Safra's Determinization

Emmanuel Filiot

Sources: Lecture* by **K. Narayan Kumar**

Chennai Mathematical Institute, India

* **pdf available at http://www.cmi.ac.in/~kumar/**

Outline

- Introduction
- Preliminaries
- 2-Exponential Determinization Procedure
- Optimal Determinization Procedure by Safra

Introduction

- Non-deterministic Büchi word automata are not determinizable
- Safra gives a construction to go to Deterministic Rabin automata
- This construction is optimal 2^O(n.log n)

Why a lecture on it?

- Safra's procedure manipulates a complex state space (trees of subsets of states)
- most explanations are quite intricate
- in this lecture: structure of accepting runs to structure of the state space

Related Work

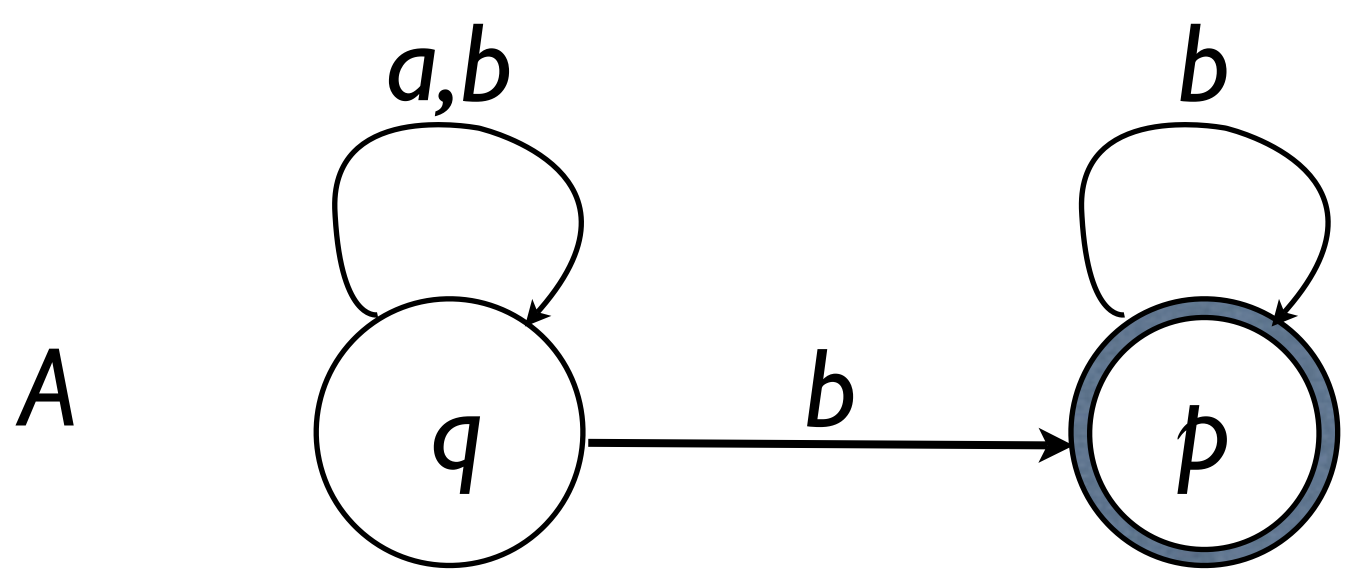
- Muller, 1963: uncorrect construction
- Mc Naughton, 1966. Gives a determinization to Muller automata.
- Safra, 1988. To Rabin automata, 2^O(n.log n)
- Muller/Schupp, 1995.

Outline

- Introduction
- Preliminaries
- 2-Exponential Determinization Procedure
- Optimal Determinization Procedure by Safra

Non-deterministic Büchi Automata (NBW)

alphabet $\Sigma = \{a,b\}$



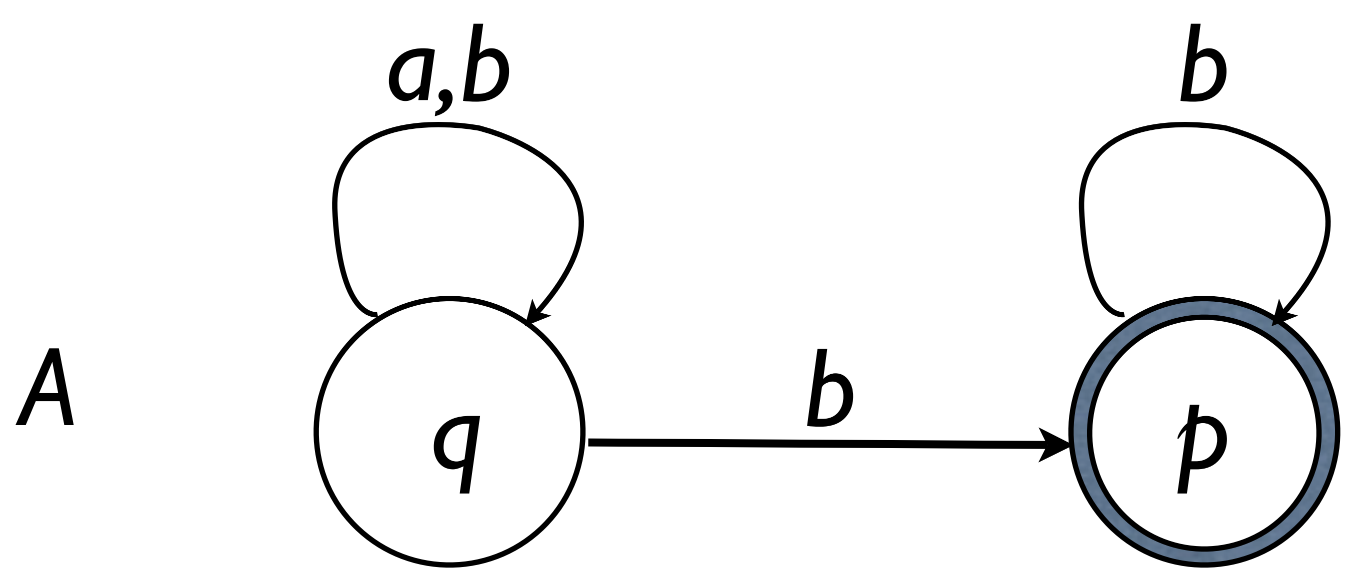
Acceptance Condition

Visits p infinitely often

$L(A) = ???$

Non-deterministic Büchi Automata (NBW)

alphabet $\Sigma = \{a,b\}$



Acceptance Condition

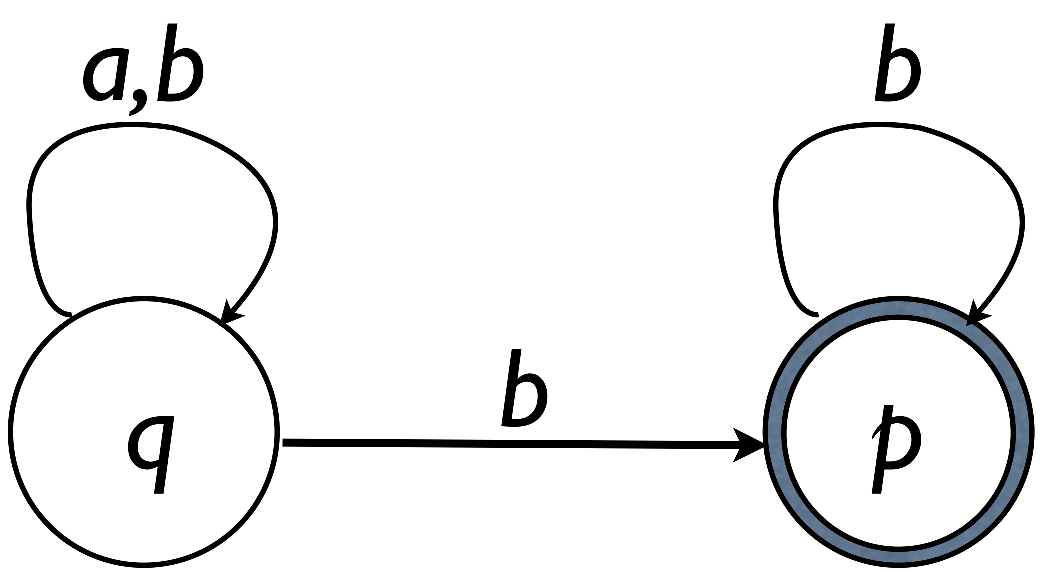
Visits p infinitely often

$L(A)$ = words w in which a occurs finitely many times

Non-deterministic Büchi Automata (NBW)

alphabet $\Sigma = \{a,b\}$

A



Acceptance Condition

Visits p infinitely often

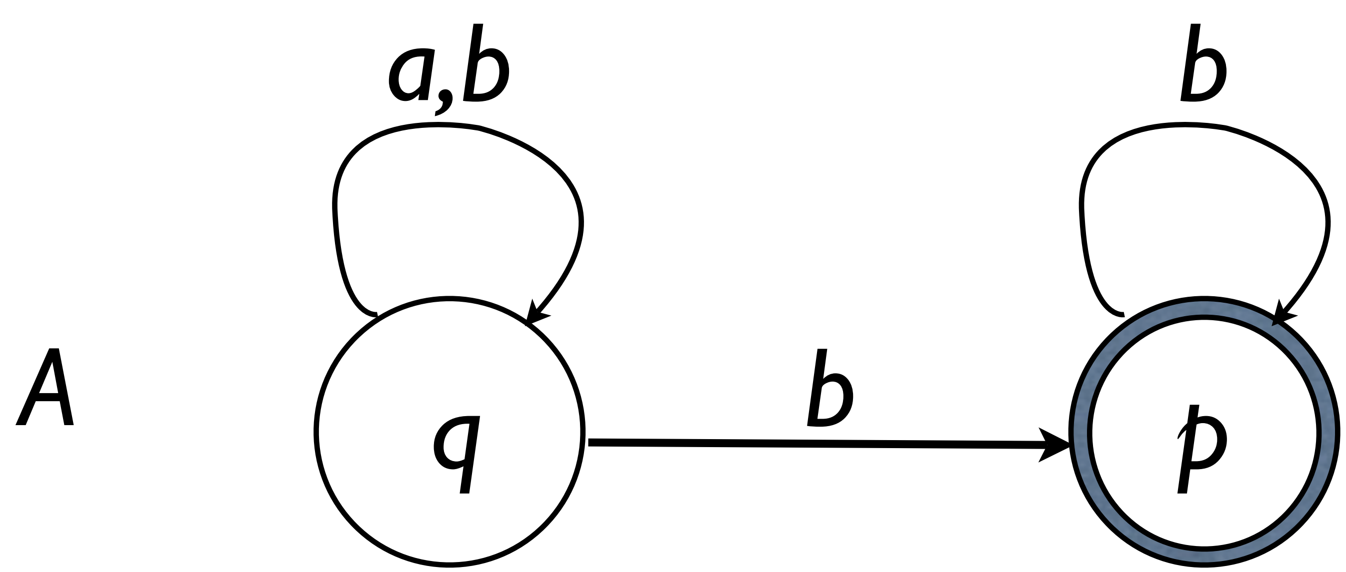
w = $ababaaabab bbbbbbbb....$

r = $qqqqqqqqqpppppppp.....$

r' = $qqqqqqqqqqqqqpppp......$

Non-deterministic Büchi Automata (NBW)

alphabet $\Sigma = \{a, b\}$



Acceptance Condition

Visits p infinitely often

$L(A)$ = words w in which a occurs finitely many times

There is no deterministic NBW B s.t. $L(A) = L(B)$

Notations

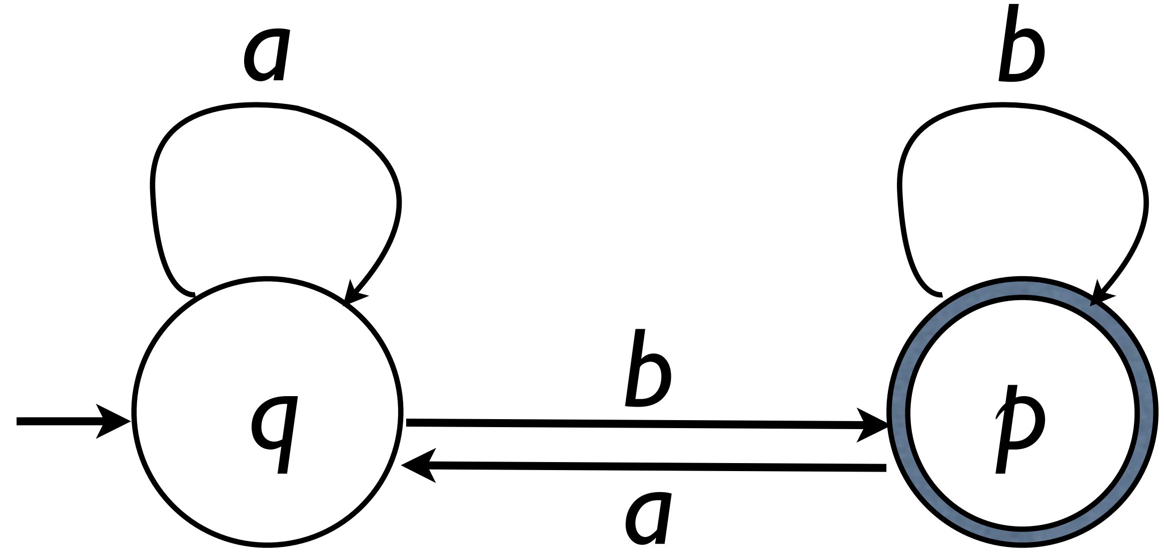
- $A = (\Sigma, Q, i, F, \Delta)$
- Q: set of states with initial state i
- F: set of final states
- Δ: set of rules
- runs: r
- inf(r) = states that occur <u>infinitely</u> often in r
- fin(r) = states that occur <u>finitely</u> often in r

Rabin Automata

- Acceptance Condition:
  Finite set $\Phi = (I_i,F_i)_{1 \leq i \leq n}$ $F_i,E_i \subseteq Q$
- A run r is accepting if $\exists$i:inf(r)$\subseteq$I$_i$, fin(r)=F$_i$

Rabin Automata (Example)

- Finitely many *a*'s

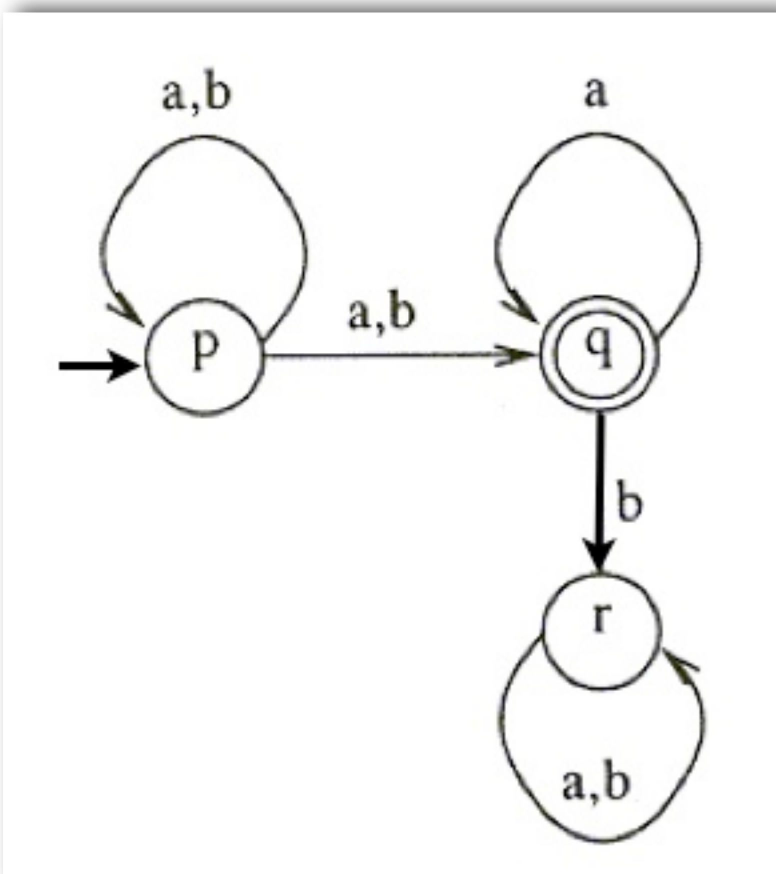


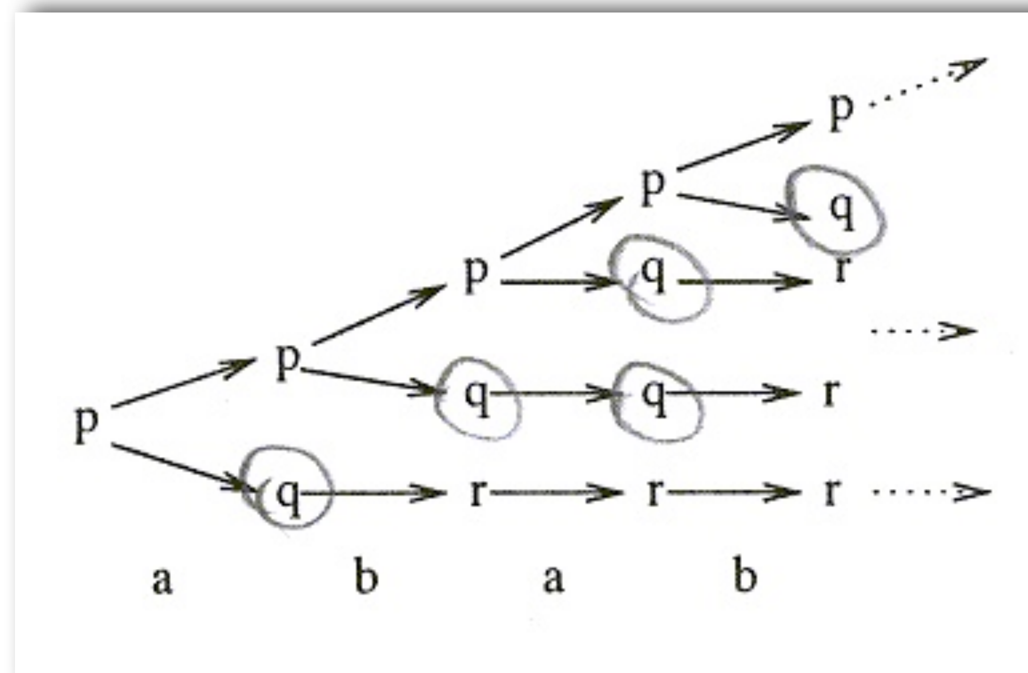
- $\Phi = \{ (\{p\}, \{q\}) \}$

Outline

- Introduction
- Preliminaries
- 2-Exponential Determinization Procedure
- Optimal Determinization Procedure by Safra

Running Example





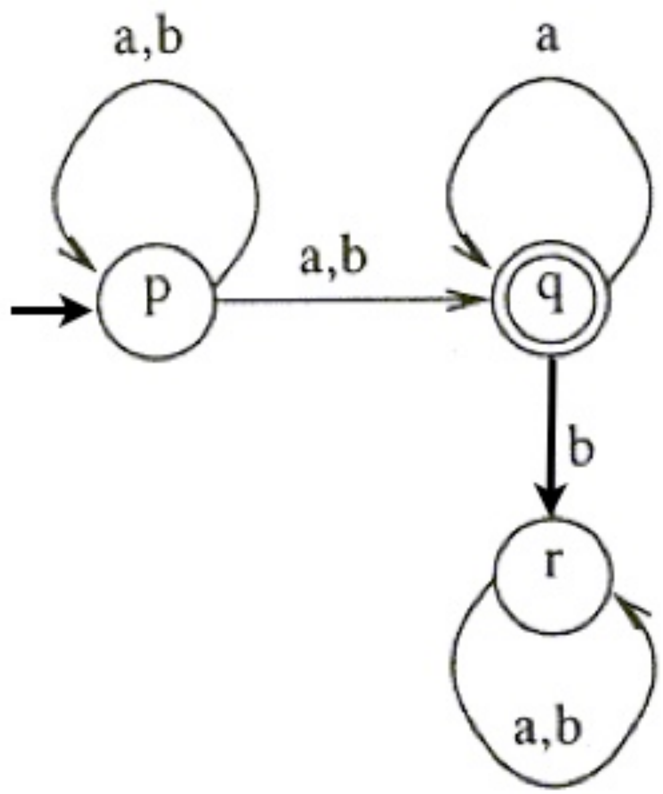
$A = (\Sigma, Q, p, F, \delta)$

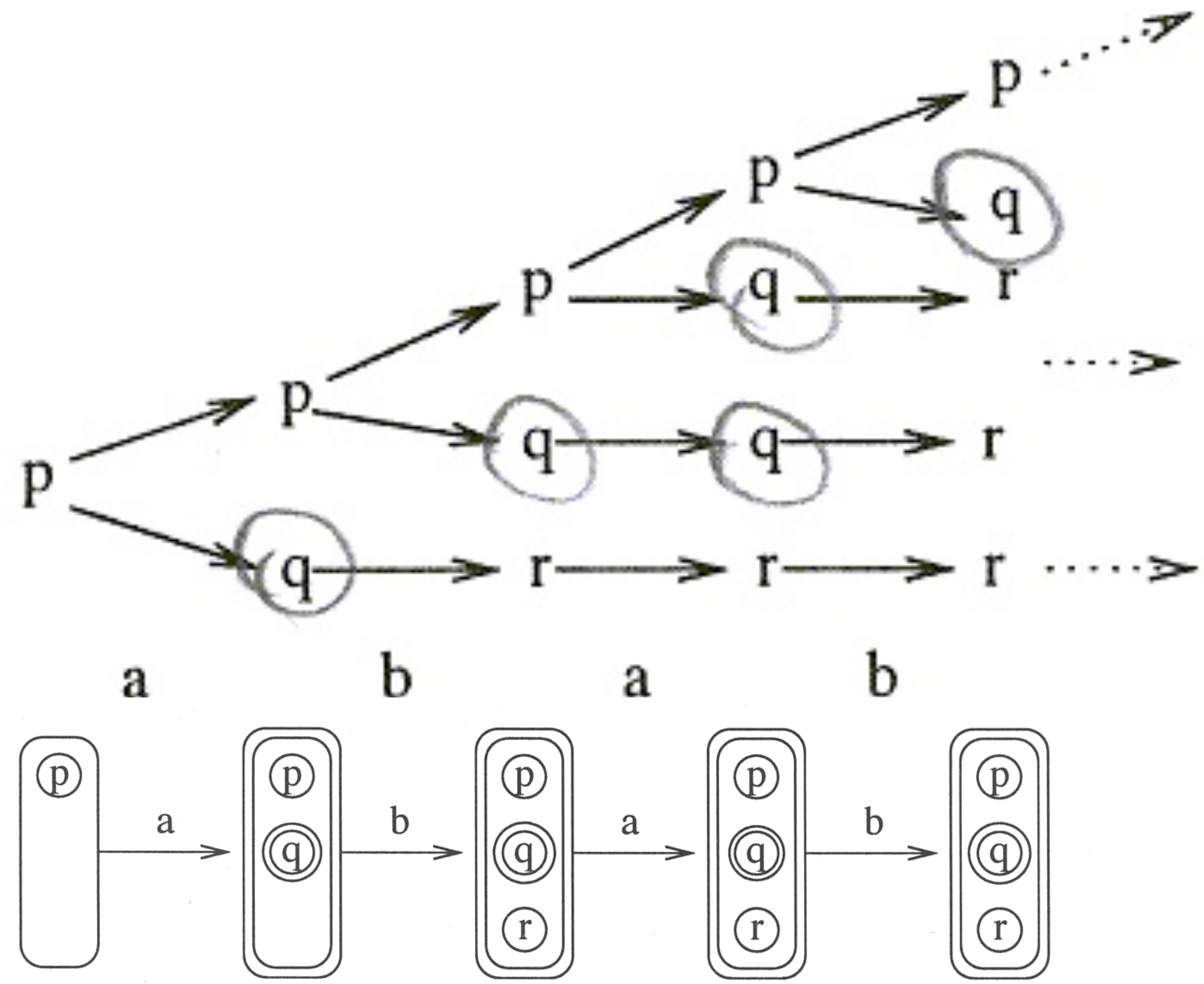
L(A) = “finitely many b”

Infinitely many runs: the deterministic automaton should simulate all those runs

Powerset Automaton

$$A_p = (2^Q, \Sigma, \delta_p, \{s\}, \{X \mid X \cap F \neq \emptyset\}) \text{ with } \delta_p(X,a) = \{q \mid \exists p \in X.\ q \in \delta(p,a)\}$$





Too generous: *abababababab* is accepted

Change the acceptance criterion

- Let $r = S_0S_1S_2S_3S_4S_5$ ... be a run of A_p on $w = a_0a_1a_2a_3a_4a_5$....
- Accept r if it can be decomposed into $S_{i1}S_{i2}S_{i3}$... such that:
    - $\forall k \forall q \in S_{i(k+1)} \exists q' \in S_{ik}$ such that there is a run from q to q' on $a_{ik}a_{ik+1}...a_{i(k+1)}$
    - this run visits an accepting state
- clearly A accepts w

Express it as a Büchi acceptance condition

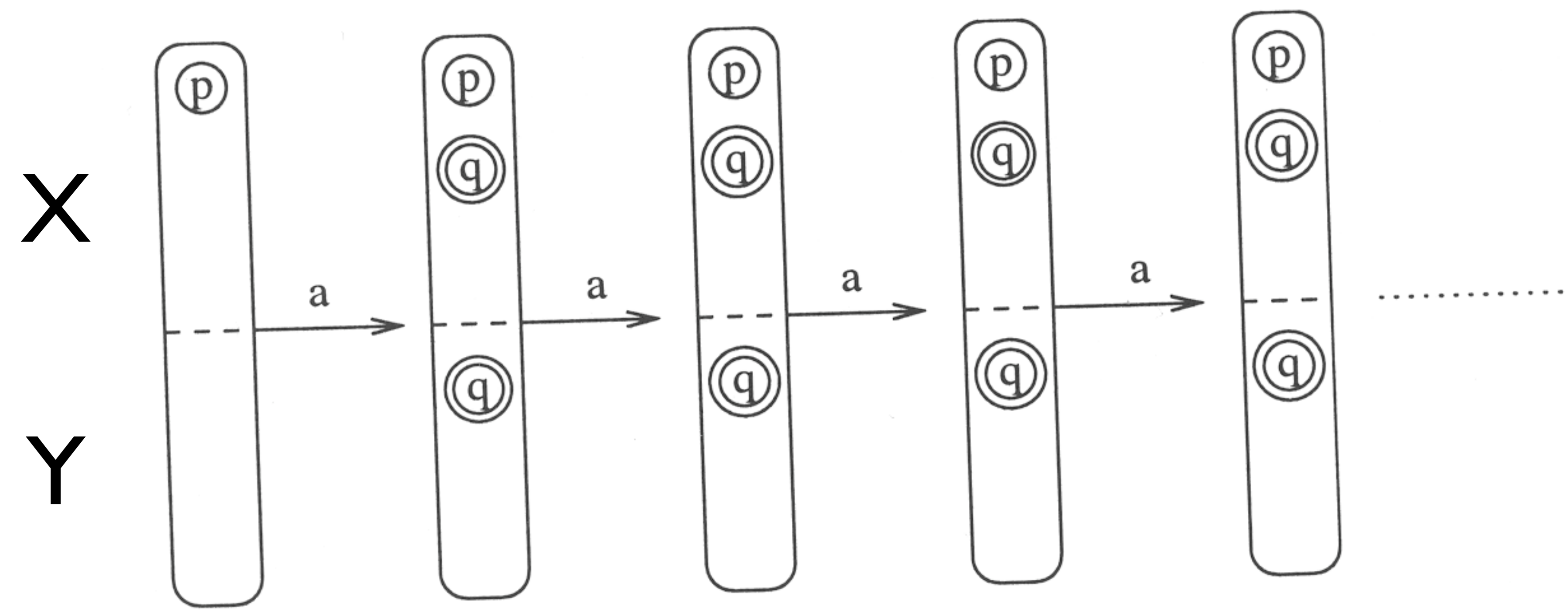
- turn the previous acceptance condition into a Büchi acceptance condition
- in a deterministic way ...

Let $A_m = (Q_m, \Sigma, \delta_m, (\{s\}, \emptyset), \{(X,X) \mid X \subseteq Q\}$ where

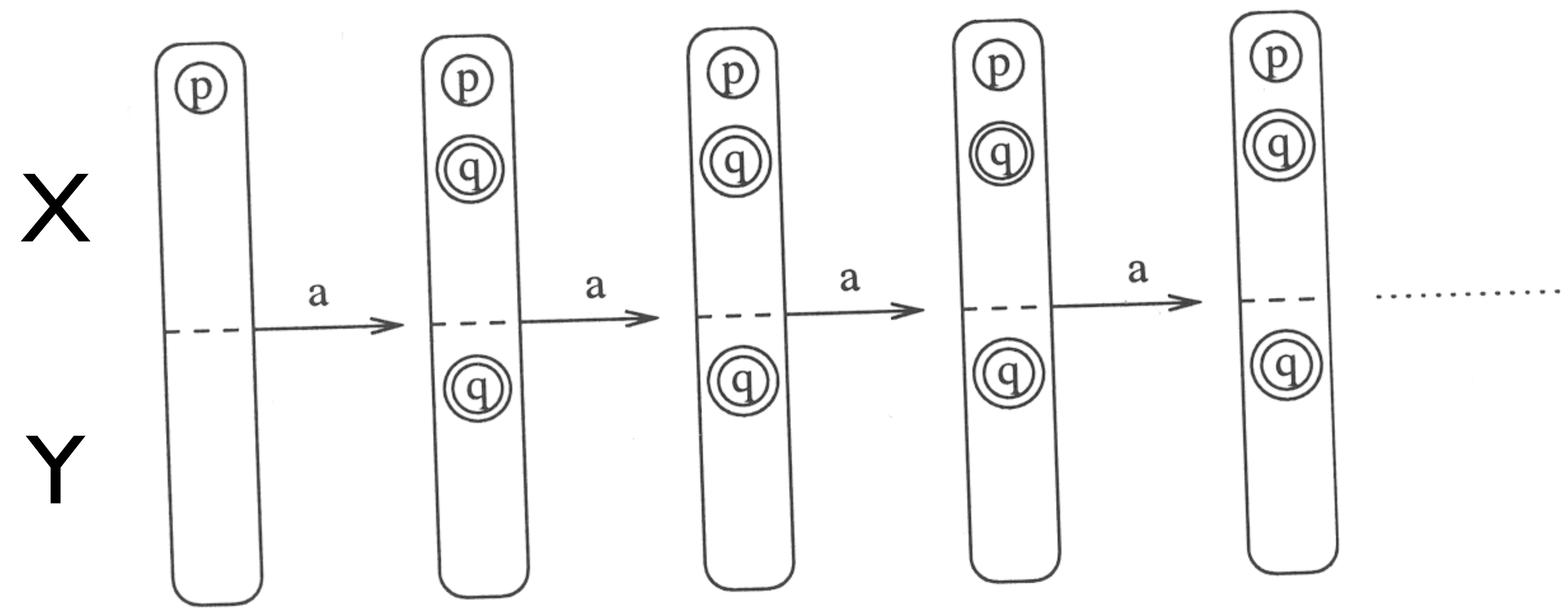
$$
\begin{array}{lcll}
Q_m & = & \{(X,Y) \mid Y \subseteq X \subseteq Q\} & \\
\delta_m((X,Y),a) & = & (\delta_p(X,a), \delta_p(Y,a) \cup \delta(X,a) \cap F) & \text{if } X \neq Y \\
\delta_m((X,X),a) & = & (\delta_p(X,a), \delta_p(X,a) \cap F) &
\end{array}
$$

$L(A_m) \subseteq L(A)$

aaaaaa... is not accepted



aaaaaa... is not accepted



Start A_m with initial state $(\{q\},\varnothing)$ after a have been read

Non-determinism is needed only on a finite prefix

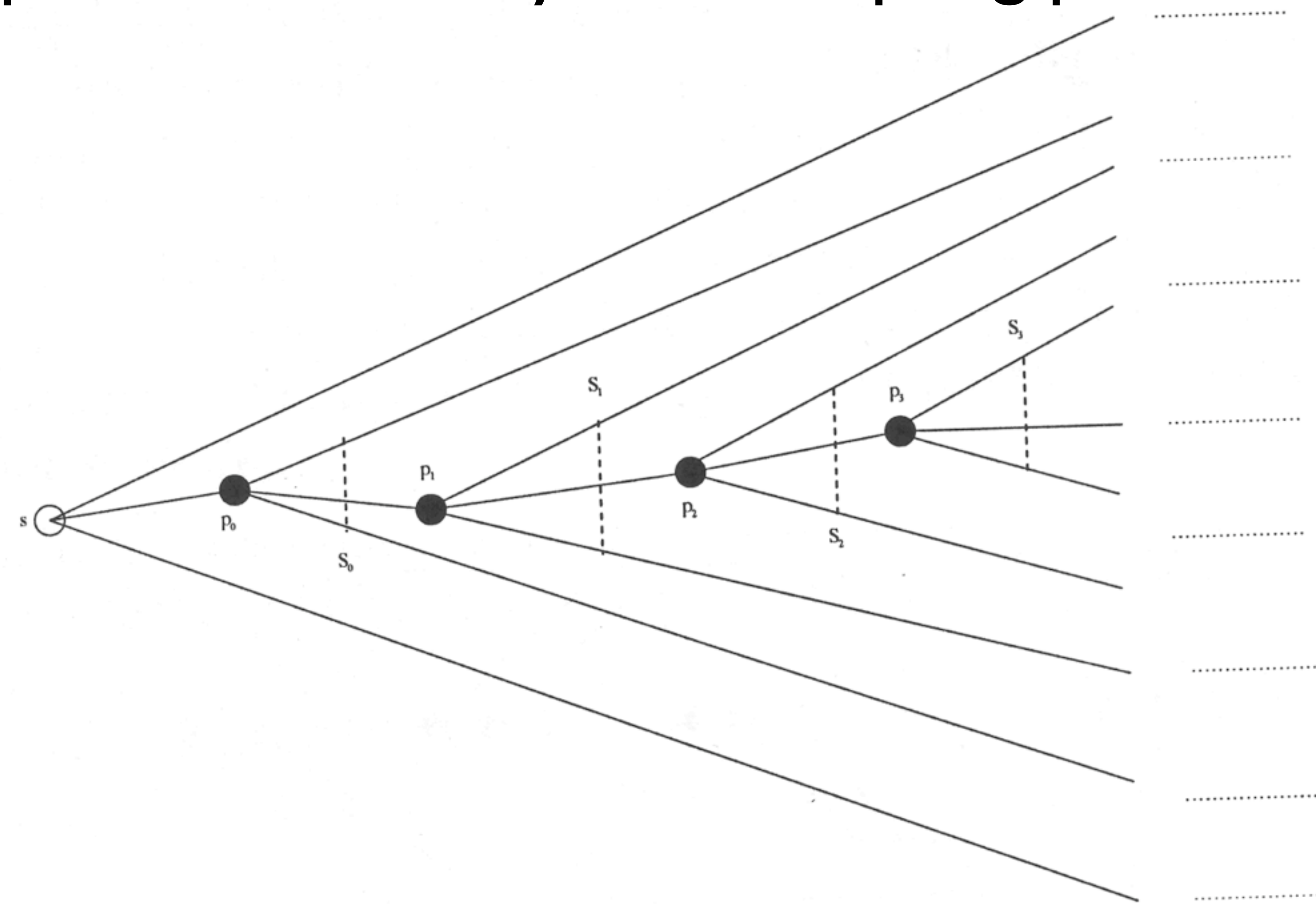
Definition: Automaton *B*

- run A
- non-deterministically choose to run A_m with initial state $(\{q\}, \emptyset)$, where q is the state reached by A so far.

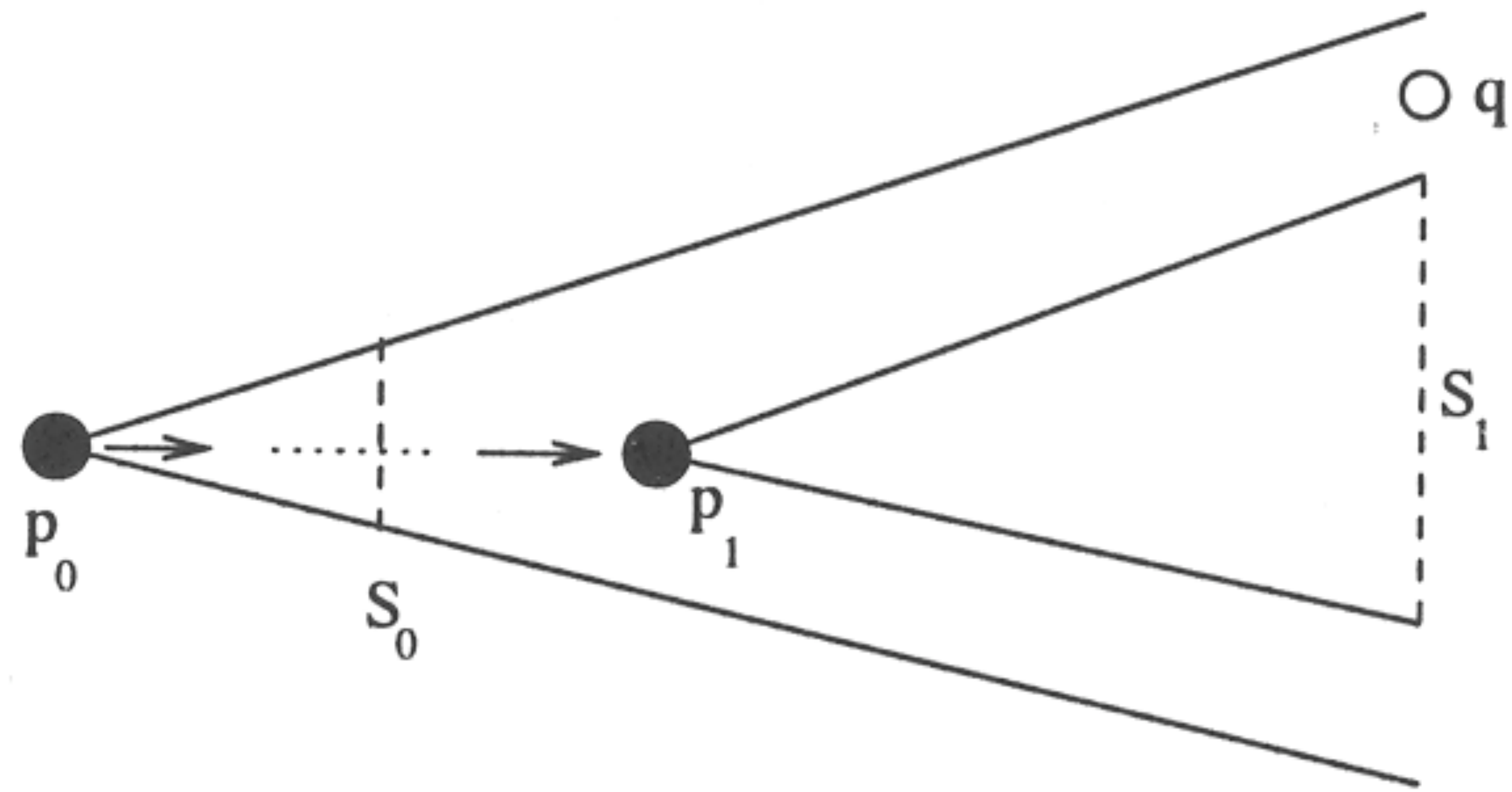
Theorem: $L(A) = L(B)$

Proof

- Consider a run tree and an accepting path of it
- Suppose there is only one accepting path



Proof

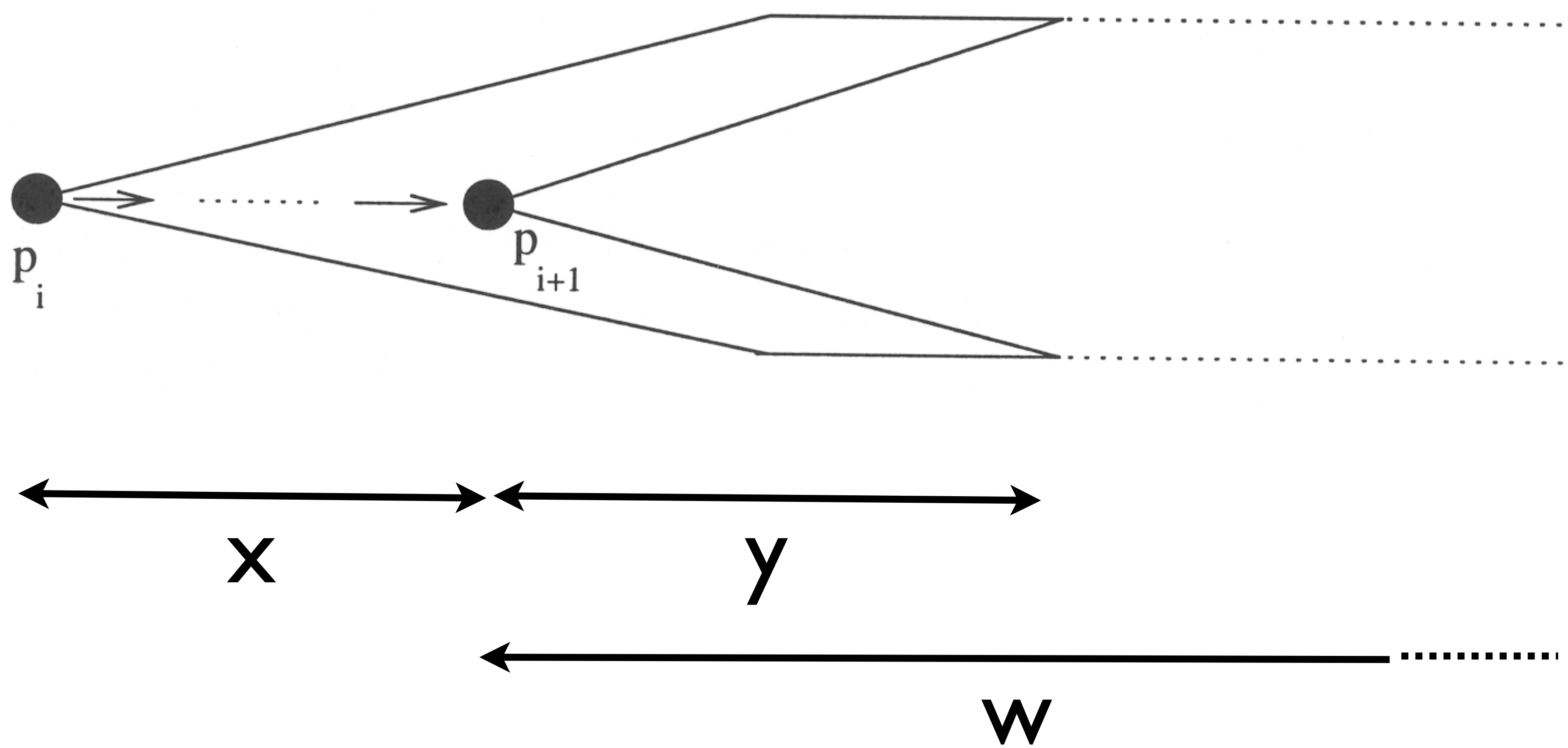


What happens if we fork a copy of A_m at position p_0 with initial state $(\{p_0\},\varnothing)$?

It might be the case this copy never reaches an accepting state of A_m (or only finitely many times)... but it can happen for finitely many positions p_i

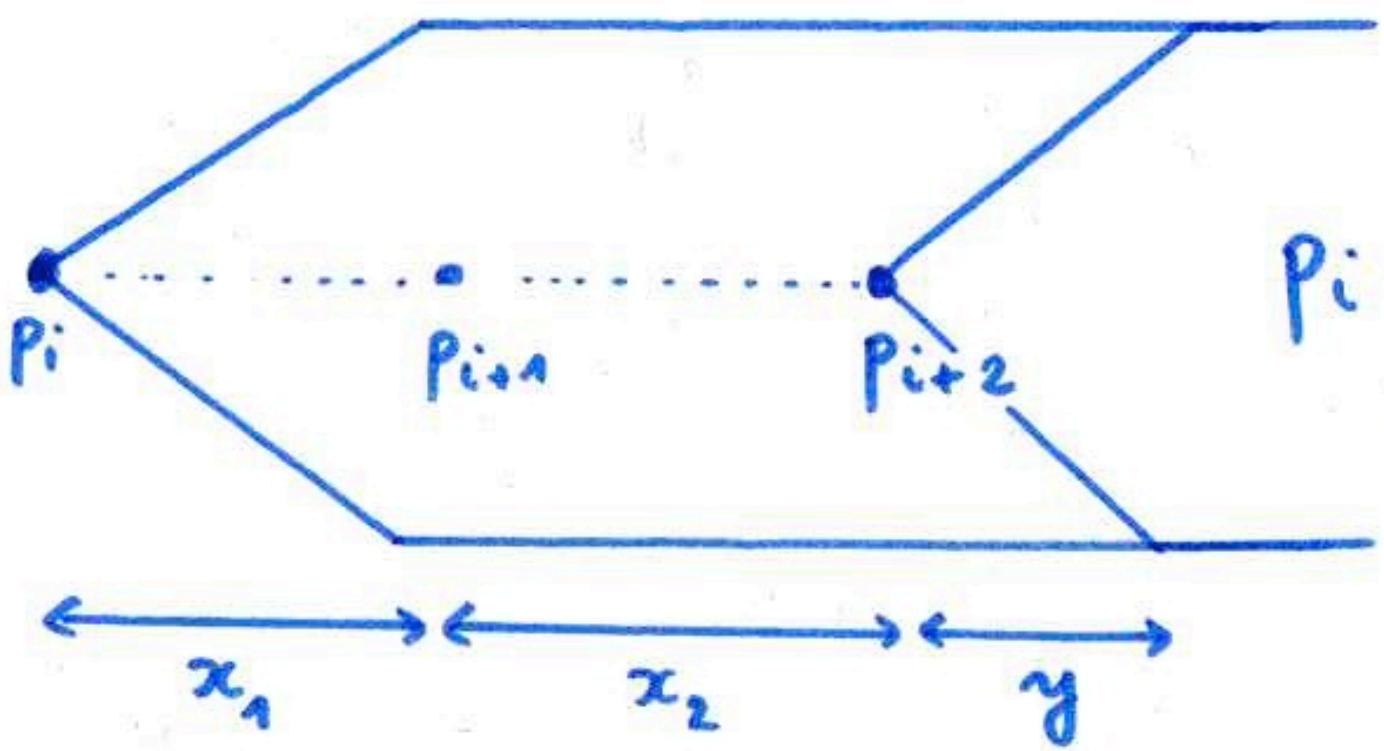
Proof

$p_i \sim p_{i+1}$ if $\exists y \in \Sigma^* \cap suff(w) : \delta_p(p_i, xy) = \delta_p(p_{i+1}, y)$



Proof

Equivalence classes are made of consecutive nodes.



$p_i \sim p_{i+2}$ since

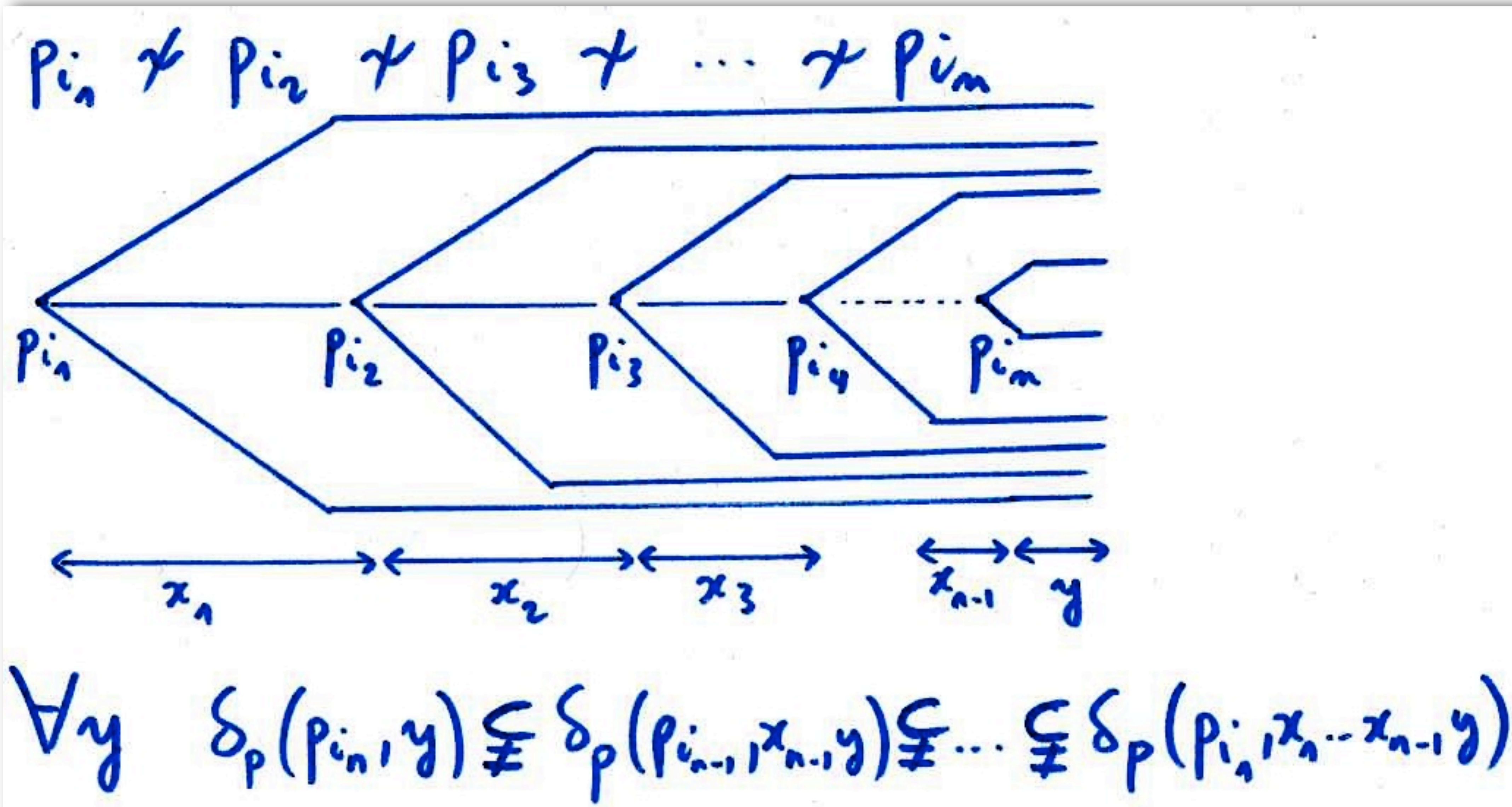
$$\delta_p(p_i, x_1 x_2 y) = \delta_p(p_{i+2}, y)$$

$$\delta_p(p_{i+2}, y) \subseteq \delta_p(p_{i+1}, x_2 y) \subseteq \delta_p(p_i, x_1 x_2 y)$$

$$\Longrightarrow \quad p_{i+2} \sim p_{i+1} \sim p_i$$

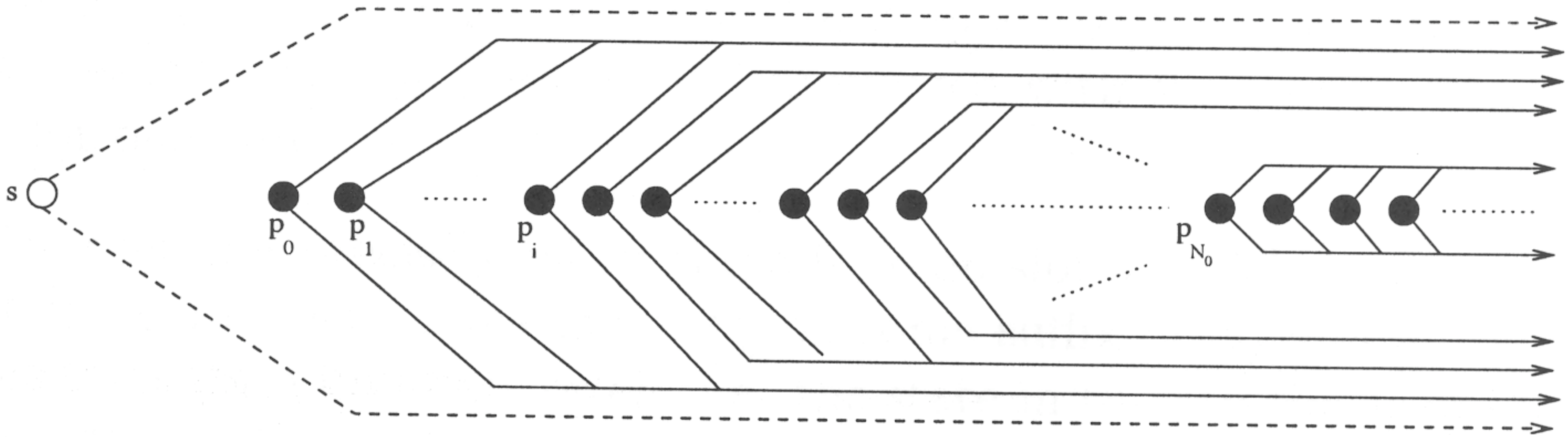
Proof

How many equivalence classes at most ?



Inclusions are strict: at most |Q| equivalence classes

Proof



$\boxed{\exists N_0, \forall i,j \geq N_0, p_i \sim p_j}$

Consequence: if we fork a copy of A_m at position p_{N0}, this copy will visit final states of A_m infinitely often

End of Proof.

Towards a deterministic Rabin

- Reminder: B = "run A and non-deterministically run A_m with initial state $(\{q_f\},\varnothing)$ where q_f is a final state reached by A so far"
- How to turn B into a deterministic Rabin?
- *Idea*:
  - run all the possible copies of A_m in parallel
  - for each final state q_f currently reached by A_p, fork a new copy of $\mathrm{A_m}$ with initial $(\{q_f\},\varnothing)$
  - merge copies that reach the same states

Formally ...

$$Q_d = 2^Q \times (2^Q \times 2^Q \cup \{\perp\})^K \qquad K = 2^{2n} + |F|$$

$$s_d = (\{s\}, \perp, \perp, \ldots \perp)$$

$\delta_d((S, W_1, W_2 \ldots, W_K), a)$ =

1/ First compute $V = (\delta_p(S,a), \delta_m(W_1,a), \delta_m(W_2,a) \ldots \delta_m(W_K,a))$

2/ for each final state p$\in \delta_p(S,a)$

add a copy of A_m with intial state *({p},∅)*

at the lowest free slot in *V*

3/ If several copies have the same state in A_m,

remove all but the one at lowest index

Acceptance: (F_i, I_i)

F_i: tuples that contain $\perp$ at ith component

I_i: tuples that contain a final state of A_m at ith component

Correctness

- $L(A) \subseteq L(A_d)$: take an accepting run, after a finite prefix, there is a finite copy of A_m that visits final states infinitely often. In the tuple, this copy may move to lower components, but only finitely many times...

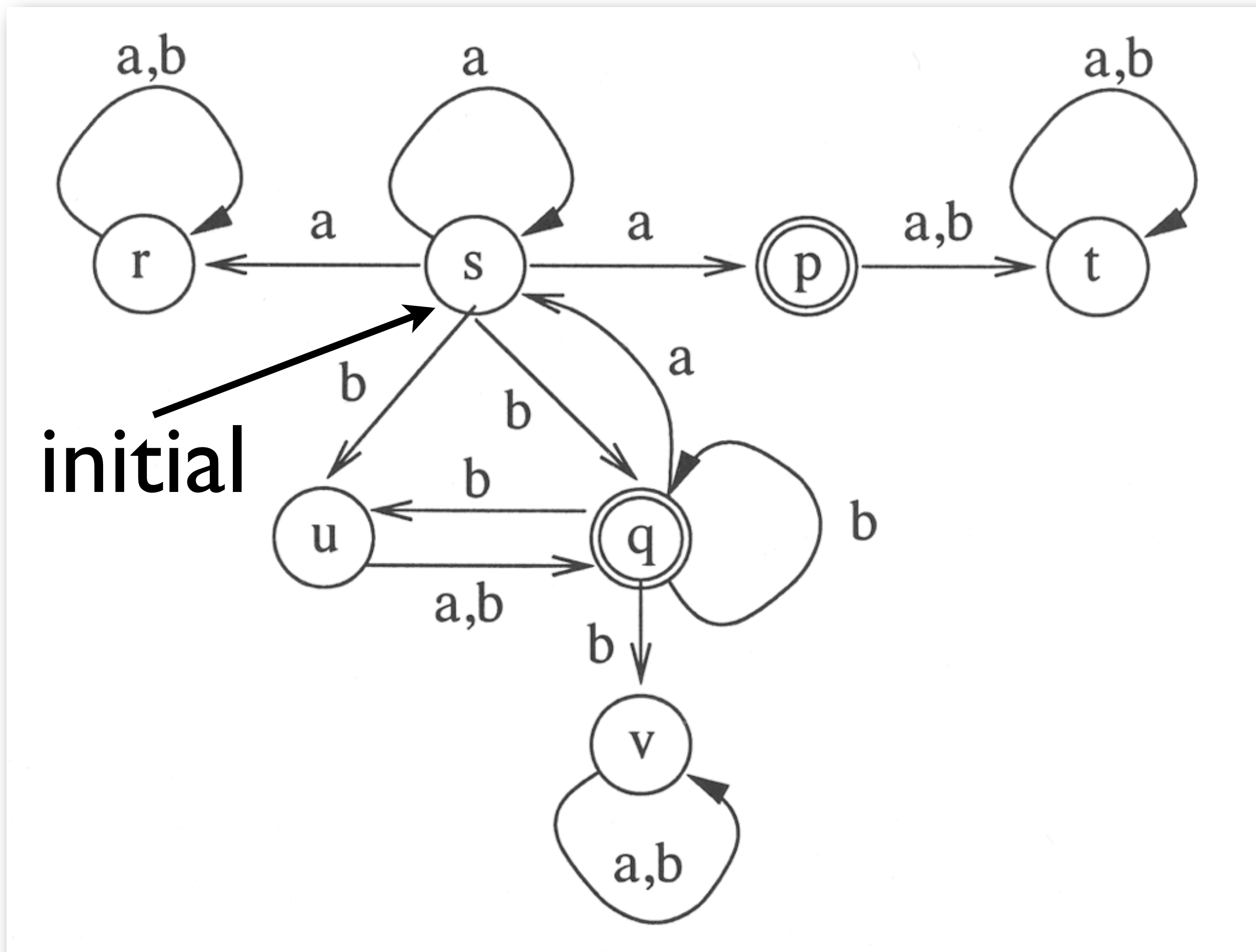
- $L(A_d) \subseteq L(A)$: Suppose that after finitely many steps, some component of the states visit final states infinitely often and is never equal to $\perp$, then it corresponds to a copy of A_m which is never merged (otherwise $\perp$ will occur), and therefore this copy accepts the word, i.e. $L(A_d) \subseteq L(A_m) \subseteq L(A)$.

Outline

- Introduction
- Preliminaries
- 2-Exponential Determinization Procedure
- Optimal Determinization Procedure by Safra

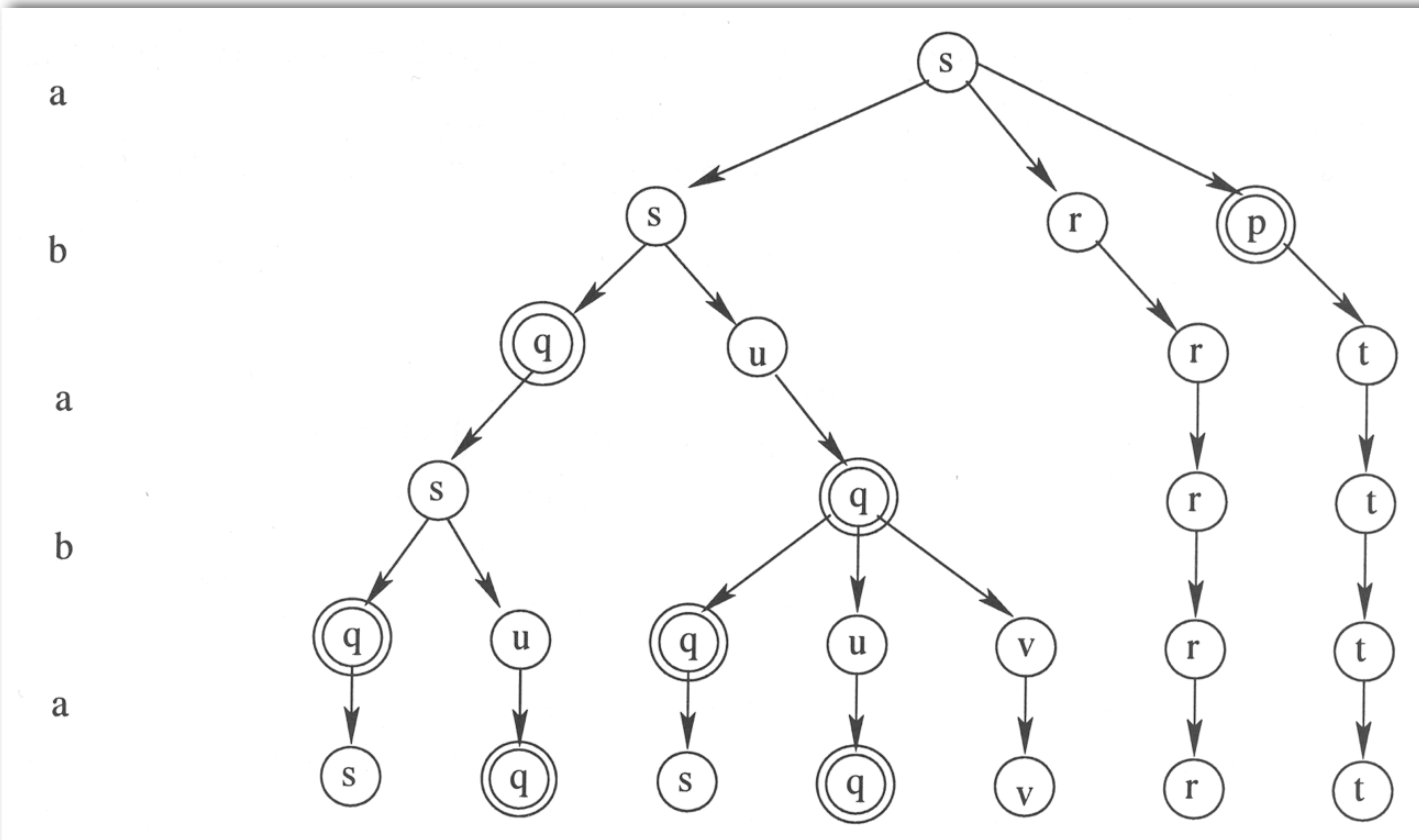
Running Example

A

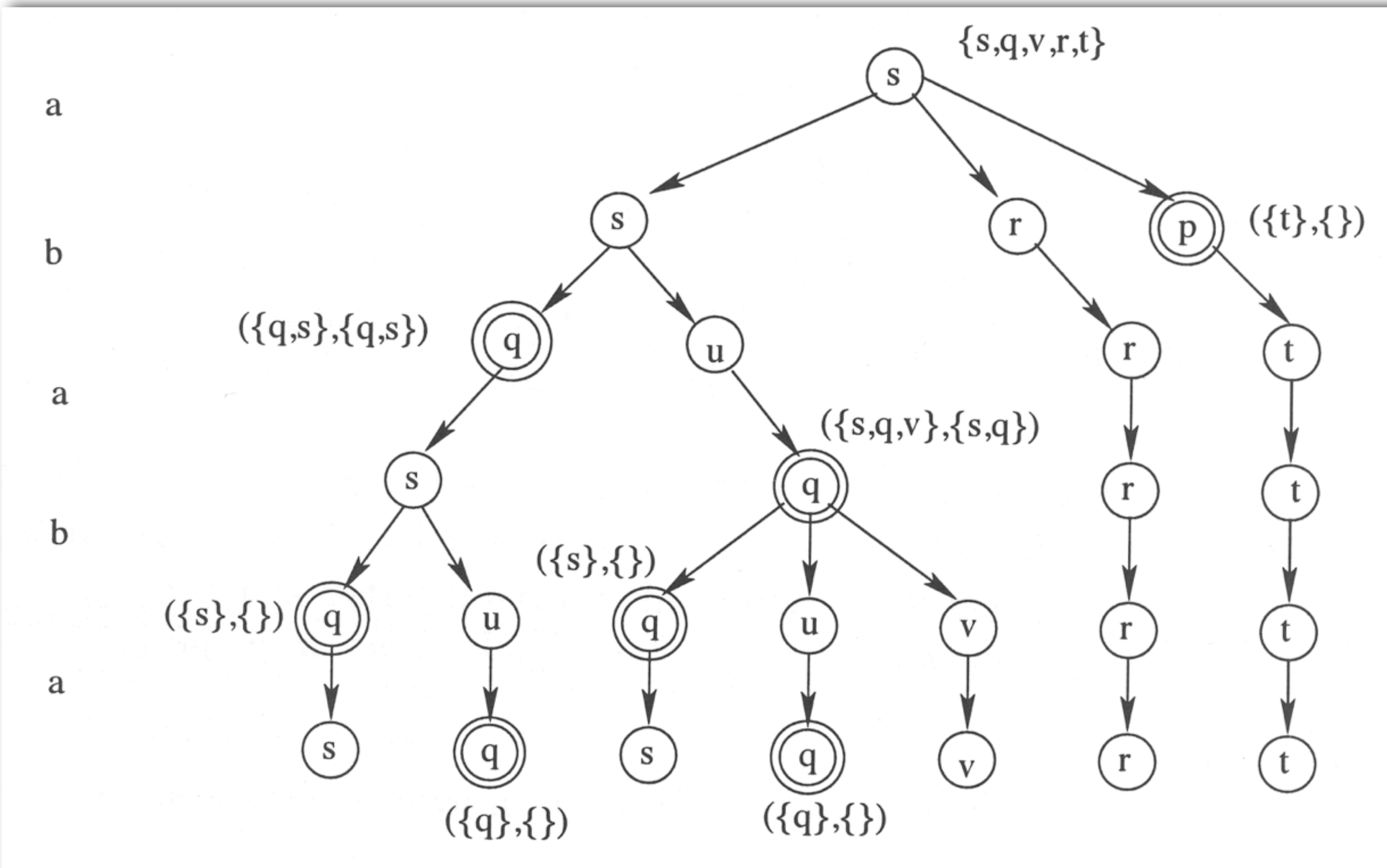


L(*A*) = “infinitely many *b*”

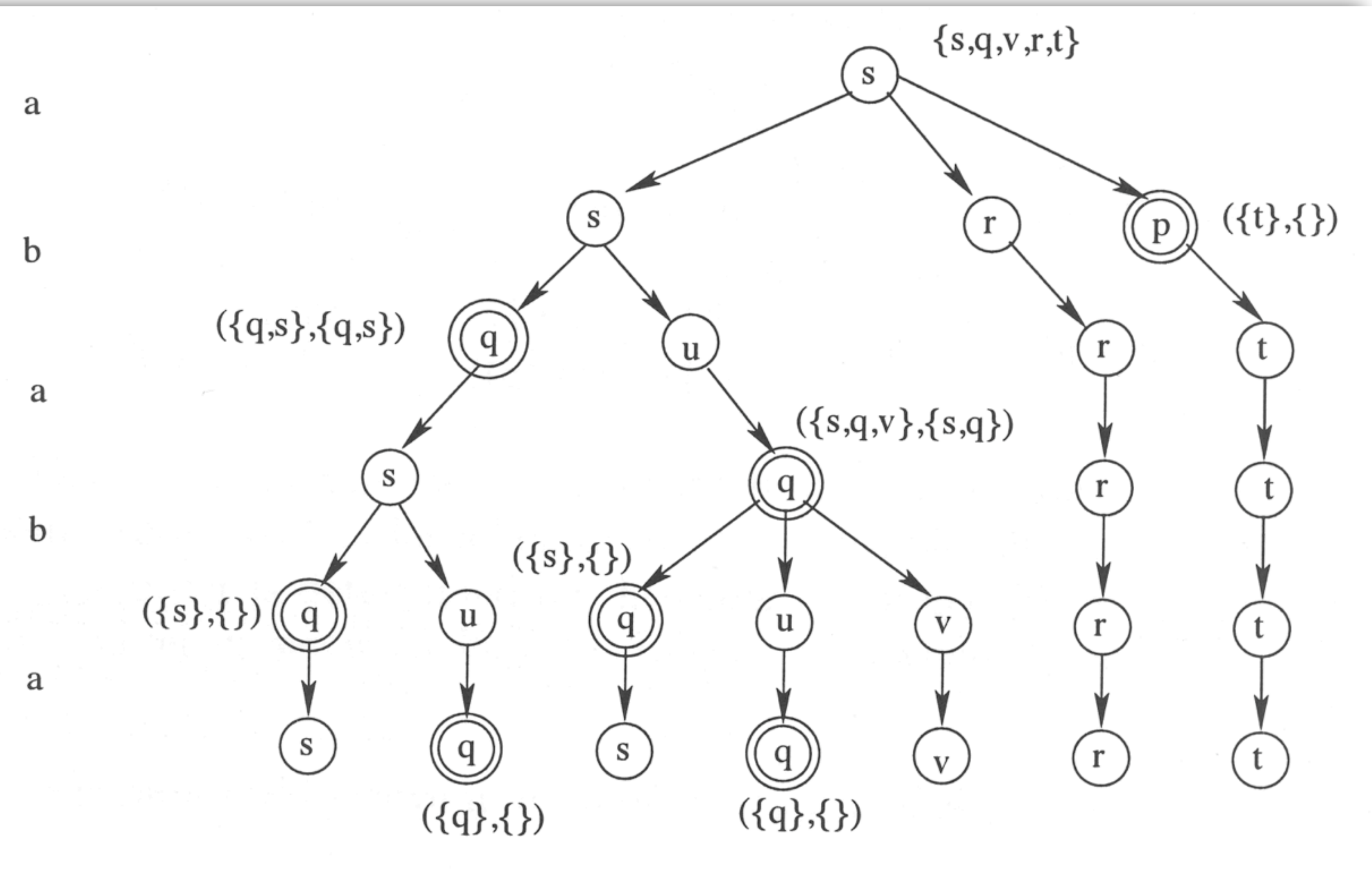
Run Tree on abababa...

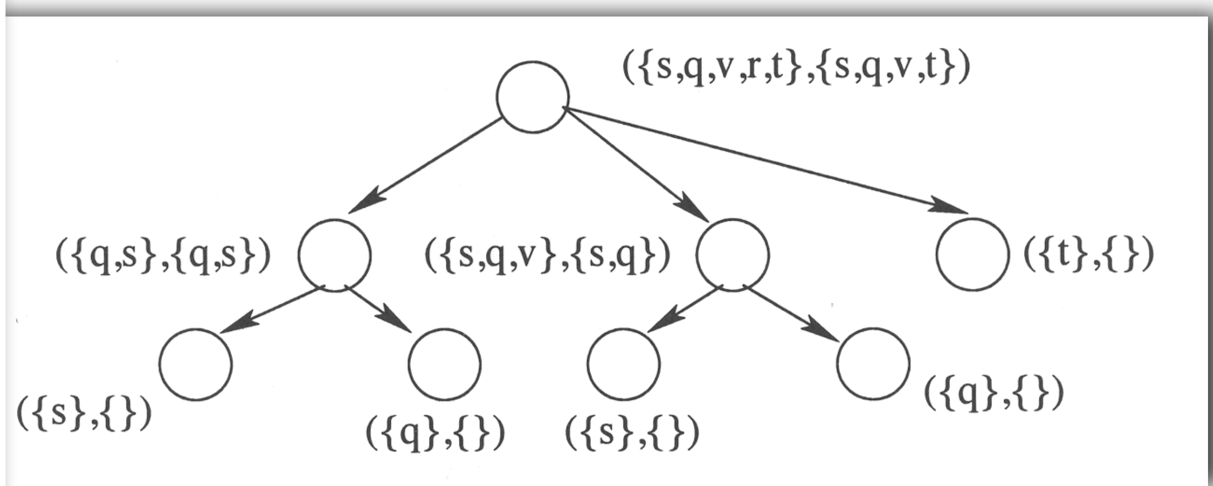


Run Tree on abababa...

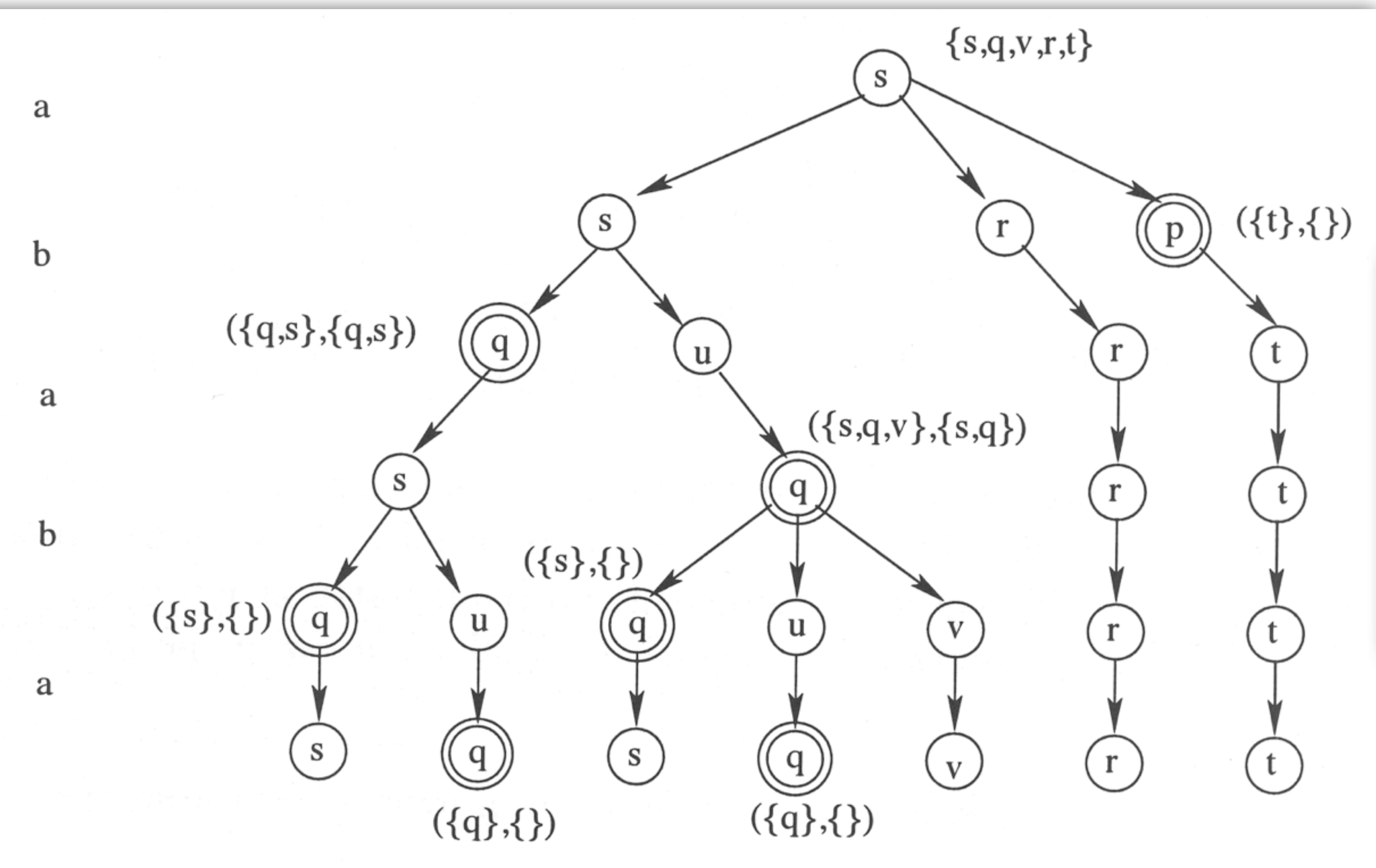


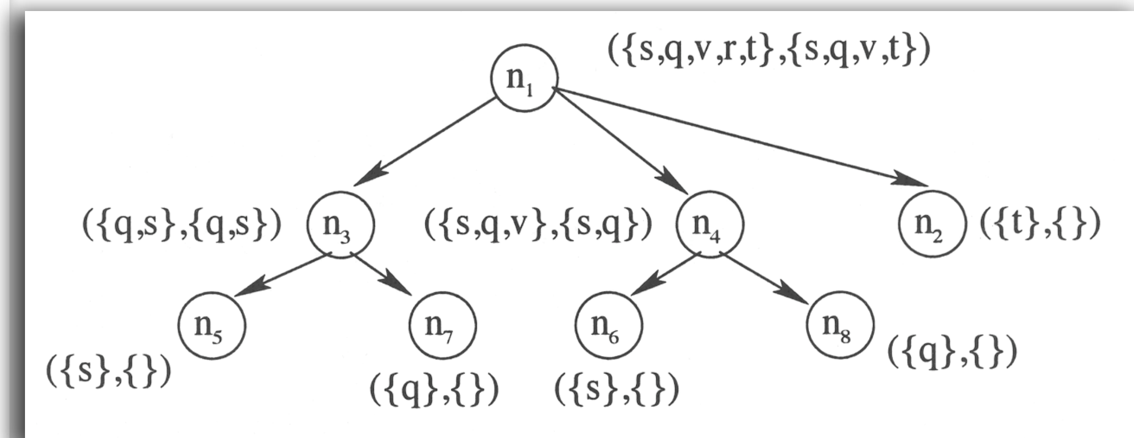
Maintain ancestor-descendant relationship between copies





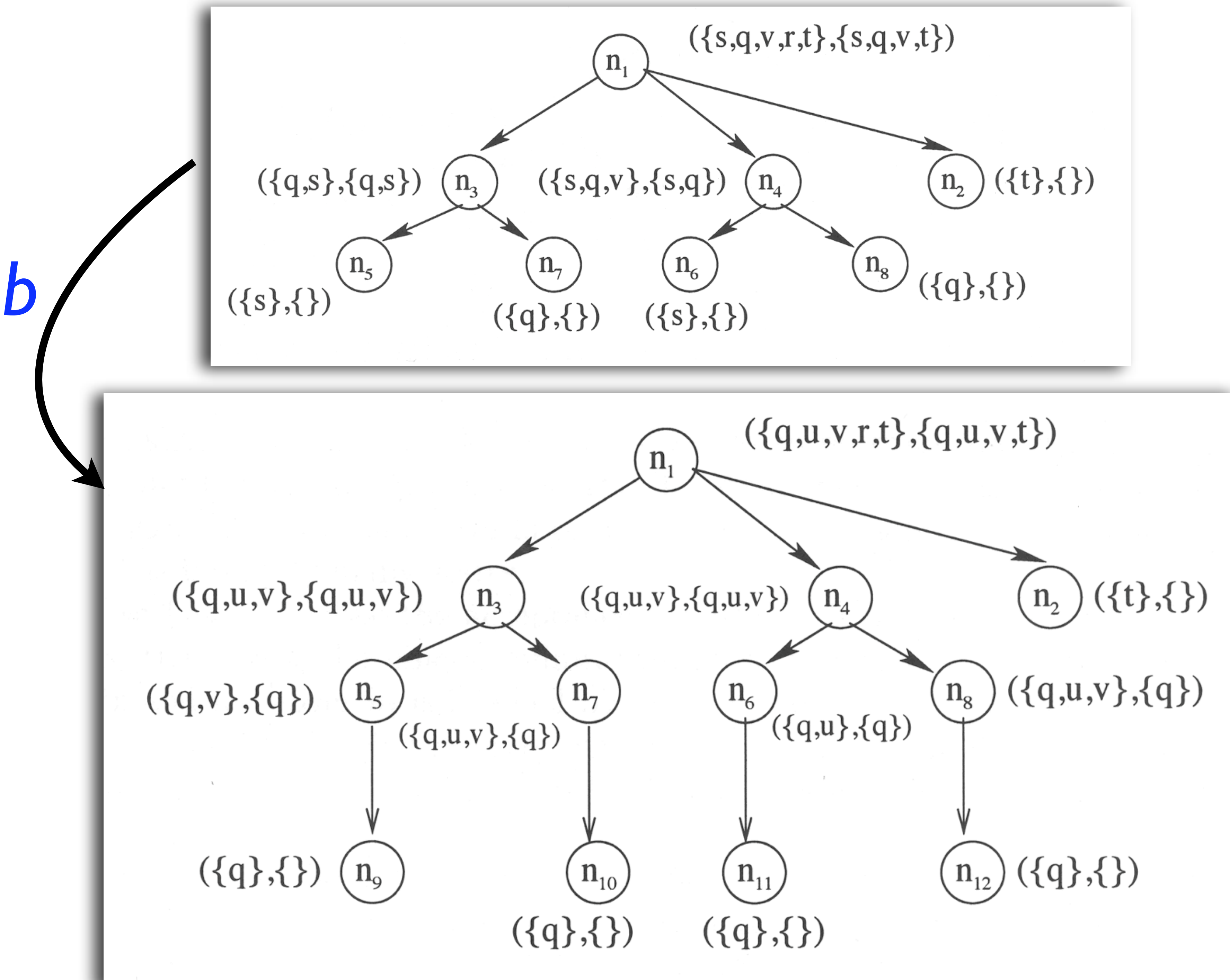
Use node names to refer to copies



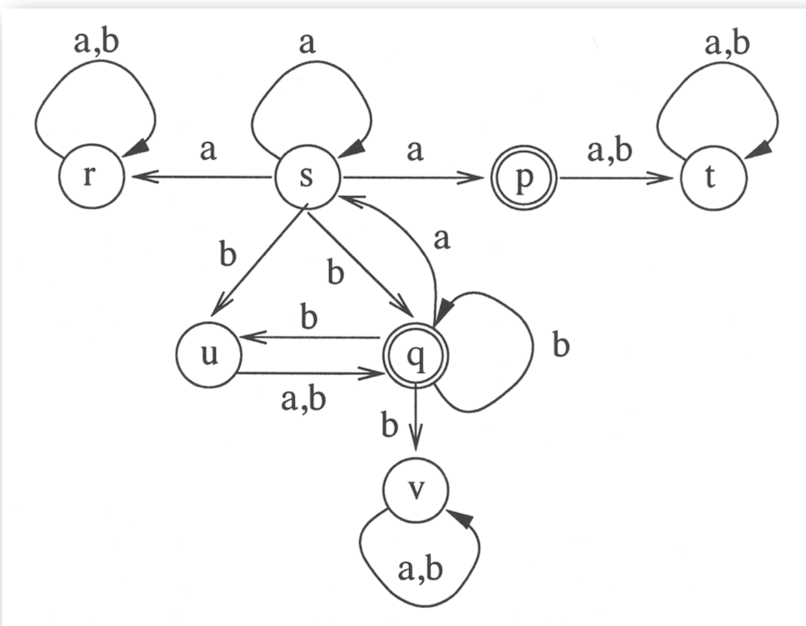


The numbering respects the order the copies were forked

After reading a *b*



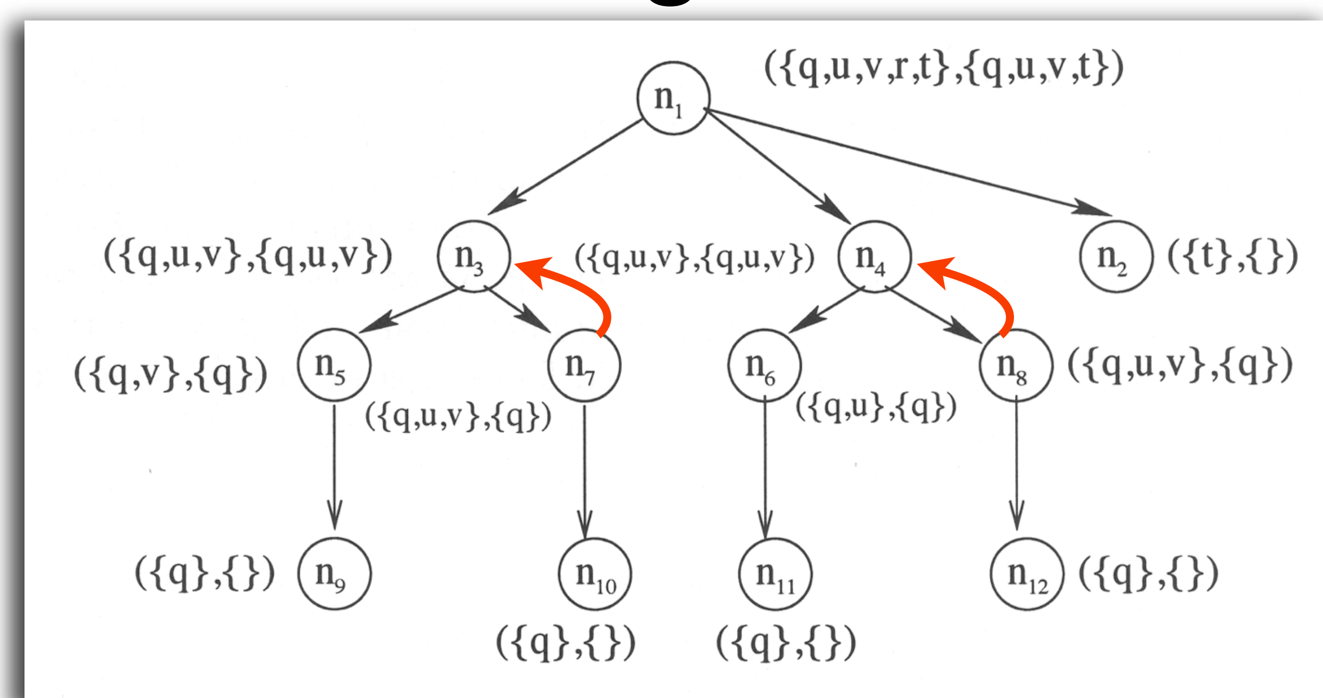
A

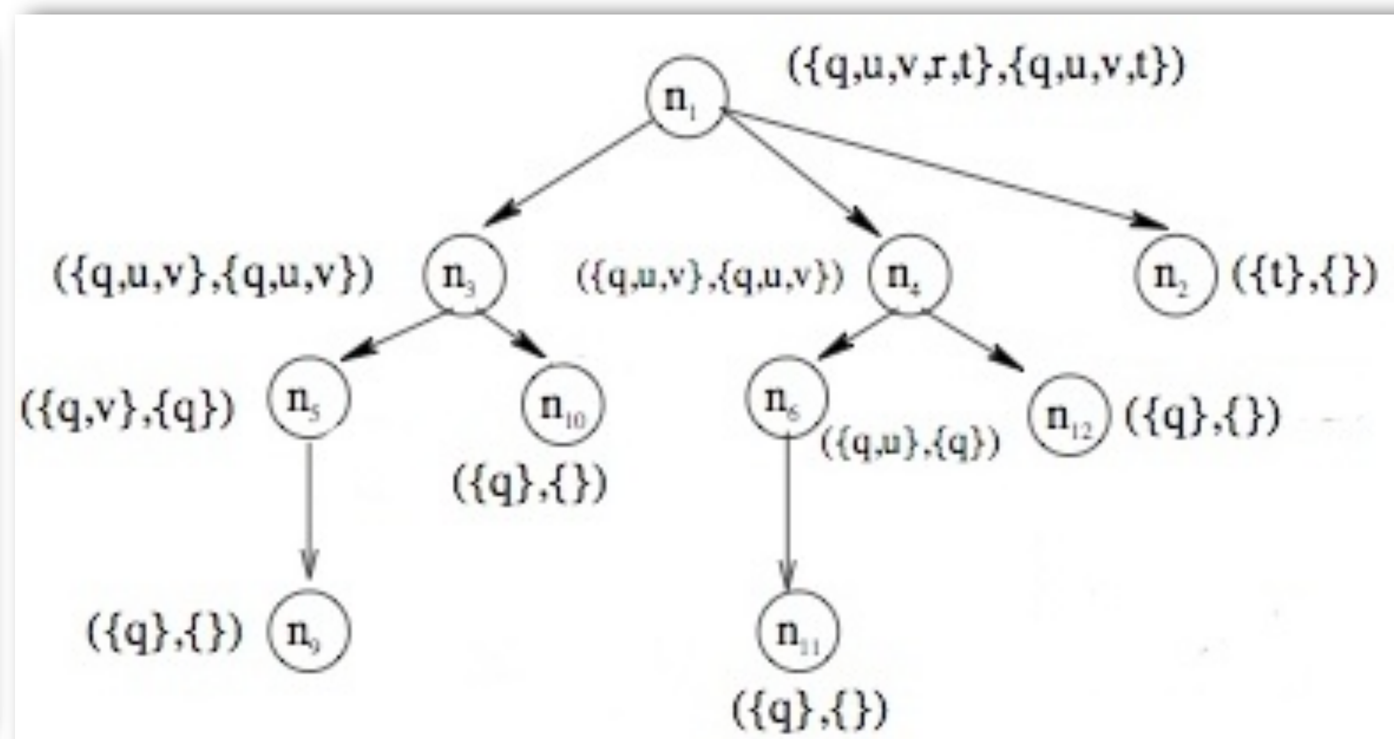


Fork new copies for each final state of *A*

Bound the height by |*Q*|

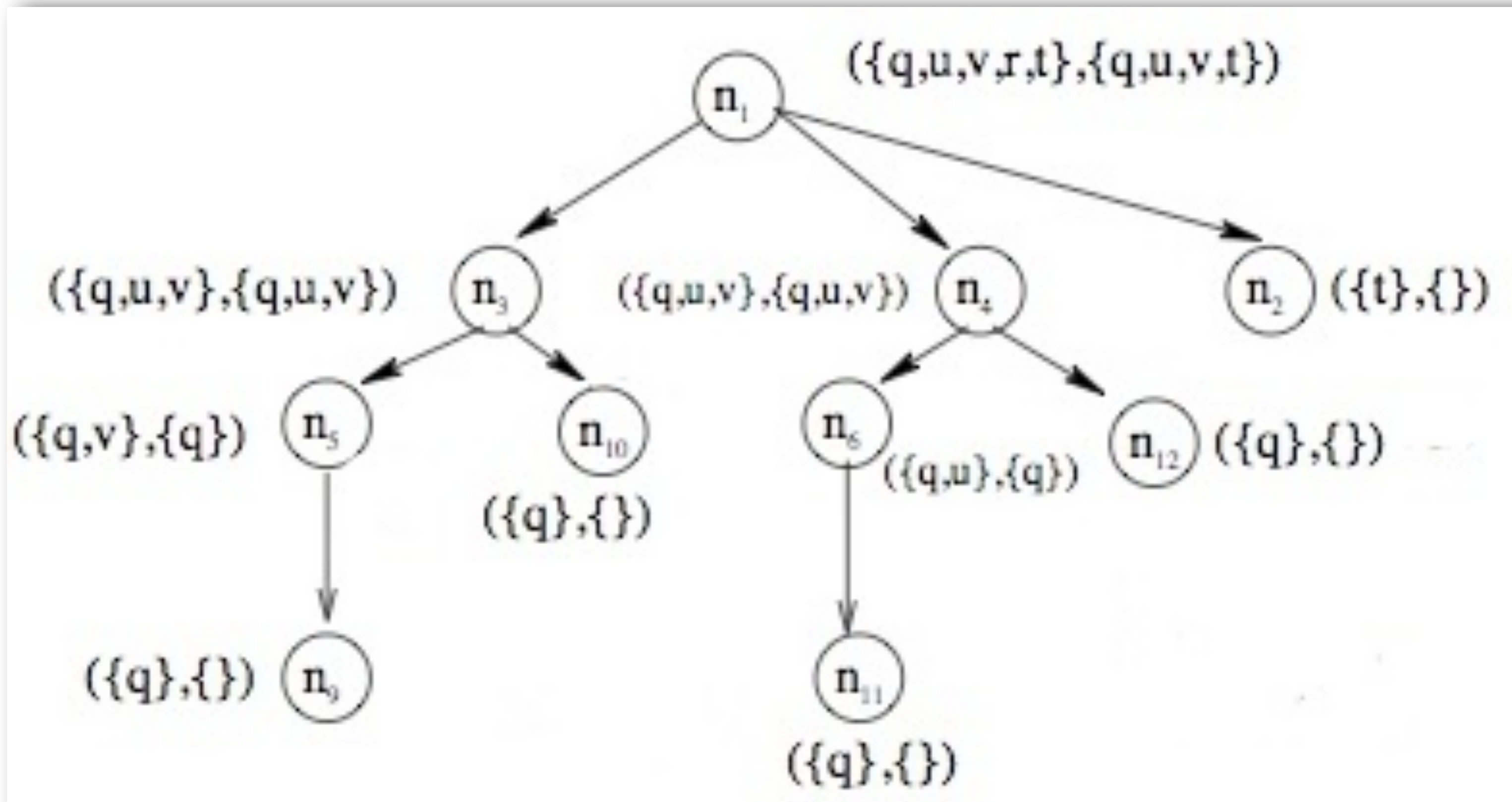
- Let n_j be the child of n_i
- Merge n_i and n_j if n_i is labeled (X,Y) and n_j is labeled (X,Z) (keep n_i)
- i.e. the two copies n_i and n_j of A_m have merged



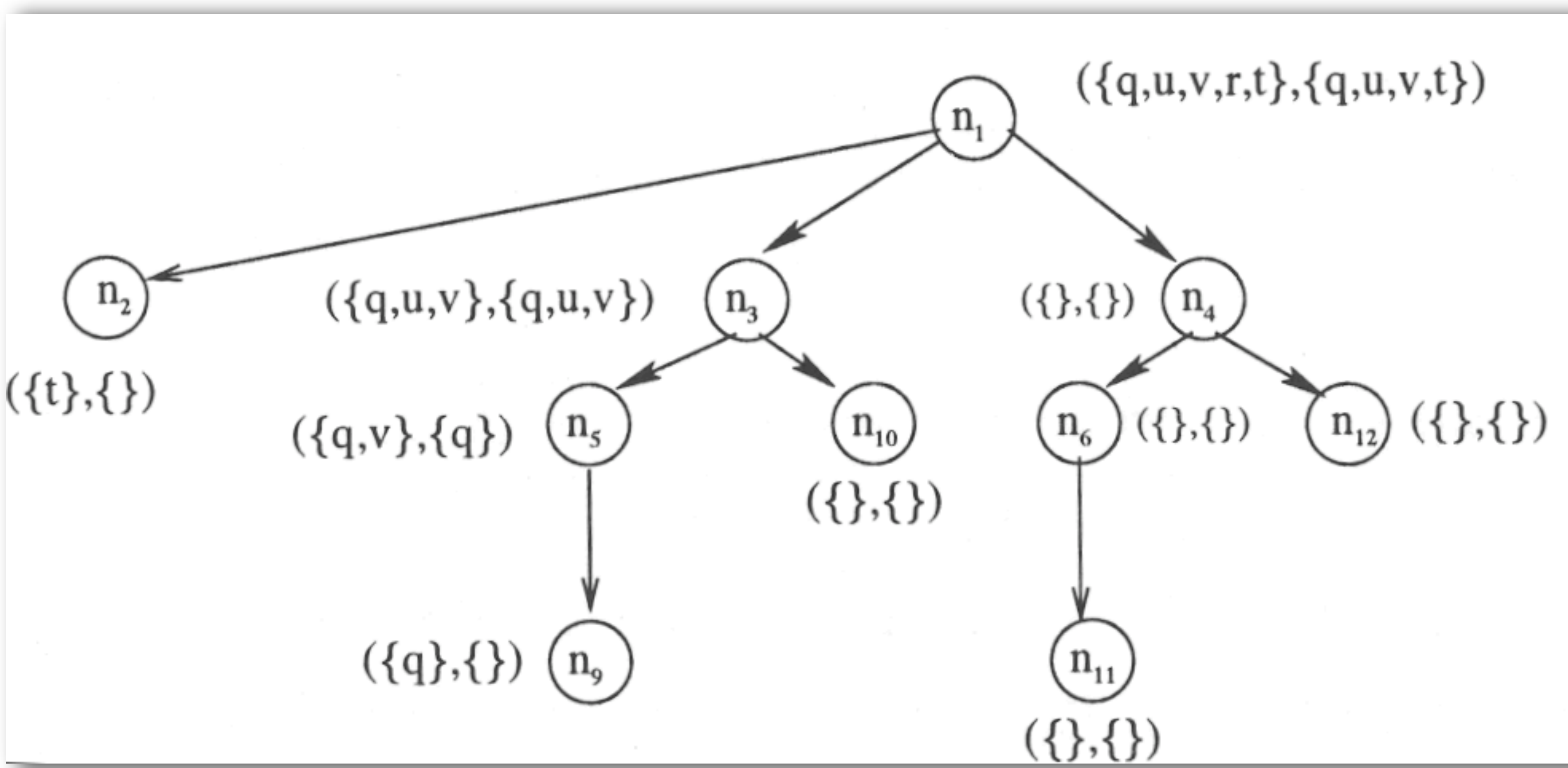


⇒ the height of the tree is bounded by |Q|

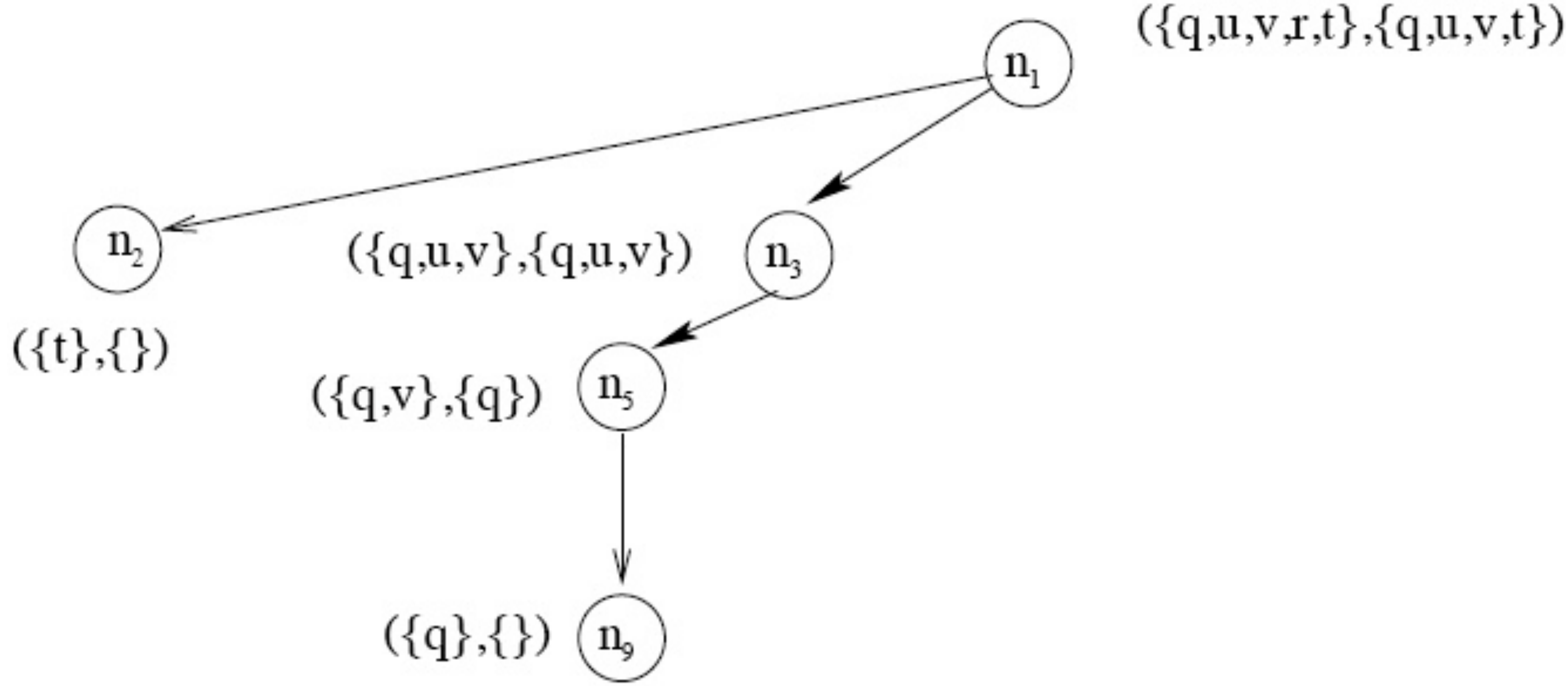
Bound The Width



For each state, keep the left-most path that contains it



Bound The Width



The labels of the children of a node are pairwise disjoint

⇒ a node as at most |Q| children

Transition Function

- 1/ Update all labels (X,Y) by $\delta_m((X,Y),a)$
- 2/ For all paths and all final states q_f, find the youngest occurence of q_f and "fork a copy of A_m", i.e. create a new right-most child labeled $\{q_f\}$. Name this node by the smallest free node name.
- 3/ Merge descendant nodes with the same first component (keep the oldest copy)
- 4/ For all state q, remove q from all paths that contain it except the left-most

Acceptance Condition

- First: at each level we refine the label of the parent with disjoint labels
- we add at most |F| nodes
- $\Rightarrow$ n = 3|Q| node names are sufficient
- Rabin AC: $(F_i, I_i)_{1 \leq i \leq n}$
- F_i = set of trees where node n_i does not occur
- I_i = set of trees where node n_i occurs and is labeled by an accepting state of A_m

Correctness (Sketch)

- *L(Safra)*$\subseteq$*L(A)*: if after a finite prefix, a node n_i always appears in the Safra states, and is labeled infinitely often by a final state of A_m, it means that the **same copy** of A_m visits a final state infinitely often. Therefore the word is accepted.

Correctness (Sketch)

- $L(A) \subseteq L(Safra)$: let $w \in L(A)$ and consider an accepting run ρ on w in its run-tree, and the run ρ' of Safra on w
- **Claim 1**: after a certain point, there is a node v that appears in all "Safra" states of ρ'
- **Claim 2**: this node is labeled infinitely often by final state of A_m

2nd component labels are useless

- The second component was needed to check if all current states visited an accepting state
- This is now the case when a merge occurs
- We can forget the 2nd component, but mark a label with a special symbol if its direct descendant have been merged

The End