Tree Automata and Tree Logics

Mikołaj Bojańczyk
(Warsaw)

1 Tree automata

What is a Tree Automaton?

Decision Problems

1 Tree automata
What is a Tree Automaton?
Decision Problems

2 Logic
Logic for Words
Logic for Trees
Transitive Closure Logic

1 Tree automata
What is a Tree Automaton?
Decision Problems

2 Logic
Logic for Words
Logic for Trees
Transitive Closure Logic

3 Temporal Logics
Temporal Logic for Words
Temporal Logic for Trees
XPath

1 Tree automata

- What is a Tree Automaton?
- Decision Problems

2 Logic

- Logic for Words
- Logic for Trees
- Transitive Closure Logic

3 Temporal Logics

- Temporal Logic for Words
- Temporal Logic for Trees
- XPath

4 Tree-Walking Automata, 1

- Tree-Walking Automata
- Expressive Power
- Pebble Automata

1 Tree automata

What is a Tree Automaton?
Decision Problems

2 Logic

Logic for Words
Logic for Trees
Transitive Closure Logic

3 Temporal Logics

Temporal Logic for Words
Temporal Logic for Trees
XPath

4 Tree-Walking Automata, 1

Tree-Walking Automata
Expressive Power
Pebble Automata

5 Tree-Walking Automata, 2

Tree-Walking Automata Cannot Be Determinized

1 Tree automata

- What is a Tree Automaton?
- Decision Problems

2 Logic

- Logic for Words
- Logic for Trees
- Transitive Closure Logic

3 Temporal Logics

- Temporal Logic for Words
- Temporal Logic for Trees
- XPath

4 Tree-Walking Automata, 1

- Tree-Walking Automata
- Expressive Power
- Pebble Automata

5 Tree-Walking Automata, 2

- Tree-Walking Automata Cannot Be Determinized

What is a Tree Automaton?

–definition and examples

–determinism, bottom-up vs top-down

–minimization

–closure properties

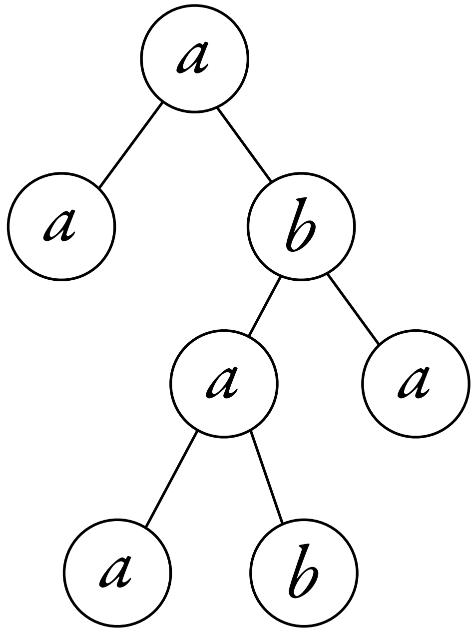
Decision Problems

–emptiness

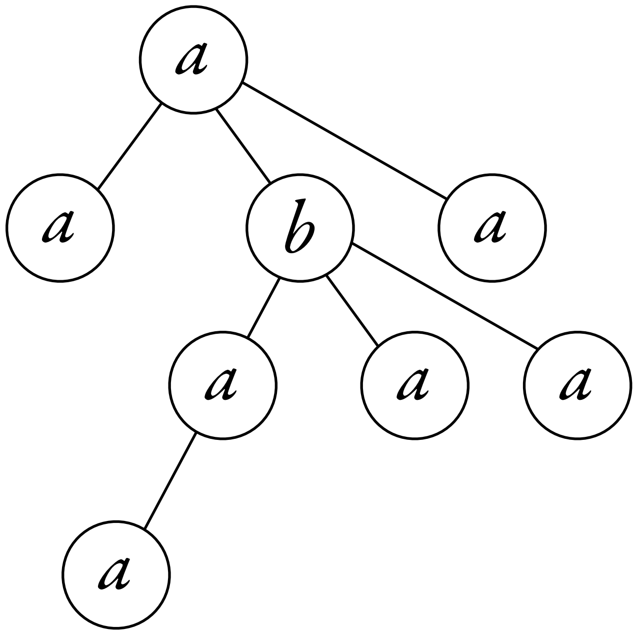
–membership

–universality

Trees are finite and labeled.



binary tree:
each node has 0 or 2 children



unranked tree

We will use automata to express properties such as...

We will use automata to express properties such as...

“there is at least one a in the tree”

We will use automata to express properties such as...

“there is at least one *a* in the tree”

“every node with label *a* has at most one *b* child”

We will use automata to express properties such as...

“there is at least one a in the tree”

“every node with label a has at most one b child”

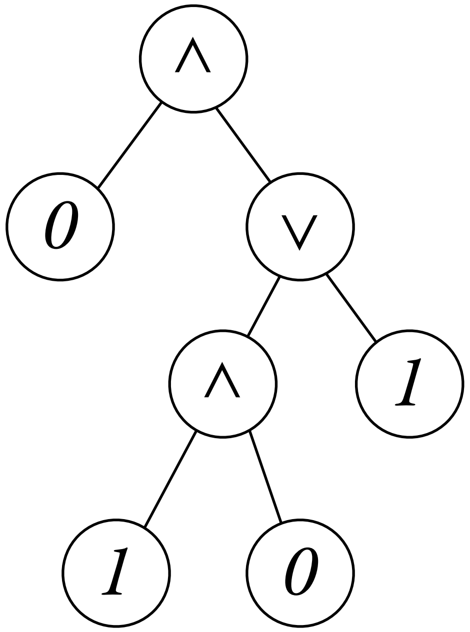
“even number of nodes”

We will use automata to express properties such as...

“there is at least one *a* in the tree”

“every node with label *a* has at most one *b* child”

“even number of nodes”



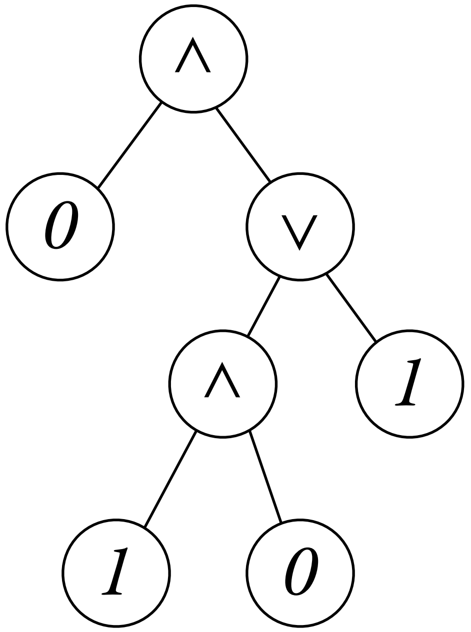
“boolean expressions with value *1*”

We will use automata to express properties such as...

"there is at least one *a* in the tree"

"every node with label *a* has at most one *b* child"

"even number of nodes"



"boolean expressions with value *1*"

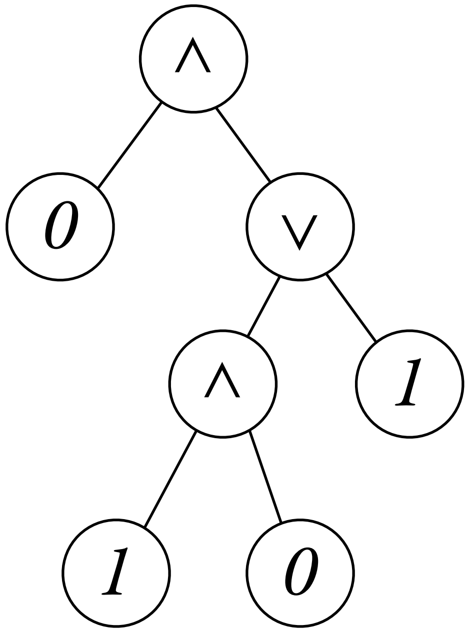
...but not properties such as

We will use automata to express properties such as...

"there is at least one a in the tree"

"every node with label a has at most one b child"

"even number of nodes"



"boolean expressions with value *1*"

...but not properties such as

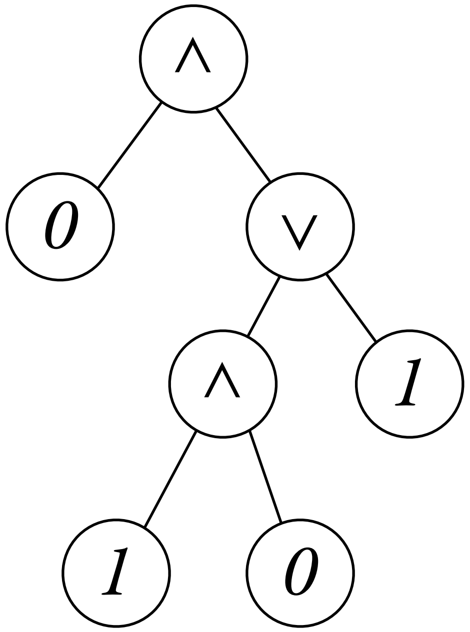
"number of a's = number of b's"

We will use automata to express properties such as...

"there is at least one *a* in the tree"

"every node with label *a* has at most one *b* child"

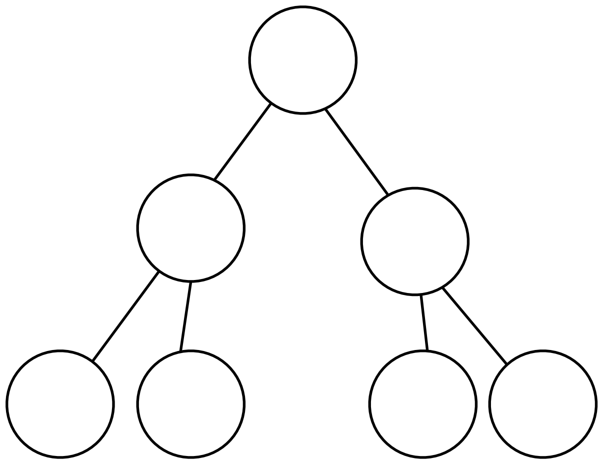
"even number of nodes"



"boolean expressions with value *1*"

...but not properties such as

"number of *a*'s = number of *b*'s"



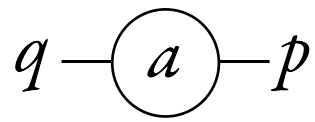
"balanced trees"

A word automaton

This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

A word automaton

automaton transitions

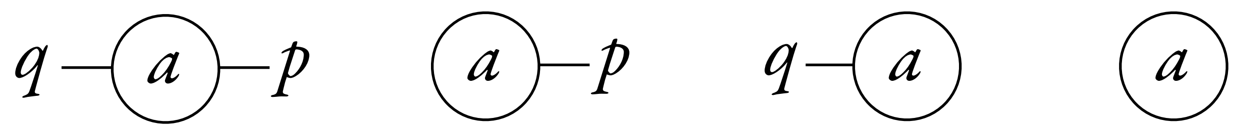


This definition best for nondeterministic automata.

No difference between bottom-up and top-down.

A word automaton

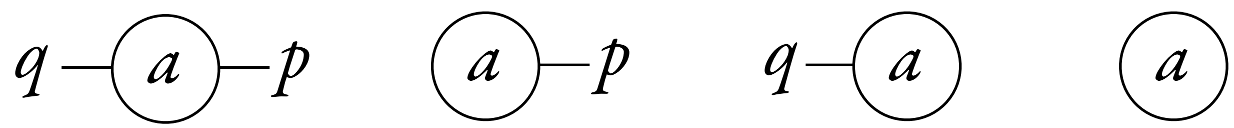
automaton transitions



This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

A word automaton

automaton transitions

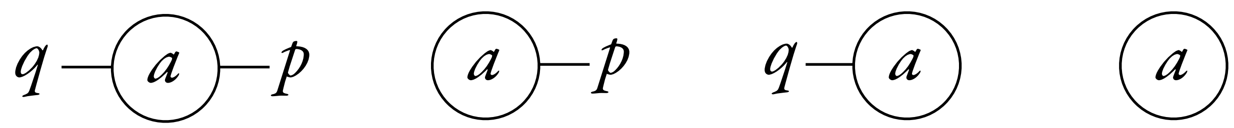


a run is a labeling of edges with states consistent with the transitions

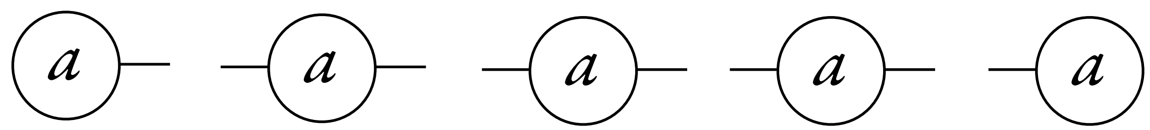
This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

A word automaton

automaton transitions



a run is a labeling of edges with states consistent with the transitions

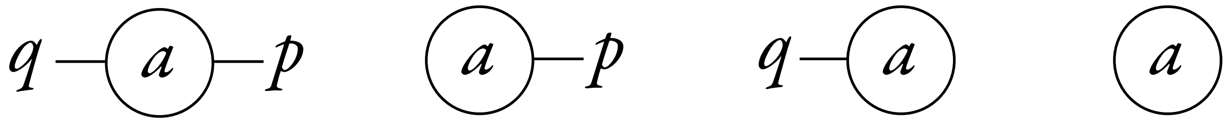


This definition best for nondeterministic automata.

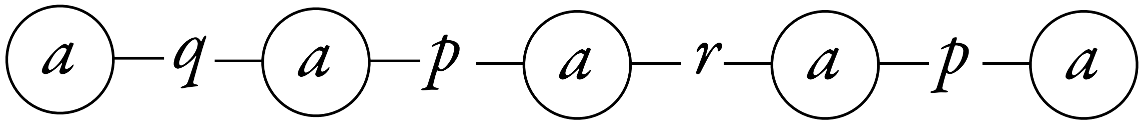
No difference between bottom-up and top-down.

A word automaton

automaton transitions



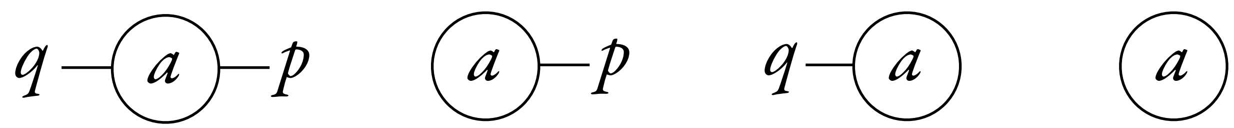
a run is a labeling of edges with states consistent with the transitions



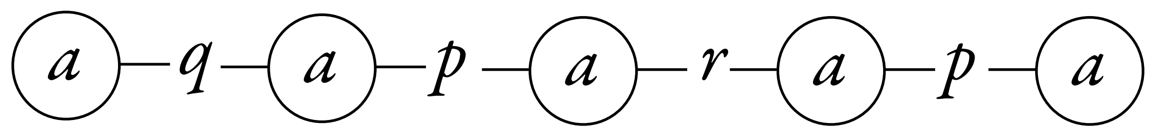
This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

A word automaton

automaton transitions

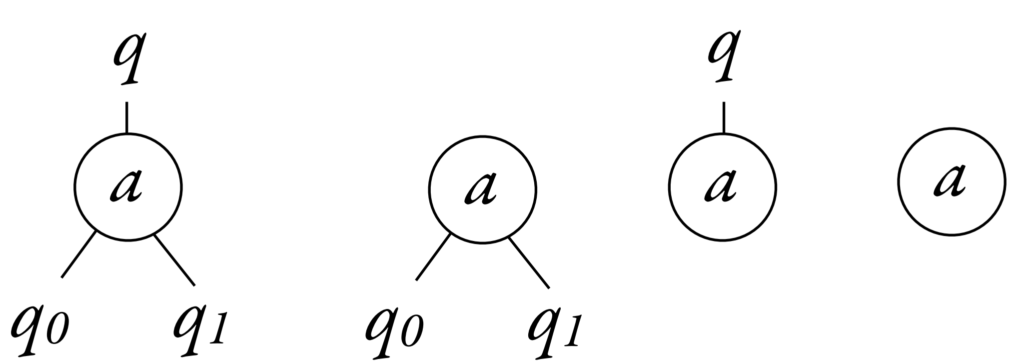


a run is a labeling of edges with states consistent with the transitions



A tree automaton

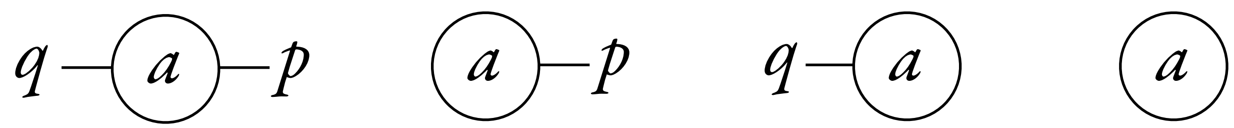
automaton transitions



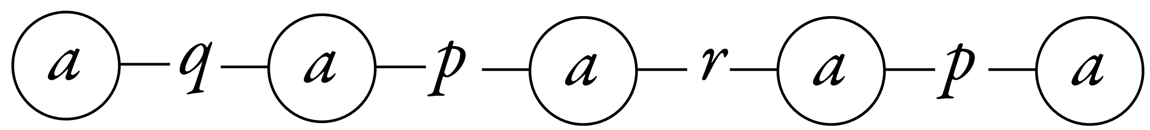
This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

A word automaton

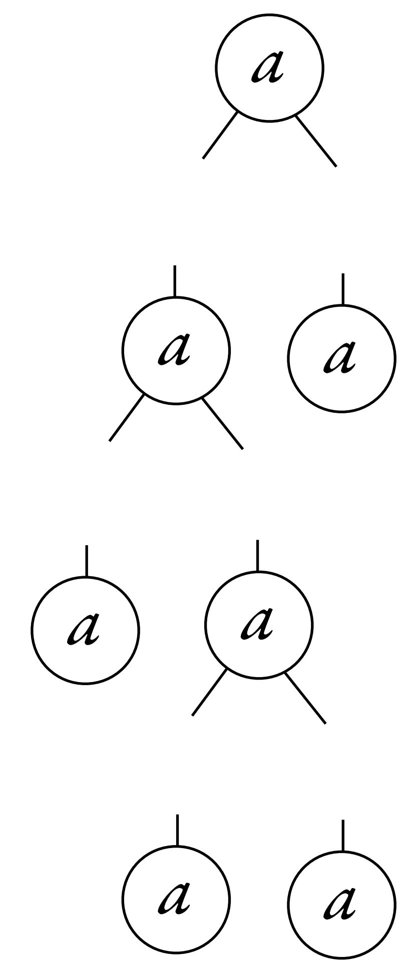
automaton transitions



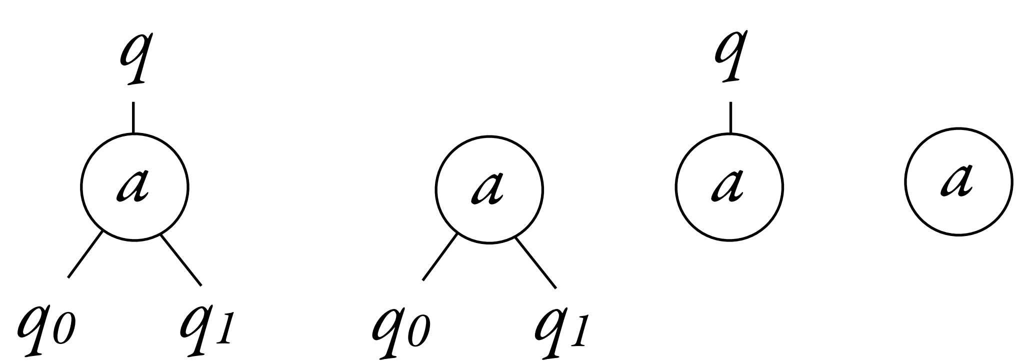
a run is a labeling of edges with states consistent with the transitions



A tree automaton



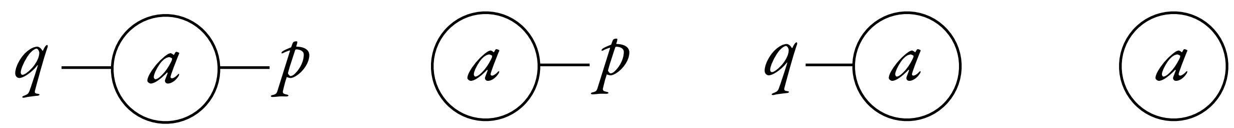
automaton transitions



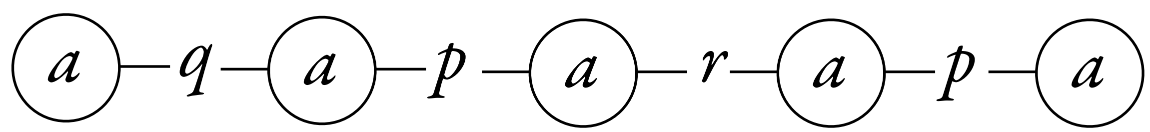
This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

A word automaton

automaton transitions

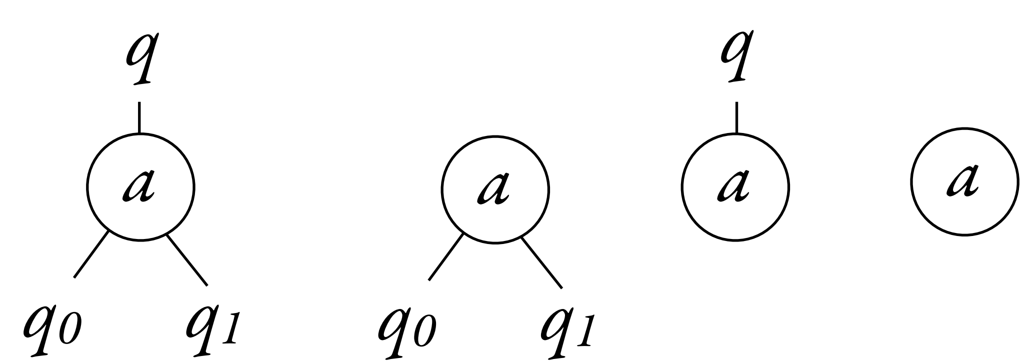


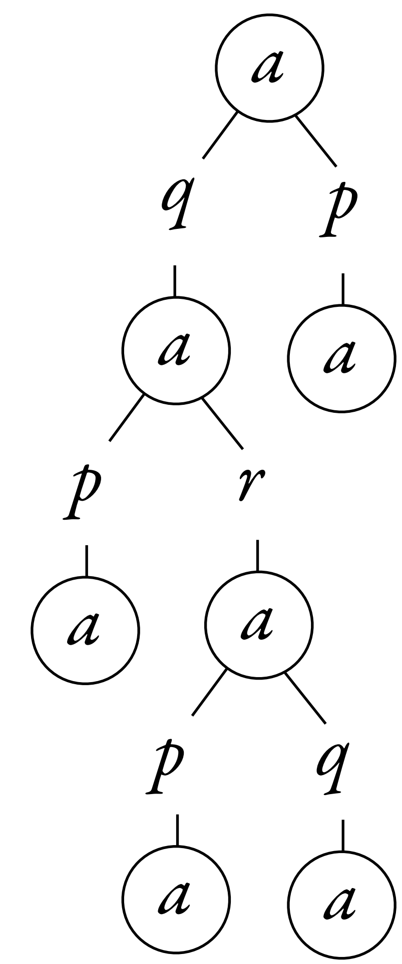
a run is a labeling of edges with states consistent with the transitions



A tree automaton

automaton transitions





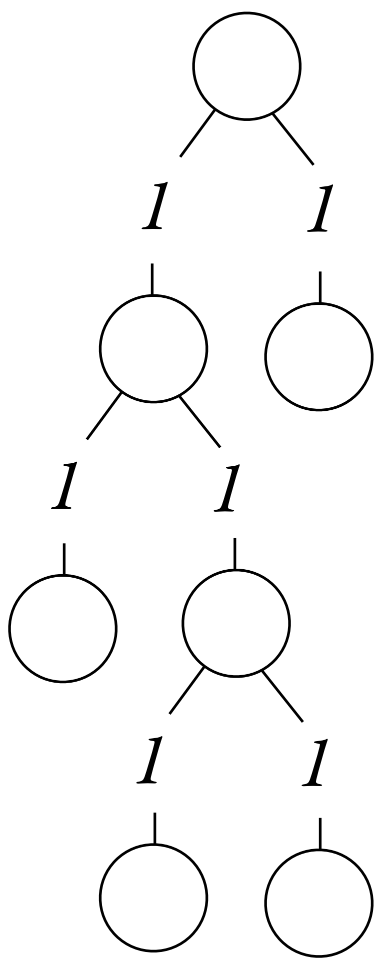
This definition best for nondeterministic automata.
No difference between bottom-up and top-down.

Examples

Examples

"even number of nodes"

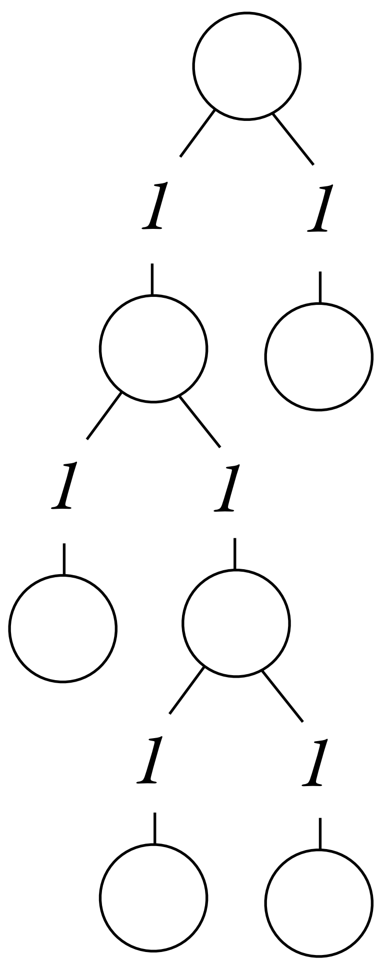
states: 0,1



Examples

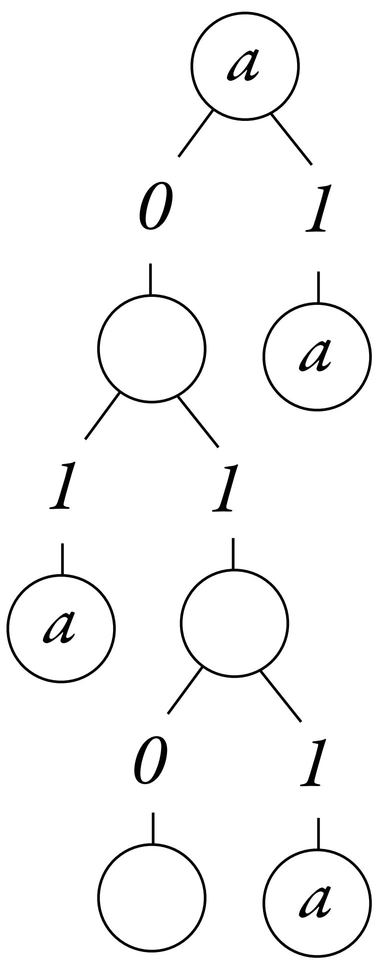
“even number of nodes”

states: 0,1



“even number of a’s”

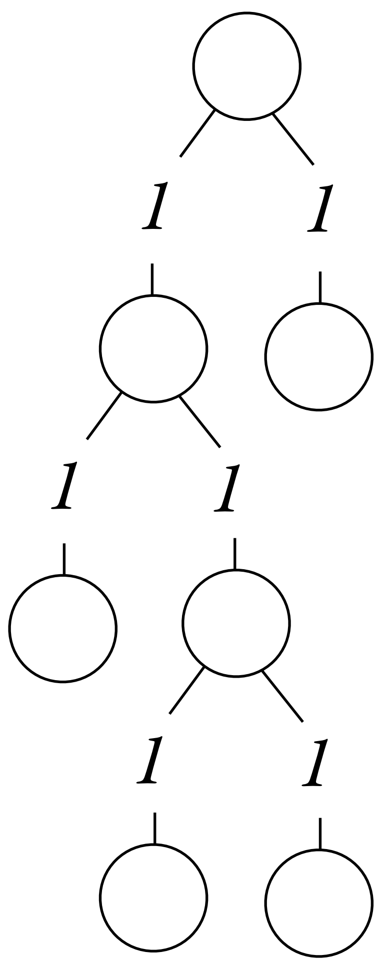
states: 0,1



Examples

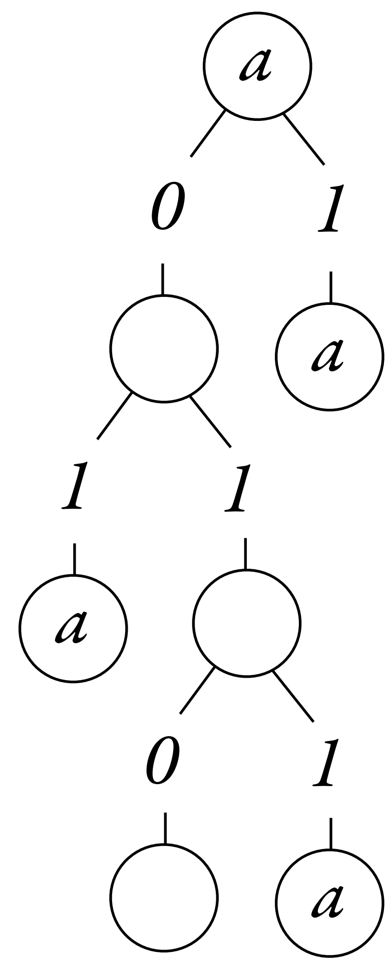
“ even number of nodes ”

states: 0,1



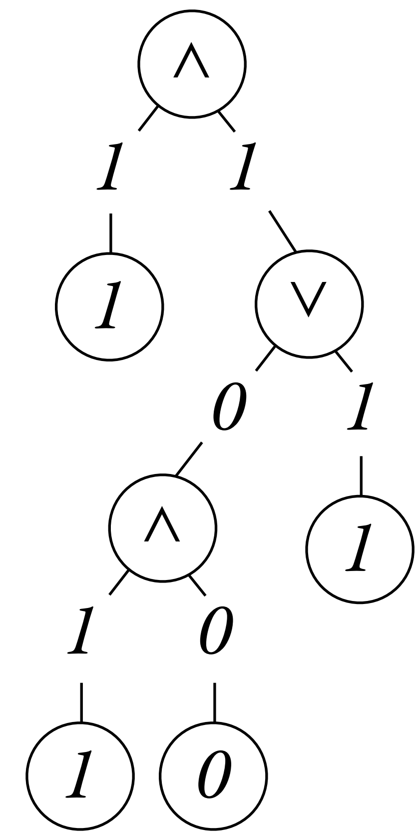
“ even number of *a*’s ”

states: 0,1



“ boolean expressions with value *1* ”

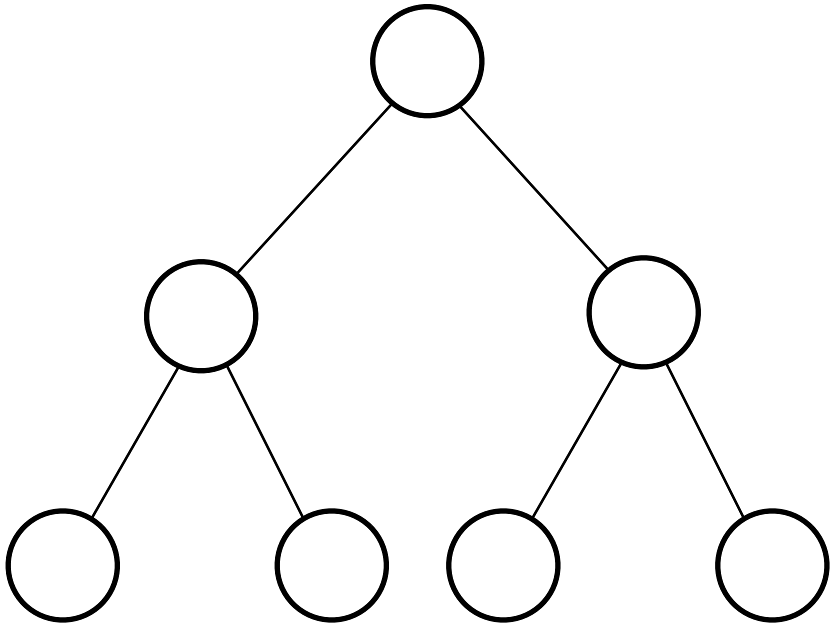
states: 0,1



Deterministic automata

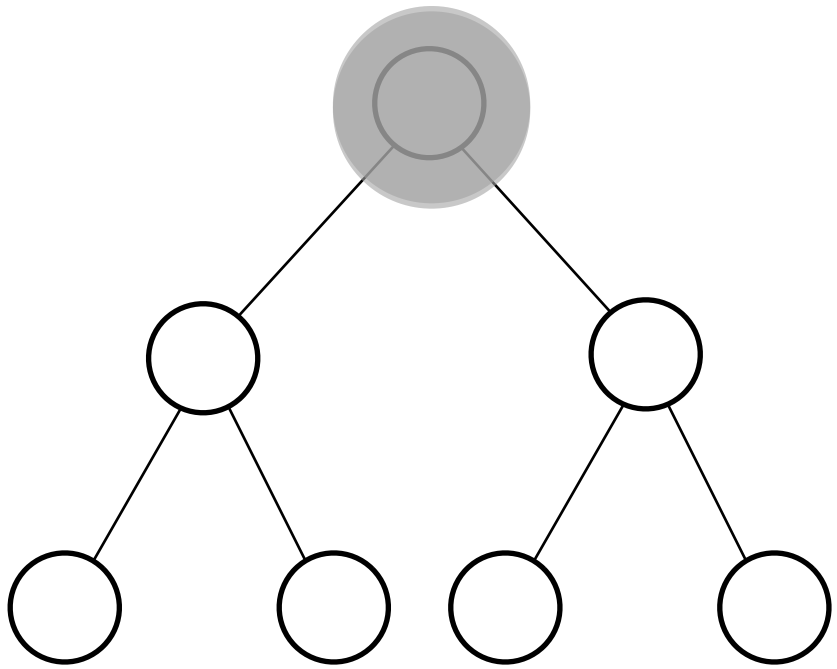
Deterministic automata

Top-down



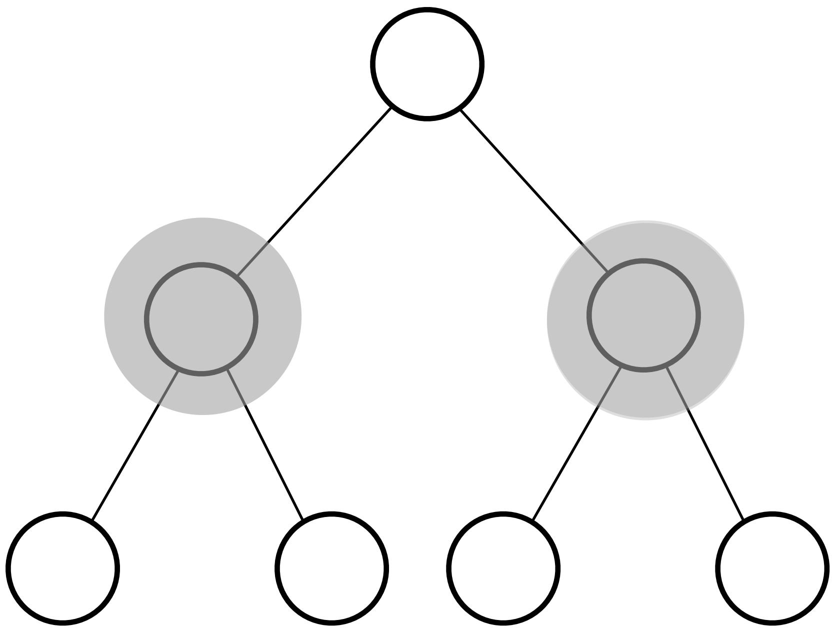
Deterministic automata

Top-down



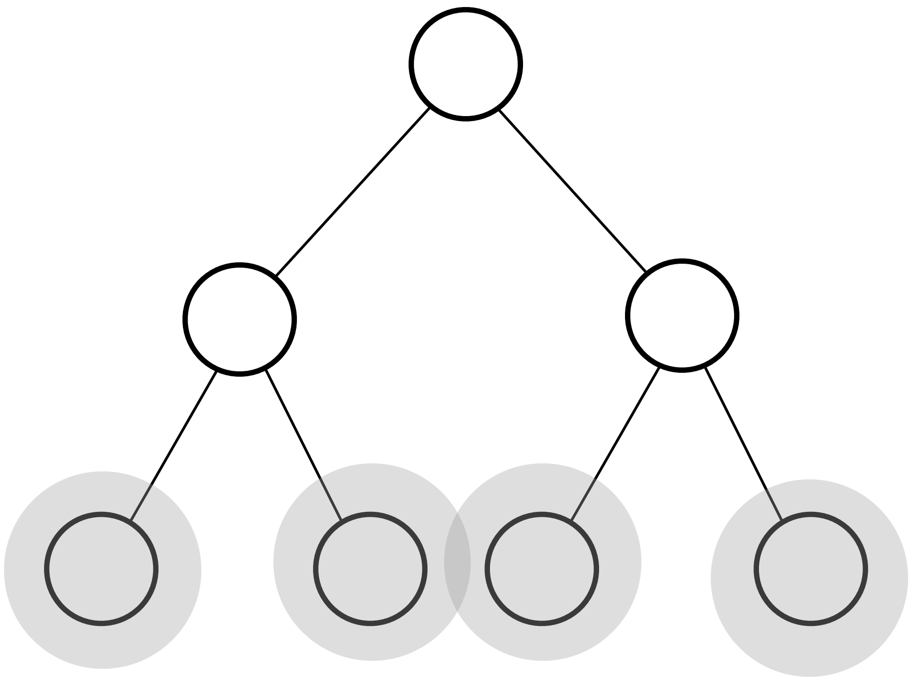
Deterministic automata

Top-down



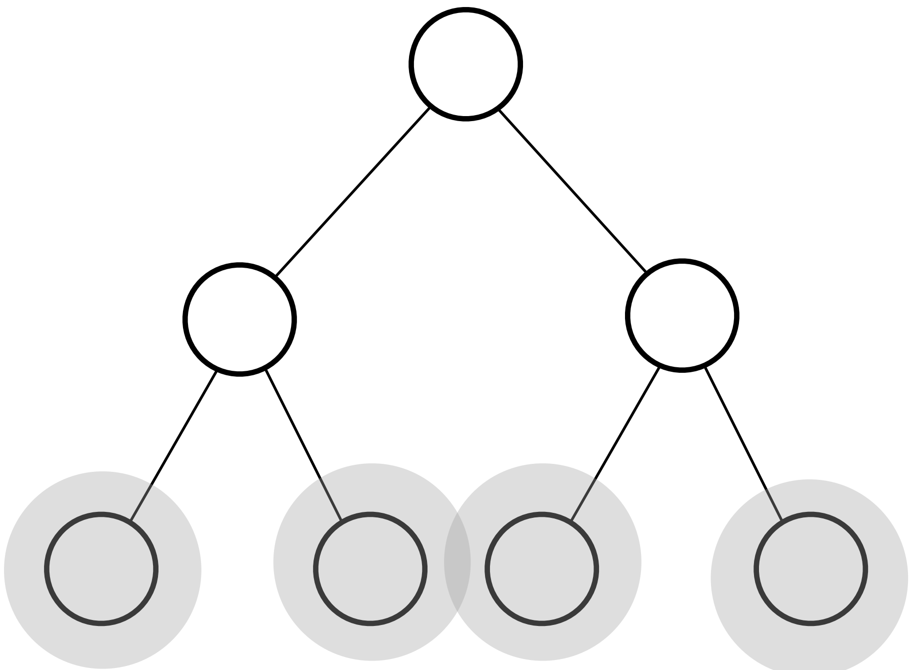
Deterministic automata

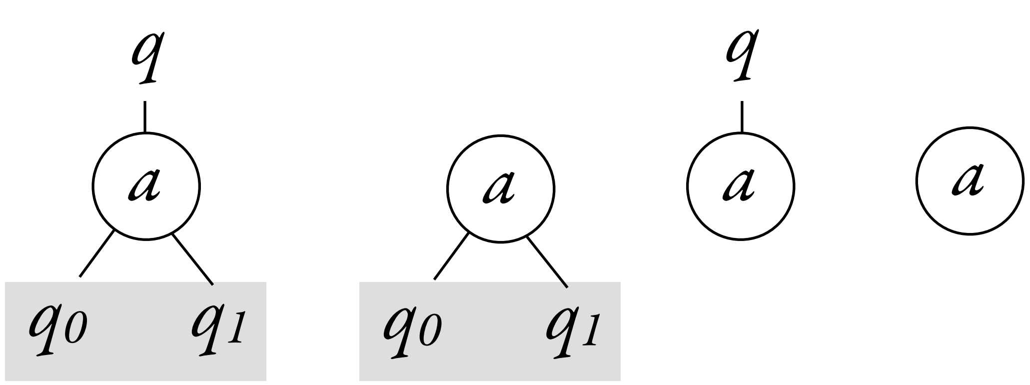
Top-down



Deterministic automata

Top-down

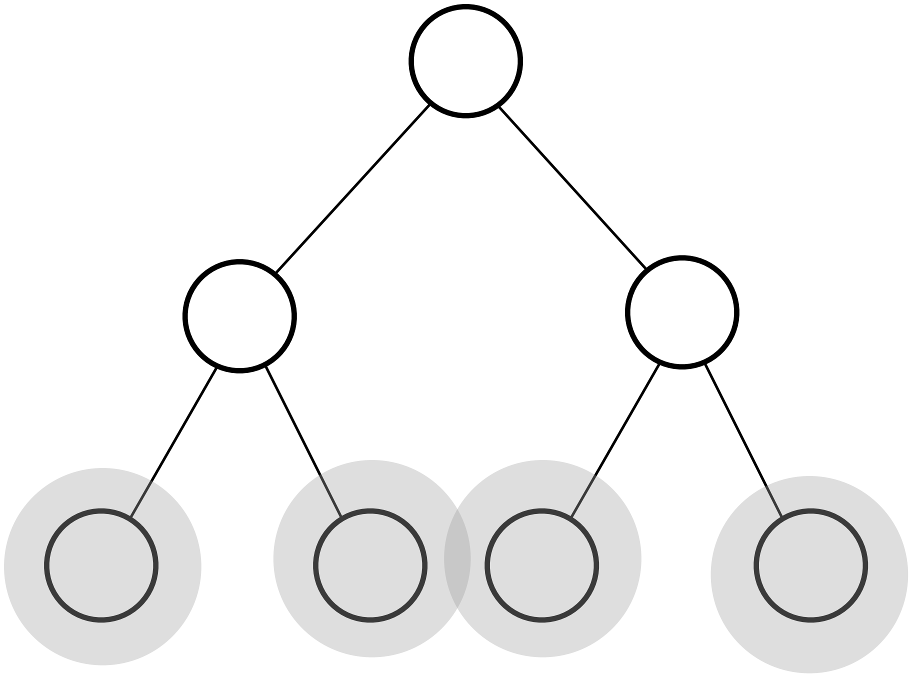




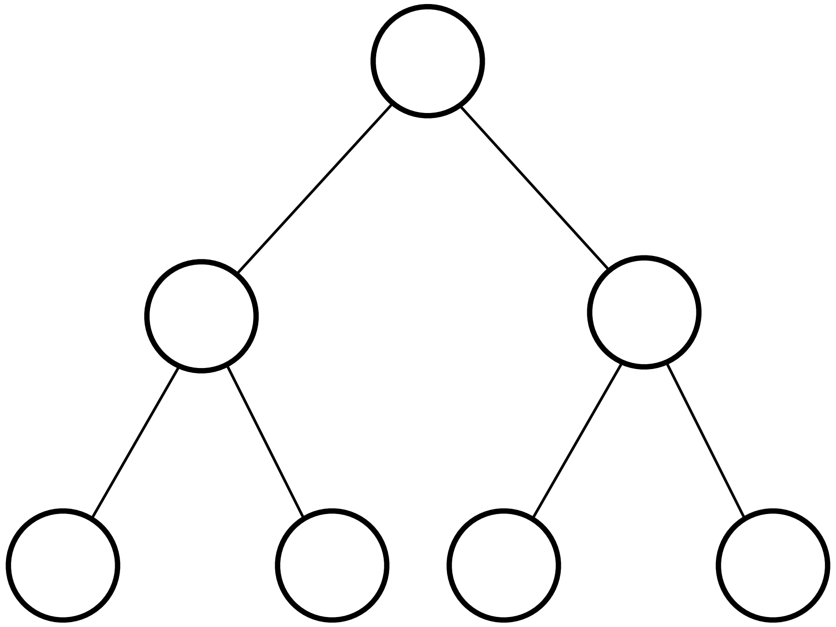
determined by other components

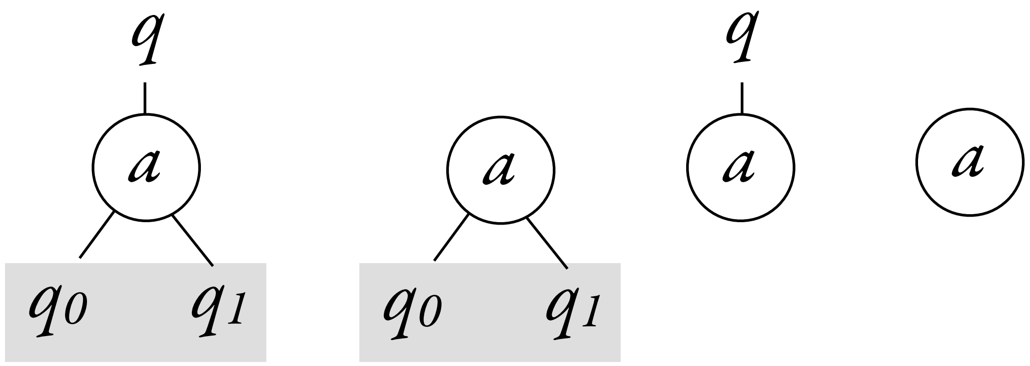
Deterministic automata

Top-down



Bottom-up

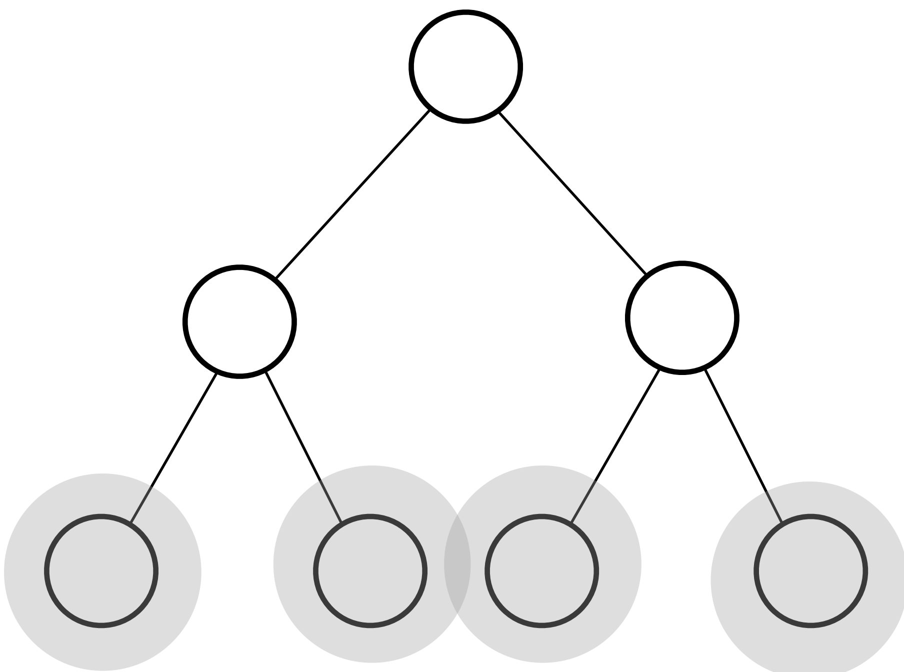




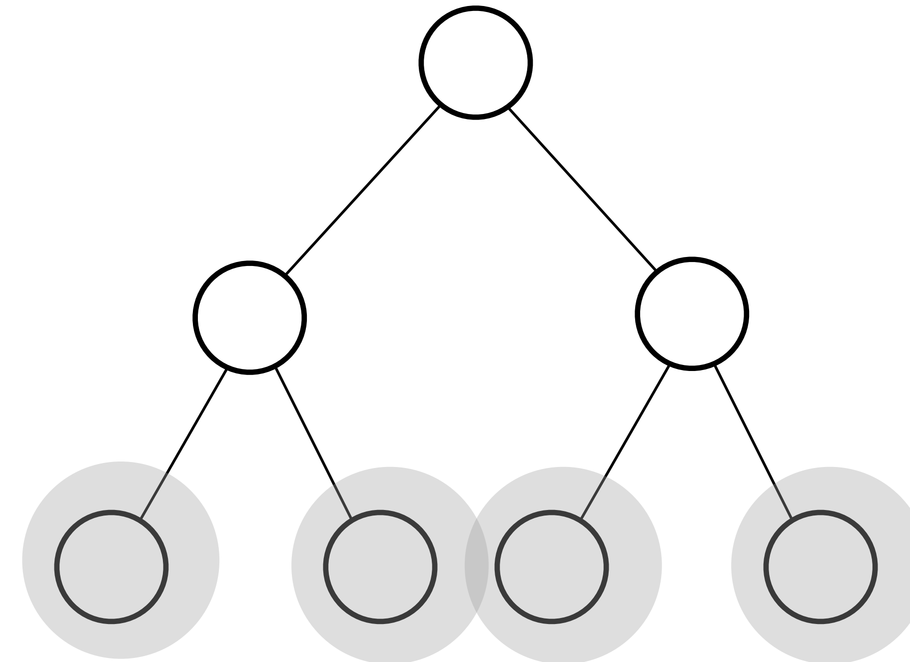
determined by other components

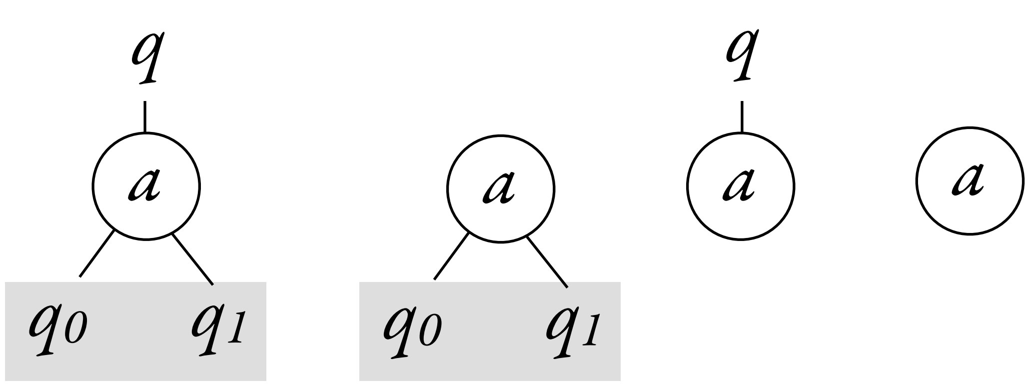
Deterministic automata

Top-down



Bottom-up

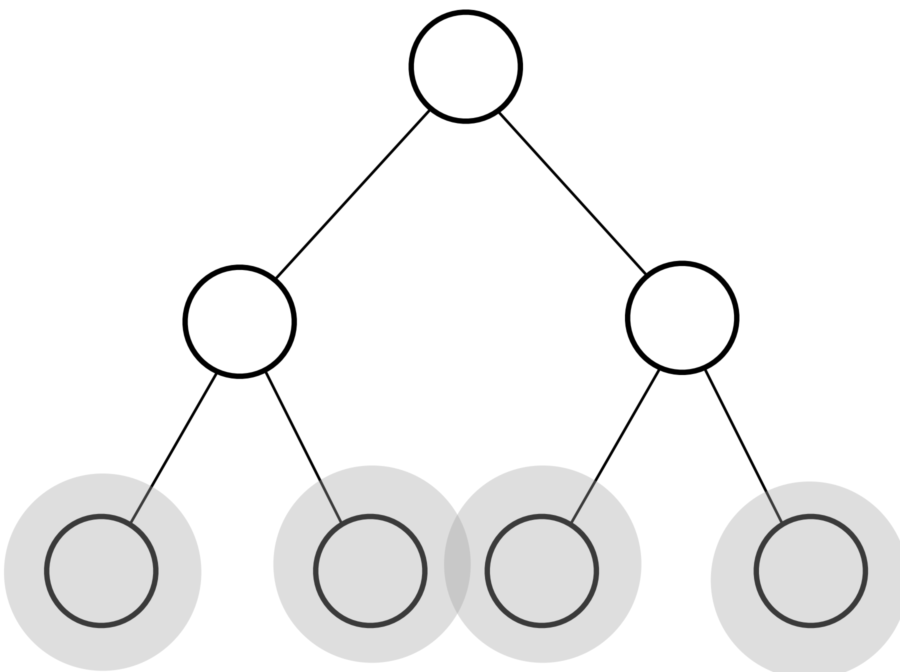




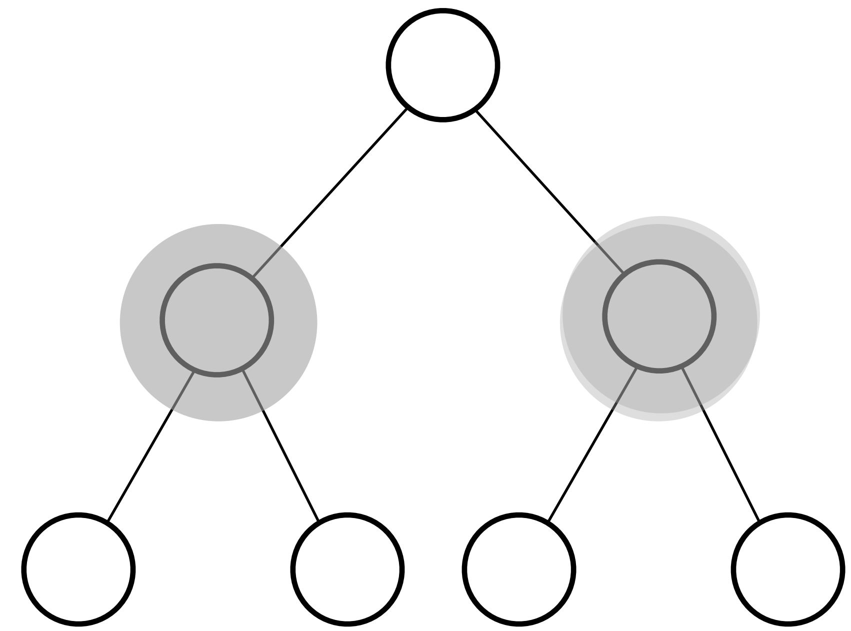
determined by other components

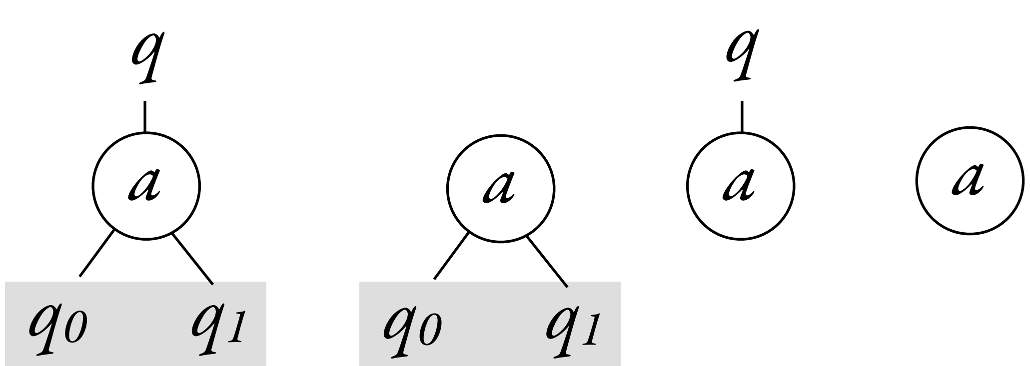
Deterministic automata

Top-down



Bottom-up

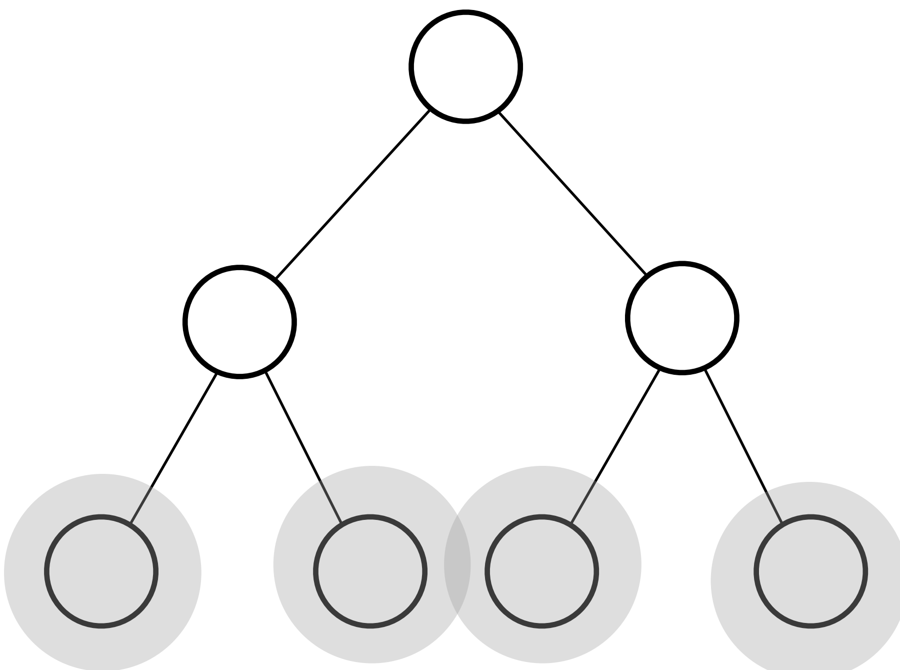




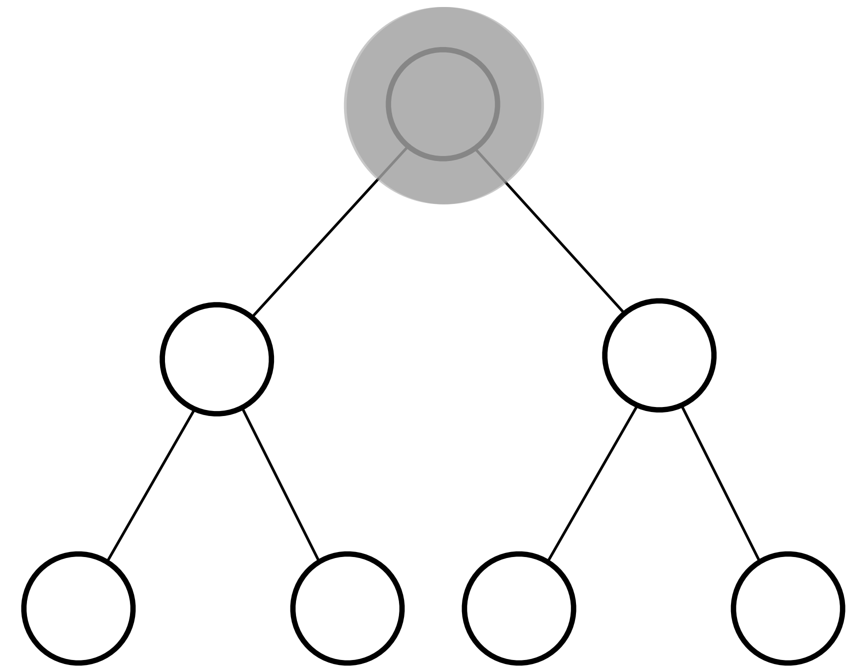
determined by other components

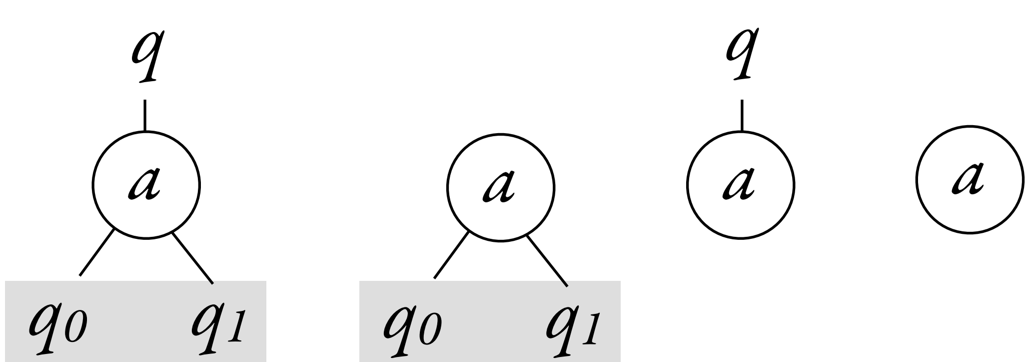
Deterministic automata

Top-down



Bottom-up

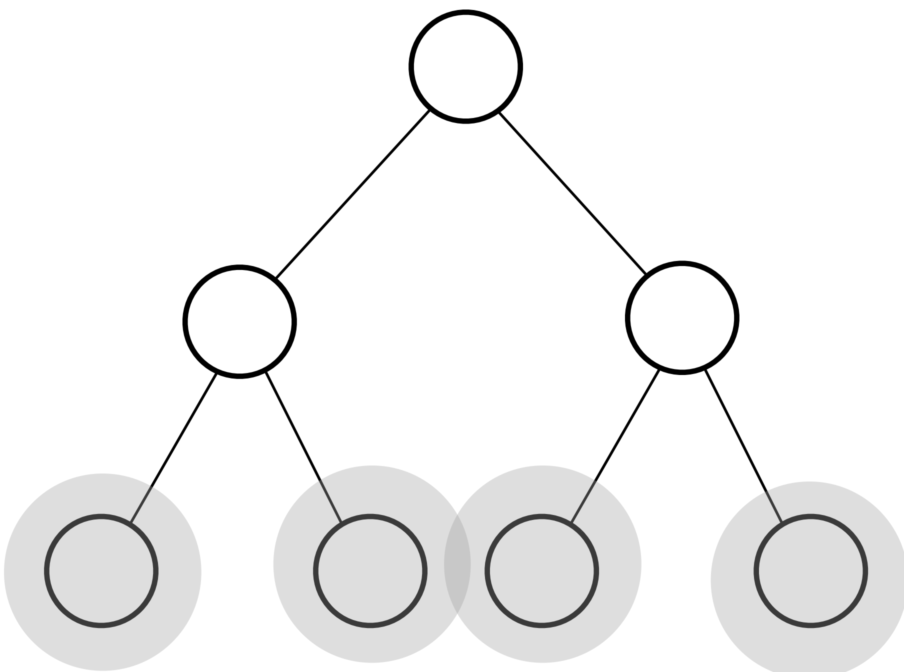


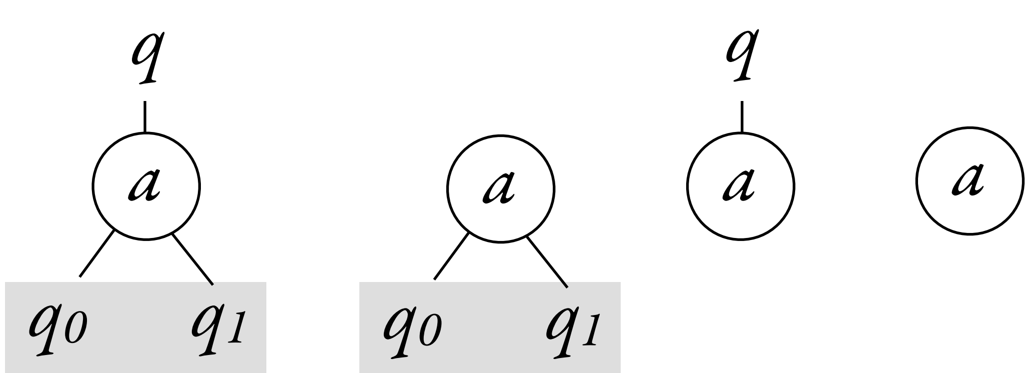


determined by other components

Deterministic automata

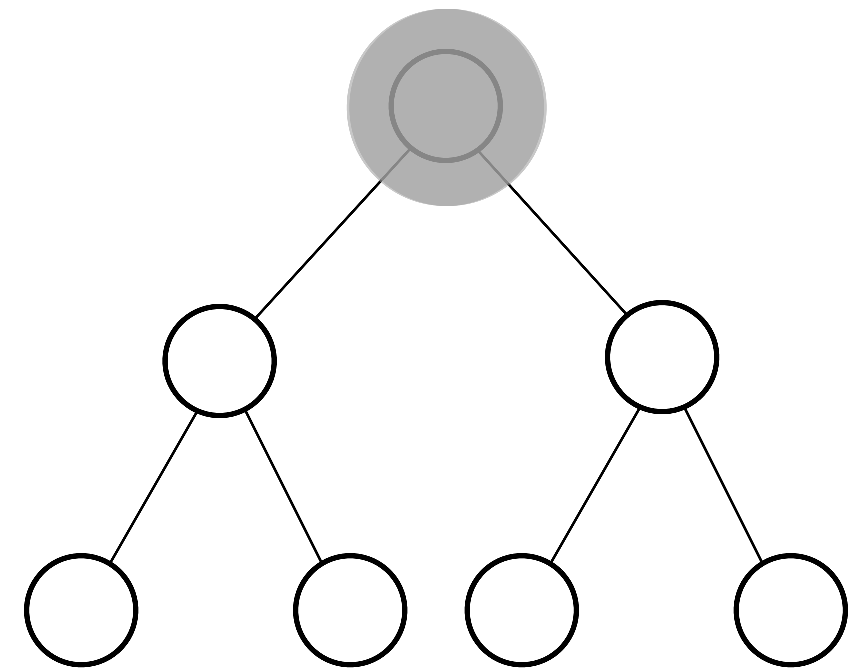
Top-down

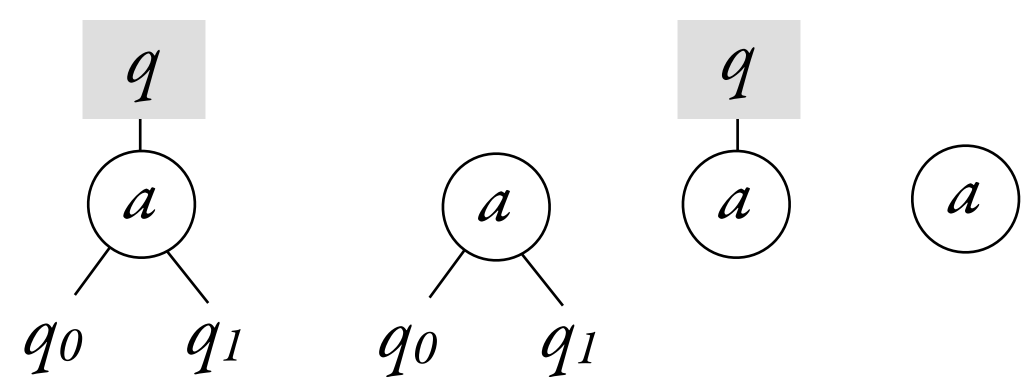




determined by other components

Bottom-up

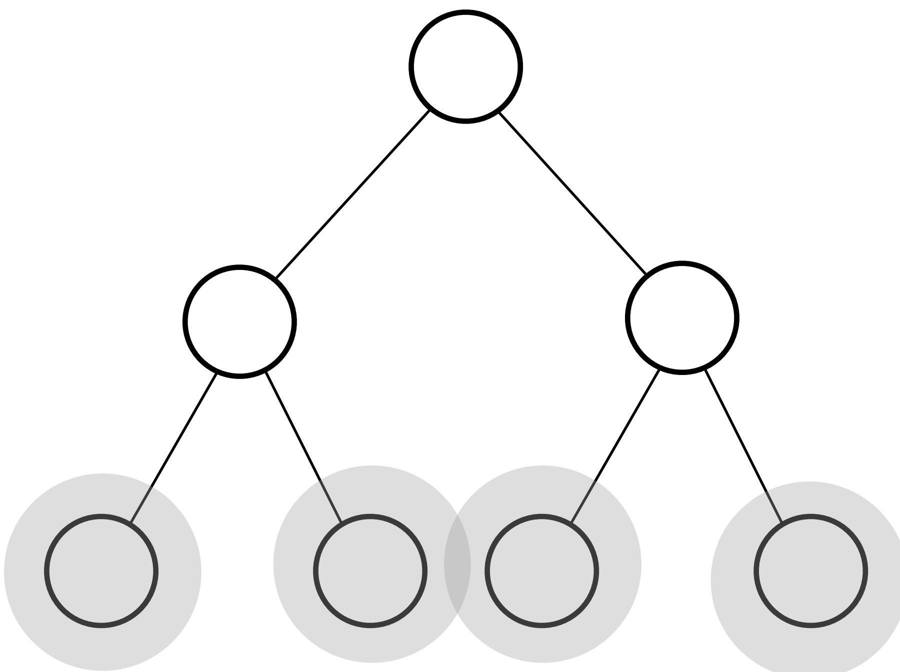


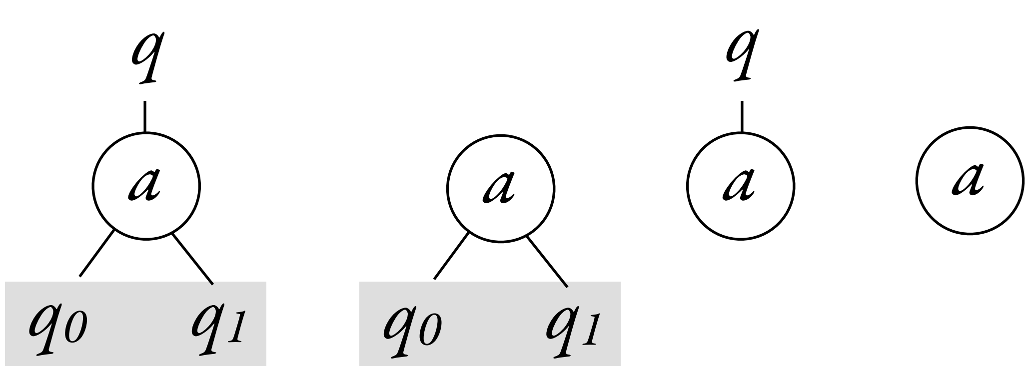


determined by other components

Deterministic automata

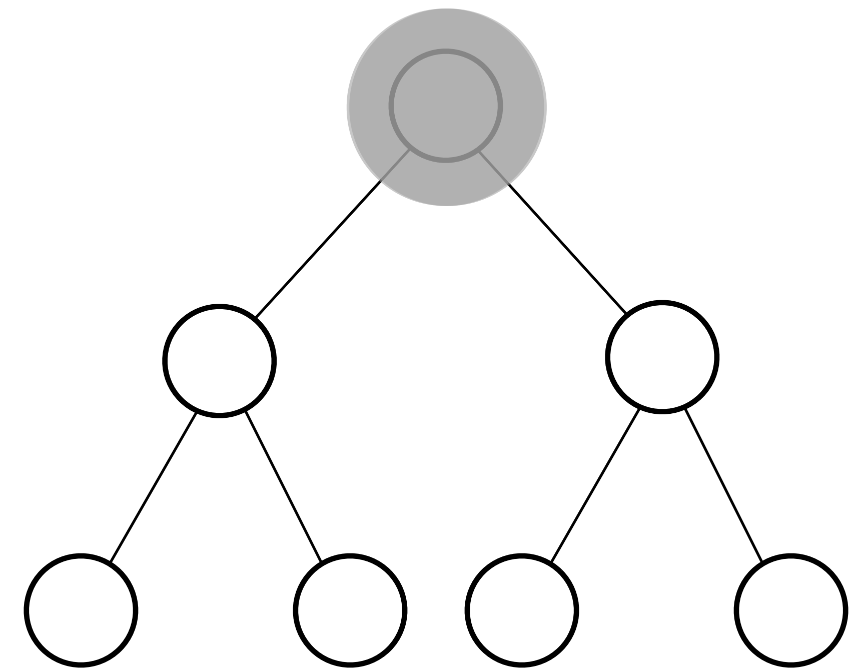
Top-down **bad**

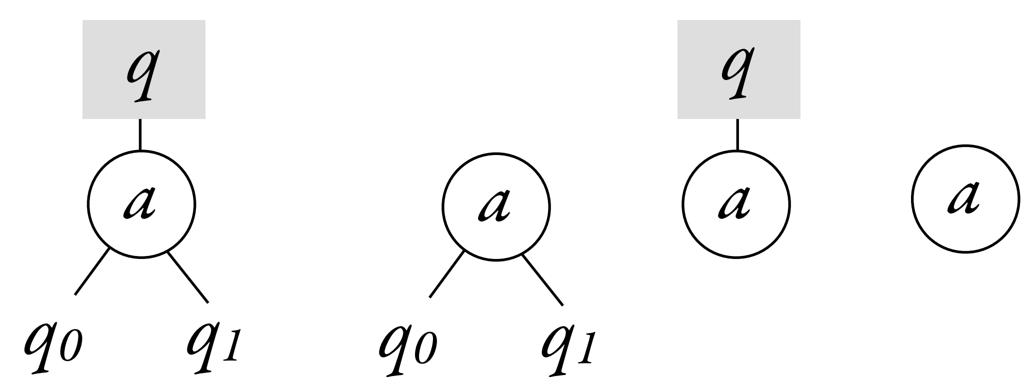




determined by other components

good Bottom-up





determined by other components

Bottom-up determinism

Fact. A nondeterministic automaton can be determinized to a bottom-up one.

Bottom-up determinism

Fact. A nondeterministic automaton can be determinized to a bottom-up one.

Standard subset construction works

Q state space of nondeterministic automaton.
$P(Q)$ state space of deterministic bottom-up automaton.

Bottom-up determinism

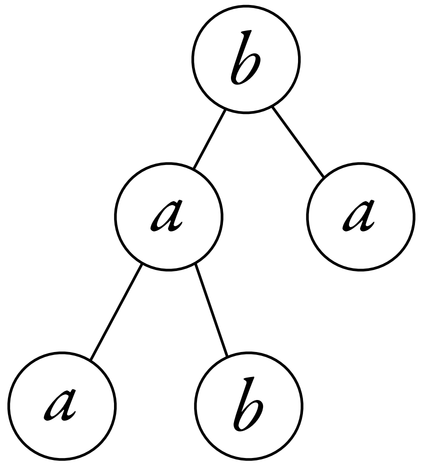
Fact. A nondeterministic automaton can be determinized to a bottom-up one.

Standard subset construction works

Q state space of nondeterministic automaton.
$P(Q)$ state space of deterministic bottom-up automaton.

set of possible states
for all possible runs

$P \subseteq Q$



Bottom-up determinism

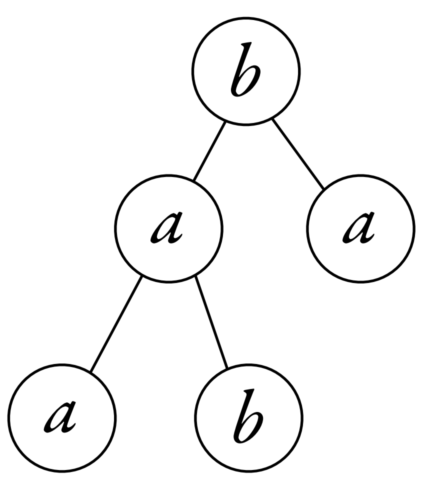
Fact. A nondeterministic automaton can be determinized to a bottom-up one.

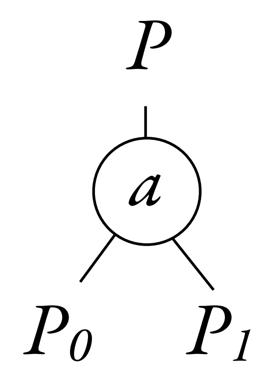
Standard subset construction works

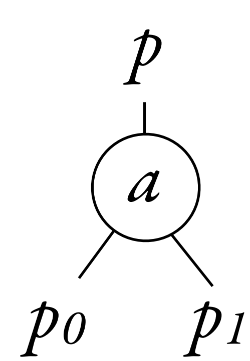
Q state space of nondeterministic automaton.
$P(Q)$ state space of deterministic bottom-up automaton.

set of possible states
for all possible runs

$P \subseteq Q$



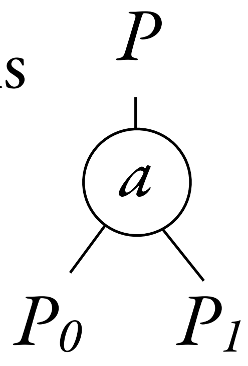


$P = \{\, p :$  holds for some $p_0 \in P_0,\, p_1 \in P_1 \,\}$

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

subset construction fails
for top-down
determinism

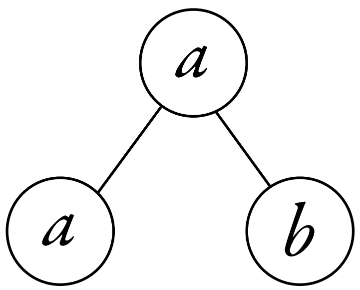


Top-down determinism

Fact. Top-down deterministic automata are not closed under union.

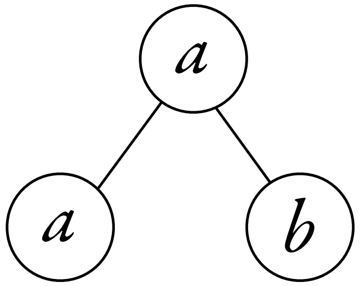
Top-down determinism

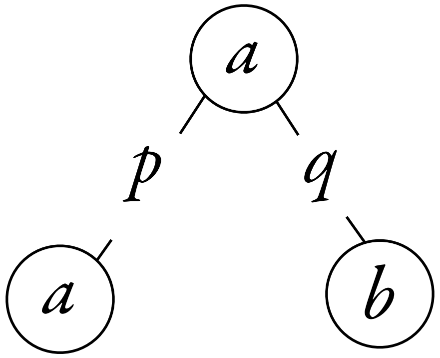
Fact. Top-down deterministic automata are not closed under union.

$L = \{$  $\}$

Top-down determinism

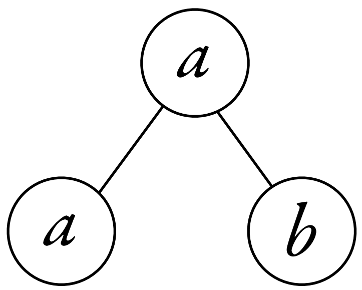
Fact. Top-down deterministic automata are not closed under union.

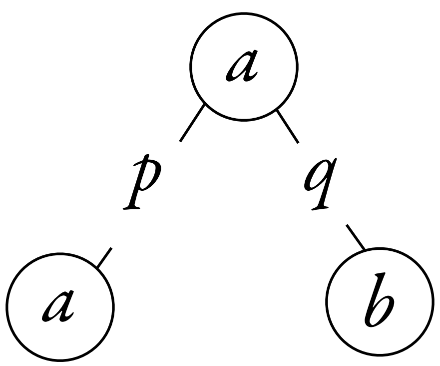
$L = \{$  $\}$

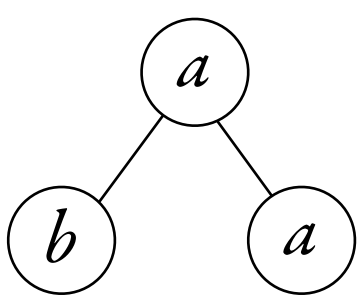


Top-down determinism

Fact. Top-down deterministic automata are not closed under union.

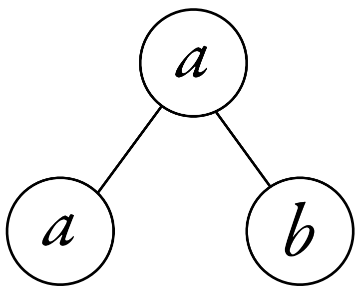
$L = \{$  $\}$

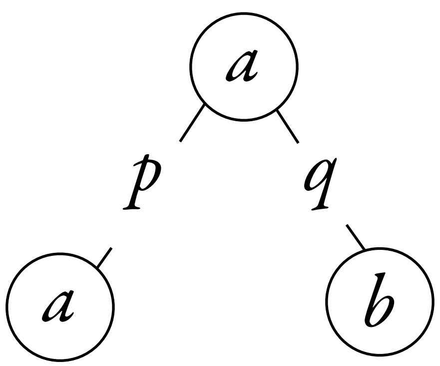


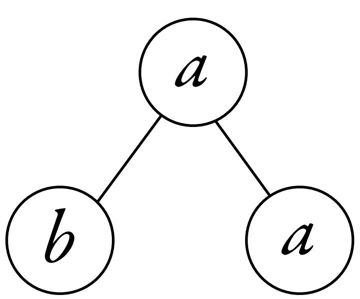
$K = \{$  $\}$

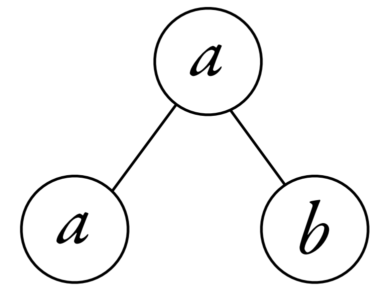
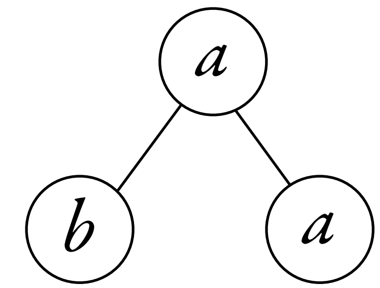
Top-down determinism

Fact. Top-down deterministic automata are not closed under union.

$L = \{$  $\}$

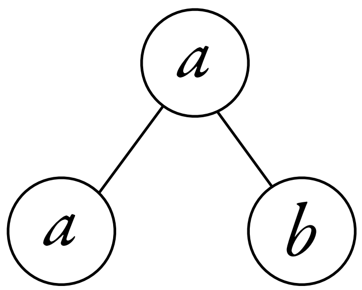


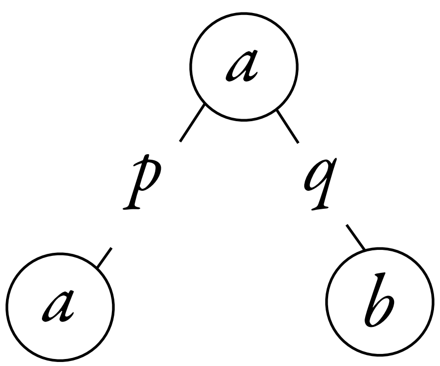
$K = \{$  $\}$

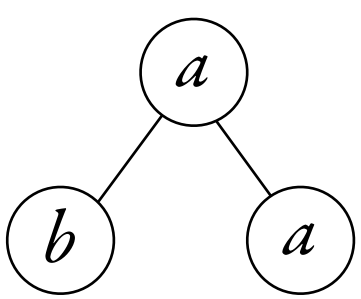
$L \cup K = \{$  ,  $\}$

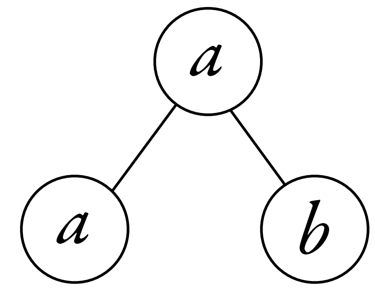
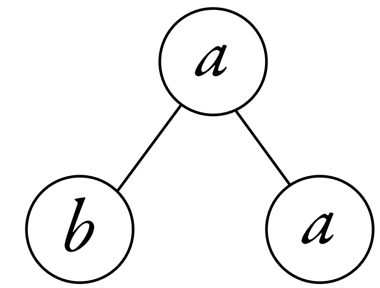
Top-down determinism

Fact. Top-down deterministic automata are not closed under union.

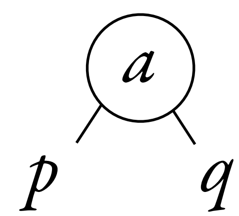
$L = \{$  $\}$



$K = \{$  $\}$

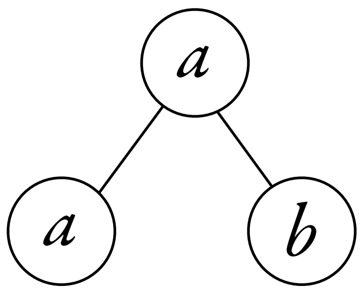
$L \cup K = \{$  ,  $\}$

There must be a transition,

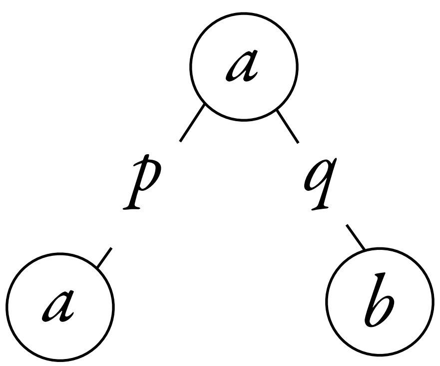


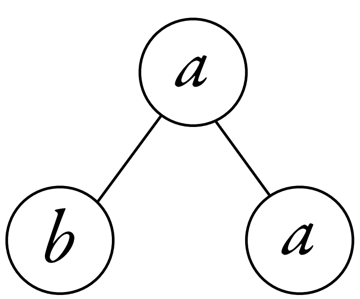
Top-down determinism

Fact. Top-down deterministic automata are not closed under union.

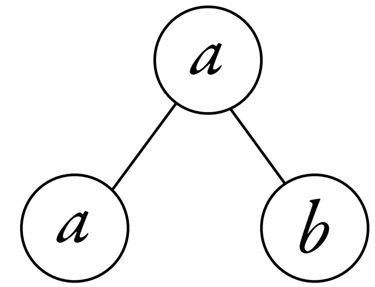
$L = \{$ 

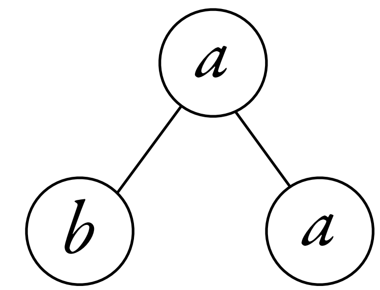
$\}$



$K = \{$ 

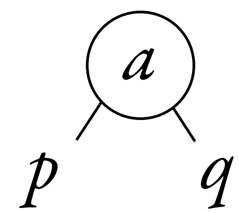
$\}$

$L \cup K = \{$ 

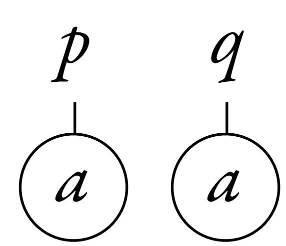
, 

$\}$

There must be a transition,

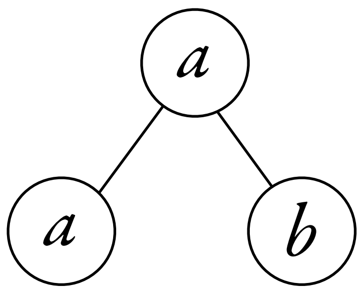


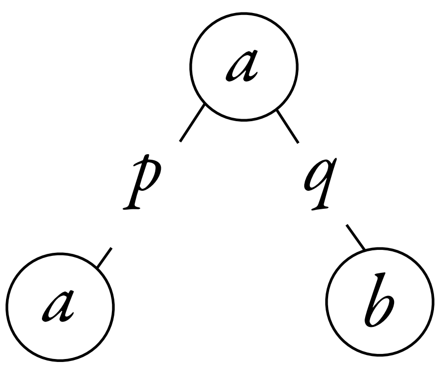
and hence also

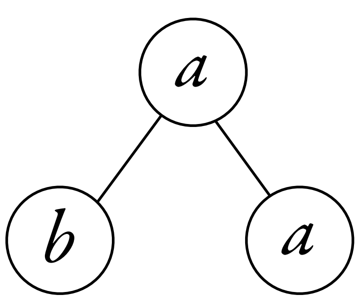


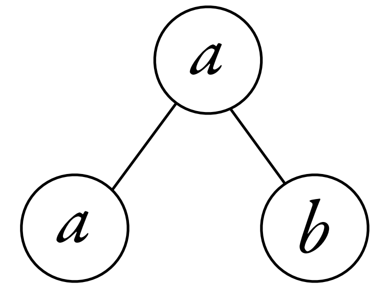
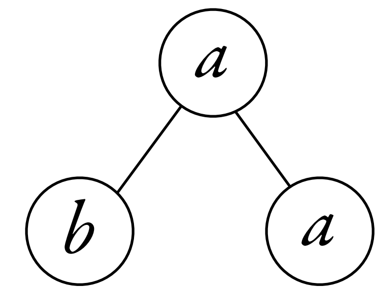
Top-down determinism

Fact. Top-down deterministic automata are not closed under union.

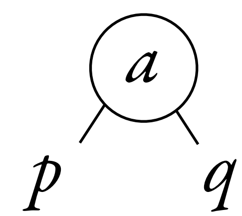
$L = \{$  $\}$



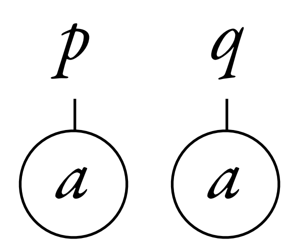
$K = \{$  $\}$

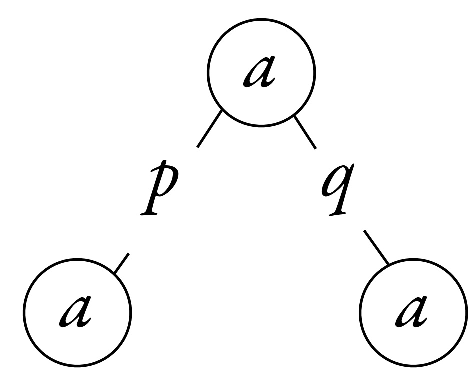
$L \cup K = \{$  ,  $\}$

There must be a transition,

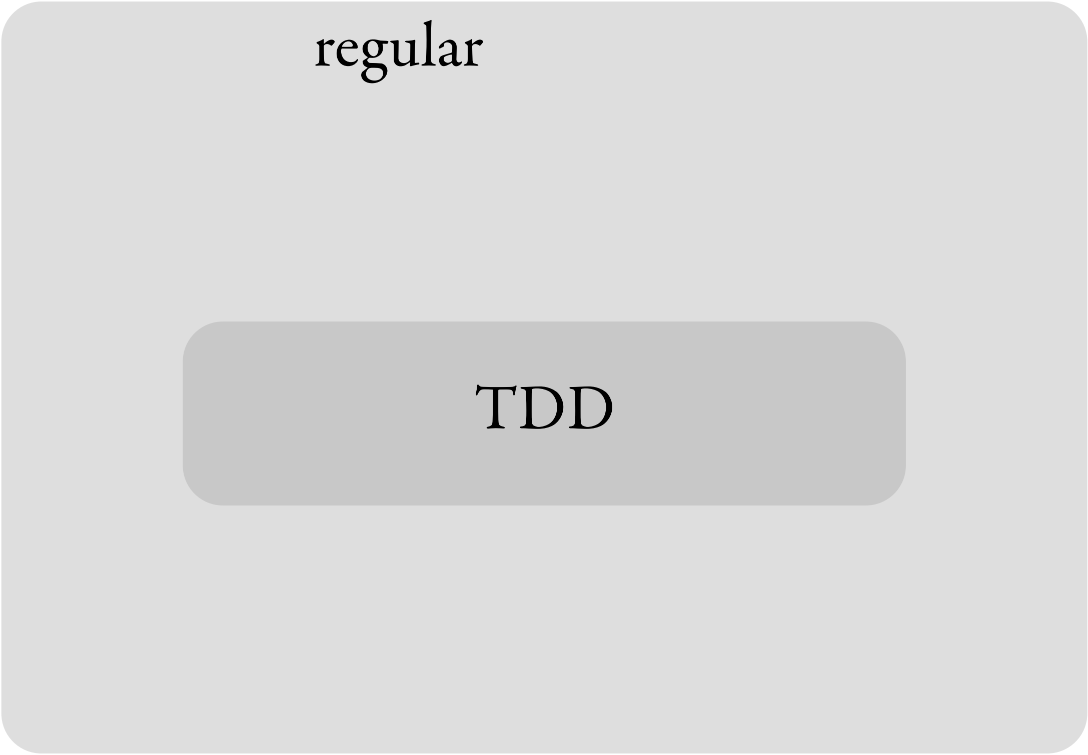


and hence also

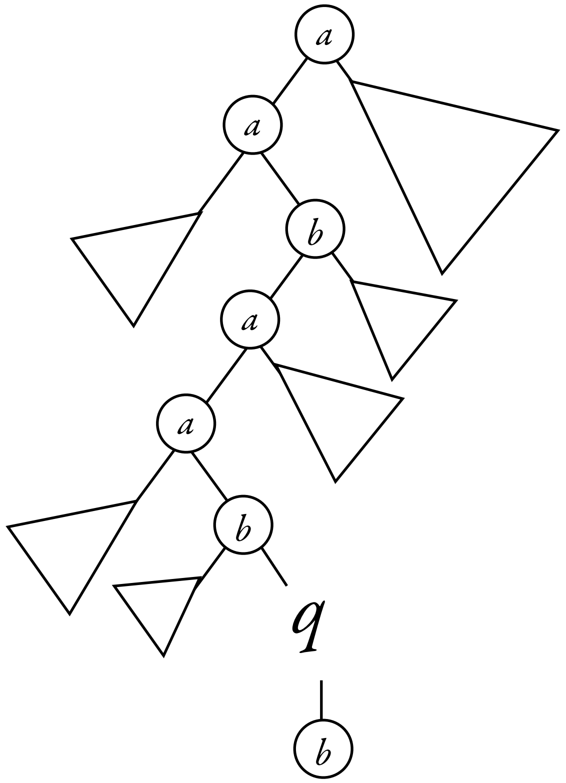


...but then: 

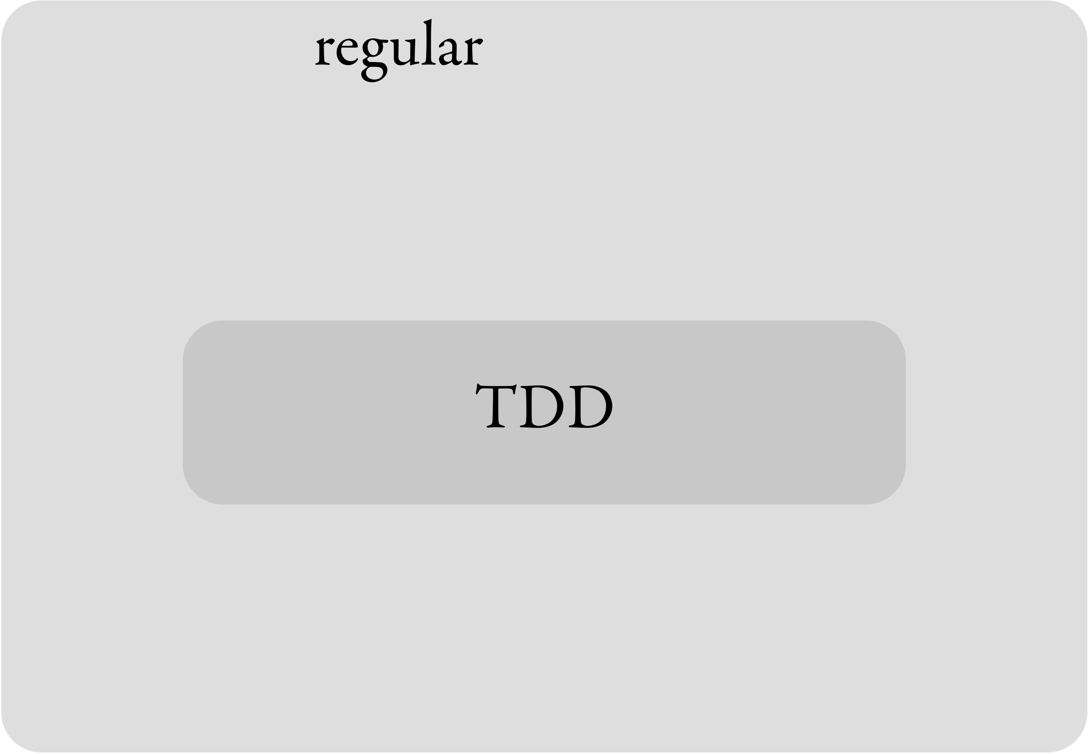
Top-down determinism, continued.



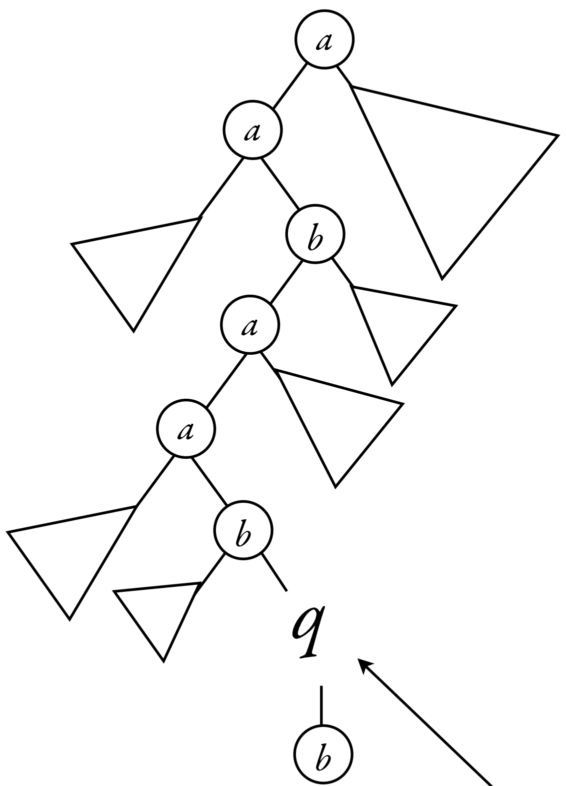
Top-down determinism, continued.



the run is accepting,
if every leaf state is
consistent with its leaf.

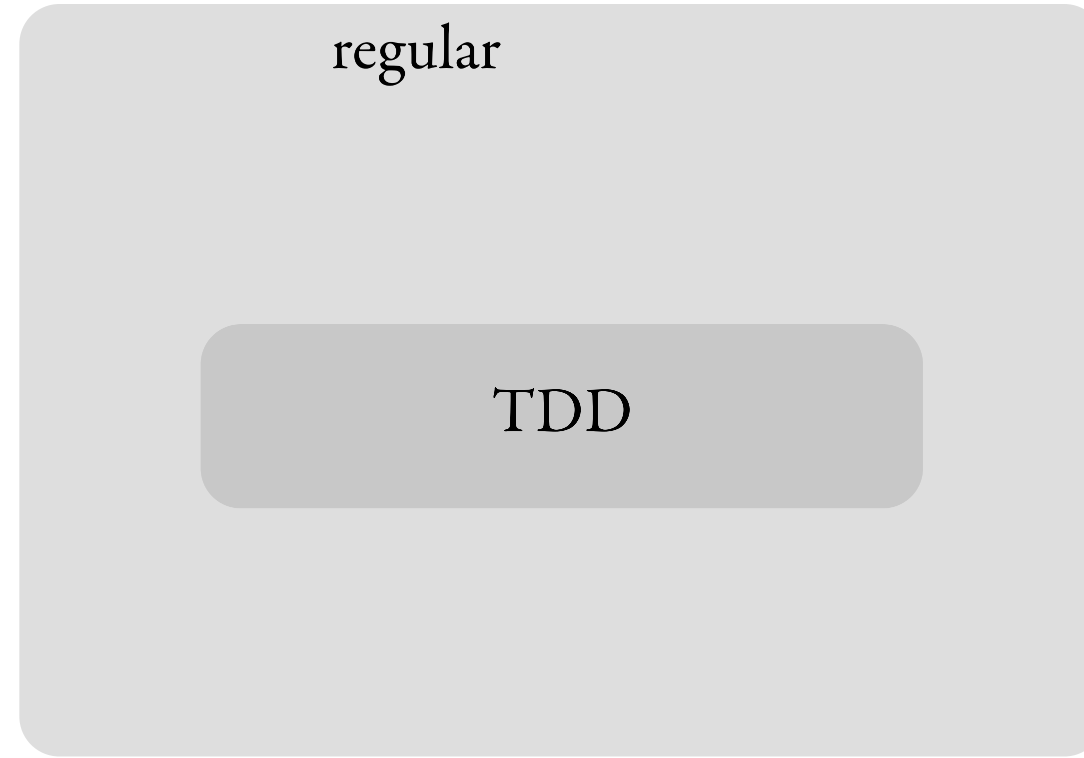


Top-down determinism, continued.

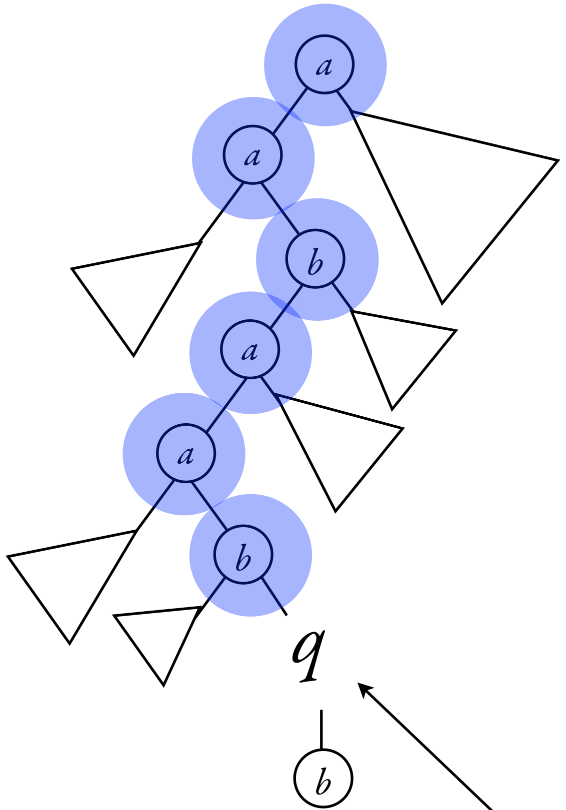


the run is accepting,
if every leaf state is
consistent with its leaf.

what does the leaf
state depend on?

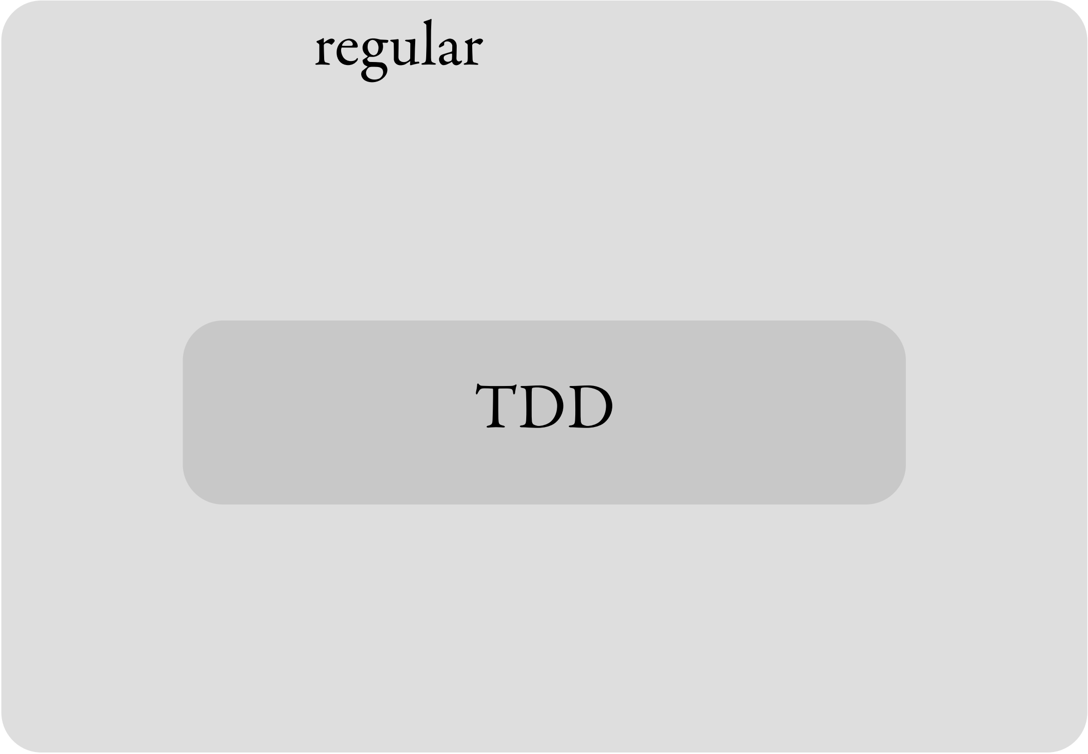


Top-down determinism, continued.

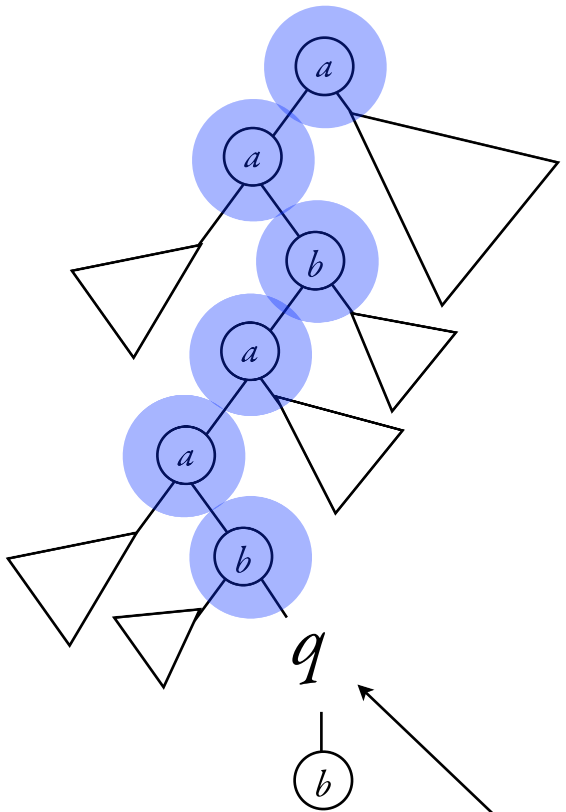


the run is accepting,
if every leaf state is
consistent with its leaf.

what does the leaf
state depend on?

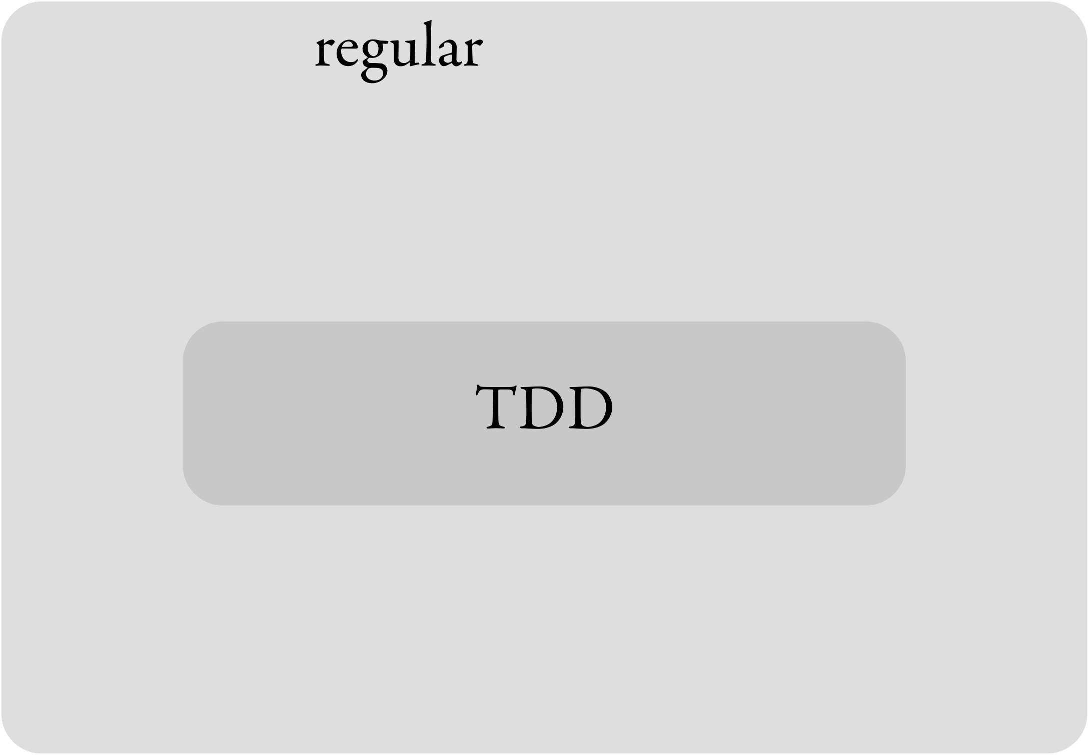


Top-down determinism, continued.



the run is accepting,
if every leaf state is
consistent with its leaf.

what does the leaf
state depend on?

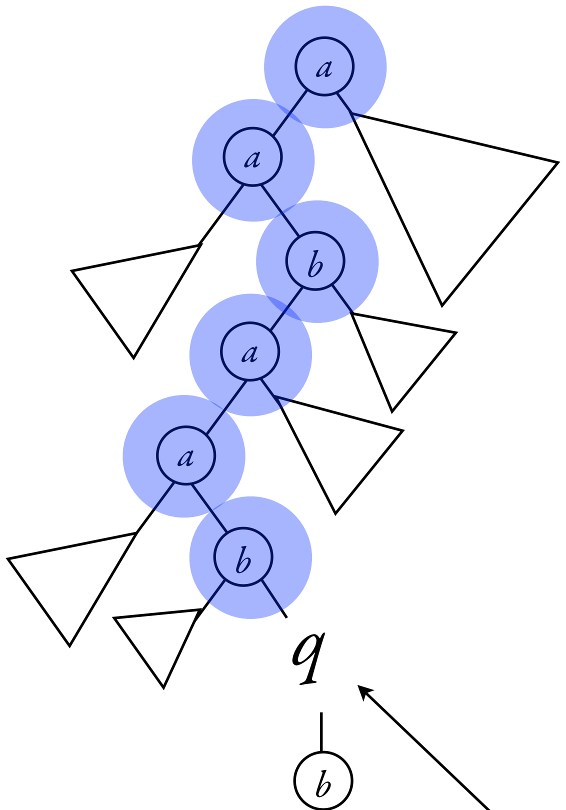


Fact. The following are equivalent for a tree language L:

– L is recognized by a deterministic top-down tree automaton

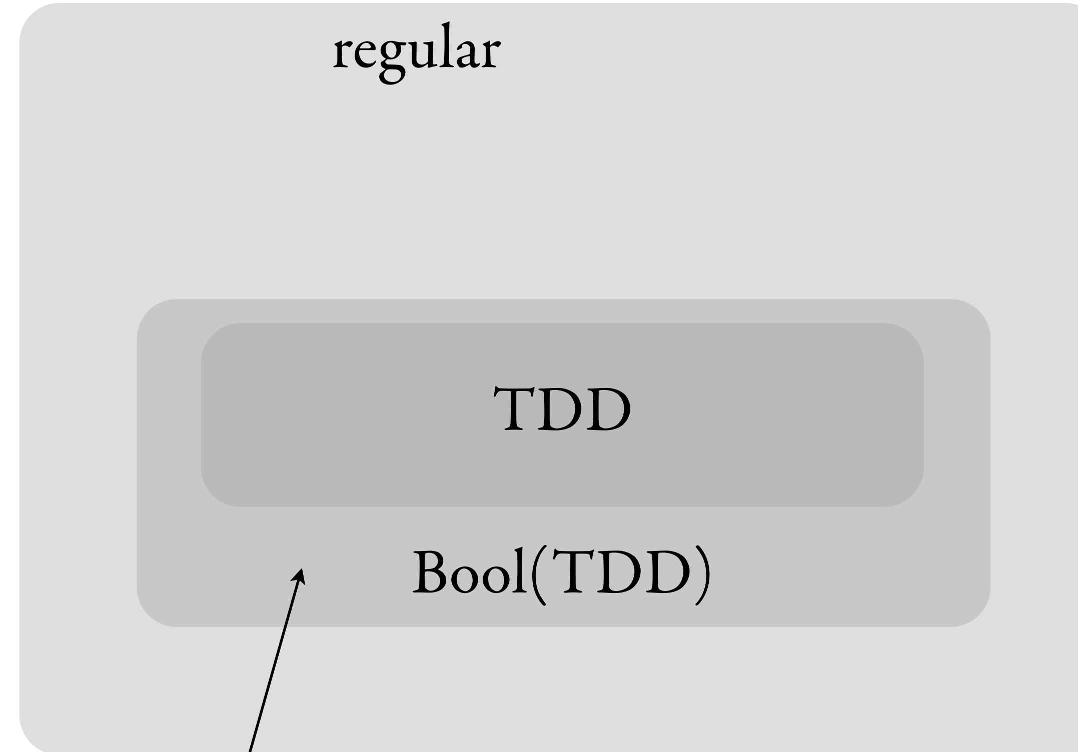
– L is equivalent to "all paths in K", for a regular word language $K \subseteq (\Sigma\times\{0,1\})^*$

Top-down determinism, continued.



the run is accepting,
if every leaf state is
consistent with its leaf.

what does the leaf
state depend on?



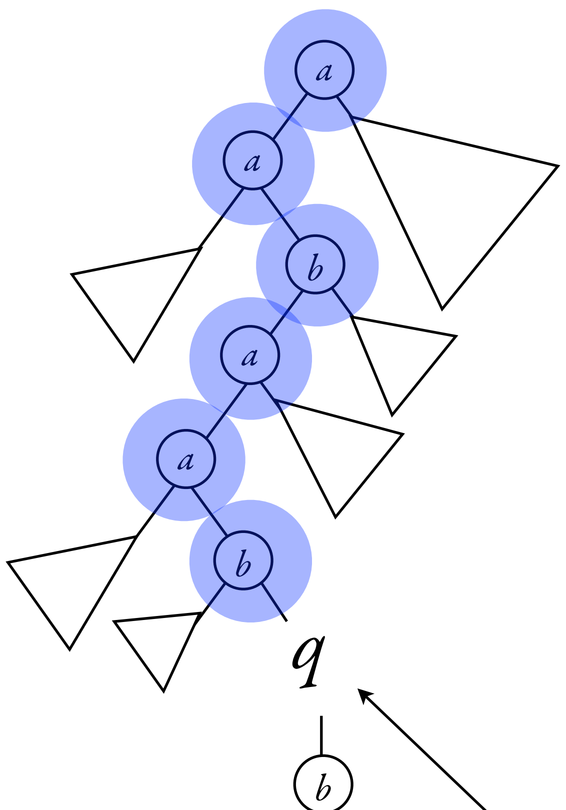
languages of the form: all paths .., some path ...

Fact. The following are equivalent for a tree language L:

– L is recognized by a deterministic top-down tree automaton

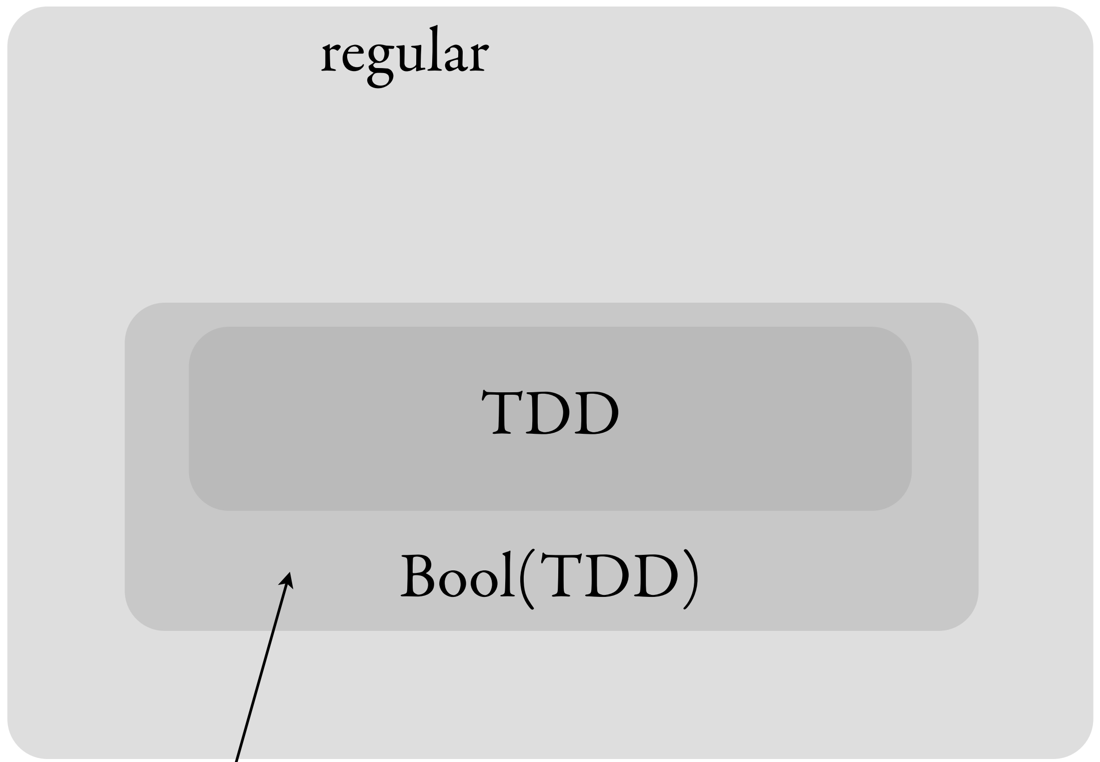
– L is equivalent to "all paths in K", for a regular word language $K \subseteq (\Sigma\times\{0,1\})^*$

Top-down determinism, continued.



the run is accepting, if every leaf state is consistent with its leaf.

what does t
state depen



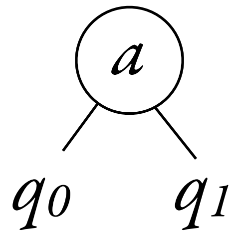
languages of the form: all paths .., some path ...

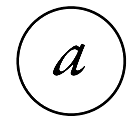
It is not difficult to give an algorithm deciding $L \in$ TDD.
Open problem: give an algorithm deciding $L \in$ Bool(TDD).

Fact. The following are equivalent for a tree language L:
– L is recognized by a deterministic top-down tree automaton
– L is equivalent to "all paths in K", for a regular word language $K \subseteq (\Sigma \times \{0,1\})^*$

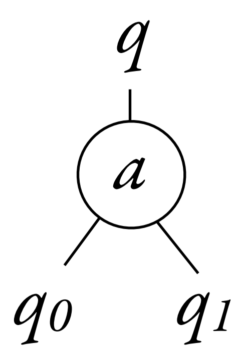
Minimal automata

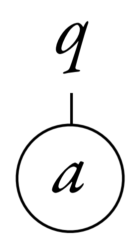
for the purpose of minimal automata, we change the definition.

We remove transitions 



and add accepting states. A run must have an accepting state at the root.

(These transitions 

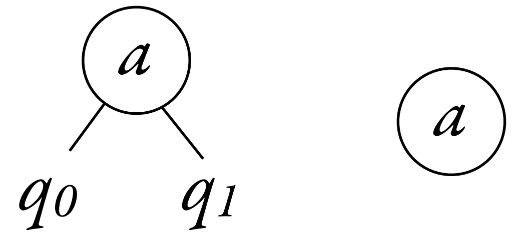


work if the node is the root or not)

Minimal automata

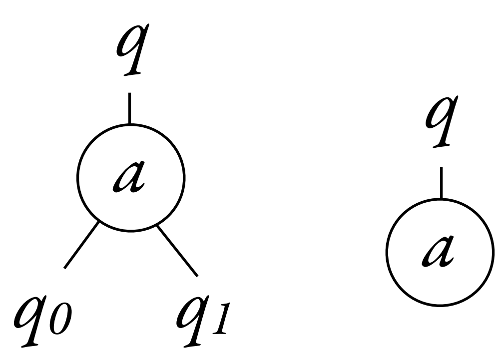
for the purpose of minimal automata, we change the definition.

We remove transitions



and add accepting states. A run must have an accepting state at the root.

(These transitions



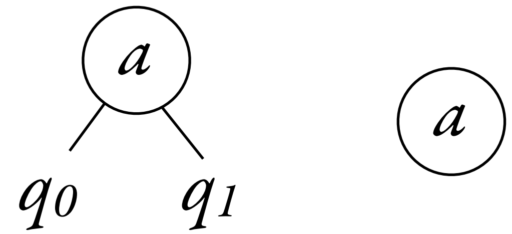
work if the node is the root or not)

Reason: in the old definition, the language $\{a\}$
is recognized by an automaton with 0 states.

Minimal automata

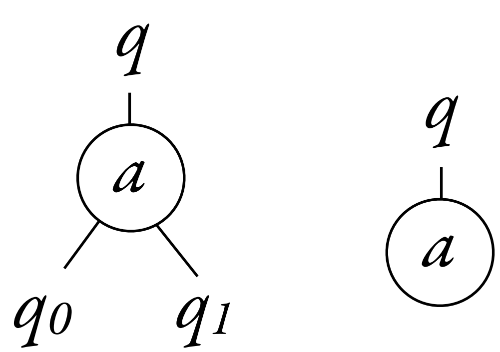
for the purpose of minimal automata, we change the definition.

We remove transitions



and add accepting states. A run must have an accepting state at the root.

(These transitions



work if the node is the root or not)

Reason: in the old definition, the language $\{ a \}$
is recognized by an automaton with 0 states.

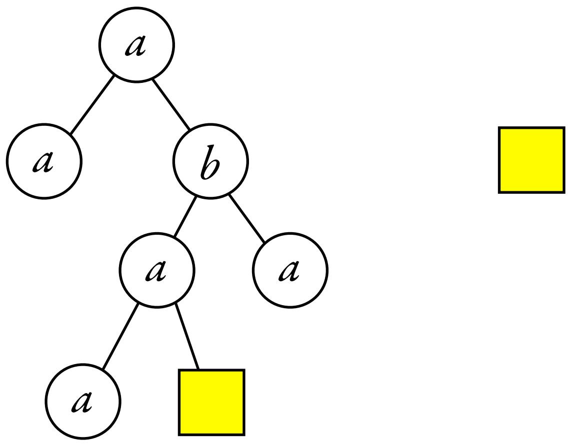
Tuple definition: $(Q, \Sigma, \delta : Q\times\Sigma\times Q \to Q,\ I : \Sigma \to Q, F \subseteq Q)$

Minimalization

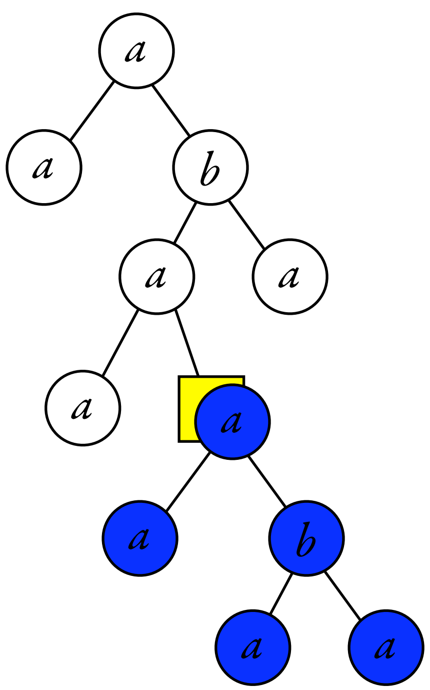
Proposition. For every regular tree language, there is a unique (up to isomorphism) minimal deterministic bottom-up tree automaton.
The minimal automaton can be obtained from any other automaton in polynomial time.

States of the minimal automaton: equivalence classes of the Myhill-Nerode congruence.

Myhill-Nerode congruence for tree languages

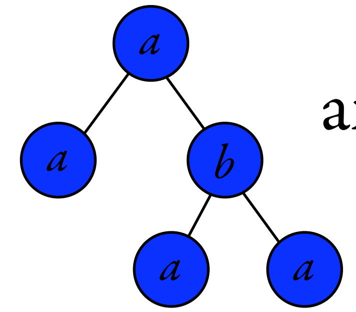


context

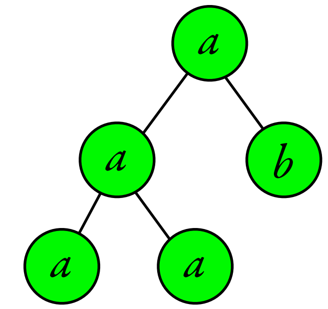


substitution

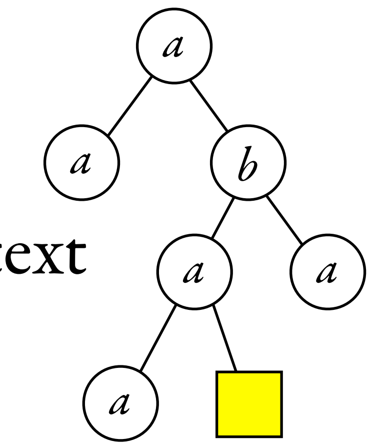
Two trees

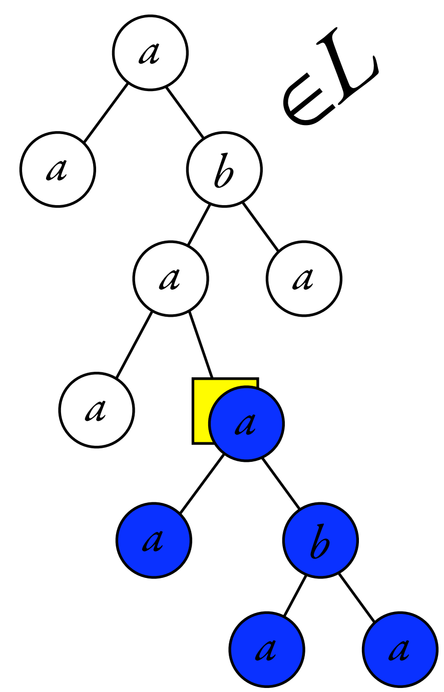


and

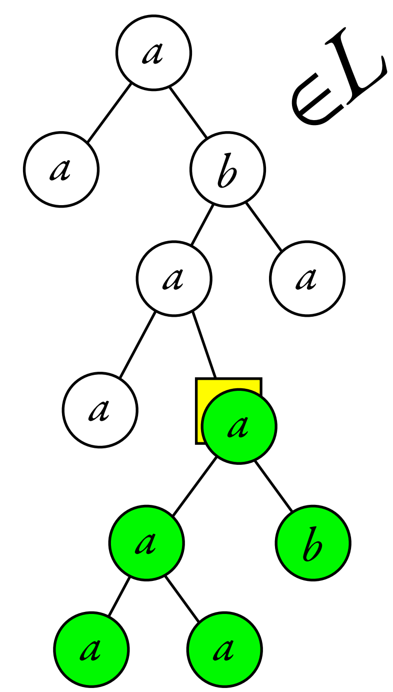


are L-equivalent if for every context





iff



Myhill-Nerode congruence for tree languages, examples

Myhill-Nerode congruence for tree languages, examples

“two a’s”

classes of the Myhill-Nerode congruence:

0=“zero a’s”

1=“one a”

2=“two a’s”

3=“three or more a’s”

Myhill-Nerode congruence for tree languages, examples

"two a's"

classes of the Myhill-Nerode congruence:

0="zero a's"

1="one a"

2="two a's"

3="three or more a's"

balanced binary tree

infinitely many classes:

"unbalanced tree"

"balanced tree of depth n"

Myhill-Nerode congruence for tree languages, examples

“two a’s”

classes of the Myhill-Nerode congruence:

0=“zero a’s”

1=“one a”

2=“two a’s”

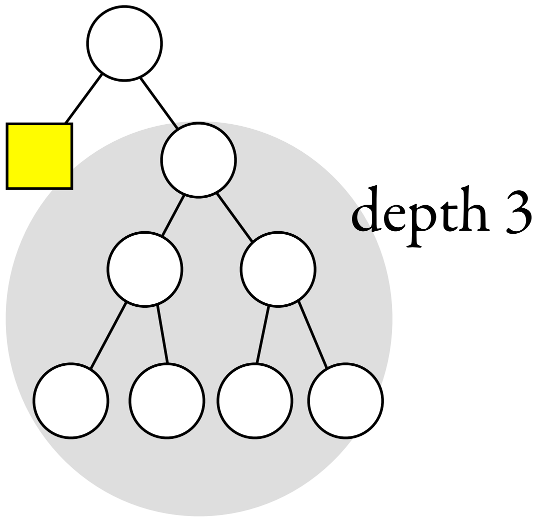
3=“three or more a’s”

balanced binary tree

infinitely many classes:

“unbalanced tree”

“balanced tree of depth n”



this context distinguishes a balanced tree of depth 3 from any other tree.

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular tree languages share many properties of regular word languages:

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular tree languages share many properties of regular word languages:

– automaton model

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular tree languages share many properties of regular word languages:

– automaton model

– efficient algorithms

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular tree languages share many properties of regular word languages:
– automaton model
– efficient algorithms
– closure properties (logic)

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular tree languages share many properties of regular word languages:

– automaton model
– efficient algorithms
– closure properties (logic)
– regular expressions

Regular languages

Def. A *regular* tree language is one recognized by a nondeterministic tree automaton or, equivalently, by a deterministic bottom up tree automaton.

Regular tree languages share many properties of regular word languages:

– automaton model
– efficient algorithms
– closure properties (logic)
– regular expressions

...but there are also some similarities to context-free languag

Regular tree languages are similar to context-free languages.

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

Yield of a tree: word with leaf labels, left-to-right.

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

from grammars to automata	from automata to grammars

Regular tree languages are similar to context-free languages.

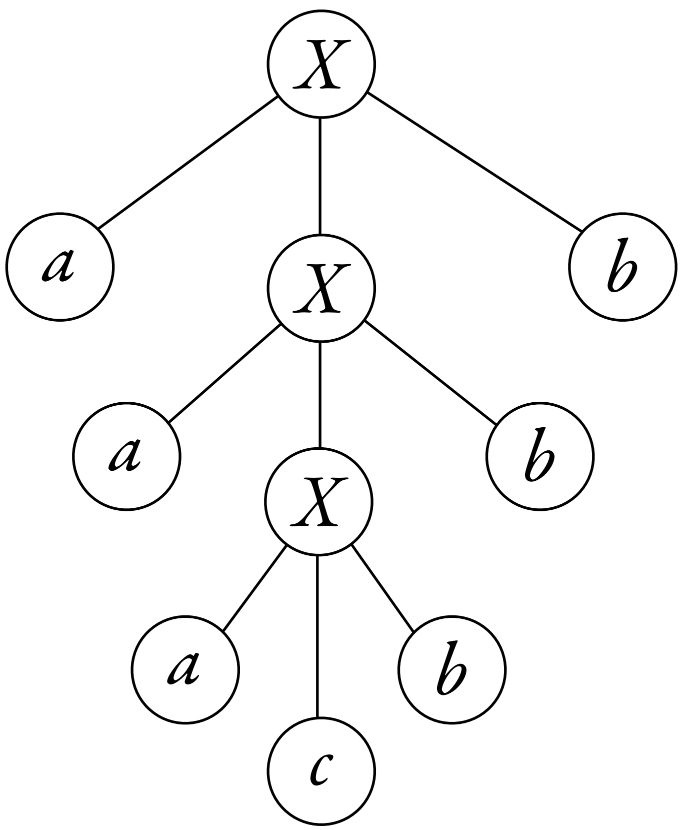
Fact. A word language is context-free iff it is the yield of some regular tree language.

from grammars to automata	from automata to grammars
$X \longrightarrow c \mid aXb$	

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

from grammars to automata	from automata to grammars

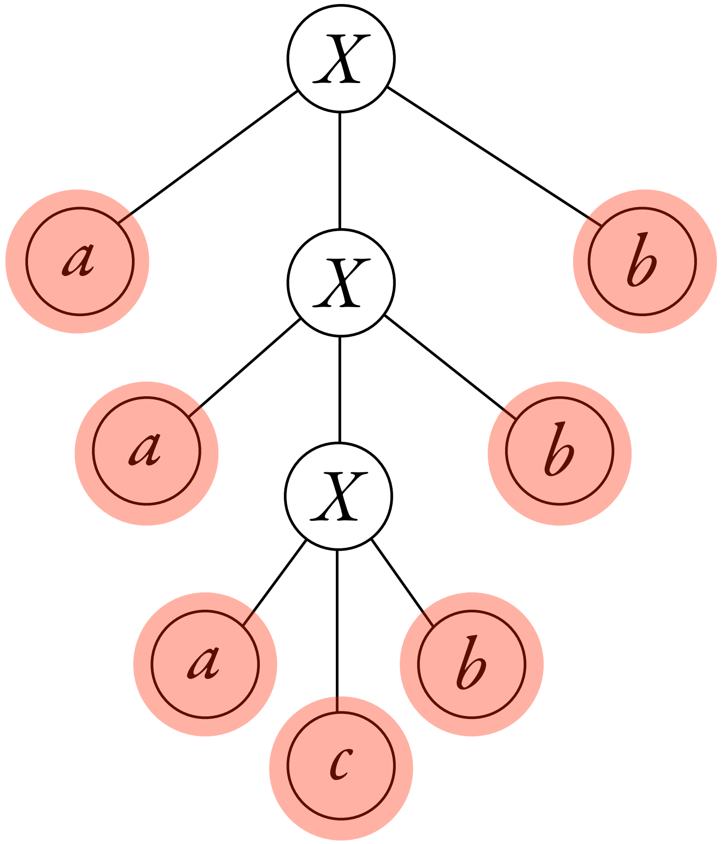


$$X \longrightarrow c \mid aXb$$

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

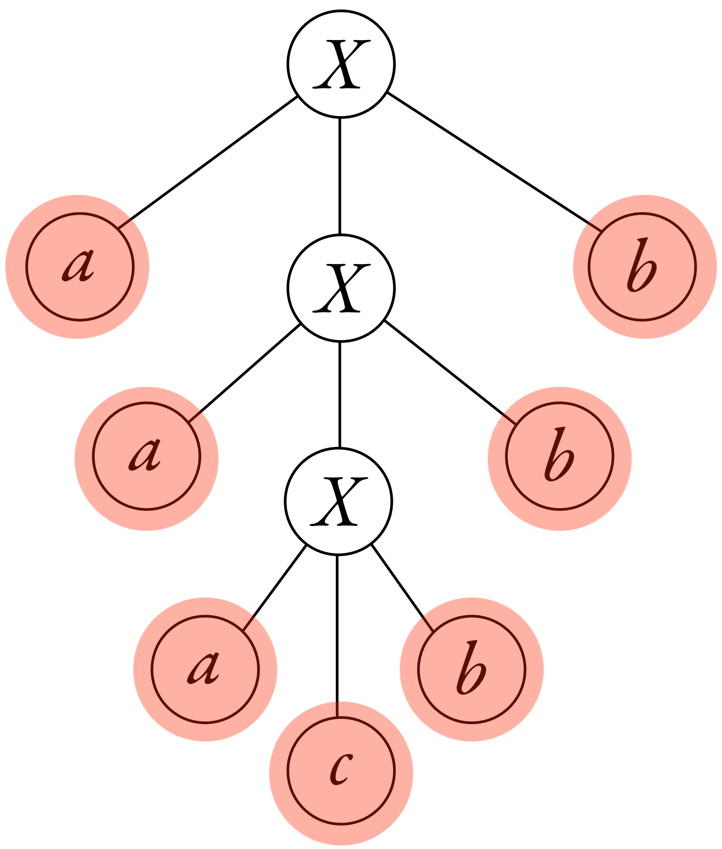
from grammars to automata	from automata to grammars



$$X \longrightarrow c \mid aXb$$

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

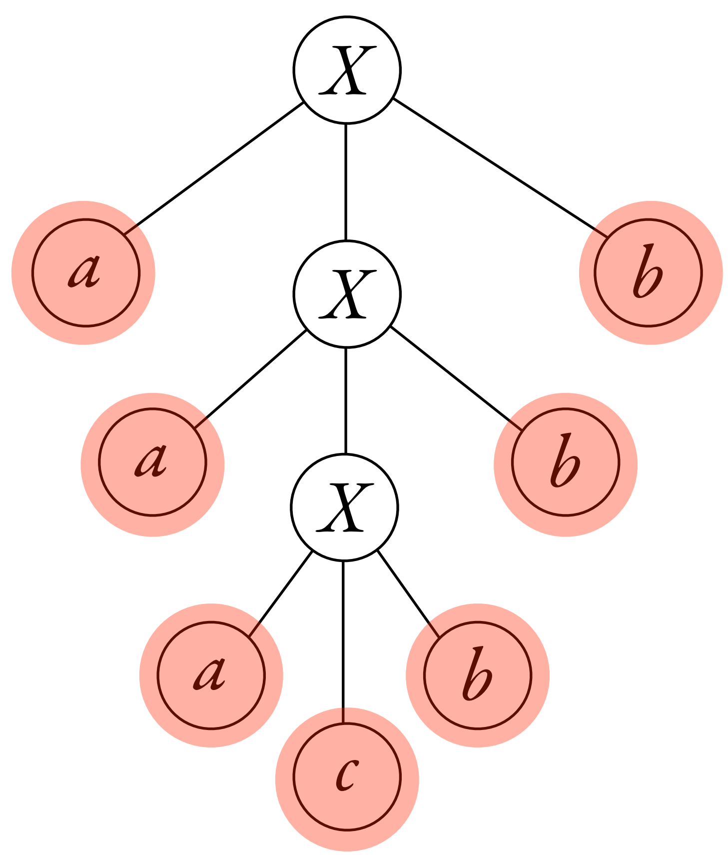
from grammars to automata	from automata to grammars
	Let A be a nondeterministic, top-down tree automaton. The grammar $G(A)$ has states of A as nonterminals, and rules:

$$X \longrightarrow c \mid aXb$$

Regular tree languages are similar to context-free languages.

Fact. A word language is context-free iff it is the yield of some regular tree language.

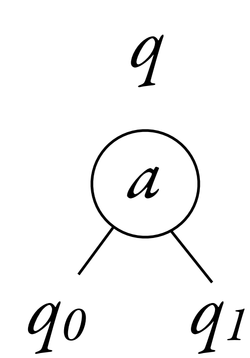
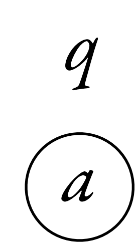
from grammars to automata



$$X \longrightarrow c \mid aXb$$

from automata to grammars

Let A be a nondeterministic, top-down tree automaton. The grammar $G(A)$ has states of A as nonterminals, and rules:

automaton transition	grammar rule
	$q \longrightarrow q_0\, q_1$
	$q \longrightarrow a$

What is a Tree Automaton?

- –definition and examples
- –determinism, bottom-up vs top-down
- –minimization
- –closure properties

Decision Problems

- –emptiness
- –membership
- –universality

What is a Tree Automaton?

- –definition and examples
- –determinism, bottom-up vs top-down
- –minimization
- –closure properties

Decision Problems

- –emptiness
- –membership
- –universality

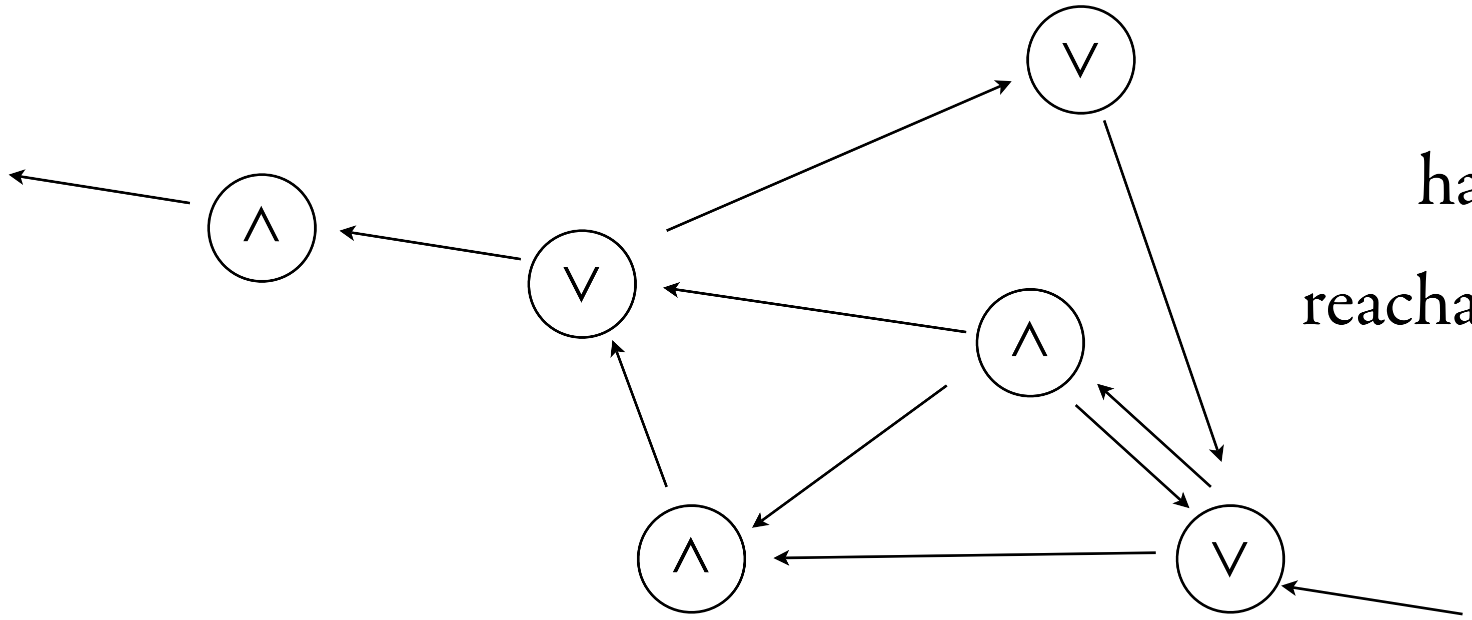
Emptiness

Thm. Emptiness for tree automata is PTIME-complete.

Tree automata emptiness can be tested in linear time.

Emptiness

Thm. Emptiness for tree automata is PTIME-complete.



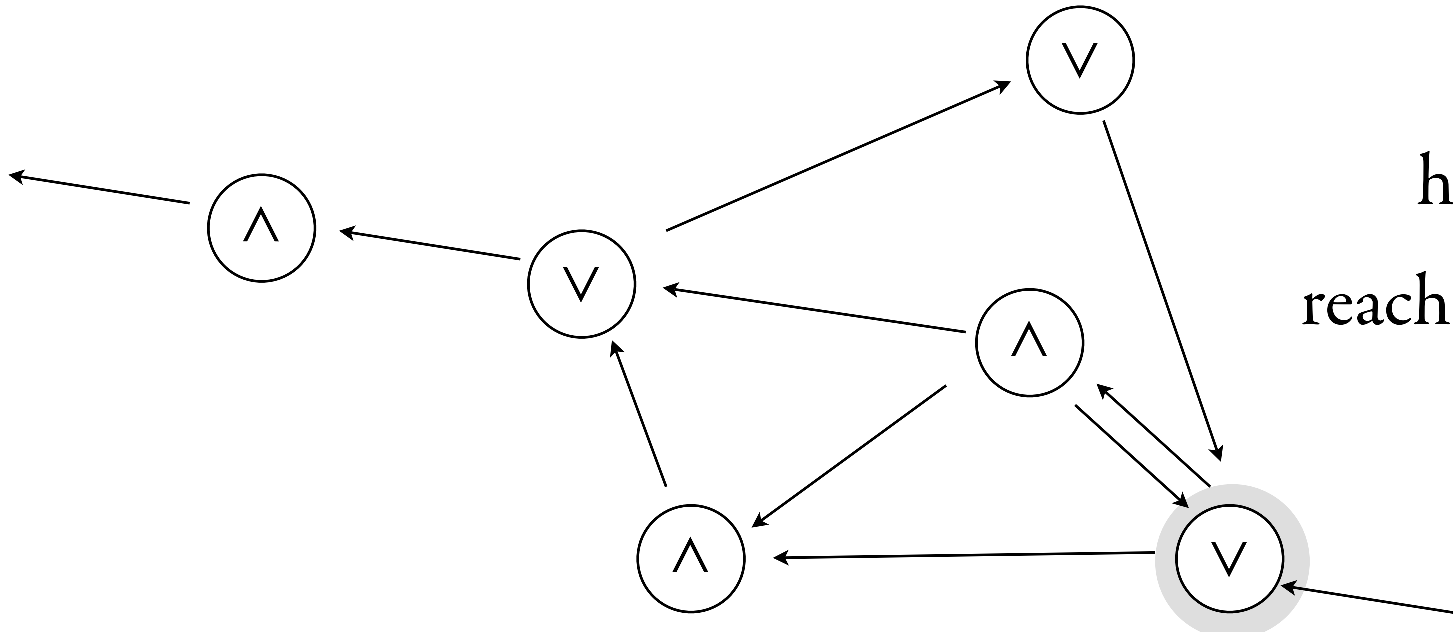
hardness:

reachability in AND-OR graphs

Tree automata emptiness can be tested in linear time.

Emptiness

Thm. Emptiness for tree automata is PTIME-complete.



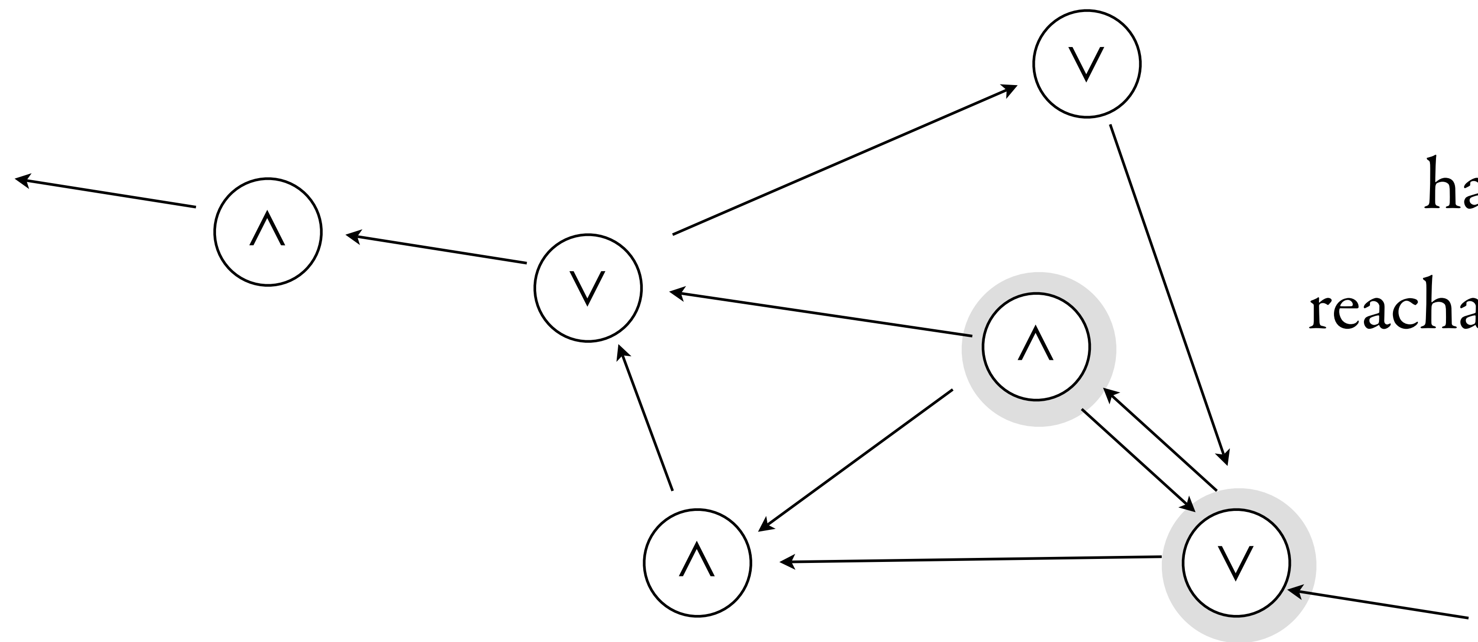
hardness:

reachability in AND-OR graphs

Tree automata emptiness can be tested in linear time.

Emptiness

Thm. Emptiness for tree automata is PTIME-complete.



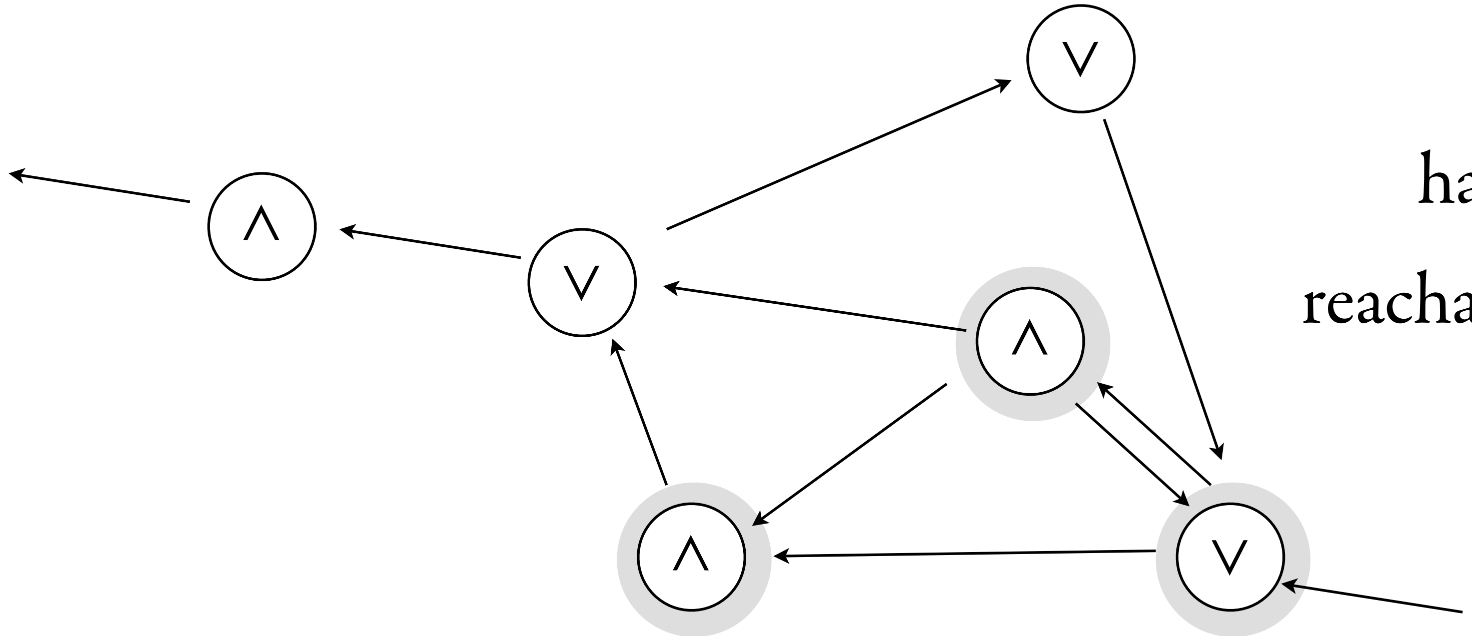
hardness:

reachability in AND-OR graphs

Tree automata emptiness can be tested in linear time.

Emptiness

Thm. Emptiness for tree automata is PTIME-complete.



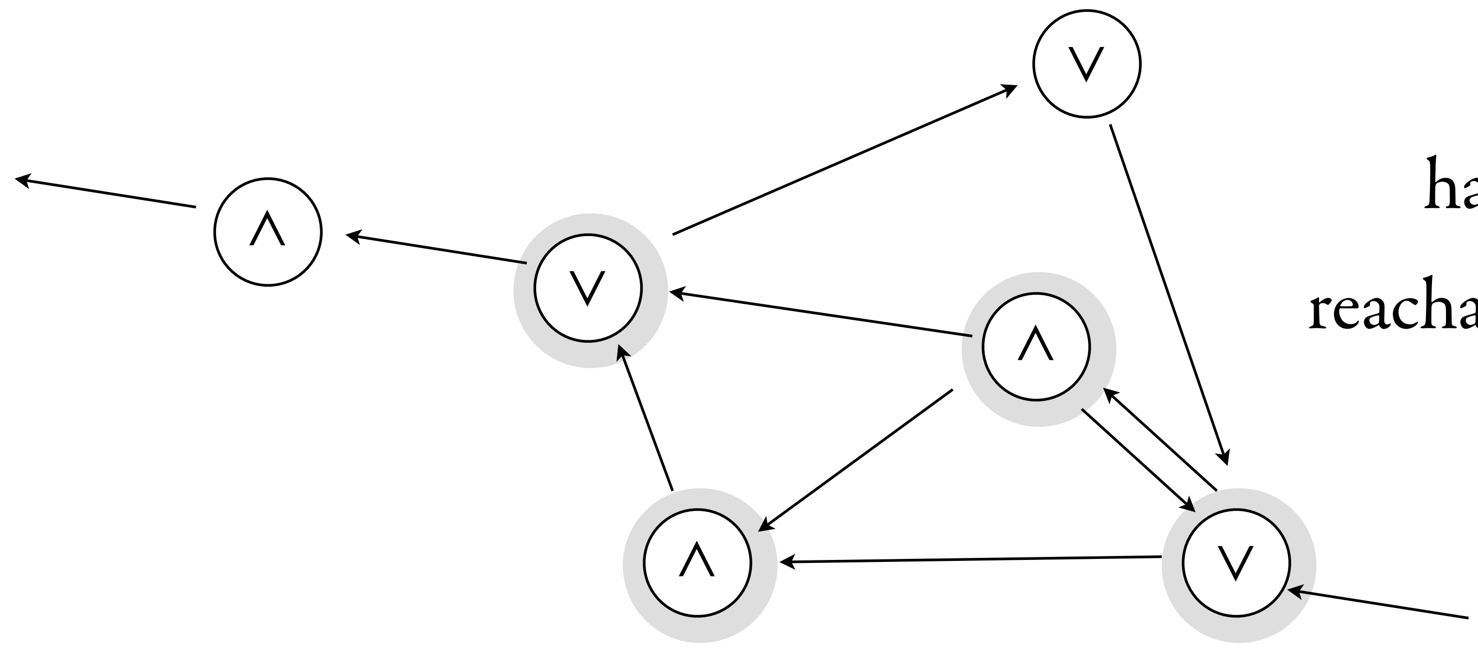
hardness:

reachability in AND-OR graphs

Tree automata emptiness can be tested in linear time.

Emptiness

Thm. Emptiness for tree automata is PTIME-complete.



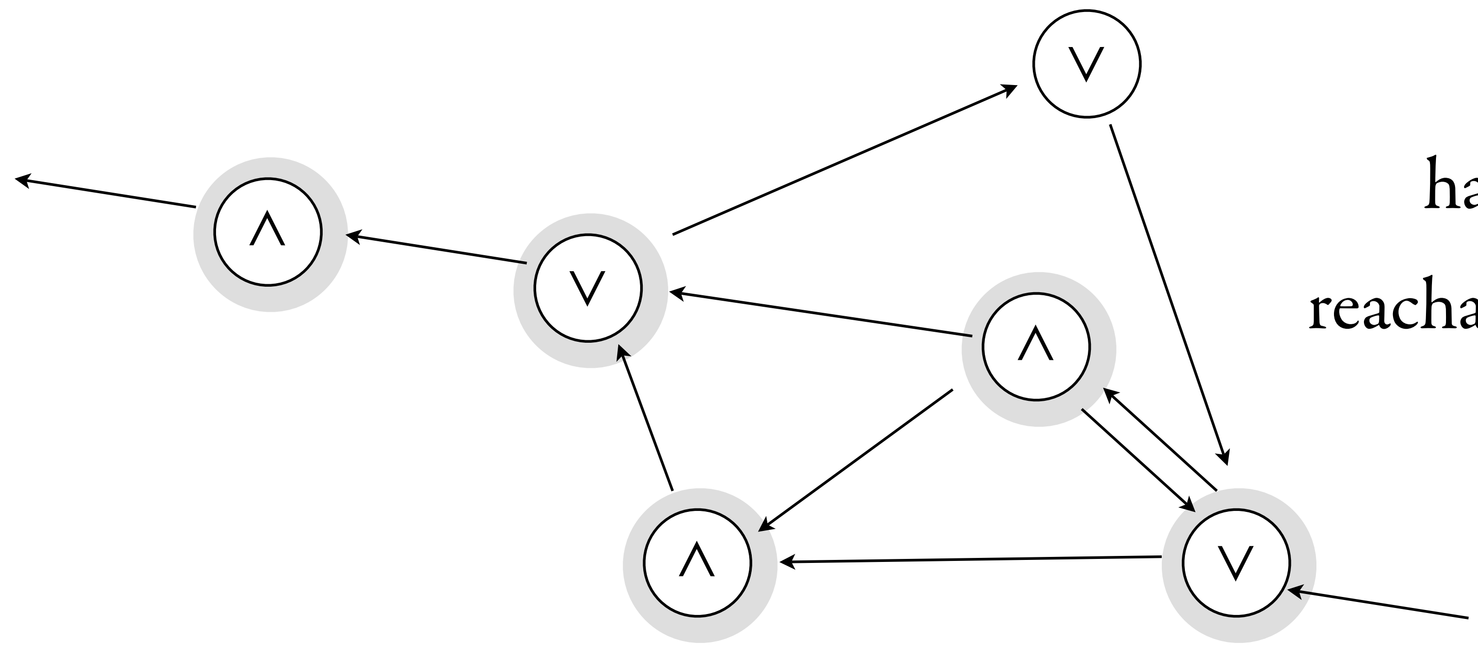
hardness:

reachability in AND-OR graphs

Tree automata emptiness can be tested in linear time.

Emptiness

Thm. Emptiness for tree automata is PTIME-complete.



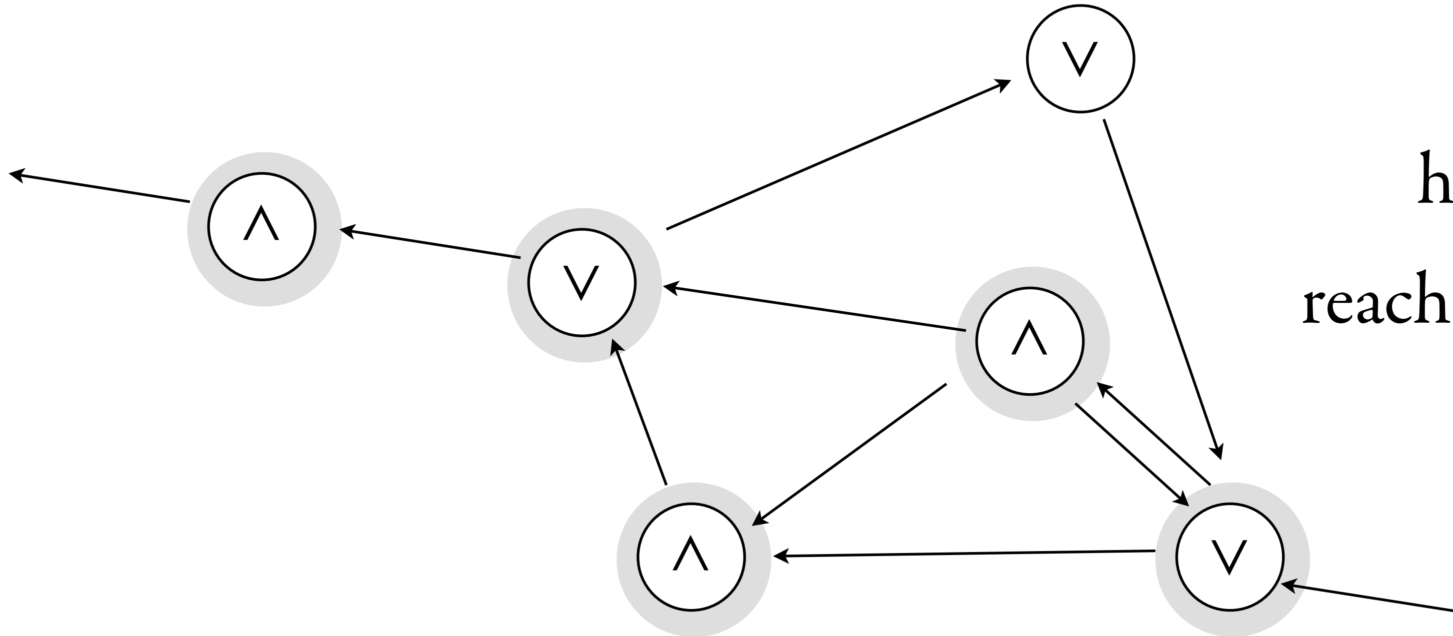
hardness:

reachability in AND-OR graphs

Tree automata emptiness can be tested in linear time.

Emptiness

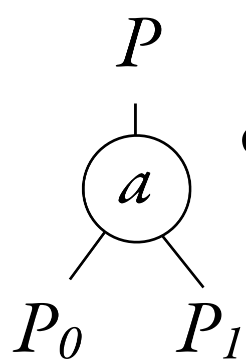
Thm. Emptiness for tree automata is PTIME-complete.



hardness:

reachability in AND-OR graphs

A transition

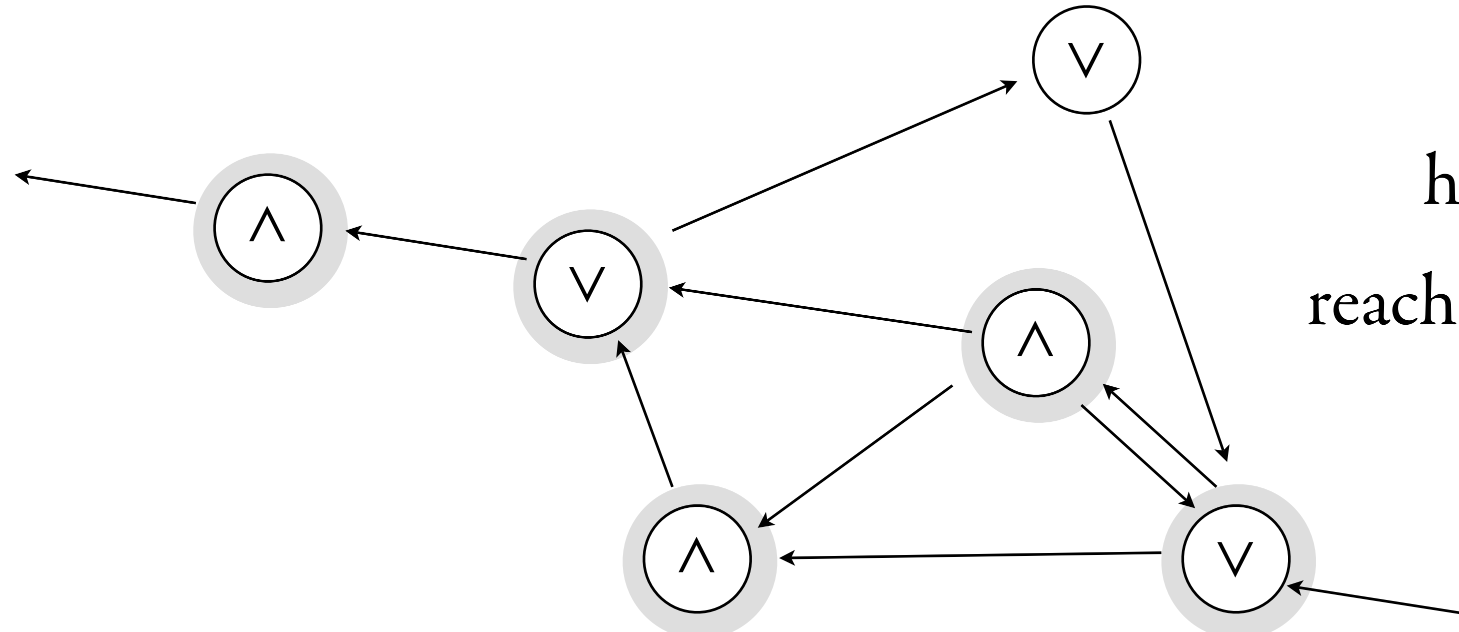


can be reached if both states P_0 and P_1 can be reached.

Tree automata emptiness can be tested in linear time.

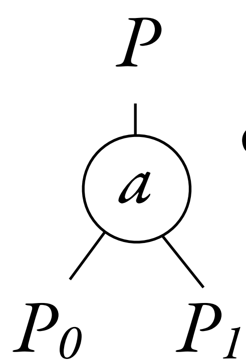
Emptiness

Thm. Emptiness for tree automata is PTIME-complete.

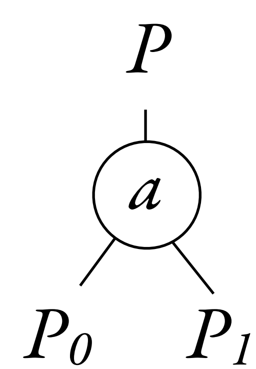


hardness:

reachability in AND-OR graphs

A transition 

can be reached if both states P_0 and P_1 can be reached.

A state p can be reached if some transition 

can be reached.

Tree automata emptiness can be tested in linear time.

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

$$2^{\,log(n)\,log(n)} = n^{\,log(n)}$$

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

we will keep a stack of
states of height $log(n)$
and one pointer to a node.

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

we will keep a stack of states of height $log(n)$ and one pointer to a node.

consider a balanced binary tree.

Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

we will keep a stack of
states of height $log(n)$
and one pointer to a node.

consider a balanced
binary tree.

we visit nodes in a DFS

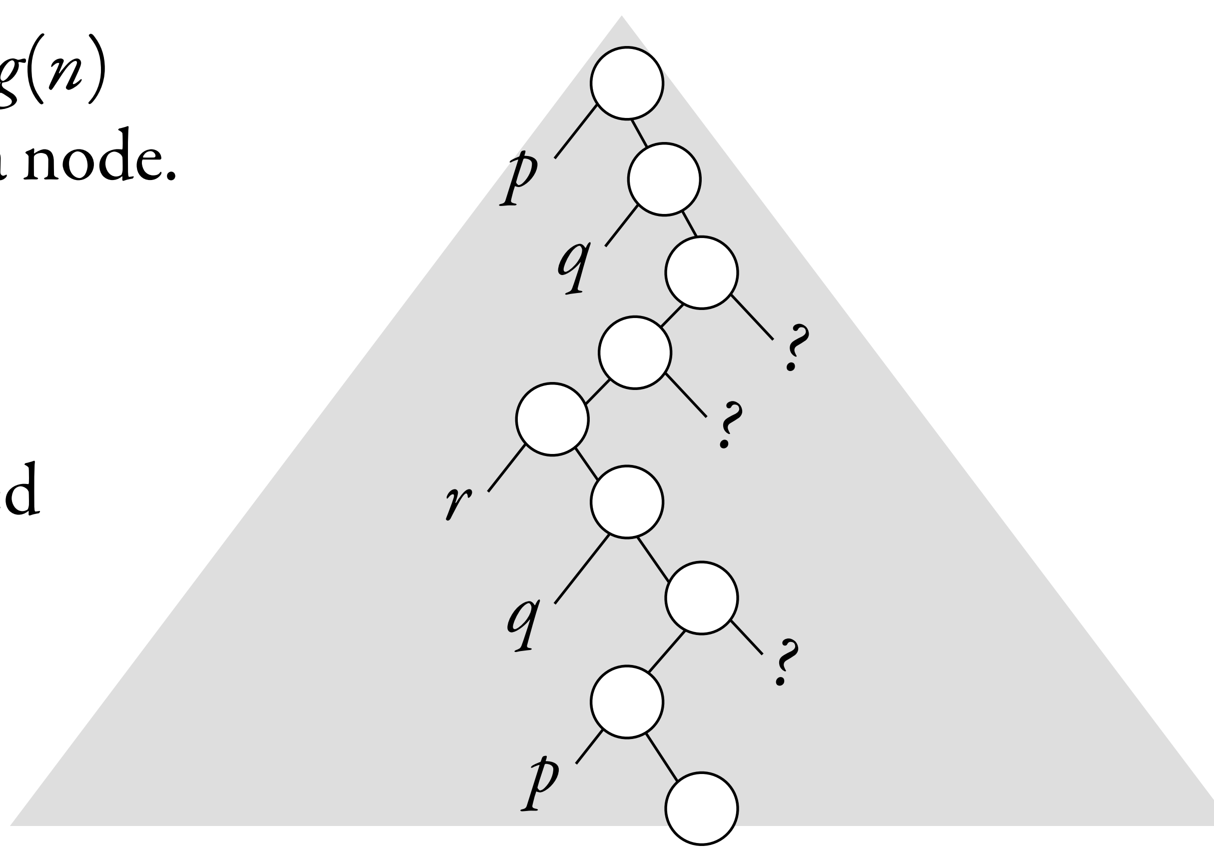
Membership

Fix a tree automaton A (deterministic, bottom up).
Input: tree t with n nodes.
Question: does A accept t?

This problem can be solved in $log(n)$ space, i.e. in LOGSPACE.

we will keep a stack of states of height $log(n)$ and one pointer to a node.

consider a balanced binary tree.



we visit nodes in a DFS

Universality

Universality

Thm. Universality for nondeterministic tree automata is EXPTIME-complete.

Universality

Thm. Universality for nondeterministic tree automata is EXPTIME-complete.

Upper bound.
Determinize automaton, complement it, check for emptiness.

Universality

Thm. Universality for nondeterministic tree automata is EXPTIME-complete.

Upper bound.
Determinize automaton, complement it, check for emptiness.

Lower bound.
Similar to

Universality

Thm. Universality for nondeterministic tree automata is EXPTIME-complete.

Upper bound.
Determinize automaton, complement it, check for emptiness.

Lower bound.
Similar to

> **Thm.** Universality for nondeterministic word automata is PSPACE-hard.

Thm. Universality for nondeterministic word automata is PSPACE-hard.

Thm. Universality for nondeterministic word automata is PSPACE-hard.

Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.

iff

The automaton A_n rejects some word.

Thm. Universality for nondeterministic word automata is PSPACE-hard.

Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.

iff

The automaton A_n rejects some word.

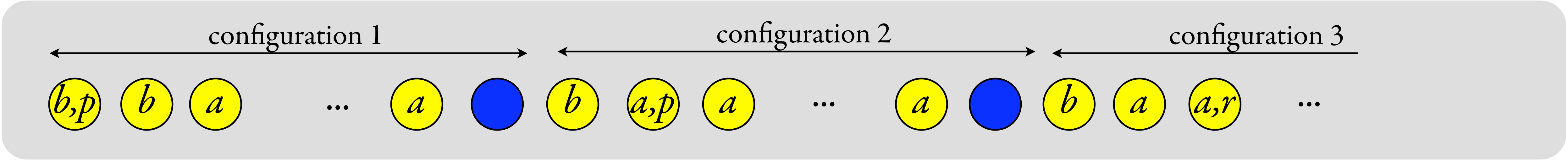
A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.

Thm. Universality for nondeterministic word automata is PSPACE-hard.

Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.
iff
The automaton A_n rejects some word.

A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.

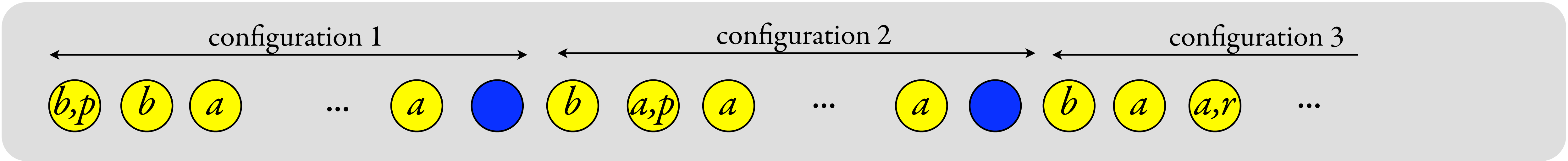


Thm. Universality for nondeterministic word automata is PSPACE-hard.

Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.
iff
The automaton A_n rejects some word.

A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.



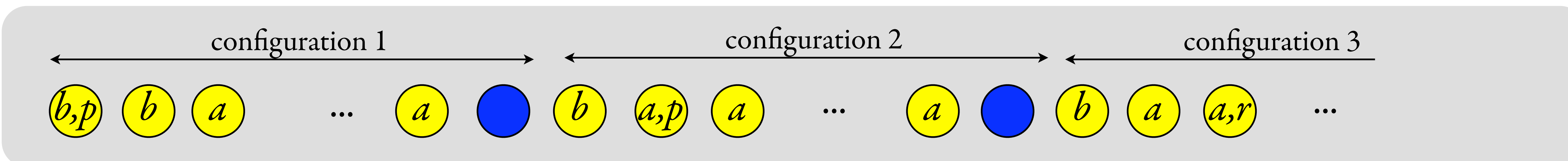
incorrect syntax:

Thm. Universality for nondeterministic word automata is PSPACE-hard.

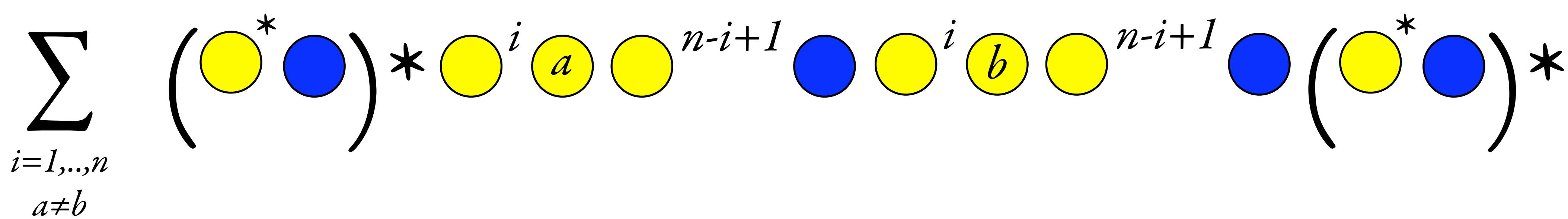
Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.
iff
The automaton A_n rejects some word.

A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.



incorrect syntax:



Thm. Universality for nondeterministic word automata is PSPACE-hard.

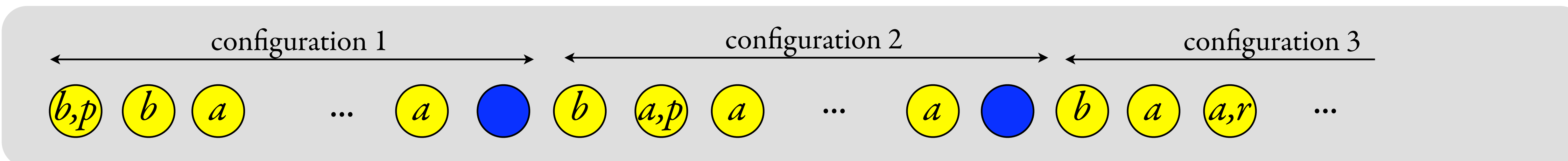
Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.

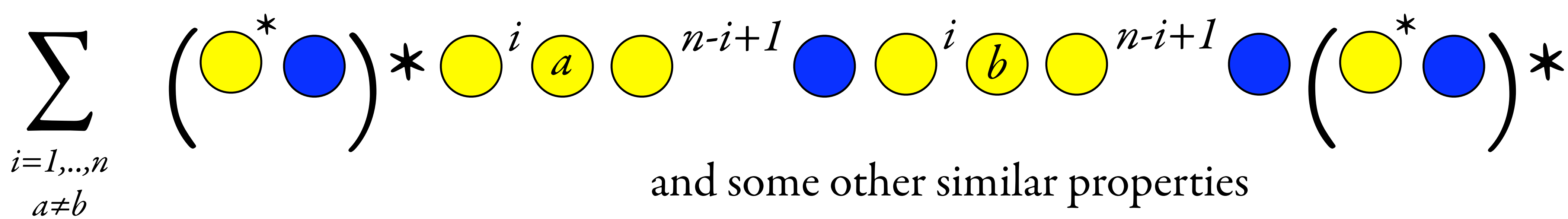
iff

The automaton A_n rejects some word.

A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.



incorrect syntax:



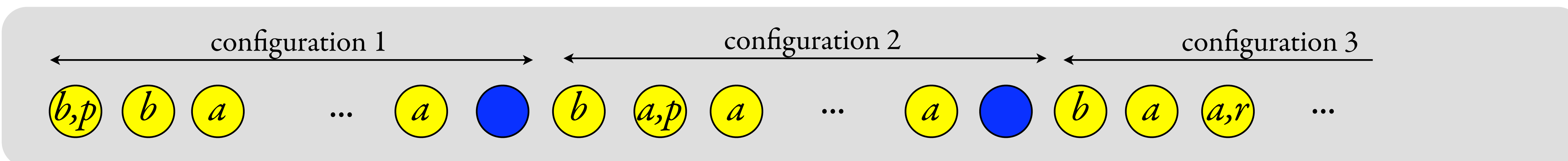
and some other similar properties

Thm. Universality for nondeterministic word automata is PSPACE-hard.

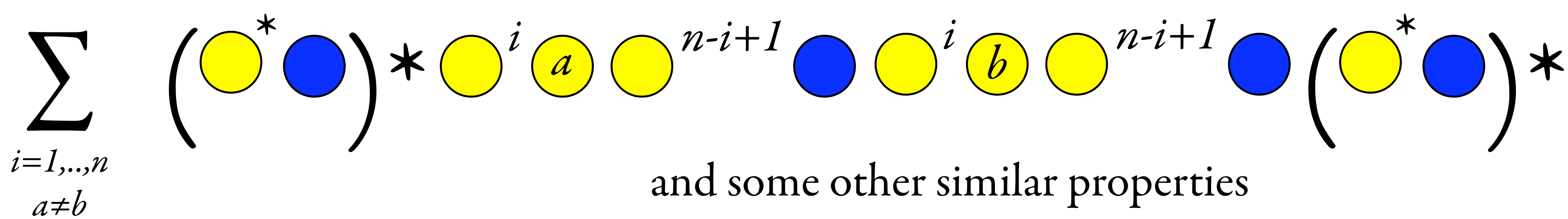
Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.
iff
The automaton A_n rejects some word.

A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.



incorrect syntax:



and some other similar properties

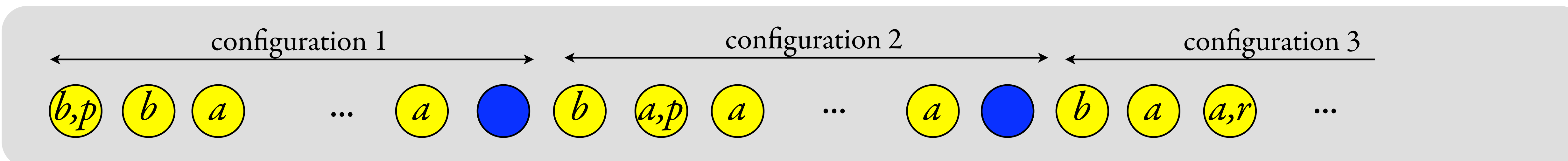
not accepting:

Thm. Universality for nondeterministic word automata is PSPACE-hard.

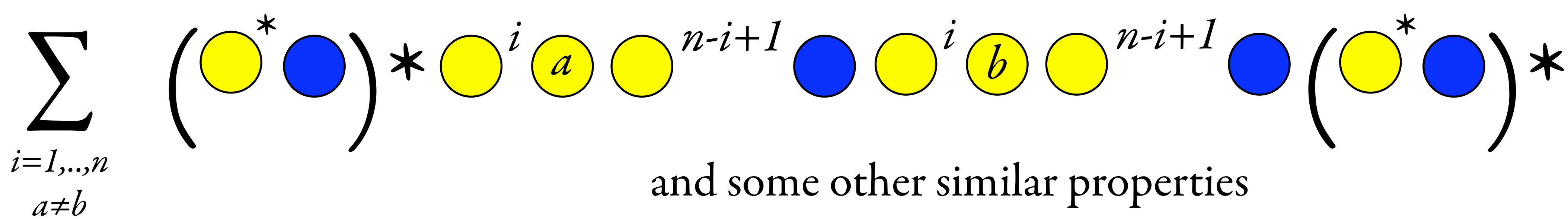
Proof. For every Turing machine M, and every n one can write a polynomial size automaton A_n with:

The machine M has a an accepting computation that uses n memory cells.
iff
The automaton A_n rejects some word.

A_n accepts incorrect encodings, and correct encodings of nonaccepting runs.

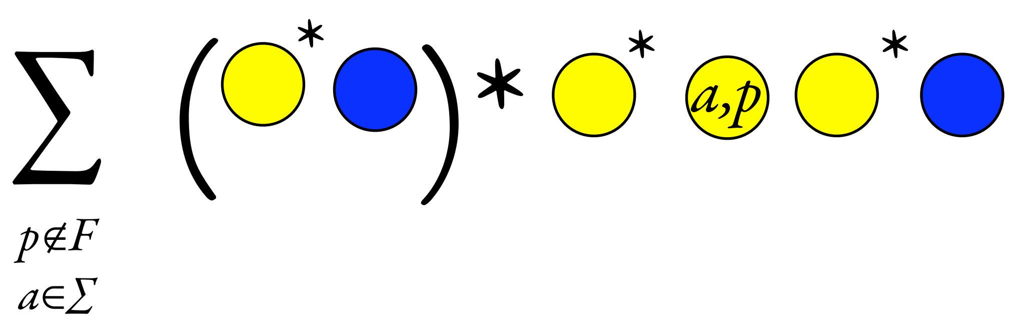


incorrect syntax:



and some other similar properties

not accepting:



Universality for nondeterminstic tree automata is EXPTIME-hard.

Universality for nondeterminstic tree automata is EXPTIME-hard.

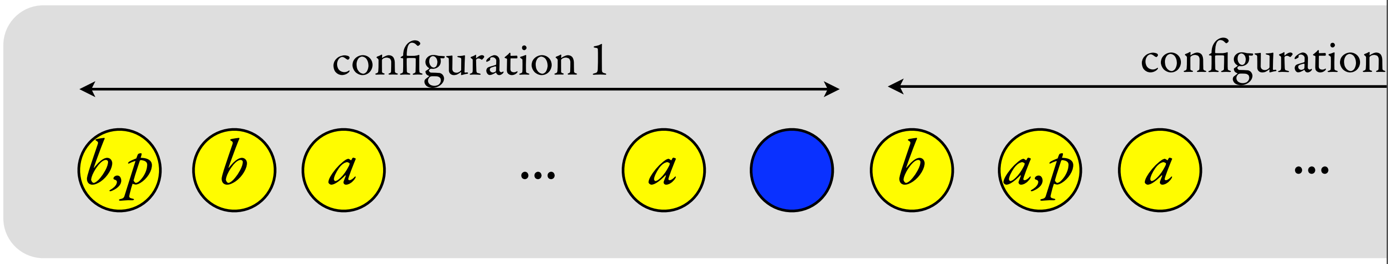
Reduction from APSPACE.

Universality for nondeterminstic tree automata is EXPTIME-hard.
Reduction from APSPACE.

instead of a computation word...

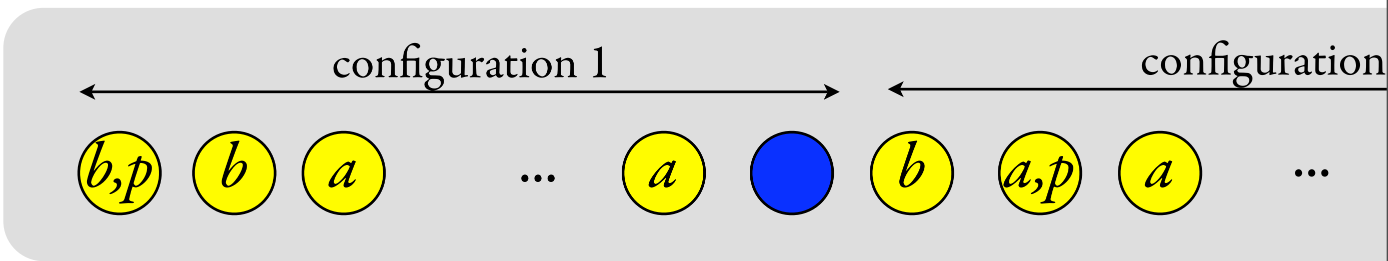
Universality for nondeterminstic tree automata is EXPTIME-hard.

Reduction from APSPACE.



instead of a computation word...

Universality for nondeterminstic tree automata is EXPTIME-hard.
Reduction from APSPACE.

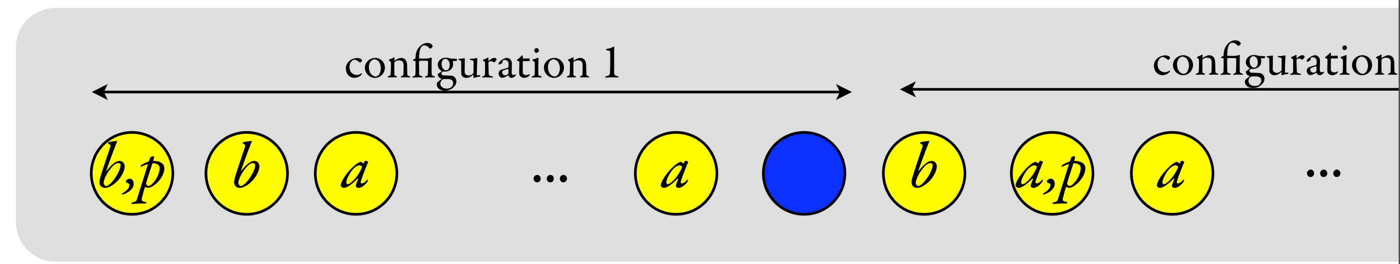


instead of a computation word...

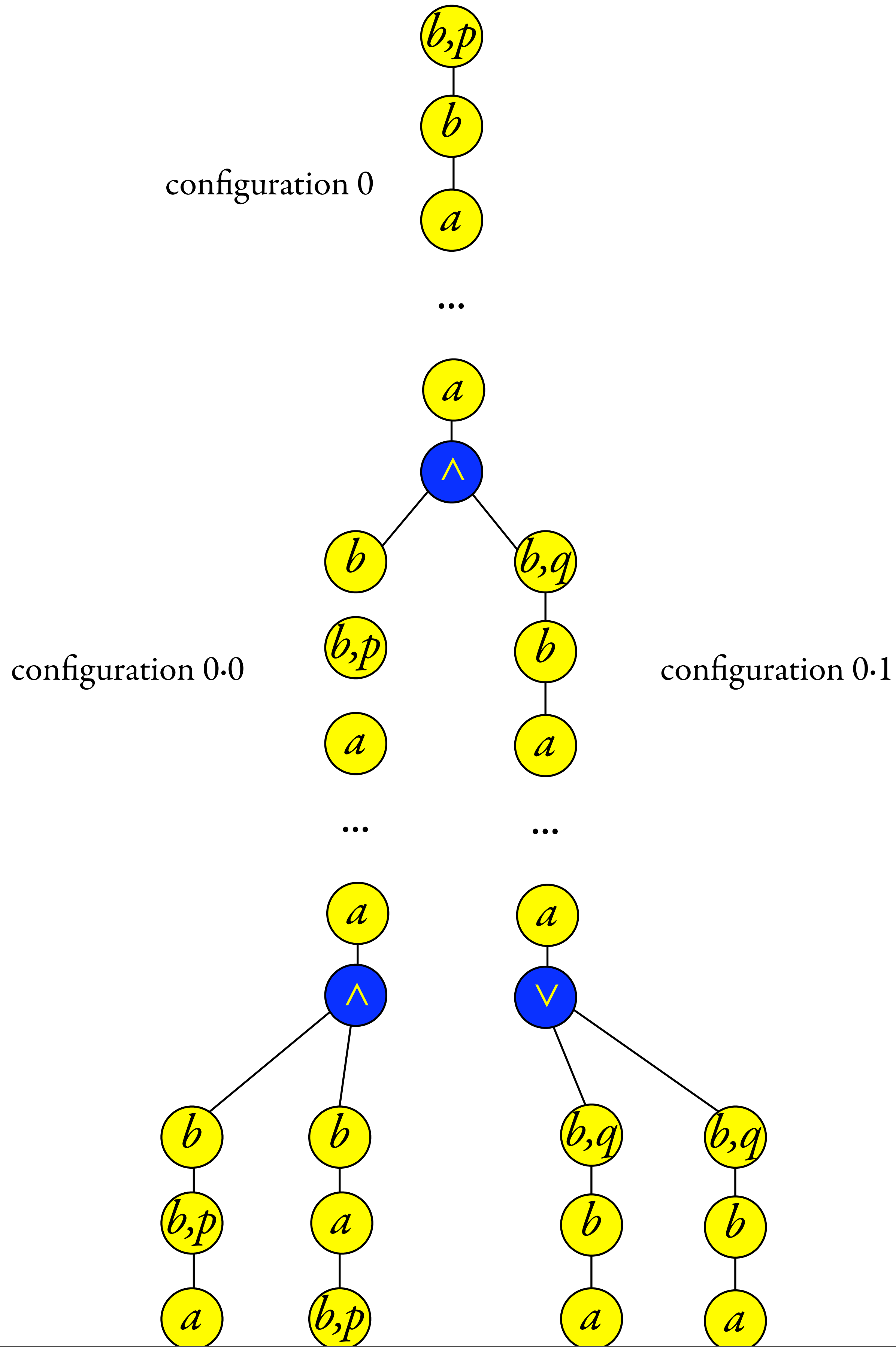
...we have a computation tree.

Universality for nondeterminstic tree automata is EXPTIME-hard.

Reduction from APSPACE.



instead of a computation word...



...we have a computation tree.