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

Some logics that describe tree properties

Some logics that describe tree properties

monadic second-order logic

"There is a set of nodes that is closed under parents, has an *a* label, and has no *c* label"

$$\exists X \quad \wedge \begin{cases} \exists x \in X \;\; a(x) \\ \forall x \in X \;\; \forall y \quad \text{parent}(x,y) \;\Rightarrow\; y \in X \\ \forall x \in X \;\; \neg c(x) \end{cases}$$

Some logics that describe tree properties

monadic second-order logic

"There is a set of nodes that is closed under parents, has an *a* label, and has no *c* label"

$$\exists X \ \wedge \begin{cases} \exists x{\in}X \ \ a(x) \\ \forall x{\in}X \ \forall y \quad \text{parent}(x,y) \ \Rightarrow \ y{\in}X \\ \forall x{\in}X \ \ \neg c(x) \end{cases}$$

first-order logic

"There is a node with label *a* that has only *b*-labeled ancestors"

$$\exists x \ \ a(x) \quad \wedge \quad (\forall y < x \quad b(y)\,)$$

Some logics that describe tree properties

monadic second-order logic

"There is a set of nodes that is closed under parents, has an *a* label, and has no *c* label"

$$\exists X \ \wedge \begin{cases} \exists x{\in}X \ \ a(x) \\ \forall x{\in}X \ \forall y \ \ \text{parent}(x,y) \Rightarrow \ y{\in}X \\ \forall x{\in}X \ \ \neg c(x) \end{cases}$$

first-order logic

"There is a node with label *a* that has only *b*-labeled ancestors"

$$\exists x \ \ a(x) \quad \wedge \quad (\forall y < x \quad b(y))$$

first-order logic with transitive closure

Instead of $<$ we can write

$$(\text{parent}(x,y))^*$$

Some logics that describe tree properties

monadic second-order logic

"There is a set of nodes that is closed under parents, has an *a* label, and has no *c* label"

$$\exists X \ \wedge \begin{cases} \exists x {\in} X \ \ a(x) \\ \forall x {\in} X \ \forall y \ \ \text{parent}(x,y) \Rightarrow \ y {\in} X \\ \forall x {\in} X \ \ \neg c(x) \end{cases}$$

temporal logics

"On some path, *b* holds until *a* holds"

$\mathbf{E}\, b\, \mathbf{U}\, a$

first-order logic

"There is a node with label *a* that has only *b*-labeled ancestors"

$$\exists x \ \ a(x) \quad \wedge \quad (\forall y < x \quad b(y)\,)$$

first-order logic with transitive closure

Instead of < we can write

$(\text{parent}(x,y))^*$

Some logics that describe tree properties

monadic second-order logic

"There is a set of nodes that is closed under parents, has an *a* label, and has no *c* label"

$$\exists X \ \wedge\begin{cases} \exists x{\in}X \ \ a(x) \\ \forall x{\in}X \ \forall y \ \ \text{parent}(x,y) \Rightarrow y{\in}X \\ \forall x{\in}X \ \ \neg c(x) \end{cases}$$

temporal logics

"On some path, *b* holds until *a* holds"

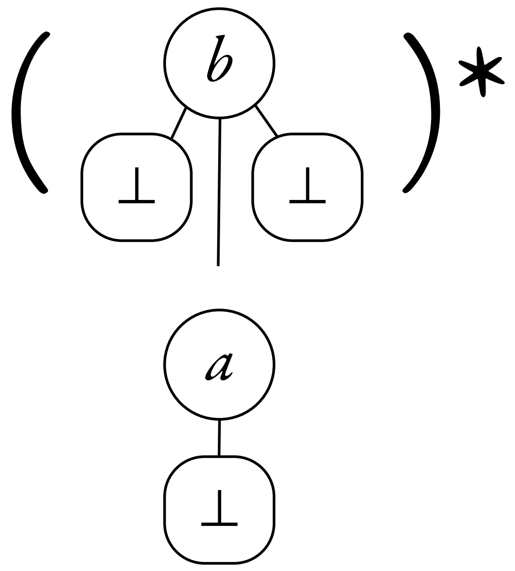
$\mathbf{E}\, b\, \mathbf{U}\, a$

first-order logic

"There is a node with label *a* that has only *b*-labeled ancestors"

$$\exists x \ \ a(x) \quad \wedge \quad (\forall y < x \ \ b(y))$$

regular expressions



first-order logic with transitive closure

Instead of < we can write

$(\text{parent}(x,y))^*$

Some logics that describe tree properties

monadic second-order logic

"There is a set of nodes that is closed under parents, has an a label, and has no c label"

$$\exists X \ \wedge \begin{cases} \exists x \in X \ \ a(x) \\ \forall x \in X \ \forall y \quad \text{parent}(x,y) \Rightarrow \ y \in X \\ \forall x \in X \ \ \neg c(x) \end{cases}$$

temporal logics

"On some path, b holds until a holds"

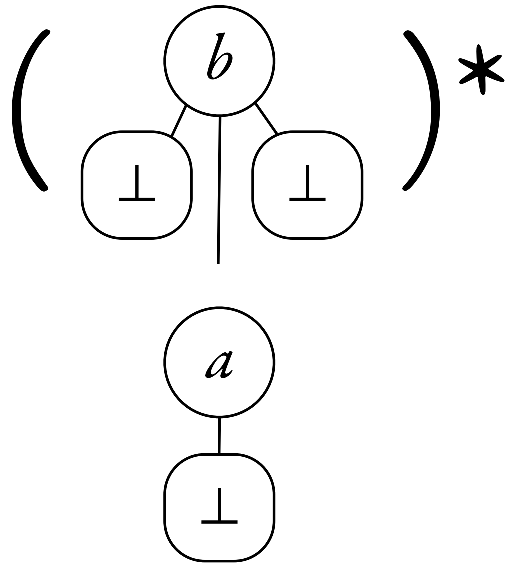
$\mathbf{E}\, b\, \mathbf{U}\, a$

first-order logic

"There is a node with label a that has only b-labeled ancestors"

$$\exists x \ \ a(x) \quad \wedge \quad (\forall y < x \quad b(y)\,)$$

regular expressions



first-order logic with transitive closure

Instead of $<$ we can write

$(\text{parent}(x,y))^*$

Monadic- and First-Order Logic for Words

definition

weakness of first-order logic

MSO=regular

Monadic- and First-Order Logic for Trees

Transitive Closure Logic and Regular Expressions

Monadic Second-Order Logic

grandfather of logics for regular languages

Monadic Second-Order Logic

grandfather of logics for regular languages

Thm. (Thatcher, Wright `68)
A tree language is regular if and only if it can be defined in monadic second-order logic.

Monadic Second-Order Logic
grandfather of logics for regular languages

words: Büchi, Trakhtenbrot, Elgot (`60, `61)
infinite words: Büchi `62

Thm. (Thatcher, Wright `68)
A tree language is regular if and only if it can be defined in monadic second-order logic.

Monadic Second-Order Logic

grandfather of logics for regular languages

words: Büchi, Trakhtenbrot, Elgot (`60, `61)
infinite words: Büchi `62

Thm. (Thatcher, Wright `68)
A tree language is regular if and only if it can be defined in monadic second-order logic.

infnite trees: Rabin `69

Monadic Second-Order Logic

grandfather of logics for regular languages

words: Büchi, Trakhtenbrot, Elgot (`60, `61)
infinite words: Büchi `62

Thm. (Thatcher, Wright `68)
A tree language is regular if and only if it can be defined in monadic second-order logic.

infnite trees: Rabin `69

Regular tree languages are closed under:

– union
– intersection
– complementation
– projection *f(L),* with *f* letter-to-letter

Monadic Second-Order Logic
grandfather of logics for regular languages

words: Büchi, Trakhtenbrot, Elgot (`60, `61)
infinite words: Büchi `62

Thm. (Thatcher, Wright `68)
A tree language is regular if and only if it can be defined in monadic second-order logic.

infnite trees: Rabin `69

Regular tree languages are closed under:

$\vee$	– union
$\wedge$	– intersection
$\neg$	– complementation
$\exists$	– projection *f(L)*, with *f* letter-to-letter

First-Order Logic for Words

Alphabet: $A=\{a,b,c\}$ $A^*ab^*aA^*$

first-order logic

$$\exists x\ \exists y\quad a(x) \land a(y) \land x<y \land (\forall z\quad x<z<y \Rightarrow b(x))$$

quantification
is over positions

label predicates

order on positions

Formal definition: a word $w{=}a_1\,a_2 \cdots a_n$ word is interpreted as
structure $\underline{w}= \langle \{1,..,n\}, <, a(x), b(x), c(x) \rangle$
A formula Ψ gives a language $L_\Psi=\{w : \Psi \text{ holds in } \underline{w}\}$

Thm. Every language definable in first-order logic is regular, but not conversely, eg. $(aa)^*$.

Ehrenfeucht-Fraïssé Game

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

Ehrenfeucht-Fraïssé Game

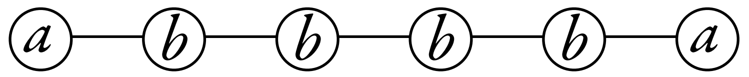
Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

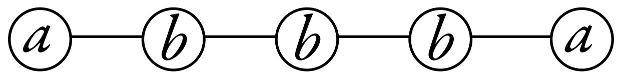
The game is played on two structures, in k rounds.

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.

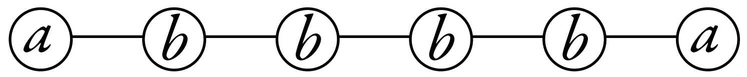


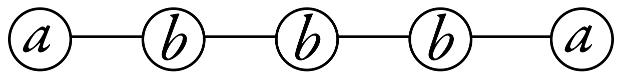


Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.





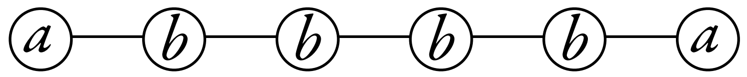
one round:

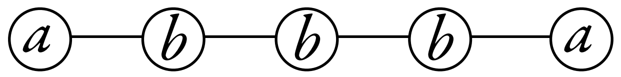
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.





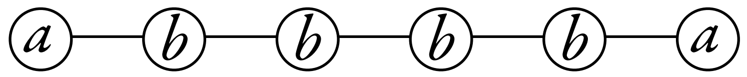
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

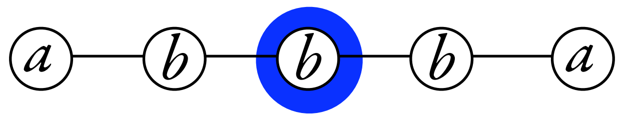
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.





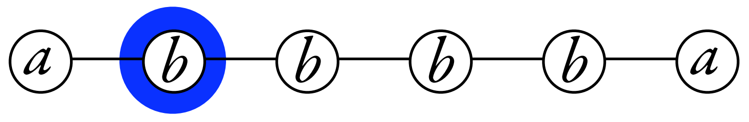
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

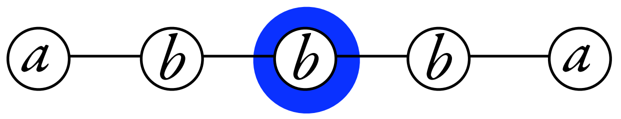
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.





one round:

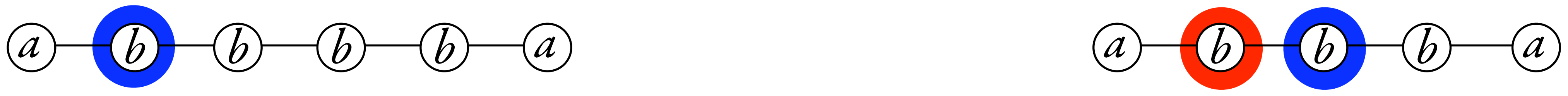
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.



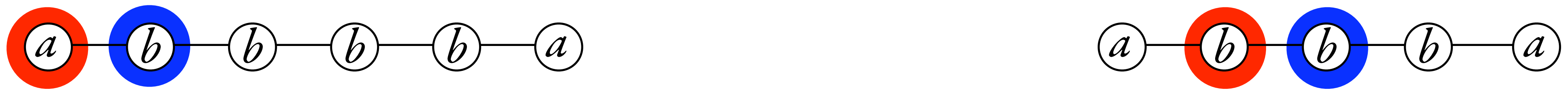
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.



one round:

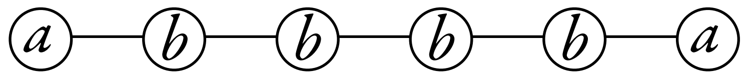
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

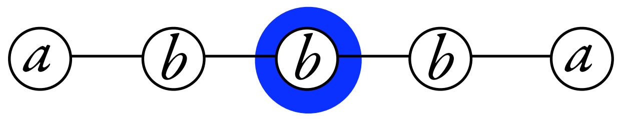
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.





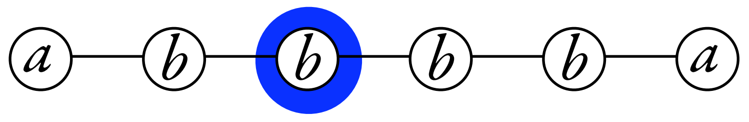
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

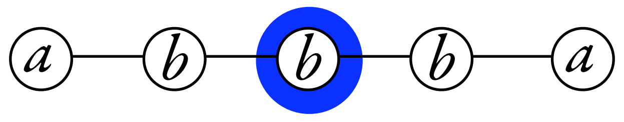
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.





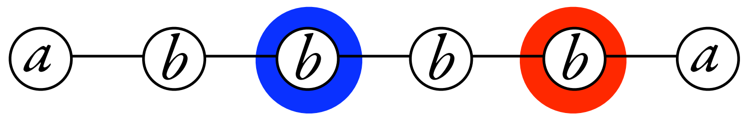
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

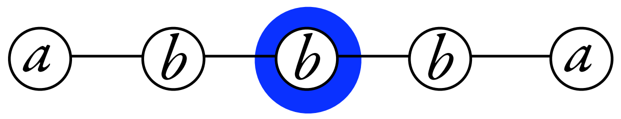
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.





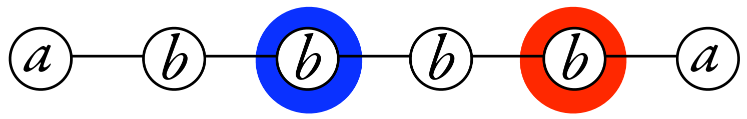
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

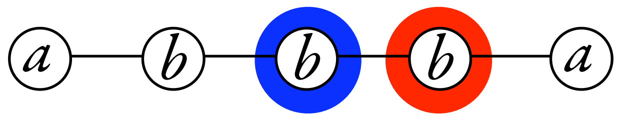
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.





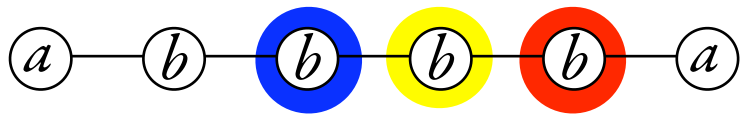
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

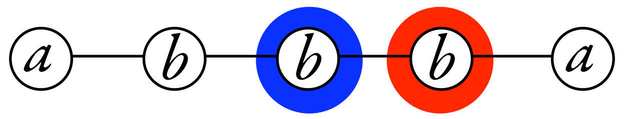
If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.





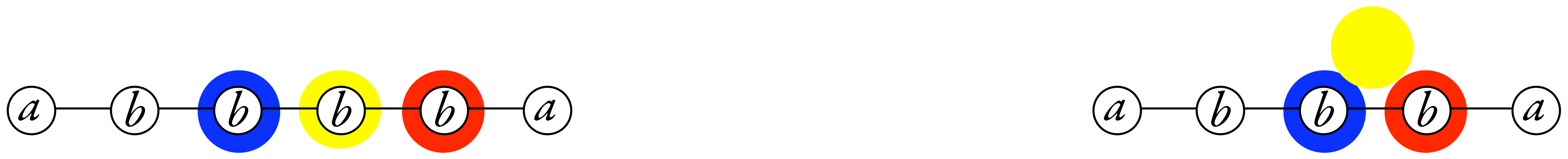
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.



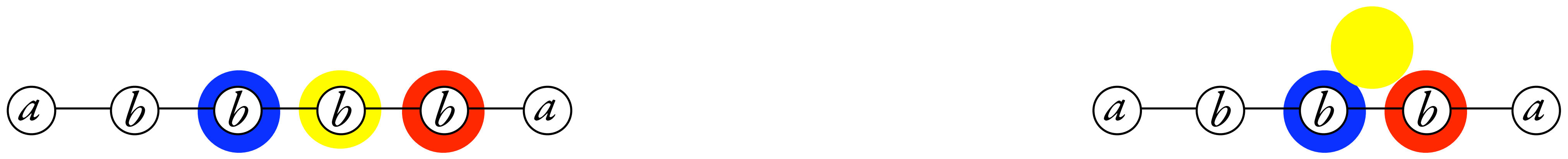
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth *k* if and only if Duplicator can survive the *k*-round game against Spoiler.

The game is played on two structures, in *k* rounds.



on these two structures, Duplicator can survive 2 rounds, but not 3.

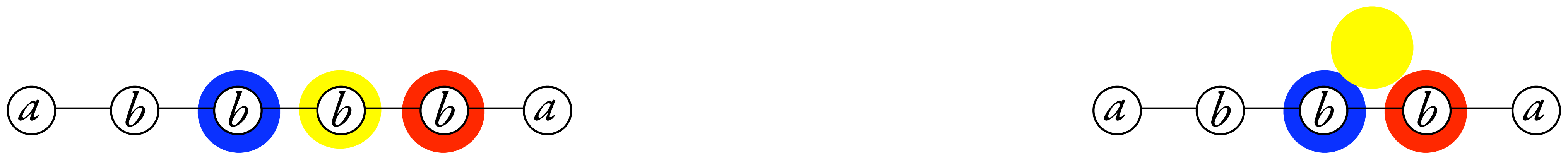
one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Ehrenfeucht-Fraïssé Game

Thm. Two structures satisfy the same sentences of quantifier depth k if and only if Duplicator can survive the k-round game against Spoiler.

The game is played on two structures, in k rounds.



on these two structures, Duplicator can survive 2 rounds, but not 3.

“exists a b-node that separates every two other b-nodes”

one round:
Spoiler creates a new color, and places a pebble of this color on some node of one of the structures.
Duplicator responds by placing a pebble of the same color on some node of the other structure.

If the pebble configurations are not the same, Duplicator dies at the end of the round.
(for words, same = same order on colors, same labels for same colors)

Fact. The language $(aa)^*$ cannot be defined in first-order logic (with order $<$ and labels).

Fact. The language $(aa)^*$ cannot be defined in first-order logic (with order < and labels).

Proof. For any number of rounds k, Duplicator has a strategy to survive the game played on words of length 2^k and 2^k+1

Fact. The language $(aa)^*$ cannot be defined in first-order logic (with order < and labels).

Proof. For any number of rounds *k*, Duplicator has a strategy to survive the game played on words of length 2^k and 2^k+1

Strategy: preserve the following invariant, when *i* rounds are left.

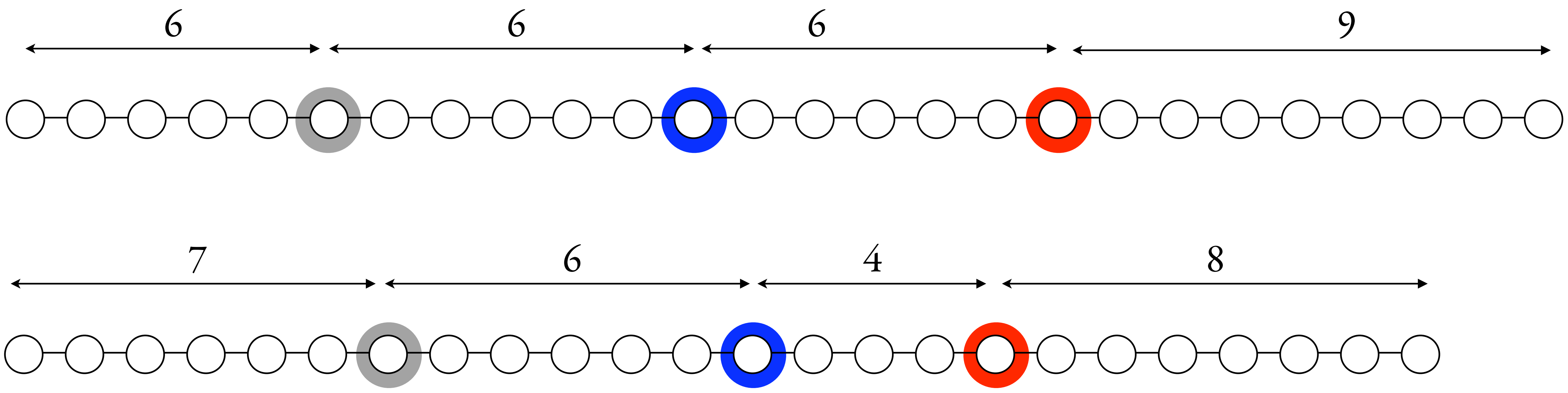
Pebbles are ordered the same way, and the distances between consecutive pebbles are either equal, or at least 2^i.

Fact. The language $(aa)^*$ cannot be defined in first-order logic (with order $<$ and labels).

Proof. For any number of rounds k, Duplicator has a strategy to survive the game played on words of length 2^k and 2^k+1

Strategy: preserve the following invariant, when i rounds are left.

Pebbles are ordered the same way, and the distances between consecutive pebbles are either equal, or at least 2^i.

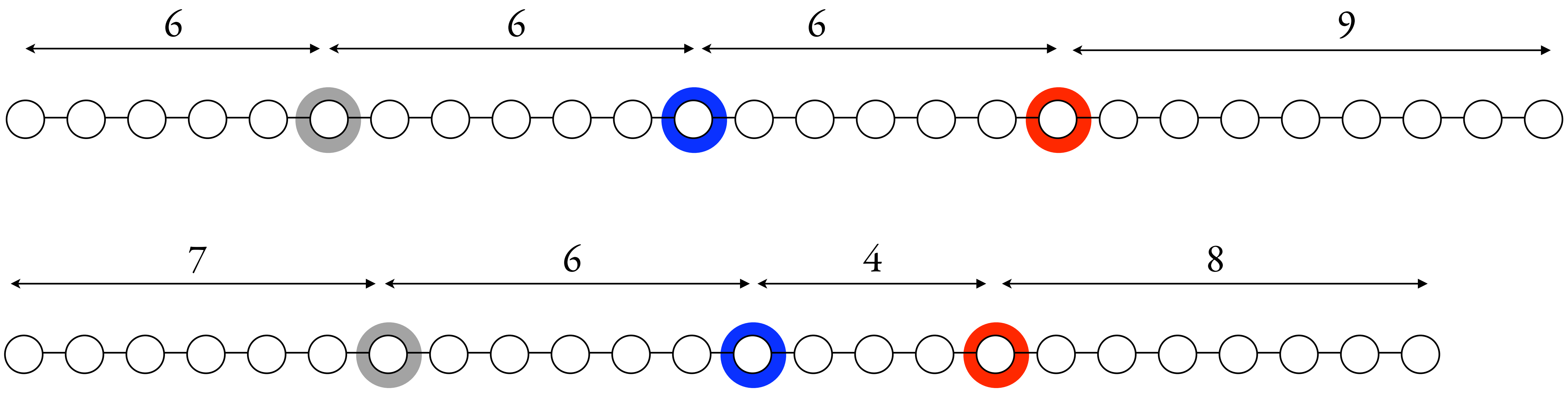


Fact. The language $(aa)^*$ cannot be defined in first-order logic (with order $<$ and labels).

Proof. For any number of rounds k, Duplicator has a strategy to survive the game played on words of length 2^k and 2^k+1

Strategy: preserve the following invariant, when i rounds are left.

Pebbles are ordered the same way, and the distances between consecutive pebbles are either equal, or at least 2^i.



if there are $i=2$ rounds left, Duplicator will survive.

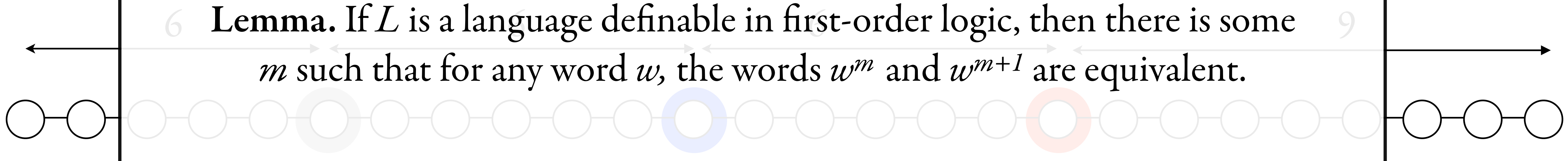
Fact. The language $(aa)^*$ cannot be defined in first-order logic (with order $<$ and labels).

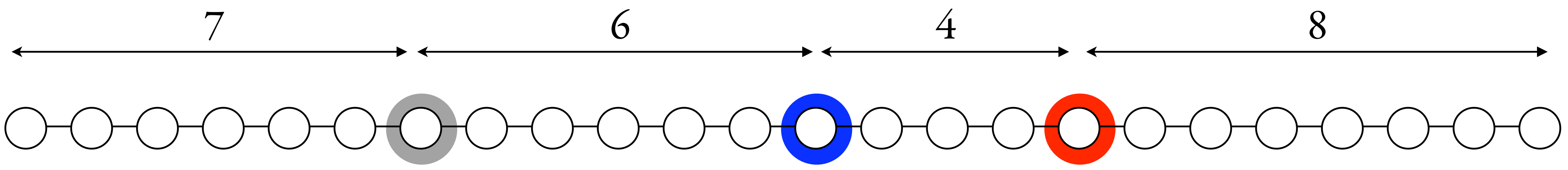
Proof. For any number of rounds k, Duplicator has a strategy to survive the game played on words of length 2^k and 2^k+1

Strategy: preserve the following invariant, when i rounds are left.

Pebbles are ordered the same way, and the distances between consecutive pebbles are either equal, or at least 2^i.

Lemma. If L is a language definable in first-order logic, then there is some m such that for any word w, the words w^m and w^{m+1} are equivalent.





if there are $i=2$ rounds left, Duplicator will survive.

Monadic Second-Order Logic for Words

A word belongs to $(aa)^*$ iff its satisfies the following formula:

Monadic Second-Order Logic for Words

A word belongs to $(aa)^*$ iff its satisfies the following formula:

there is a set of positions

$\exists X$

that contains every second position,

$$\forall x \forall y \ suc(x,y) \Rightarrow \big(x \in X \iff y \notin X\big)$$

and contains the first position,

$$\forall x \exists y \ \ y \geq x \ \wedge \ y \notin X$$

but does not contain the last position

$$\forall x \exists y \ \ y \leq x \ \wedge \ y \in X$$

Monadic Second-Order Logic for Words

A word belongs to $(aa)^*$ iff its satisfies the following formula:

there is a set of positions

$\exists X$

$\Bigg\{$

that contains every second position,

$$\forall x \forall y\ suc(x,y) \ \Rightarrow\ \big(x \in X \iff y \notin X\big)$$

and contains the first position,

$$\forall x \exists y \ \ y \geq x \ \wedge\ y \notin X$$

but does not contain the last position

$$\forall x \exists y \ \ y \leq x \ \wedge\ y \in X$$

MSO is the extension of first-order logic with set quantification.

Monadic Second-Order Logic for Words

A word belongs to $(aa)^*$ iff its satisfies the following formula:

there is a set of positions

$\exists X$

$\Bigg\{$

that contains every second position,

$$\forall x \forall y \; suc(x,y) \;\Rightarrow\; \big(x \in X \iff y \notin X\big)$$

and contains the first position,

$$\forall x \exists y \;\; y \geq x \;\wedge\; y \notin X$$

but does not contain the last position

$$\forall x \exists y \;\; y \leq x \;\wedge\; y \in X$$

MSO is the extension of first-order logic with set quantification.
Contrary to what the above suggests, MSO is more succint than regular expressions.

MSO=regular for words

MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$\longrightarrow$

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q$ $\quad \delta \subseteq Q \times A \times Q$

MSO=regular for words

Thm. For every regular language *L,* there is an equivalent formula of MSO, and vice versa.

$\longrightarrow$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\qquad I, F \subseteq Q$ $\qquad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$$\longrightarrow$$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q \quad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

exists state assignment

$$\exists X_1 \cdots \exists X_n \left\{ \right.$$

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$$\longrightarrow$$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q$ $\quad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

the first position has an initial state

$$\forall x \ first(x) \Rightarrow \bigvee_{q_i \in I} X_i$$

exists state assignment

$$\exists X_1 \cdots \exists X_n \left\{ \right.$$

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$\longrightarrow$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q \quad$ $\delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

the first position has an initial state

$$\forall x \; first(x) \Rightarrow \bigvee_{q_i \in I} X_i$$

exists state assignment

$$\exists X_1 \cdots \exists X_n \left\{ \right.$$

the transitions are respected

$$\bigwedge_{a \in A} \left(\forall x \forall y \; a(x) \wedge suc(x, y) \; \Rightarrow \bigvee_{(q_i, a, q_j) \in \delta} x \in X_i \wedge y \in X_j \right)$$

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$\longrightarrow$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q$ $\quad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

the first position has an initial state

$$\forall x \; first(x) \Rightarrow \bigvee_{q_i \in I} X_i$$

exists state assignment

$$\exists X_1 \cdots \exists X_n \left\{ \right.$$

the transitions are respected

$$\bigwedge_{a \in A} \left(\forall x \forall y \; a(x) \wedge suc(x,y) \;\Rightarrow \bigvee_{(q_i,a,q_j)\in\delta} x \in X_i \wedge y \in X_j\right)$$

the last position has an accepting state

$$\forall x \; last(x) \Rightarrow \bigvee_{q_i \in F} X_i$$

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$\longrightarrow$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q$ $\quad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

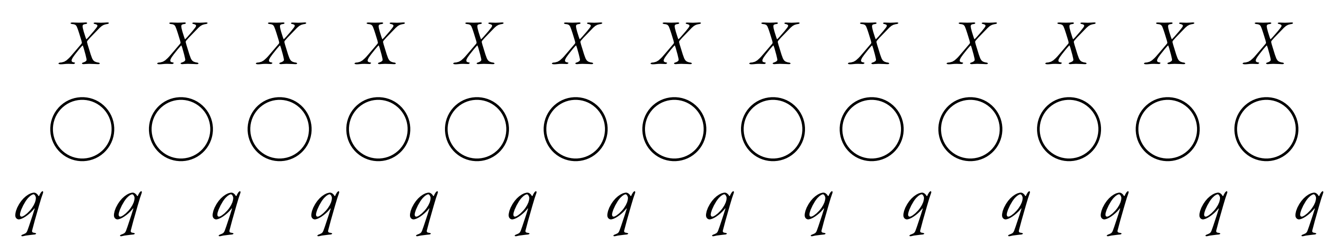
exists state assignment

the first position has an initial state

the transitions are respected

the last position has an accepting state

$$\exists X_1 \cdots \exists X_n \left\{ \begin{array}{l} \forall x\ first(x) \Rightarrow \bigvee_{q_i \in I} X_i \\ \bigwedge_{a \in A} \left(\forall x \forall y\ a(x) \wedge suc(x,y) \ \Rightarrow \bigvee_{(q_i, a, q_j) \in \delta} x \in X_i \wedge y \in X_j \right) \\ \forall x\ last(x) \Rightarrow \bigvee_{q_i \in F} X_i \end{array} \right.$$



MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

$\longrightarrow$

Take an automaton: $Q = \{q_1, \ldots, q_n\}$ $\quad I, F \subseteq Q \quad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

the first position has an initial state

$$\forall x \; first(x) \Rightarrow \bigvee_{q_i \in I} X_i$$

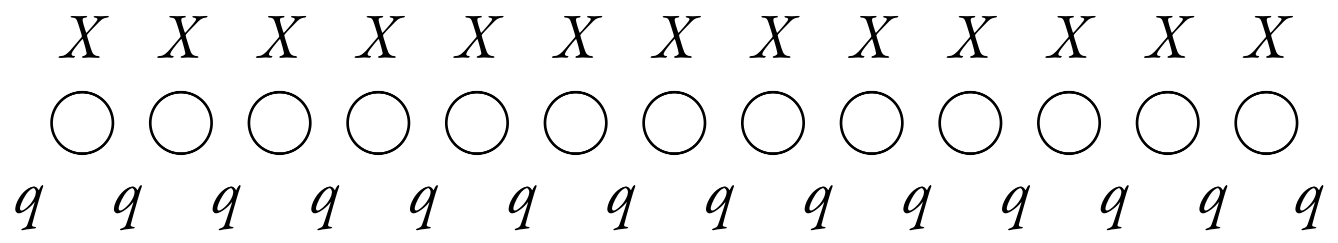
exists state assignment

$$\exists X_1 \cdots \exists X_n \left\{ \right.$$

the transitions are respected

$$\bigwedge_{a \in A} \left(\forall x \forall y \; a(x) \wedge suc(x,y) \;\Rightarrow \bigvee_{(q_i,a,q_j)\in\delta} x \in X_i \wedge y \in X_j\right)$$

the last position has an accepting state



MSO=regular for words

Thm. For every regular language *L,* there is an equivalent formula of MSO, and vice versa.

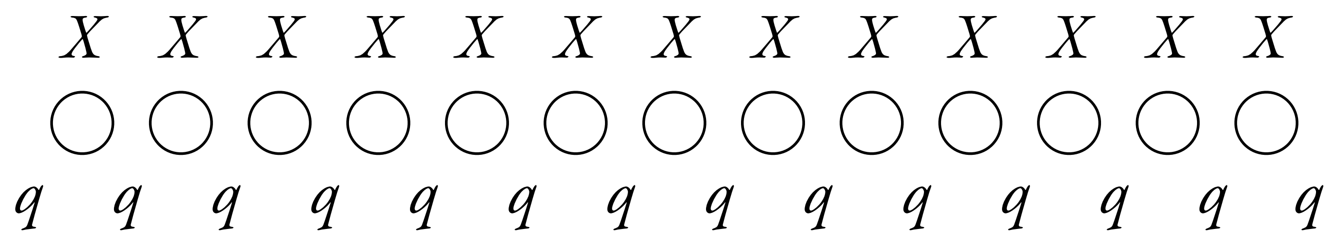
$$\longrightarrow$$

Take an automaton: $Q = \{q_1, \ldots, q_n\} \qquad I, F \subseteq Q \qquad \delta \subseteq Q \times A \times Q$

A word is accepted by the automaton iff it satisfies the following formula of MSO:

exists state assignment

$$\exists X_1 \cdots \exists X_n \left\{ \begin{array}{l} \text{the first position has an initial state} \\ \forall x \ first(x) \Rightarrow \bigvee_{q_i \in I} X_i \\ \text{the transitions are respected} \\ \bigwedge_{a \in A} \left(\forall x \forall y \ a(x) \wedge suc(x,y) \ \Rightarrow \bigvee_{(q_i,a,q_j) \in \delta} x \in X_i \wedge y \in X_j \right) \\ \text{the last position has an accepting state} \\ \bigwedge_{a \in A} \forall x \ last(x) \wedge a(x) \ \Rightarrow \bigvee_{(q_i,a,q_j) \in \delta, q_j \in F} X_i \end{array} \right.$$



Corollary of the proof:

For every regular language, there is an equivalent MSO formula of the form

$$\exists X_1 \ldots \exists X_n \underbrace{\varphi(X_1, \ldots, X_n)}_{\text{first-order}}$$

Corollary of the proof:

For every regular language, there is an equivalent MSO formula of the form

$$\exists X_1 \ldots \exists X_n \underbrace{\varphi(X_1, \ldots, X_n)}_{\text{first-order}}$$

By encoding states in binary, we only need *log(n)* set variables.

Corollary of the proof:

For every regular language, there is an equivalent MSO formula of the form

$$\exists X_1 \ldots \exists X_n \underbrace{\varphi(X_1, \ldots, X_n)}_{\text{first-order}}$$

By encoding states in binary, we only need *log(n)* set variables.
Actually, we only need one.

Corollary of the proof:

For every regular language, there is an equivalent MSO formula of the form

$$\exists X_1 \ldots \exists X_n \underbrace{\varphi(X_1, \ldots, X_n)}_{\text{first-order}}$$

By encoding states in binary, we only need *log(n)* set variables.
Actually, we only need one.

For every regular language, there is an equivalent MSO formula of the form

$$\exists X \underbrace{\varphi(X)}_{\text{first-order}}$$

Corollary of the proof:

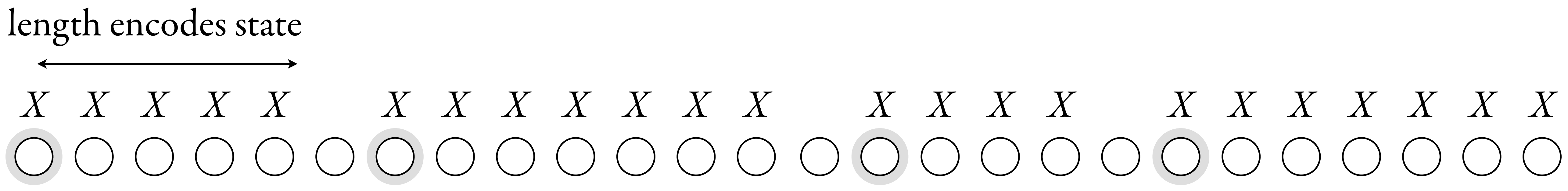
For every regular language, there is an equivalent MSO formula of the form

$$\exists X_1 \ldots \exists X_n \underbrace{\varphi(X_1, \ldots, X_n)}_{\text{first-order}}$$

By encoding states in binary, we only need *log(n)* set variables.
Actually, we only need one.

For every regular language, there is an equivalent MSO formula of the form

$$\exists X \underbrace{\varphi(X)}_{\text{first-order}}$$



Corollary of the proof:

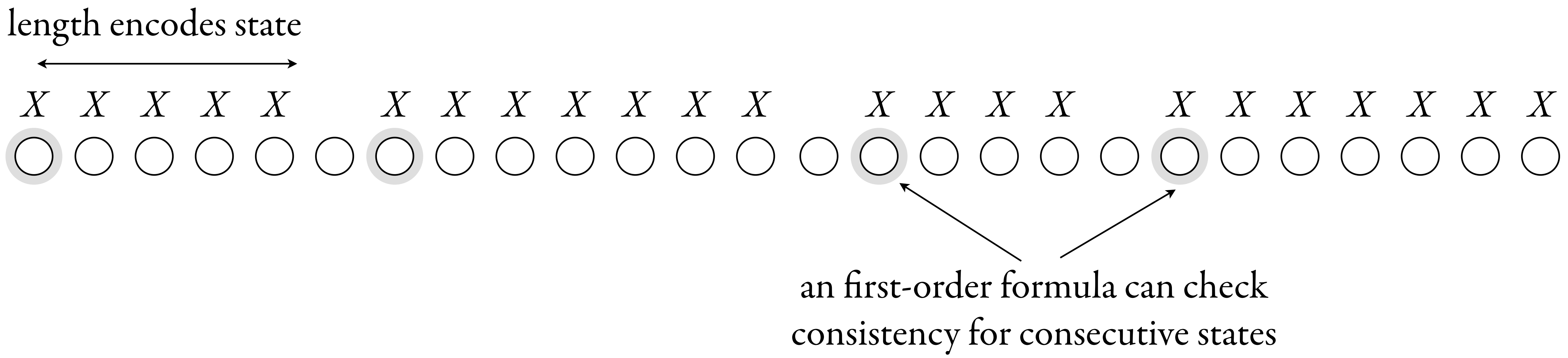
For every regular language, there is an equivalent MSO formula of the form

$$\exists X_1 \ldots \exists X_n \ \underbrace{\varphi(X_1, \ldots, X_n)}_{\text{first-order}}$$

By encoding states in binary, we only need *log(n)* set variables.
Actually, we only need one.

For every regular language, there is an equivalent MSO formula of the form

$$\exists X \ \underbrace{\varphi(X)}_{\text{first-order}}$$


length encodes state
X X X X X X X X X X X X X X X X X X X X X X X X X X
an first-order formula can check consistency for consecutive states

MSO=regular for words

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

$\longleftarrow$

MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

$\longleftarrow$

Proposition. For every sentence Ψ of MSO, the set L_Ψ is regular.

MSO=regular for words

Thm. For every regular language *L*, there is an equivalent formula of MSO, and vice versa.

$\longleftarrow$

Proposition. For every sentence Ψ of MSO, the set L_Ψ is regular.

Proof. Induction on the structure of the formula.

MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

$\longleftarrow$

Proposition. For every sentence Ψ of MSO, the set L_Ψ is regular.

Proof. Induction on the structure of the formula.

Claim. For every formula $\Psi(x_1,x_2,\ldots, x_n, X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

$\longleftarrow$

Proposition. For every sentence Ψ of MSO, the set L_Ψ is regular.

Proof. Induction on the structure of the formula.

Claim. For every formula $\Psi(x_1, x_2, \ldots, x_n, X_1, X_2, \ldots, X_m)$ of MSO, the set L_Ψ is regular.

to simplify, we remove individual variables $x_1, x_2, \ldots, x_n$ from MSO syntax.

MSO=regular for words

Thm. For every regular language L, there is an equivalent formula of MSO, and vice versa.

$\longleftarrow$

Proposition. For every sentence Ψ of MSO, the set L_Ψ is regular.

Proof. Induction on the structure of the formula.

Claim. For every formula $\Psi(x_1,x_2,\dots, x_n, X_1,X_2,\dots, X_m)$ of MSO, the set L_Ψ is regular.

to simplify, we remove individual variables $x_1,x_2,\dots, x_n$ from MSO syntax.

$X \subseteq Y \quad X = \varnothing \quad X \subseteq a \quad X < Y$

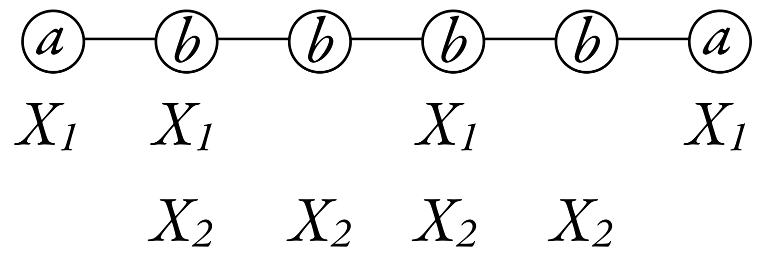
Claim. For every formula $\Psi(X_1, X_2, \ldots, X_m)$ of MSO, the set L_Ψ is regular.

Claim. For every formula $\Psi(X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w\in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.

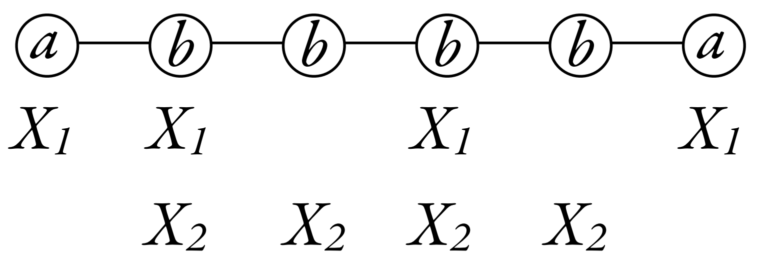
Claim. For every formula $\Psi(X_1, X_2, \ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1, \ldots, X_n$?
A word $w \in A^*$ together with valuations for sets $X_1, \ldots, X_n$ is represented as a word over $A \times \{0,1\}^n$.

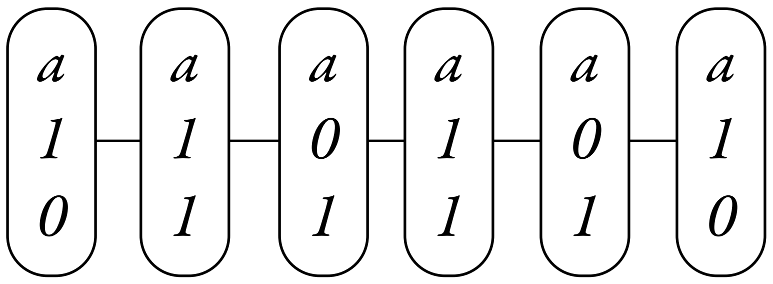


Claim. For every formula $\Psi(X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w\in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.

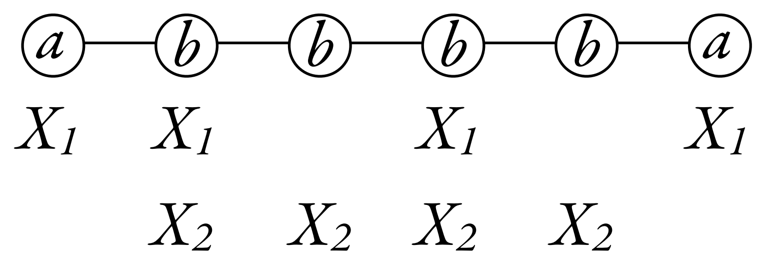


is encoded as

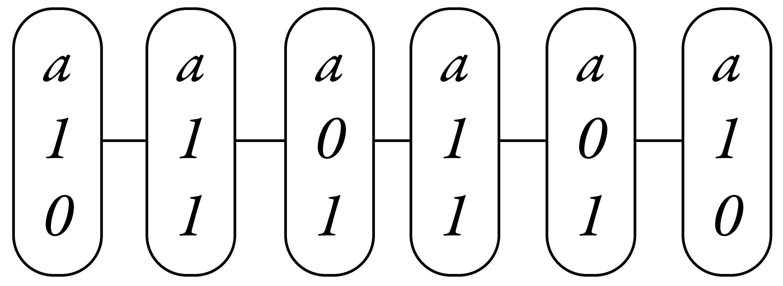


Claim. For every formula $\Psi(X_1, X_2, \dots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1, \dots, X_n$?
A word $w \in A^*$ together with valuations for sets $X_1, \dots, X_n$ is represented as a word over $A \times \{0,1\}^n$.



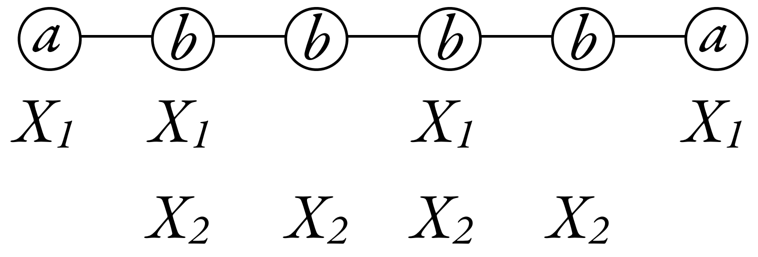
is encoded as



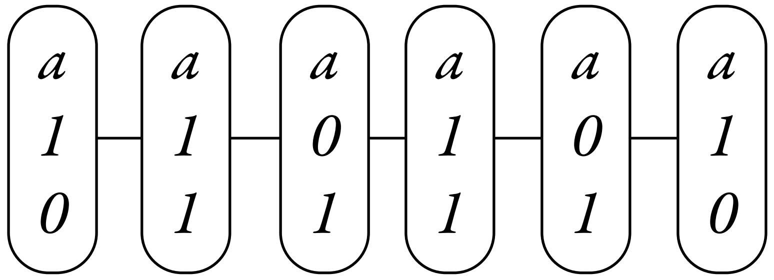
Under this encoding, L_Ψ is a language over $A \times \{0,1\}^n$.

Claim. For every formula $\Psi(X_1, X_2, \ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1, \ldots, X_n$?
A word $w \in A^*$ together with valuations for sets $X_1, \ldots, X_n$ is represented as a word over $A \times \{0,1\}^n$.



is encoded as

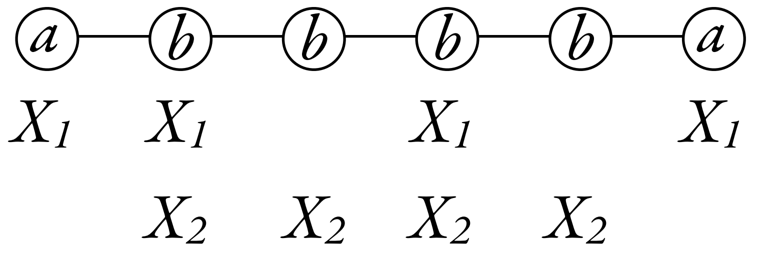


Under this encoding, L_Ψ is a language over $A \times \{0,1\}^n$.

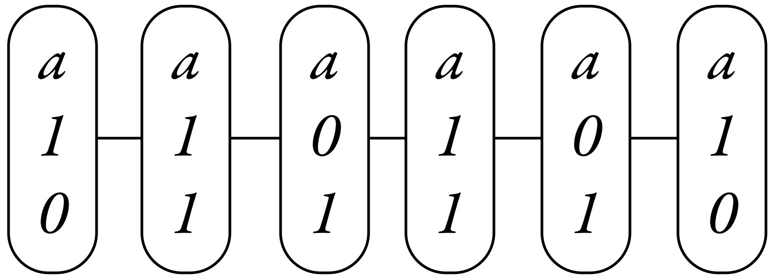
Induction proof of claim.

Claim. For every formula $\Psi(X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w \in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.



is encoded as



Under this encoding, L_Ψ is a language over $A\times\{0,1\}^n$.

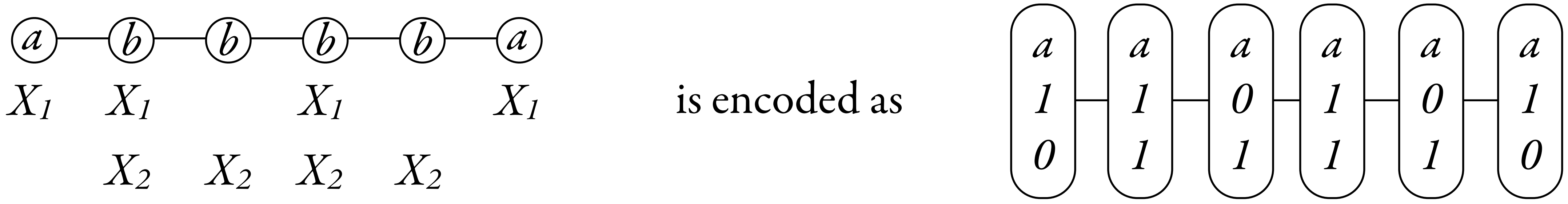
Induction proof of claim.

induction base.

simple. Eg. $X_i \subseteq X_j$ is the regular language “if true on bit i then true on bit j”

Claim. For every formula $\Psi(X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w\in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.



Under this encoding, L_Ψ is a language over $A\times\{0,1\}^n$.

Induction proof of claim.

induction base.

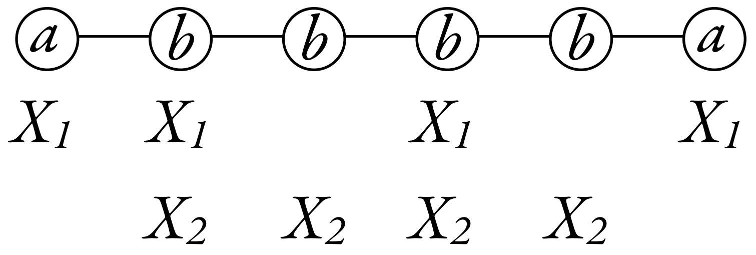
simple. Eg. $X_i\subseteq X_j$ is the regular language “if true on bit i then true on bit j”

boolean operations

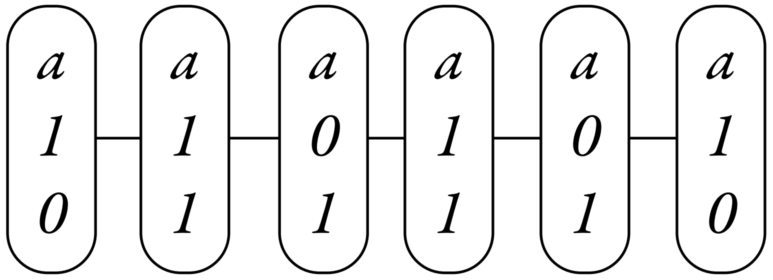
standard constructions for automata.

Claim. For every formula $\Psi(X_1, X_2, \ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w \in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.



is encoded as



Under this encoding, L_Ψ is a language over $A\times\{0,1\}^n$.

Induction proof of claim.

induction base.

simple. Eg. $X_i \subseteq X_j$ is the regular language "if true on bit i then true on bit j"

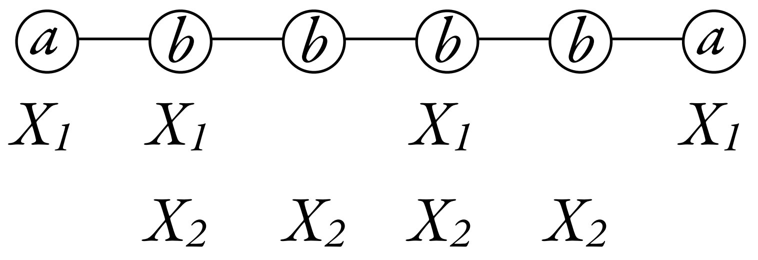
boolean operations

standard constructions for automata.

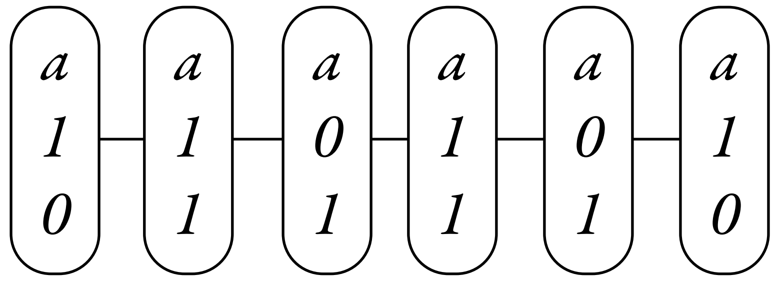
existential quantification $\exists X_m.\ \Psi(X_1, X_2, \ldots, X_m)$

Claim. For every formula $\Psi(X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w \in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A \times \{0,1\}^n$.



is encoded as



Under this encoding, L_Ψ is a language over $A \times \{0,1\}^n$.

Induction proof of claim.

induction base.

simple. Eg. $X_i \subseteq X_j$ is the regular language "if true on bit i then true on bit j"

boolean operations

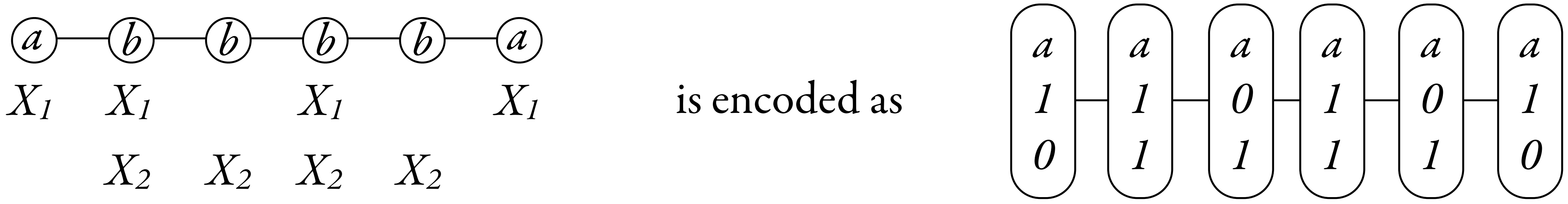
standard constructions for automata.

existential quantification $\exists X_m.\ \Psi(X_1,X_2,\ldots, X_m)$

run the automaton for $\Psi(X_1,X_2,\ldots, X_m)$, nondeterministically guessing values for X_m.

Claim. For every formula $\Psi(X_1,X_2,\ldots, X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w \in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.



Under this encoding, L_Ψ is a language over $A\times\{0,1\}^n$.

Induction proof of claim.

induction base.

simple. Eg. $X_i \subseteq X_j$ is the regular language "if true on bit i then true on bit j"

boolean operations

standard constructions for automata.

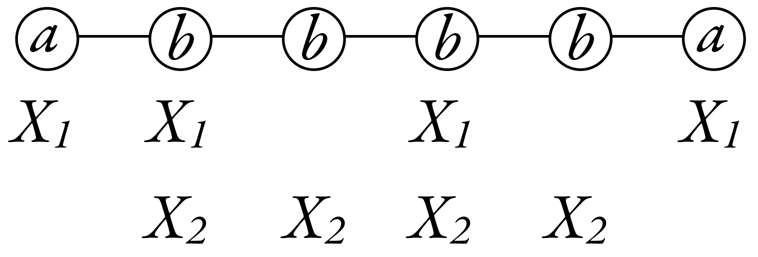
existential quantification $\exists X_m. \Psi(X_1,X_2,\ldots, X_m)$

run the automaton for $\Psi(X_1,X_2,\ldots, X_m)$, nondeterministically guessing values for X_m.

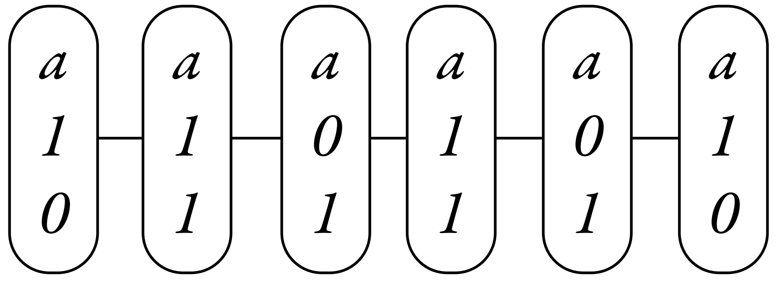
Language of $\exists X_m. \Psi(X_1,X_2,\ldots, X_m)$ = Projection under π of language of $\Psi(X_1,X_2,\ldots, X_m)$

Claim. For every formula $\Psi(X_1,X_2,\ldots,X_m)$ of MSO, the set L_Ψ is regular.

How do we define L_Ψ for formulas with free set variables $X_1,\ldots,X_n$?
A word $w\in A^*$ together with valuations for sets $X_1,\ldots,X_n$ is represented as a word over $A\times\{0,1\}^n$.



is encoded as



Under this encoding, L_Ψ is a language over $A\times\{0,1\}^n$.

Induction proof of claim.

induction base.

simple. Eg. $X_i\subseteq X_j$ is the regular language “if true on bit i then true on bit j”

boolean operations

standard constructions for automata.

existential quantification $\exists X_m.\,\Psi(X_1,X_2,\ldots,X_m)$

run the automaton for $\Psi(X_1,X_2,\ldots,X_m)$, nondeterministically guessing values for X_m.

Language of $\exists X_m.\,\Psi(X_1,X_2,\ldots,X_m)$ = Projection under π of language of $\Psi(X_1,X_2,\ldots,X_m)$

$$\pi : (A\times\{0,1\}^n)^* \longrightarrow (A\times\{0,1\}^{n-1})^*$$

Monadic- and First-Order Logic for Words

definition

weakness of first-order logic

MSO=regular

Monadic- and First-Order Logic for Trees

Transitive Closure Logic and Regular Expressions

Monadic- and First-Order Logic for Words

- definition
- weakness of first-order logic
- MSO=regular

Monadic- and First-Order Logic for Trees

- definition
- problems with parity
- problems with aperiodicity

Transitive Closure Logic and Regular Expressions

MSO for Trees

A binary tree has an even number of nodes

iff

there is a set of positions

$\exists X$

$\left\{ \right.$

that contains no leaf

$\forall x \exists y \;\; y \geq x \;\wedge\; y \notin X$

but contains the root

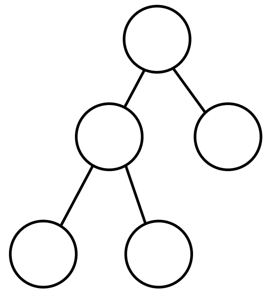
$\forall x \exists y \;\; y \leq x \;\wedge\; y \in X$

and contains a node iff exactly on of its children is in X

$\forall x \forall y_0 \forall y_1 \; \big(suc_0(x, y_0) \wedge suc_1(x, y_1)\big) \;\Rightarrow\; \big(x \notin X$

iff

false



MSO for Trees

A binary tree has an even number of nodes

iff

there is a set of positions

$\exists X$

that contains no leaf

$\forall x \exists y \quad y \geq x \ \wedge \ y \notin X$

but contains the root

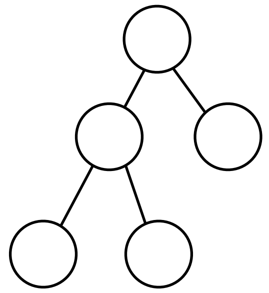
$\forall x \exists y \quad y \leq x \ \wedge \ y \in X$

and contains a node iff exactly on of its children is in X

$\forall x \forall y_0 \forall y_1 \ \big(suc_0(x, y_0) \wedge suc_1(x, y_1)\big) \ \Rightarrow \ \big(x \notin X$

iff

false



Thm. (Thatcher, Wright `68)

MSO for Trees

A binary tree has an even number of nodes

iff

there is a set of positions

$\exists X$

that contains no leaf

$\forall x \exists y \;\; y \geq x \;\wedge\; y \notin X$

but contains the root

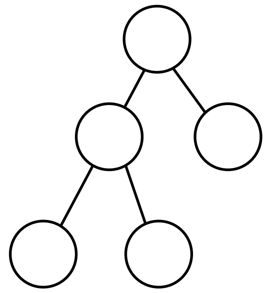
$\forall x \exists y \;\; y \leq x \;\wedge\; y \in X$

and contains a node iff exactly on of its children is in X

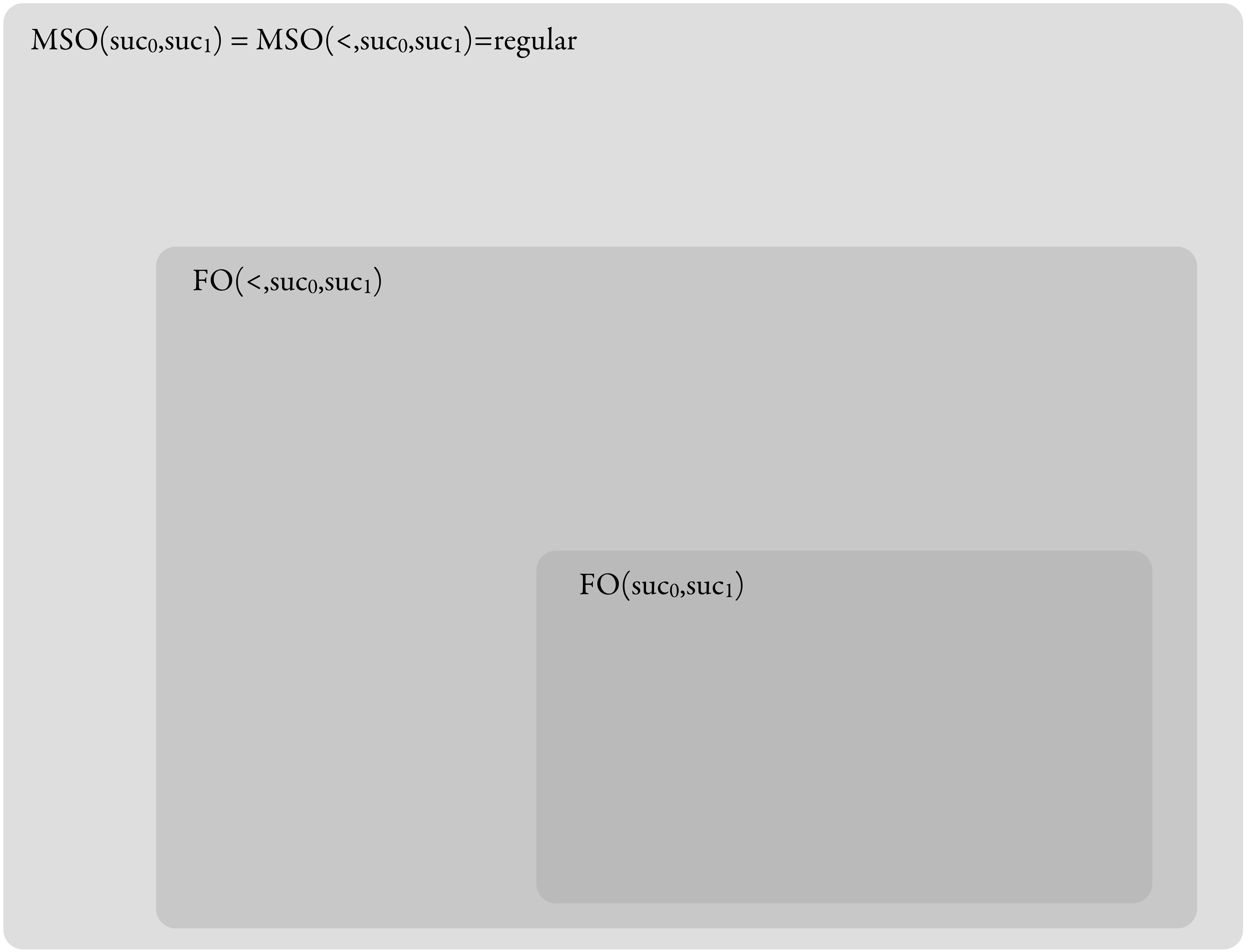
$\forall x \forall y_0 \forall y_1 \; \big(suc_0(x, y_0) \wedge suc_1(x, y_1)\big) \;\Rightarrow\; \big(x \notin X$

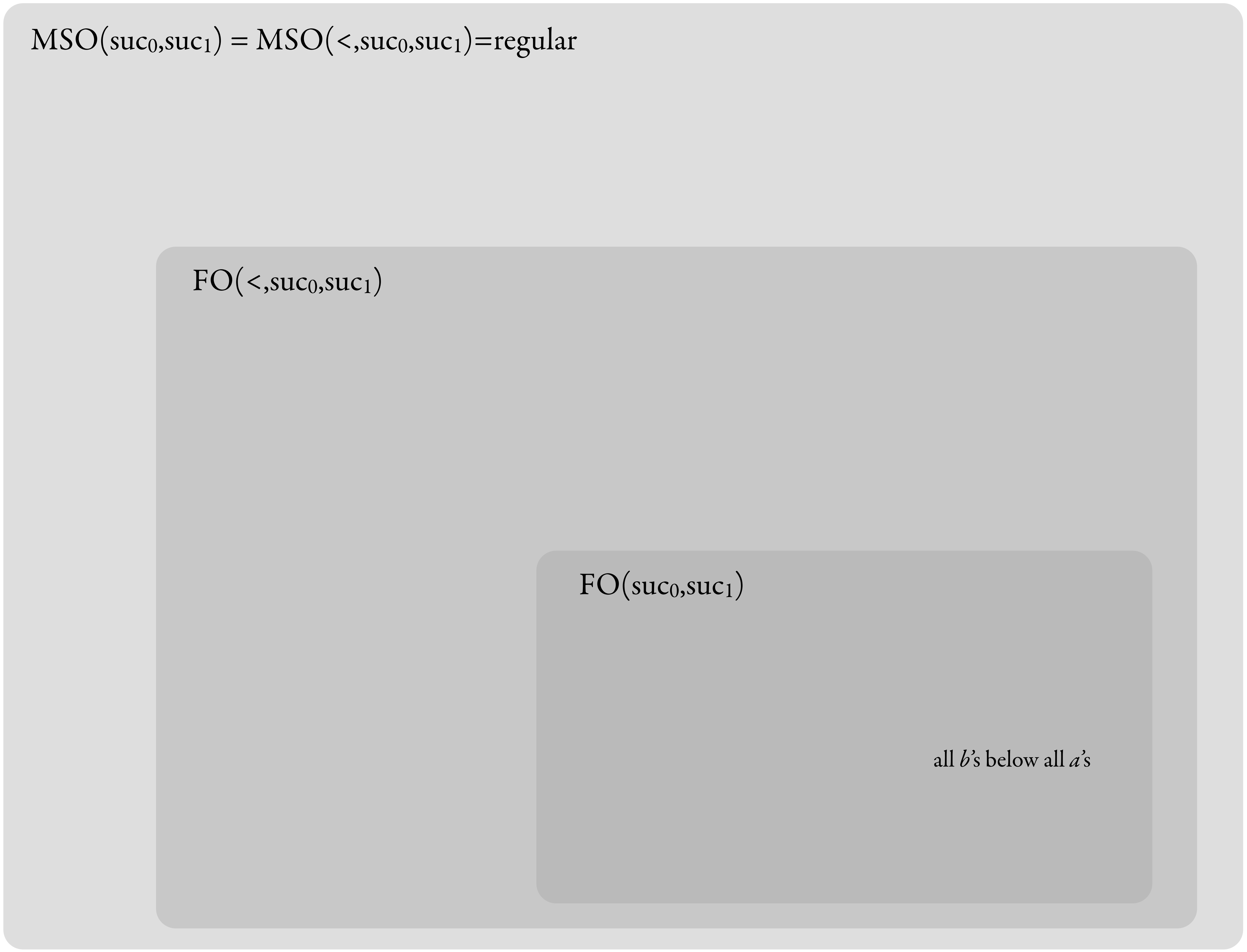
iff

false



Thm. (Thatcher, Wright `68)
MSO = regular languages for finite trees.


$\mathrm{MSO}(\mathrm{suc}_0,\mathrm{suc}_1) = \mathrm{MSO}(<,\mathrm{suc}_0,\mathrm{suc}_1)=\mathrm{regular}$
$\mathrm{FO}(<,\mathrm{suc}_0,\mathrm{suc}_1)$
$\mathrm{FO}(\mathrm{suc}_0,\mathrm{suc}_1)$

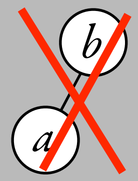

MSO(suc_0,suc_1) = MSO(<,suc_0,suc_1)=regular
FO(<,suc_0,suc_1)
FO(suc_0,suc_1)
all *b*’s below all *a*’s

MSO(suc_0,suc_1) = MSO(<,suc_0,suc_1)=regular

FO(<,suc_0,suc_1)

FO(suc_0,suc_1)

all *b*'s below all *a*'s



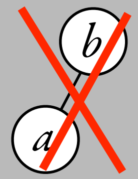
MSO(suc_0,suc_1) = MSO(<,suc_0,suc_1)=regular

FO(<,suc_0,suc_1)

all *b*'s below all *a*'s
for alphabet *a,b,c*

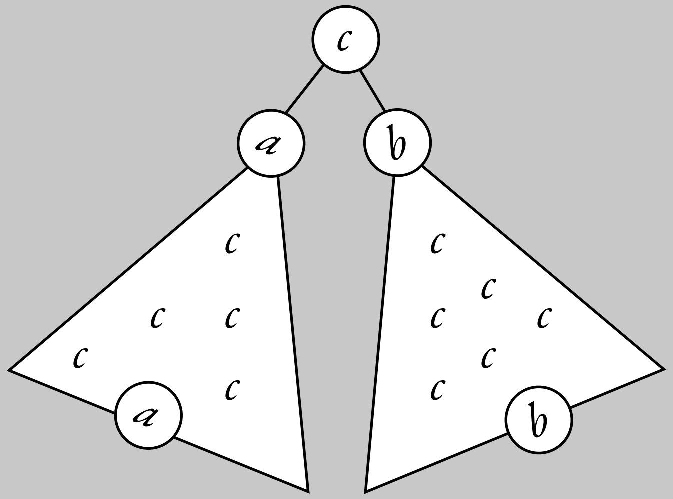
FO(suc_0,suc_1)

all *b*'s below all *a*'s



MSO(suc_0,suc_1) = MSO(<,suc_0,suc_1)=regular

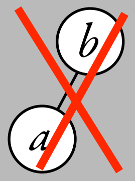
FO(<,suc_0,suc_1)



all *b*'s below all *a*'s
for alphabet *a,b,c*

FO(suc_0,suc_1)

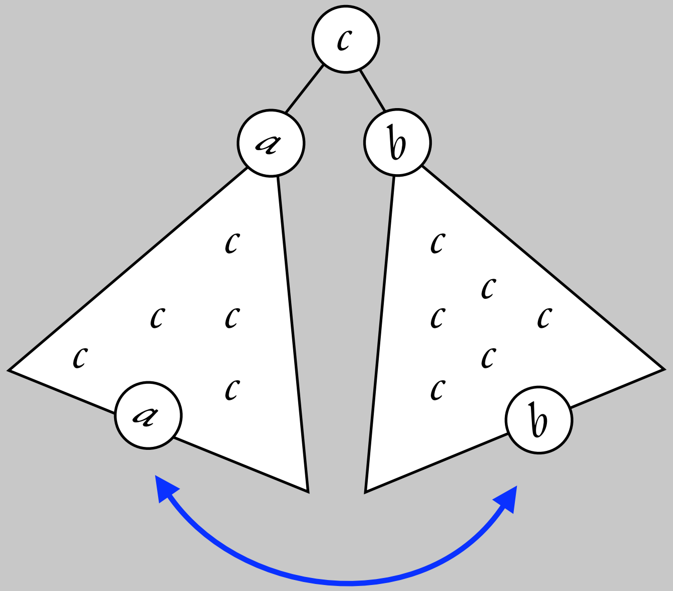
all *b*'s below all *a*'s



MSO(suc_0,suc_1) = MSO(<,suc_0,suc_1)=regular

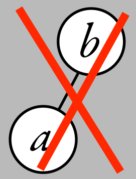
FO(<,suc_0,suc_1)

all b's below all a's
for alphabet a,b,c



FO(suc_0,suc_1)

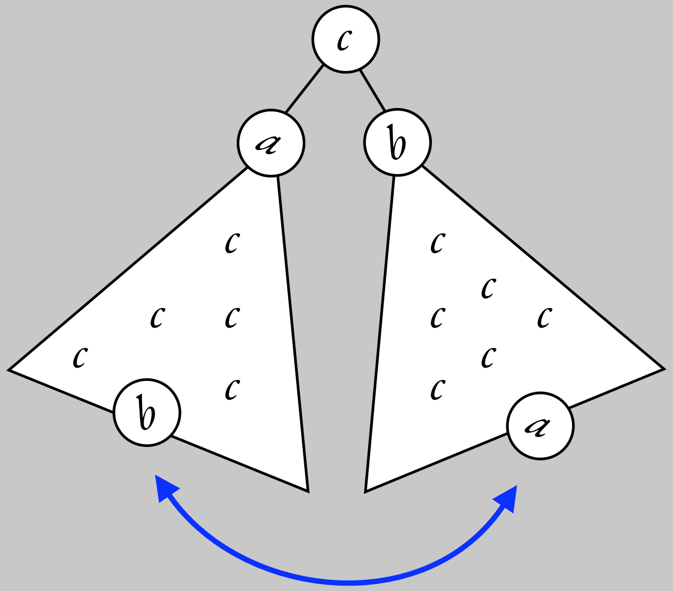
all b's below all a's



MSO(suc_0,suc_1) = MSO($<$,suc_0,suc_1)=regular

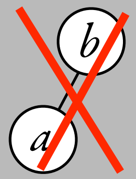
FO($<$,suc_0,suc_1)

all *b*'s below all *a*'s
for alphabet *a*,*b*,*c*



FO(suc_0,suc_1)

all *b*'s below all *a*'s

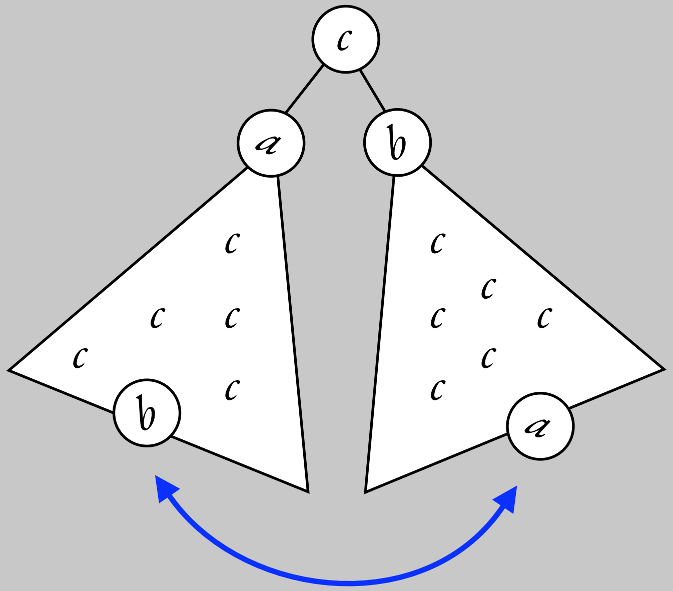


MSO(suc_0,suc_1) = MSO(<,suc_0,suc_1)=regular

parity

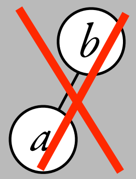
FO(<,suc_0,suc_1)

all *b*'s below all *a*'s
for alphabet *a*,*b*,*c*



FO(suc_0,suc_1)

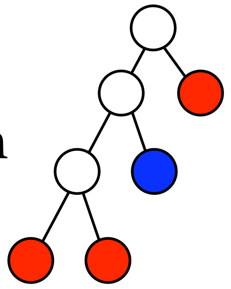
all *b*'s below all *a*'s



Parity

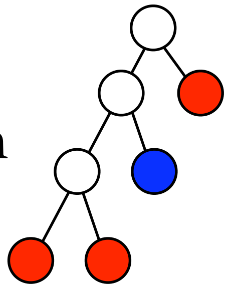
Parity

L=Exists a leaf at even depth



Parity

L=Exists a leaf at even depth

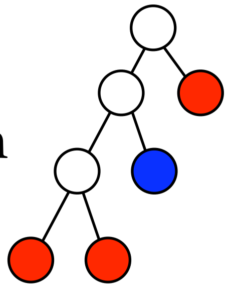


Surprise (Potthof)

This language is definable in FO($<$,suc_0,suc_1)

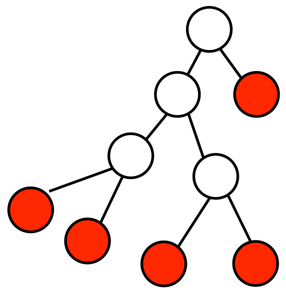
Parity

L=Exists a leaf at even depth

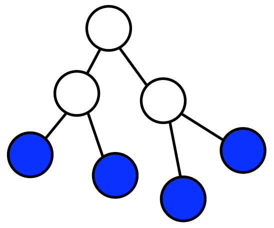


Surprise (Potthof)
This language is definable in FO(<,suc_0,suc_1)

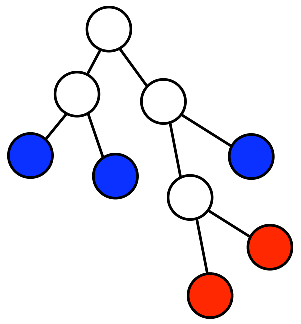
all leaves at even depth



all leaves at odd depth

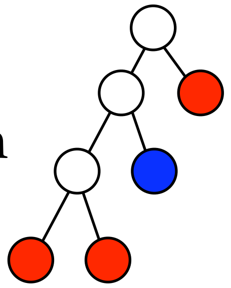


both parities



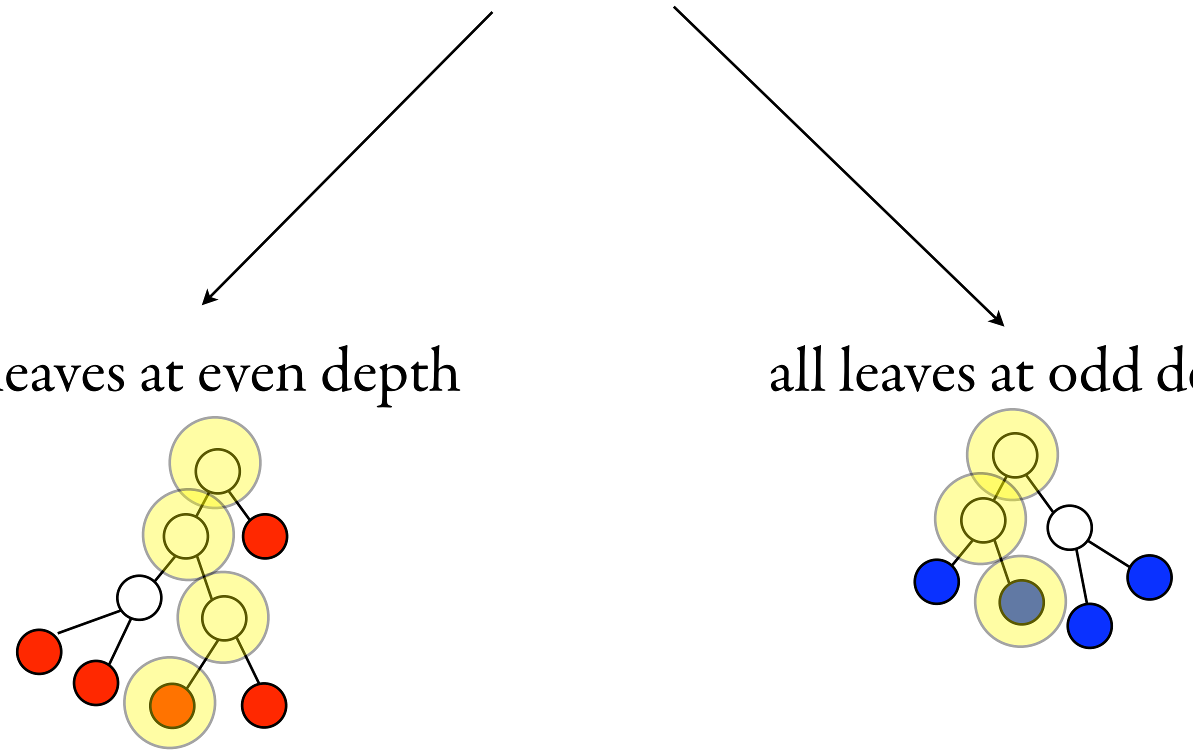
Parity

L=Exists a leaf at even depth

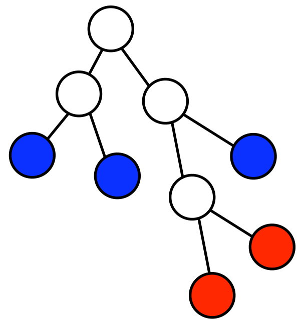


Surprise (Potthof)
This language is definable in FO(<,suc_0,suc_1)

to disinguish between these two,
follow the left zigzag

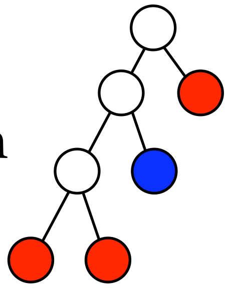


both parities



Parity

L=Exists a leaf at even depth

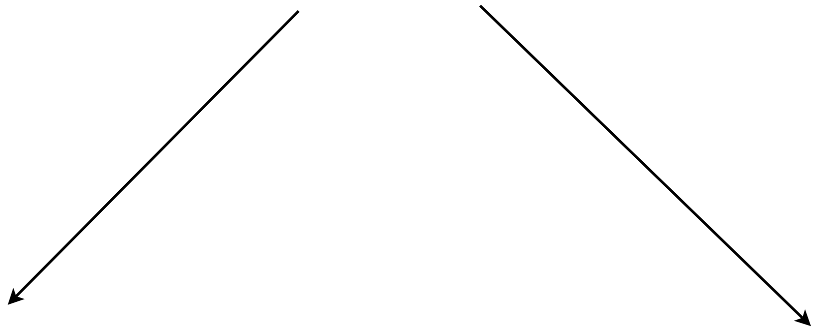


Surprise (Potthof)

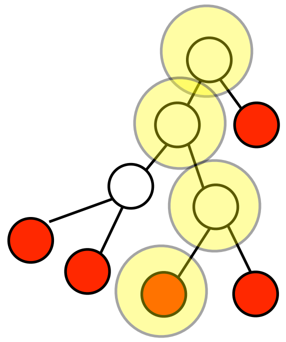
This language is definable in FO($<$,suc_0,suc_1)

A node is on the zigzag if for every left child ancestor,
its parent is a right child or the root (and vice versa).
The left zigzag starts with a left turn.

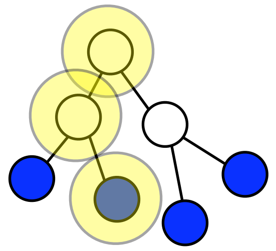
to disinguish between these two,
follow the left zigzag



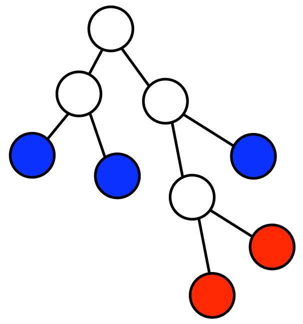
all leaves at even depth



all leaves at odd depth

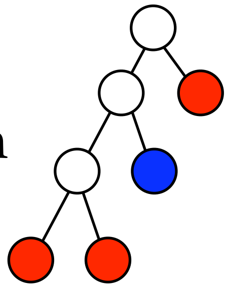


both parities



Parity

L=Exists a leaf at even depth

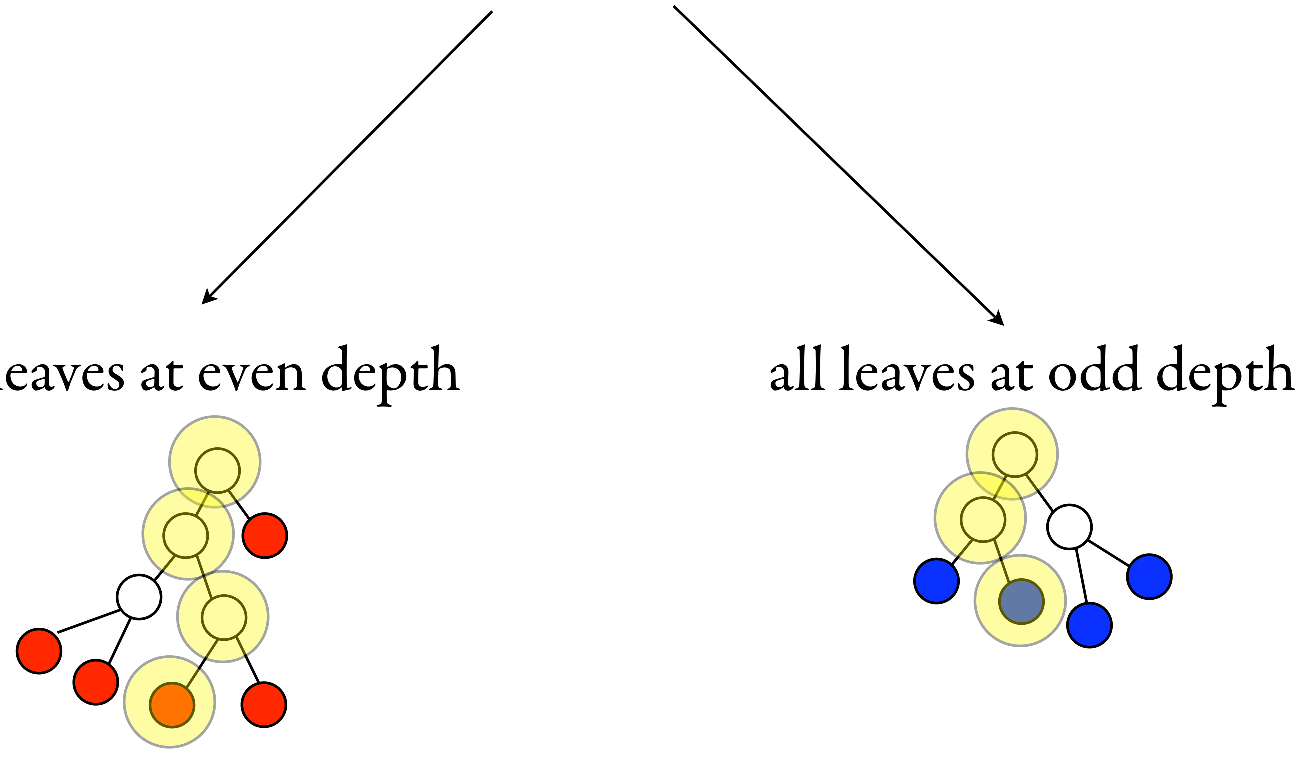


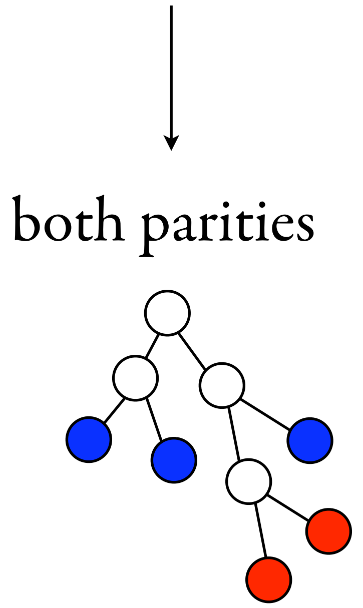
Surprise (Potthof)
This language is definable in FO($<$,suc$_0$,suc$_1$)

A node is on the zigzag if for every left child ancestor, its parent is a right child or the root (and vice versa). The left zigzag starts with a left turn.

to disinguish between these two, follow the left zigzag

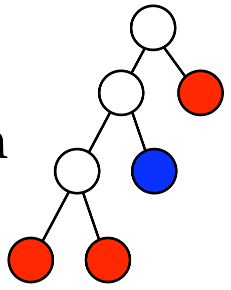
to detect this one, search for conflicting zigzags



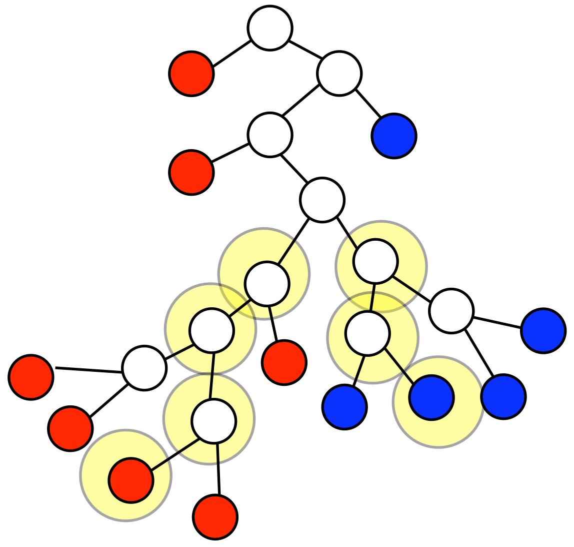


Parity

L=Exists a leaf at even depth



Surprise (Potthof)
This language is definable in FO(<,suc_0,suc_1)

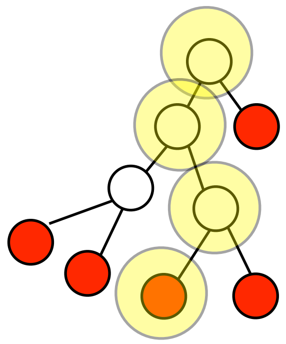


A node is on the zigzag if for every left child ancestor,
its parent is a right child or the root (and vice versa).
The left zigzag starts with a left turn.

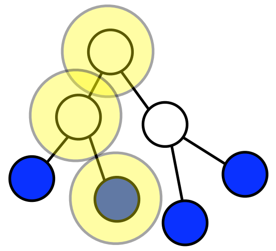
to disinguish between these two,
follow the left zigzag

to detect this one,
search for conflicting zigzags

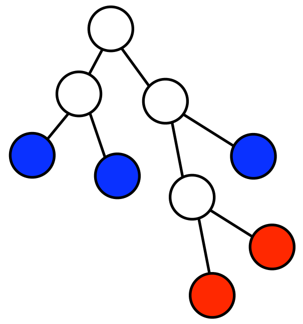
all leaves at even depth



all leaves at odd depth

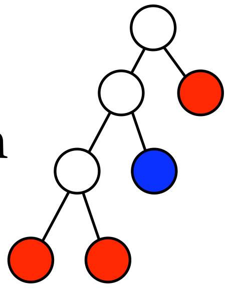


both parities

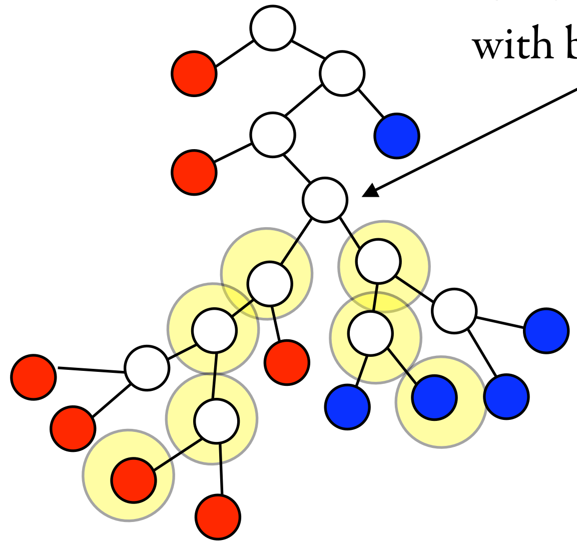


Parity

L=Exists a leaf at even depth



Surprise (Potthof)
This language is definable in FO(<,suc_0,suc_1)

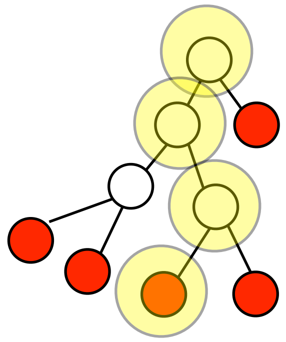


A node is on the zigzag if for every left child ancestor,
its parent is a right child or the root (and vice versa).
The left zigzag starts with a left turn.

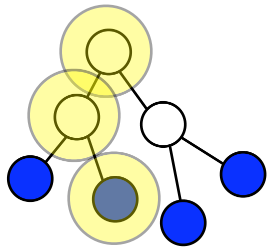
to disinguish between these two,
follow the left zigzag

to detect this one,
search for conflicting zigzags

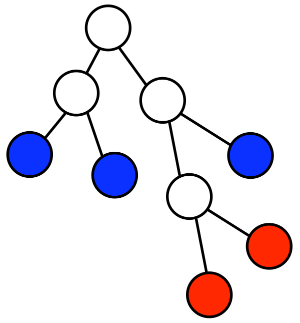
all leaves at even depth



all leaves at odd depth

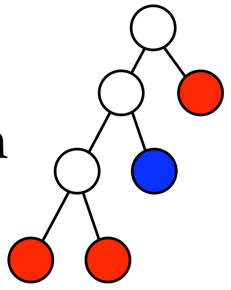


both parities

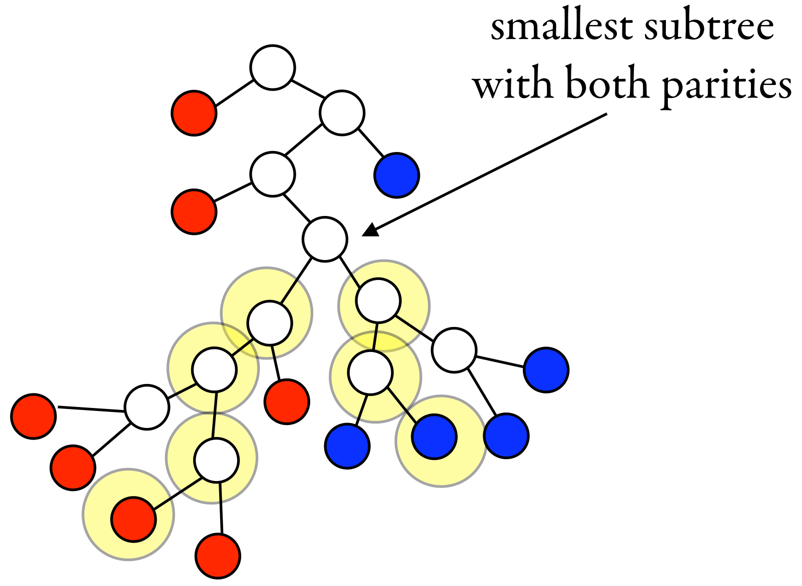


Parity

L=Exists a leaf at even depth



Surprise (Potthof)
This language is definable in FO(<,suc_0,suc_1)

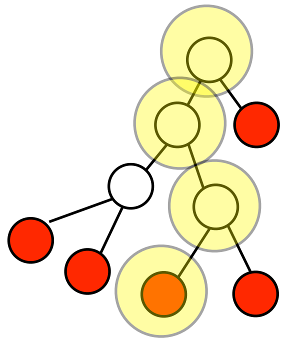


A node is on the zigzag if for every left child ancestor,
its parent is a right child or the root (and vice versa).
The left zigzag starts with a left turn.

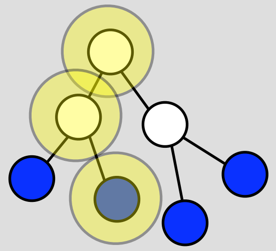
to disinguish between these two,
follow the left zigzag

to detect this one,
search for conflicting zigzags

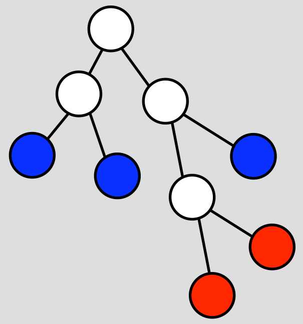
all leaves at even depth



all leaves at odd depth



both parities

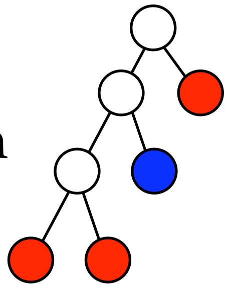


FO(<) $\not\subseteq$ FO(<,suc_0,suc_1)

+

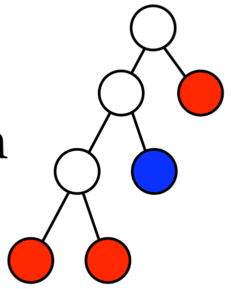
commutative children

L=Exists a leaf at even depth



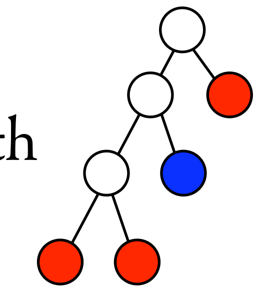
$$\mathrm{FO}(<) \not\subseteq \begin{array}{c}\mathrm{FO}(<,\mathrm{suc}_0,\mathrm{suc}_1)\\ + \\ \text{commutative children}\end{array}$$

L=Exists a leaf at even depth



This language is definable in FO(<,suc$_0$,suc$_1$)...

FO(<) $\not\subset$ FO(<,suc$_0$,suc$_1$) + commutative children

L=Exists a leaf at even depth 

This language is definable in FO(<,suc_0,suc_1)...

...but not in FO(<)

FO(<) $\not\subseteq$ FO(<,suc_0,suc_1) + commutative children

Parity

So what parity language lies outside FO($<$,suc_0,suc_1)?

Parity

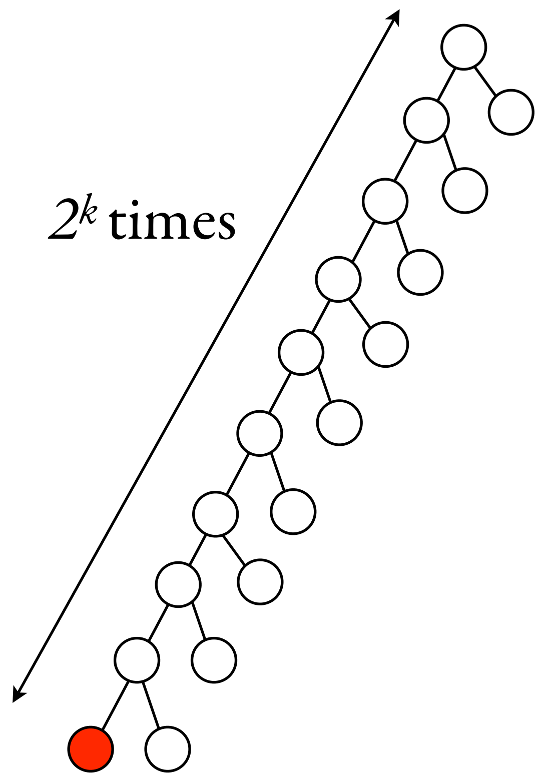
So what parity language lies outside FO($<$,suc_0,suc_1)?

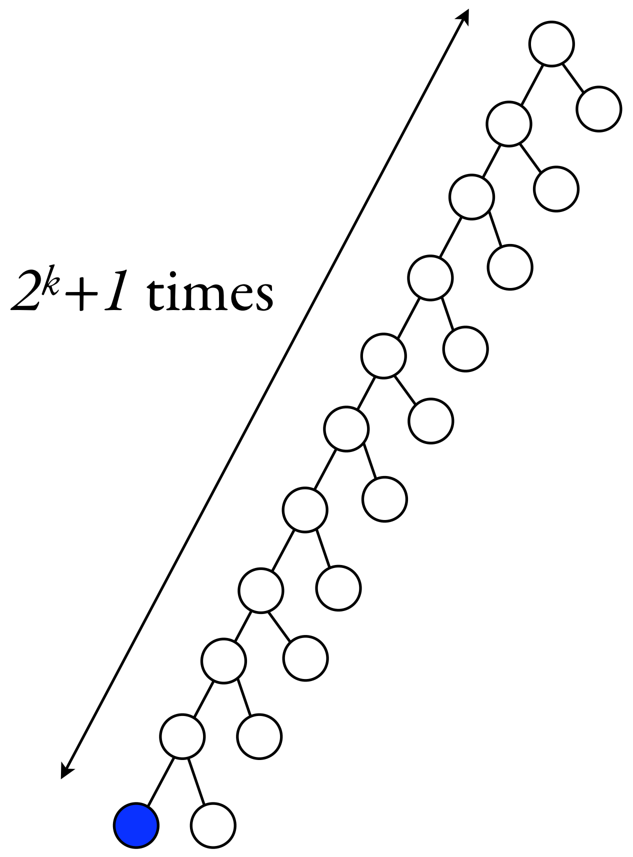
L= "Leftmost leaf has even depth."

Parity

So what parity language lies outside FO($<$,suc_0,suc_1)?

L= "Leftmost leaf has even depth."





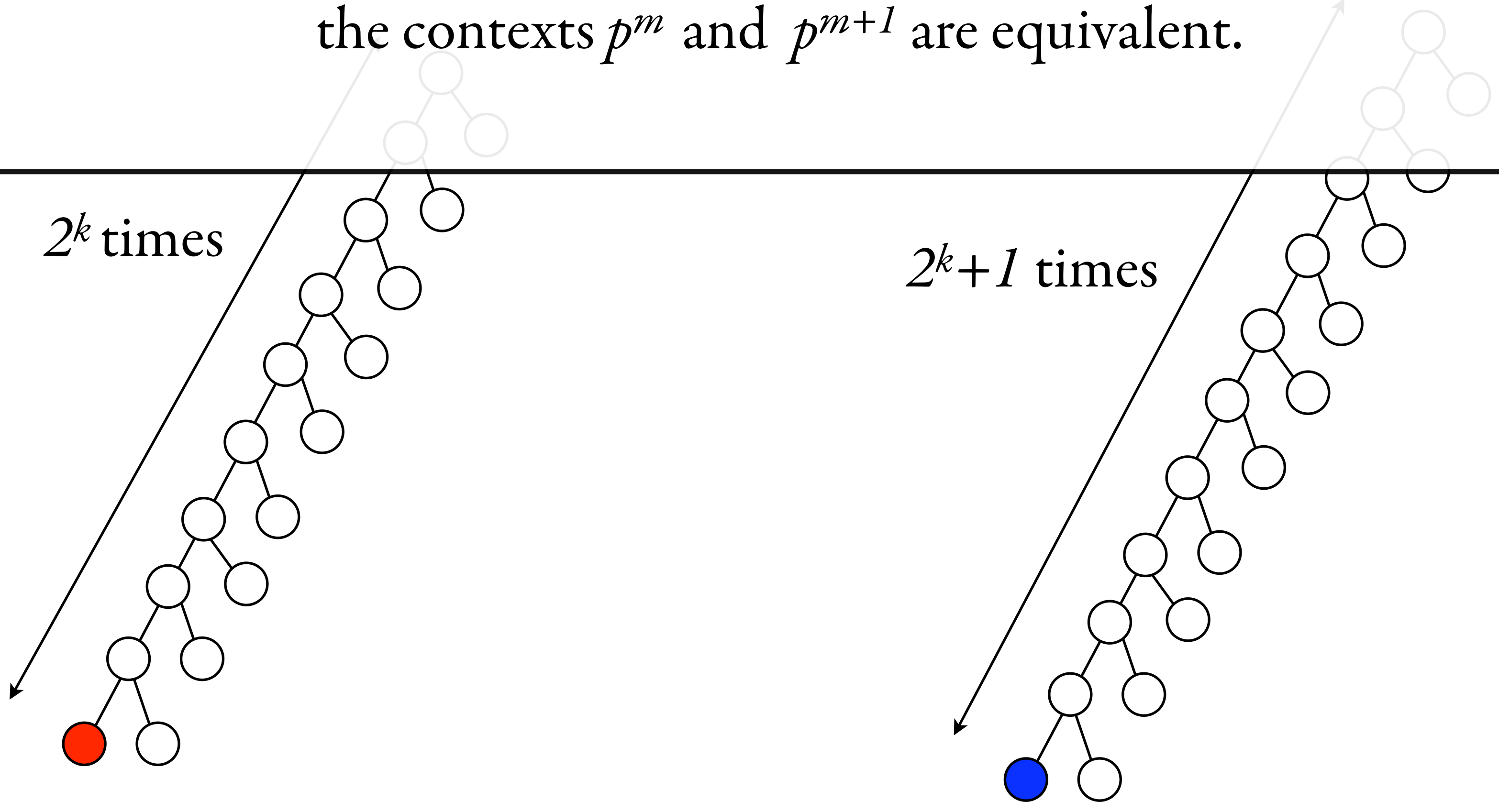
Duplicator survives the k round game on trees

Parity

So what parity language lies outside FO(<,suc$_0$,suc$_1$)?

L= “Leftmost leaf has even depth.”

Lemma. Every tree language definable in FO(<,suc$_0$,suc$_1$) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.



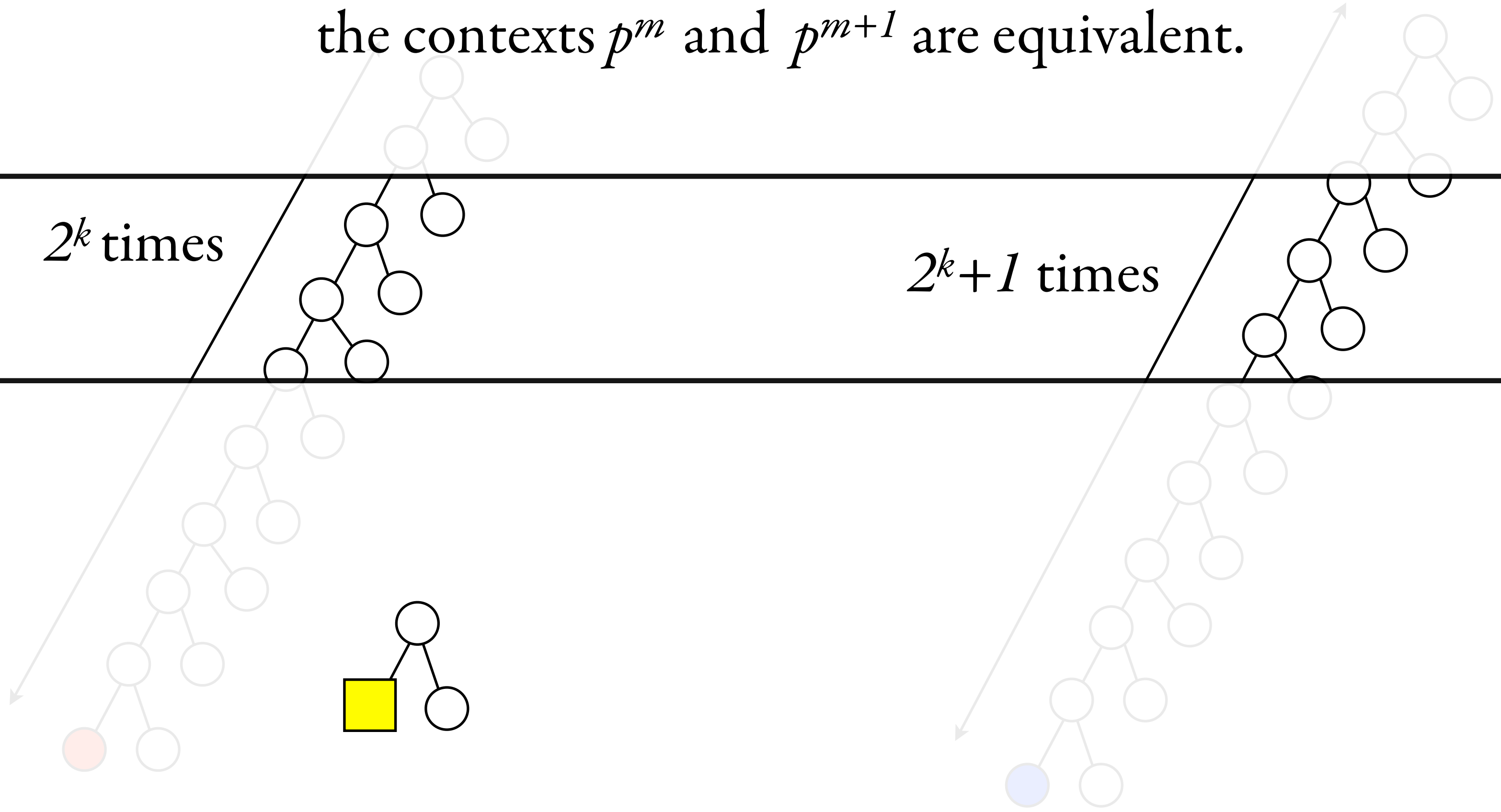
Duplicator survives the k round game on trees

Parity

So what parity language lies outside FO(<,suc_0,suc_1)?

L= "Leftmost leaf has even depth."

Lemma. Every tree language definable in FO(<,suc_0,suc_1) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.



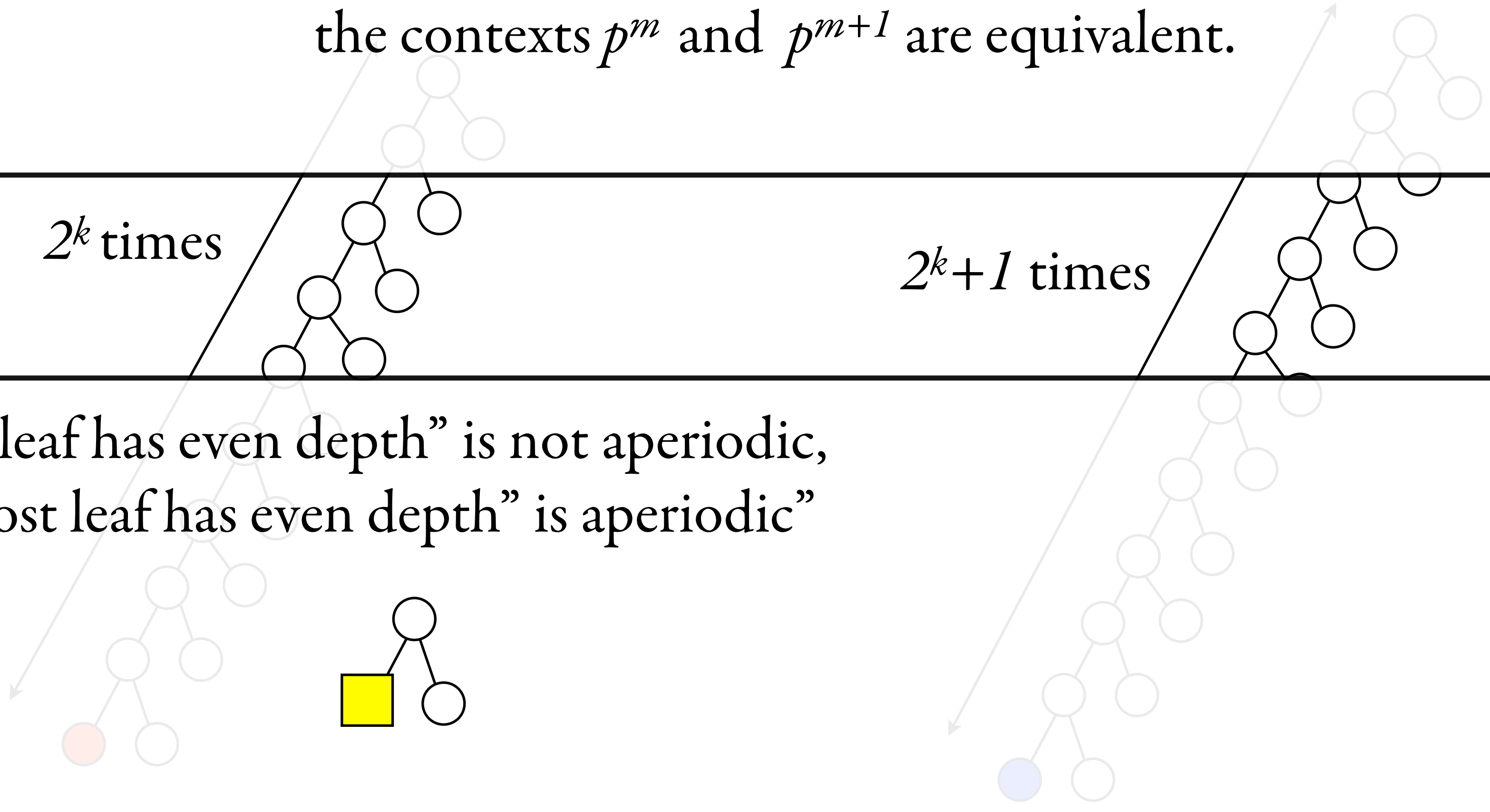
Duplicator survives the k round game on trees

Parity

So what parity language lies outside FO($<$,suc_0,suc_1)?

L= "Leftmost leaf has even depth."

Lemma. Every tree language definable in FO($<$,suc_0,suc_1) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.



"some leaf has even depth" is not aperiodic,
"leftmost leaf has even depth" is aperiodic"

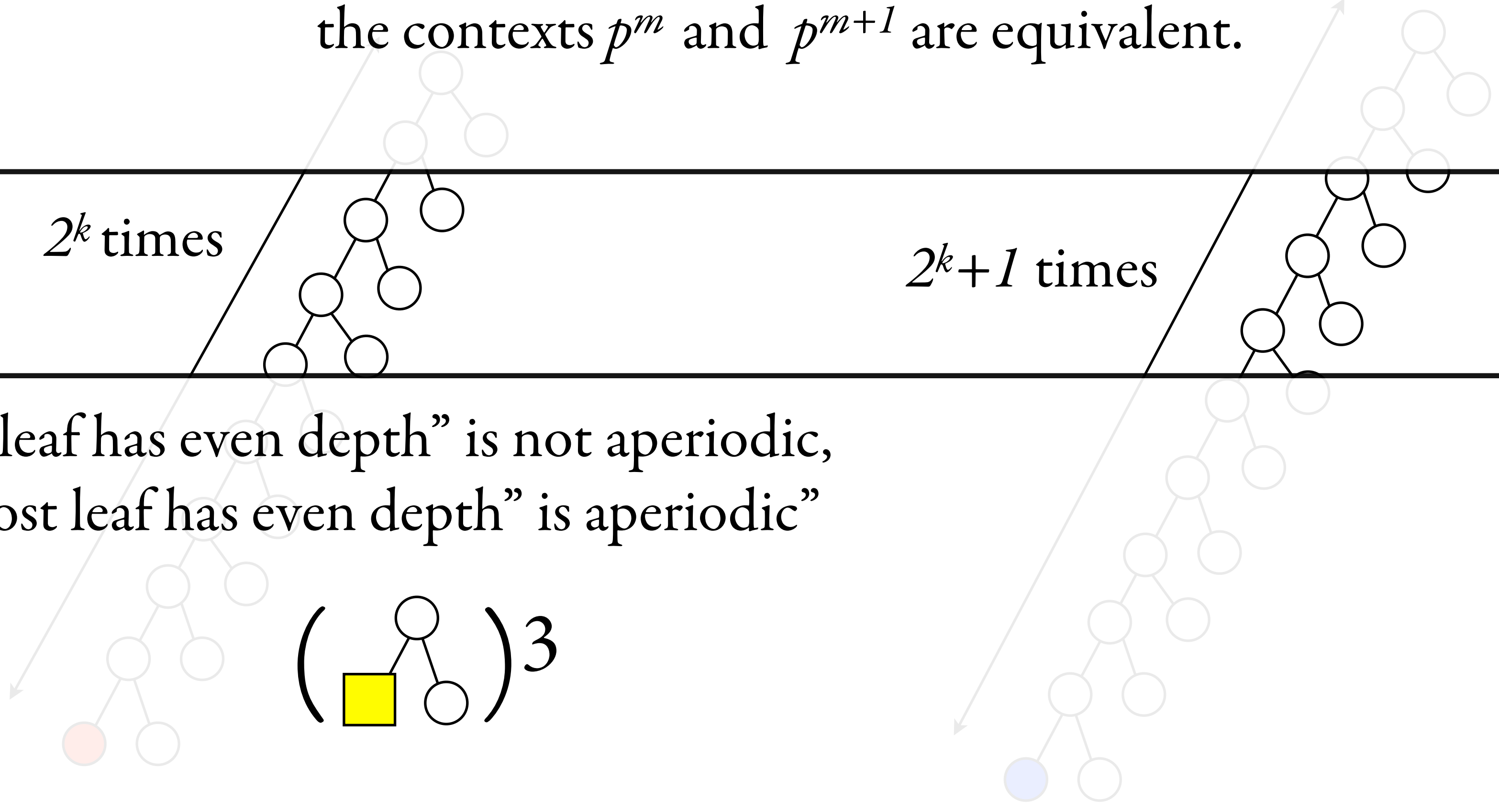
Duplicator survives the k round game on trees

Parity

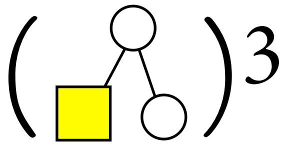
So what parity language lies outside FO(<,suc_0,suc_1)?

L= "Leftmost leaf has even depth."

Lemma. Every tree language definable in FO(<,suc_0,suc_1) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.



"some leaf has even depth" is not aperiodic,
"leftmost leaf has even depth" is aperiodic"



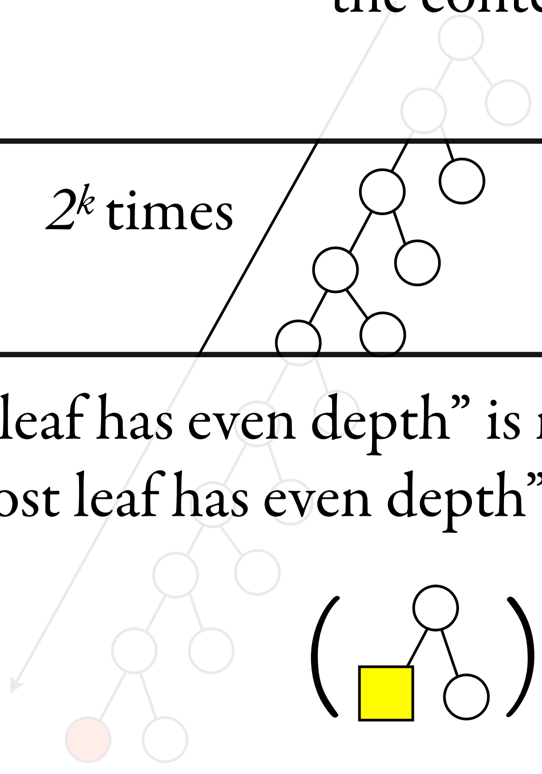
Duplicator survives the k round game on trees

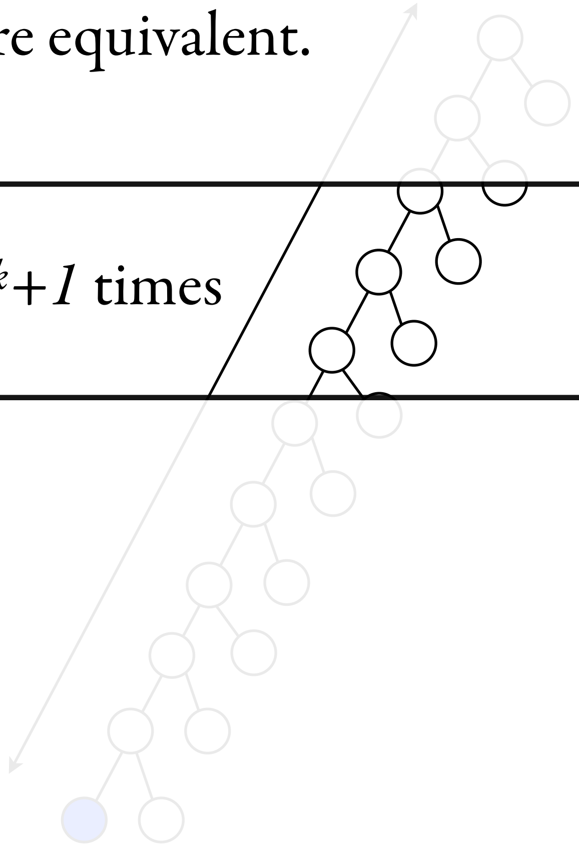
Parity

So what parity language lies outside FO(<,suc$_0$,suc$_1$)?

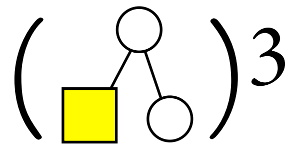
L= "Leftmost leaf has even depth."

Lemma. Every tree language definable in FO(<,suc$_0$,suc$_1$) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.





"some leaf has even depth" is not aperiodic,
"leftmost leaf has even depth" is aperiodic"



$=$

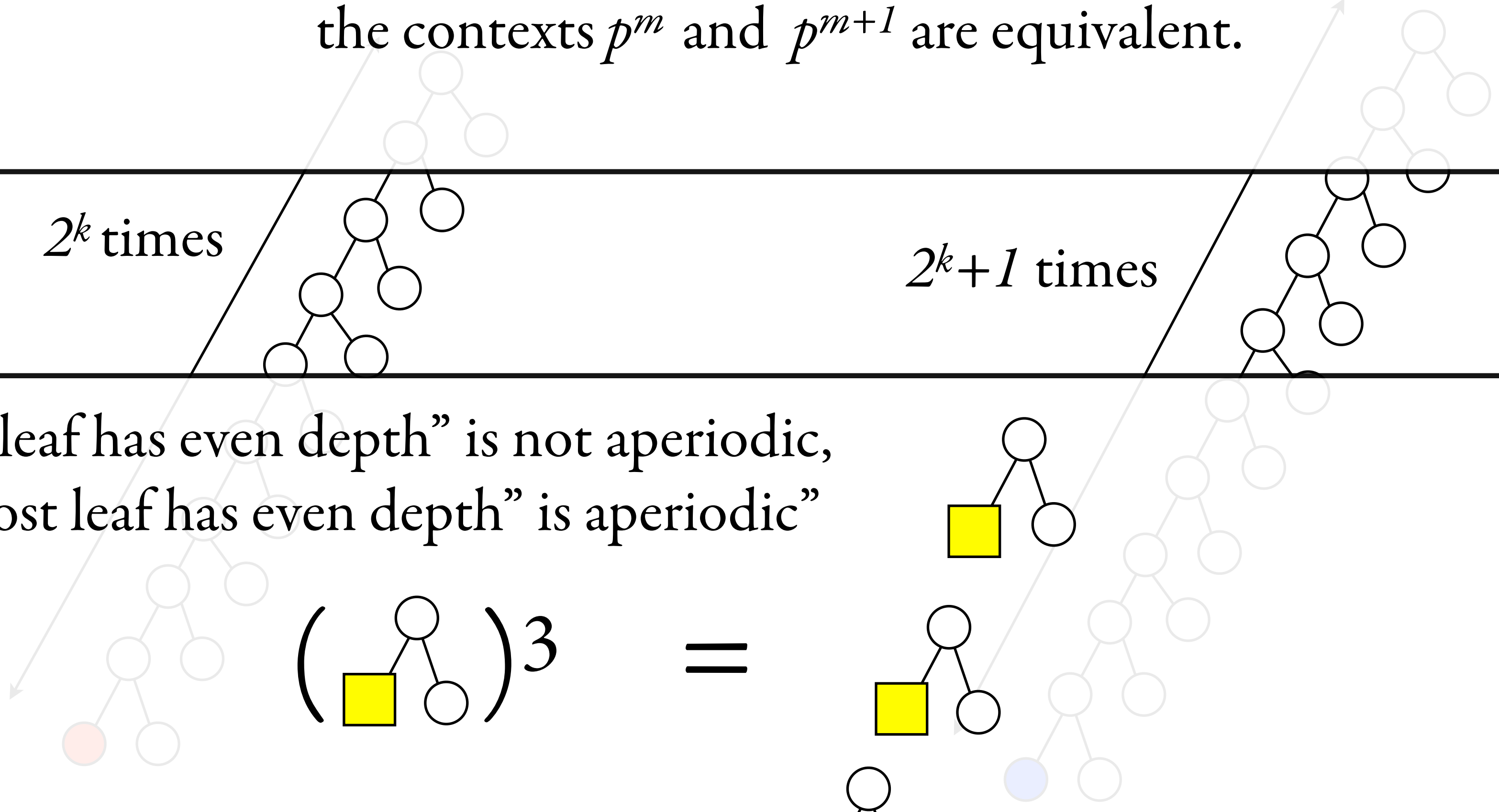
Duplicator survives the k round game on trees

Parity

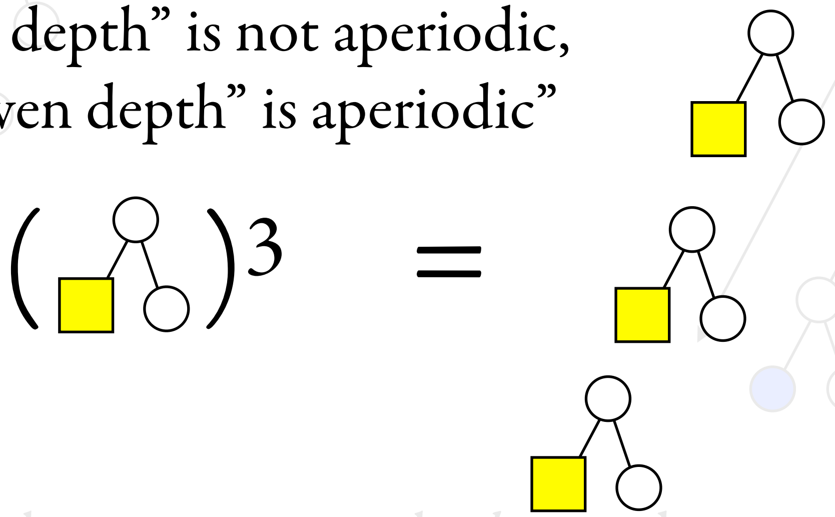
So what parity language lies outside FO(<,suc$_0$,suc$_1$)?

L= “Leftmost leaf has even depth.”

Lemma. Every tree language definable in FO(<,suc$_0$,suc$_1$) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.



“some leaf has even depth” is not aperiodic,
“leftmost leaf has even depth” is aperiodic”



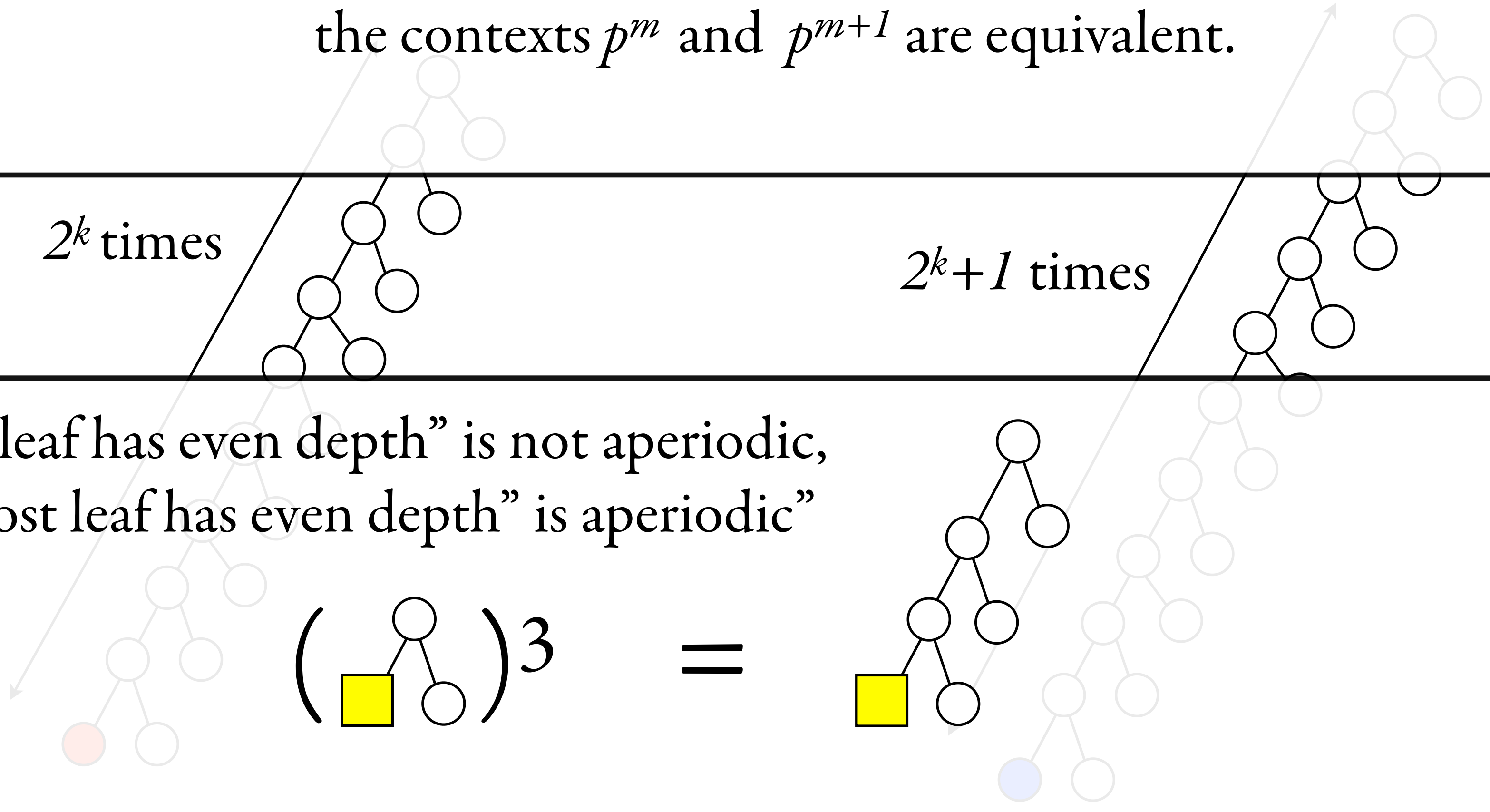
Duplicator survives the k round game on trees

Parity

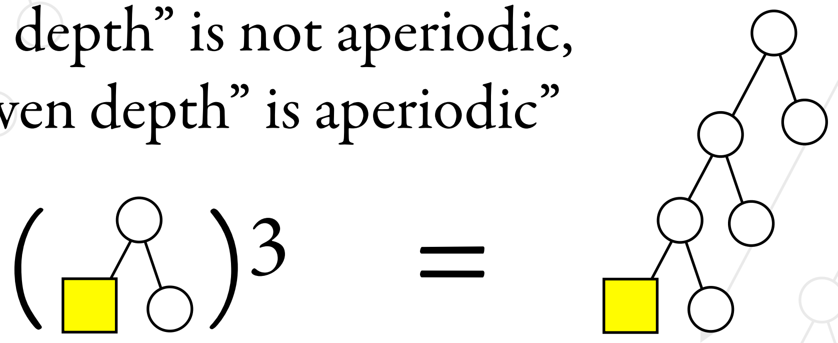
So what parity language lies outside FO(<,suc_0,suc_1)?

L= “Leftmost leaf has even depth.”

Lemma. Every tree language definable in FO(<,suc_0,suc_1) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.



“some leaf has even depth” is not aperiodic,
“leftmost leaf has even depth” is aperiodic”



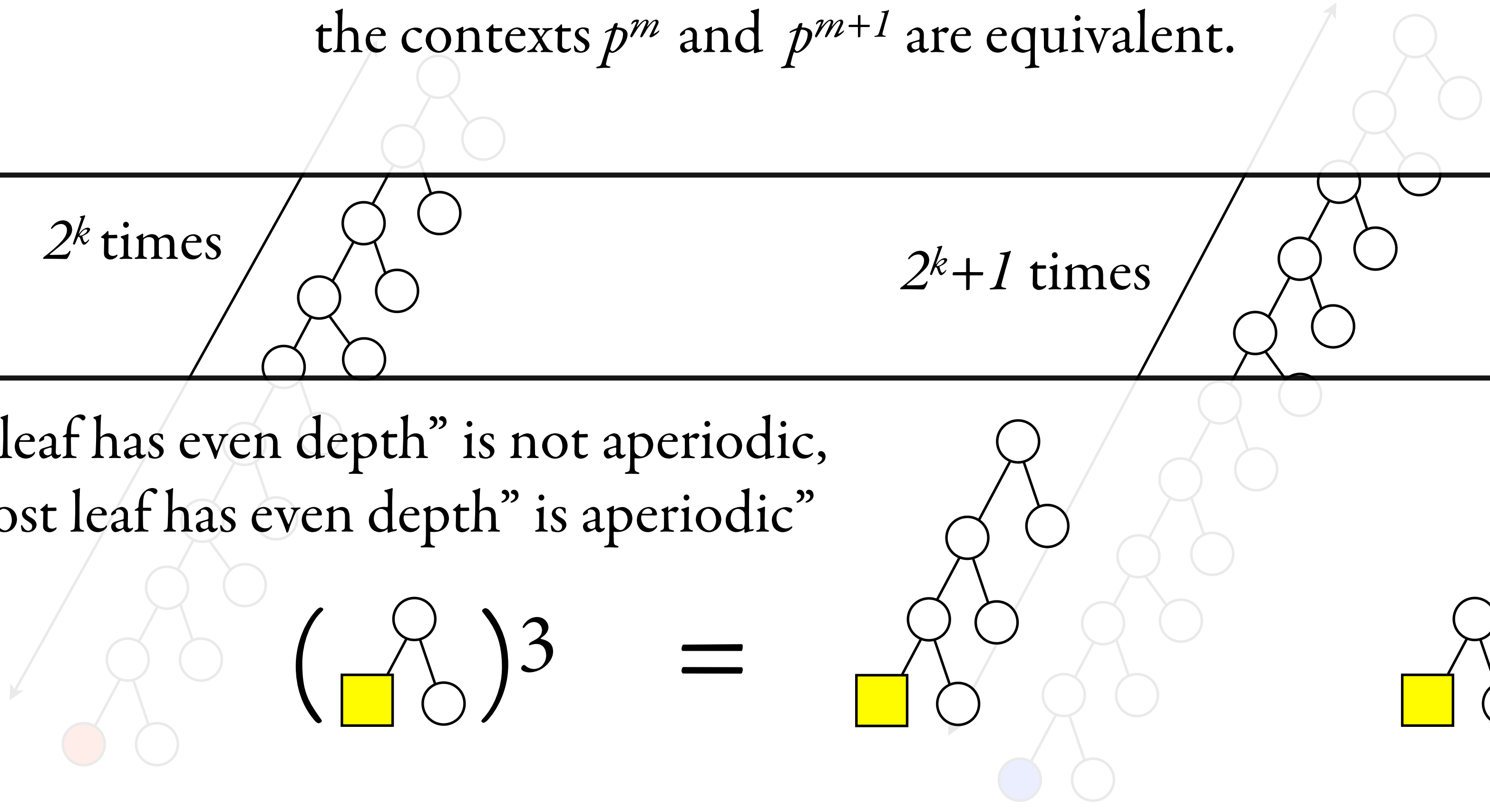
Duplicator survives the k round game on trees

Parity

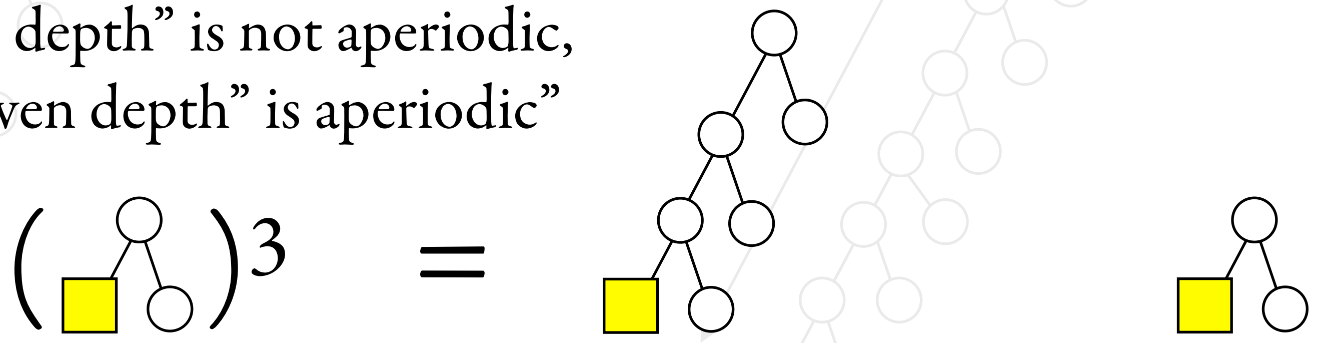
So what parity language lies outside FO(<,suc$_0$,suc$_1$)?

L= "Leftmost leaf has even depth."

Lemma. Every tree language definable in FO(<,suc$_0$,suc$_1$) is aperiodic.
That is, there is some m such that for any context p,
the contexts p^m and p^{m+1} are equivalent.

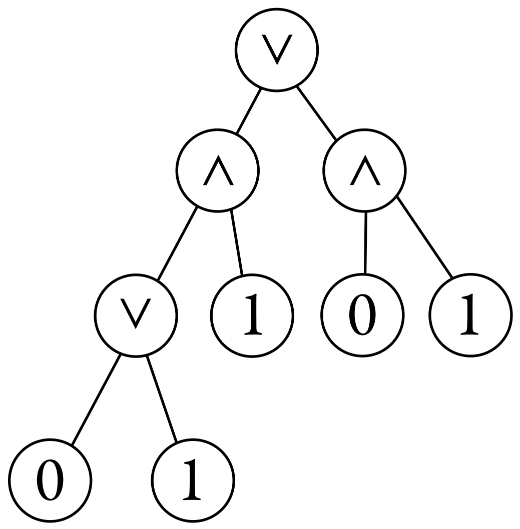


"some leaf has even depth" is not aperiodic,
"leftmost leaf has even depth" is aperiodic"



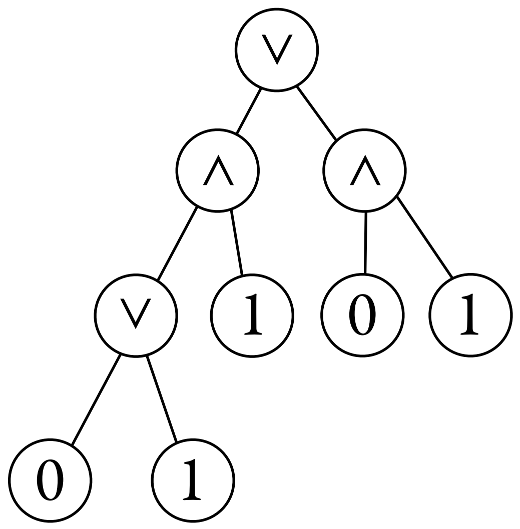
Duplicator survives the k round game on trees

Boolean Expressions



L = “Boolean expressions with value 1”

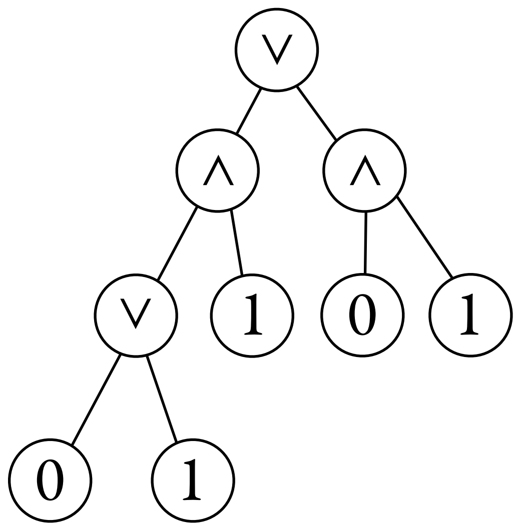
Boolean Expressions



L = “Boolean expressions with value 1”

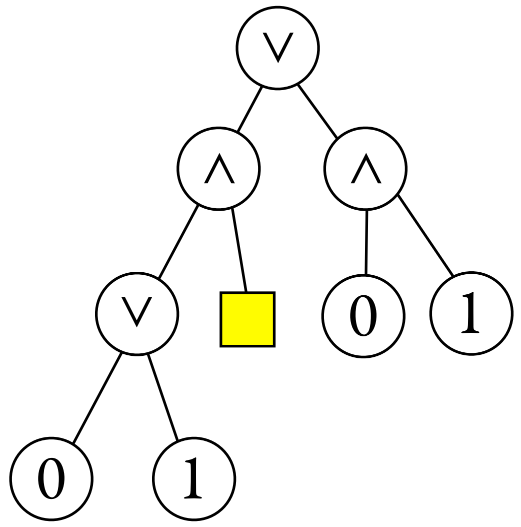
Fact. This language is aperiodic but not definable in FO($<$,suc_0,suc_1).

Boolean Expressions

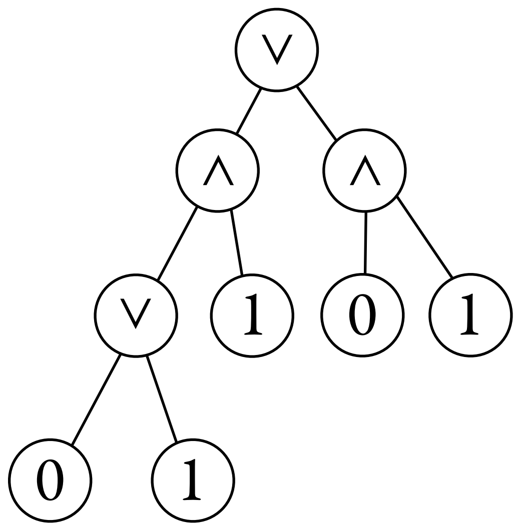


L = “Boolean expressions with value 1”

Fact. This language is aperiodic but not definable in FO(<,suc_0,suc_1).

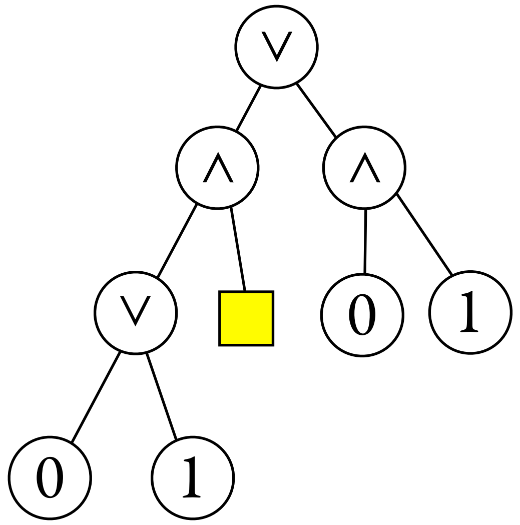


Boolean Expressions



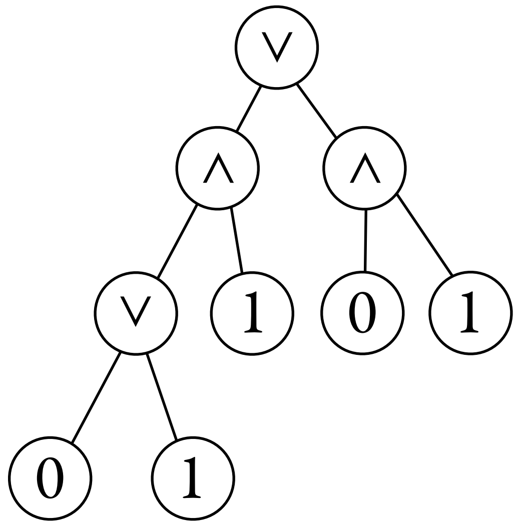
L = "Boolean expressions with value 1"

Fact. This language is aperiodic but not definable in FO($<$,suc_0,suc_1).



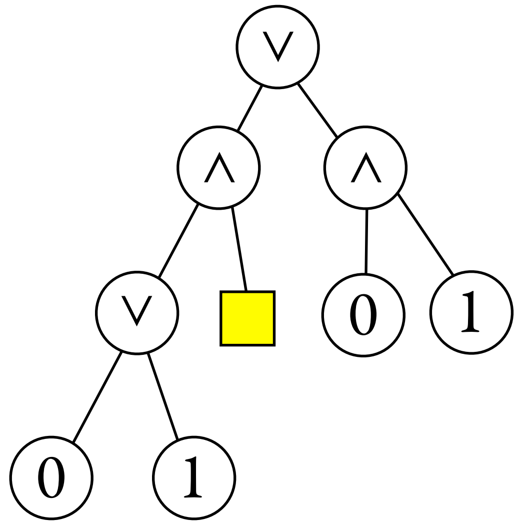
identity

Boolean Expressions

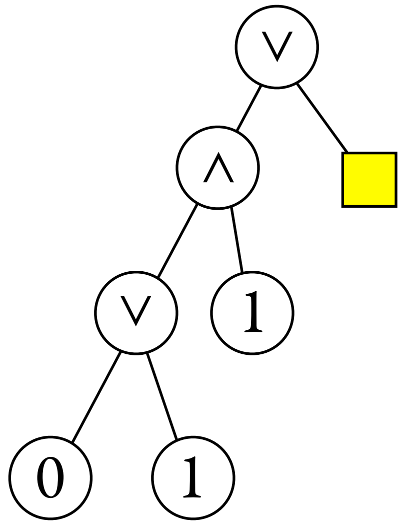


L = "Boolean expressions with value 1"

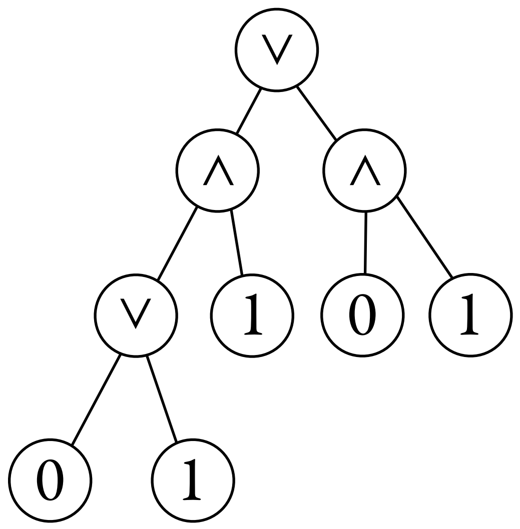
Fact. This language is aperiodic but not definable in FO($<$,suc_0,suc_1).



identity

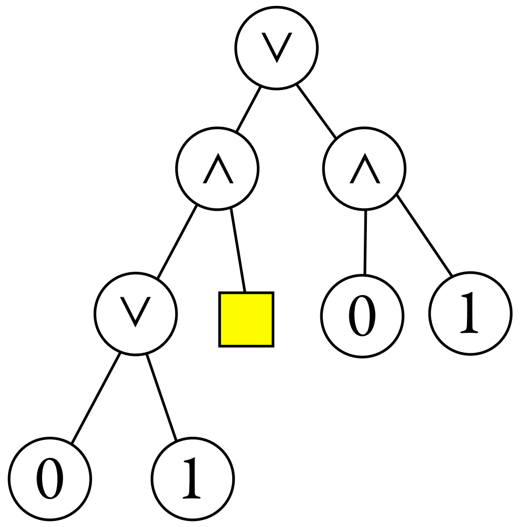


Boolean Expressions

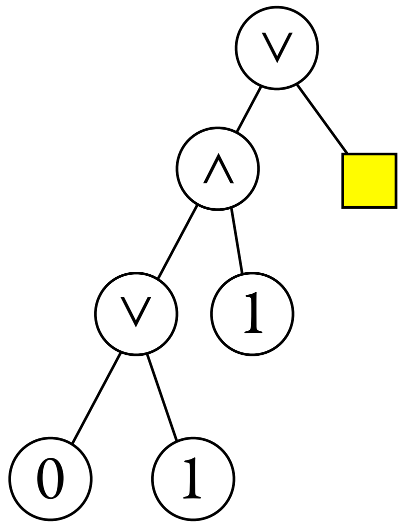


L = “Boolean expressions with value 1”

Fact. This language is aperiodic but not definable in FO($<$,suc_0,suc_1).

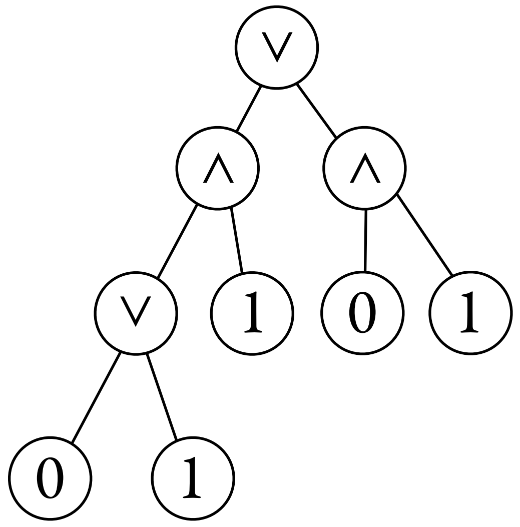


identity



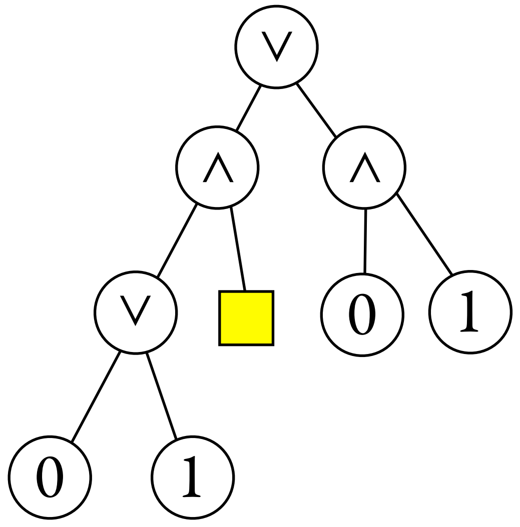
constant 1

Boolean Expressions

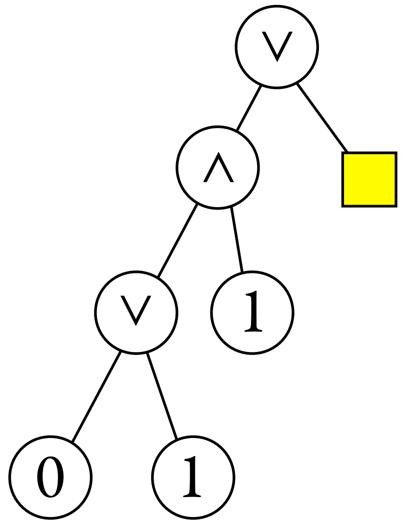


L = “Boolean expressions with value 1”

Fact. This language is aperiodic but not definable in FO($<$,suc_0,suc_1).



identity



constant 1

generally, monotone functions, which are an aperiodic set.

Monadic- and First-Order Logic for Words

- definition
- weakness of first-order logic
- MSO=regular

Monadic- and First-Order Logic for Trees

- definition
- problems with parity
- problems with aperiodicity

Transitive Closure Logic and Regular Expressions

Monadic- and First-Order Logic for Words

definition
weakness of first-order logic
MSO=regular

Monadic- and First-Order Logic for Trees

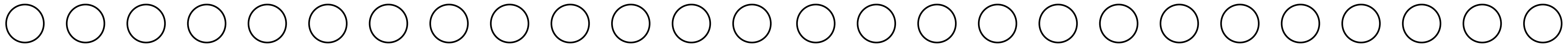
definition
problems with parity
problems with aperiodicity

Transitive Closure Logic and Regular Expressions

transitive closure logic for words...
...and for trees
regular expressions for trees

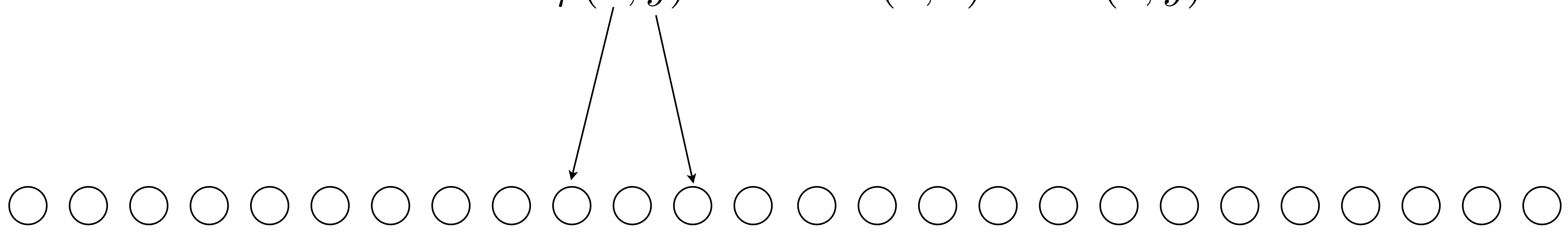
Transitive closure logic

Transitive closure logic



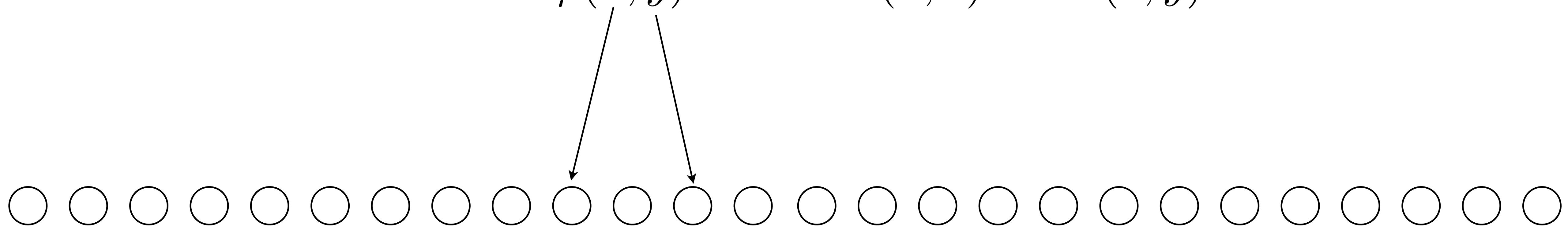
Transitive closure logic

$$\varphi(x, y) = \exists z\ suc(x, z) \wedge suc(z, y)$$



Transitive closure logic

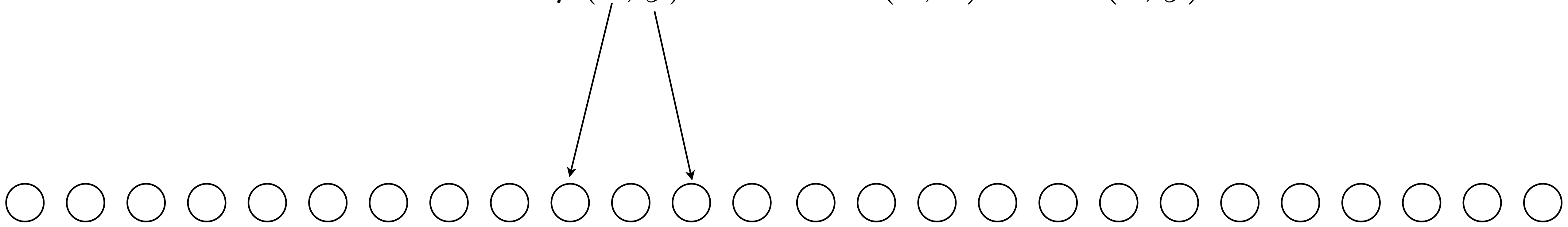
$$\varphi(x,y) = \exists z \; suc(x,z) \wedge suc(z,y)$$



$$\big(TC\varphi(x,y)\big)\;(x,y)$$

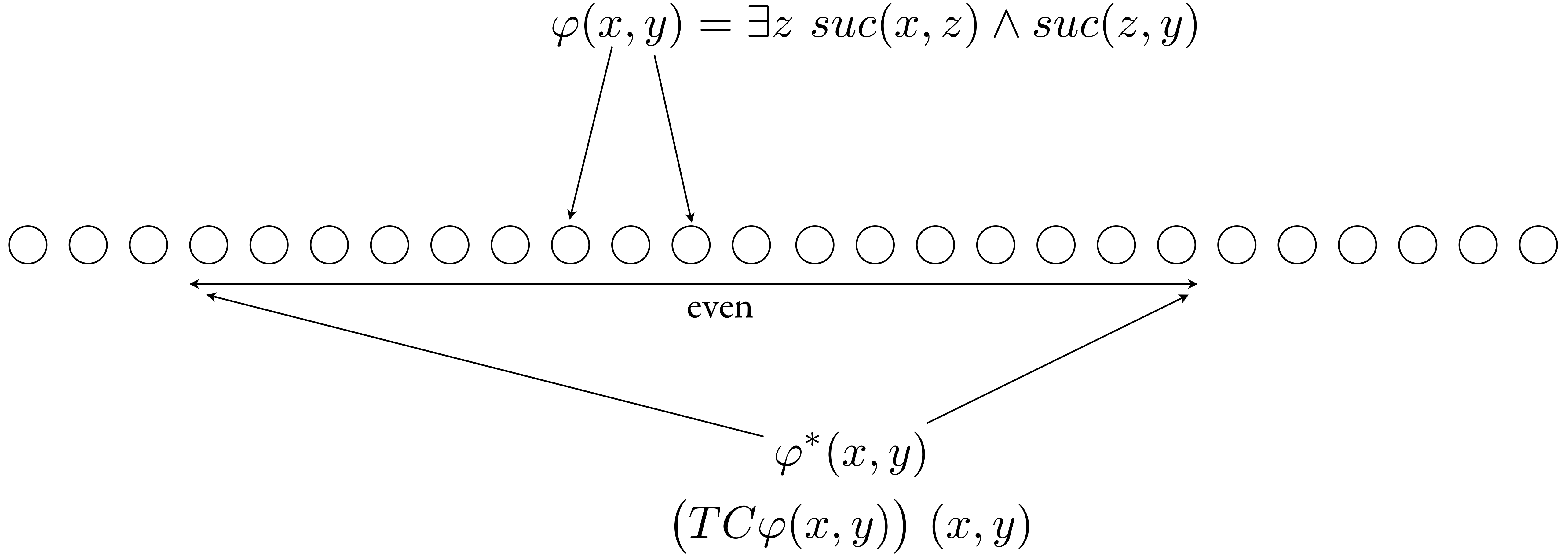
Transitive closure logic

$$\varphi(x, y) = \exists z \ suc(x, z) \wedge suc(z, y)$$



$$\varphi^*(x, y)$$
$$\big(TC\varphi(x, y)\big) \ (x, y)$$

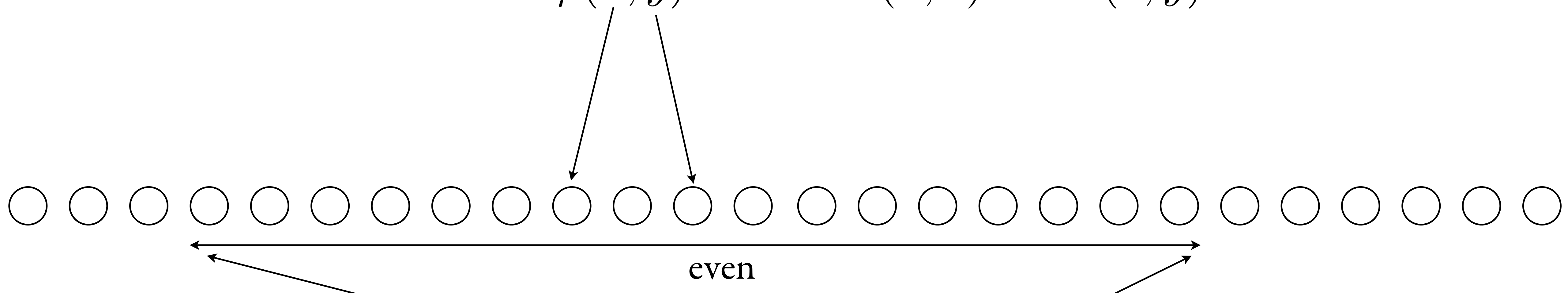
Transitive closure logic



$\varphi(x,y) = \exists z\ suc(x,z) \land suc(z,y)$
even
$\varphi^*(x,y)$
$\big(TC\varphi(x,y)\big)\ (x,y)$

Transitive closure logic

$$\varphi(x, y) = \exists z\ suc(x, z) \wedge suc(z, y)$$

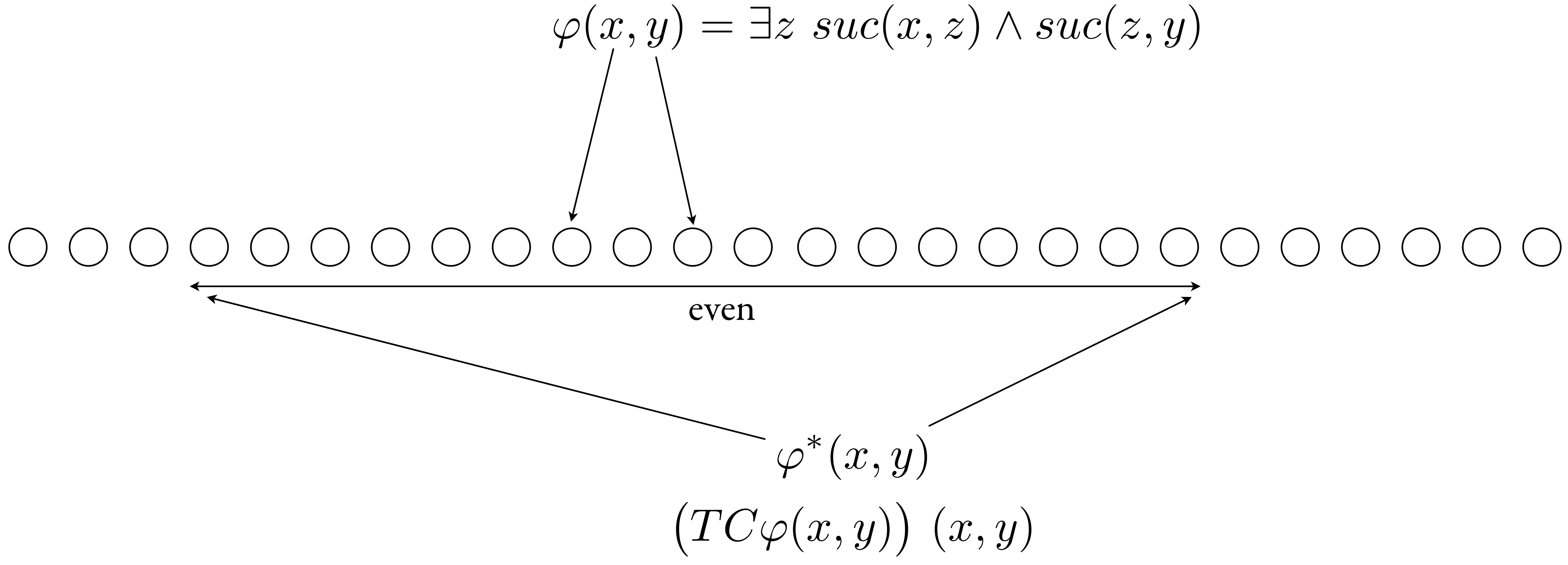


$$\varphi^*(x, y)$$

$$\big(TC\varphi(x, y)\big)\ (x, y)$$

Fact.
Transitive closure logic is a fragment of MSO.

Transitive closure logic



$\varphi(x,y) = \exists z\ suc(x,z) \wedge suc(z,y)$
even
$\varphi^*(x,y)$
$\big(TC\varphi(x,y)\big)\ (x,y)$

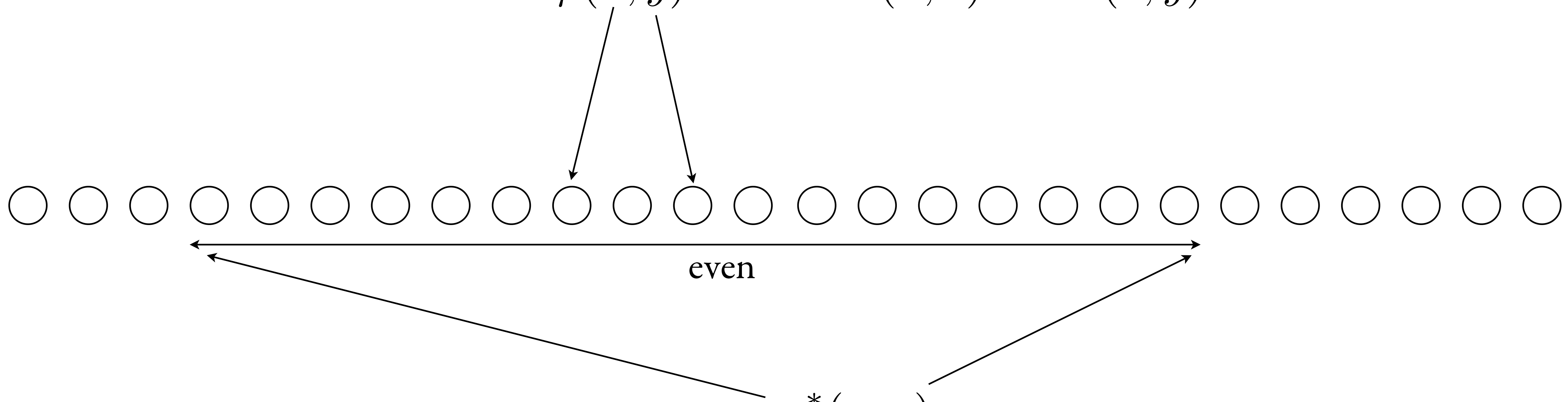
Fact.
Transitive closure logic is a fragment of MSO.

X is closed under φ

$$\varphi^*(x,y) \quad \text{iff} \quad \forall X \left\{ \begin{array}{c} \forall z_1 \forall z_2\ \ \big(\varphi(z_1,z_2)\ \wedge\ z_1 \in X\big)\ \Rightarrow\ z_2 \in X \\ \Downarrow \\ x \in X\ \Rightarrow y\ \in X \end{array} \right.$$

Transitive closure logic

$$\varphi(x, y) = \exists z \; suc(x, z) \wedge suc(z, y)$$

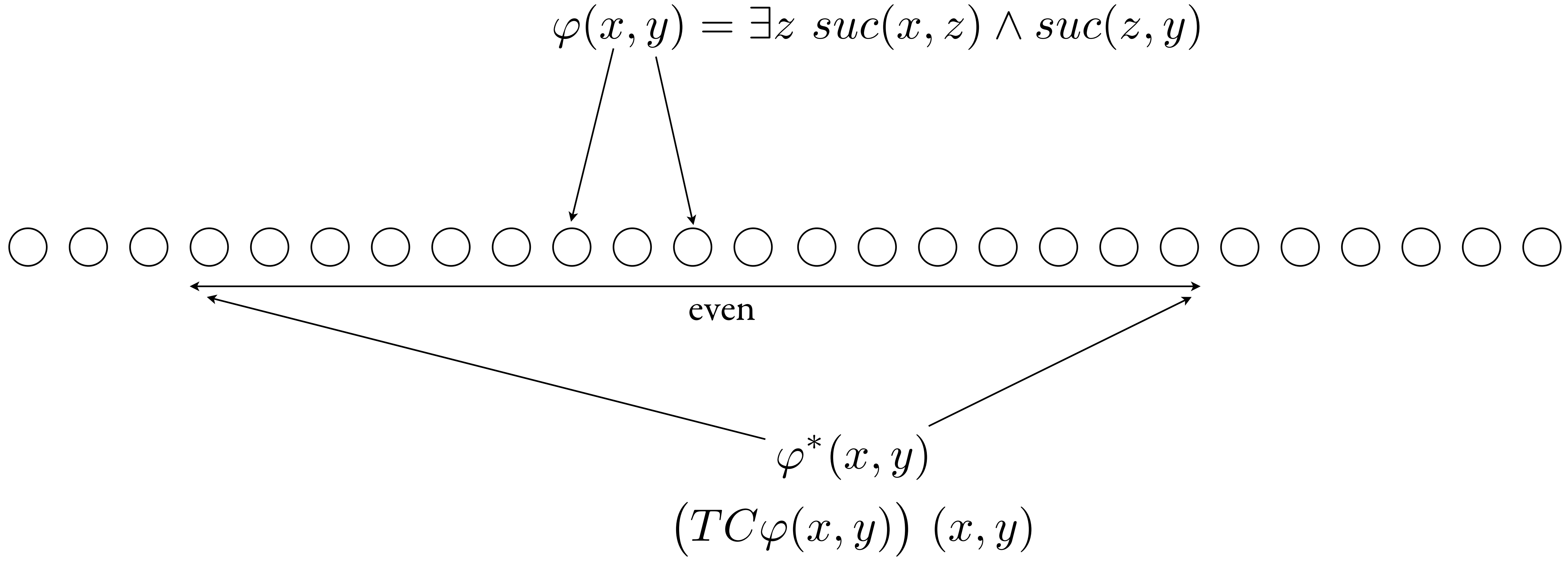


$$\varphi^*(x, y)$$

$$\big(TC\varphi(x, y)\big) \; (x, y)$$

Fact.
Transitive closure logic is a fragment of MSO.

Transitive closure logic


$\varphi(x, y) = \exists z\ suc(x, z) \wedge suc(z, y)$
even
$\varphi^*(x, y)$
$\big(TC\varphi(x, y)\big)\ (x, y)$

Fact.
Transitive closure logic is a fragment of MSO.

For every regular expression (on words), there is an equivalent formula of transitive closure logic. Hence, transitive closure logic = MSO for words.

Transitive closure logic

For trees, transitive closure logic is closely related to tree-walking pebble automata, and shares their weaknesses.

Thm. ten Cate, Segoufin `08
For trees, transitive closure logic is less expressive than MSO.

Meta-Corollary.
There is no nice regular expression syntax for regular tree languages.