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

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$$L \in \text{TWA}$$
$$L \notin \text{DTWA}$$

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

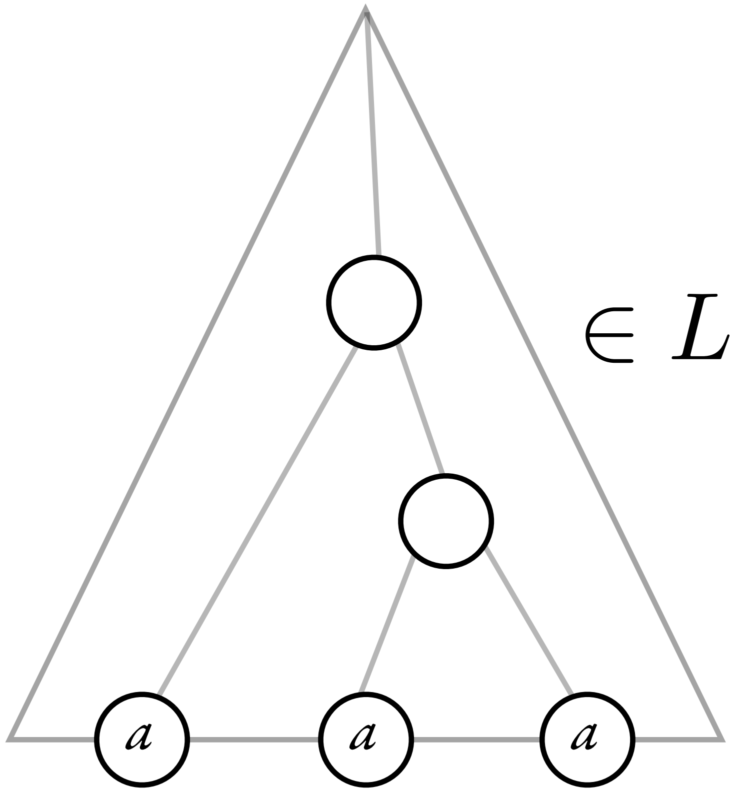
$L \notin$ DTWA

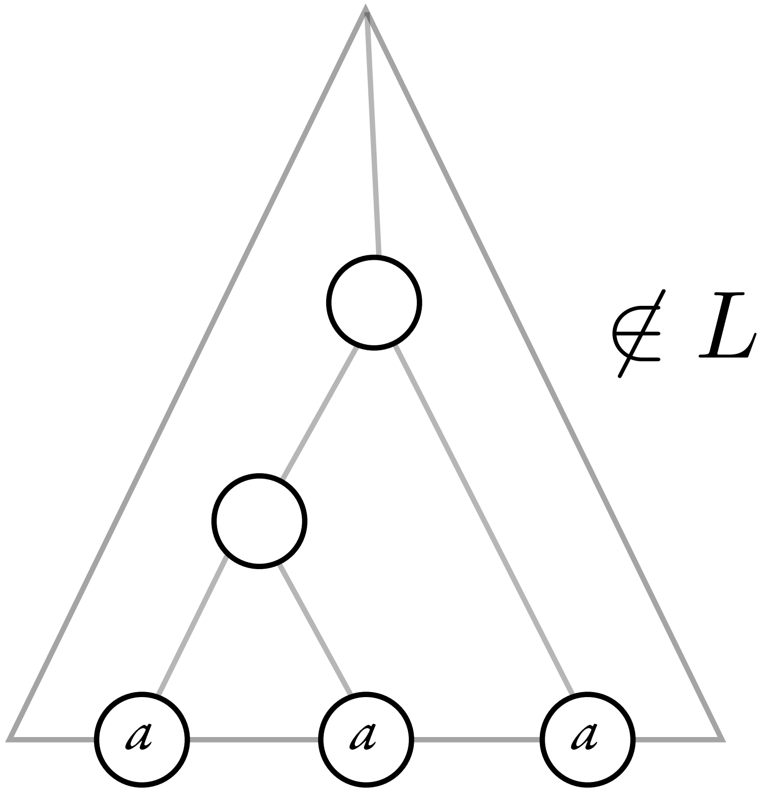
All nodes have label b, except three leaves with label a.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA

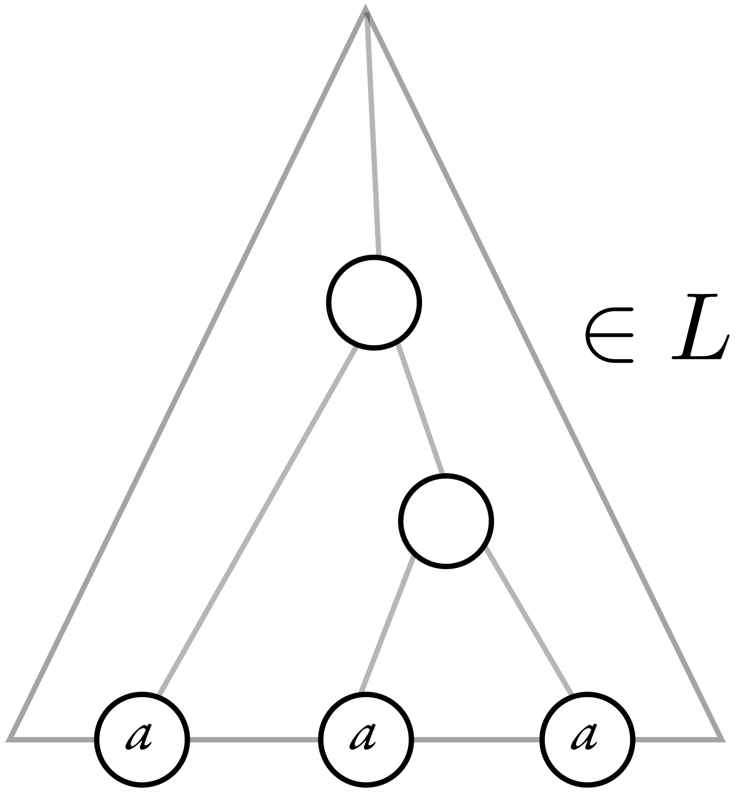


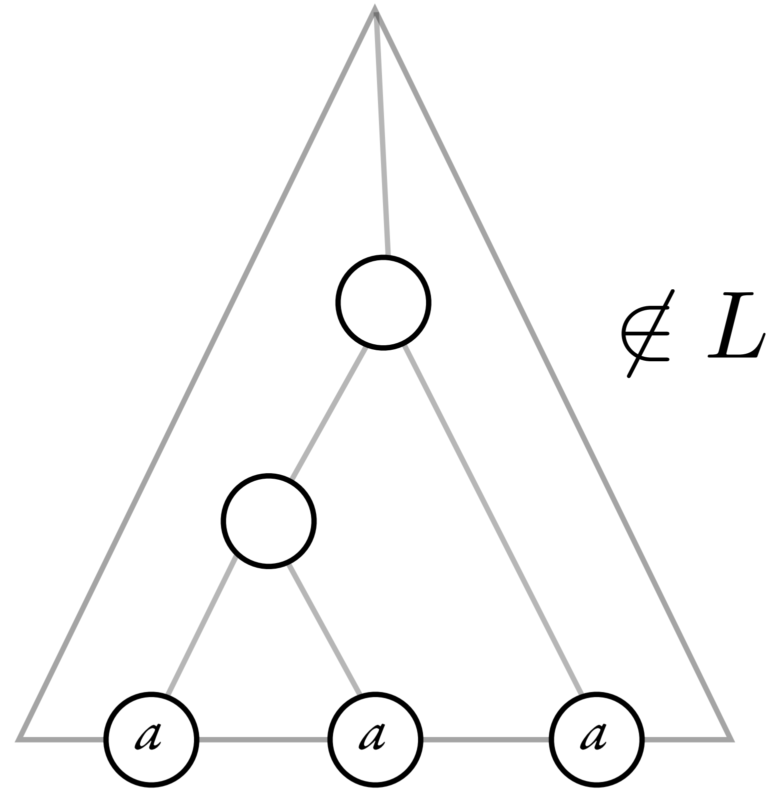


All nodes have label b, except three leaves with label a.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA
$L \notin$ DTWA

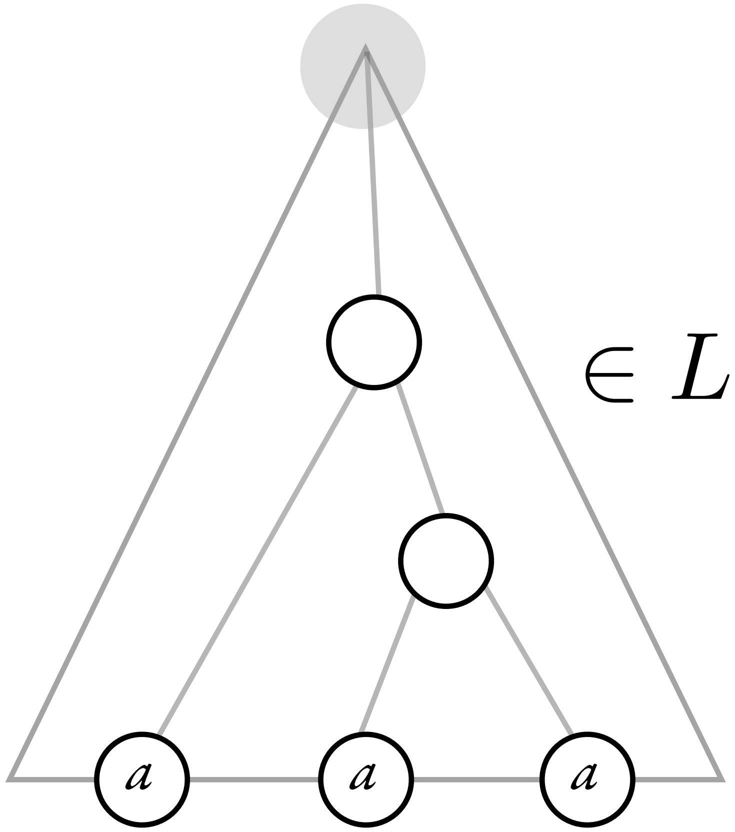




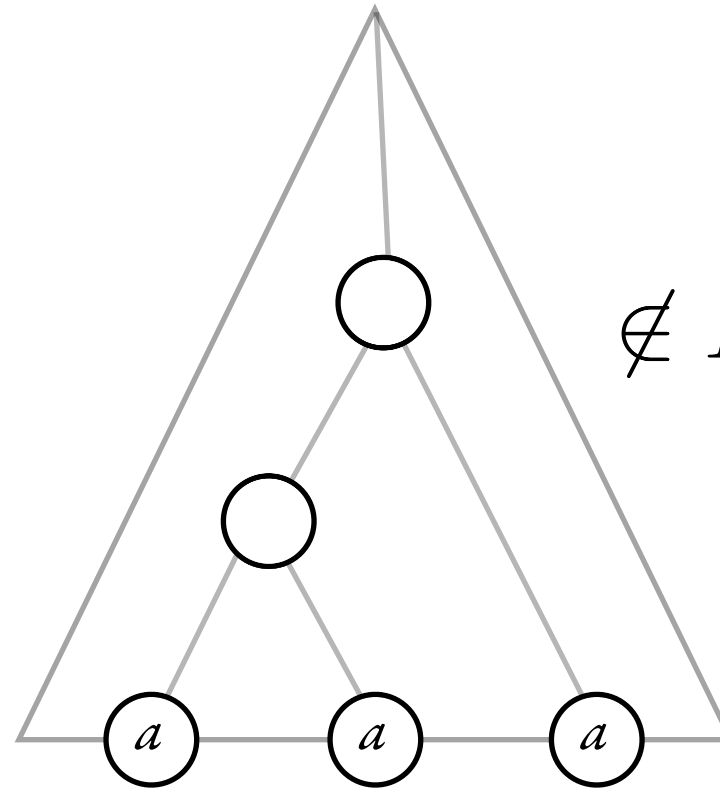
Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA



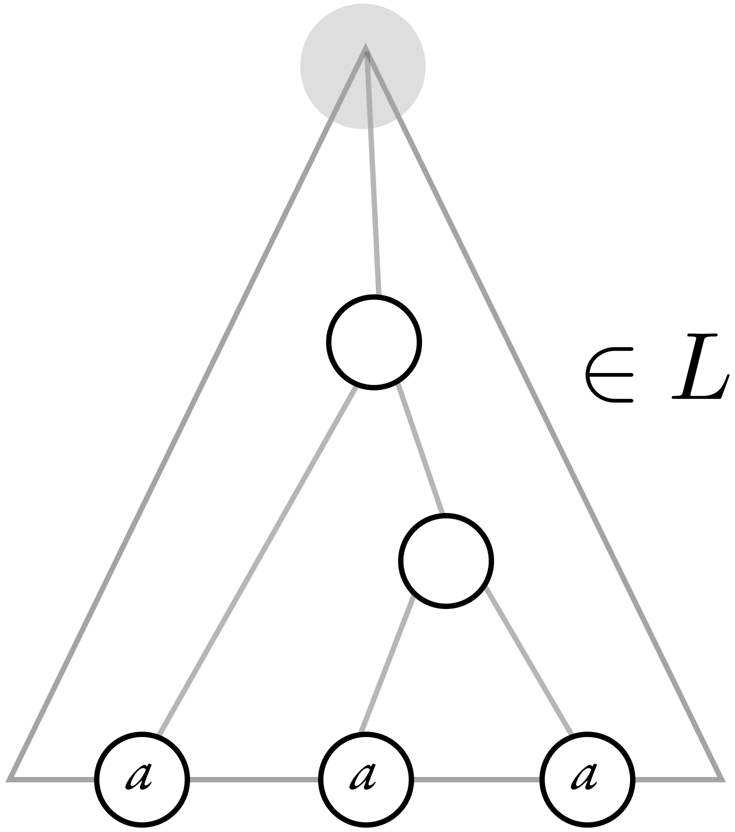
$\in L$



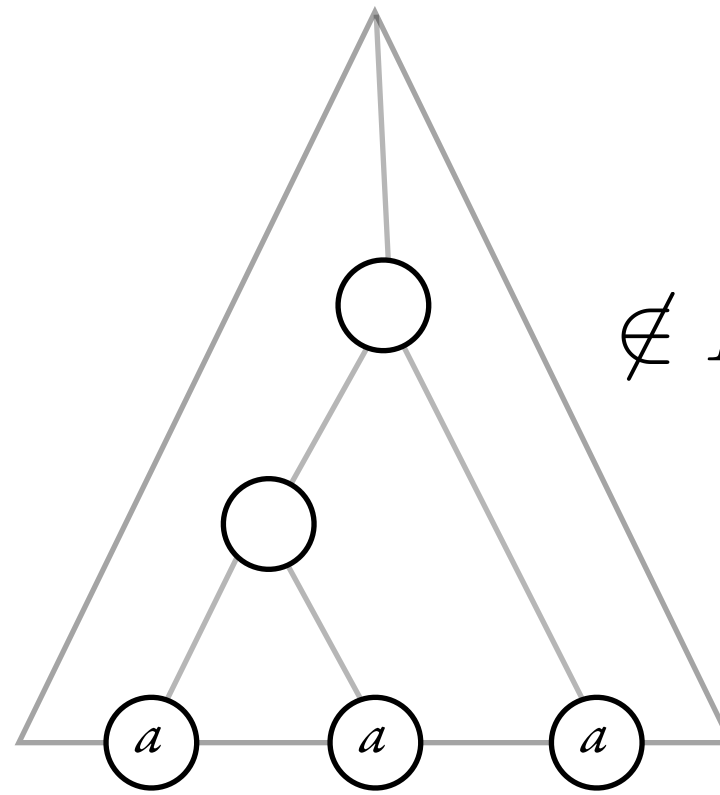
$\notin L$

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA
$L \notin$ DTWA



$\in L$



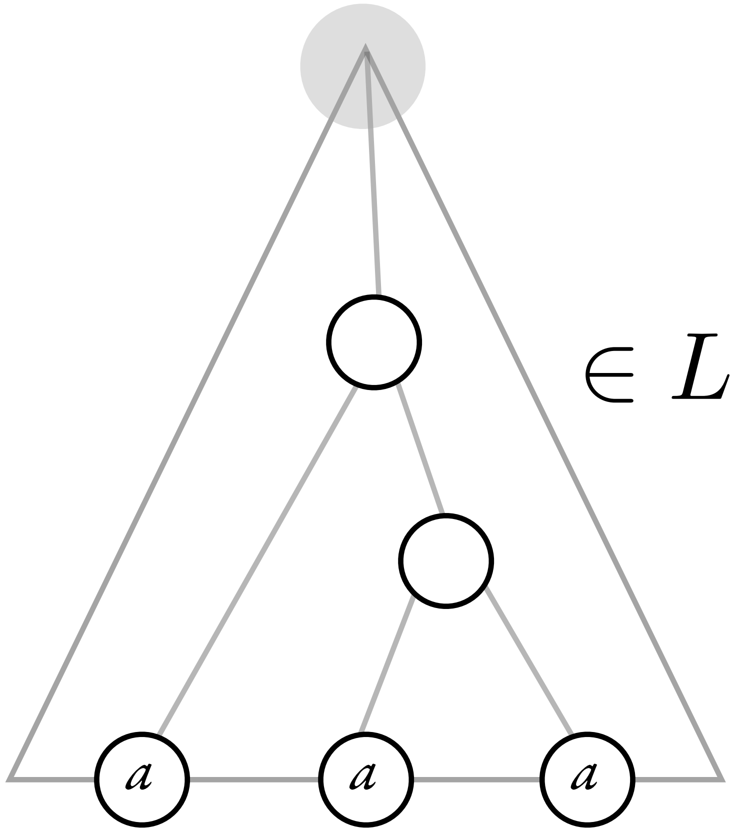
$\notin L$

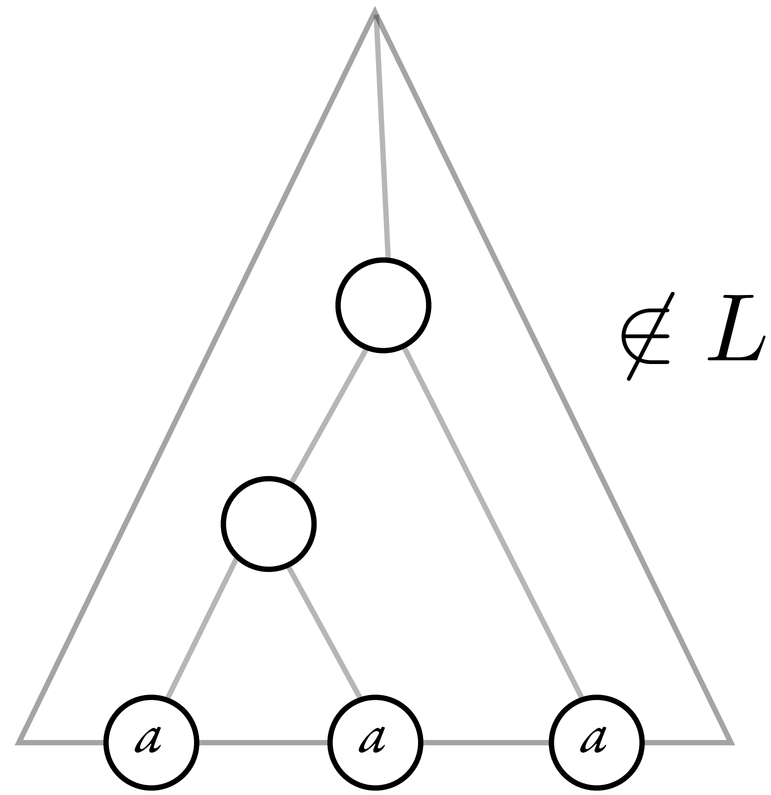
Using DFS, check that all nodes have *b*, except three leaves with *a*.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA





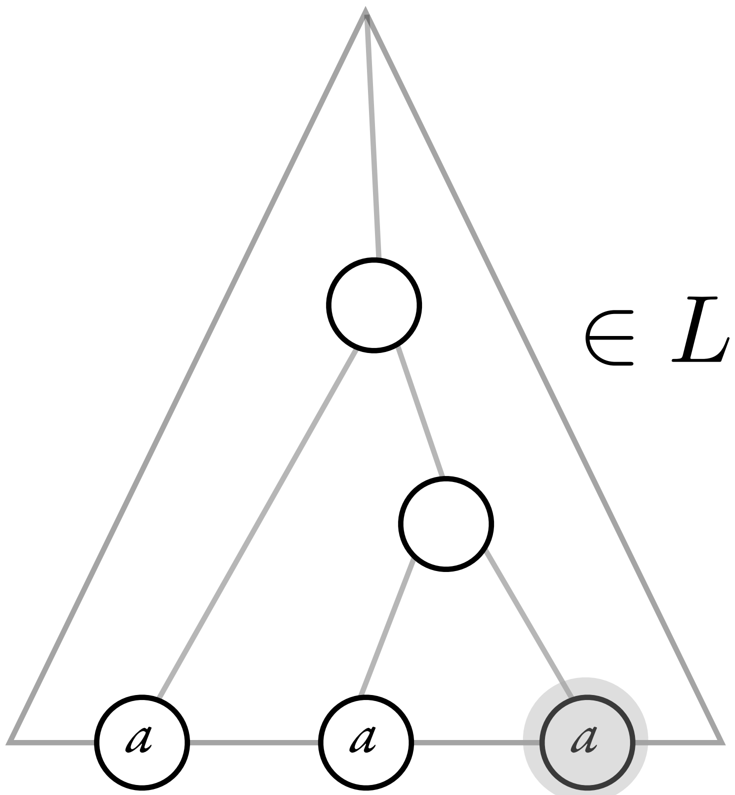
Using DFS, check that all nodes have *b*, except three leaves with *a*.

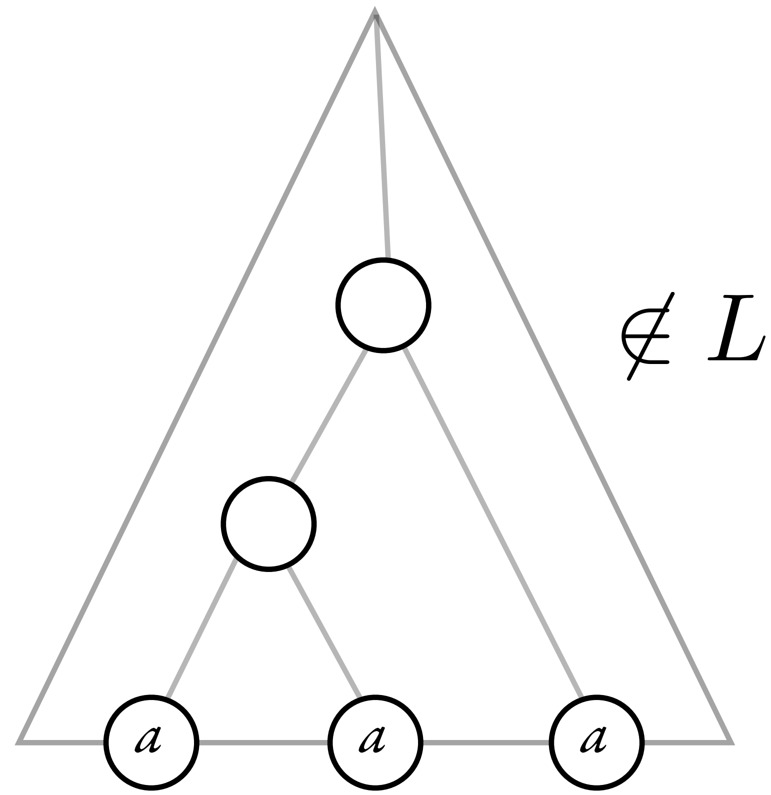
Go to the rightmost *a*.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA





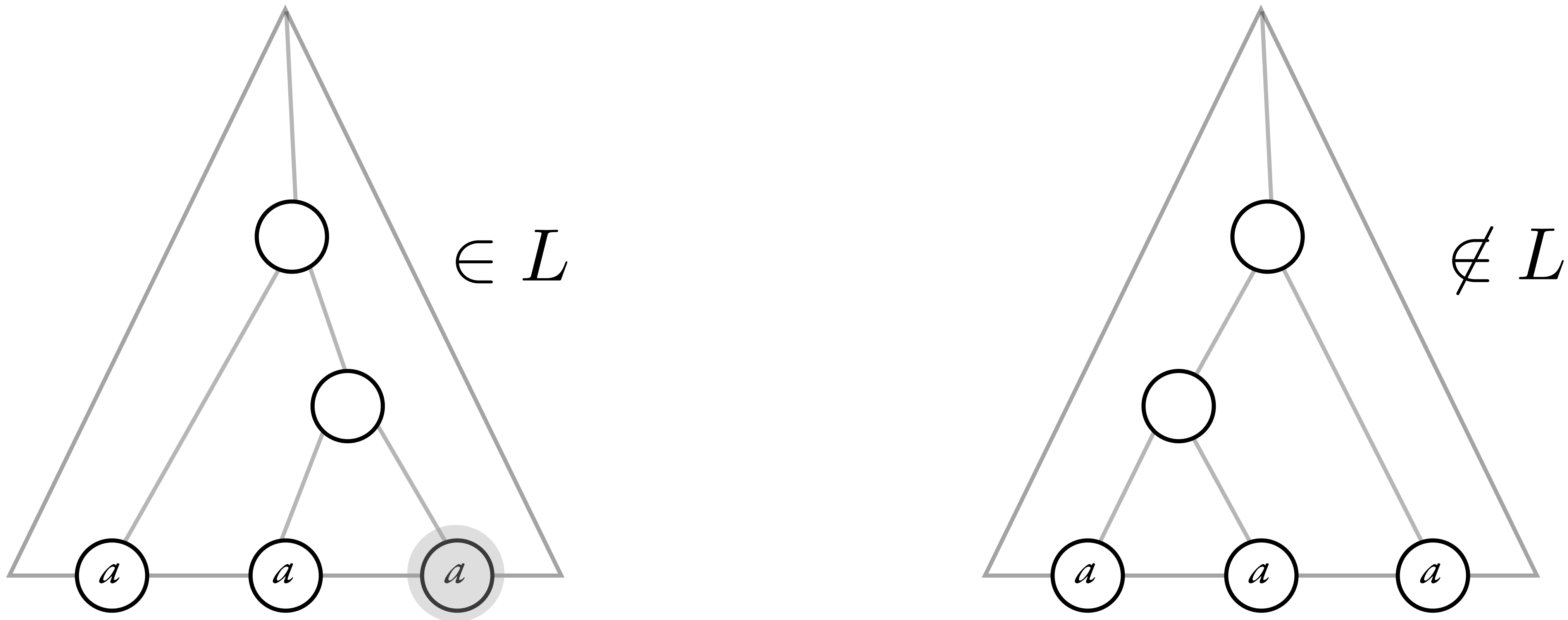
Using DFS, check that all nodes have *b*, except three leaves with *a*.

Go to the rightmost *a*.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA



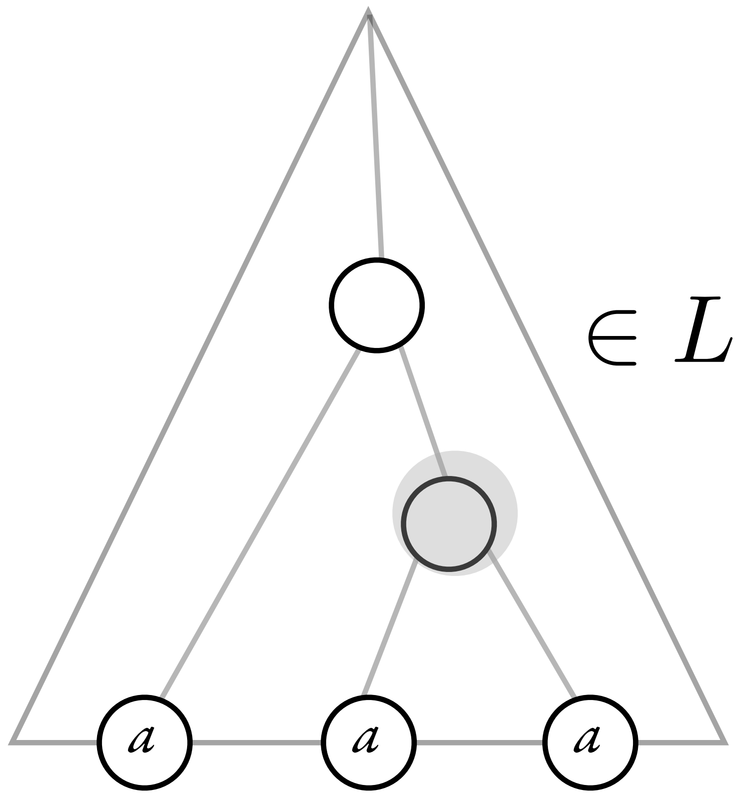
Using DFS, check that all nodes have *b*, except three leaves with *a*.

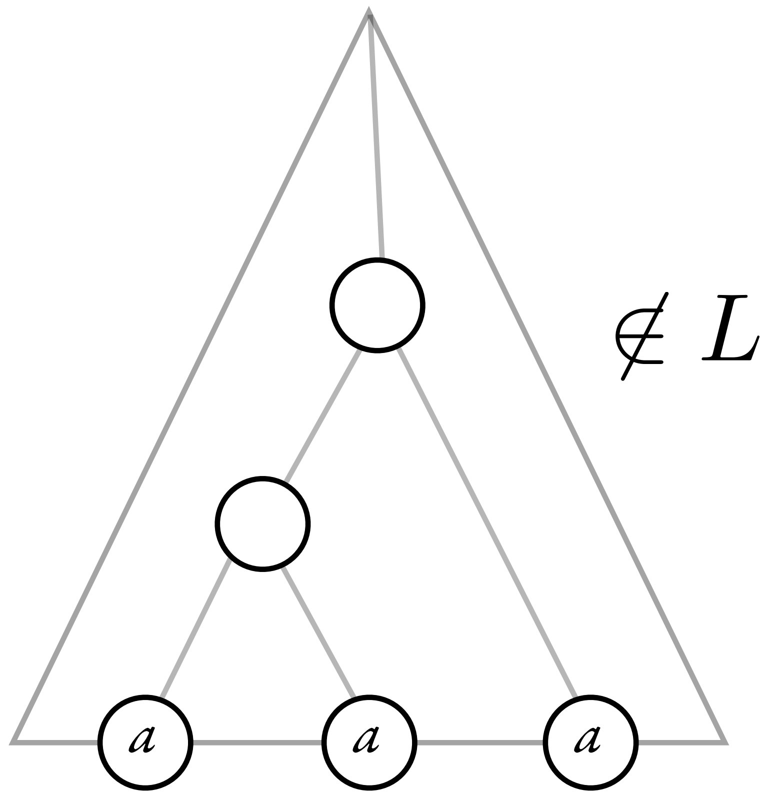
Go to the rightmost *a*.

Nondeterministically pick an ancestor.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA
$L \notin$ DTWA





Using DFS, check that all nodes have *b*, except three leaves with *a*.

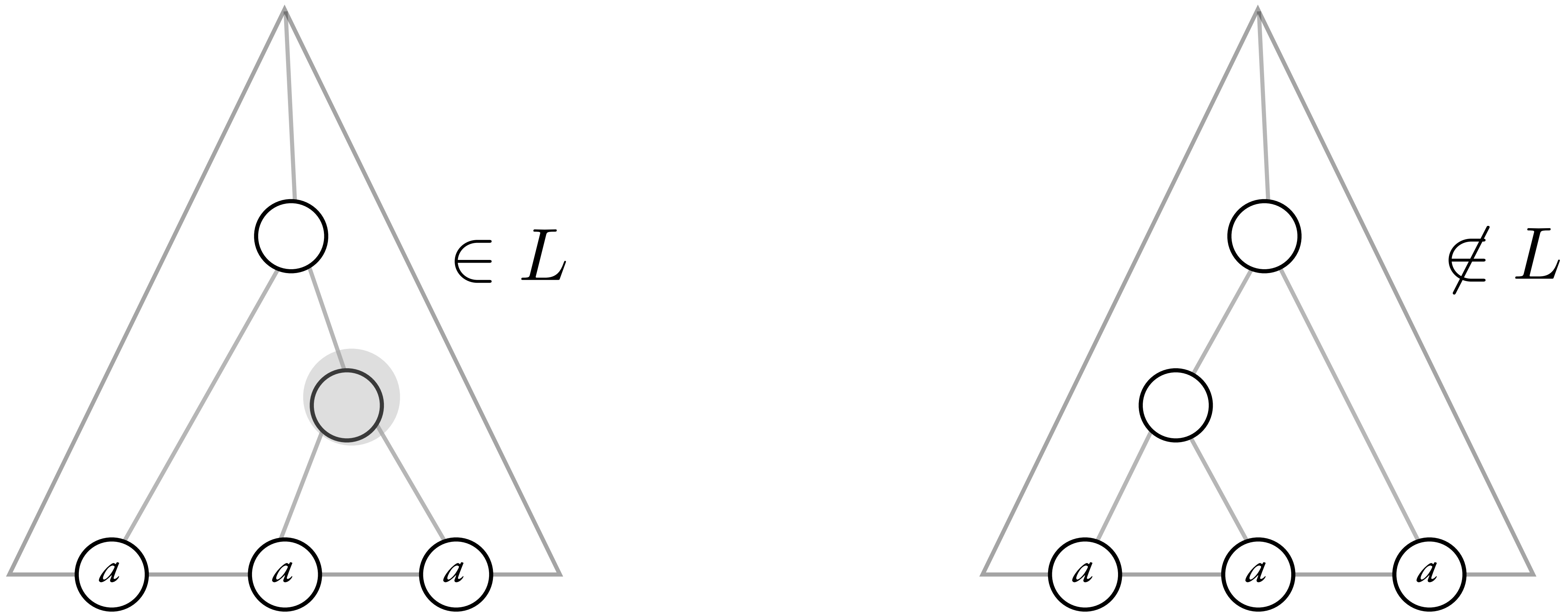
Go to the rightmost *a*.

Nondeterministically pick an ancestor.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA



Using DFS, check that all nodes have *b*, except three leaves with *a*.

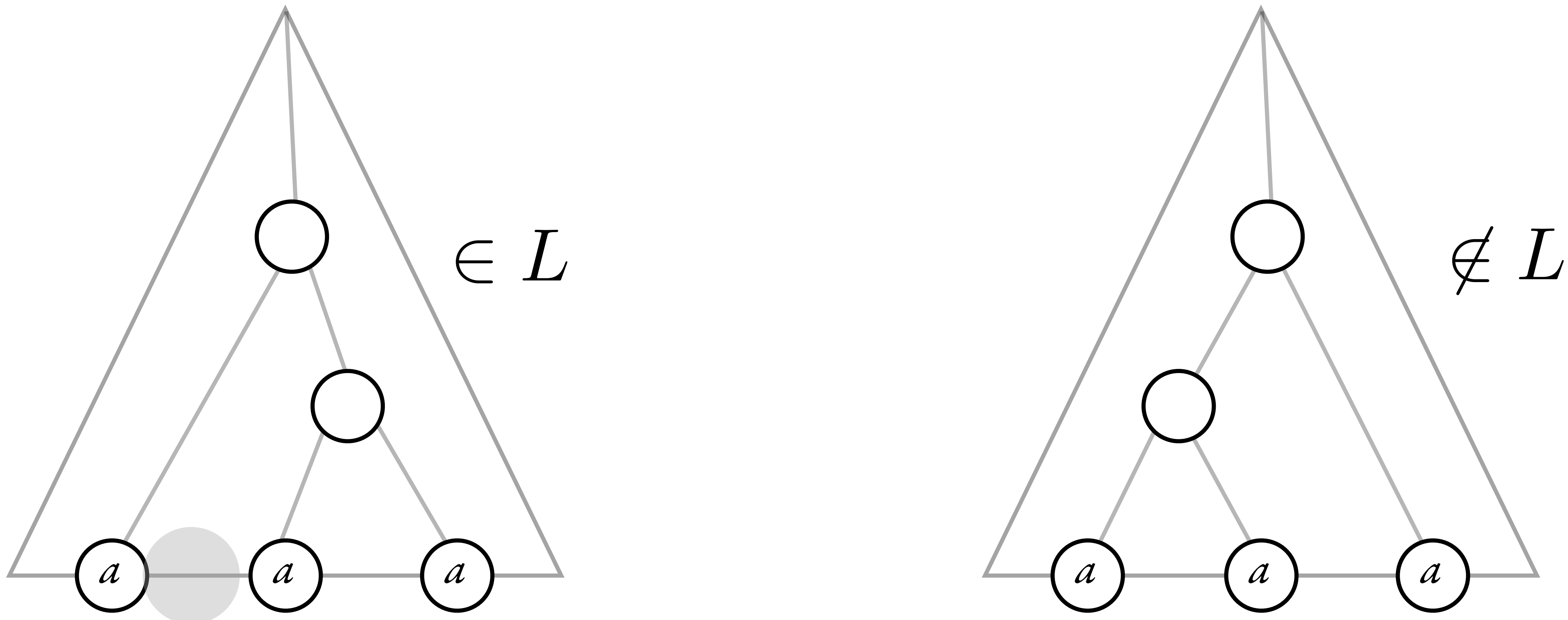
Go to the rightmost *a*.

Nondeterministically pick an ancestor.

Descend to the leaf on the leftmost path.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA
$L \notin$ DTWA



Using DFS, check that all nodes have *b*, except three leaves with *a*.

Go to the rightmost *a*.

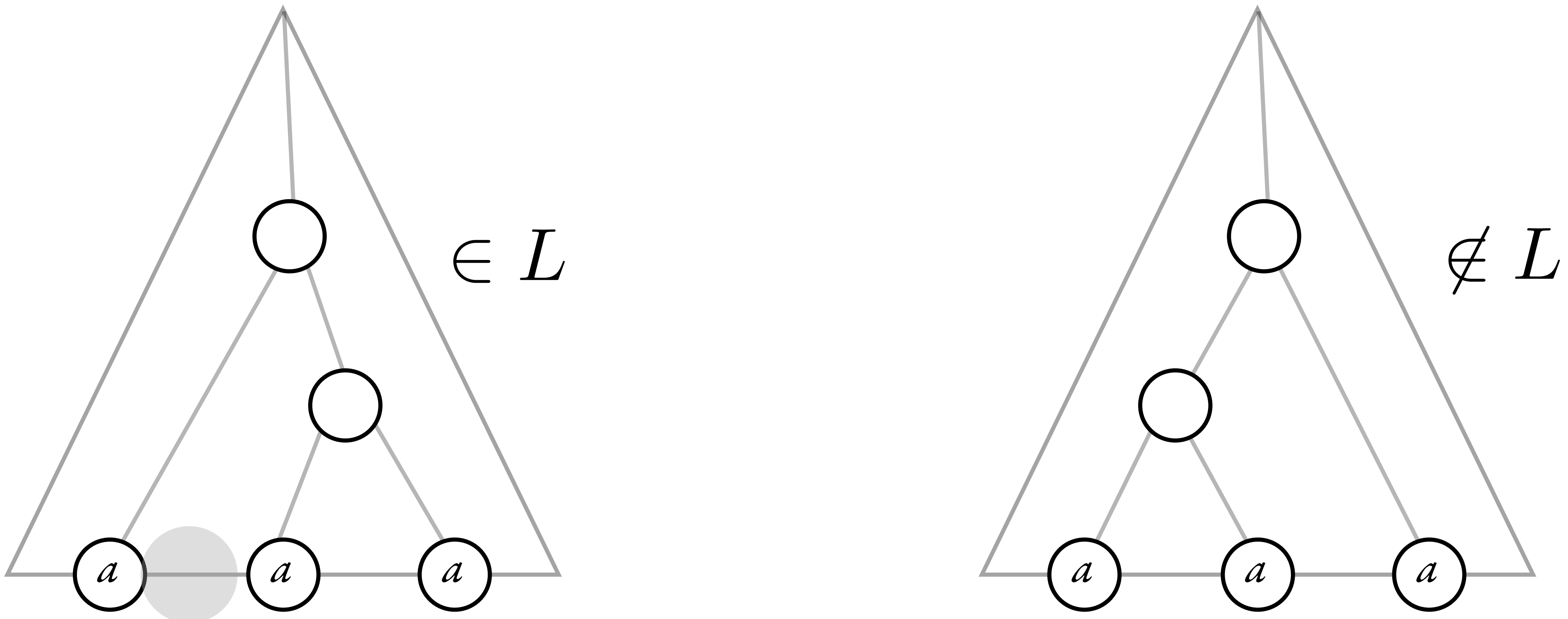
Nondeterministically pick an ancestor.

Descend to the leaf on the leftmost path.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA



Using DFS, check that all nodes have *b*, except three leaves with *a*.

Go to the rightmost *a*.

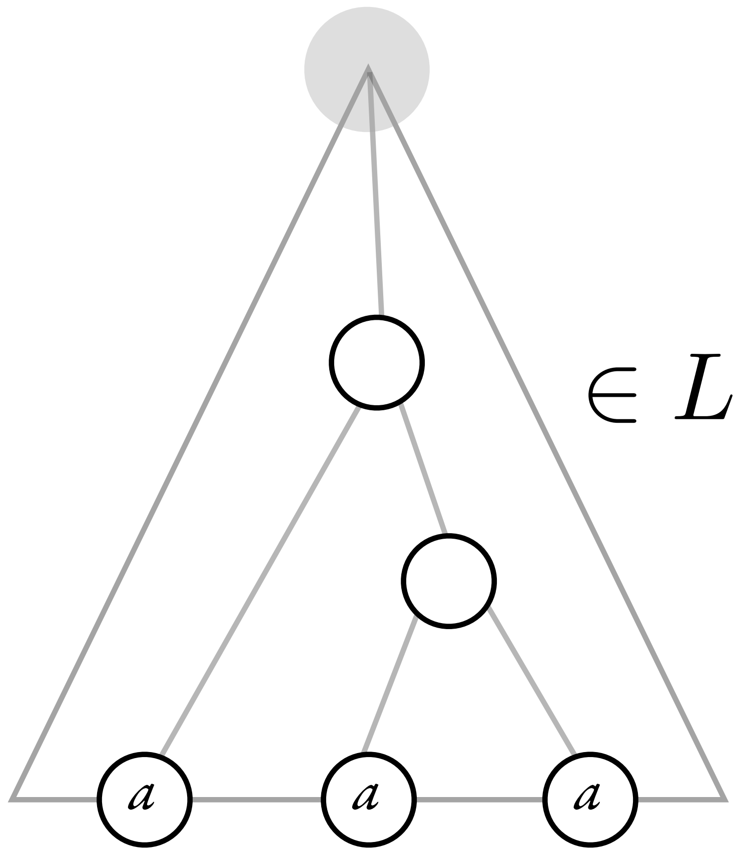
Nondeterministically pick an ancestor.

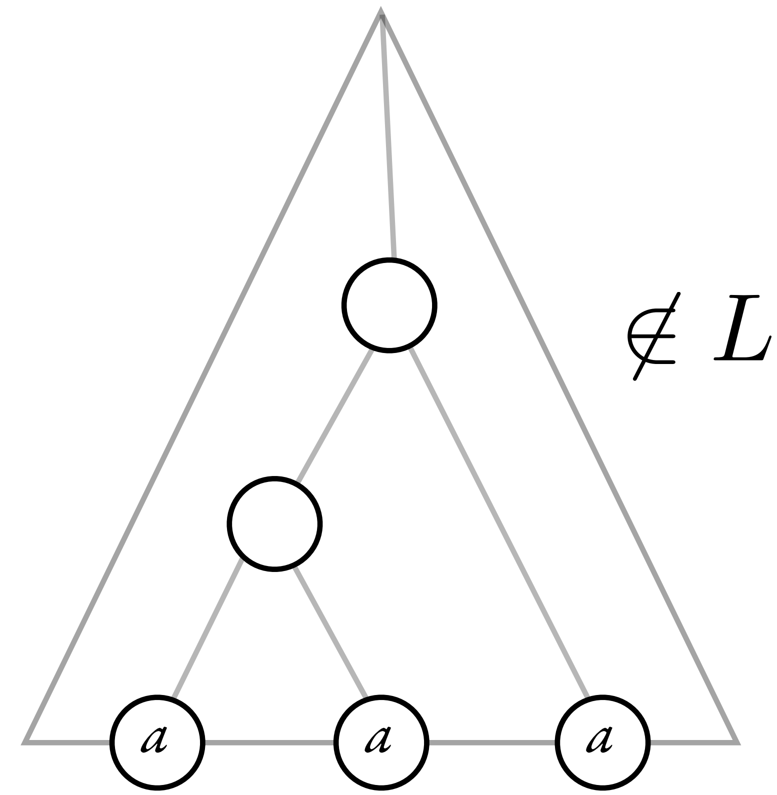
Descend to the leaf on the leftmost path.

Accept if there are exactly two *a*'s to the right.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA
$L \notin$ DTWA





Using DFS, check that all nodes have *b*, except three leaves with *a*.

Go to the rightmost *a*.

Nondeterministically pick an ancestor.

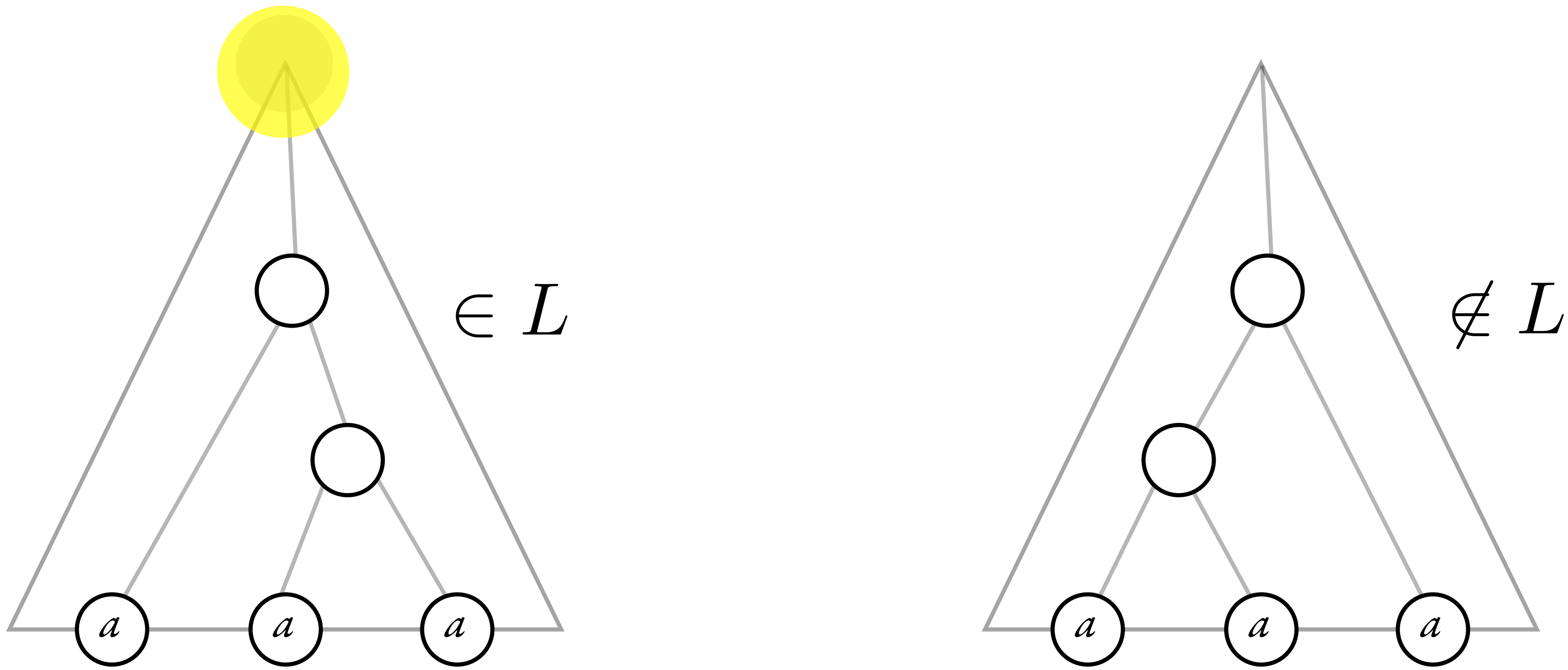
Descend to the leaf on the leftmost path.

Accept if there are exactly two *a*'s to the right.

Theorem (B., Colcombet '04)
Tree-walking automata cannot be determinized.

$L \in$ TWA

$L \notin$ DTWA



Using DFS, check that all nodes have *b*, except three leaves with *a*.

Go to the rightmost *a*.

Nondeterministically pick an ancestor.

Descend to the leaf on the leftmost path.

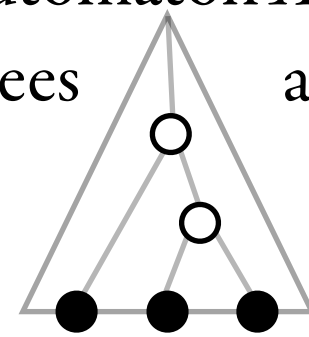
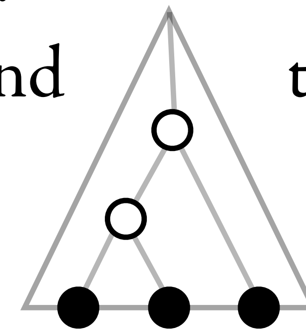
Accept if there are exactly two *a*'s to the right.

Goal: No deterministic tree-walking automaton recognizes the language L.

Goal: No deterministic tree-walking automaton recognizes the language L.

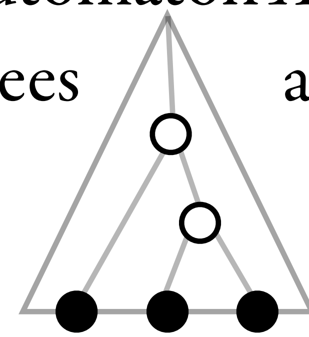
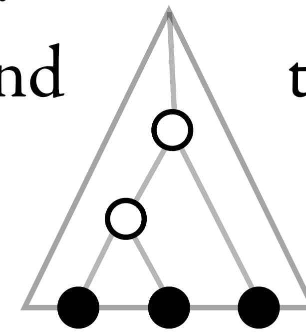
Goal: No deterministic tree-walking automaton recognizes the language L.

Fix a deterministic tree-walking automaton A.

We will find trees  and  that cannot be distinguished by A.

Goal: No deterministic tree-walking automaton recognizes the language L.

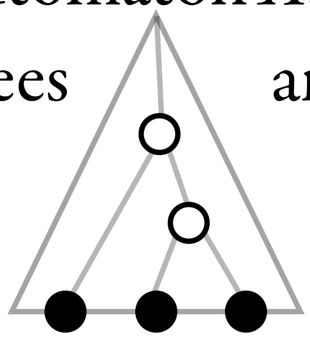
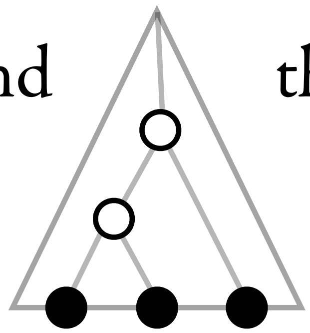
Fix a deterministic tree-walking automaton A.

We will find trees  and  that cannot be distinguished by A.

Strategy:

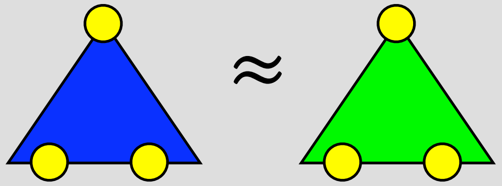
Goal: No deterministic tree-walking automaton recognizes the language L.

Fix a deterministic tree-walking automaton A.

We will find trees  and  that cannot be distinguished by A.

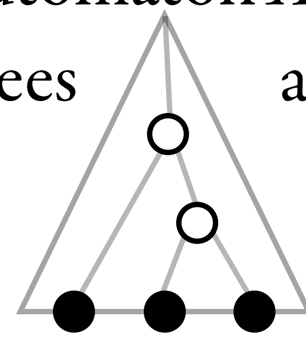
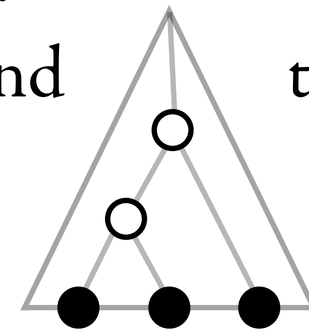
Strategy:

1 Define notion of pattern, together with pattern equivalence



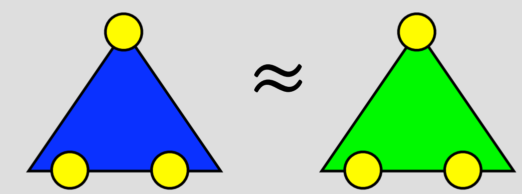
Goal: No deterministic tree-walking automaton recognizes the language *L*.

Fix a deterministic tree-walking automaton *A*.

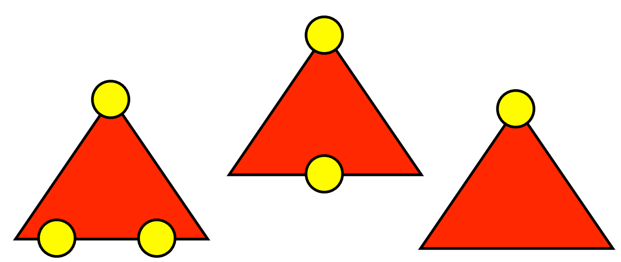
We will find trees  and  that cannot be distinguished by *A*.

Strategy:

1 Define notion of pattern, together with pattern equivalence

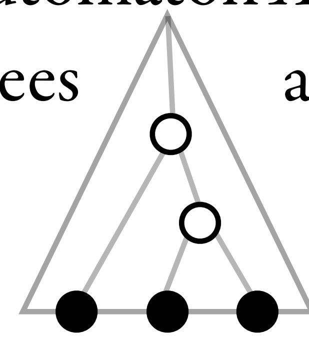
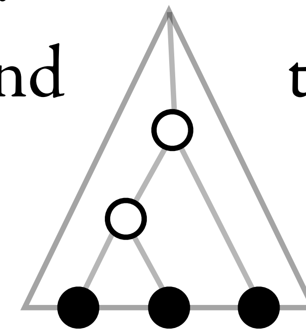


2 Using algebra, find some confusing patterns



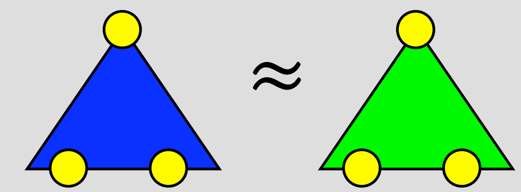
Goal: No deterministic tree-walking automaton recognizes the language L.

Fix a deterministic tree-walking automaton A.

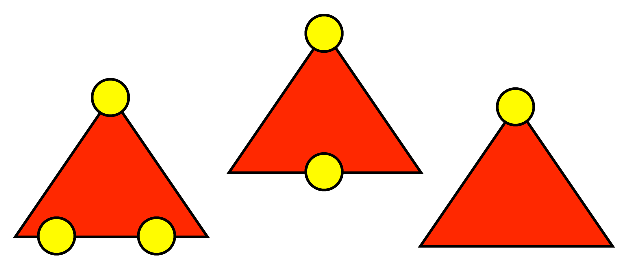
We will find trees  and  that cannot be distinguished by A.

Strategy:

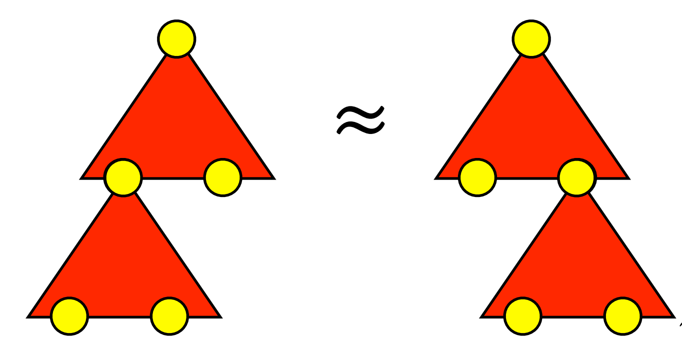
1 Define notion of pattern, together with pattern equivalence



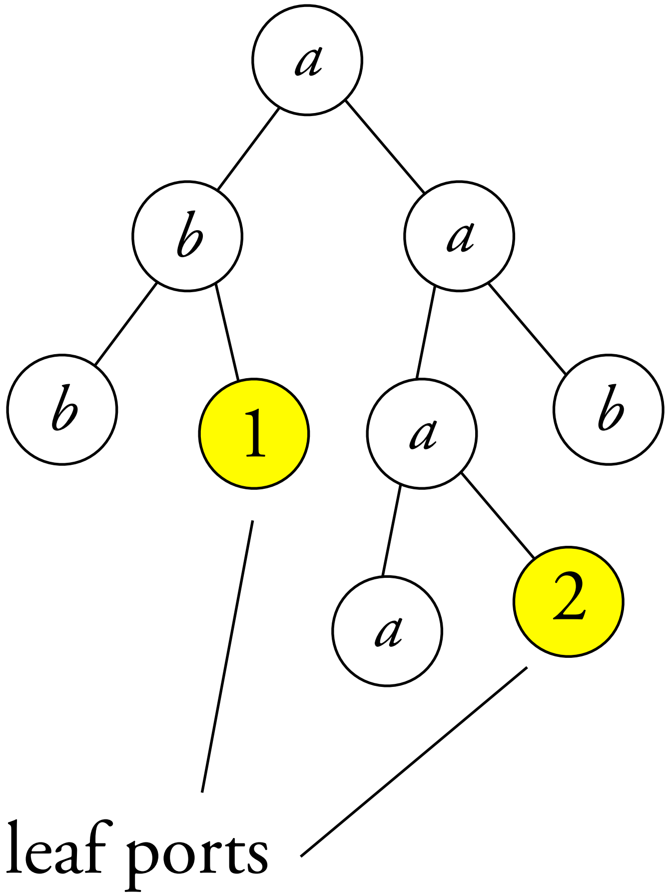
2 Using algebra, find some confusing patterns



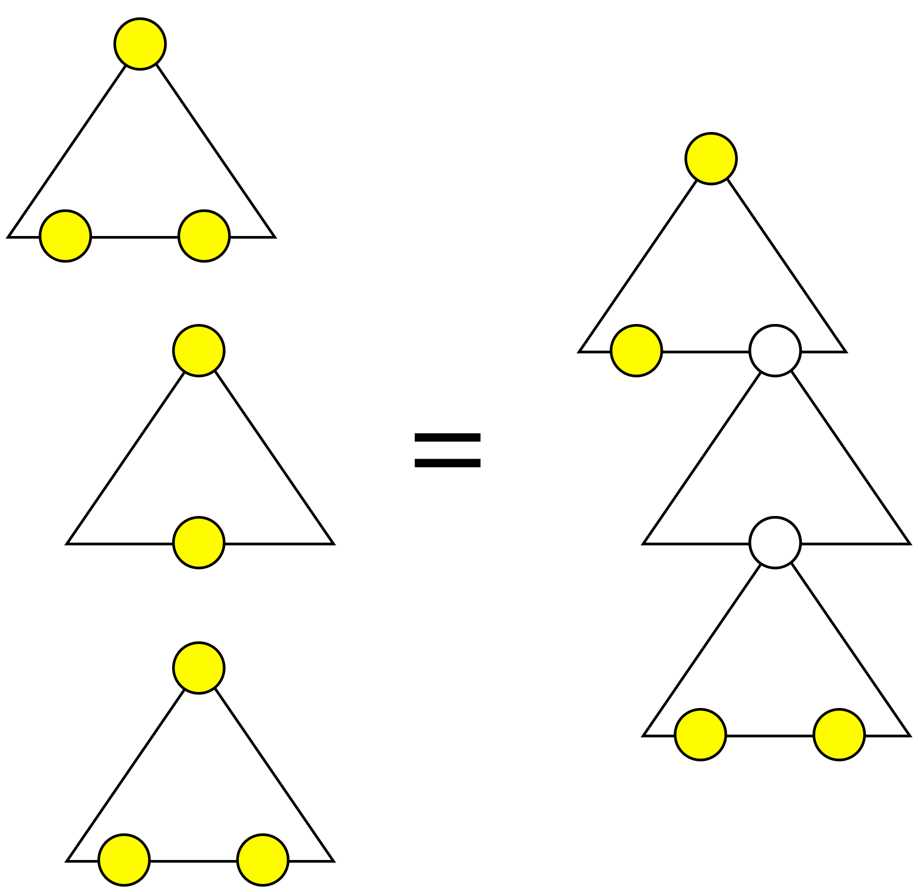
3 Build the counterexample using these confusing patterns



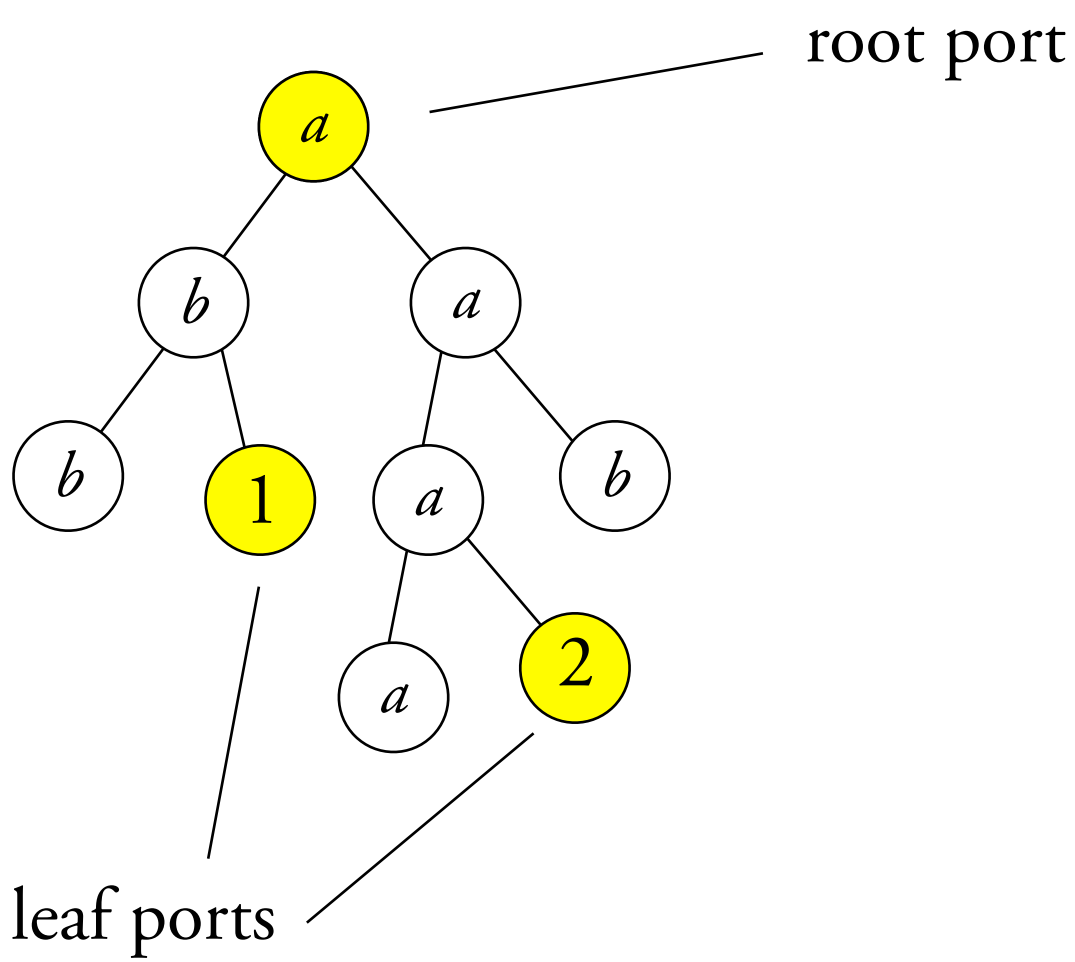
A pattern is a tree with some distinguished leaves, called *leaf ports.* The number of leaf ports is the *arity* of the pattern.

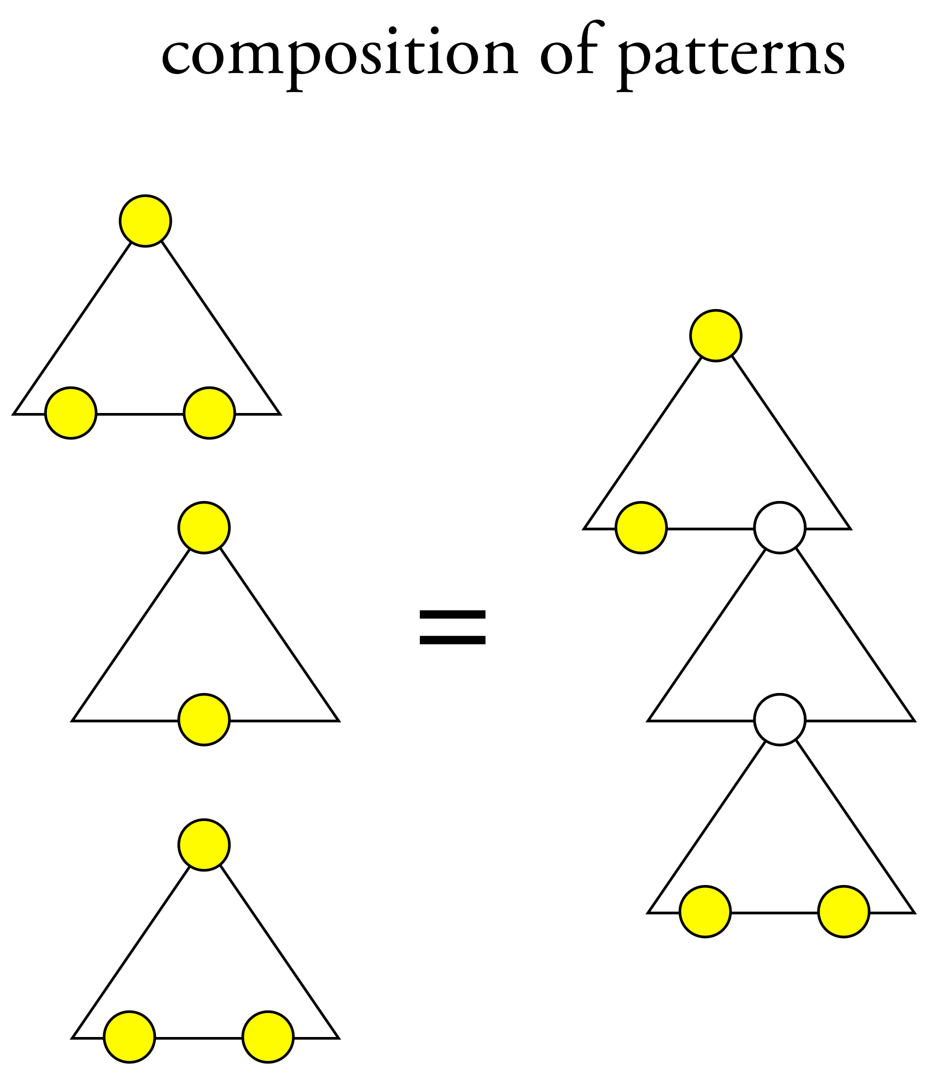


composition of patterns

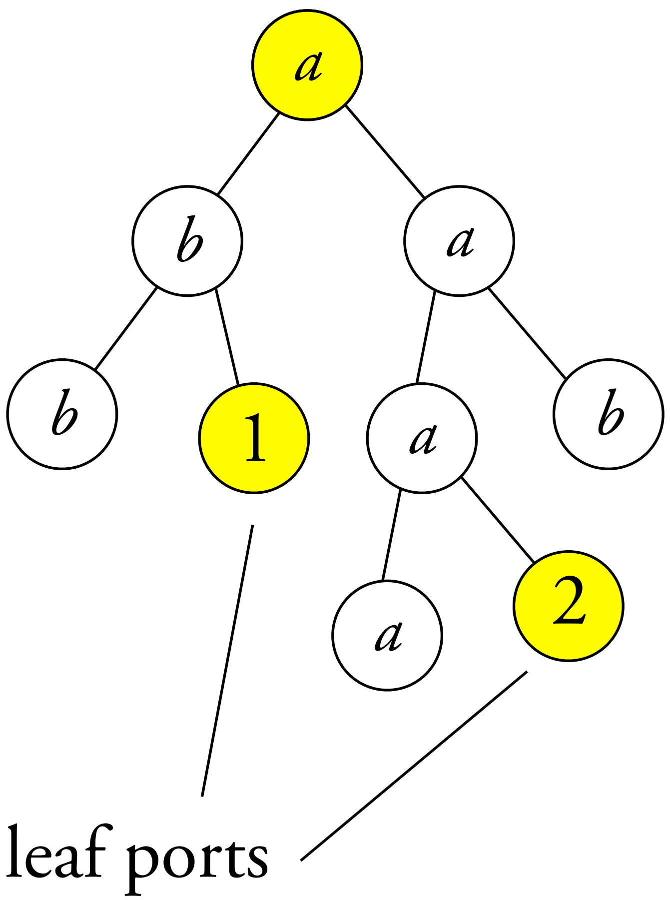


A pattern is a tree with some distinguished leaves, called *leaf ports*. The number of leaf ports is the *arity* of the pattern.

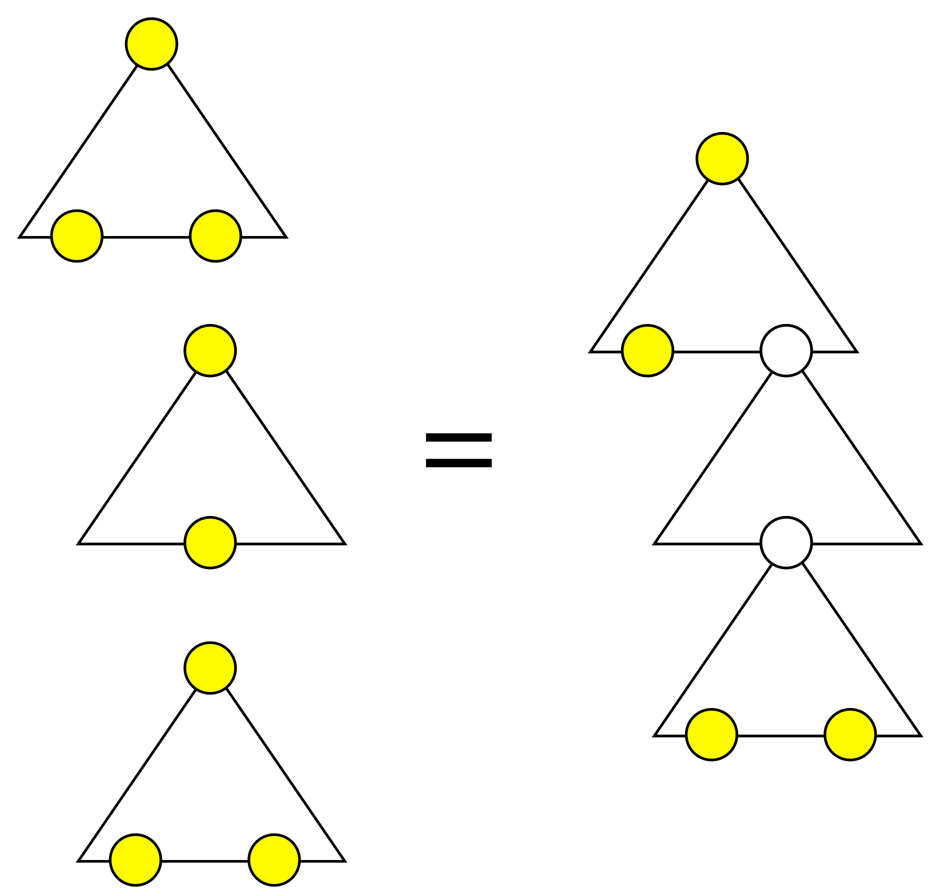




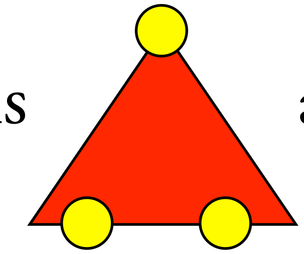
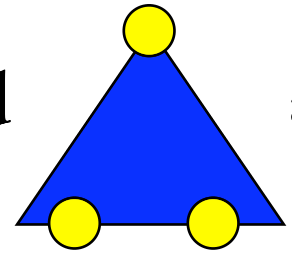
A pattern is a tree with some distinguished leaves, called *leaf ports*. The number of leaf ports is the *arity* of the pattern.

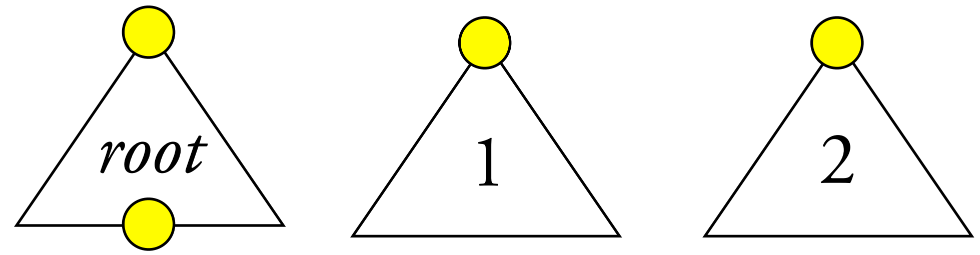


composition of patterns

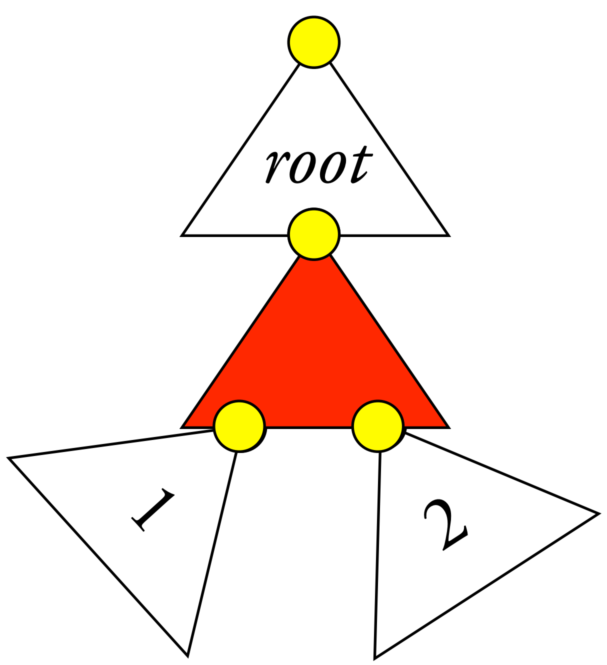


Fix a tree-walking automaton *A*.

Two patterns  and  are considered *A*-equivalent if

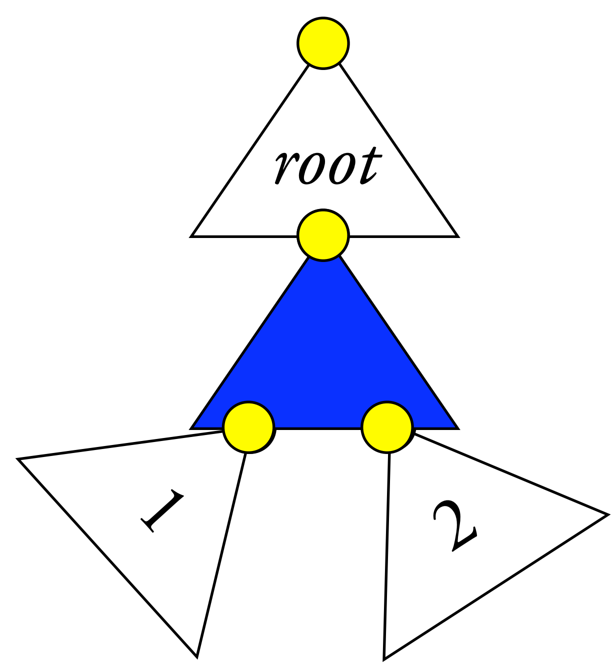
for every completion 

A accepts the tree

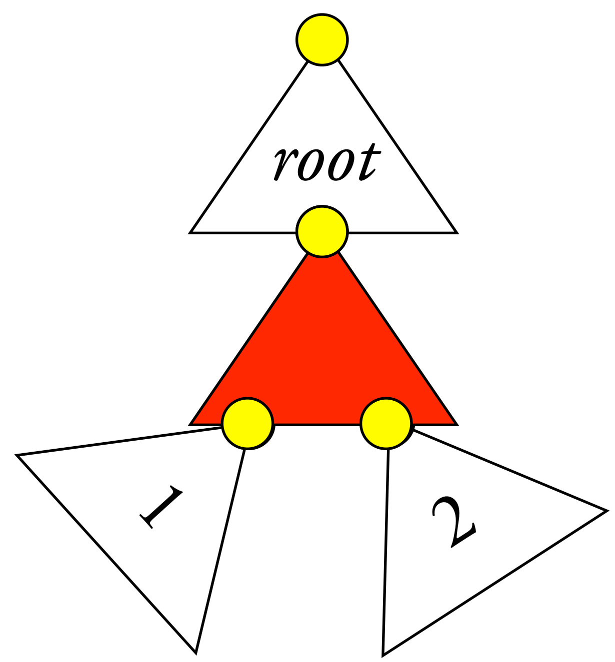


iff

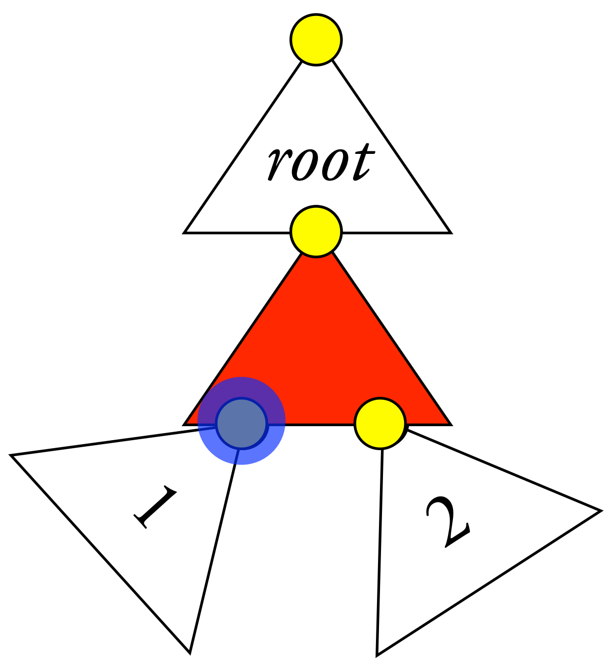
A accepts the tree



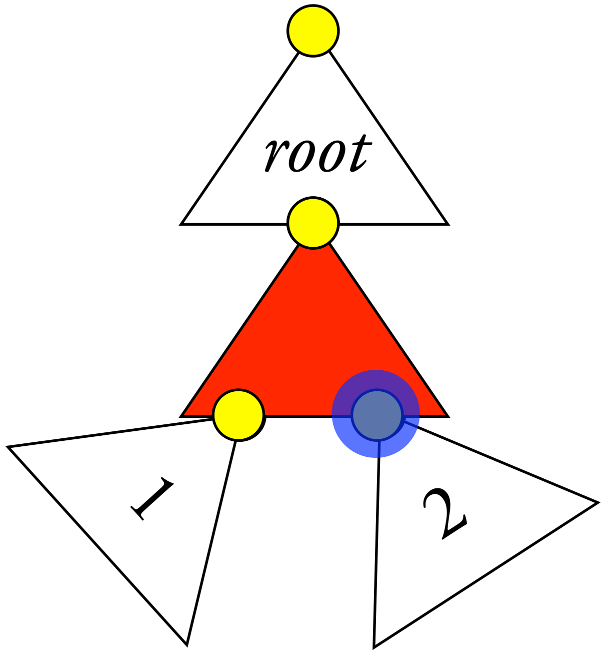
Fact. For a fixed number of ports, *A*-equivalence has finitely many equivalence classes.



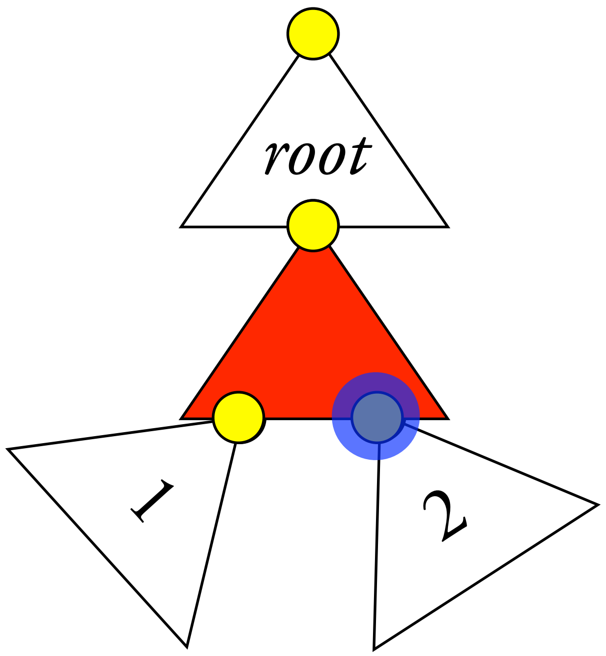
Fact. For a fixed number of ports, *A*-equivalence has finitely many equivalence classes.



Fact. For a fixed number of ports, *A*-equivalence has finitely many equivalence classes.

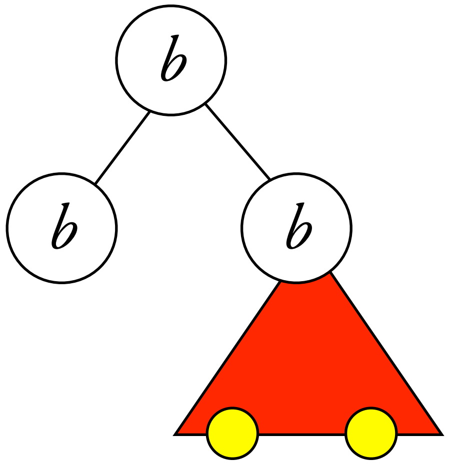


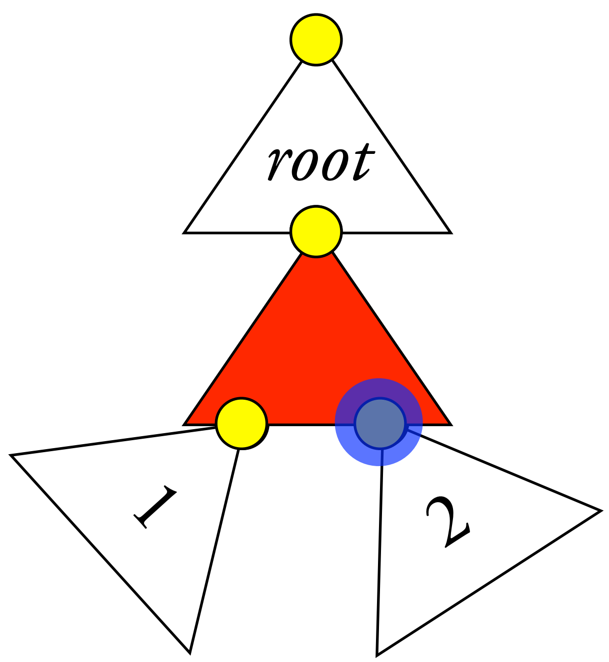
Fact. For a fixed number of ports, *A*-equivalence has finitely many equivalence classes.

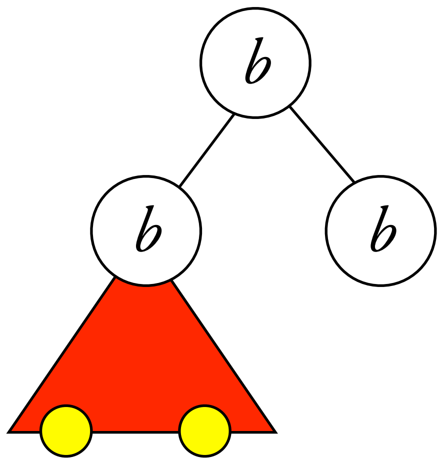


$$\delta_{\Delta} \subseteq Q \times \{\varepsilon, 1, .., n\} \times Q \times \{\varepsilon, 1, .., n\}$$

Fact. For a fixed number of ports, *A*-equivalence has finitely many equivalence classes.

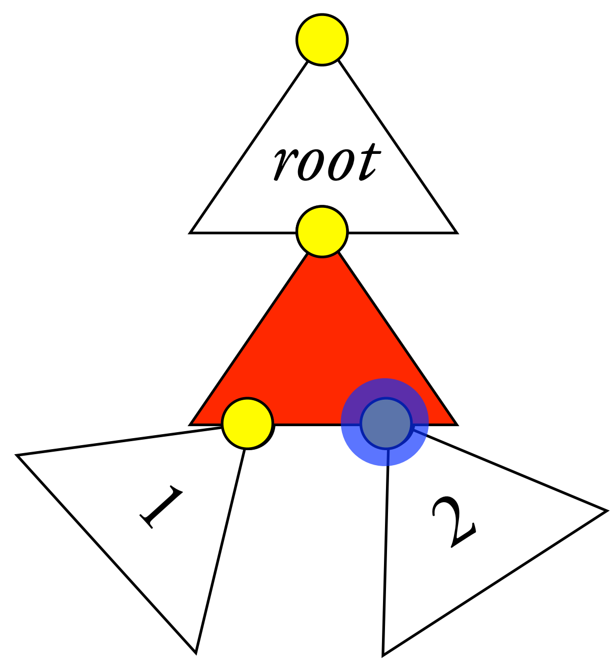


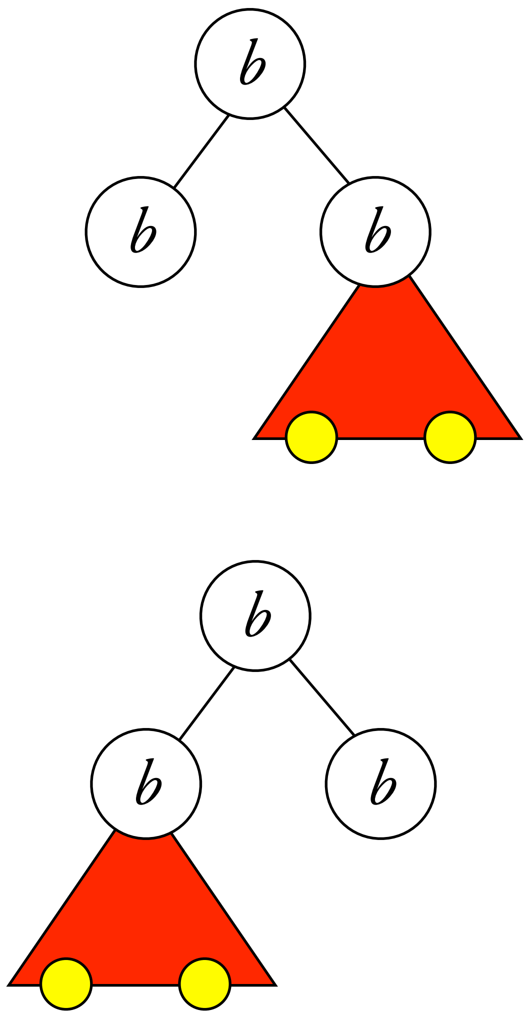




$$\delta_\Delta \subseteq Q \times \{\varepsilon, 1, .., n\} \times Q \times \{\varepsilon, 1, .., n\}$$

Fact. For a fixed number of ports, *A*-equivalence has finitely many equivalence classes.

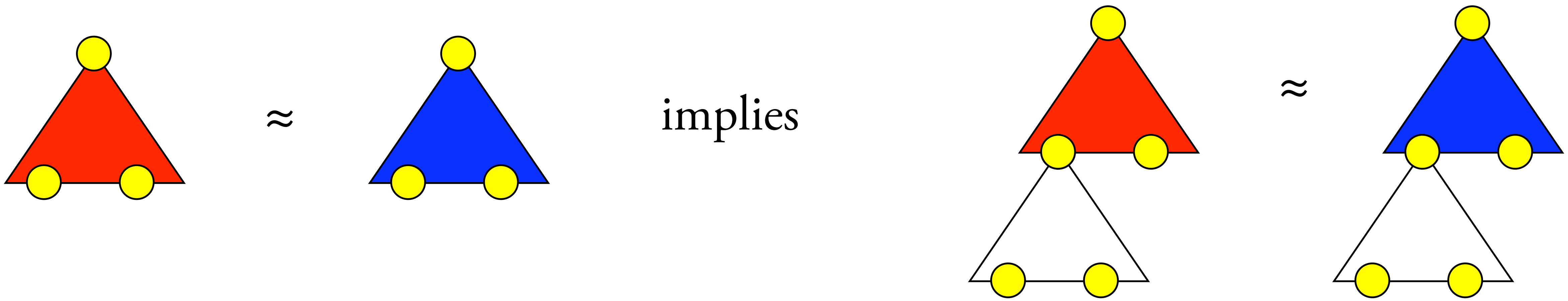




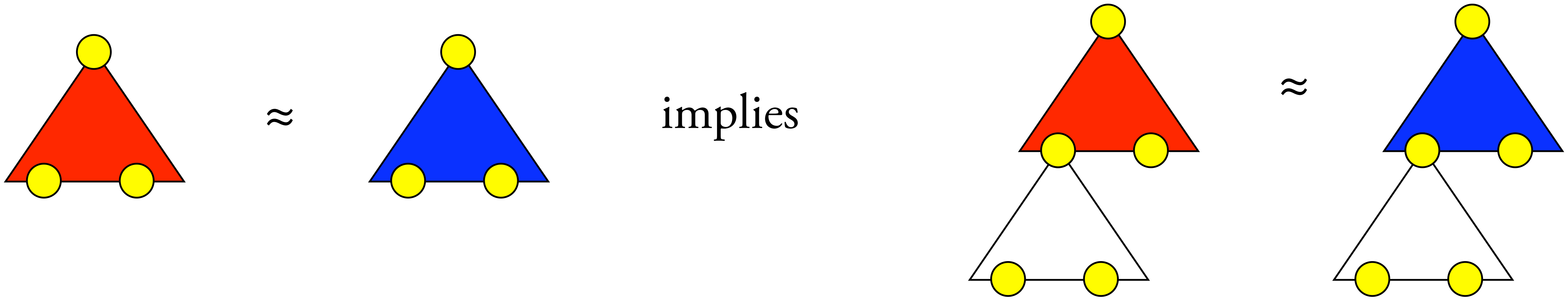
$$\delta_\Delta \subseteq Q \times \{\varepsilon, 1, .., n\} \times Q \times \{\varepsilon, 1, .., n\}$$

$$\delta_\Delta : types^{\{\varepsilon, 1, .., n\}} \longrightarrow Q \times \{\varepsilon, 1, .., n\} \times Q \times \{\varepsilon, 1, .., n\}$$

A-equivalence is a congruence with respect to pattern composition.

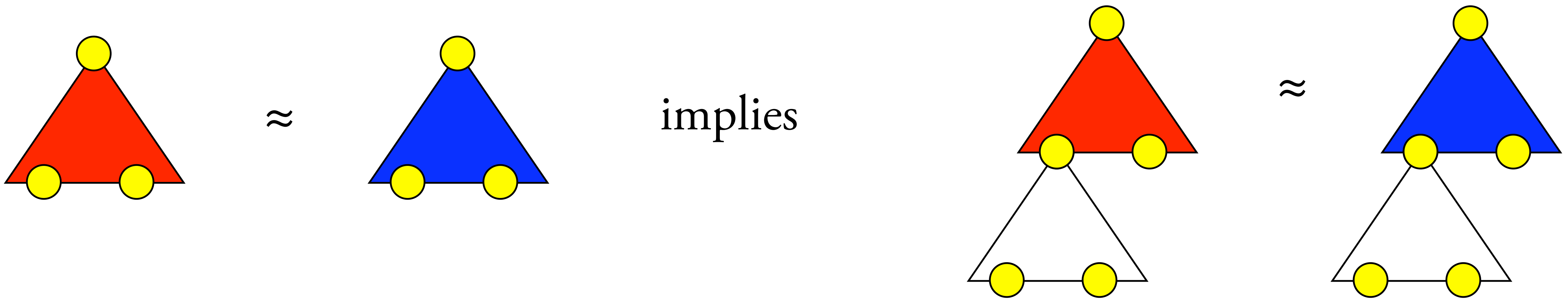


A-equivalence is a congruence with respect to pattern composition.

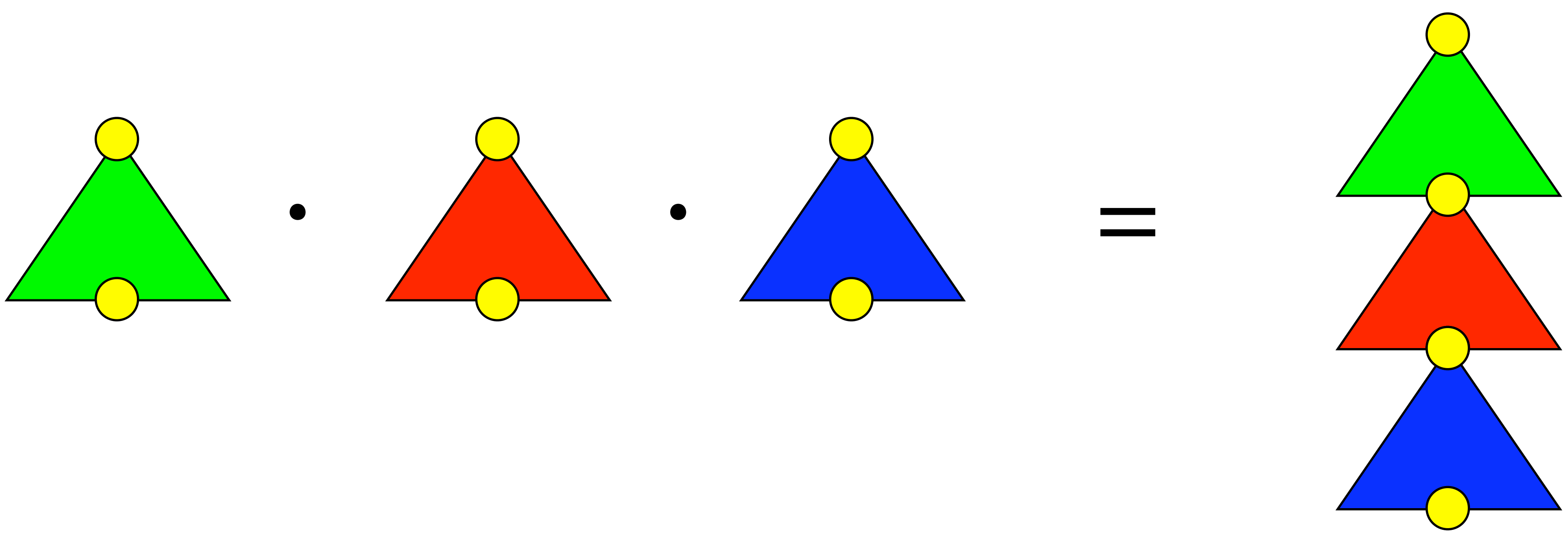


Corollary. *A*-equivalence classes of unary patterns form a finite semigroup.

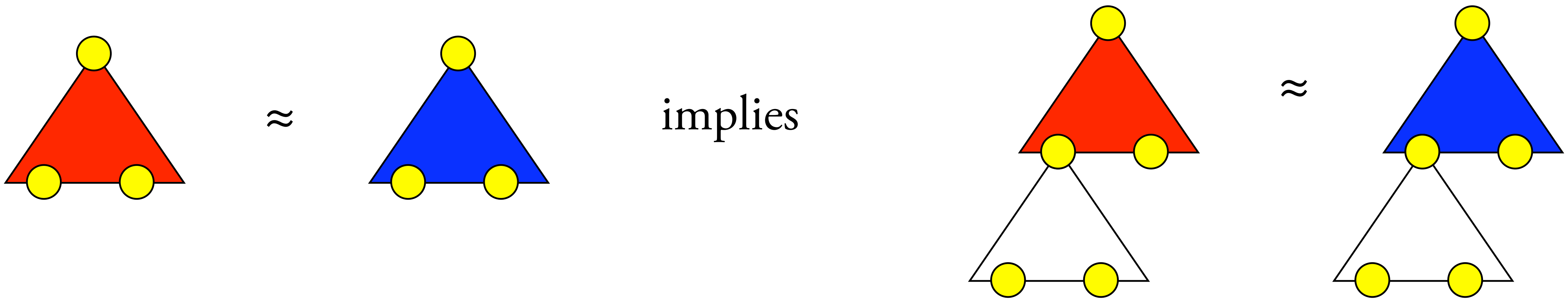
A-equivalence is a congruence with respect to pattern composition.



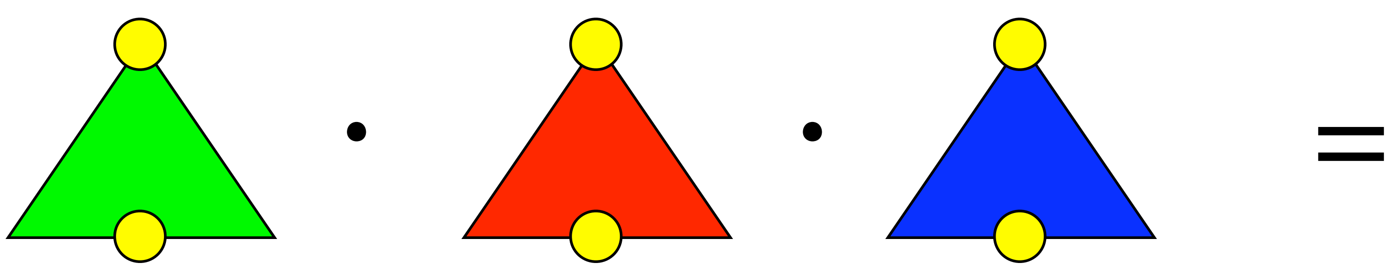
Corollary. *A*-equivalence classes of unary patterns form a finite semigroup.

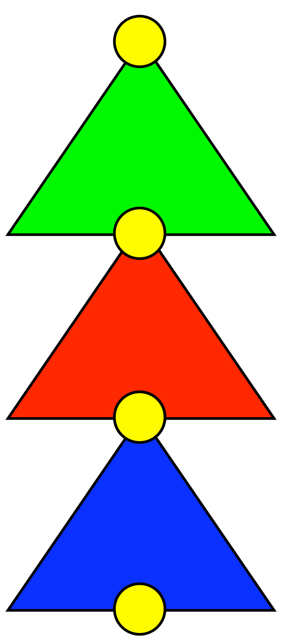


A-equivalence is a congruence with respect to pattern composition.



Corollary. *A*-equivalence classes of unary patterns form a finite semigroup.

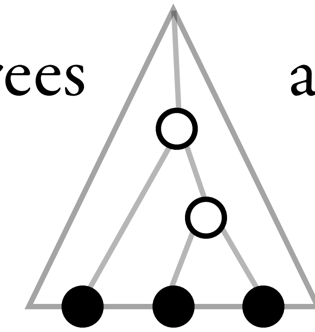
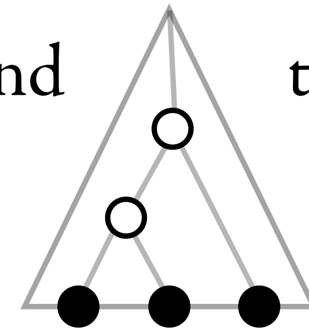




(this semigroup does not contain all
information on the automaton)

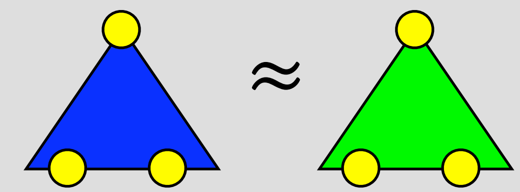
Goal: No deterministic tree-walking automaton recognizes the language *L*.

Fix a deterministic tree-walking automaton *A*.

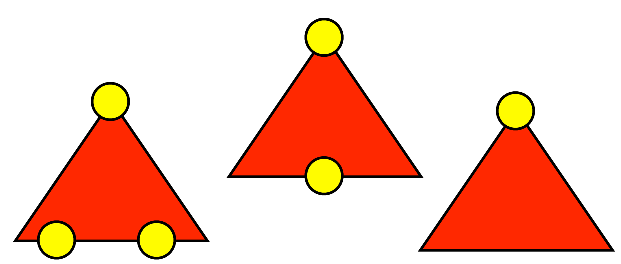
We will find trees  and  that cannot be distinguished by *A*.

Strategy:

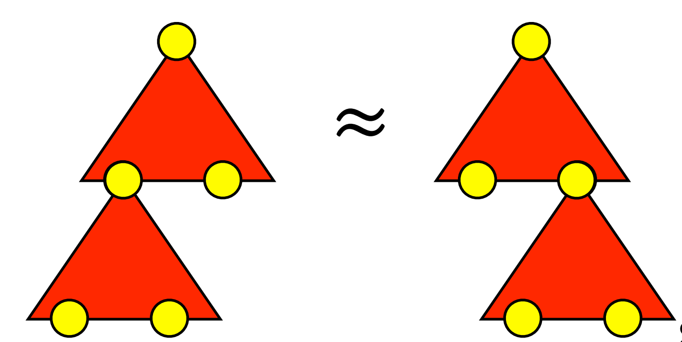
1 Define notion of pattern, together with pattern equivalence



2 Using algebra, find some confusing patterns

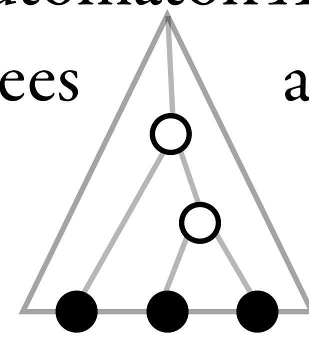
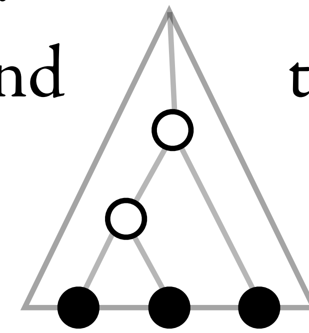


3 Build the counterexample using these confusing patterns



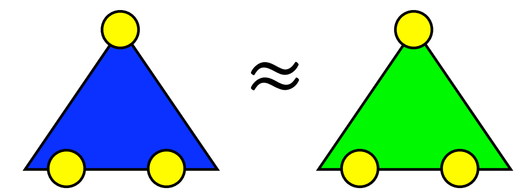
Goal: No deterministic tree-walking automaton recognizes the language L.

Fix a deterministic tree-walking automaton A.

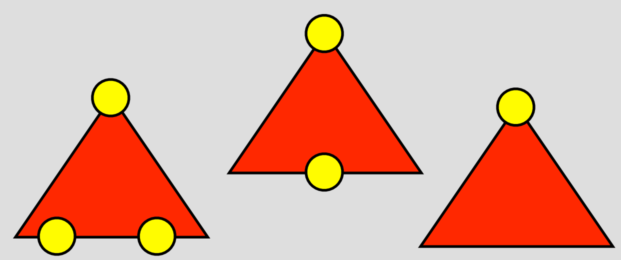
We will find trees  and  that cannot be distinguished by A.

Strategy:

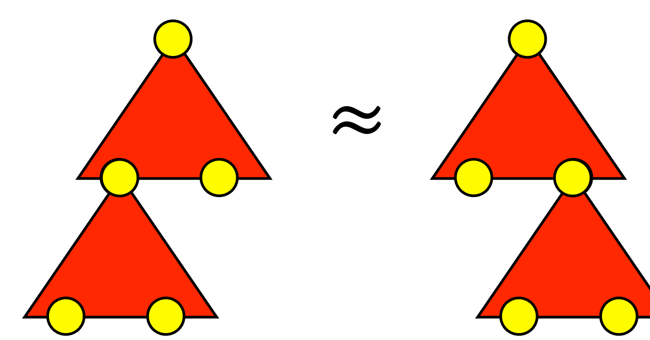
1 Define notion of pattern, together with pattern equivalence



2 Using algebra, find some confusing patterns



3 Build the counterexample using these confusing patterns



A semigroup lemma. For any finite semigroup S and elements $x, y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,
there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

Example. $S = \{0,1\}$ with addition mod 2. In this case, set $X=Y=0$.

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

Example. $S = \{0,1\}$ with addition mod 2. In this case, set $X=Y=0$.

Example. $S = \{a(a+b)^*, b(a+b)^*\}$. In this case, set $X=x$ and $Y=y$.

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

Proof.

$$X := (x^{\omega} \cdot y^{\omega})^{\omega} \qquad Y := y^{\omega} \cdot X$$

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$

$$Y = Y \cdot Y = Y \cdot X$$

Proof.

$$X := (x^\omega \cdot y^\omega)^\omega \qquad Y := y^\omega \cdot \mathrm{X}$$

$$X \cdot X = (x^\omega \cdot y^\omega)^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega = X$$

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

Proof.

$$X := (x^\omega \cdot y^\omega)^\omega \qquad Y := y^\omega \cdot X$$

$$X \cdot X = (x^\omega \cdot y^\omega)^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega = X$$
$$X \cdot Y = (x^\omega \cdot y^\omega)^\omega \cdot y^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega = X$$

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

Proof.

$$X := (x^\omega \cdot y^\omega)^\omega \qquad Y := y^\omega \cdot X$$

$$X \cdot X = (x^\omega \cdot y^\omega)^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega = X$$
$$X \cdot Y = (x^\omega \cdot y^\omega)^\omega \cdot y^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega \cdot (x^\omega \cdot y^\omega)^\omega = (x^\omega \cdot y^\omega)^\omega = X$$
$$Y \cdot X = y \cdot X \cdot X = y \cdot X = Y$$

A semigroup lemma. For any finite semigroup S and elements $x,y \in S$,

there are $X \in xS$ and $Y \in yS$ with

$$X = X \cdot X = X \cdot Y$$
$$Y = Y \cdot Y = Y \cdot X$$

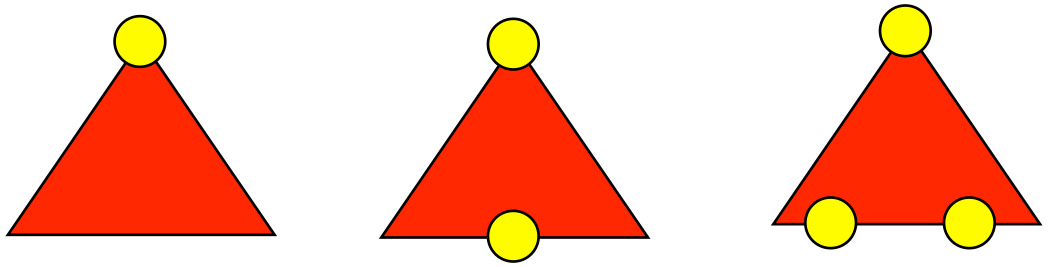
Proof.

$$X := (x^{\omega} \cdot y^{\omega})^{\omega} \qquad Y := y^{\omega} \cdot X$$

$$X \cdot X = (x^{\omega} \cdot y^{\omega})^{\omega} \cdot (x^{\omega} \cdot y^{\omega})^{\omega} = (x^{\omega} \cdot y^{\omega})^{\omega} = X$$
$$X \cdot Y = (x^{\omega} \cdot y^{\omega})^{\omega} \cdot y^{\omega} \cdot (x^{\omega} \cdot y^{\omega})^{\omega} = (x^{\omega} \cdot y^{\omega})^{\omega} \cdot (x^{\omega} \cdot y^{\omega})^{\omega} = (x^{\omega} \cdot y^{\omega})^{\omega} = X$$
$$Y \cdot X = y \cdot X \cdot X = y \cdot X = Y$$

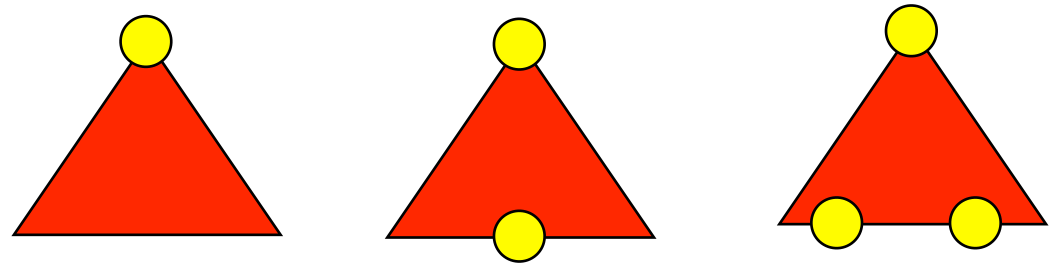
$$Y \cdot Y = y \cdot X \cdot Y = y \cdot X = Y$$

Pattern Lemma. Fix a tree-walking automaton *A*. There exist patterns

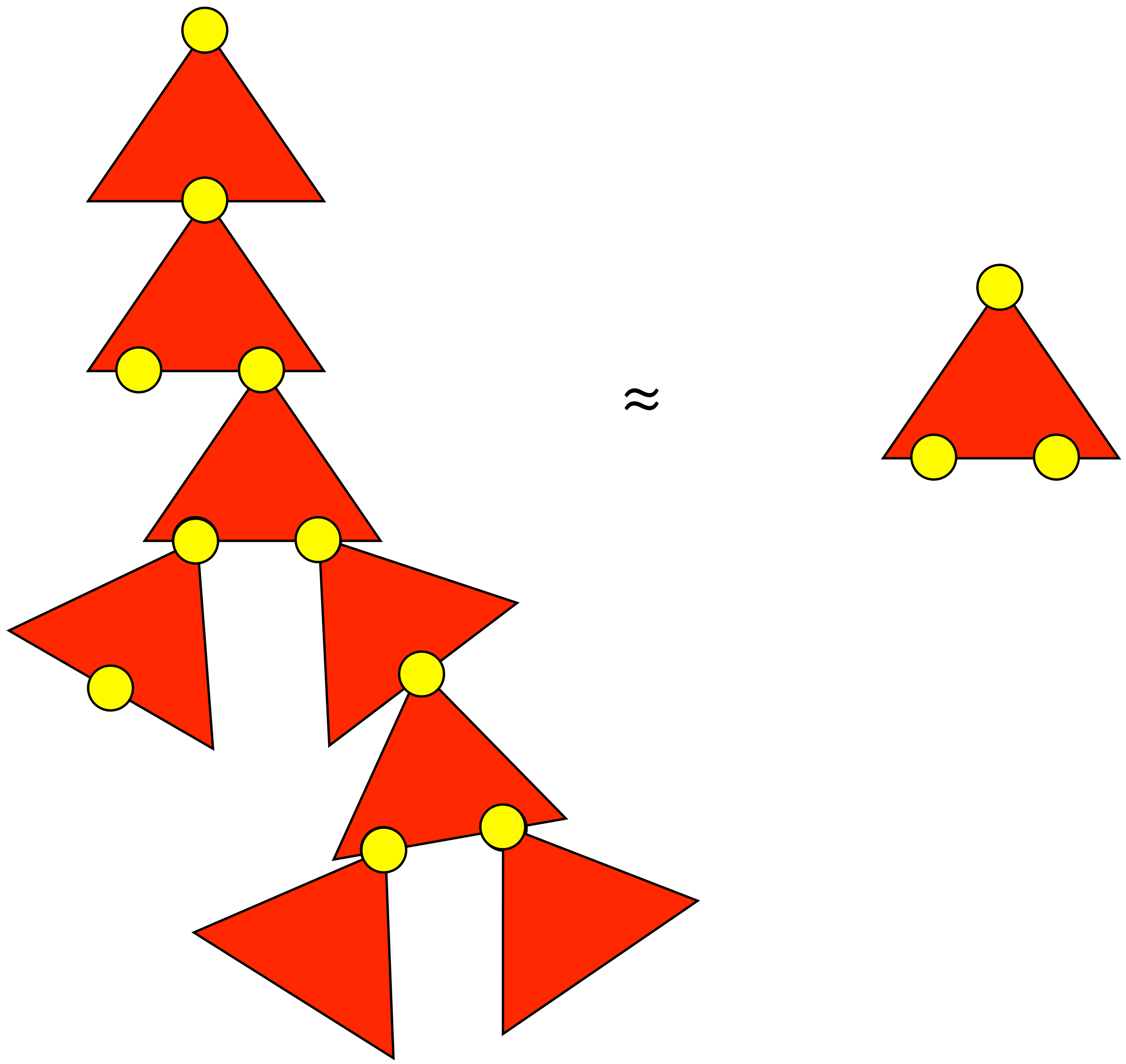


such that all compositions of these patterns with 0 ports (resp., 1 port, 2 ports) have the same *A*-equivalence class.

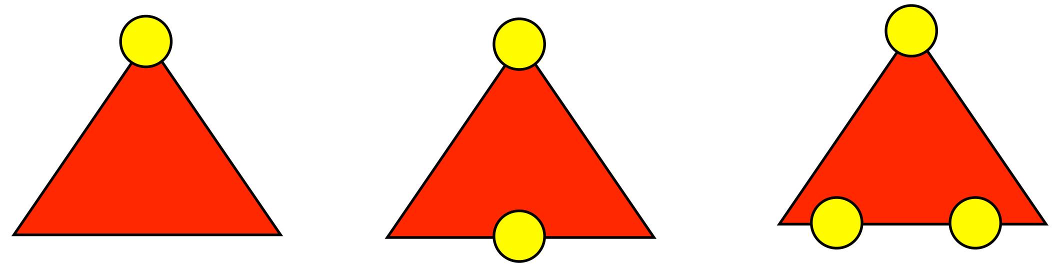
Pattern Lemma. Fix a tree-walking automaton *A*. There exist patterns



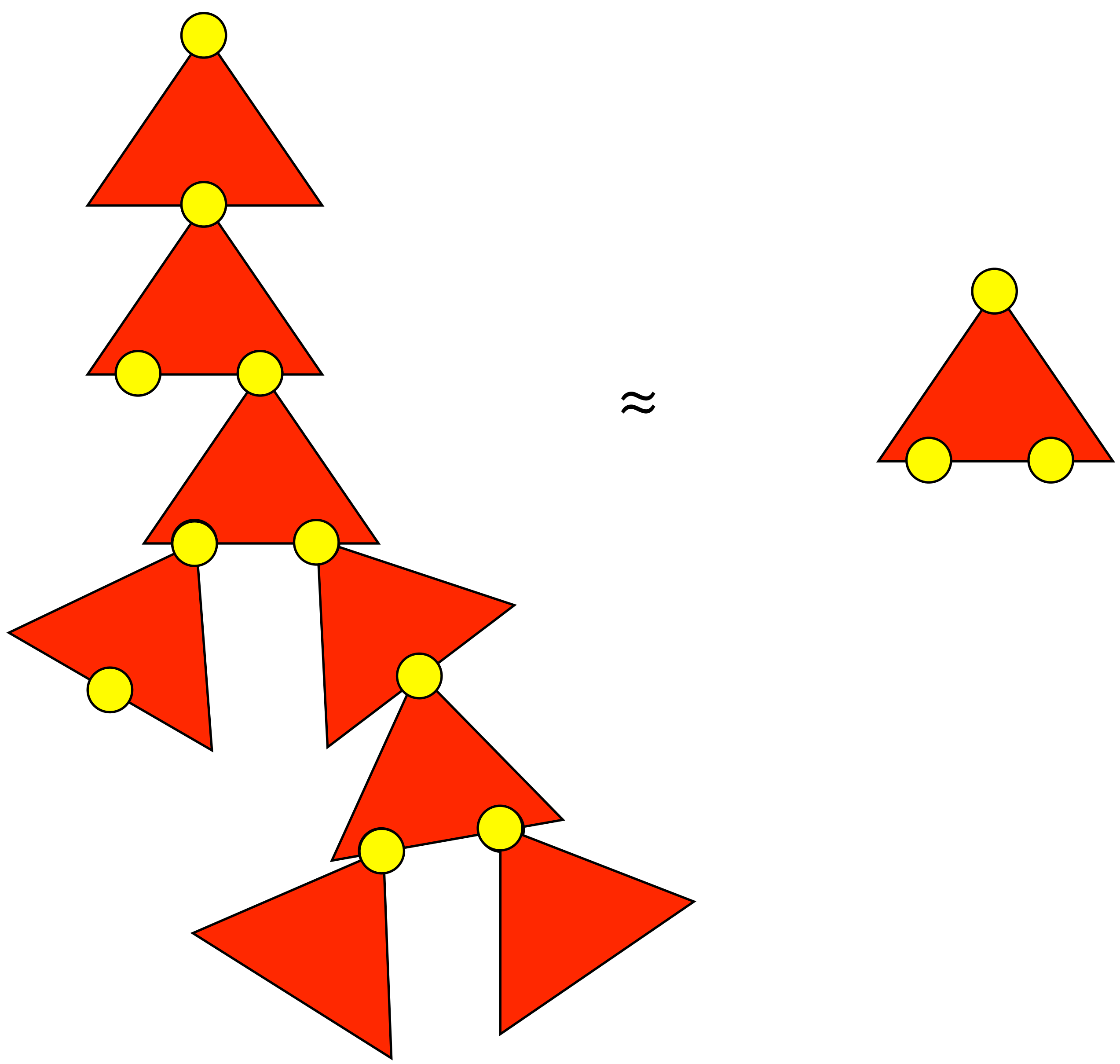
such that all compositions of these patterns with 0 ports (resp., 1 port, 2 ports) have the same *A*-equivalence class.



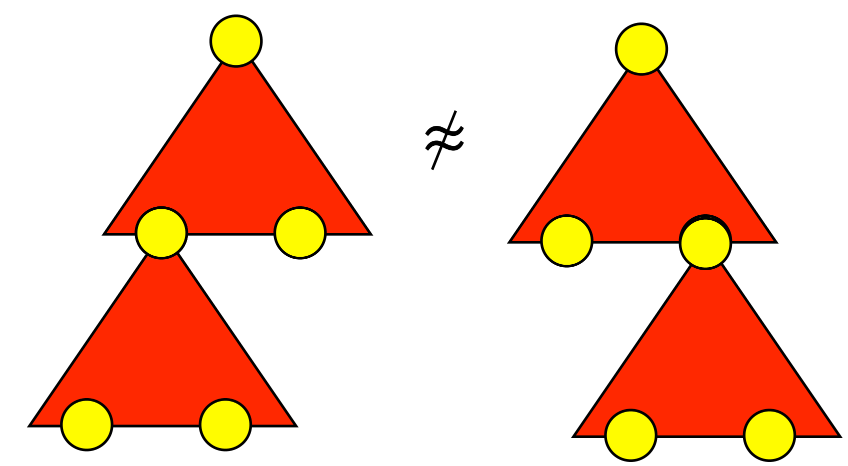
Pattern Lemma. Fix a tree-walking automaton *A*. There exist patterns

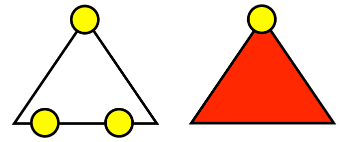
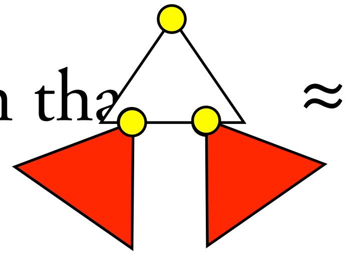
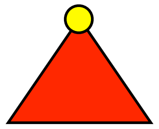


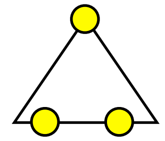
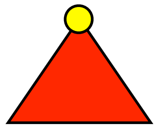
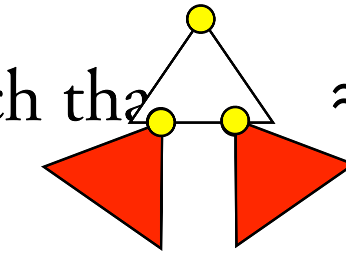
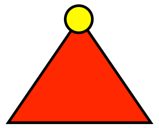
such that all compositions of these patterns with 0 ports (resp., 1 port, 2 ports) have the same *A*-equivalence class.



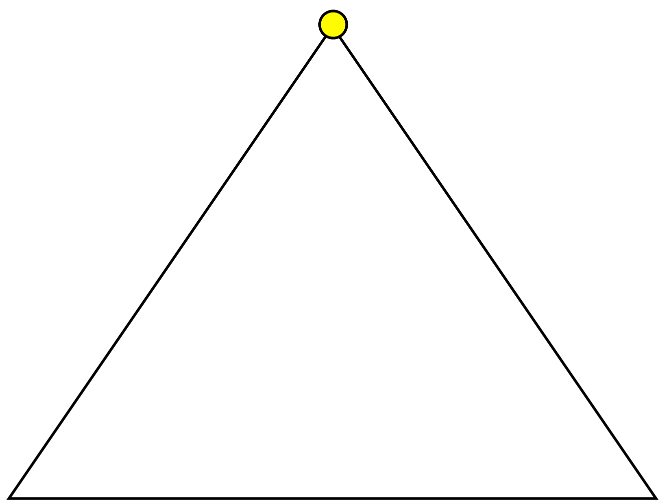
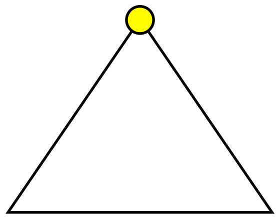
for nondeterministic automata,
the lemma fails for 3 ports.



We start out with patterns  such tha  ≈ 

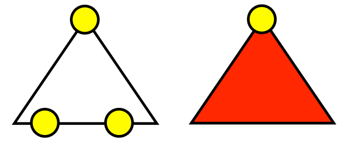
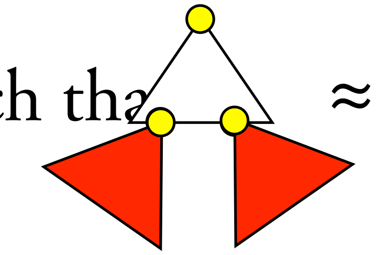
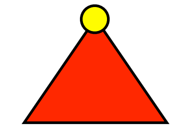
We start out with patterns   such that  ≈ 

by pumping, there are $n < m$ such that

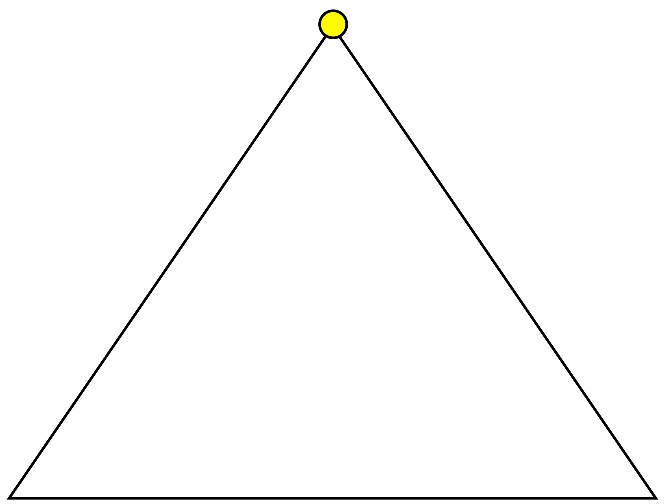
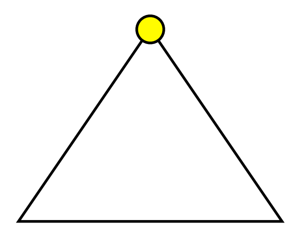
 ≈ 

balanced binary tree of depth m

balanced binary tree of depth n

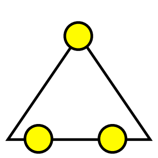
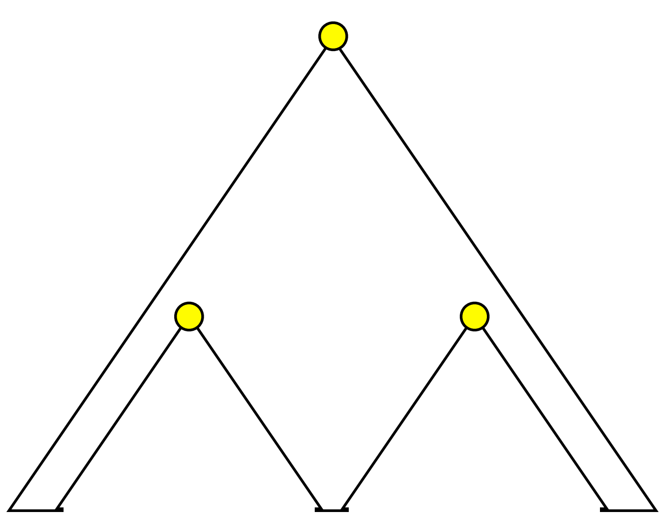
We start out with patterns  such tha  ≈ 

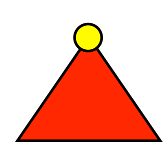
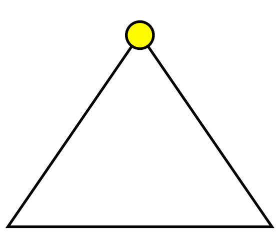
by pumping, there are $n < m$ such that

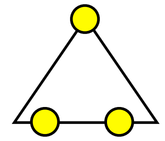
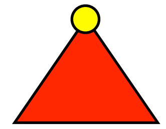
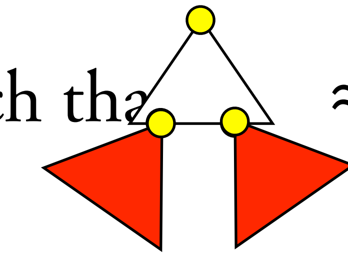
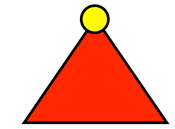
 ≈ 

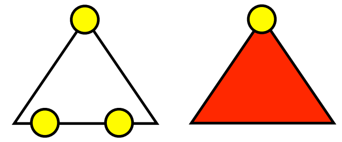
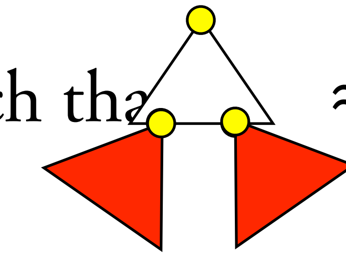
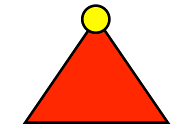
balanced binary
tree of depth m

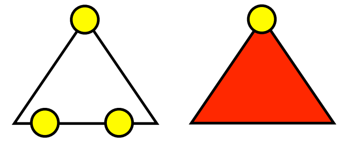
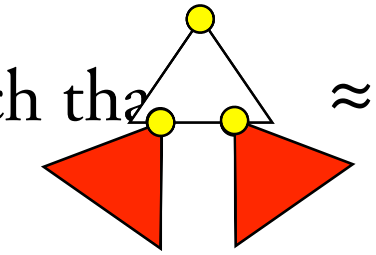
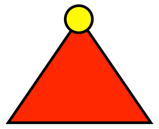
balanced binary
tree of depth n

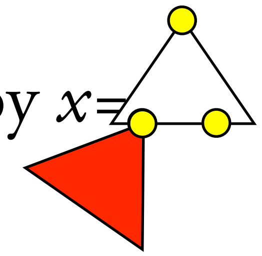
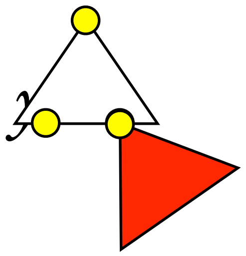
 = 

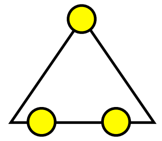
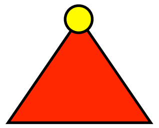
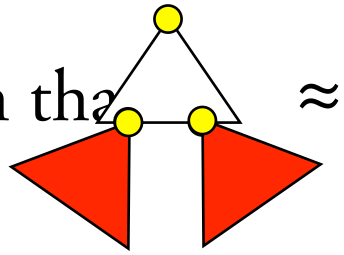
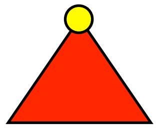
 = 

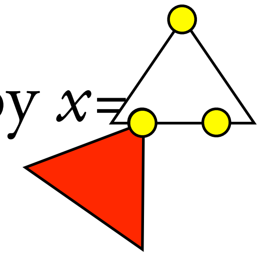
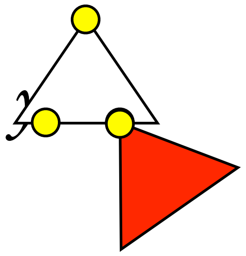
We start out with patterns   such tha  ≈ 

We start out with patterns  such that  ≈ 

We start out with patterns  such that  ≈ 

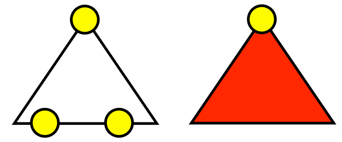
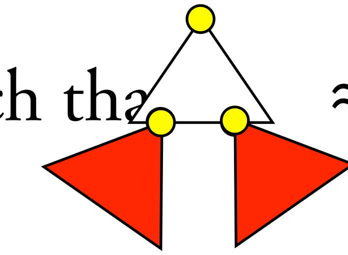
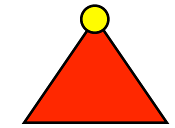
Let S be the semigroup generated by x=  and y= 

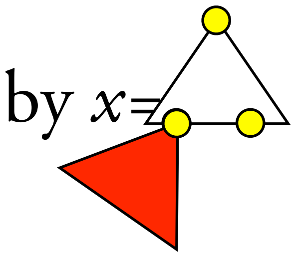
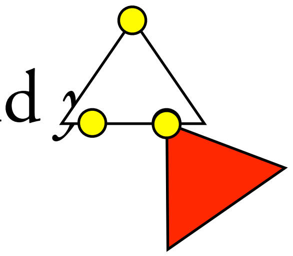
We start out with patterns   such tha  ≈ 

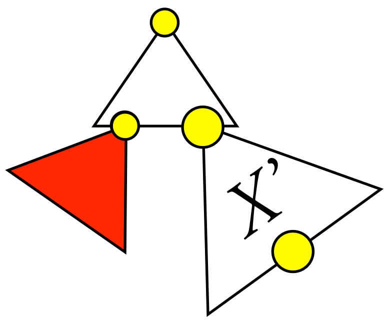
Let S be the semigroup generated by $x =$  and y 

applying the semigroup lemma, we get

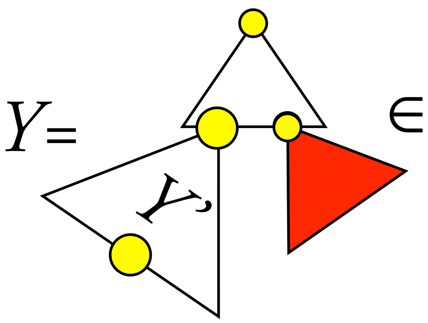
$X \approx XX \approx XY$ and $Y \approx YY \approx YX$

We start out with patterns  such tha  ≈ 

Let S be the semigroup generated by x=  and y= 

X= 

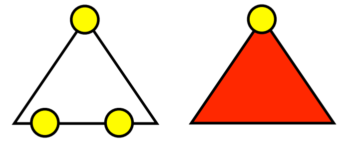
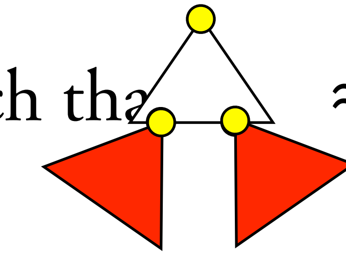
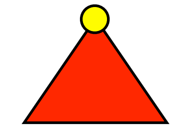
$\in xS$

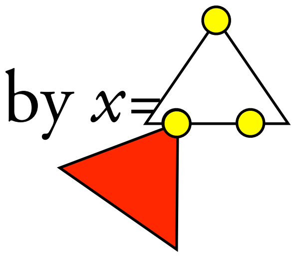
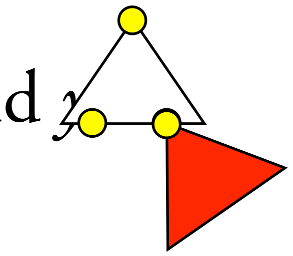
Y= 

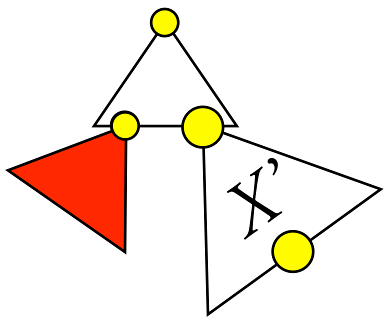
$\in yS$

applying the semigroup lemma, we get

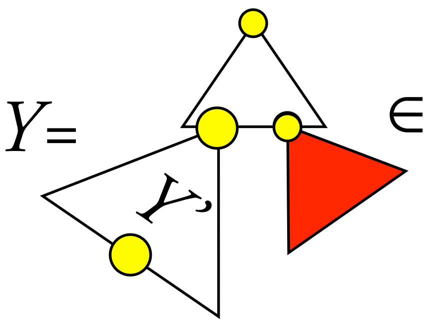
$X \approx XX \approx XY$ and $Y \approx YY \approx YX$

We start out with patterns  such that  ≈ 

Let S be the semigroup generated by x=  and y= 

X= 

$\in xS$

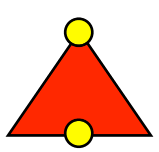
Y= 

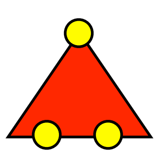
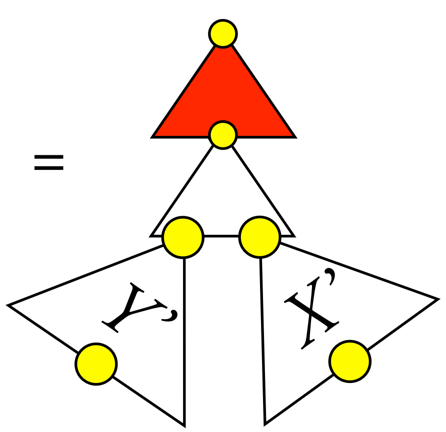
$\in yS$

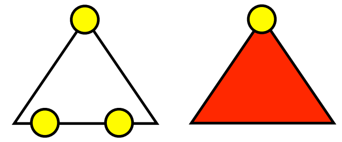
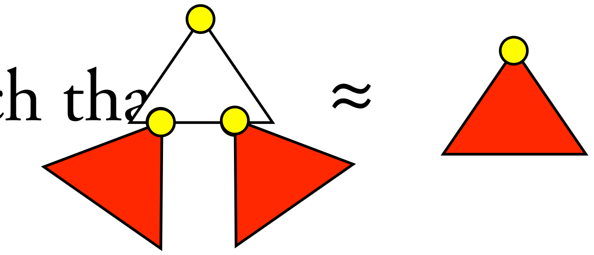
applying the semigroup lemma, we get

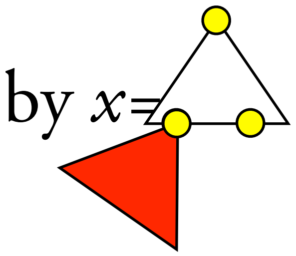
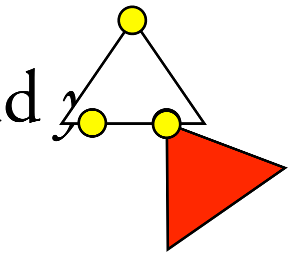
$X \approx XX \approx XY$ and $Y \approx YY \approx YX$

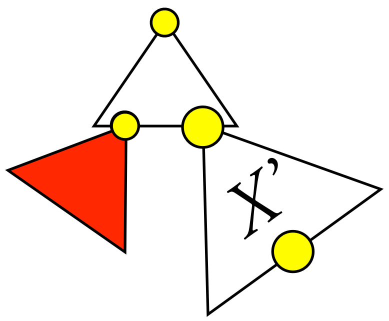
choose:

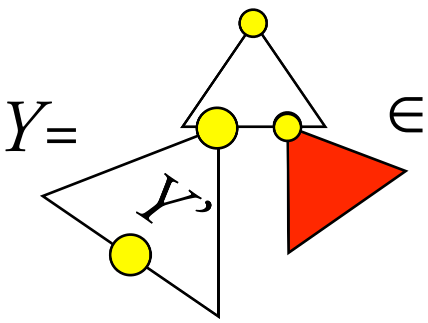
 = X

 = 

We start out with patterns  such that 

Let S be the semigroup generated by $x=$  and $y=$ 

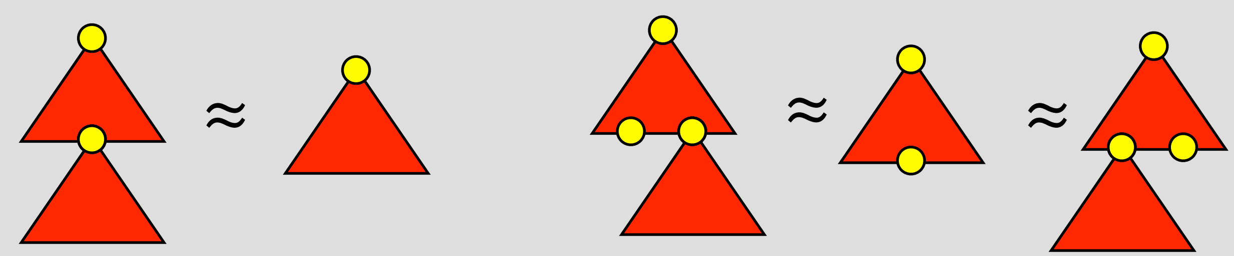
$X=$  $\in xS$

$Y=$  $\in yS$

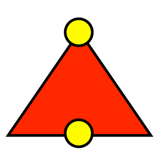
applying the semigroup lemma, we get

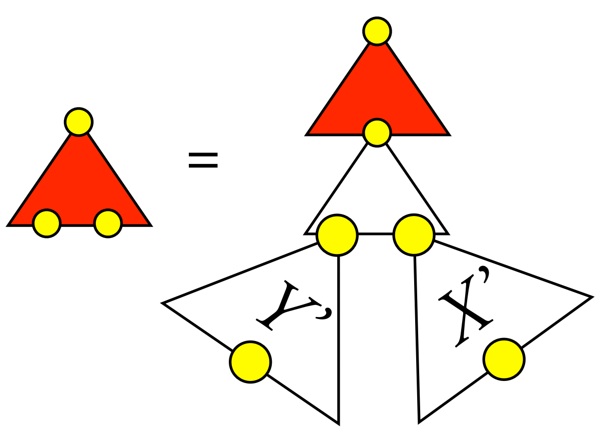
$X \approx XX \approx XY$ and $Y \approx YY \approx YX$

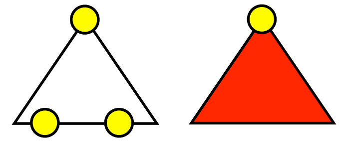
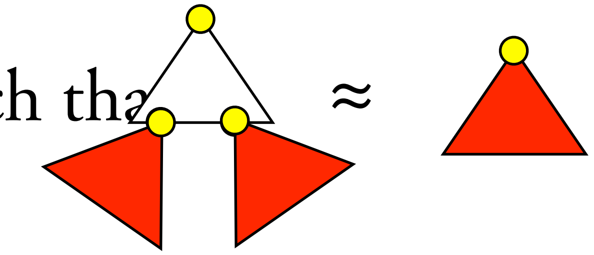
To prove the pattern lemma, it suffices to show:

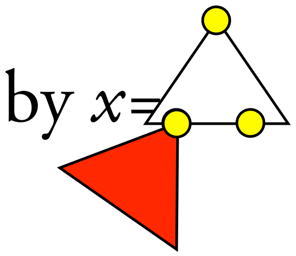
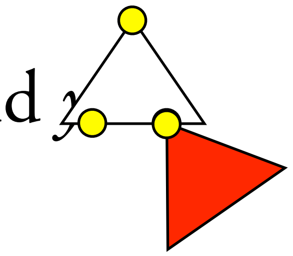


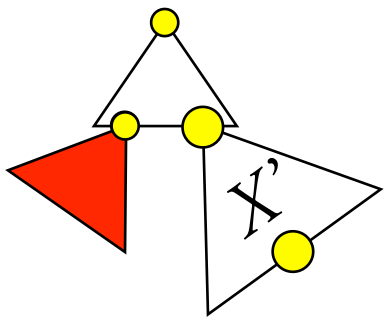
choose:

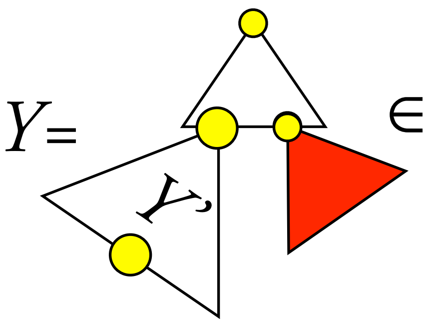
 $= X$



We start out with patterns  such that 

Let S be the semigroup generated by $x=$  and $y=$ 

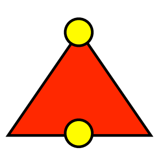
$X=$  $\in xS$

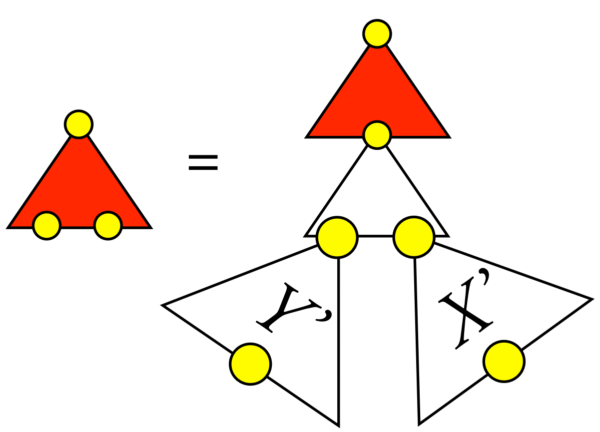
$Y=$  $\in yS$

applying the semigroup lemma, we get

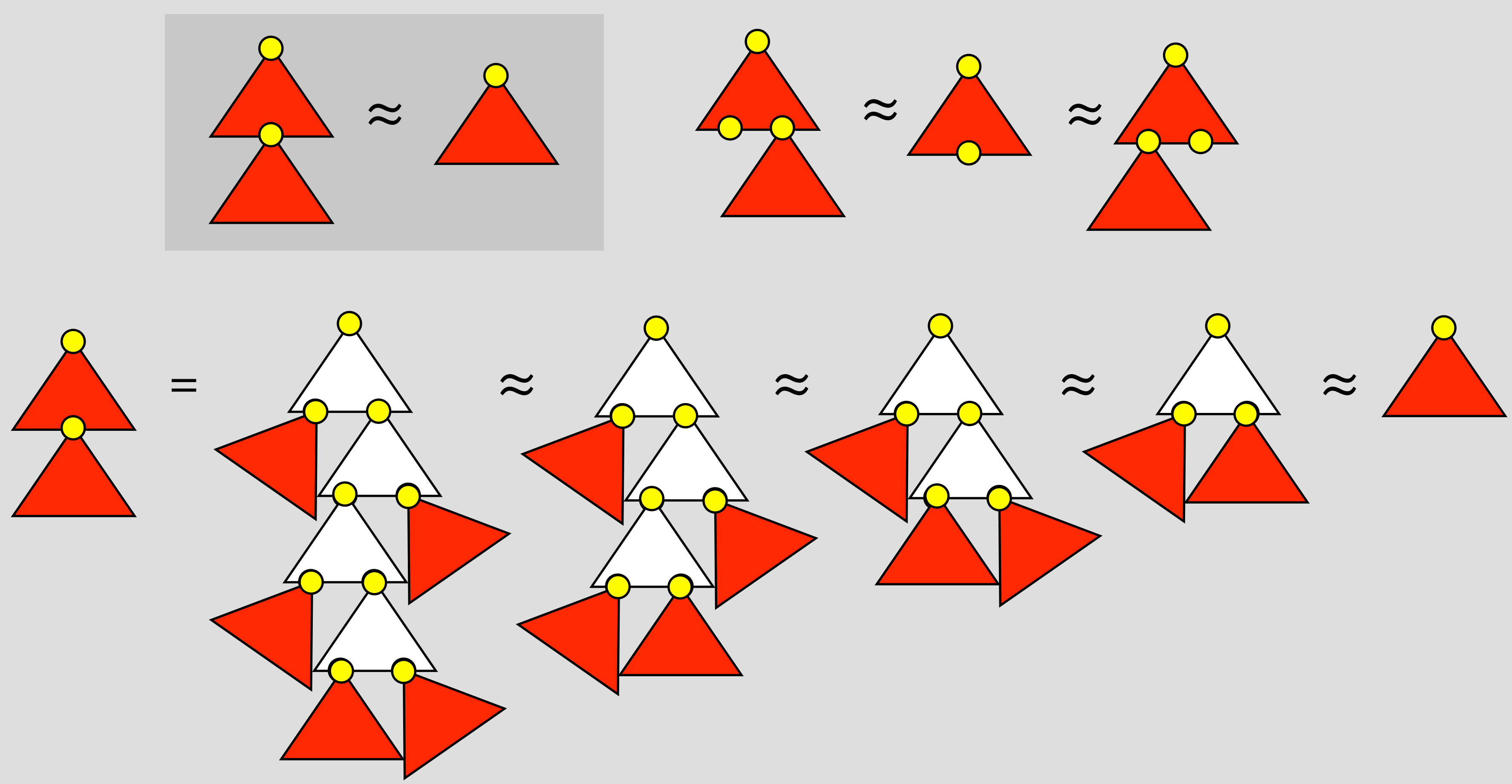
$X \approx XX \approx XY$ and $Y \approx YY \approx YX$

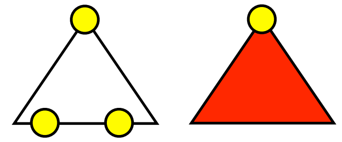
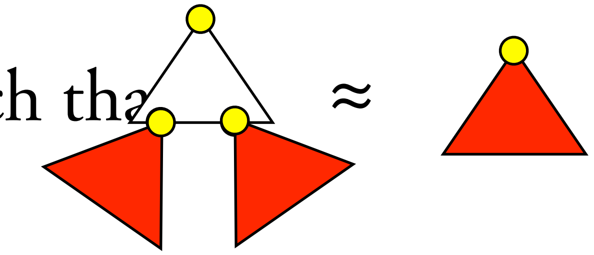
choose:

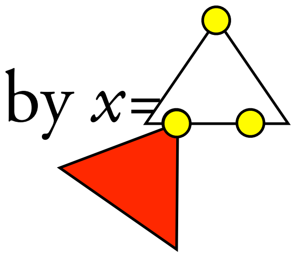
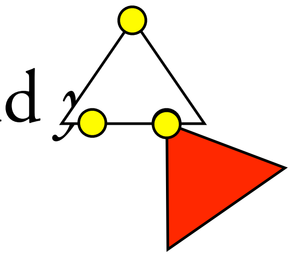
 $=$ X

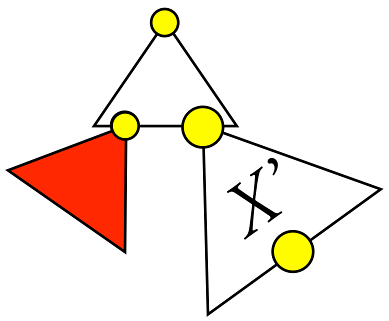


To prove the pattern lemma, it suffices to show:

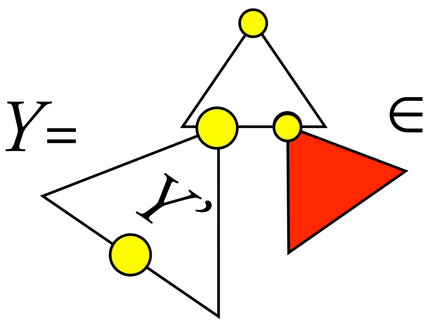


We start out with patterns  such that 

Let S be the semigroup generated by $x=$  and $y=$ 

$X=$ 

$\in xS$

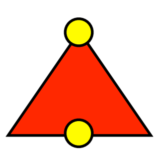
$Y=$ 

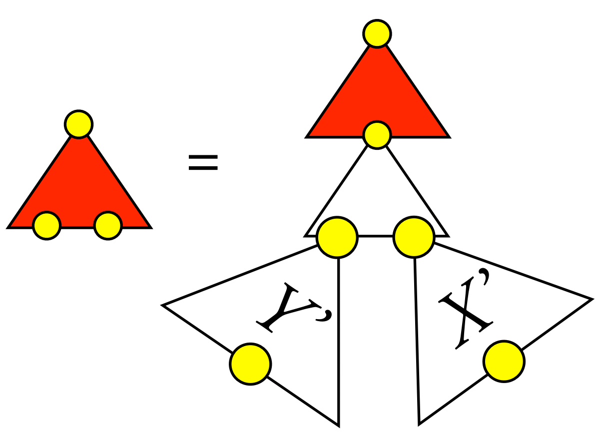
$\in yS$

applying the semigroup lemma, we get

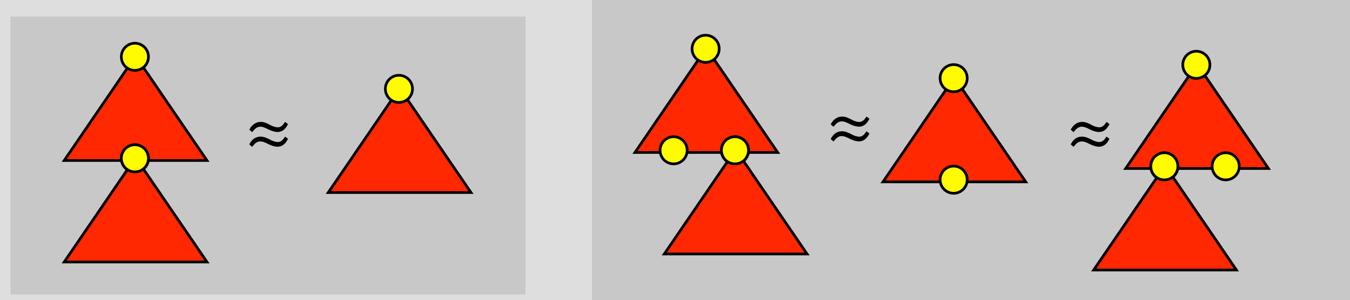
$X \approx XX \approx XY$ and $Y \approx YY \approx YX$

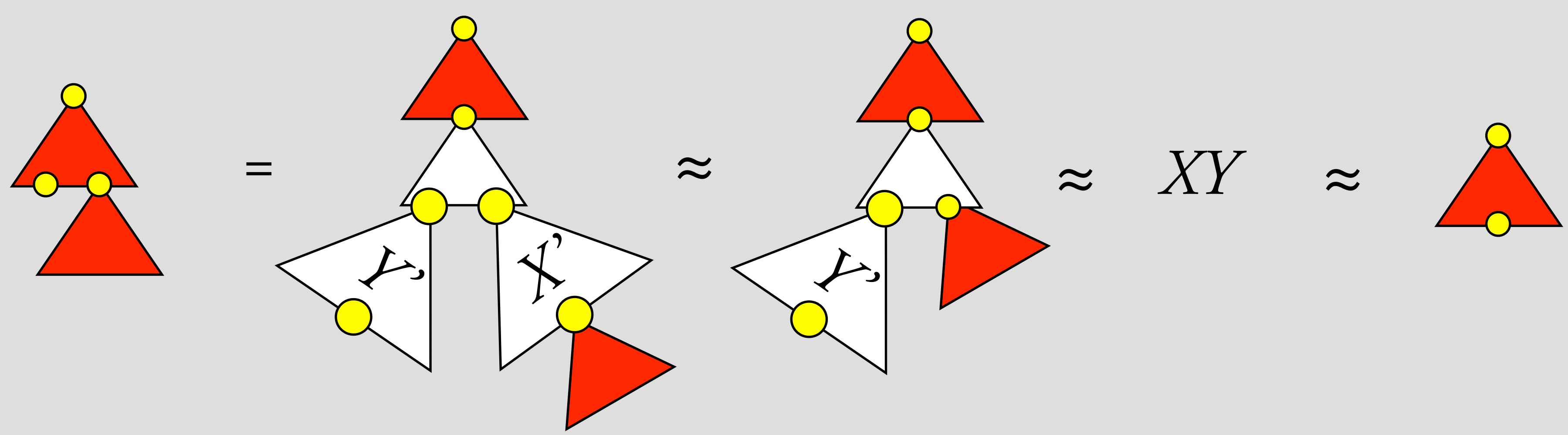
choose:

 $= X$



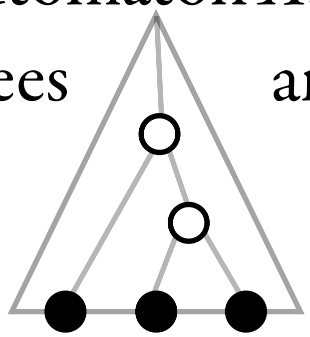
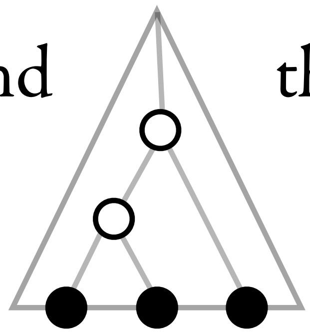
To prove the pattern lemma, it suffices to show:





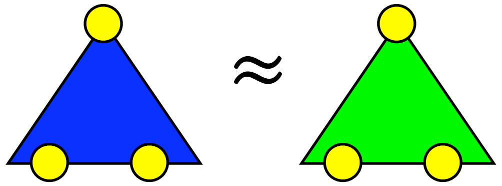
Goal: No deterministic tree-walking automaton recognizes the language L.

Fix a deterministic tree-walking automaton A.

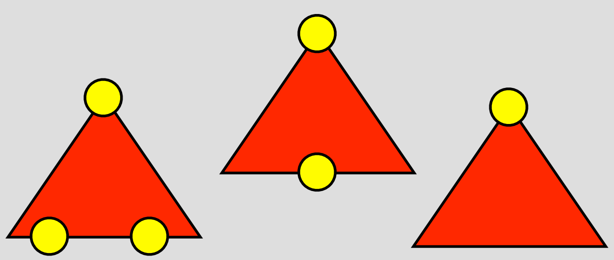
We will find trees  and  that cannot be distinguished by A.

Strategy:

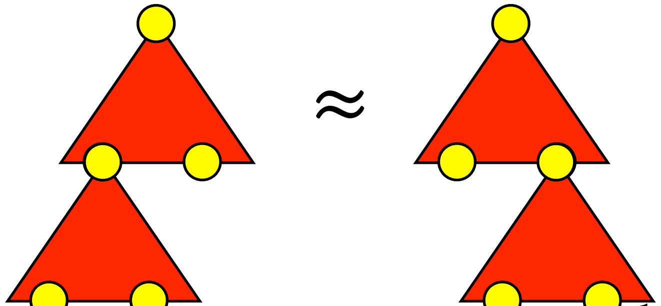
1 Define notion of pattern, together with pattern equivalence



2 Using algebra, find some confusing patterns

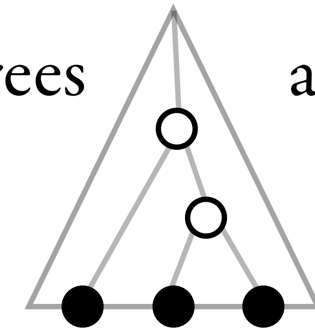
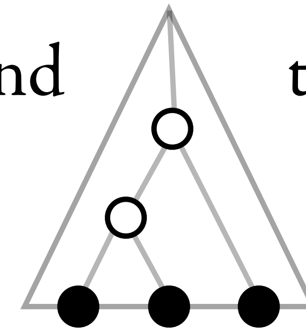


3 Build the counterexample using these confusing patterns



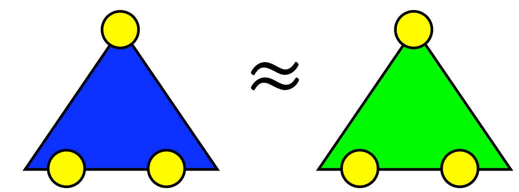
Goal: No deterministic tree-walking automaton recognizes the language L.

Fix a deterministic tree-walking automaton A.

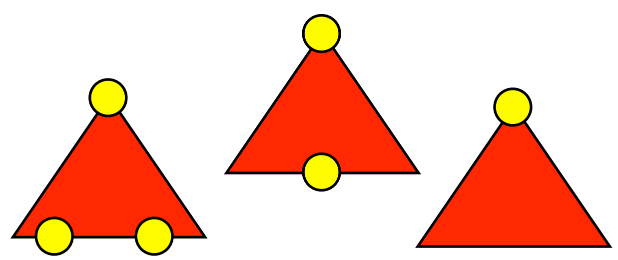
We will find trees  and  that cannot be distinguished by A.

Strategy:

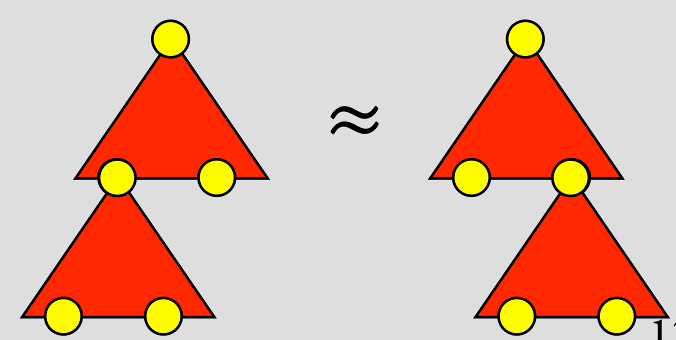
1 Define notion of pattern, together with pattern equivalence



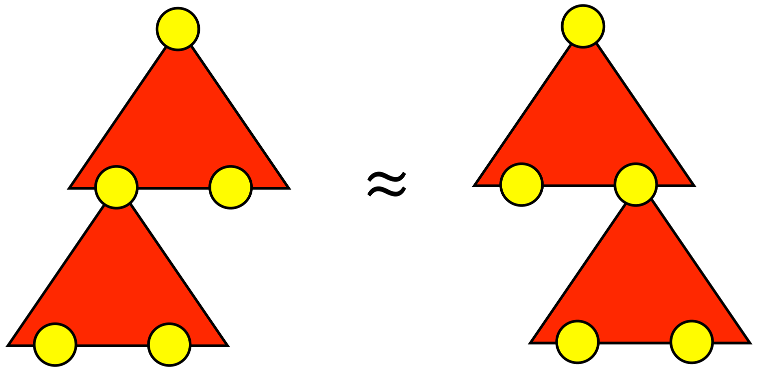
2 Using algebra, find some confusing patterns



3 Build the counterexample using these confusing patterns

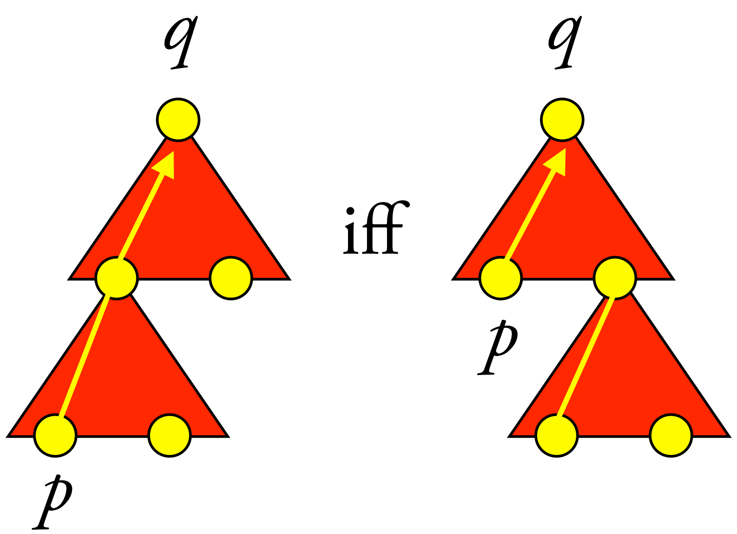


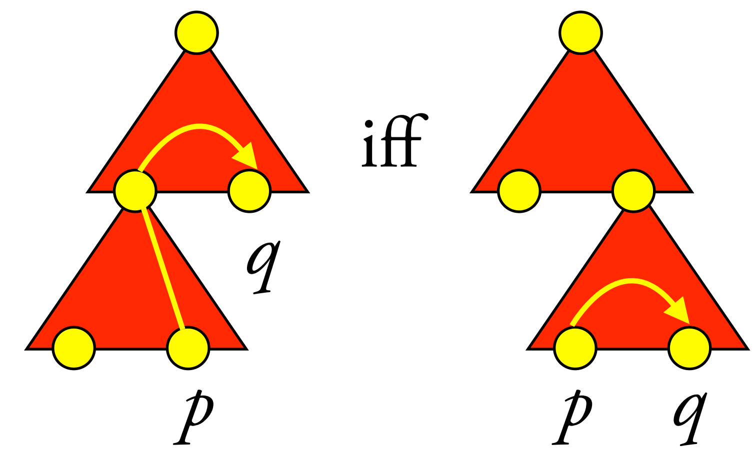
we will show that

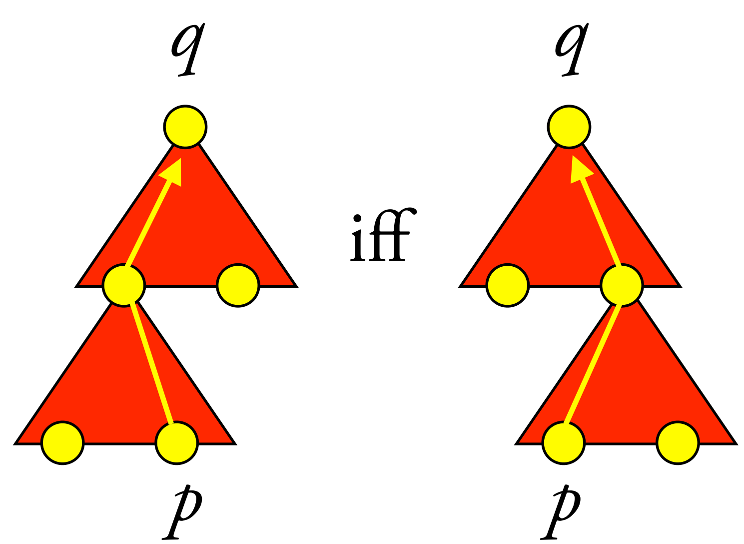


holds for a deterministic tree-walking automaton

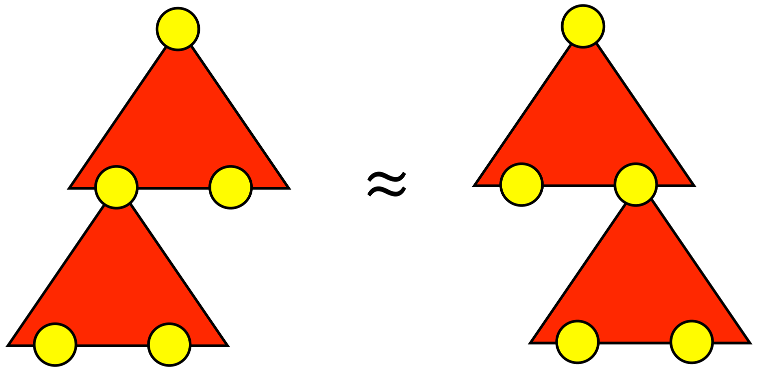
type of things we need to show:





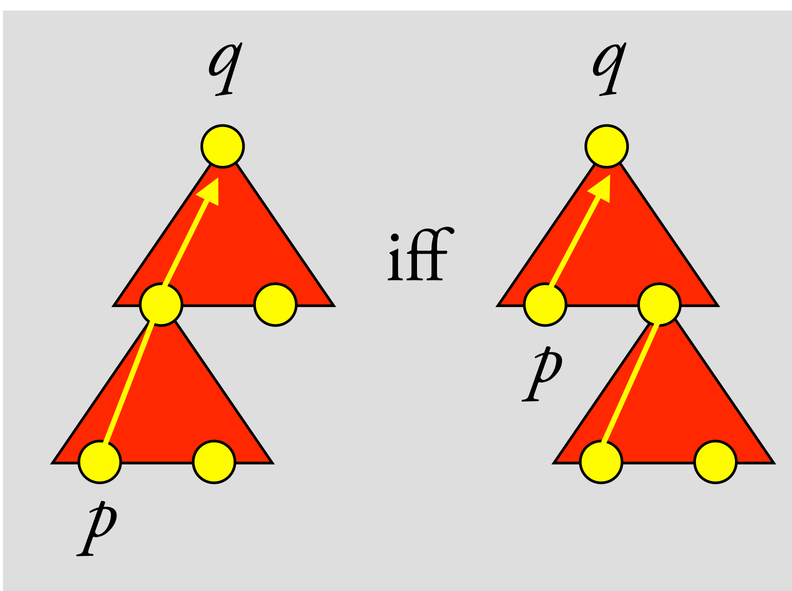


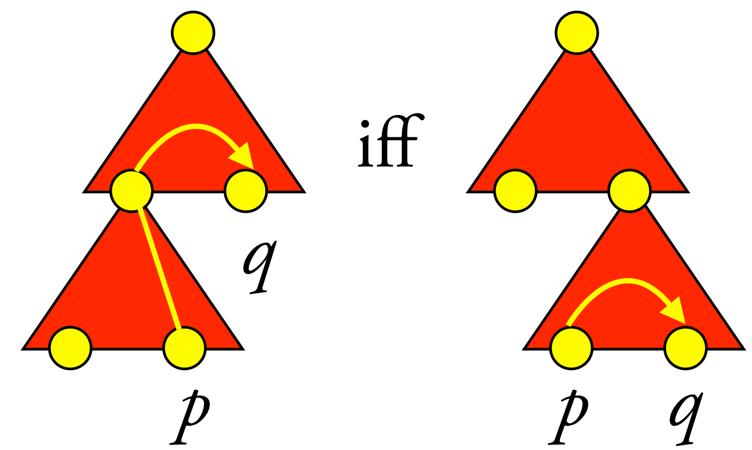
we will show that

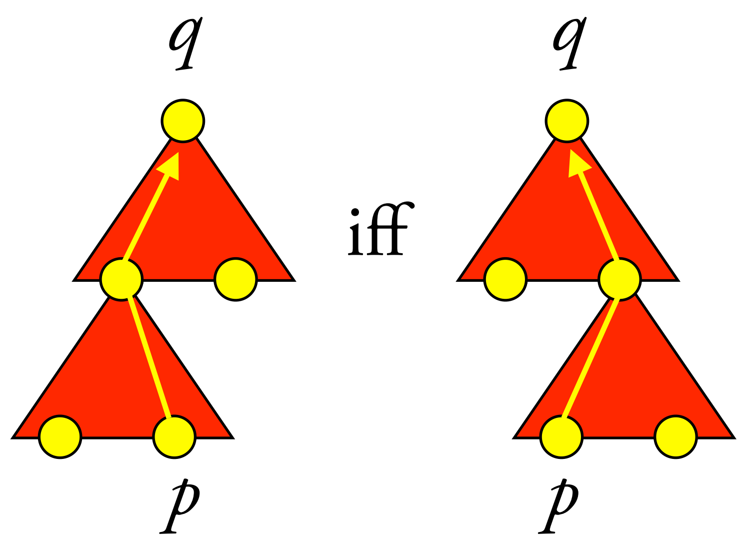


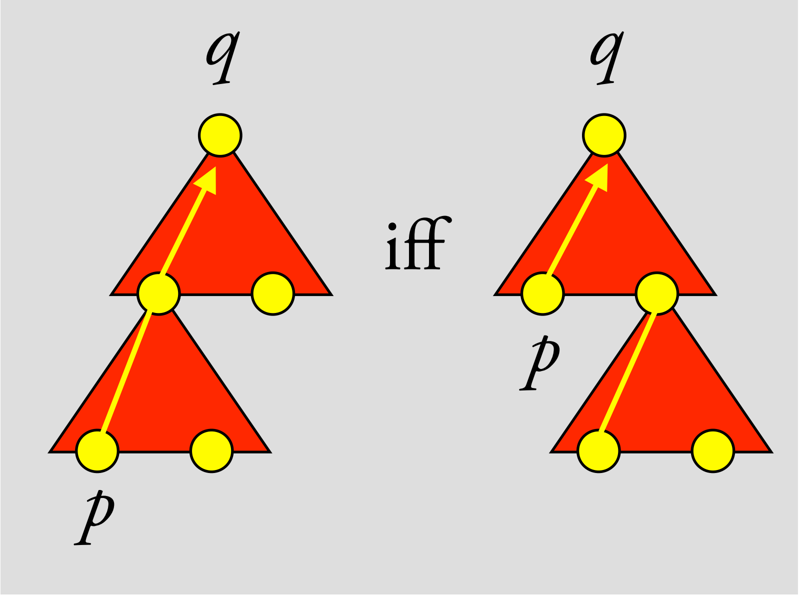
holds for a deterministic tree-walking automaton

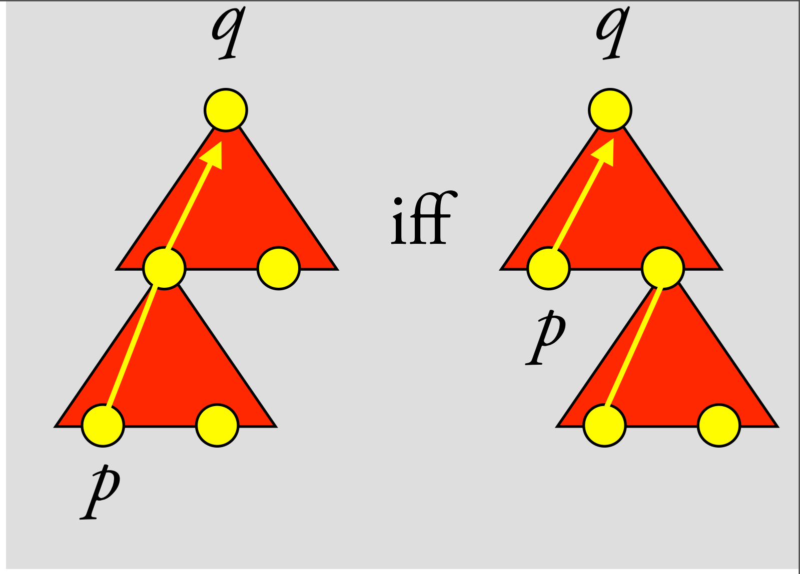
type of things we need to show:



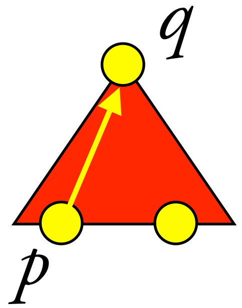




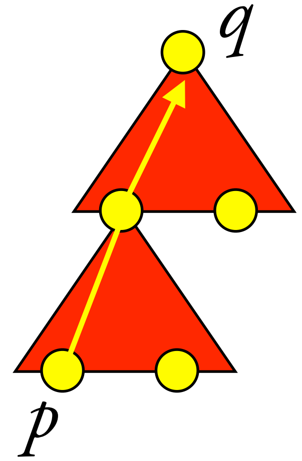

q
iff
q
p
p

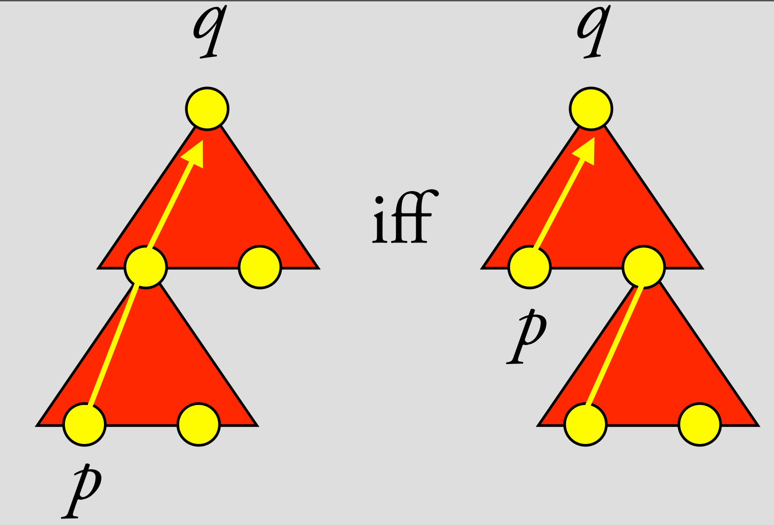

q
q
iff
p
p

Lemma

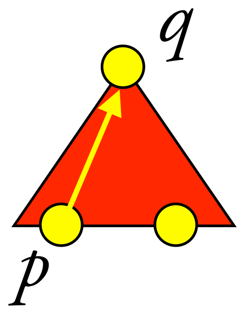


implies

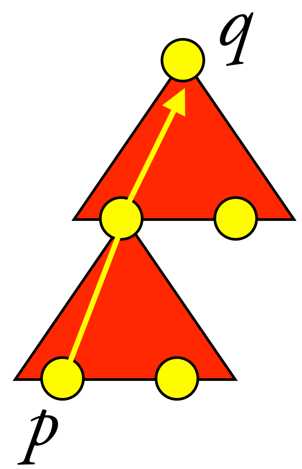


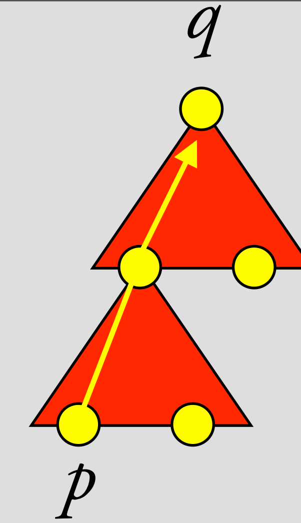


Lemma

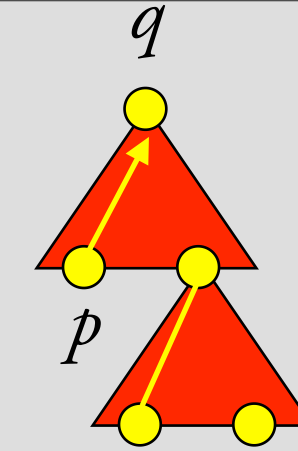


implies



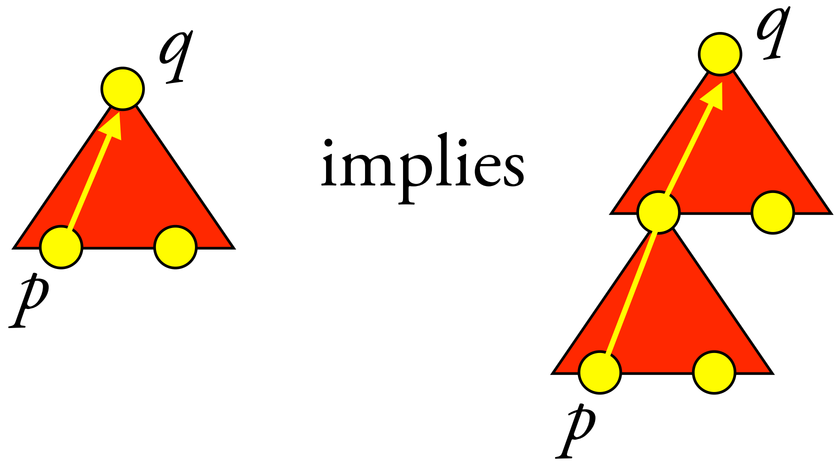


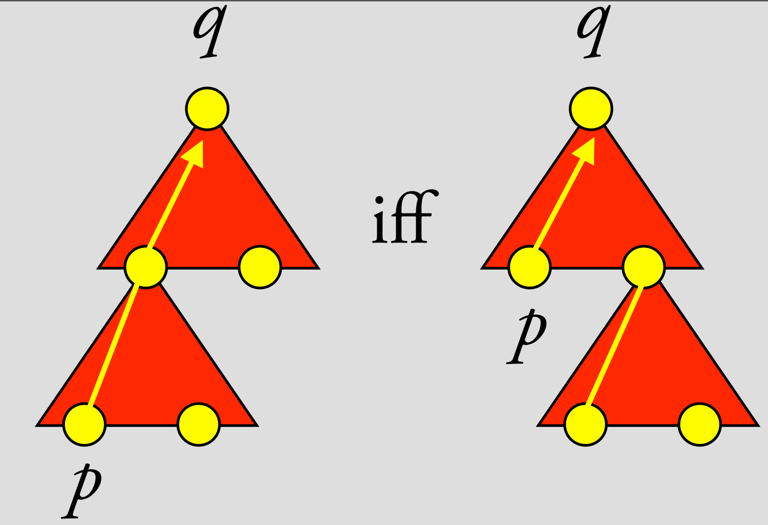
iff



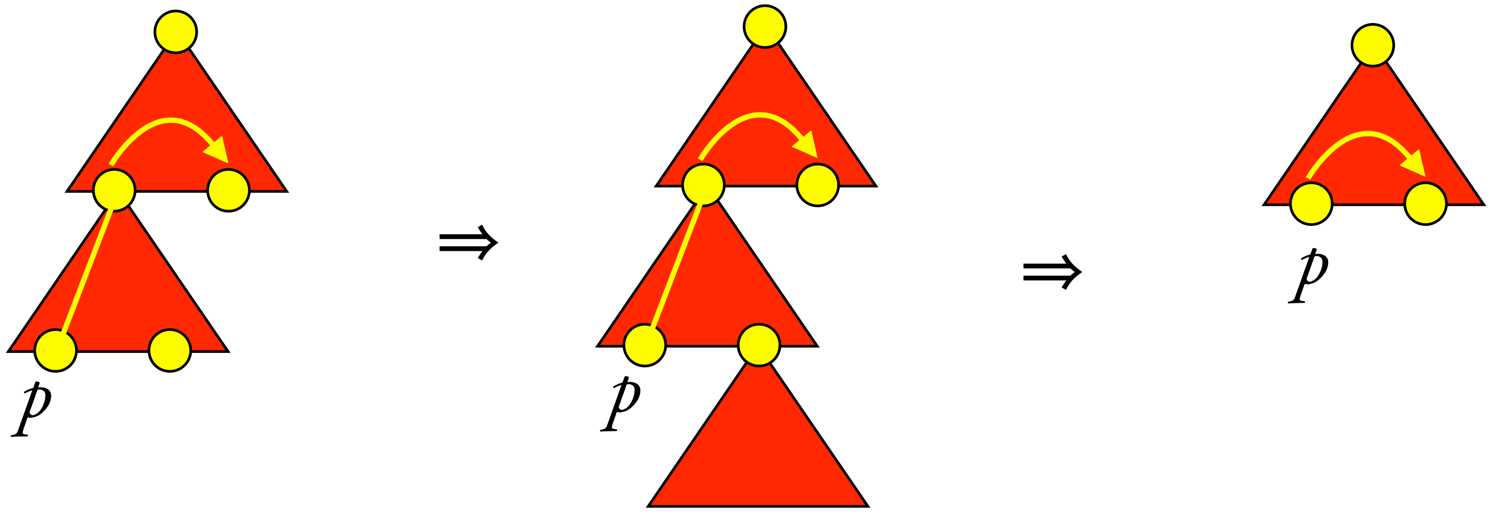
Proof

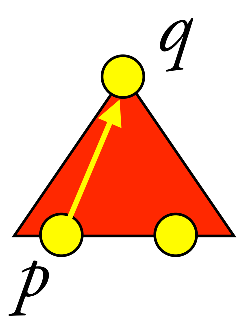
Lemma

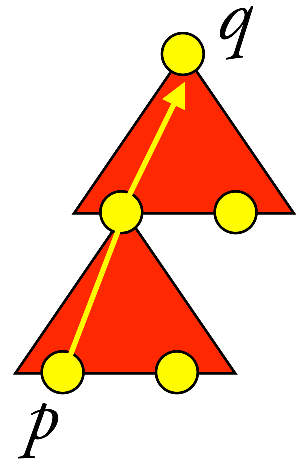


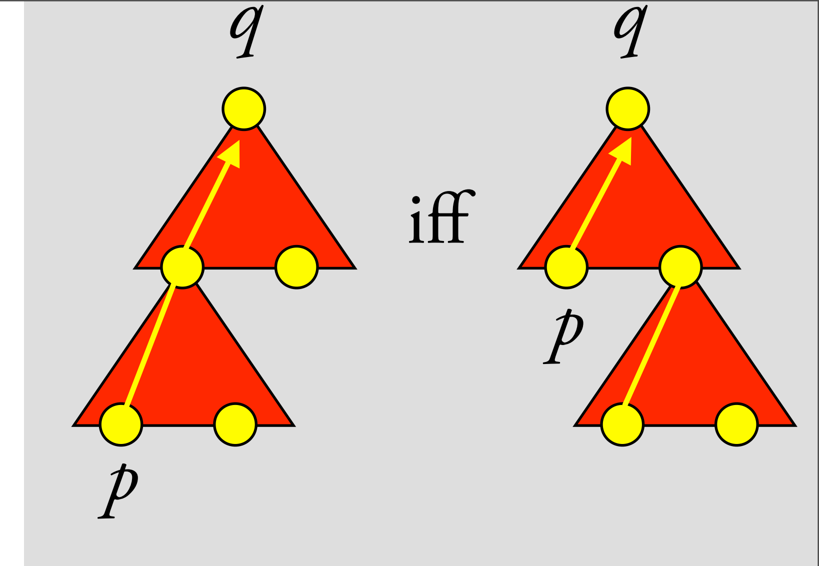


Proof

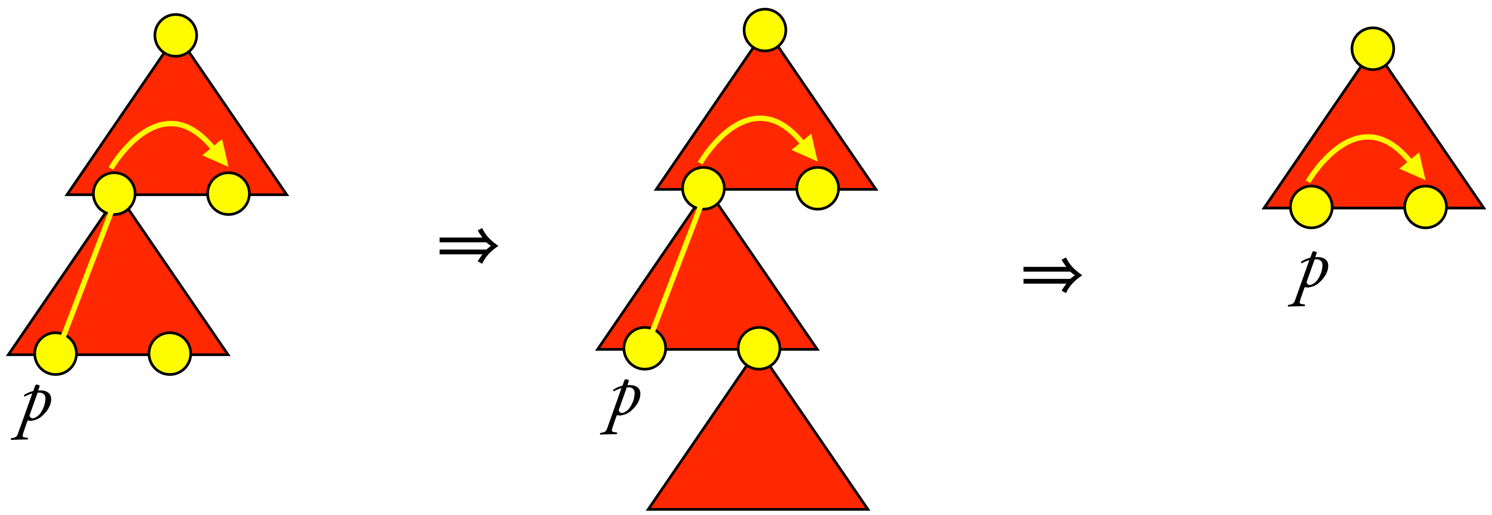


Lemma 

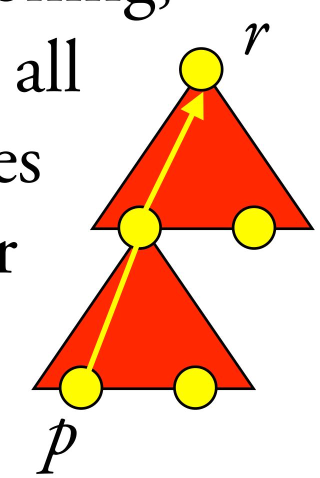
implies 



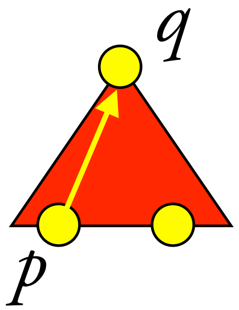
Proof



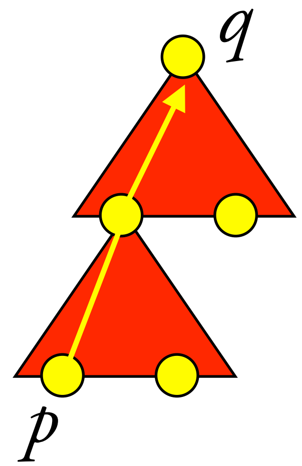
by similar reasoning, we rule out all possibilities except for

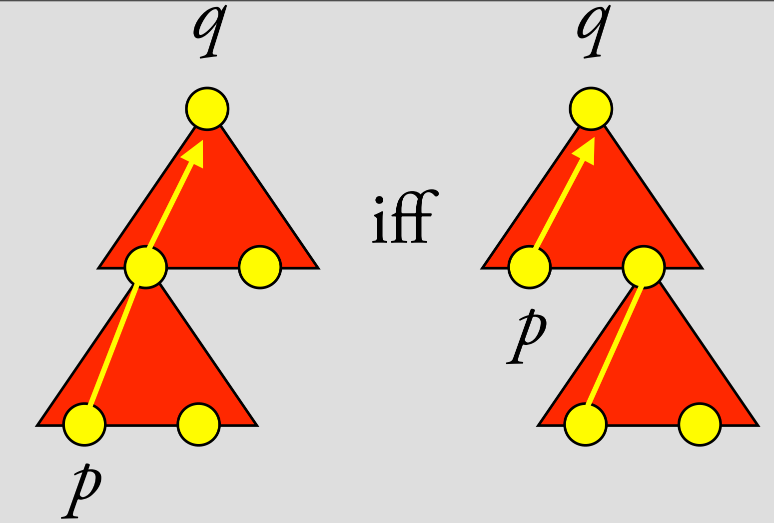


Lemma

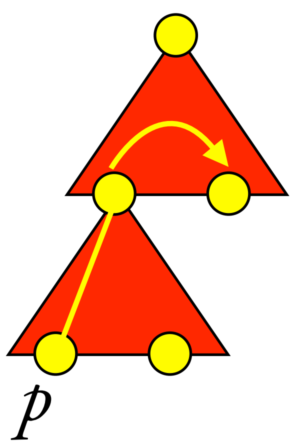


implies

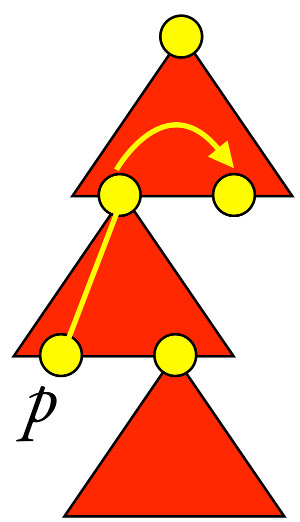




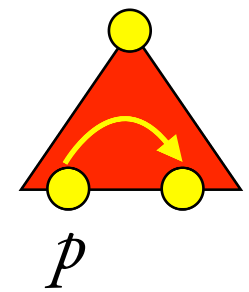
Proof



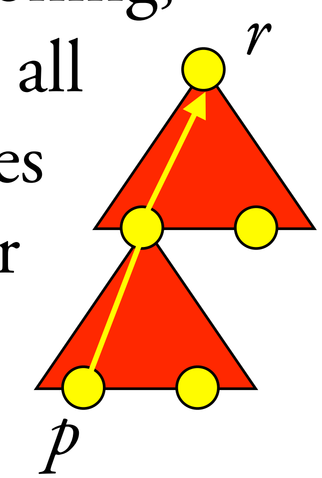
⇒



⇒

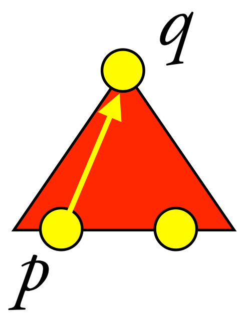


by similar reasoning,
we rule out all
possibilities
except for

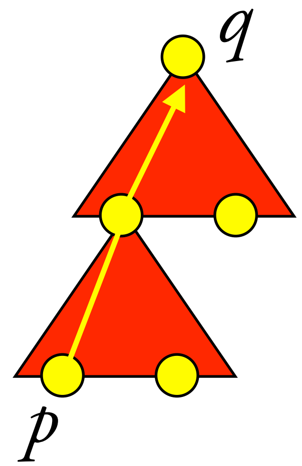


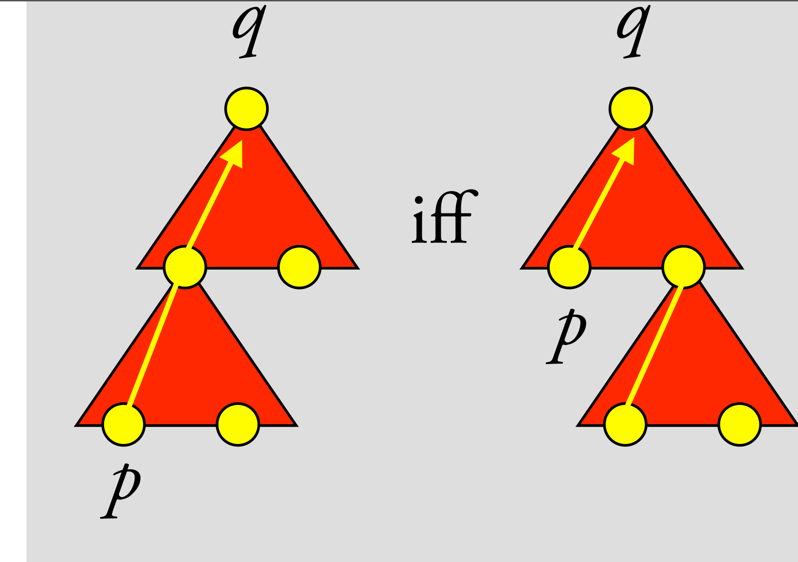
why $r=q$?

Lemma

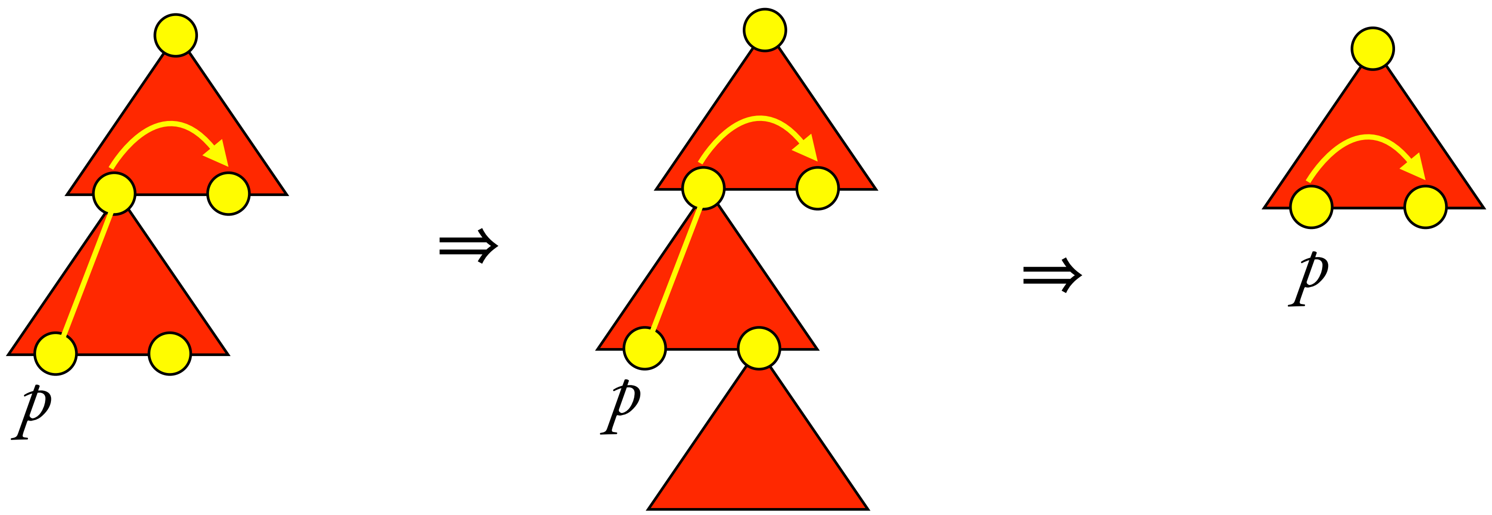


implies

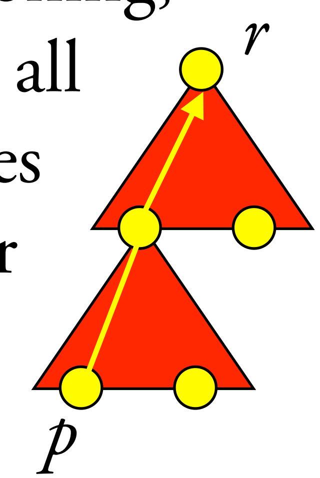




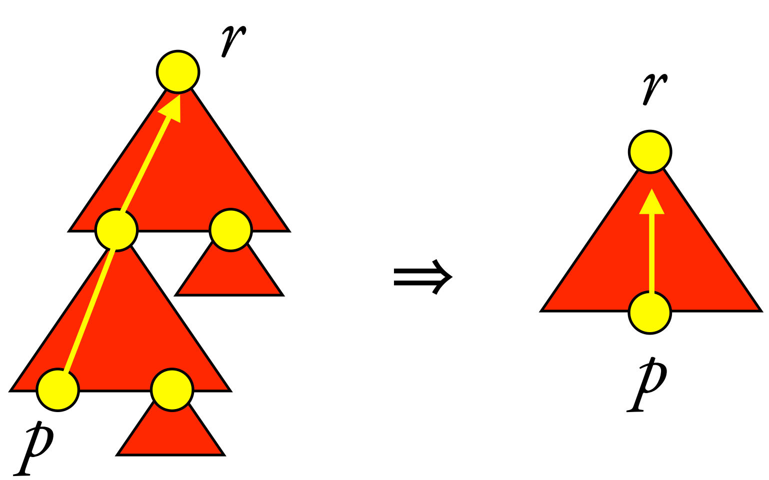
Proof

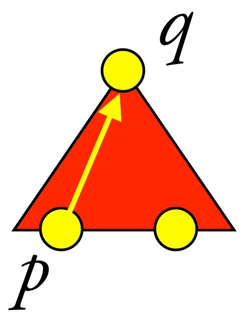


by similar reasoning, we rule out all possibilities except for

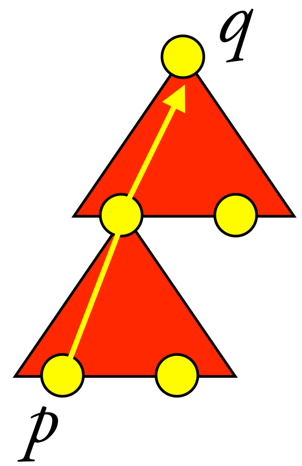


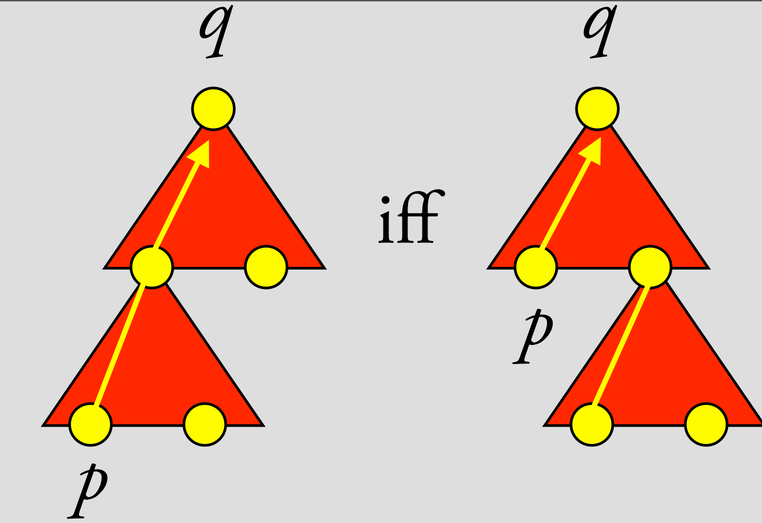
why $r=q$?



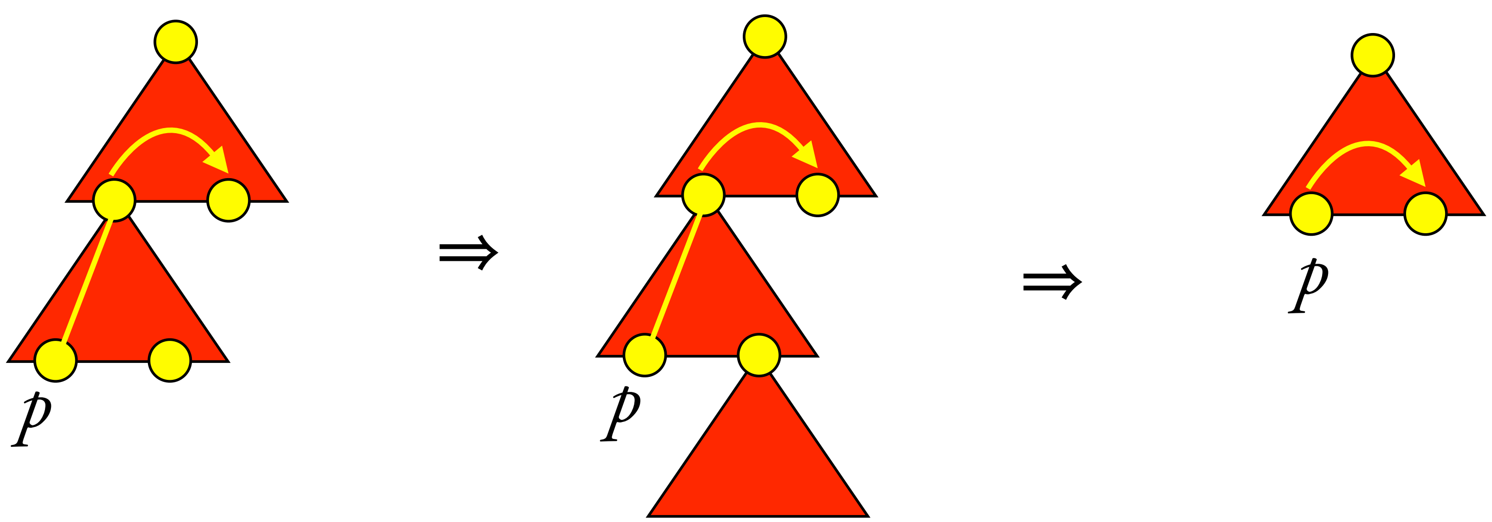
Lemma 

implies

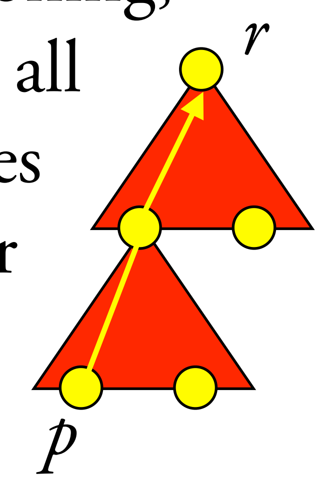




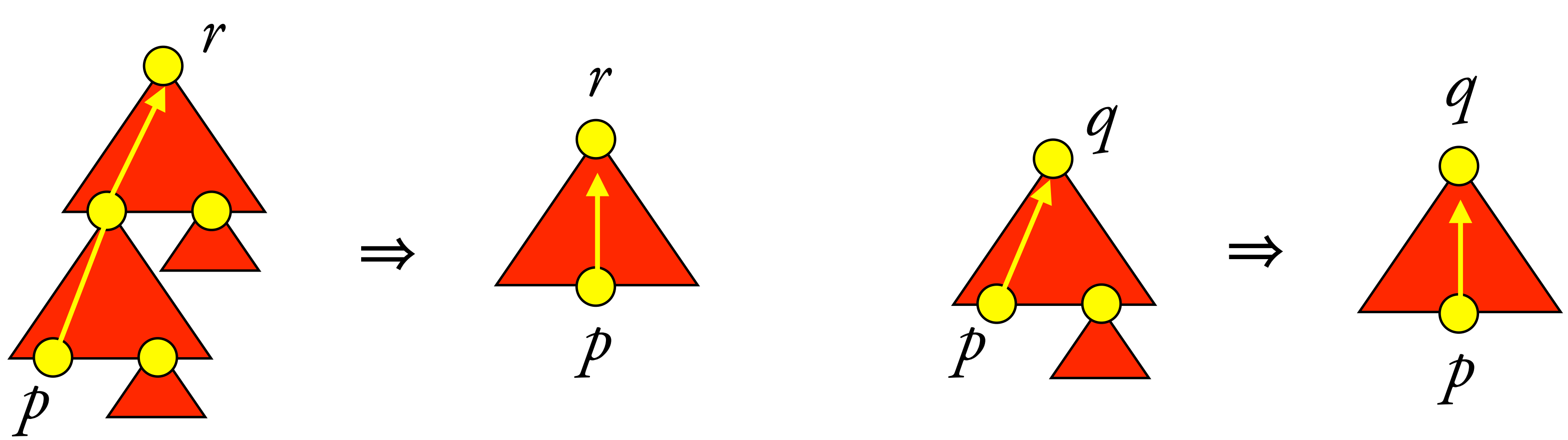
Proof



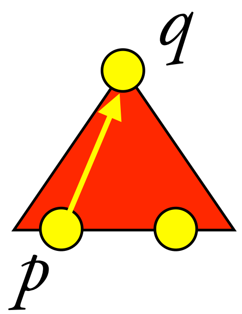
by similar reasoning, we rule out all possibilities except for



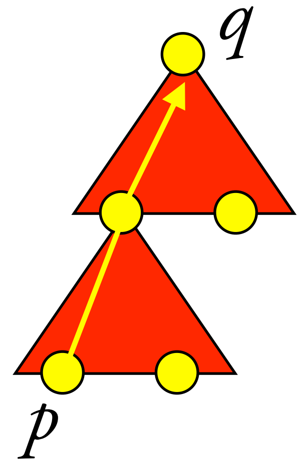
why $r=q$?

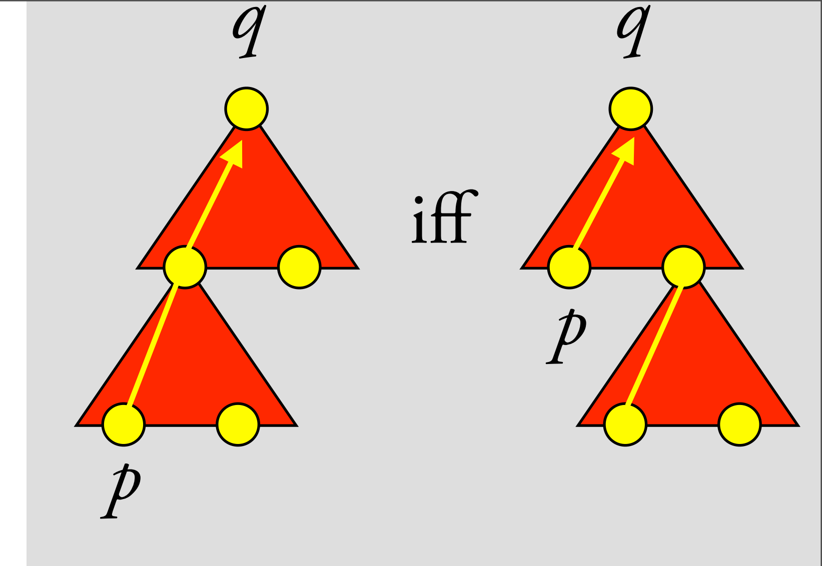


Lemma

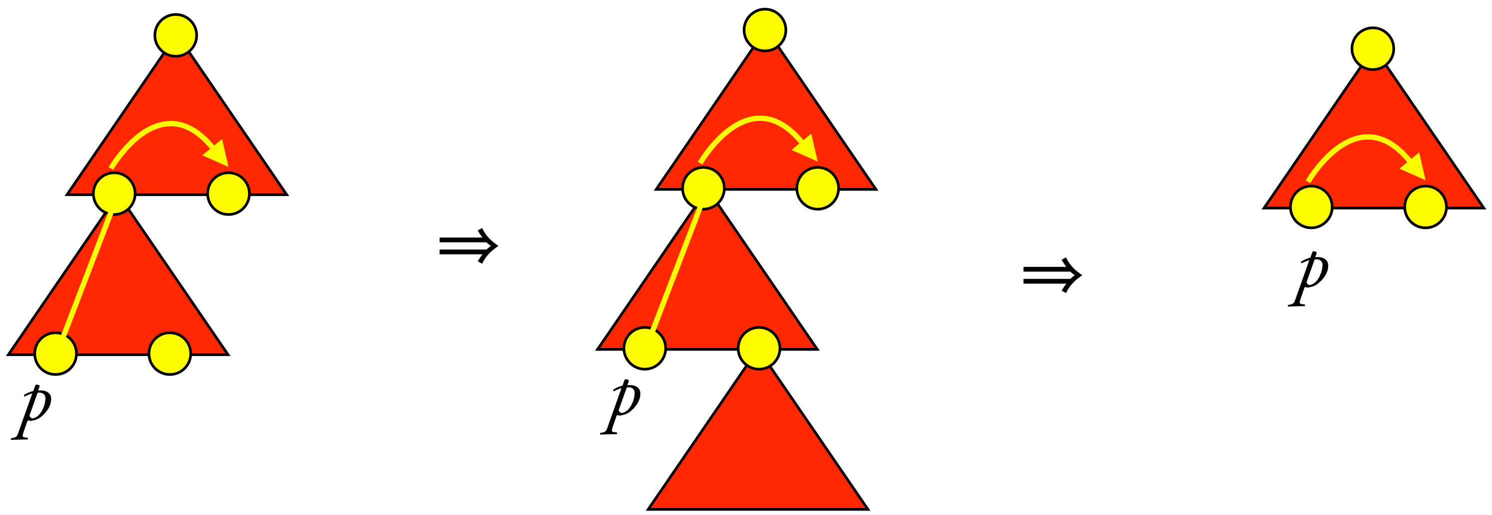


implies

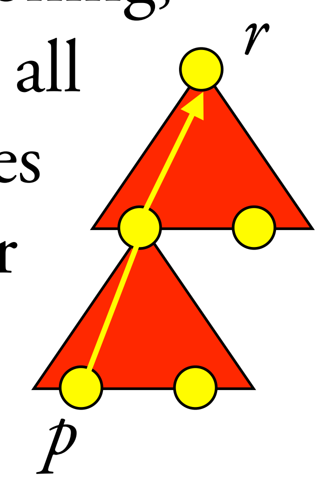




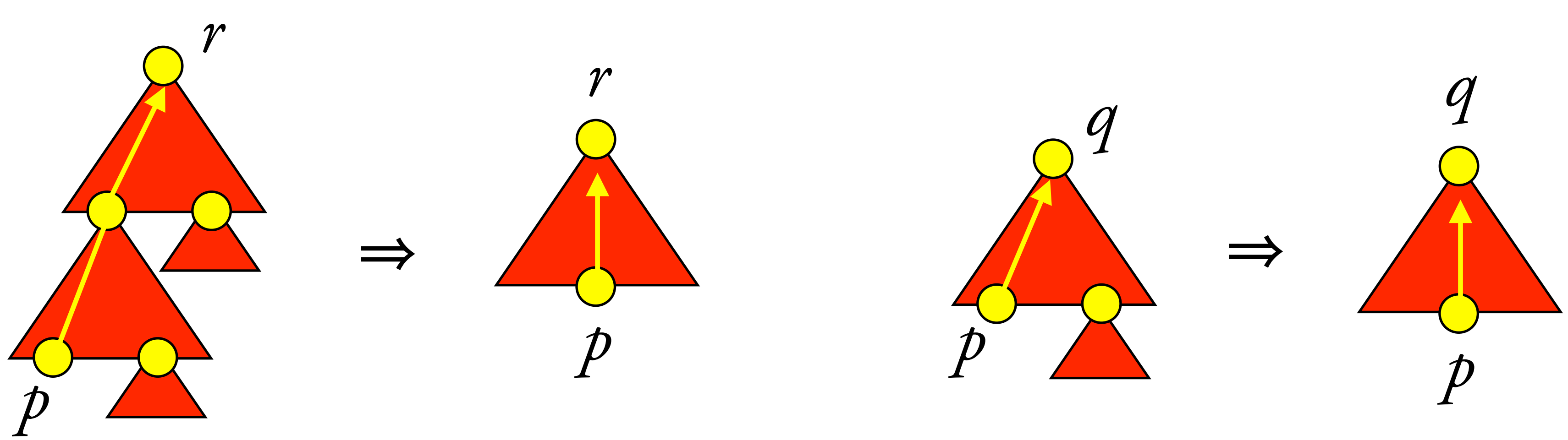
Proof



by similar reasoning, we rule out all possibilities except for

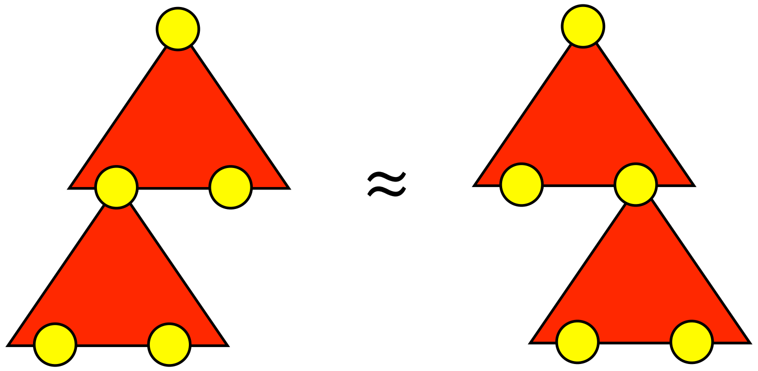


why $r=q$?



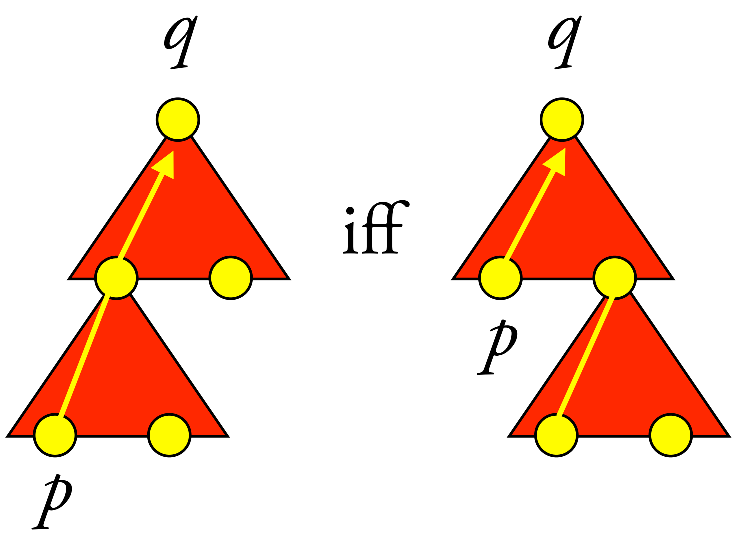
by determinism, we get $q=r$

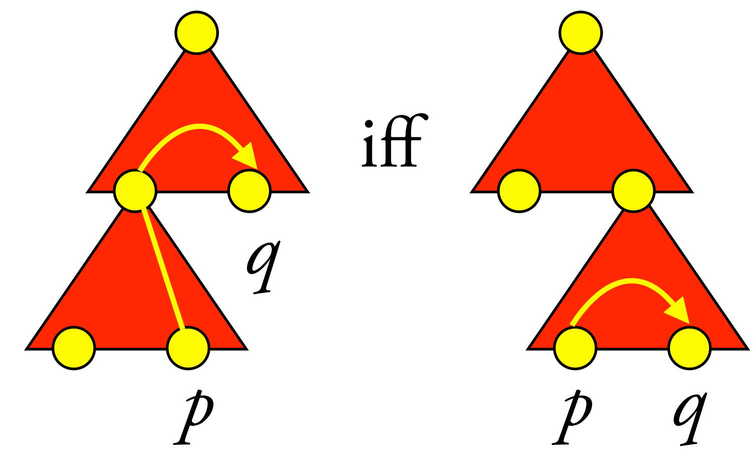
we will show that

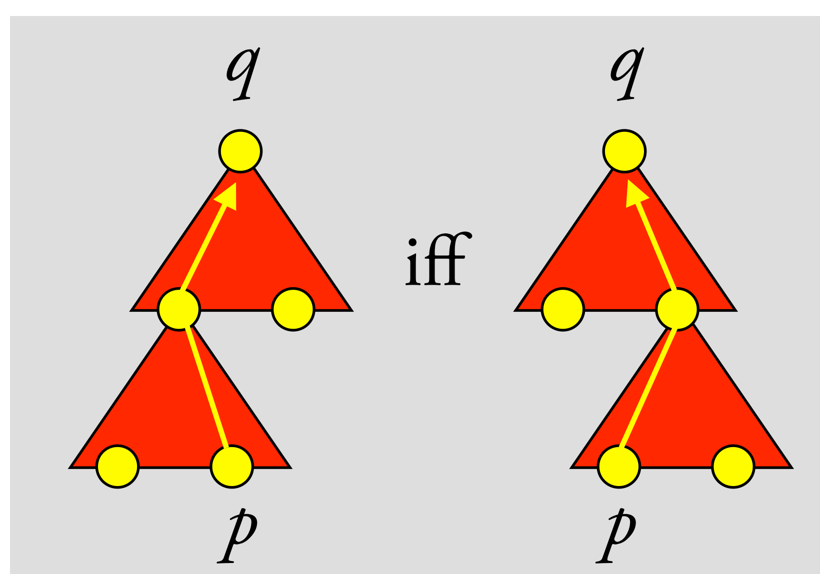


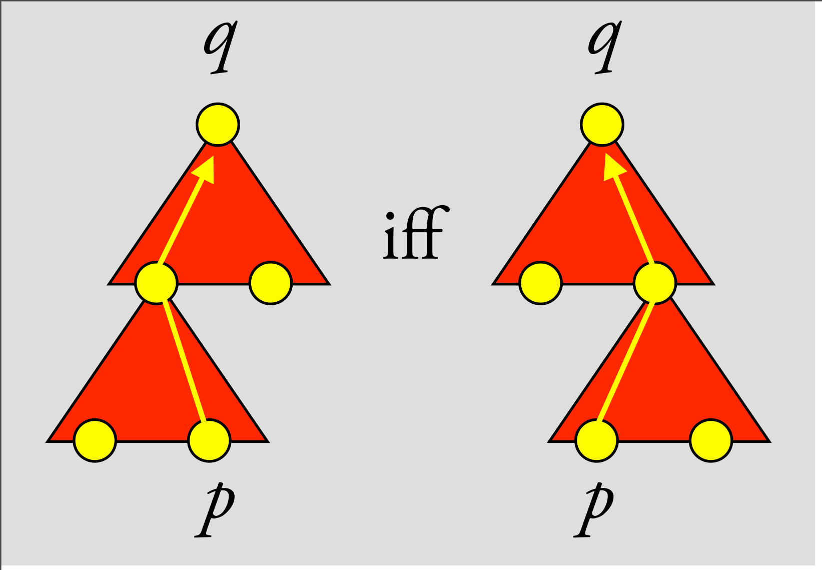
holds for a deterministic tree-walking automaton

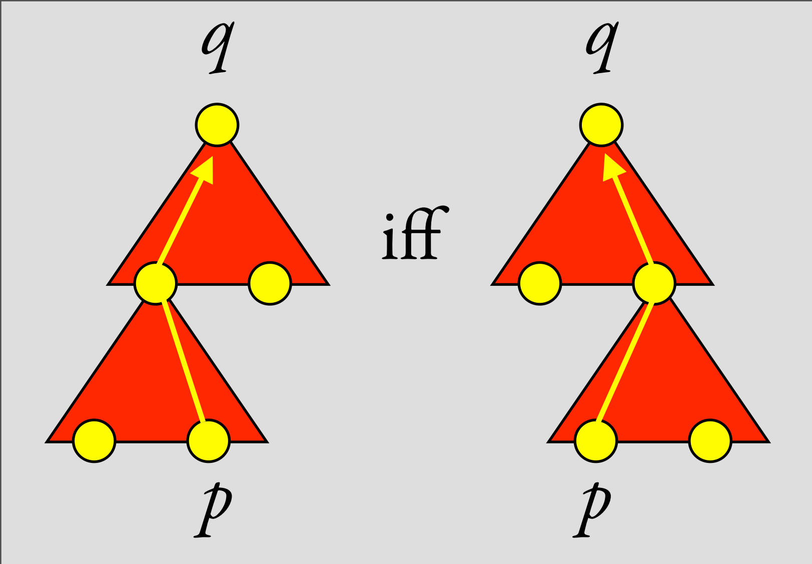
type of things we need to show:



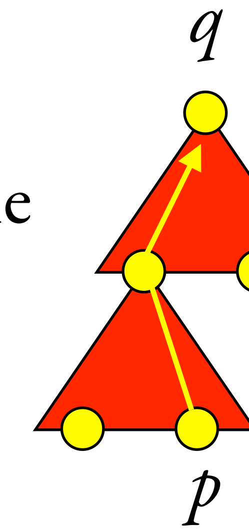


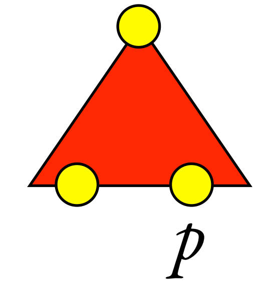



q
q
iff
p
p



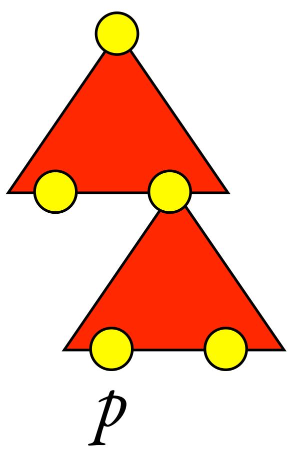
assume

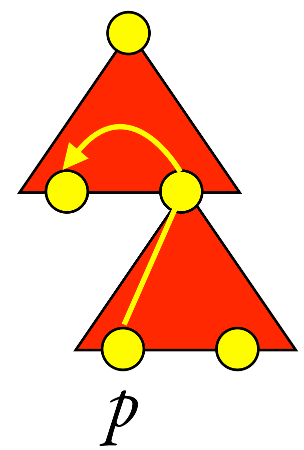


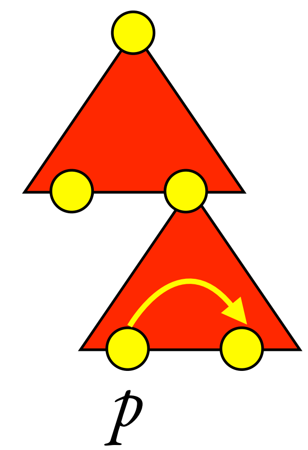


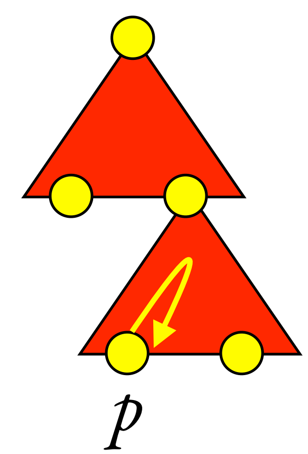
impossible

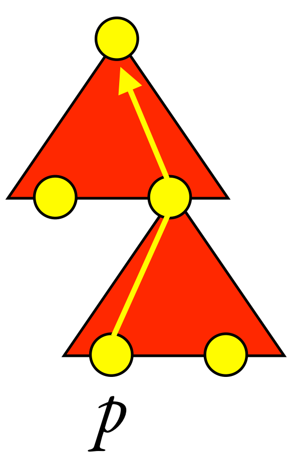
what happens in this situation?

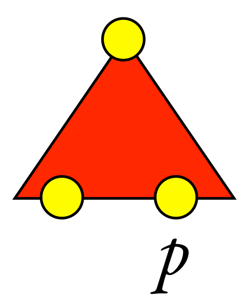




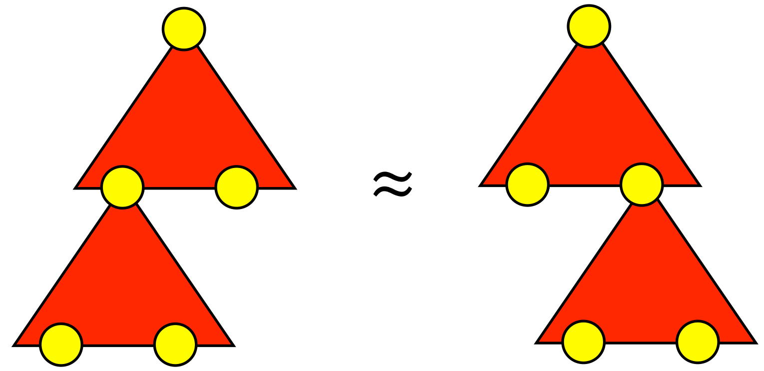




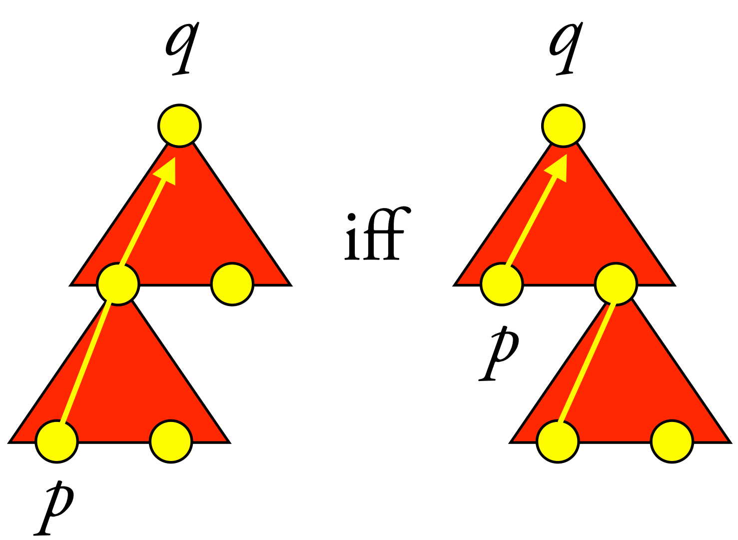


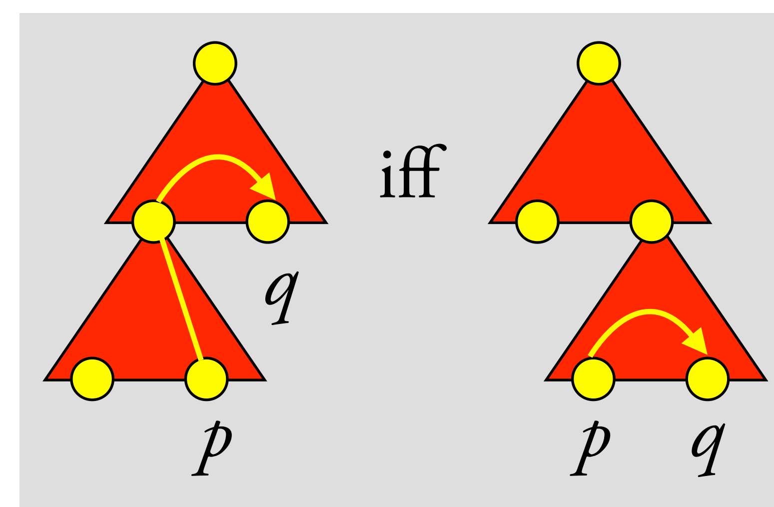


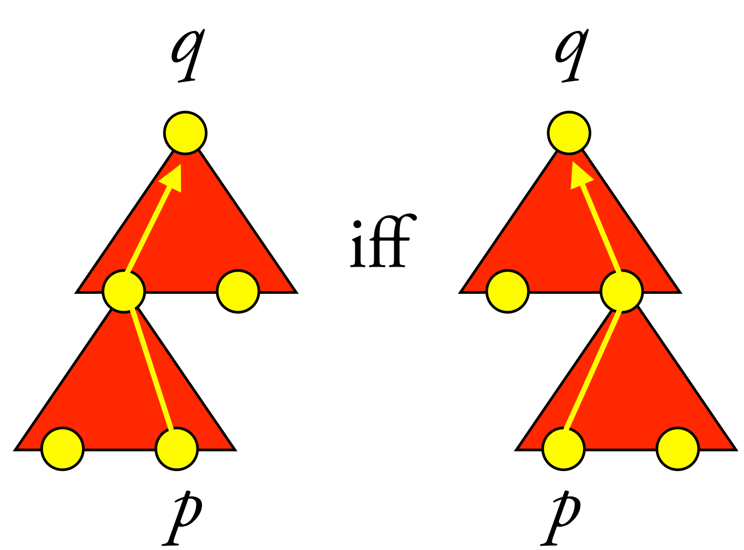
impossible

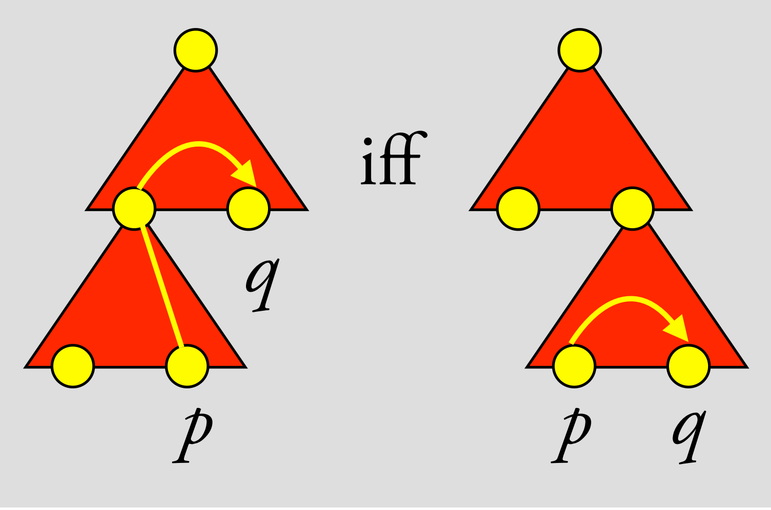
we will show that  holds for a deterministic tree-walking automaton

type of things we need to show:

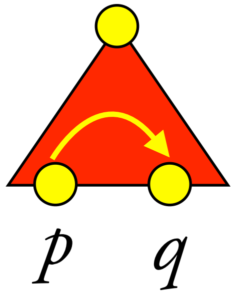


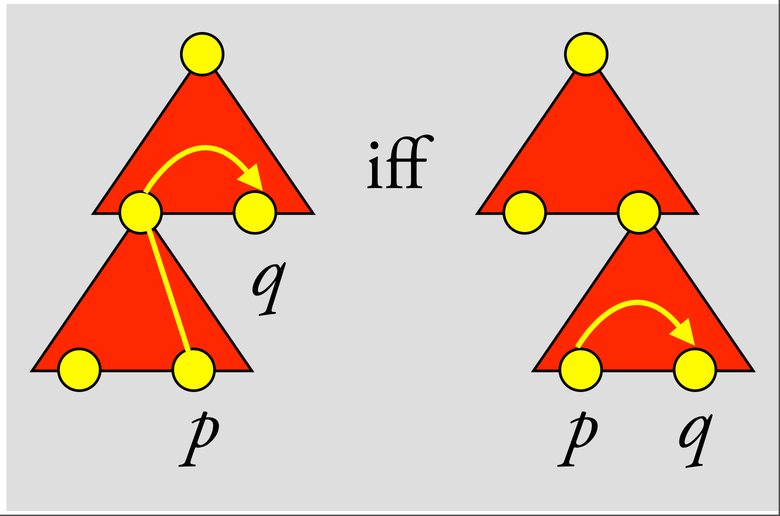




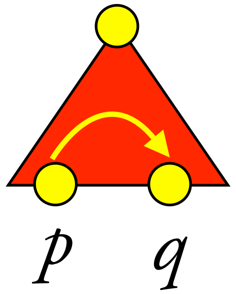

iff
q
p
p
q

assume

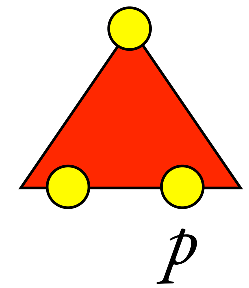


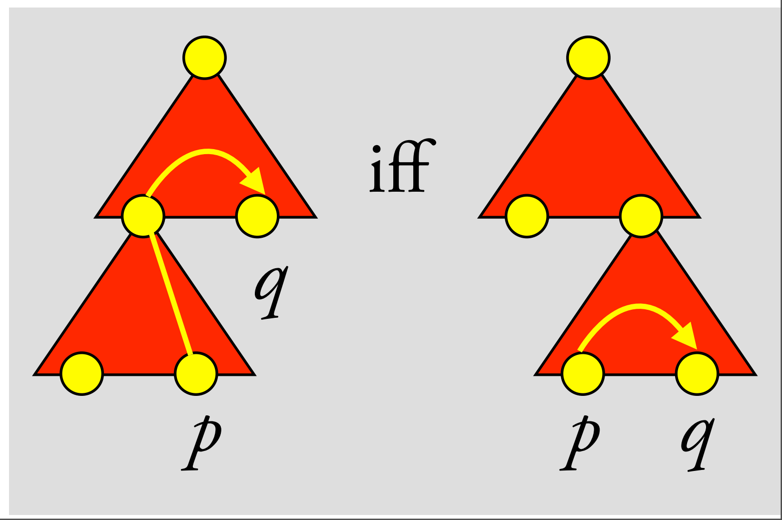


assume

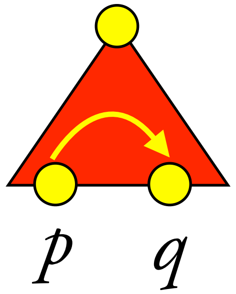


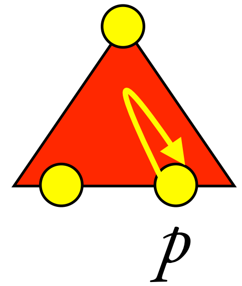
what happens
in this situation?





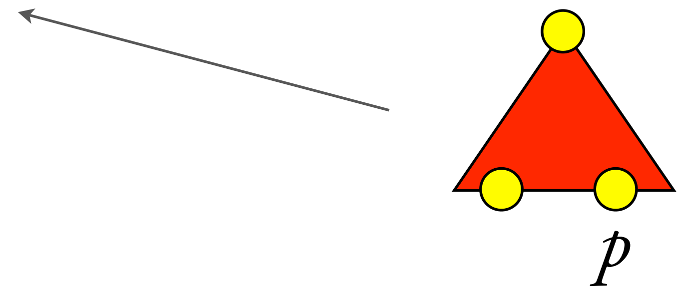
assume

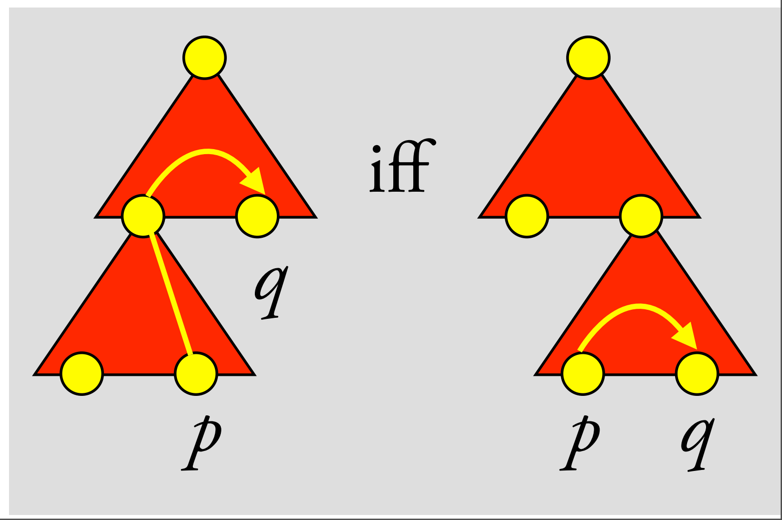




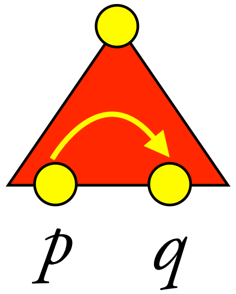
impossible

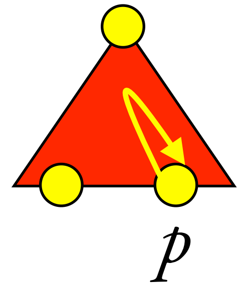
what happens
in this situation?





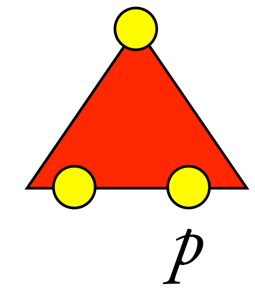
assume

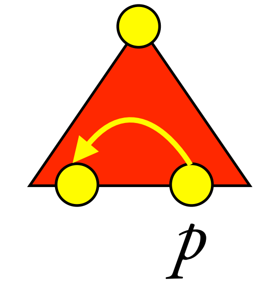




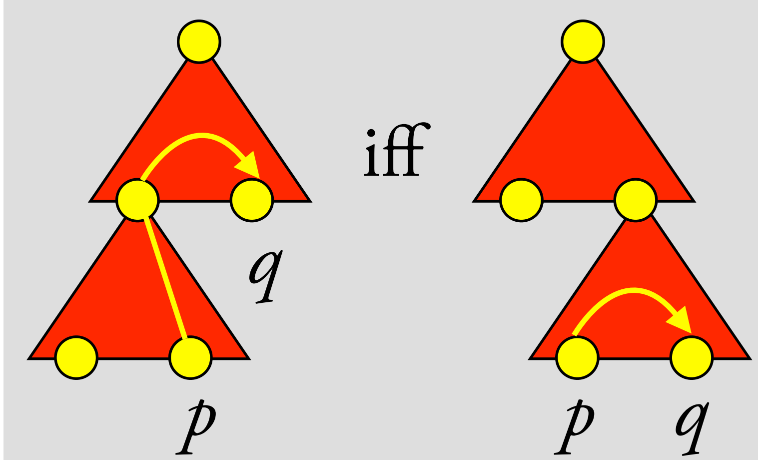
impossible

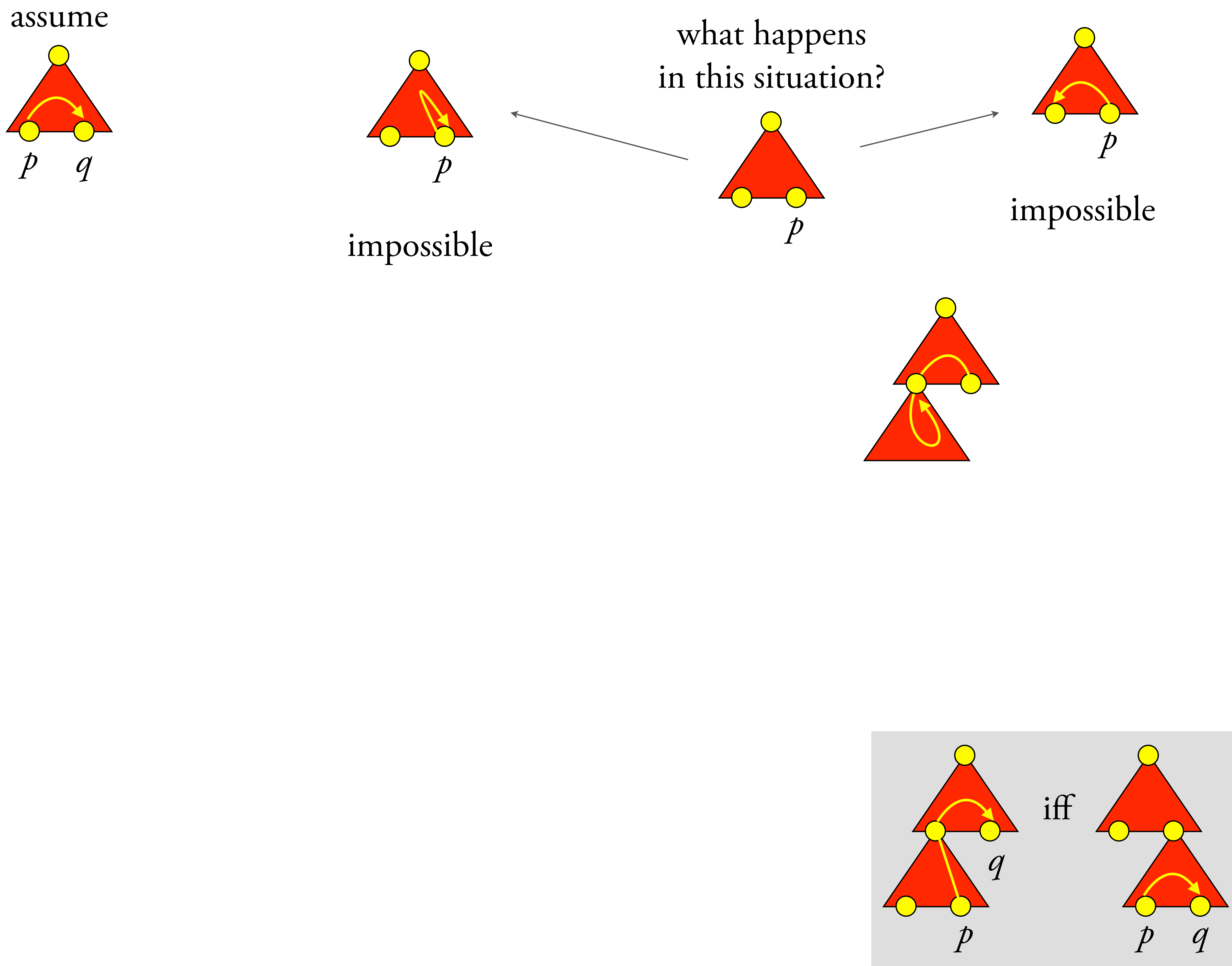
what happens
in this situation?

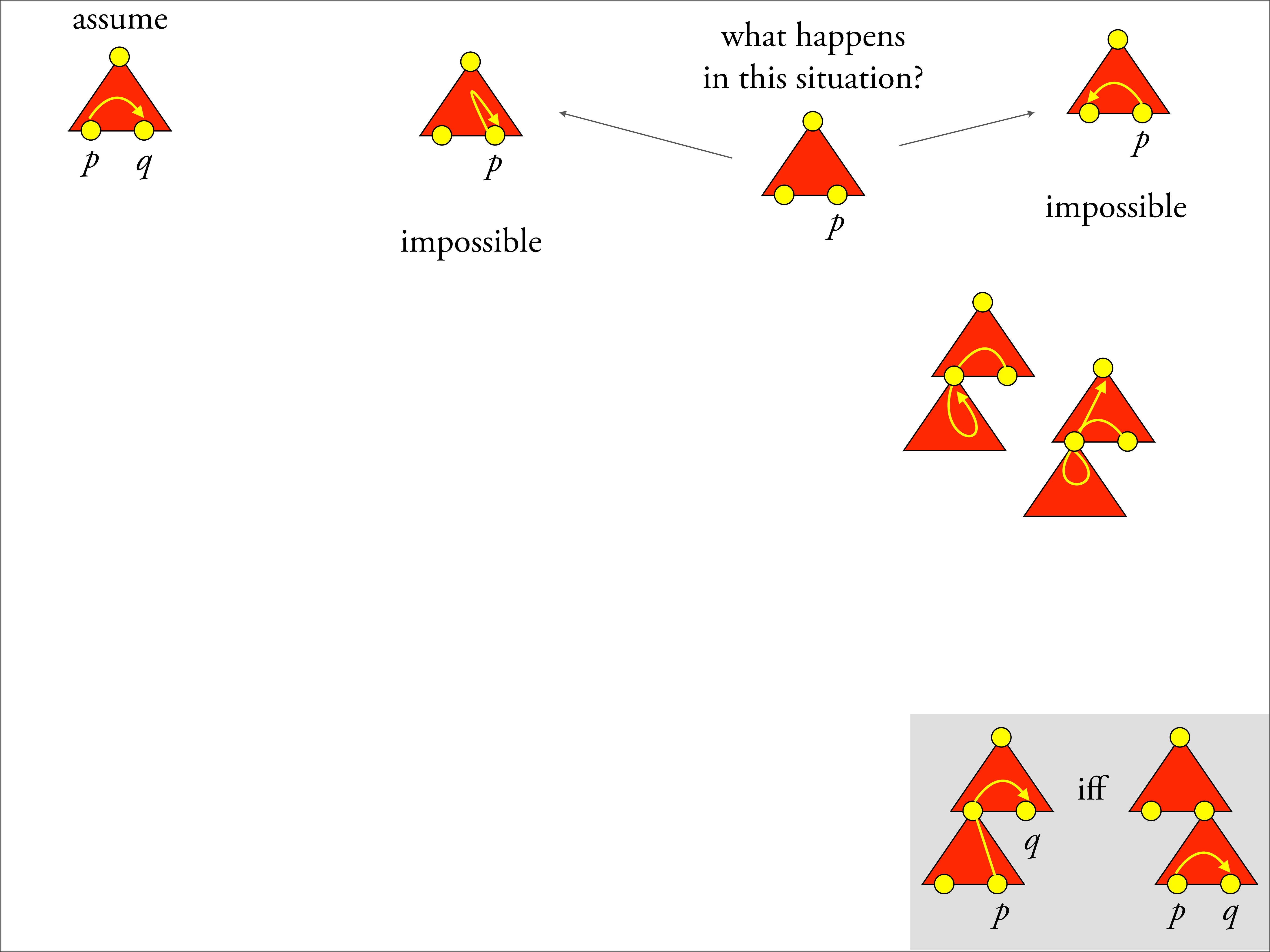




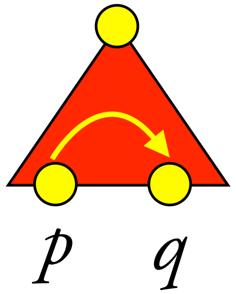
impossible

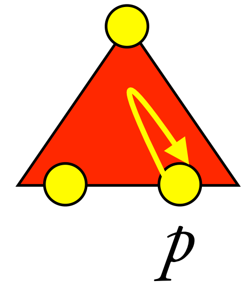



assume
p
q
p
impossible
what happens
in this situation?
p
p
impossible
iff
q
p
p
q


assume
p
q
p
impossible
what happens
in this situation?
p
p
impossible
iff
q
p
p
q

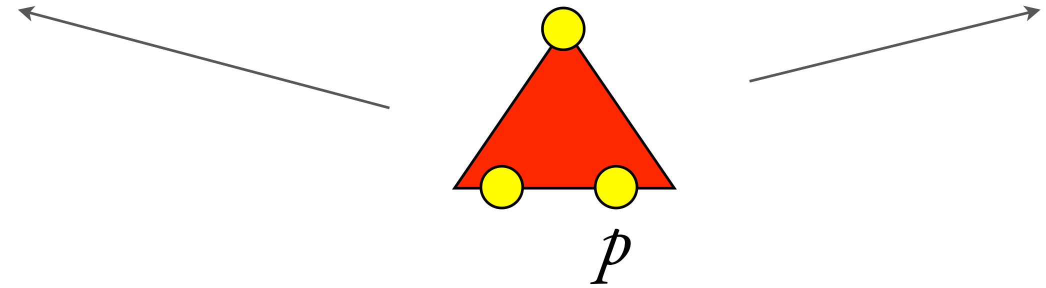
assume

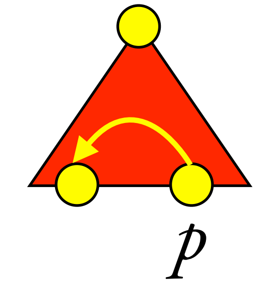




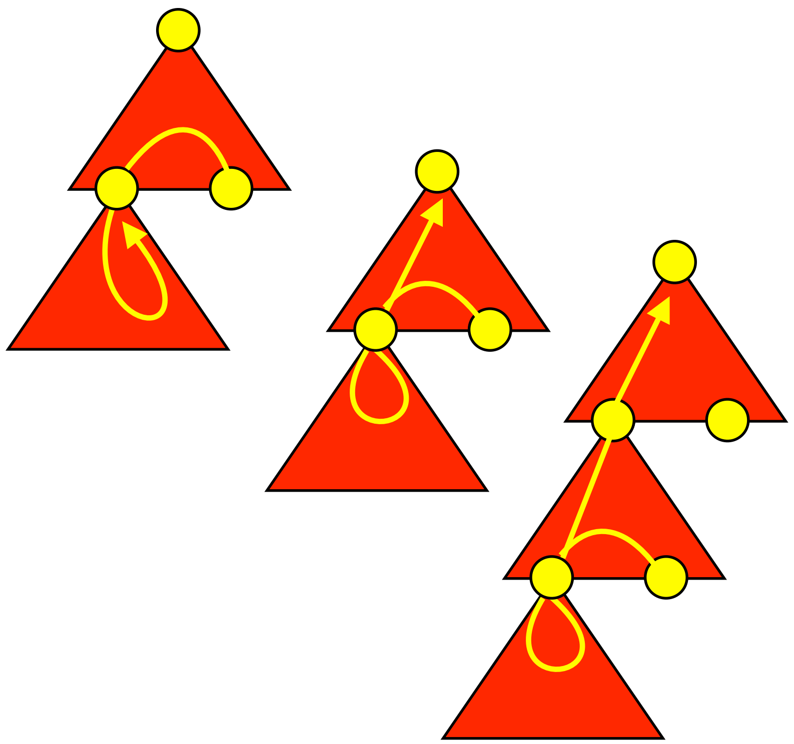
impossible

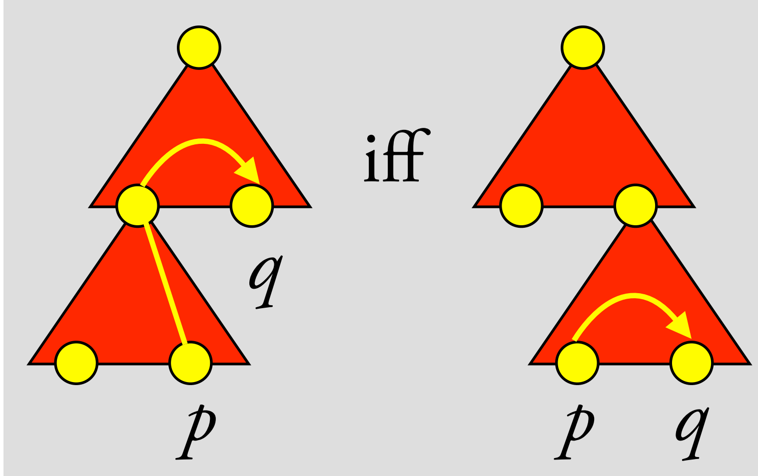
what happens
in this situation?

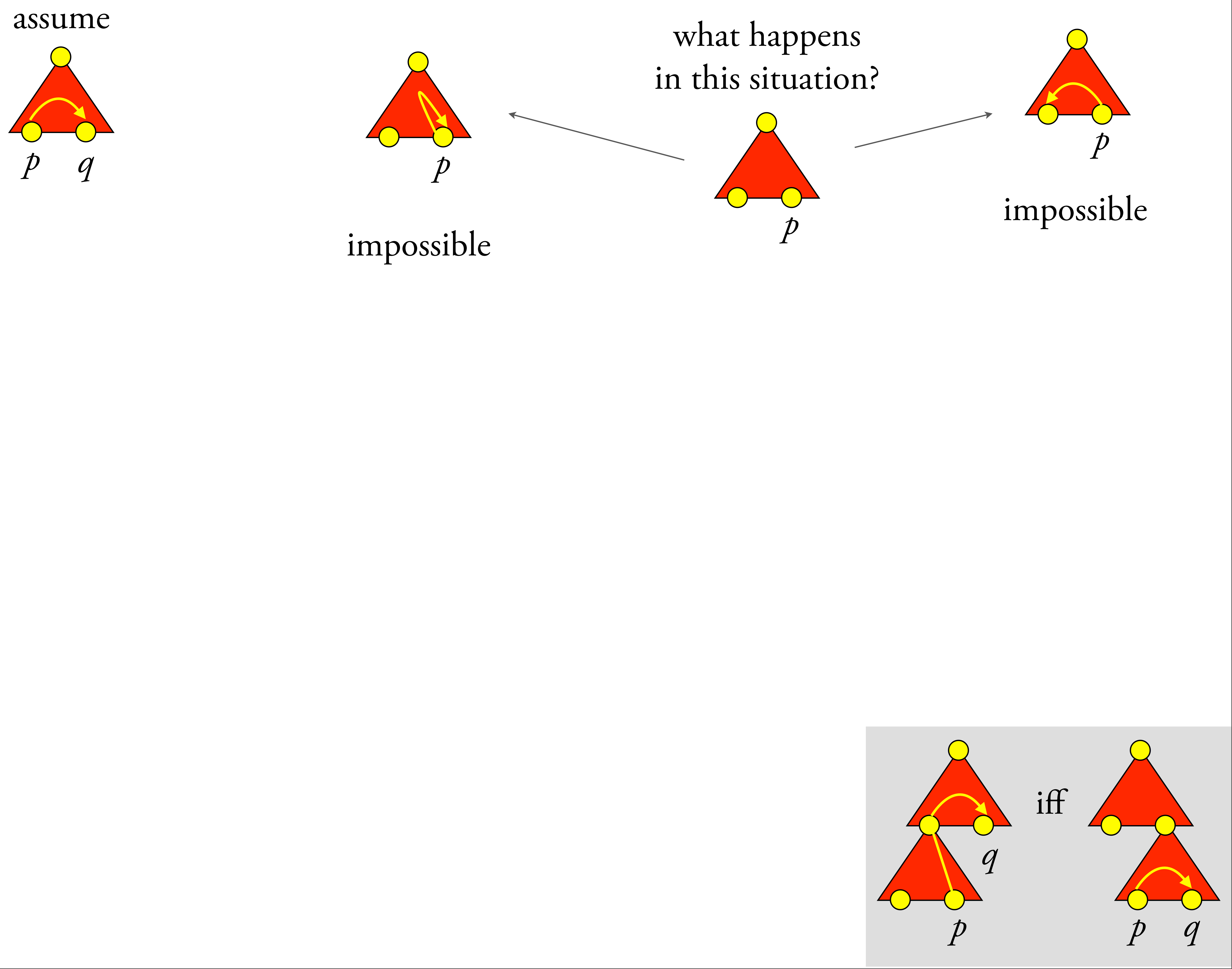




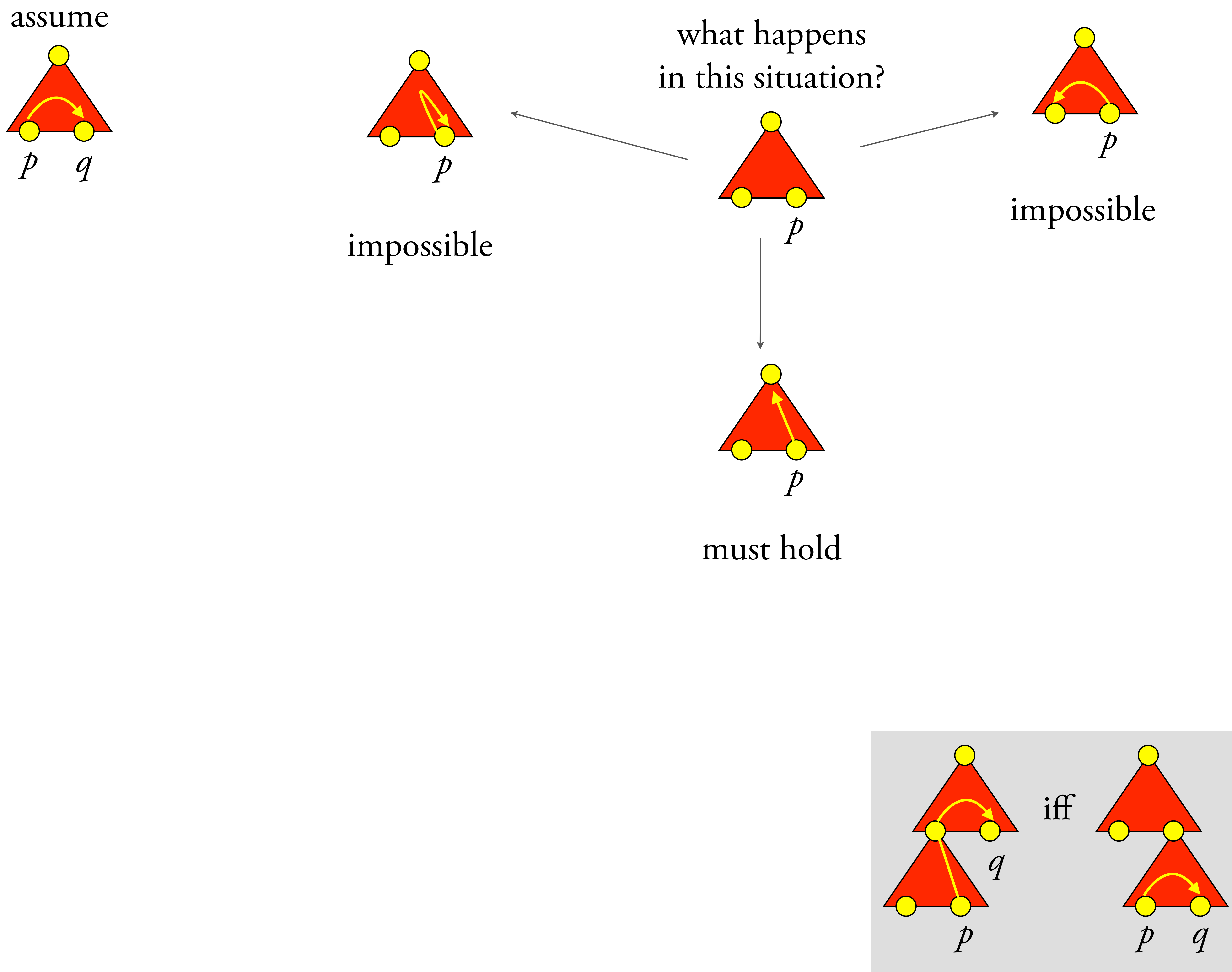
impossible

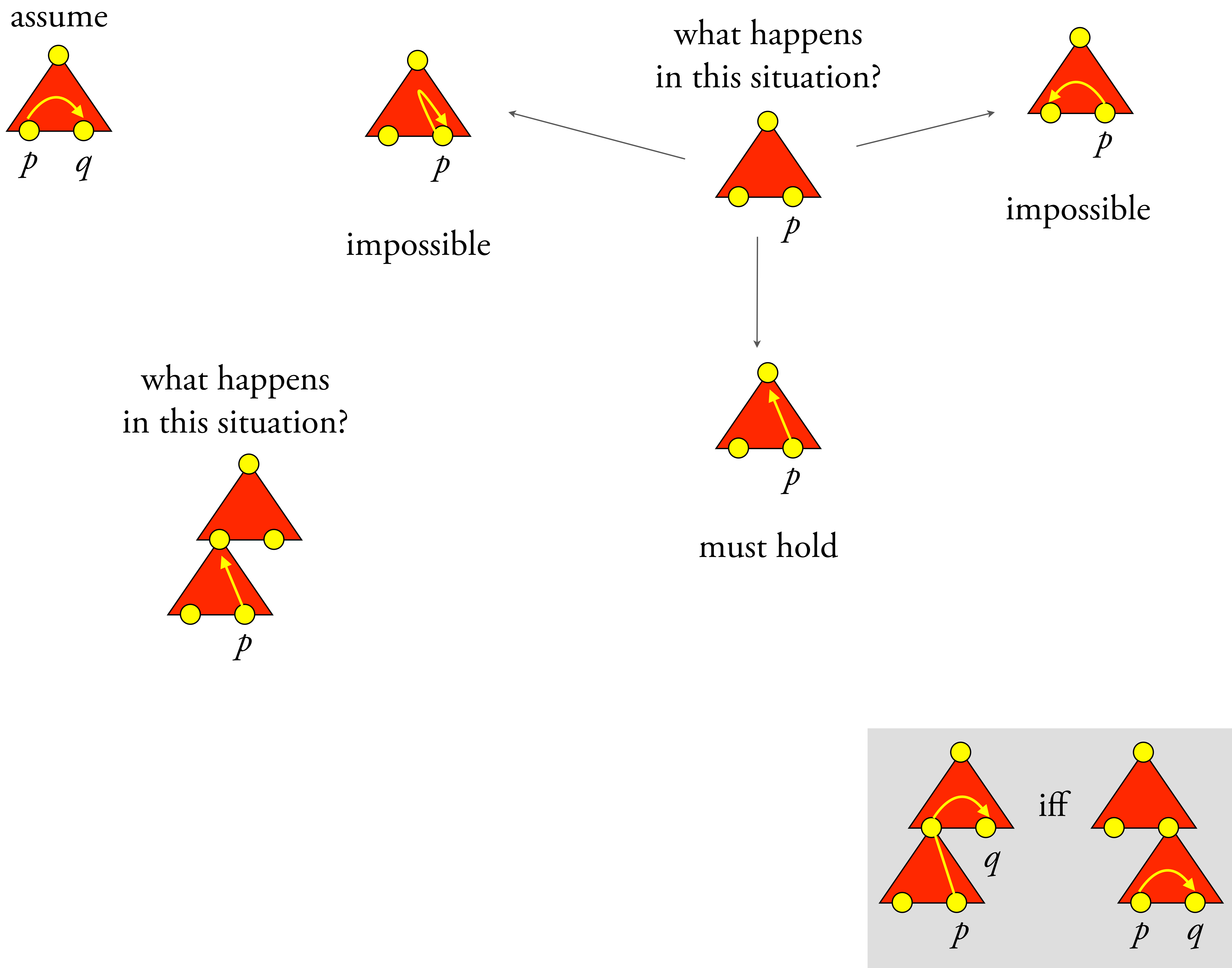


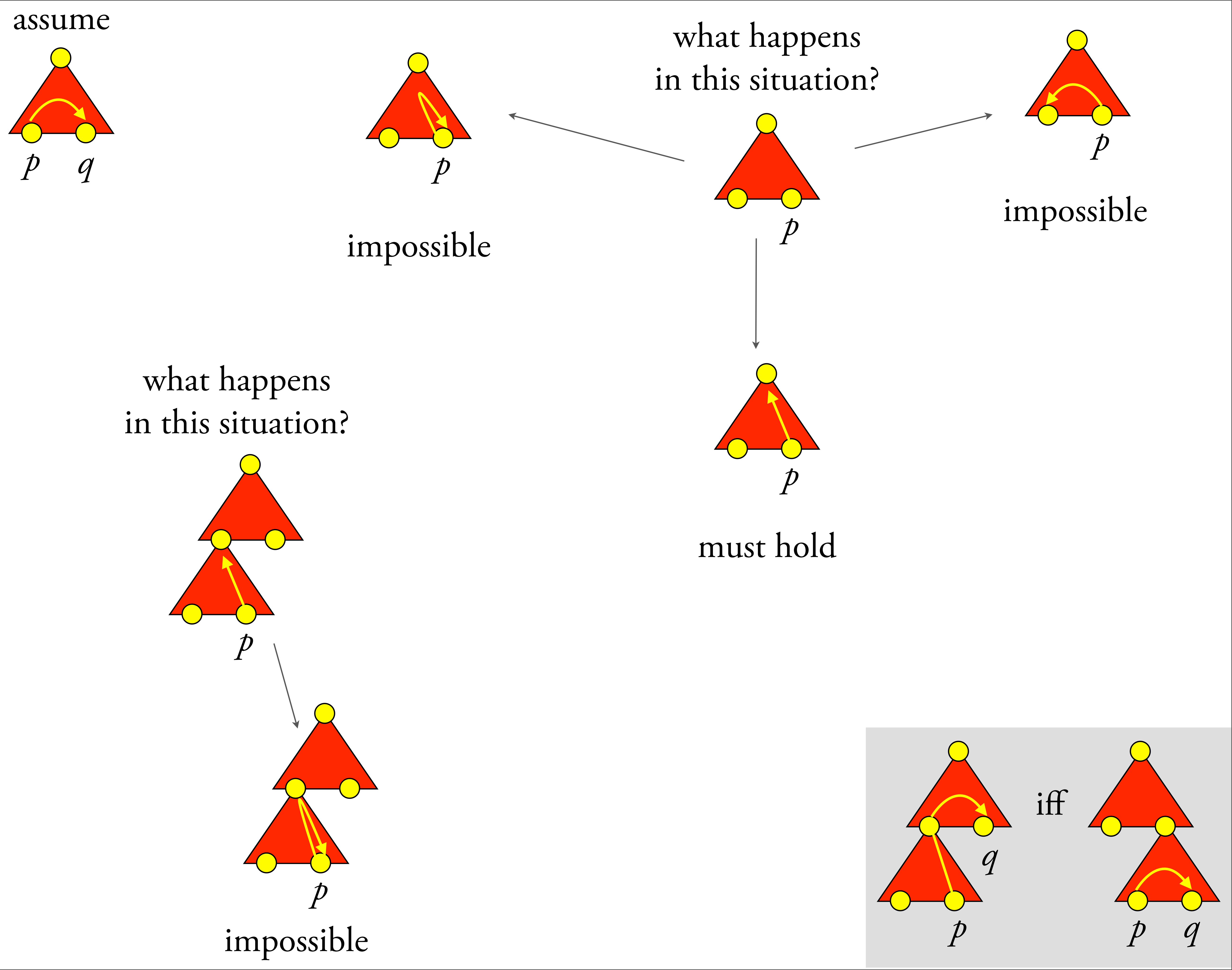




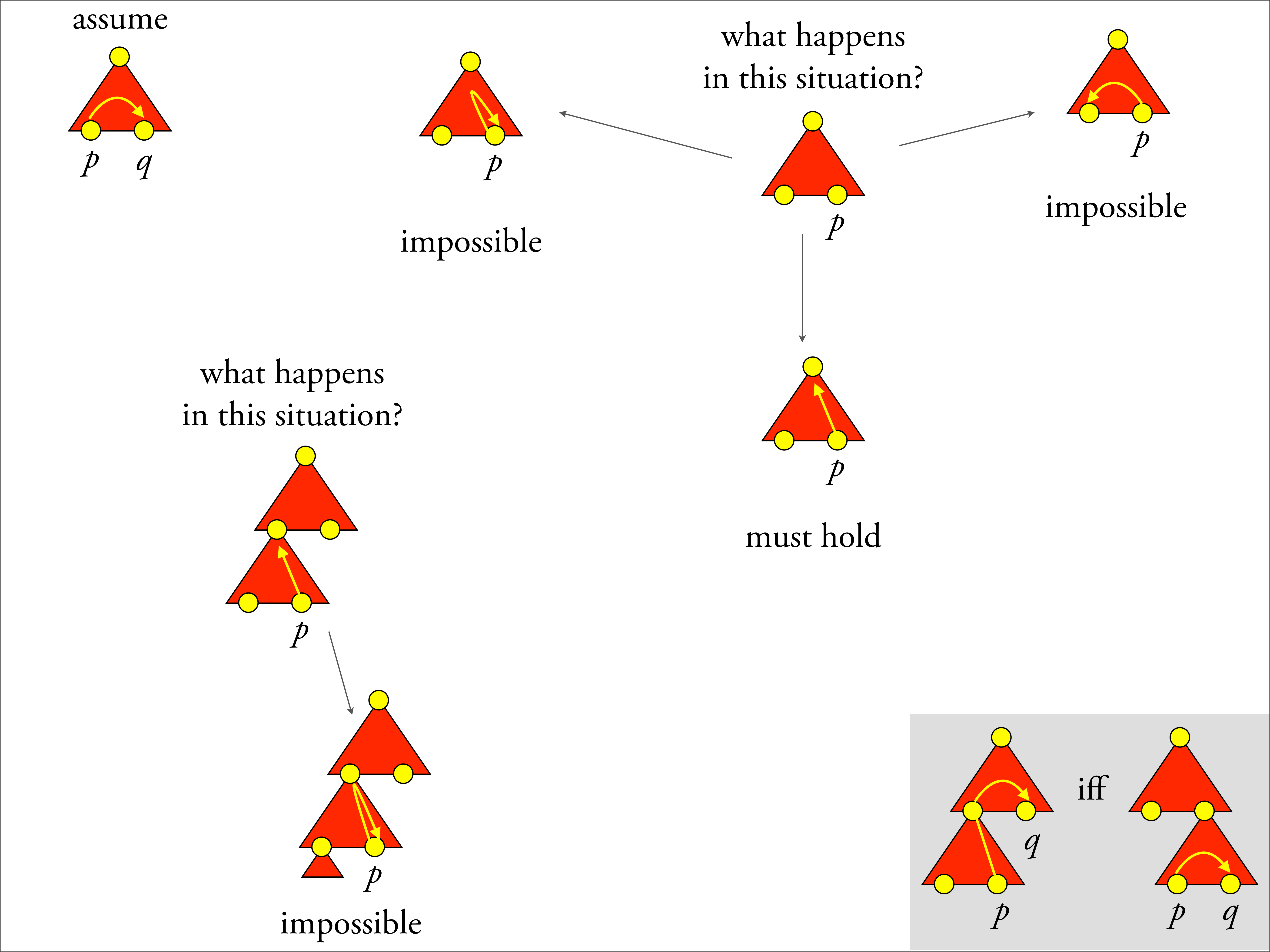
assume
p
q
impossible
p
what happens
in this situation?
p
p
impossible
iff
q
p
p
q


assume
p
q
what happens
in this situation?
p
impossible
p
impossible
p
p
must hold
iff
q
p
p
q

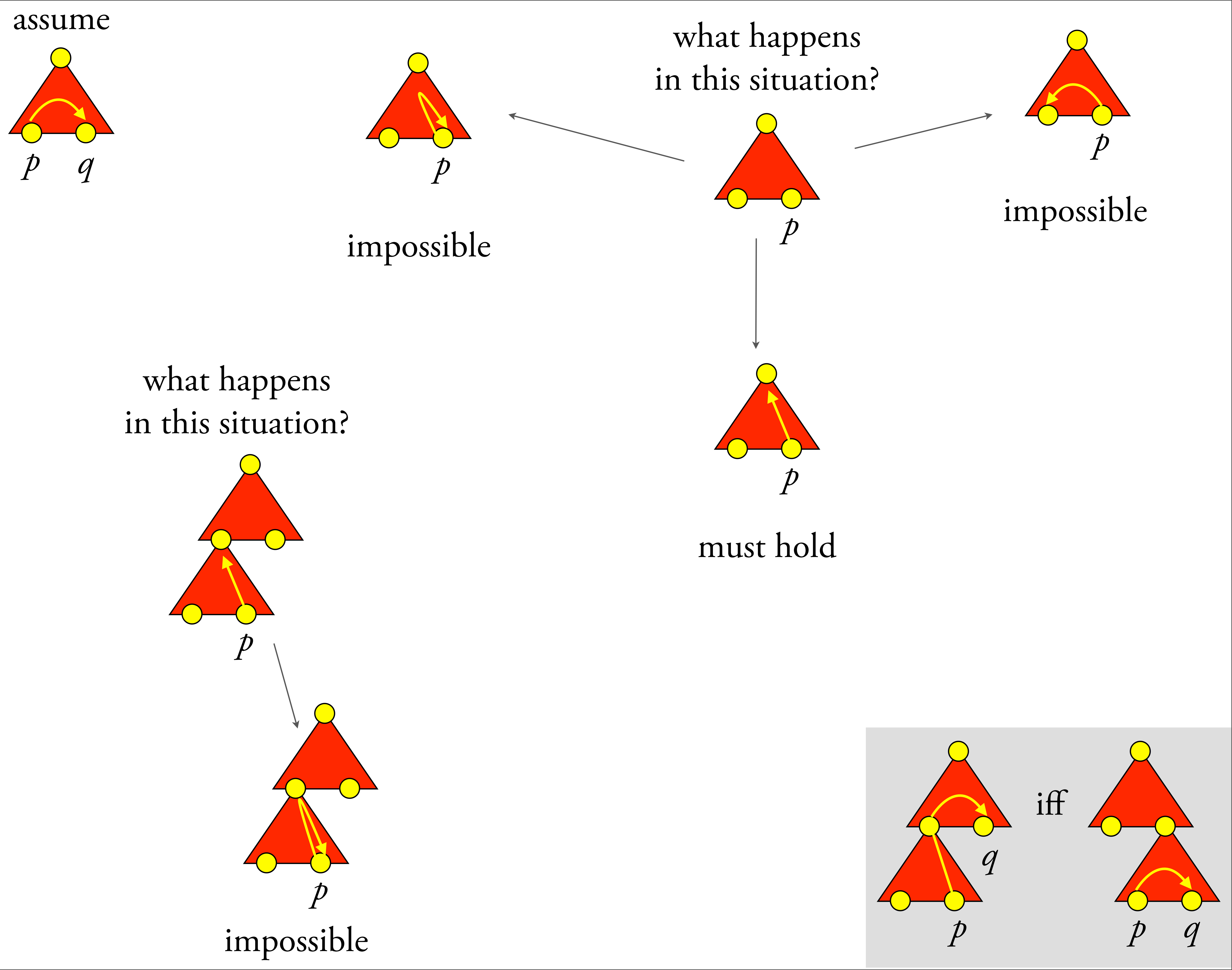

assume
p
q
impossible
p
what happens
in this situation?
p
impossible
p
must hold
p
what happens
in this situation?
p
iff
q
p
p
q

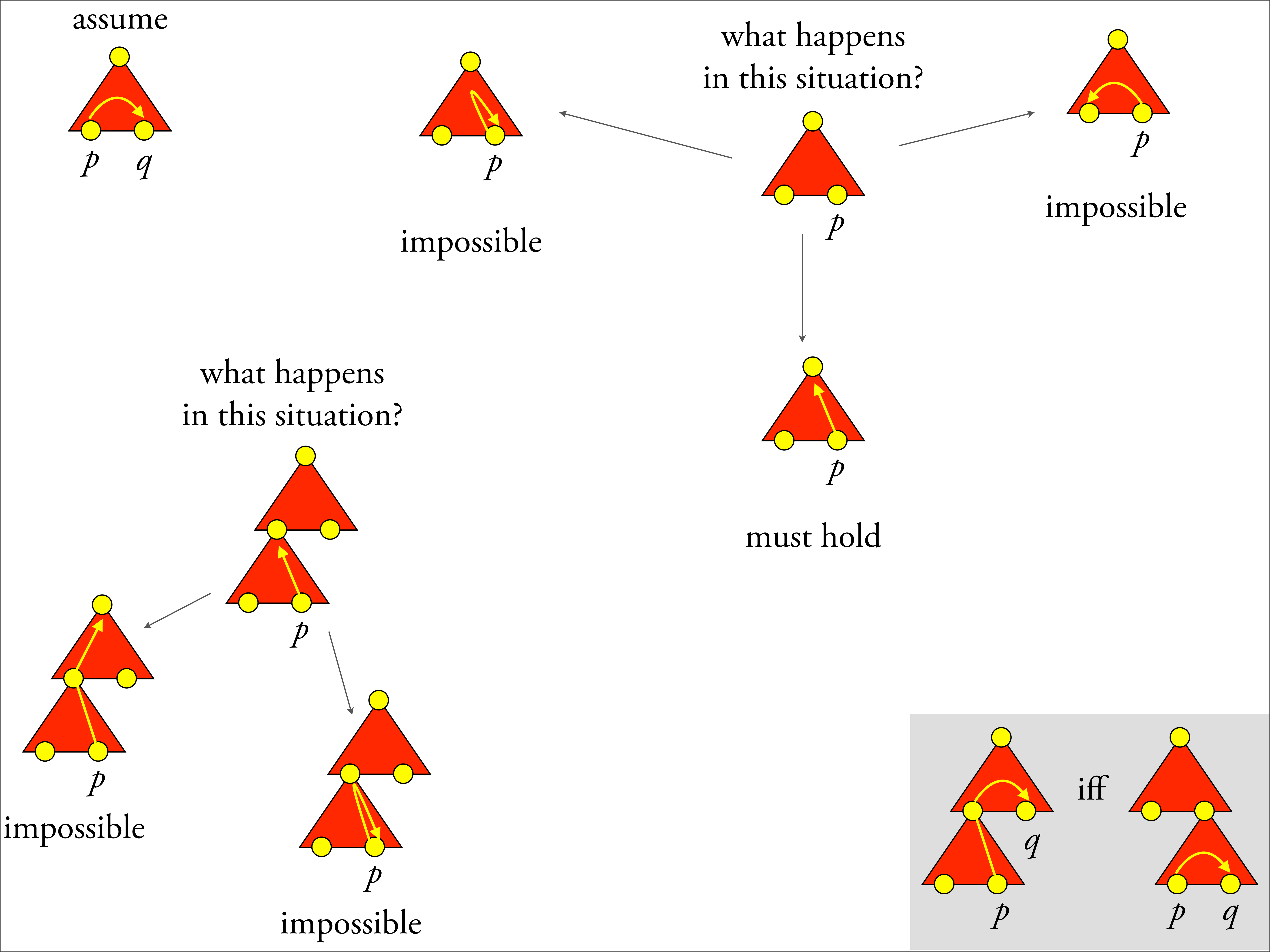


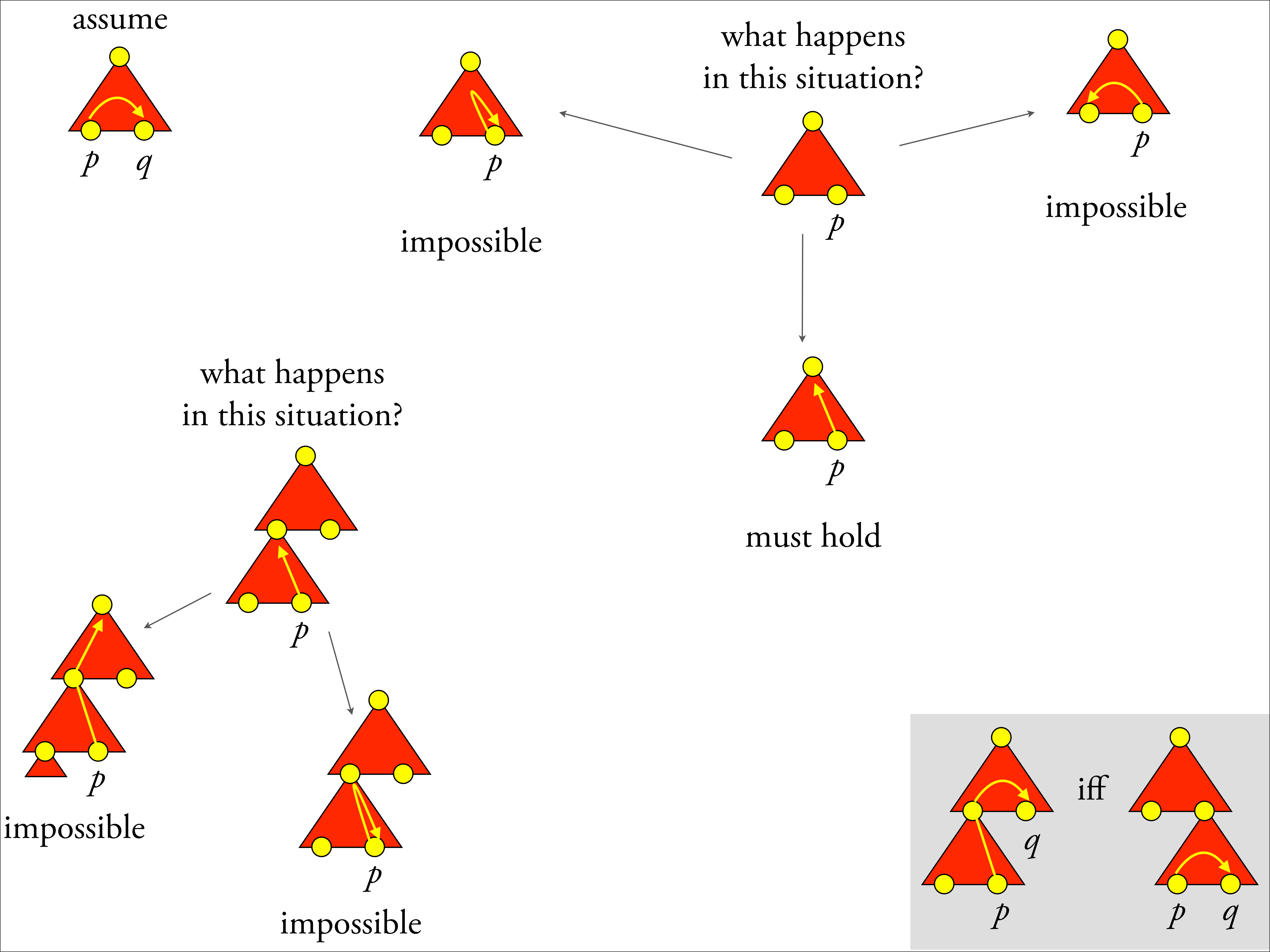
assume
p
q
p
impossible
what happens
in this situation?
p
p
impossible
p
must hold
what happens
in this situation?
p
p
impossible
iff
q
p
p
q

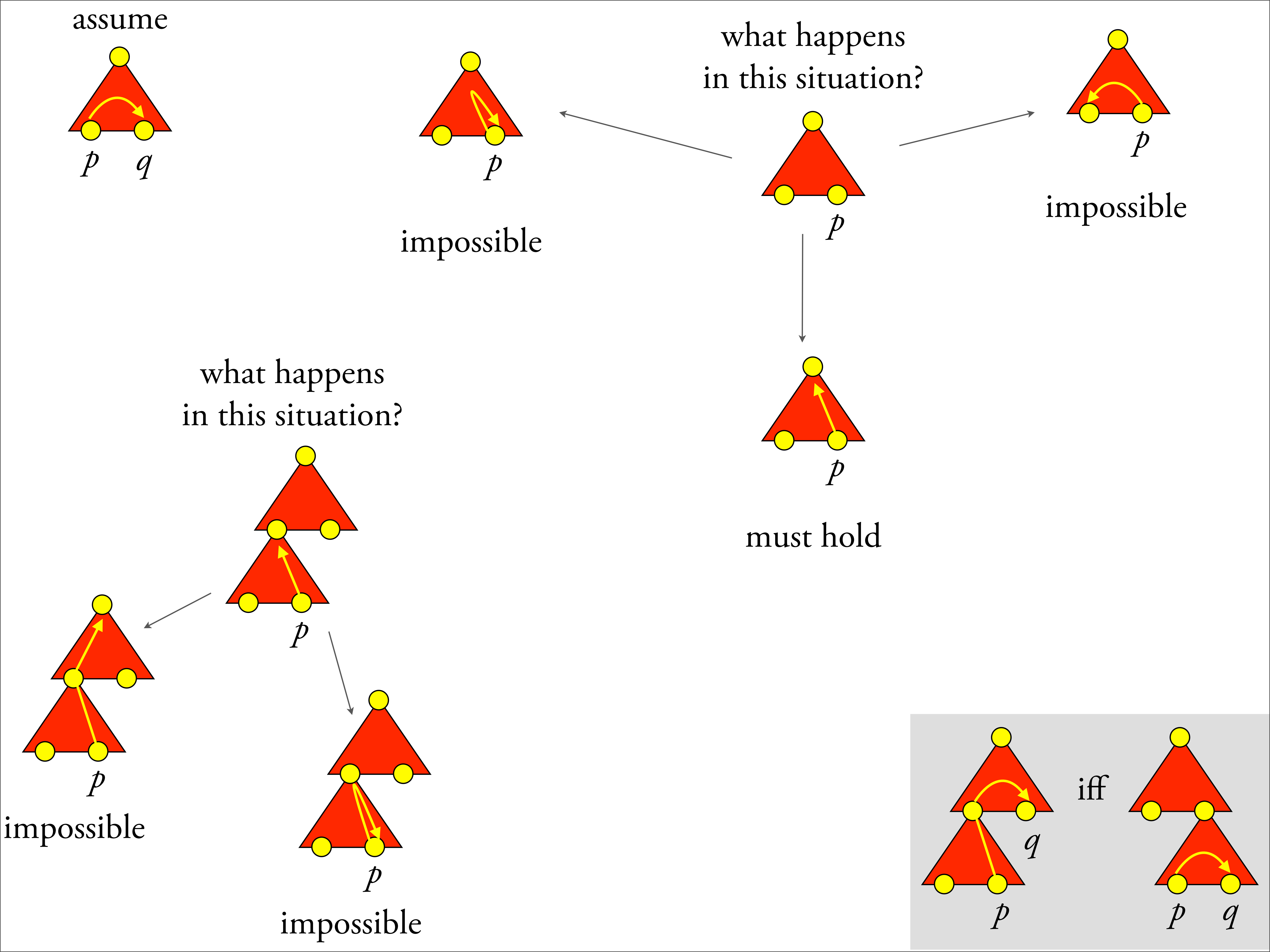


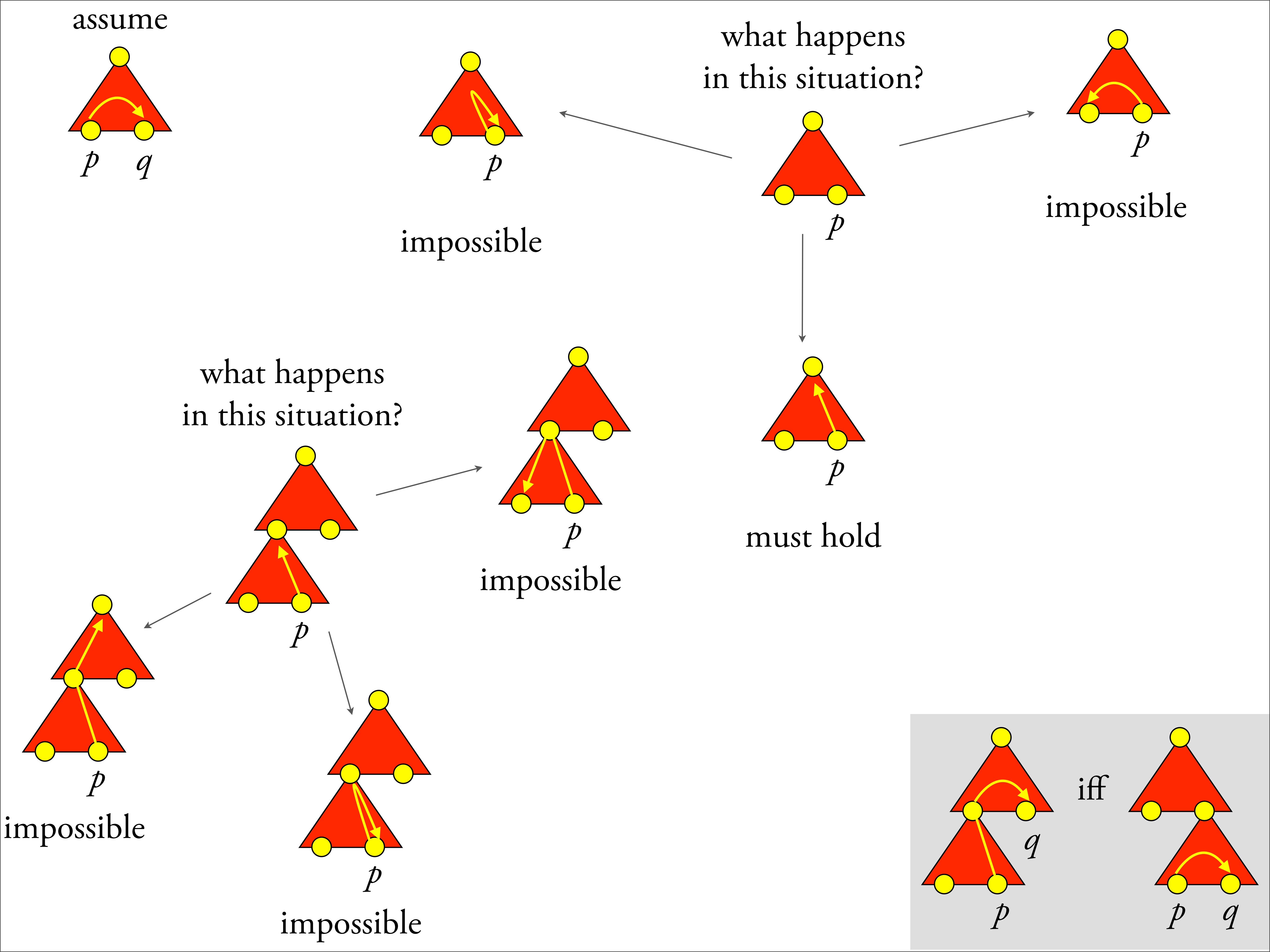
assume
p
q
impossible
p
what happens
in this situation?
p
impossible
p
must hold
p
what happens
in this situation?
p
impossible
p
iff
q
p
p
q

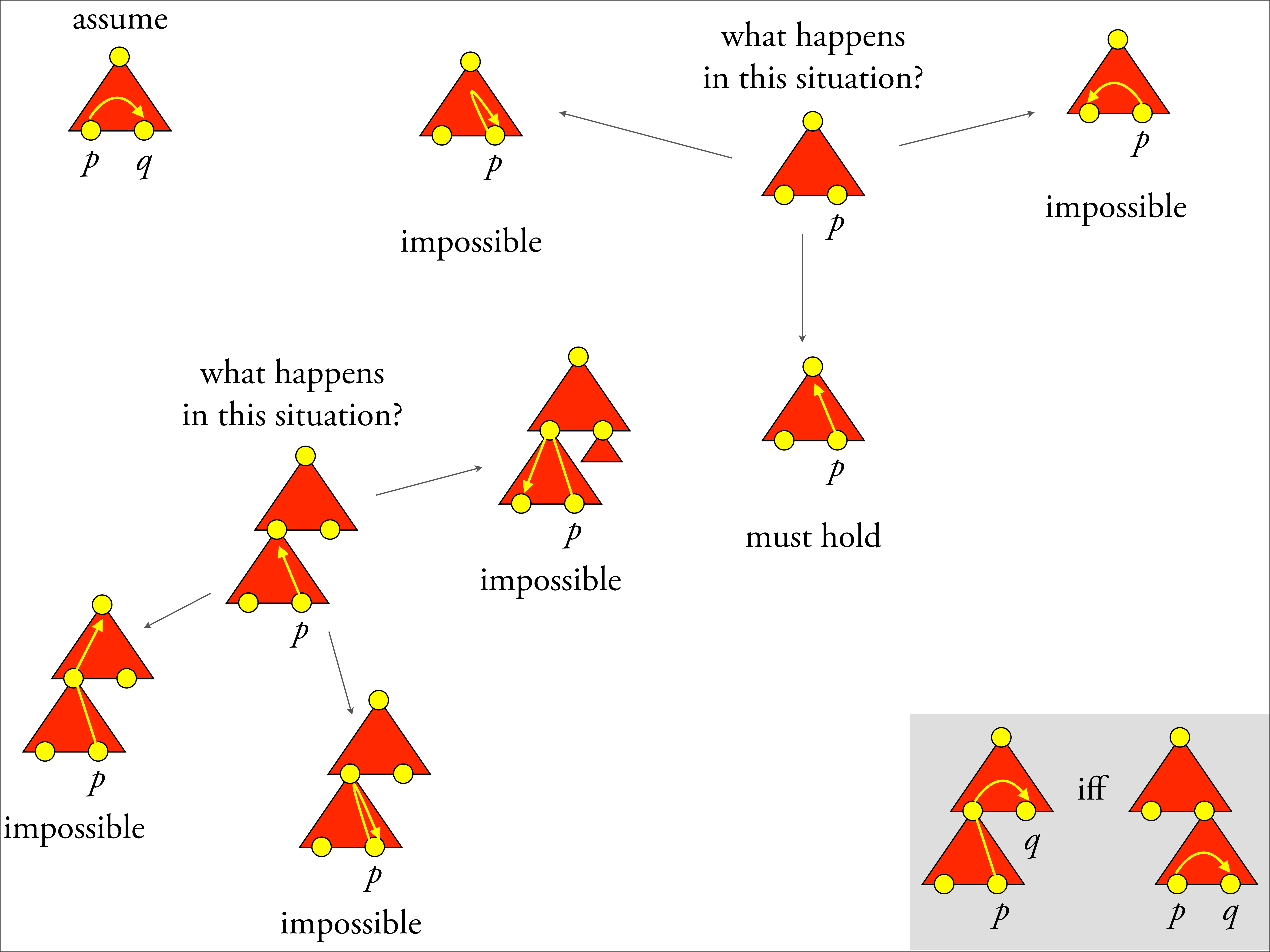

assume
p
q
impossible
p
what happens
in this situation?
p
must hold
p
impossible
p
what happens
in this situation?
p
impossible
p
iff
q
p
p
q

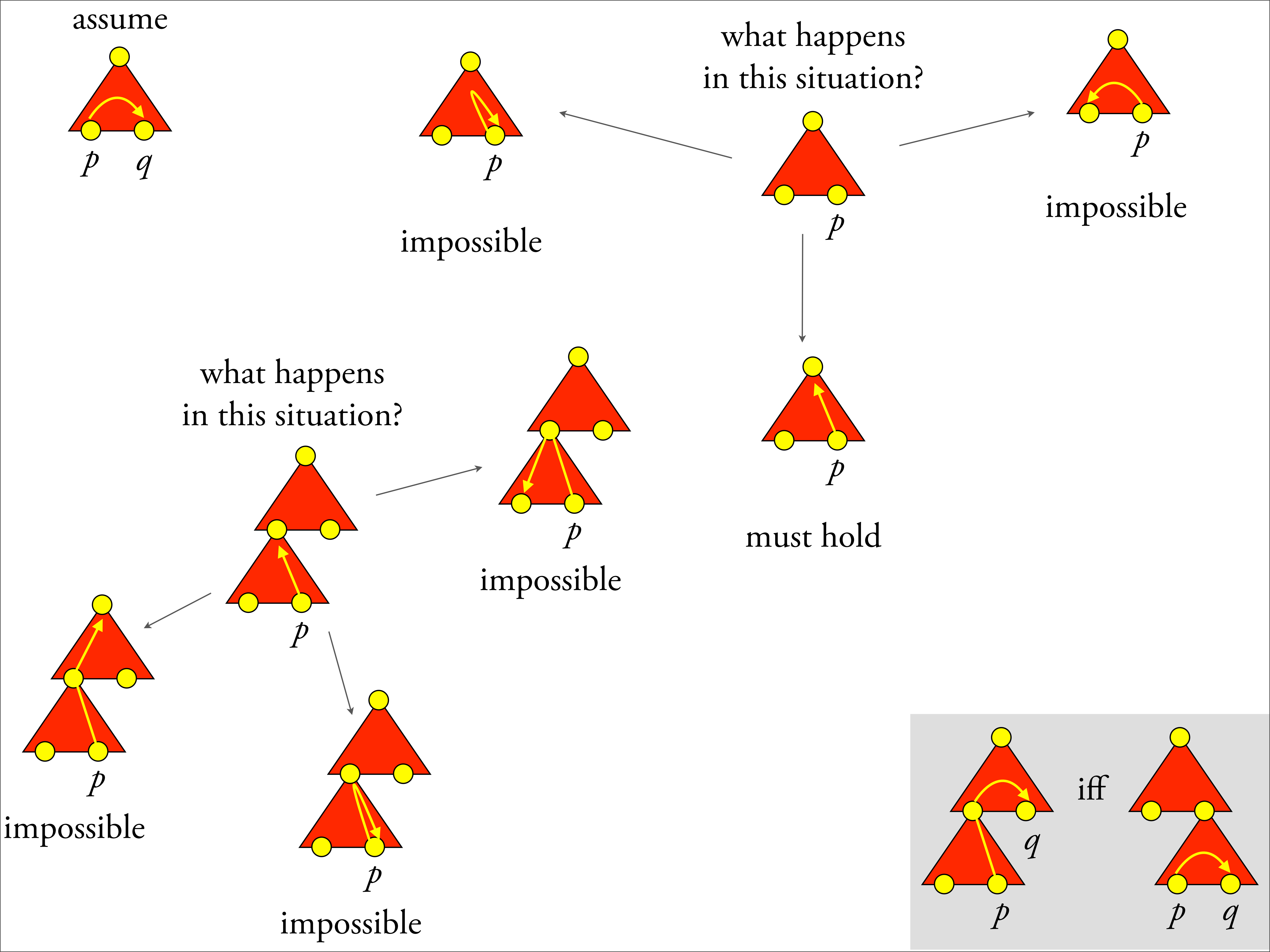

assume
p q
what happens
in this situation?
p
impossible
p
impossible
p
must hold
what happens
in this situation?
p
p
impossible
p
impossible
iff
q
p
p q

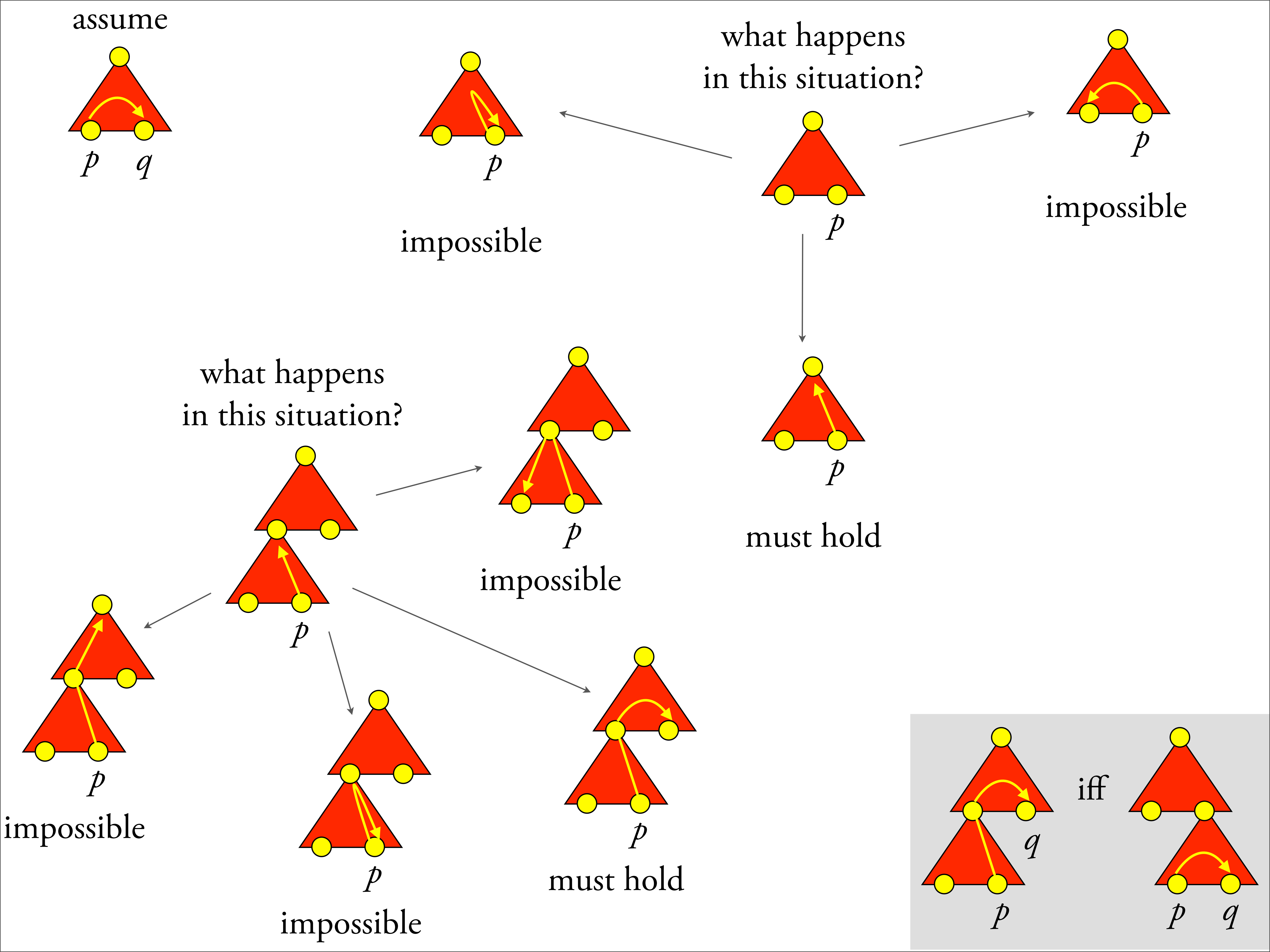

assume
p
q
p
impossible
what happens
in this situation?
p
p
impossible
p
must hold
what happens
in this situation?
p
p
impossible
p
impossible
iff
q
p
p
q

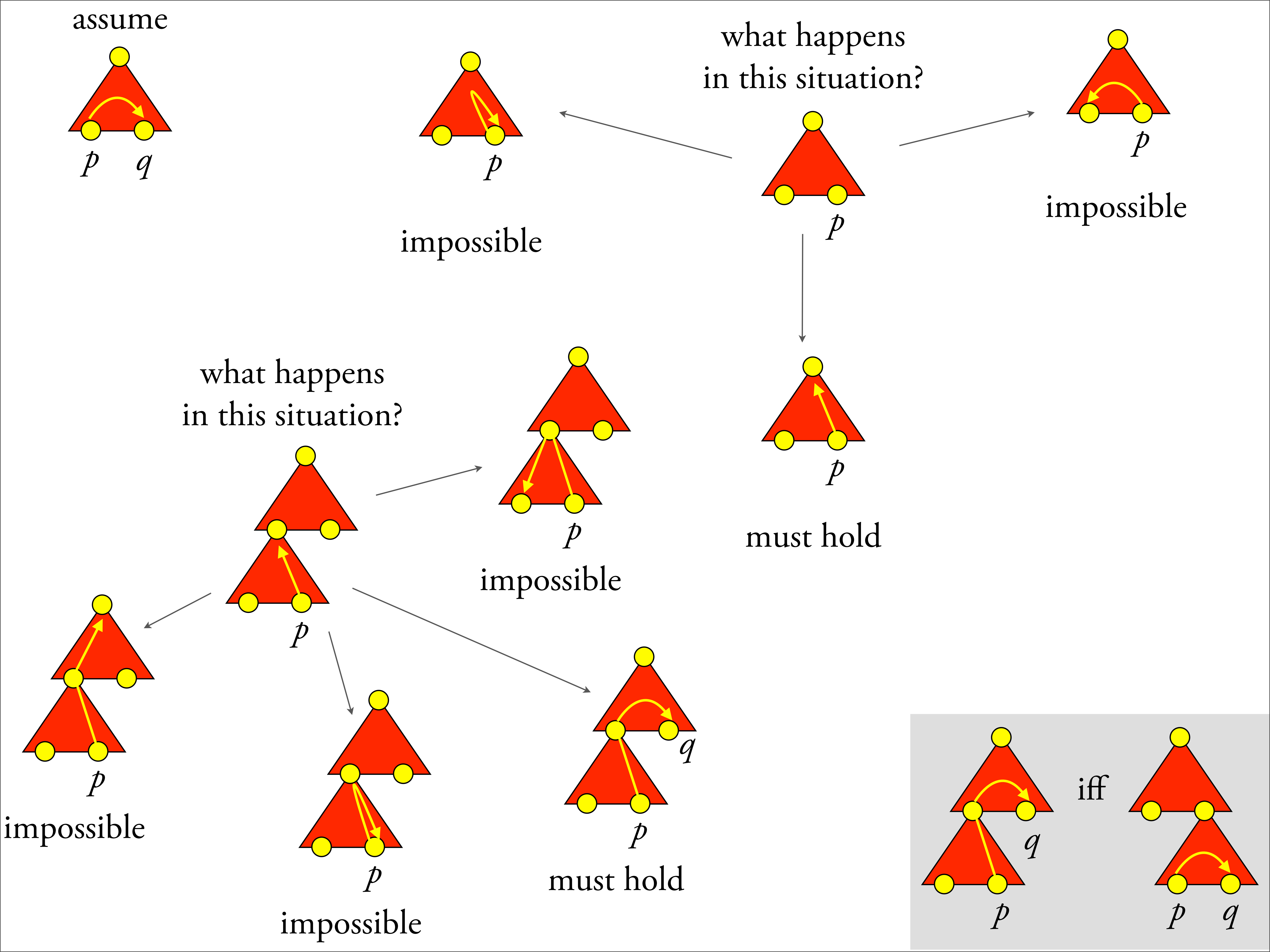

assume
p
q
p
impossible
what happens
in this situation?
p
impossible
p
must hold
what happens
in this situation?
p
p
impossible
p
impossible
q
p
iff
p
q


assume
p
q
impossible
p
what happens
in this situation?
p
impossible
p
must hold
p
what happens
in this situation?
p
impossible
p
impossible
p
impossible
p
q
iff
p
q


assume
p q
p
impossible
what happens
in this situation?
p
p
impossible
p
must hold
what happens
in this situation?
p
p
impossible
p
impossible
p
impossible
q
p
iff
p q


assume
p
q
impossible
p
what happens
in this situation?
p
impossible
p
must hold
p
what happens
in this situation?
p
impossible
p
impossible
p
impossible
p
iff
q
p
p
q


assume
p
q
impossible
what happens
in this situation?
p
p
impossible
p
must hold
what happens
in this situation?
p
impossible
p
impossible
p
impossible
p
must hold
q
iff
p
p
q


assume
p
q
impossible
p
what happens
in this situation?
p
p
impossible
must hold
p
what happens
in this situation?
p
p
impossible
p
impossible
p
impossible
q
p
must hold
iff
q
p
p
q