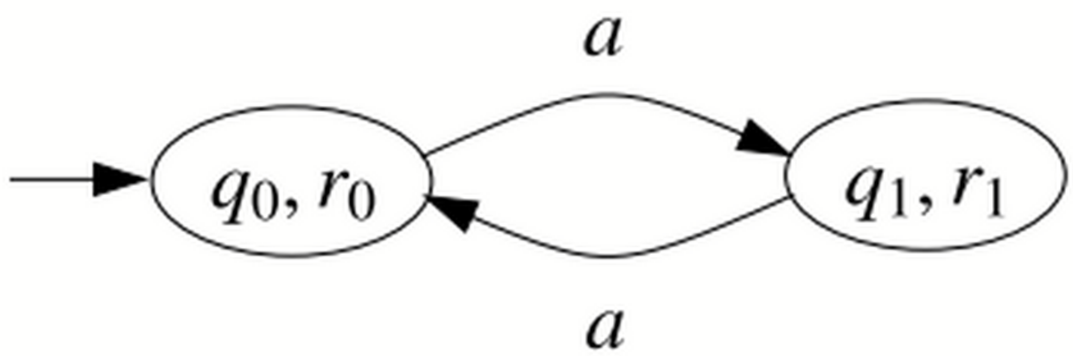
Implementing boolean operations

Intersection

The algorithm for NFAs does not work ...



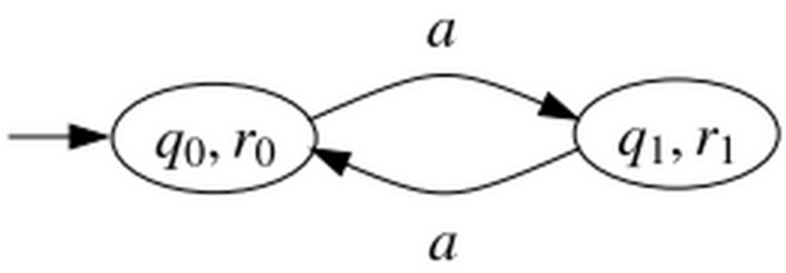


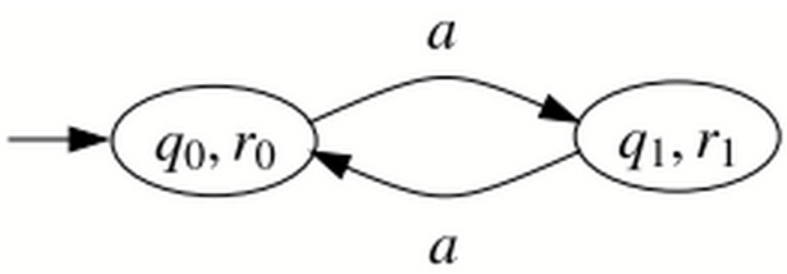


Solution

Apply the same idea as in the conversion NGA => NBA

1. Take two copies of the pairing A1 X A2
2.
3.
4.

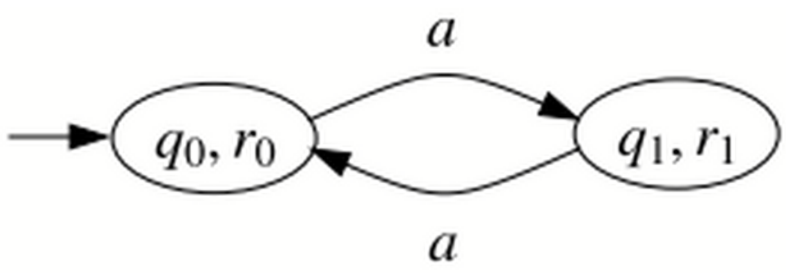


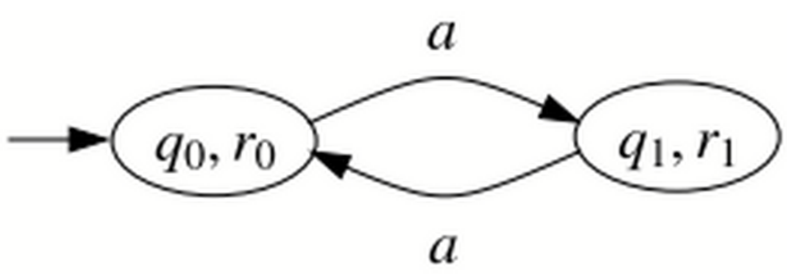


Solution

Apply the same idea as in the conversion NGA => NBA

1. Take two copies of the pairing A1 X A2
2. Redirect the arcs leaving F1 in the first copy to the second copy
3.
4.

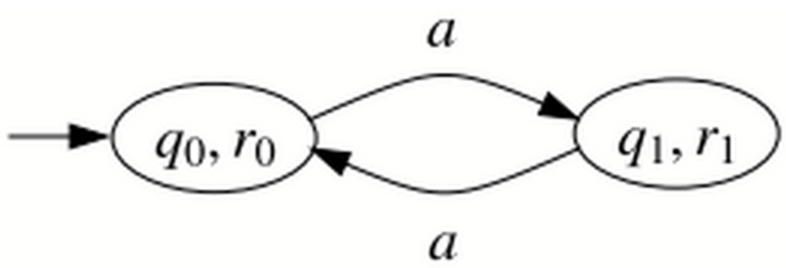


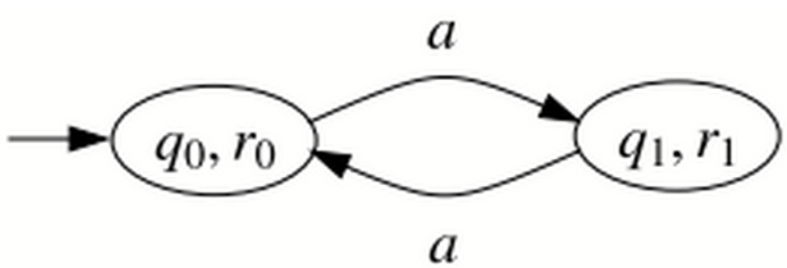


Solution

Apply the same idea as in the conversion NGA => NBA

1. Take two copies of the pairing A1 X A2
2. Redirect the arcs leaving F1 in the first copy to the second copy
3. Redirect the arcs leaving F2 in the second copy to the first copy
4.

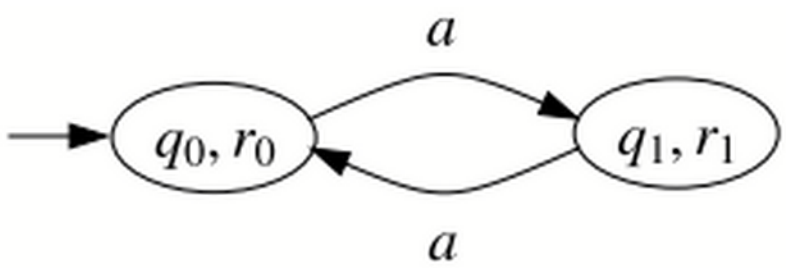


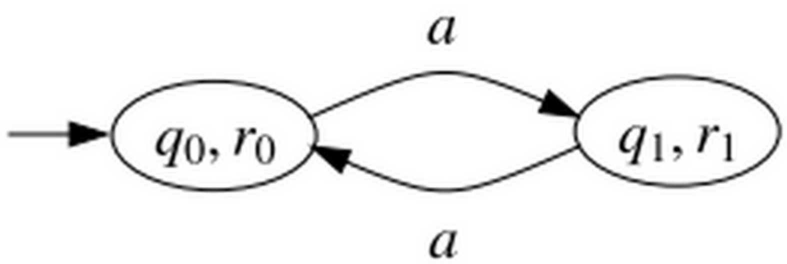


Solution

Apply the same idea as in the conversion NGA => NBA

1. Take two copies of the pairing A1 X A2
2. Redirect the arcs leaving F1 in the first copy to the second copy
3. Redirect the arcs leaving F2 in the second copy to the first copy
4. The final states are the F1-states of the first copy





IntersNBA(A_1, A_2)

Input: NBAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: NBA $A_1 \cap A_2 = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}_\omega(A_1 \cap^\omega A_2) = \mathcal{L}_\omega(A_1) \cap \mathcal{L}_\omega(A_2)$

```
Q, δ, F ← ∅
q0 ← [q01, q02, 1]
W ← { [q01, q02, 1] }
while W ≠ ∅ do
   pick [q1.q2, i] from W
   add [q1, q2, i] to Q′
   if q1 ∈ F1 and i = 1  then add  [q1, q2, 1]
      for all a ∈ Σ do
         for all q′1 ∈ δ1(q1, a), q′2 ∈ δ(q2, a) do
            if i = 1 and q′1 ∉ F1 then
               add ([q1, q2, 1], a, [q′1, q′2, 1]) to δ
               if [q′1, q′2, 1] ∉ Q′ then add  [q′1, q′2, 1]  to W
            if i = 1 and q′1 ∈ F1 then
               add ([q1, q2, 1], a, [q′1, q′2, 2]) to δ
               if [q′1, q′2, 2] ∉ Q′ then add  [q′1, q′2, 2]  to W
            if i = 2 and q′2 ∉ F2 then
               add ([q1, q2, 2], a, [q′1, q′2, 2]) to δ
               if [q′1, q′2, 2] ∉ Q′ then add  [q′1, q′2, 2]  to W
            if i = 2 and q′2 ∈ F2 then
               add ([q1, q2, 2], a, [q′1, q′2, 1]) to δ
               if [q′1, q′2, 1] ∉ Q′ then add  [q′1, q′2, 1]  to W
return (Q, Σ, δ, q0, F)
```

Special cases / Improvements

- All states of at least one of A1 and A2 are accepting

  In this case we do not need the second copy of A1 X A2. The algorithm for NFAs works.

- Intersection of NBAs A1, A2, ..., Ak:

  Do NOT apply the algorithm for two NBAs (k-1) times. Proceed instead as in NGA => NBA (k n1 ... nk states instead of exponential number in k)

Complement

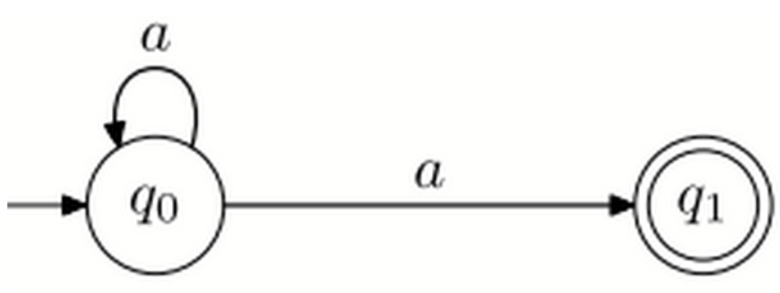
Main result proved by Büchi: NBAs are closed under complement.

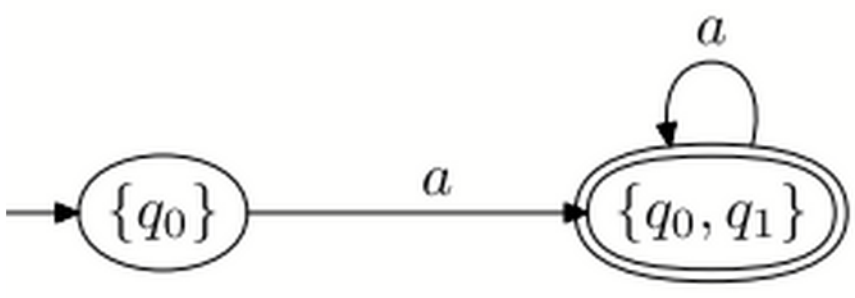
Many later improvements in recent years.

Construction radically different from the one for NFAs.

Problems

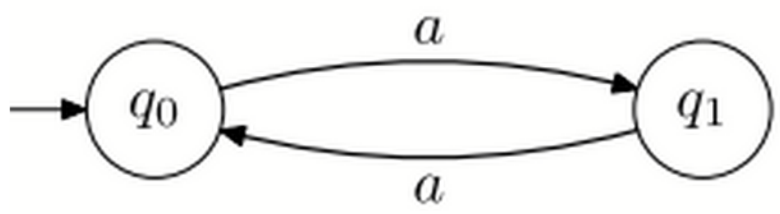
- The powerset construction does not work

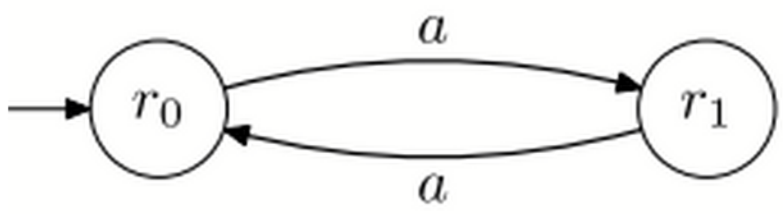




- No other determinization construction works

- Exchanging accepting and non-accepting states in DBAs also fails





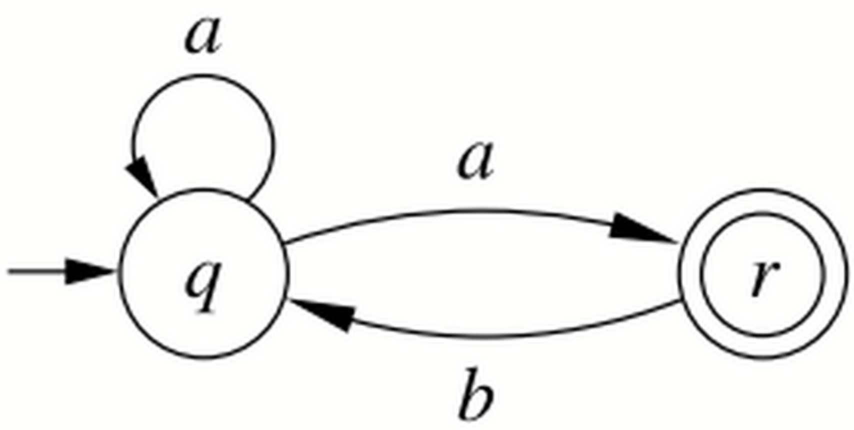
First informal ideas

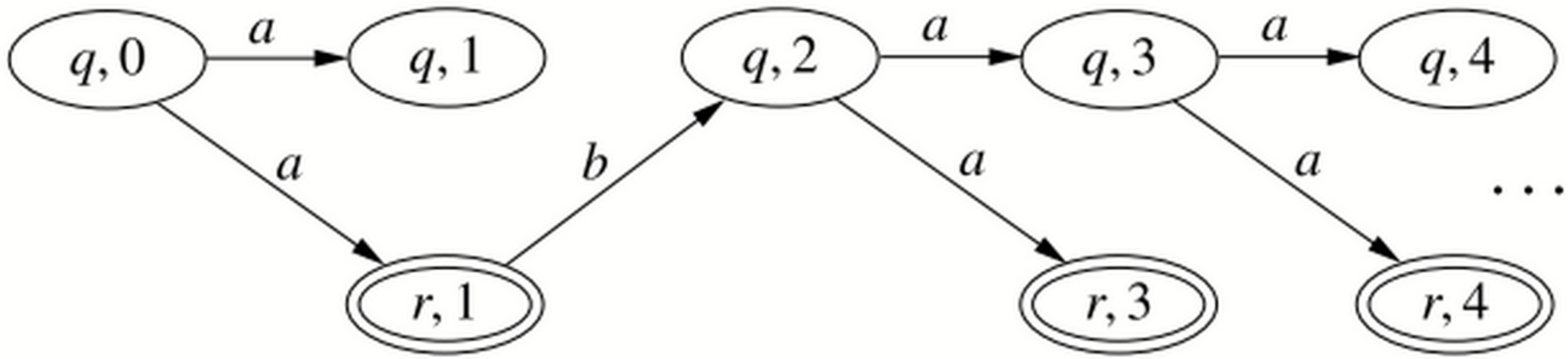
Let NBA A with n states. We search for a complement automaton C(A).

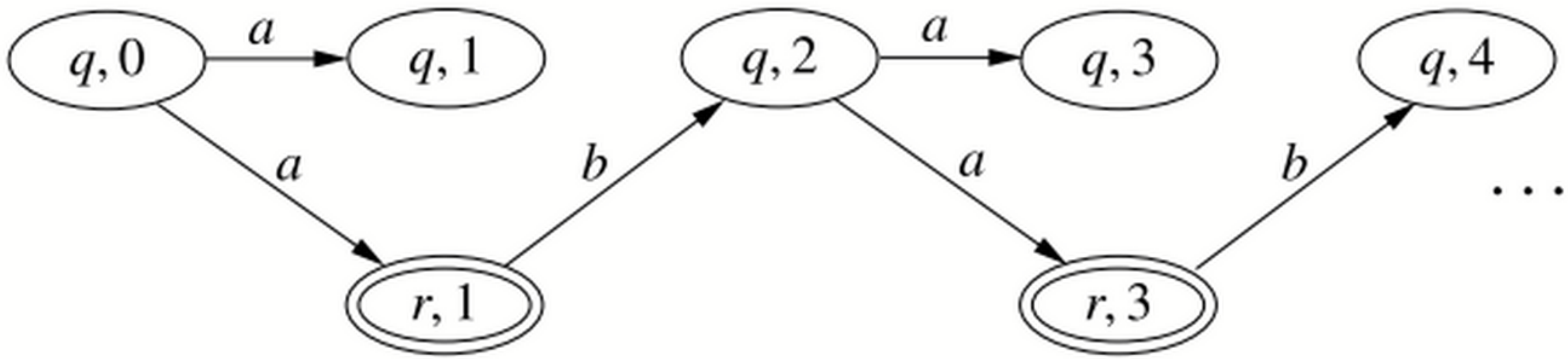
If A does not accept a word w, then no run of A on w is accepting. C(A) must "get this information" from at least one one of its runs on w , so that it can accept.

We first have a closer look at why the powerset construction does not work.

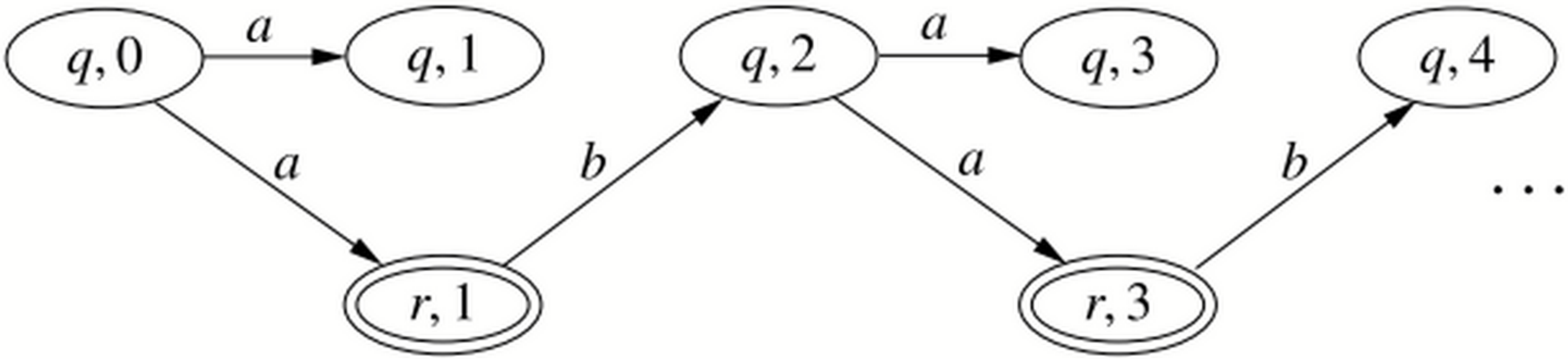
Dag of A on a word w

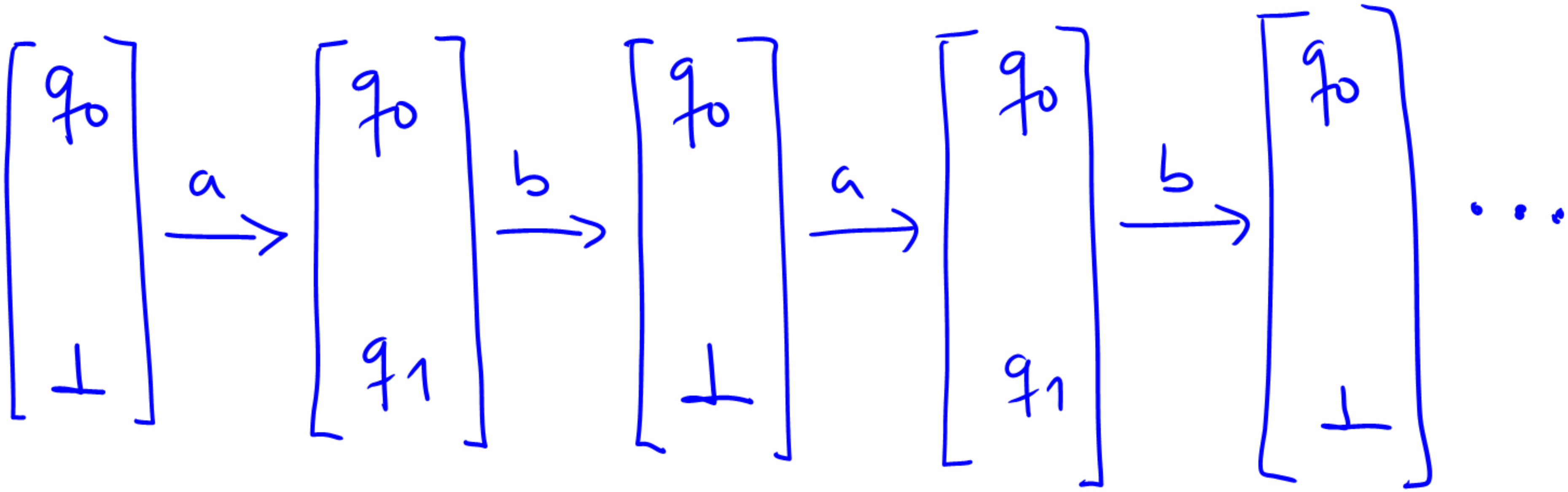


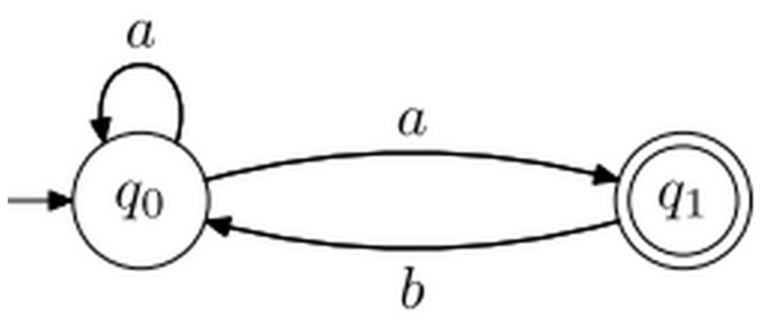


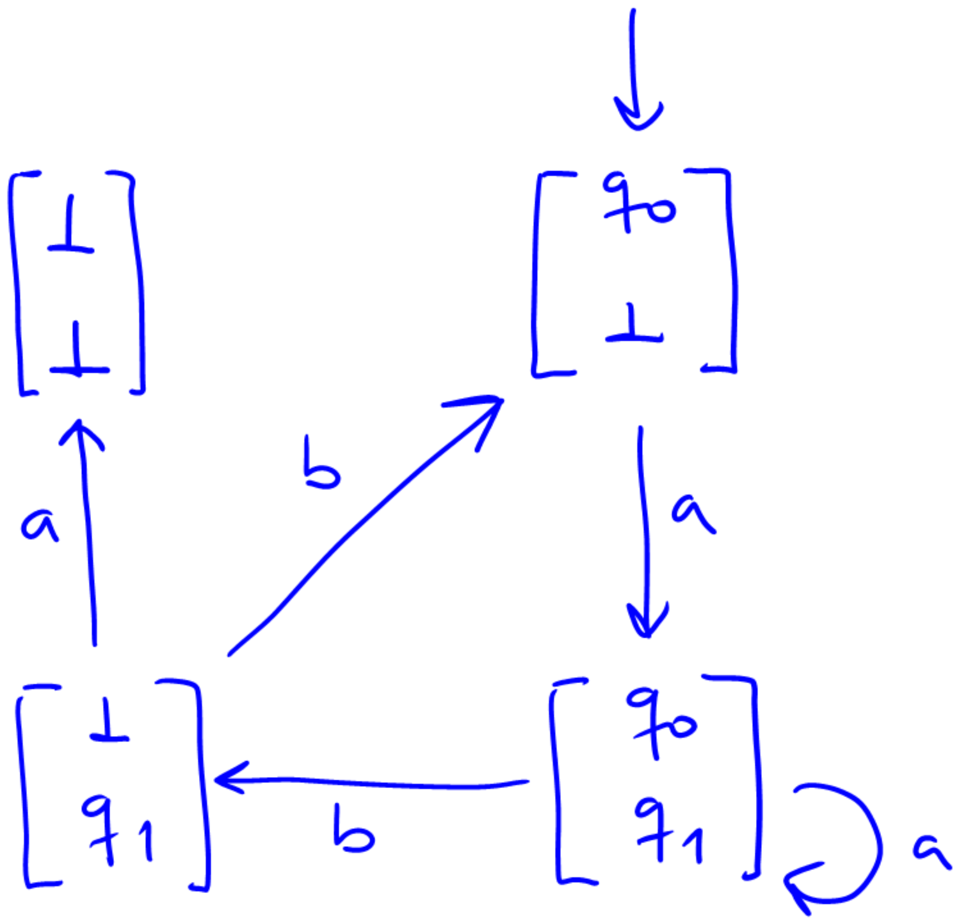


Slicing a dag






a
a
q_0
q_1
b


⊥ ⊥
q0 ⊥
a
b
a
⊥ q1
q0 q1
b
a

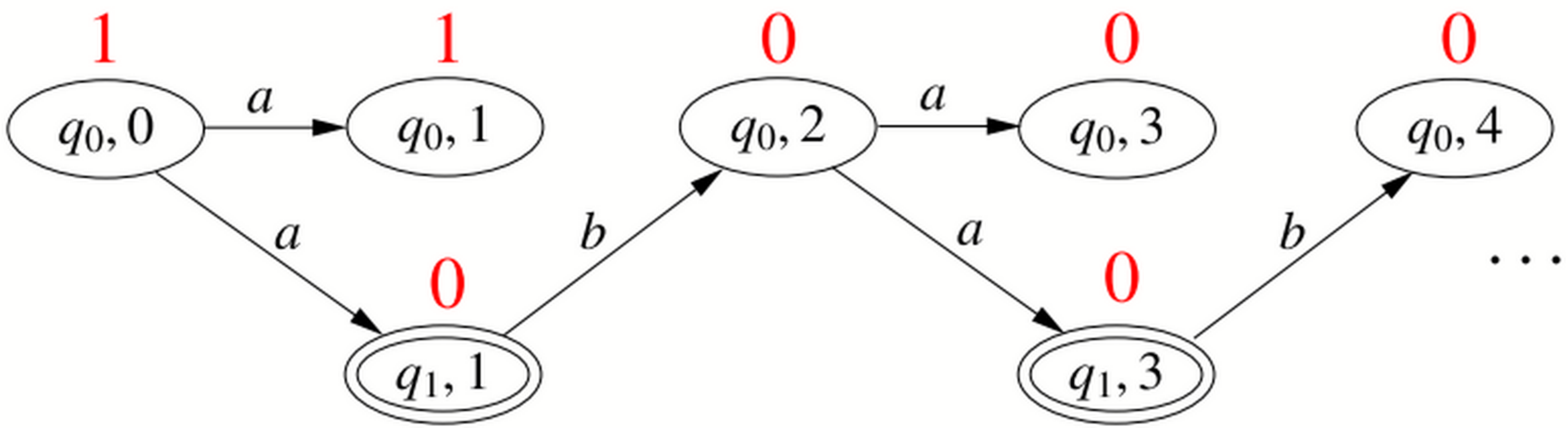
The powerset construction only retains information about which states are visited, but no information about the connections between them.

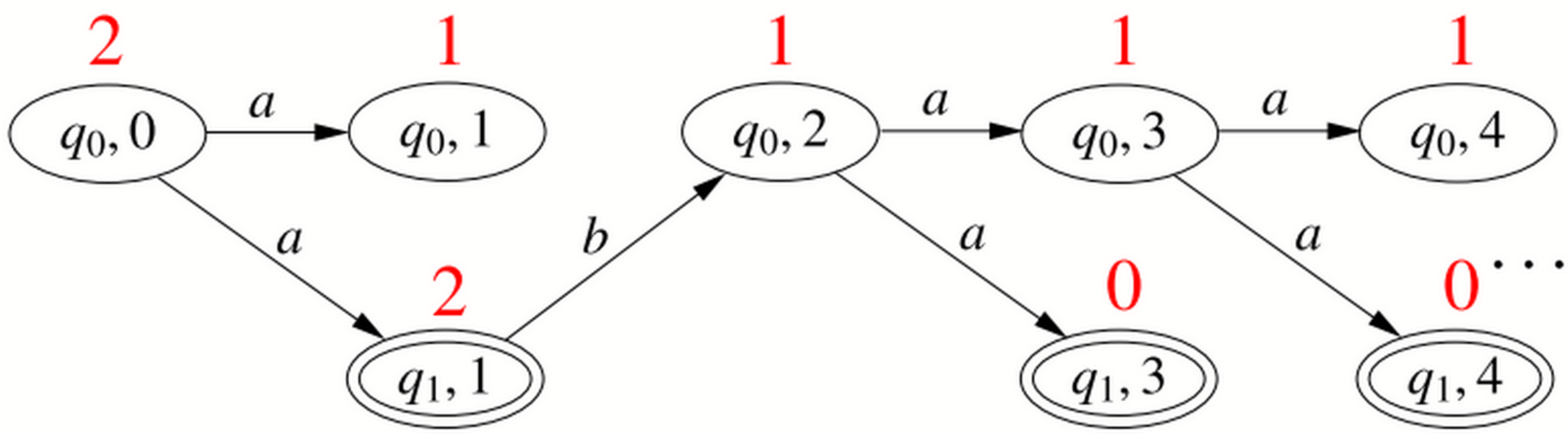
So we have to add information to the states …

Rankings

A ranking of dag(w) is a function that assigns to each node of dag(w) a rank: a number in the range [0 … 2n] satisfying two conditions:

- ranks never increase along paths, and
- ranks of accepting states are even





The ranks along an infinite path of dag(w) stabilize at an stable rank.
Observe: if the stable rank of a path is odd, then the path cannot visit accepting states infinitely often.
So: if all paths have odd stable ranks, the word is not accepted.
If all paths have odd stable ranks, we say the ranking is odd.

If dag(w) admits an odd ranking, then A does not accept w.

Imagine we can prove: if A does not accept w, then dag(w) admits an odd ranking. Then:

design C(A) so that it accepts w if and only if dag(w) admits an odd ranking.

A does not accept w => dag(w) admits an odd ranking

Assume A does not accept w. We construct an odd ranking for dag(w).

Procedure:

- we proceed in n rounds, each round with steps n.0 and n.1
- each step removes a set of nodes together with all its descendants
- the nodes removed at step i.j get rank 2i+j

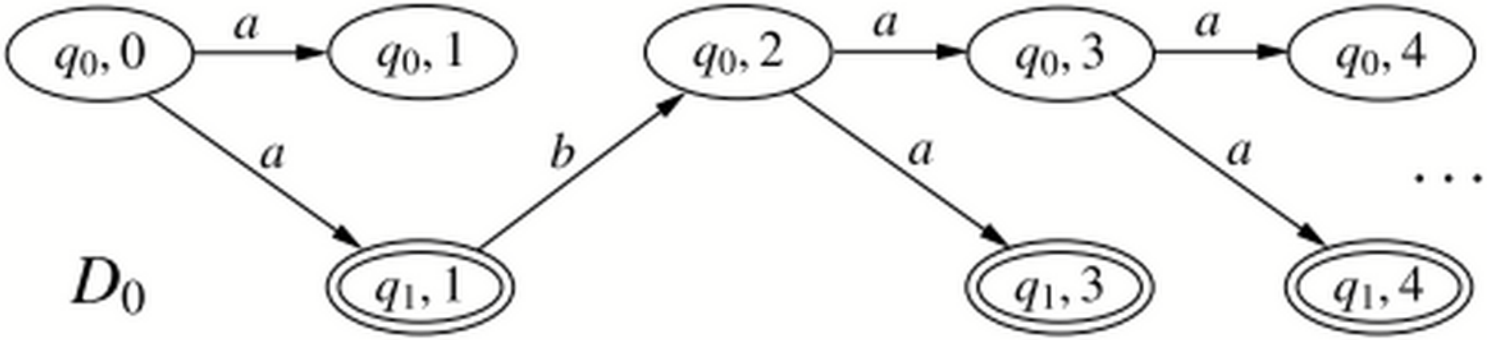
The steps

Step i.0: remove all nodes having only finitely many successors.

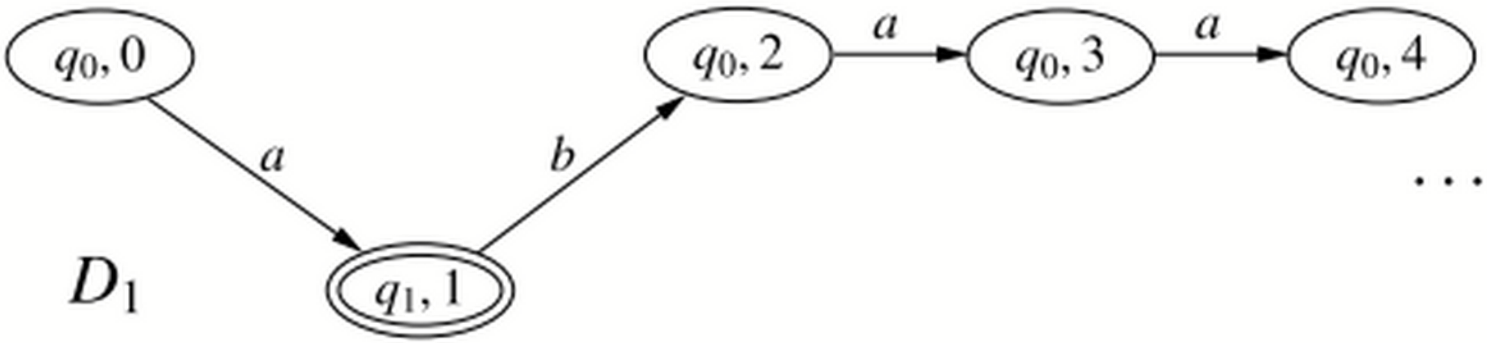
Step i.1: remove nodes none of whose descendants (including themselves) is accepting

Observe: ranks along a path cannot increase
accepting states can only be removed at step i.0

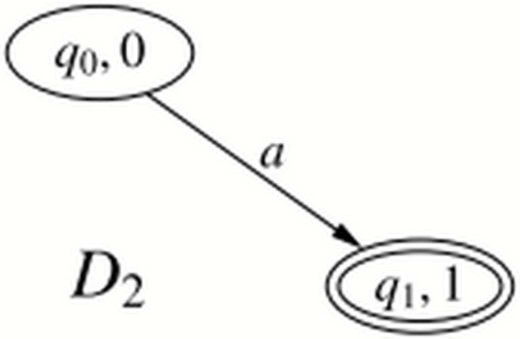
Remains to prove: after n rounds there are no nodes left.



$q_0, 0$
a
$q_0, 1$
a
$q_1, 1$
b
$q_0, 2$
a
$q_0, 3$
a
$q_0, 4$
a
$q_1, 3$
a
$q_1, 4$
$\cdots$
D_0



$q_0, 0$
a
$q_1, 1$
b
$q_0, 2$
a
$q_0, 3$
a
$q_0, 4$
$\cdots$
D_1

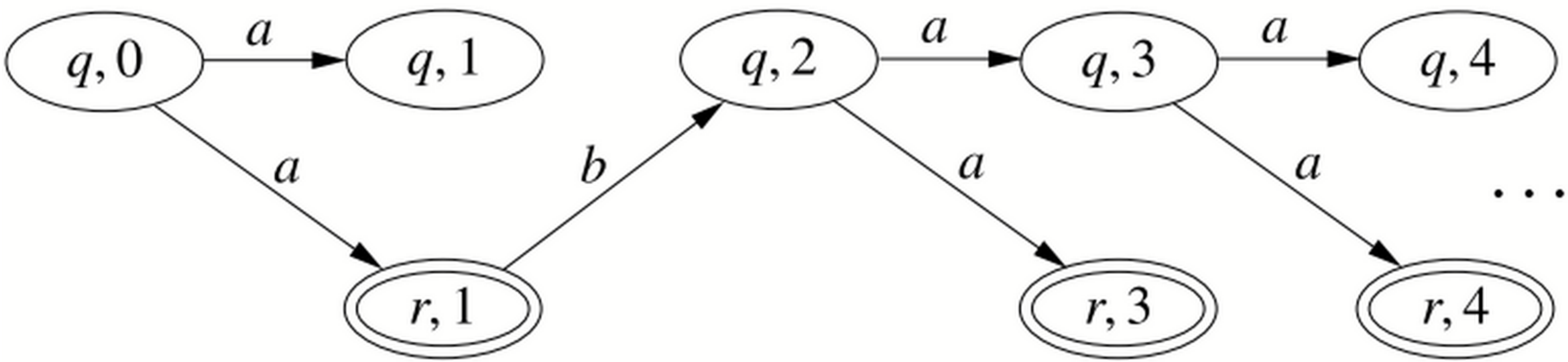


$q_0, 0$
a
$q_1, 1$
D_2

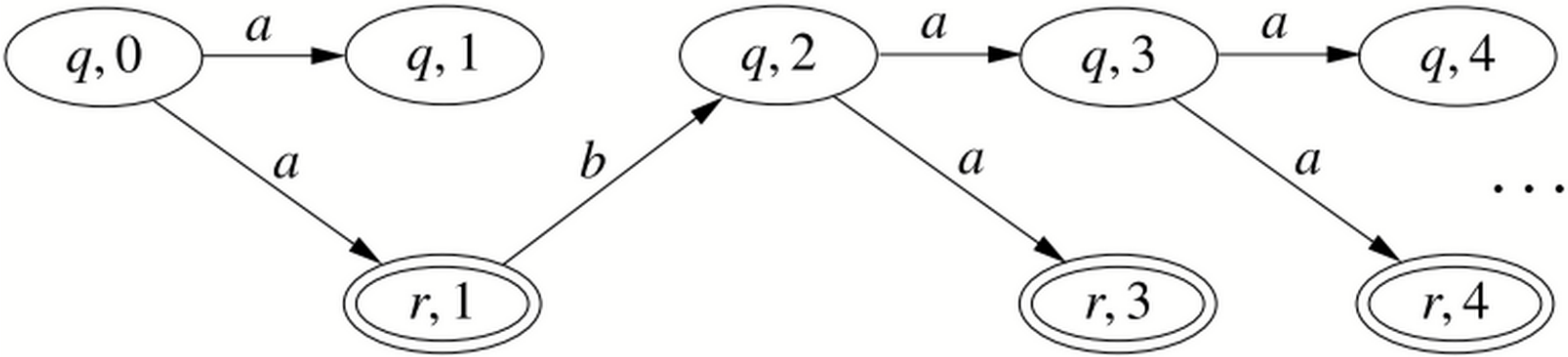
Observe:
- ranks along a path cannot increase
- accepting states can only be removed at step i.0

Remains to prove: after n rounds there are no nodes left.

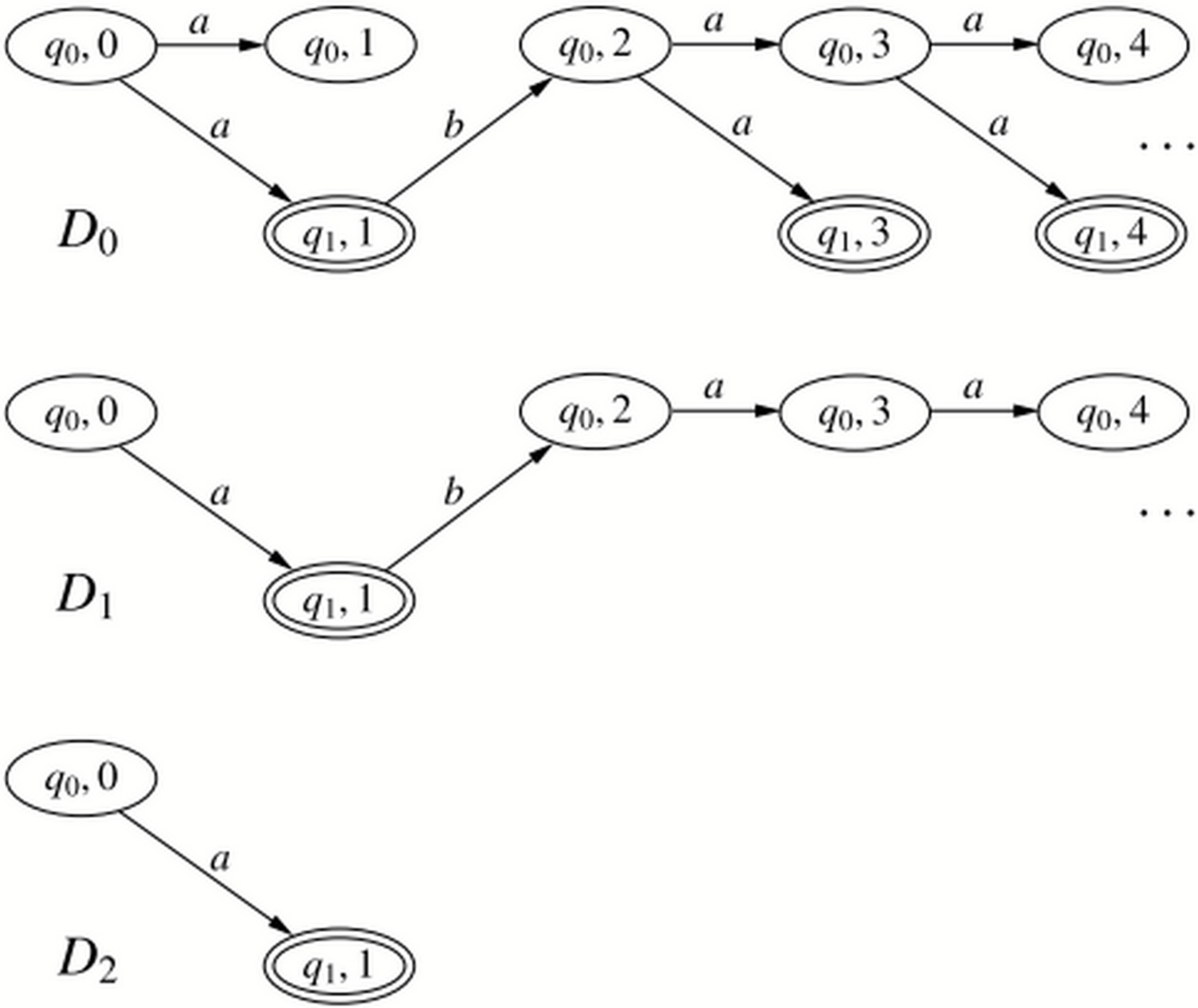
For this we first split the dag into "slices"



Each slice has a "width"



The "width" of the whole dag is defined as the largest width that appears infinitely often.



$q_0, 0$
a
$q_0, 1$
a
$q_1, 1$
b
$q_0, 2$
a
$q_0, 3$
a
$q_0, 4$
a
$q_1, 3$
a
$q_1, 4$
...
D_0
$q_0, 0$
a
$q_1, 1$
b
$q_0, 2$
a
$q_0, 3$
a
$q_0, 4$
...
D_1
$q_0, 0$
a
$q_1, 1$
D_2

Little lemma: Each round decreases the width of the dag by at least 1.

Since the initial width is at most n, it follows that there are at most n rounds.

So every node gets assigned a number between 0 and 2n.

Where are we?

Given: An NBA A over alphabet Sigma.

Achieved so far:

1. We define a mapping *dag* which assigns to each word $w \in \Sigma^\omega$ a directed acyclic graph $dag(w)$. We also define an *odd ranking* of $dag(w)$ as a labelling of the nodes of $dag(w)$ by natural numbers satisfying certain properties.

2. We prove that w is rejected by A if and only if $dag(w)$ admits an odd ranking.

To be done:

3. We construct an NBA $\overline{A}$ which accepts w if and only if $dag(w)$ admits an odd ranking.

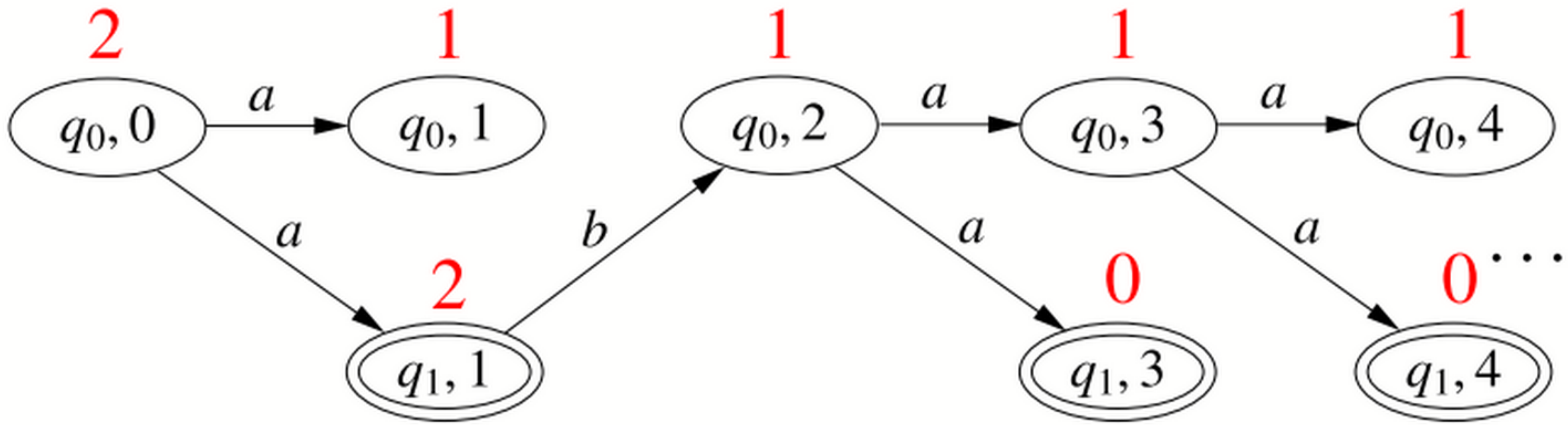
3. We construct an NBA $\overline{A}$ which accepts w if and only if $dag(w)$ admits an odd ranking.

First idea:

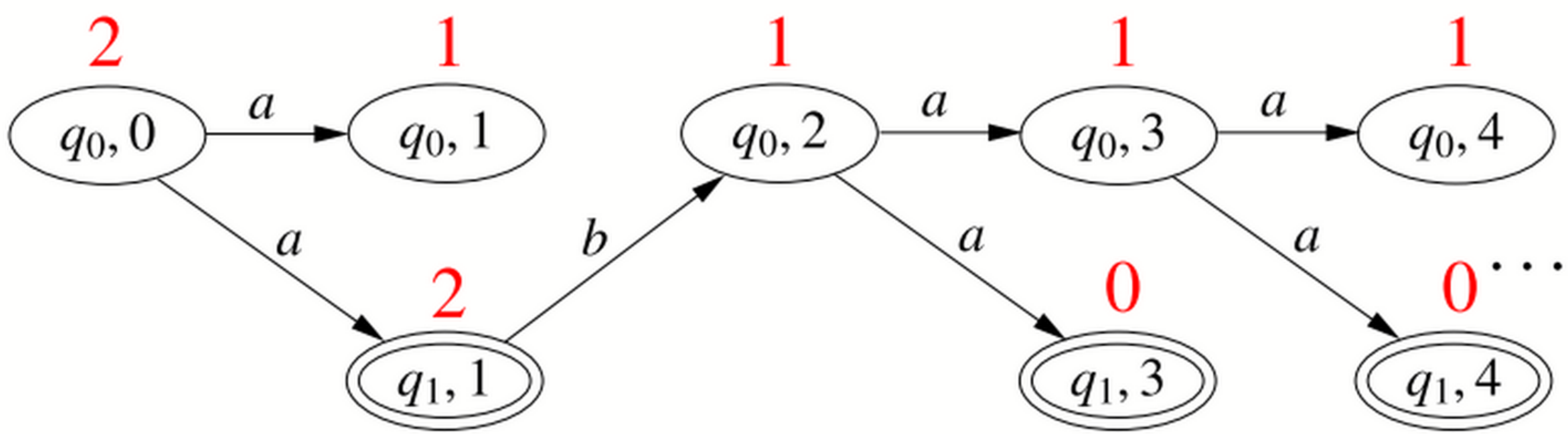
- Choose the states and transitions of C(A) so that the runs of C(A) on w correspond to the rankings of dag(w).

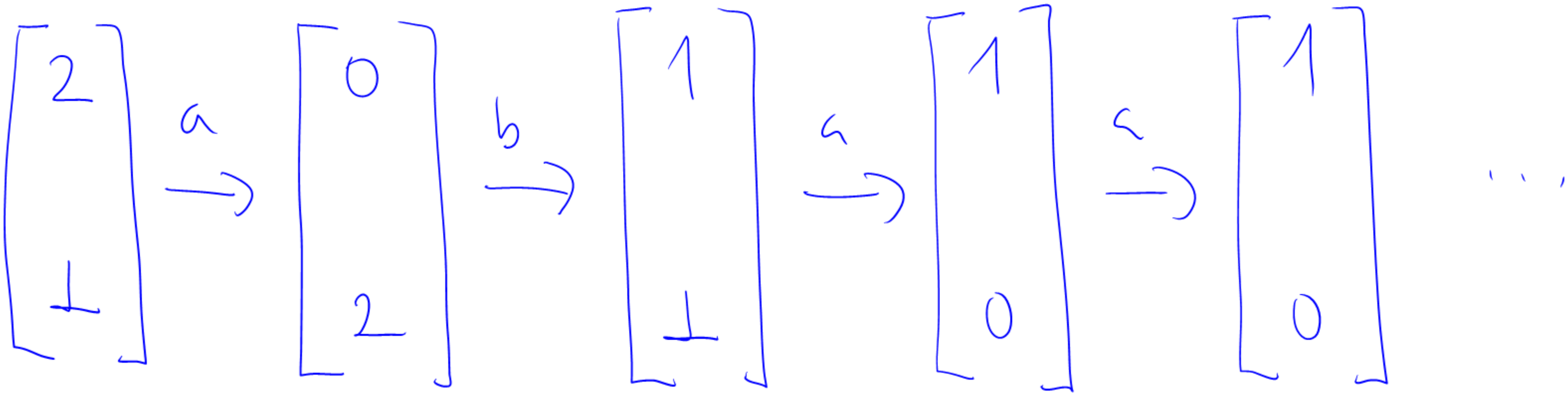
- Choose the accepting states so that the accepting runs correspond to the odd rankings.

Choosing the states and transitions



Choosing the states and transitions





States: level rankings, vectors over [0,...,2n] and

Transitions: there is a transition between two level rankings (states) if we can see them as two consecutive slices in a ranking.

- Textual notation (lecture notes): lr |--> lr'

Choosing the accepting states

Unfortunately, there is no way to characterize the odd ranks by means of a Büchi condition.

Solution: add more information to the states.

Recall: a ranking is odd if every infinite path contains infinitely many nodes of odd rank.

Breakpoint set: set of levels such that between any two consecutive levels every path visits an state of odd rank at least once.

A ranking is odd iff it has an infinite breakpoint set.

The additional information will allow C(A) to identify breakpoints

Add to each level ranking a new component: A set O of states.

A state q belongs to O if there is a path starting at the last breakpoint and ending at q that does not visit any states of odd rank.

Informally: a state of O "owes" is the endpoint of a path that "owes" a visit to nodes of odd rank.

State: level ranking + set of owing states

Transitions take care of suitably updating the owing set.

- The initial state is the pair $[lr_0, \{q_0\}]$, where $lr_0(q_0) = 2n$, and $lr_0(q) = \perp$ for every $q \neq q_0$. Observe that q_0 ‘owes” a visit to a node of odd rank.

When the set of owing states is nonempty, $\overline{A}$ updates it:

- If $O \neq \emptyset$, then $\langle lr', O' \rangle \in \overline{\delta}(\langle lr, O \rangle, a)$ iff $lr \overset{a}{\mapsto} lr'$ and $O' = \{q' \in \delta(O, a) \mid lr'(q') \text{ is even }\}$.

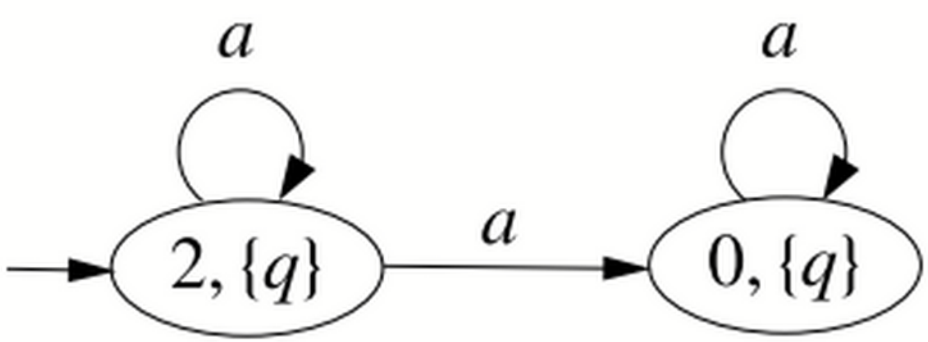
When the set of owing states is empty, $\overline{A}$ has reached a checkpoint, and it starts searching for the next one; all states of even rank are owing:

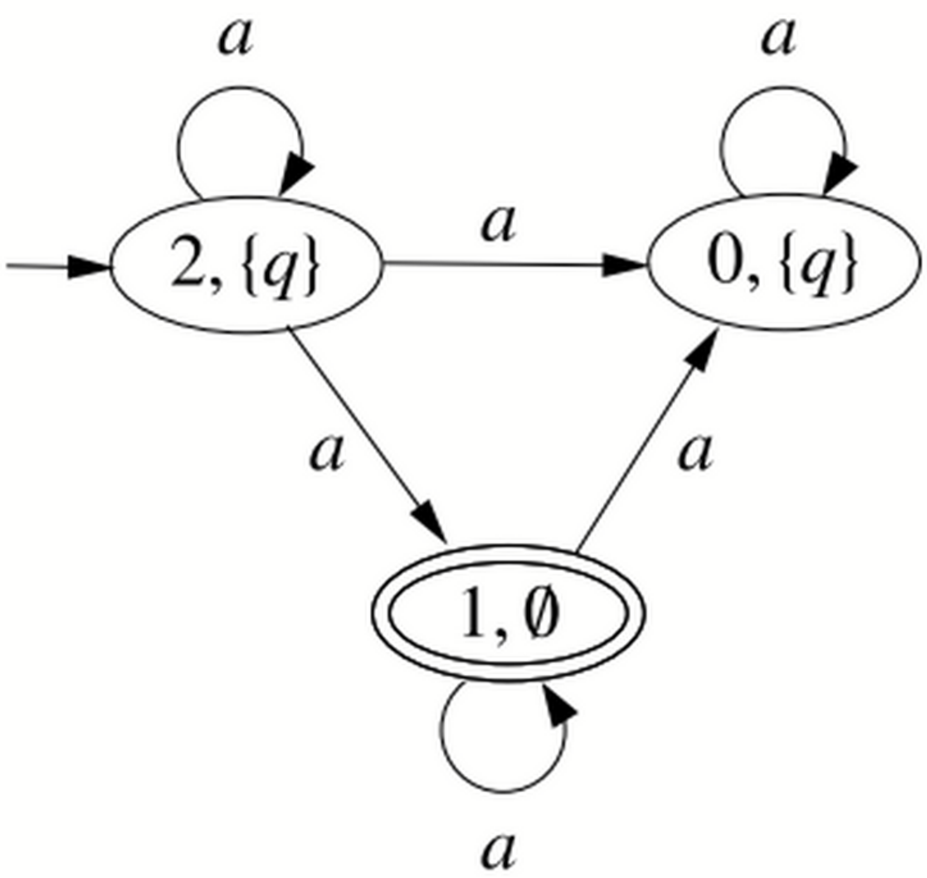
- If $O = \emptyset$, then $\langle lr', O' \rangle \in \overline{\delta}(\langle lr, O \rangle, a)$ iff $lr \overset{a}{\mapsto} lr'$ and $O' = \{q' \in Q \mid lr'(q') \text{ is even }\}$.

The accepting states are those at which a checkpoint is reached:

- a state $[lr, O]$ is accepting if $O = \emptyset$.

Example 12.4 We construct the complements $\overline{A}_1$ and $\overline{A}_2$ of the two possible NBAs over the alphabet $\{a\}$ having one state and one transition: $B_1 = (\{q\}, \{a\}, \delta, \{q\}, \{q\})$ and $B_2 = (\{q\}, \{a\}, \delta, \{q\}, \emptyset)$, where $\delta(q, a) = \{q\}$. The only difference between B_1 and B_2 is that the state q is accepting in B_1, but not in B_2. We have $L_\omega(A_1) = a^\omega$ and $L_\omega(A_2) = \emptyset$.





CompNBA(A)

Input: NBA $A = (Q, \Sigma, \delta, q_0, F)$

Output: NBA $\overline{A} = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F})$ with $\mathcal{L}_\omega(\overline{A}) = \overline{\mathcal{L}_\omega(A)}$

$\overline{Q}, \overline{\delta}, \overline{F} \leftarrow \emptyset$
$\overline{q}_0 \leftarrow [lr_0, \{q_0\}]$
$W \leftarrow \{\ [lr_0, \{q_0\}]\ \}$
while $W \neq \emptyset$ **do**
    pick $[lr, P]$ **from** W; **add** $[lr, P]$ **to** $\overline{Q}$
    if $P = \emptyset$ **then add** $[lr, P]$ **to** $\overline{F}$
    for all $a \in \Sigma, lr' \in \mathfrak{R}$ such that $lr \overset{a}{\mapsto} lr'$ **do**
        if $P \neq \emptyset$ **then** $P' \leftarrow \{q \in \delta(P, a) \mid lr'(q) \text{ is even }\}$
        else $P' \leftarrow \{q \in Q \mid lr'(q) \text{ is even }\}$
        add $([lr, P], a, [lr', P'])$ **to** $\overline{\delta}$
        if $[lr', P'] \notin \overline{Q}$ **then add** $[lr', P']$ **to** W
return $(\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F})$

Size of C(A)

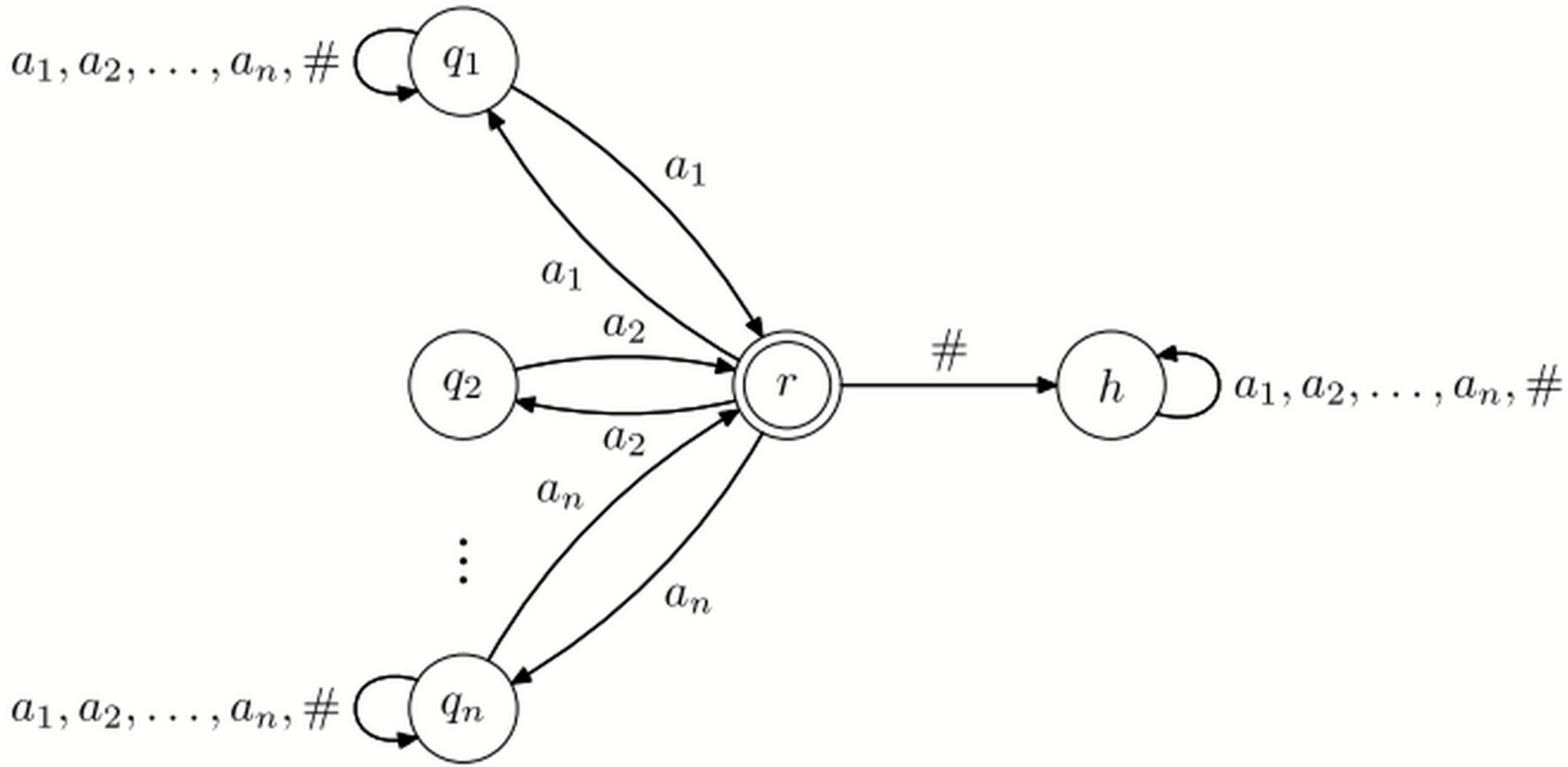
Assume A has n states.

Upper bound: number of level rankings is $(2n+2)^n$
number of owing sets is 2^n
So C(A) has at most $4^n (n+1)^n$ states
i.e., $2^{O(n \log n)}$ states.

Lower bound: $n!$ states, which is also $2^{O(n \log n)}$

Let $\Sigma_n = \{1, \ldots, n, \#\}$. We associate to a word $w \in \Sigma_n^\omega$ the following directed graph $G(w)$: the nodes of $G(w)$ are $1, \ldots, n$ and there is an edge from i to j if w contains infinitely many occurrences of the word ij. Define L_n as the language of infinite words $w \in A^\omega$ for which $G(w)$ has at least a cycle and define $\overline{L}_n$ as the complement of L_n.

We first show that for all $n \geq 1$, L_n is recognized by a Büchi automaton with $n + 2$ states. Let A_n be the automaton shown in Figure 12.10. We show that A_n accepts L_n.



Proposition 12.4 *For all $n \geq 1$, every NBA recognizing $\overline{L}_n$, has at least $n!$ states.*

Proof: We need some preliminaries. Given a permutation $\tau = \langle \tau(1), \ldots, \tau(n) \rangle$ of $\langle 1, \ldots, n \rangle$, we identify τ and the word $\tau(1) \ldots \tau(n)$. We make two observations:

(a) $(\tau\#)^\omega \in \overline{L}_n$ for every permutation τ.

(b) If a word w contains infinitely many occurrences of two different permutations τ and τ' of $1 \ldots n$, then $w \in L_n$.

Now, let A be a Büchi automaton recognizing $\overline{L}_n$, and let τ, τ' be two arbitrary permutations of $(1, \ldots, n)$. By (a), there exist runs ρ and ρ' of A accepting $(\tau\#)^\omega$ and $(\tau'\#)^\omega$, respectively. We prove that the intersection of $\inf(\rho)$ and $\inf(\rho')$ is empty. This implies that A contains at least as many final states as permutations of $(1, \ldots, n)$, which proves the Proposition.