# *Model Checking Lower Bounds for Simple Graphs*

Michael Lampis
KTH Royal Institute of Technology

March 11, 2013

# Algorithmic Meta-Theorems

Positive results

- Problem X is tractable.

Negative results

- Problem X is hard.

# Algorithmic Meta-Theorems

Positive results

- Problem X is tractable.

Negative results

- Problem X is hard.

- An algorithmic meta-theorem is a statement of the form:

  "All problems in a class C are tractable"

# Algorithmic Meta-Theorems

Positive results

- Problem X is tractable.

Negative results

- Problem X is hard.

- An algorithmic meta-theorem is a statement of the form:

  “All problems in a class C are tractable”

- Meta-theorems are great! (more in a second)

# Algorithmic Meta-Theorems

Positive results

- Problem X is tractable.

Negative results

- Problem X is hard.

- An algorithmic meta-theorem is a statement of the form:

  "All problems in a class C are tractable"

- Meta-theorems are great! (more in a second)

Main objective of today's talk: barriers to meta-theorems:

"There exists a problem in class C that is hard"

# Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

# Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?

# Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?
  - More graphs?
  - Wider classes of problems?
  - Faster?

# Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?

  - More graphs? ✓
  - Wider classes of problems?
  - Faster?

  Meta-theorems for clique-width, local treewidth,. . .

## Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?
  - More graphs? ✓
  - Wider classes of problems? ✓
  - Faster?

  This can be extended to optimization versions of MSO.

# Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?
  - More graphs? ✓
  - Wider classes of problems? ✓
  - Faster? ?

## Good news so far

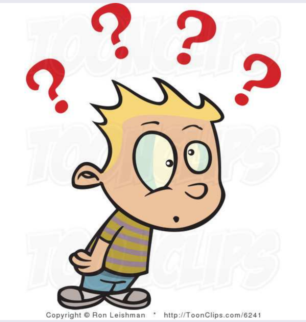
- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?

  - More graphs? ✓
  - Wider classes of problems? ✓
  - Faster? ?

  Faster than linear time?



## Good news so far

- Most famous meta-theorem: Courcelle's theorem

  All MSO-expressible properties are solvable in linear time on graphs of bounded treewidth.

- Can we do better?

  - More graphs? ✓
  - Wider classes of problems? ✓
  - Faster? ?

  Faster than linear time?

This is the main question we are concerned with today.

# Some bad news

- Courcelle's theorem:

  There exists an algorithm which, given an MSO formula $\phi$ and a graph $G$ with treewidth $w$ decides if $G \models \phi$ in time $f(w, \phi)|G|$.

# Some bad news

- Courcelle's theorem:

  There exists an algorithm which, given an MSO formula $\phi$ and a graph $G$ with treewidth $w$ decides if $G \models \phi$ in time $f(w, \phi)|G|$.

- But the function $f$ is a tower of exponentials!

# Some bad news

- Courcelle's theorem:

  There exists an algorithm which, given an MSO formula $\phi$ and a graph $G$ with treewidth $w$ decides if $G \models \phi$ in time $f(w, \phi)|G|$.

- But the function $f$ is a tower of exponentials!

- Unfortunately, this is not Courcelle's fault.

  Thm: If $G \models \phi$ can be decided in $f(w, \phi)|G|^c$ for elementary $f$ then P=NP. [Frick & Grohe '04]

## Some bad news

- Courcelle's theorem:

  There exists an algorithm which, given an MSO formula $\phi$ and a graph $G$ with treewidth $w$ decides if $G \models \phi$ in time $f(w, \phi)|G|$.

- But the function $f$ is a tower of exponentials!

- Unfortunately, this is not Courcelle's fault.

  Thm: If $G \models \phi$ can be decided in $f(w, \phi)|G|^c$ for elementary $f$ then P=NP. [Frick & Grohe '04]

- In fact, Frick and Grohe's lower bound applies to FO logic on trees!

# There is still hope

This is bad! Can we somehow escape the Frick and Grohe lower bound?

# There is still hope

This is bad! Can we somehow escape the Frick and Grohe lower bound?

## There is still hope

This is bad! Can we somehow escape the Frick and Grohe lower bound?

Recently, a series of meta-theorems that evade it give “better” parameter dependence.

- For vertex cover, neighborhood diversity, max-leaf [L. ’10]
- For twin cover [Ganian ’11]
- For shrub-depth [Ganian et al. ’12]
- For tree-depth [Gajarský and Hliňený ’12]

# There is still hope

This is bad! Can we somehow escape the Frick and Grohe lower bound?

Recently, a series of meta-theorems that evade it give “better” parameter dependence.

- For vertex cover, neighborhood diversity, max-leaf [L. ’10]
- For twin cover [Ganian ’11]
- For shrub-depth [Ganian et al. ’12]
- For tree-depth [Gajarský and Hliňený ’12]

Predominant idea: Removing isomorphic parts of the graph, when we have too many

# There is still hope

This is bad! Can we somehow escape the Frick and Grohe lower bound?

Recently, a series of meta-theorems that evade it give “better” parameter dependence.

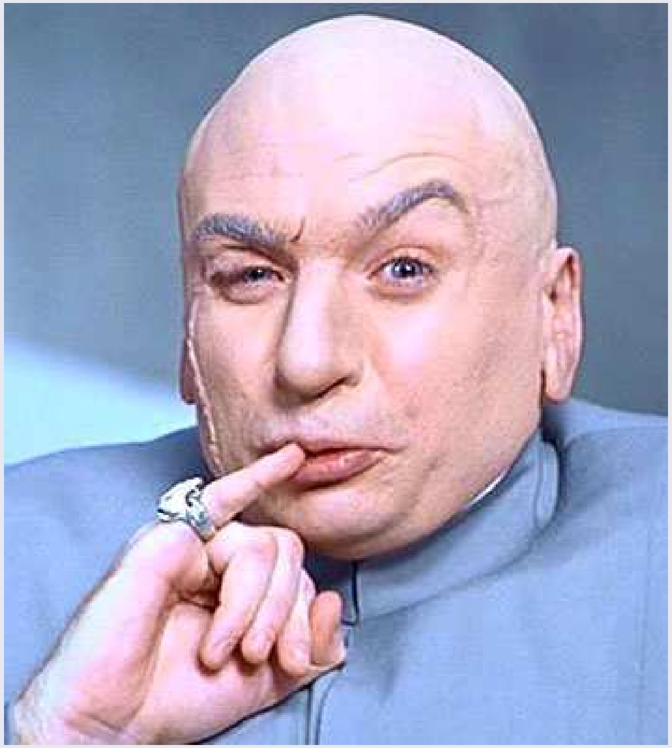
- For vertex cover, neighborhood diversity, max-leaf [L. ’10]
- For twin cover [Ganian ’11]
- For shrub-depth [Ganian et al. ’12]
- For tree-depth [Gajarský and Hliňený ’12]

Predominant idea: Removing isomorphic parts of the graph, when we have too many

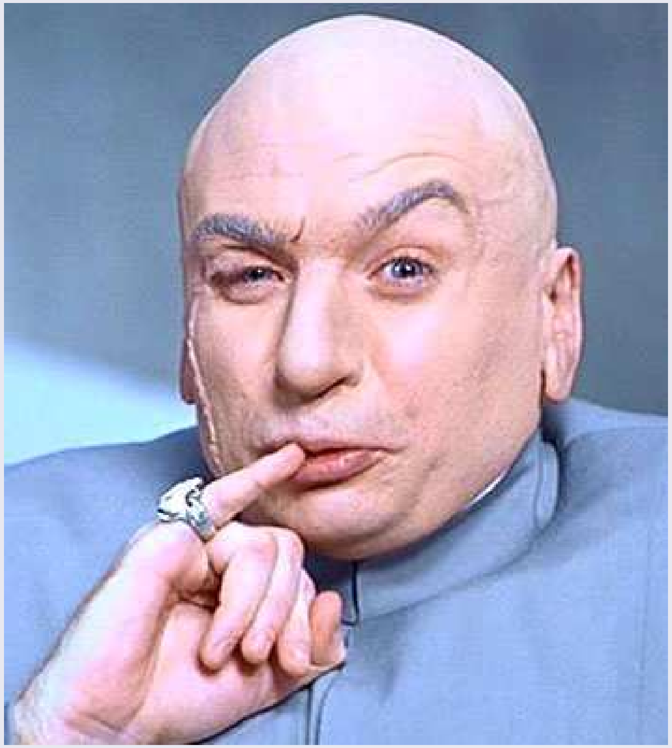
What’s next?

# Let's destroy all hope!

- In this talk the pendulum swings again.
- Main goal: prove hardness results even more devastating than Frick& Grohe.
- Motivation: If we know what we can't do, we might find things we can do.



# Let's destroy all hope!



- In this talk the pendulum swings again.
- Main goal: prove hardness results even more devastating than Frick& Grohe.
- Motivation: If we know what we can't do, we might find things we can do.

Today: Three new hardness results.

- Threshold graphs
- Paths
- Bounded-height trees

An appetizer:

Threshold Graphs

# More background

Theorem:

- $\mathrm{MSO}_1$ expressible properties can be decided in linear time on graphs of bounded clique-width [Courcelle, Makowsky, Rotics '00]

# More background

Theorem:

- $MSO_1$ expressible properties can be decided in linear time on graphs of bounded clique-width [Courcelle, Makowsky, Rotics '00]

A graph has clique-width $k$ if it can be constructed with the following operations using $\leq k$ labels

- Introduce a new vertex with label $i \in [k]$.
- Connect all vertices with label $i$ to all vertices with label $j$.
- Rename all vertices with label $i$ to label $j$.
- Take the disjoint union of two clique-width $k$ graphs.

# More background

Theorem:

- $\text{MSO}_1$ expressible properties can be decided in linear time on graphs of bounded clique-width [Courcelle, Makowsky, Rotics ’00]

An $\text{MSO}_1$ formula $\phi$ may contain:

- $\exists x, \forall x$ (quantifying over a graph’s vertices)
- $\exists X, \forall X$ (quantifying over a set of vertices)
- Relation $E(x, y)$ (edges), $x = y$
- Boolean connectives

# More background

Theorem:

- $\text{MSO}_1$ expressible properties can be decided in linear time on graphs of bounded clique-width [Courcelle, Makowsky, Rotics ’00]

- Trees have clique-width 3.

  Frick&Grohe $\rightarrow$ non-elementary dependence.

- Graphs with clique-width 1 are easy for $\text{MSO}_1$.

What about clique-width 2?

# Threshold Graphs

A graph is a threshold graph if it can be constructed with the following operations:

- Add a new vertex and connect it to everything.
- Add a new vertex and connect it to nothing.

# Threshold Graphs

A graph is a threshold graph if it can be constructed with the following operations:

- Add a new vertex and connect it to everything.
- Add a new vertex and connect it to nothing.

$u$

# Threshold Graphs

A graph is a threshold graph if it can be constructed with the following operations:

- Add a new vertex and connect it to everything.
- Add a new vertex and connect it to nothing.

$uj$

## Threshold Graphs

A graph is a threshold graph if it can be constructed with the following operations:

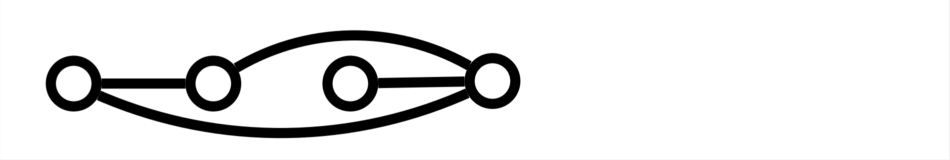
- Add a new vertex and connect it to everything.
- Add a new vertex and connect it to nothing.

$$uju$$

## Threshold Graphs

A graph is a threshold graph if it can be constructed with the following operations:

- Add a new vertex and connect it to everything.
- Add a new vertex and connect it to nothing.

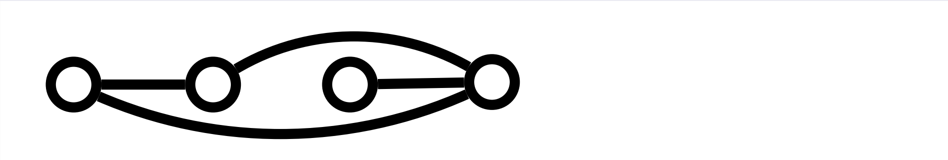


$ujuj$

## Threshold Graphs

A graph is a threshold graph if it can be constructed with the following operations:

- Add a new vertex and connect it to everything.
- Add a new vertex and connect it to nothing.



$ujuj$

Thm: Threshold graphs have clique-width 2.

## Hardness for threshold graphs

We use the following result of Frick& Grohe:

- There is no elementary-dependence model-checking algorithm for FO logic on binary strings.

## Hardness for threshold graphs

We use the following result of Frick& Grohe:

- There is no elementary-dependence model-checking algorithm for FO logic on binary strings.

Input:

- String $w$, FO formula $\phi$:
  - $\exists x$, $\forall x$ ($x$ will correspond to a character in the string)
  - Relation $\prec$ ($x \prec y$ if $x$ comes before $y$ in the string)
  - Relation $P_1(x)$ (the character $x$ is a 1)
  - Boolean connectives

## Hardness for threshold graphs

We use the following result of Frick& Grohe:

- There is no elementary-dependence model-checking algorithm for FO logic on binary strings.

Input:

- String $w$, FO formula $\phi$:
  - $\exists x$, $\forall x$ ($x$ will correspond to a character in the string)
  - Relation $\prec$ ($x \prec y$ if $x$ comes before $y$ in the string)
  - Relation $P_1(x)$ (the character $x$ is a 1)
  - Boolean connectives

Example:

$$\forall x P_1(x) \to \exists y \neg P_1(y) \wedge x \prec y$$

## Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w$ :
- $G$ : $uuj$

## Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w :$ 0
- $G :$ $uuj\ uj$

## Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w$ : 0 1
- $G$ : $uuj\ uj\ \ ujj$

## Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w$ : 0 1 1
- $G$ : $uuj\ uj\ \ ujj\ \ ujj$

## Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w$ : $\quad$ 0 $\quad$ 1 $\quad$ 1 $\quad$ 0...
- $G$ : $uuj\ uj\quad ujj\quad ujj\quad uj\ldots$

# Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w$ : 0 1 1 0...
- $G$ : $uuj\ uj\quad ujj\quad ujj\quad uj\ldots$

Idea: union vertices represent the characters

$$\begin{aligned} union(x) &:= \forall y \forall z (E(x,y) \wedge E(x,z) \wedge y \neq z) \rightarrow E(y,z) \\ main(x) &:= union(x) \wedge (\exists y \neg union(y) \wedge \neg E(x,y)) \end{aligned}$$

# Hardness for threshold graphs

Given a string $w$ we construct a threshold graph $G$

- $w$ : 0 1 1 0...
- $G$ : $uuj\ uj\ \ ujj\ \ ujj\ \ uj\ldots$

Idea: union vertices represent the characters

$$
\begin{aligned}
union(x) &:= \forall y \forall z (E(x,y) \wedge E(x,z) \wedge y \neq z) \rightarrow E(y,z) \\
main(x) &:= union(x) \wedge (\exists y \neg union(y) \wedge \neg E(x,y))
\end{aligned}
$$

This allows us to interpret $\exists x \psi(x)$ (in the string) to $\exists x(main(x) \wedge \psi^I(x))$ (in the graph).

## Hardness for threshold graphs

Interpretation continued:

- The $\prec$ relation can be expressed as

$$prec(x, y) := \exists z \neg union(z) \wedge E(x, z) \wedge \neg E(y, z)$$

# Hardness for threshold graphs

Interpretation continued:

- The $\prec$ relation can be expressed as

$$prec(x, y) := \exists z \neg union(z) \wedge E(x, z) \wedge \neg E(y, z)$$

- The $P_1$ relation can also be expressed in FO logic...

# Hardness for threshold graphs

Interpretation continued:

- The $\prec$ relation can be expressed as

$$prec(x, y) := \exists z \neg union(z) \wedge E(x, z) \wedge \neg E(y, z)$$

- The $P_1$ relation can also be expressed in FO logic...

Thm: There is no elementary-dependence model-checking algorithm for FO logic on threshold graphs.

# Consequences

Recall some of the “good” graph classes we know

- Some are closed under complement (neighborhood diversity, shrub-depth)
- Some are closed under union (tree-depth)

## Consequences

Recall some of the "good" graph classes we know

- Some are closed under complement (neighborhood diversity, shrub-depth)
- Some are closed under union (tree-depth)
- None are closed under both operations...

Any class of graph closed under both operations must* contain threshold graphs.

Main course:

Paths

## Why paths?

Main question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on paths?

# Why paths?

Main question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on paths?

Equivalent question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on unary strings?

# Why paths?

Main question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on paths?

Equivalent question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on unary strings?

Why?

- Do Frick and Grohe really need all trees?
- FO is easy on paths.
- MSO is hard on binary strings/colored paths.

## Why paths?

Main question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on paths?

Equivalent question:

- Is there an elementary-dependence algorithm for $\mathrm{MSO}_1$ on unary strings?

Why?

- Do Frick and Grohe really need all trees?
- FO is easy on paths.
- MSO is hard on binary strings/colored paths.
- MSO for max-leaf is open!

## Why would this be easy?

- MSO on paths = Regular language over unary alphabet
- FO is easy

## Why would this be easy?

- MSO on paths = Regular language over unary alphabet
- FO is easy
- Reduction seems impossible…

“Normal” reduction:

- Start with $n$-variable 3-SAT
- Construct graph $G$ with $|G| = n^c$
- Construct formula $\phi$ with $|\phi| = \log^* n$
- Prove YES instance $\leftrightarrow G \models \phi$

Problem: New instance would be encodable with $O(\log n)$ bits. We are making a sparse NP-hard language!

## How the reduction can work

Key idea: do not use P$\neq$NP but EXP$\neq$NEXP

- Motivation: reduction must construct exponential-size graph, so should be allowed exponential time.

## How the reduction can work

Key idea: do not use P$\neq$NP but EXP$\neq$NEXP

- Motivation: reduction must construct exponential-size graph, so should be allowed exponential time.

Plan:

- Start with an NEXP-complete problem and $n$ bits of input.
- Construct a path on $2^{n^c}$ vertices.
- Construct a formula $\phi$ with $|\phi| = \log^* n$.
- Prove YES instance $\leftrightarrow G \models \phi$.

Elementary parameter dependence gives EXP=NEXP.

## How the reduction can work

Key idea: do not use P$\neq$NP but EXP$\neq$NEXP

- Motivation: reduction must construct exponential-size graph, so should be allowed exponential time.

Plan:

- Start with an NEXP-complete problem and $n$ bits of input.
- Construct a path on $2^{n^c}$ vertices.
- Construct a formula $\phi$ with $|\phi| = \log^* n$.
- Prove YES instance $\leftrightarrow G \models \phi$.

Elementary parameter dependence gives EXP=NEXP.

- Formula will be somewhat larger, but still small enough.

# Counting with MSO

- The basic obstacle (as in Frick and Grohe) is counting efficiently.
- Given two sets of elements $S_1, S_2$ with $|S_1| \neq |S_2|$, what is the smallest MSO formula that can verify this?

# Counting with MSO

- The basic obstacle (as in Frick and Grohe) is counting efficiently.
- Given two sets of elements $S_1, S_2$ with $|S_1| \neq |S_2|$, what is the smallest MSO formula that can verify this?
- Example: For independent sets, $q$ quantifiers work for size $2^q$.

Main goal:

- Increase counting power exponentially with each added quantifier.
- Frick and Grohe do this, but they are allowed to design their graphs. We are (essentially) not!

## Counting with MSO

- The basic obstacle (as in Frick and Grohe) is counting efficiently.
- Given two sets of elements $S_1, S_2$ with $|S_1| \neq |S_2|$, what is the smallest MSO formula that can verify this?
- Example: For independent sets, $q$ quantifiers work for size $2^q$.

Main goal:

- Increase counting power exponentially with each added quantifier.
- Frick and Grohe do this, but they are allowed to design their graphs. We are (essentially) not!

Today: $q$ quantifiers count up to size $tow(\Omega(\log q))$ on unary strings.

# Learning to count

Induction:

- We have a MSO formula $eq_L(P_1, P_2)$ which correctly compares sets up to size $L$.
- The formula is only true for equal sets (independent of size).

Use this to compare larger sets economically.

First idea: division

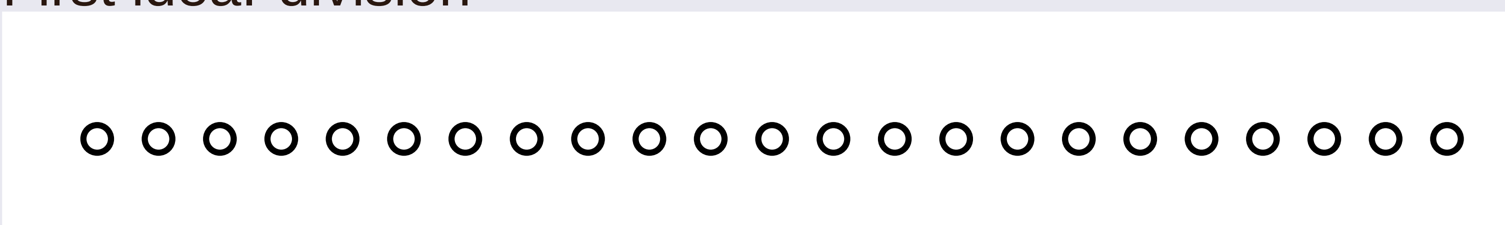
## Learning to count

Induction:

- We have a MSO formula $eq_L(P_1, P_2)$ which correctly compares sets up to size $L$.
- The formula is only true for equal sets (independent of size).

Use this to compare larger sets economically.

First idea: division



Given an ordered set of elements to compare with another, we first select a subset of it.

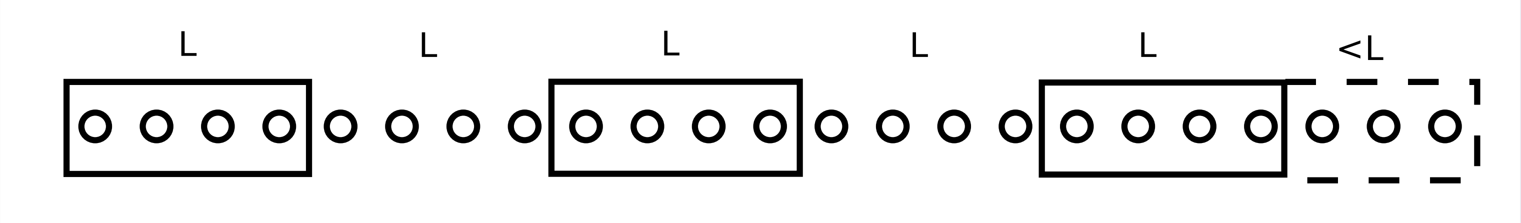
## Learning to count

Induction:

- We have a MSO formula $eq_L(P_1, P_2)$ which correctly compares sets up to size $L$.
- The formula is only true for equal sets (independent of size).

Use this to compare larger sets economically.

First idea: division




We can impose some structure: each “section” must have the same length ($\leq L$). We do this on both sets.

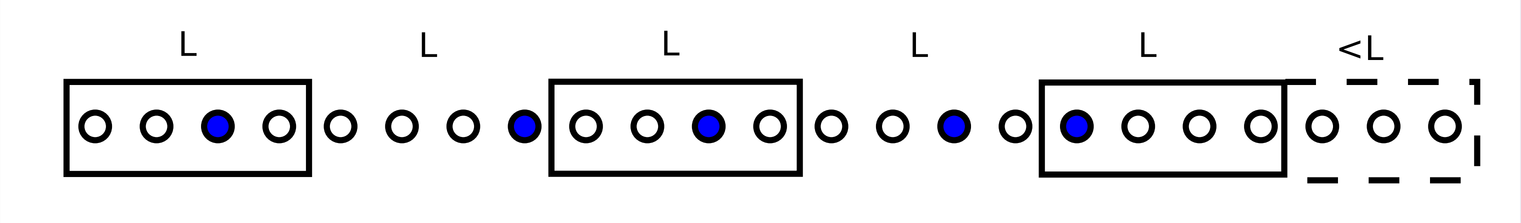
# Learning to count

Induction:

- We have a MSO formula $eq_L(P_1, P_2)$ which correctly compares sets up to size $L$.
- The formula is only true for equal sets (independent of size).

Use this to compare larger sets economically.

First idea: division




Now we need to count the number of sections. Select one representative from each. Compare the two sets of representatives.

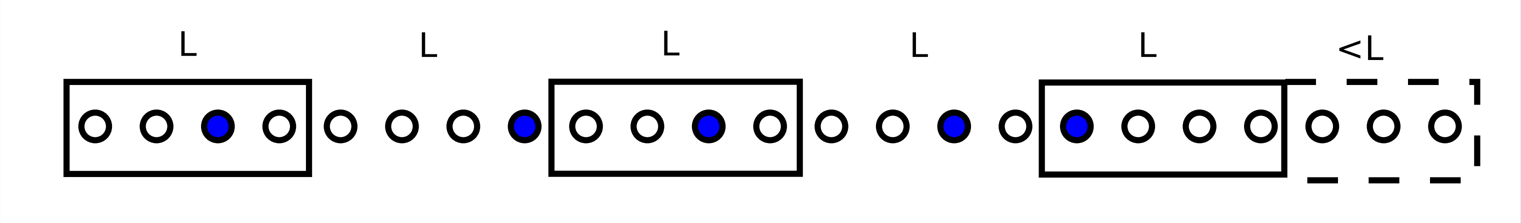
# Learning to count

Induction:

- We have a MSO formula $eq_L(P_1, P_2)$ which correctly compares sets up to size $L$.
- The formula is only true for equal sets (independent of size).

Use this to compare larger sets economically.

First idea: division




This allows us to go from $L$ to $L^2$ with $O(1)$ additional quantifiers (if done carefully).

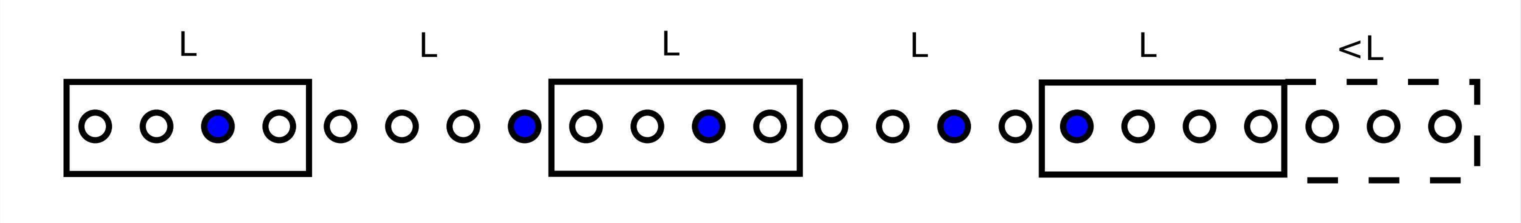
# Learning to count

Induction:

- We have a MSO formula $eq_L(P_1, P_2)$ which correctly compares sets up to size $L$.
- The formula is only true for equal sets (independent of size).

Use this to compare larger sets economically.

First idea: division




This allows us to go from $L$ to $L^2$ with $O(1)$ additional quantifiers (if done carefully).
Counting power: $2^{2^q}$. Not good enough, but we're moving.

# Learning to count better

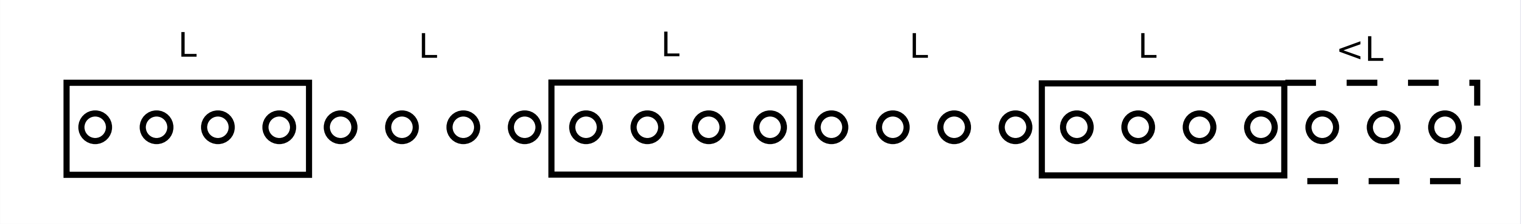
- Good: a single set gives many sections.
- Bad: hard to count how many sections we have. Using induction not good enough.

Idea: count in binary!

## Learning to count better

- Good: a single set gives many sections.
- Bad: hard to count how many sections we have. Using induction not good enough.

Idea: count in binary!

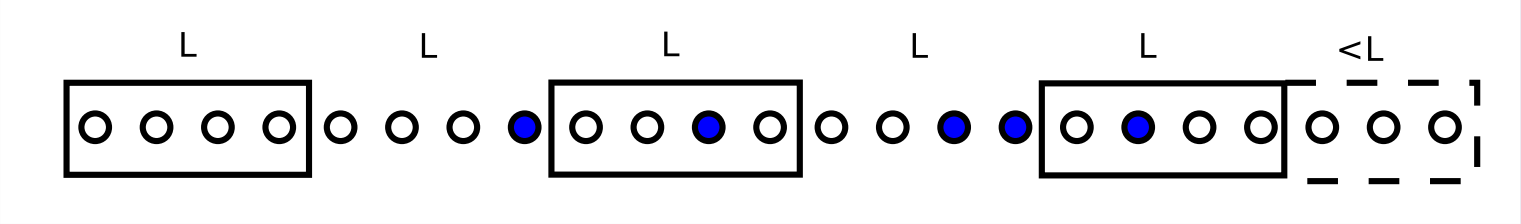



Select the same division into sections.

# Learning to count better

- Good: a single set gives many sections.
- Bad: hard to count how many sections we have. Using induction not good enough.

Idea: count in binary!

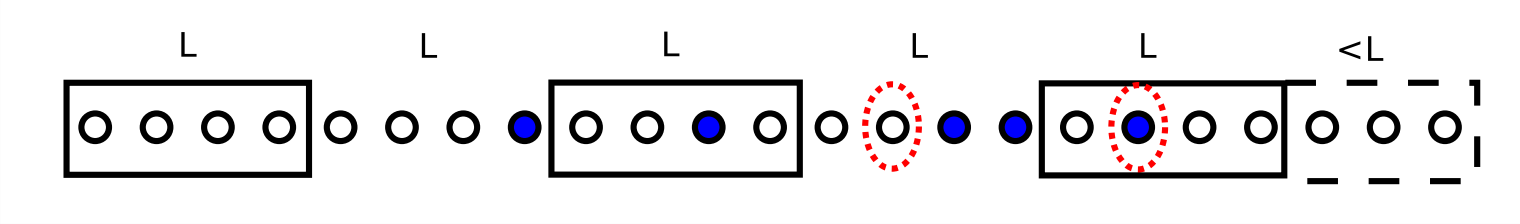



To count sections, select a subset that “writes” a binary number in each section.

## Learning to count better

- Good: a single set gives many sections.
- Bad: hard to count how many sections we have. Using induction not good enough.

Idea: count in binary!



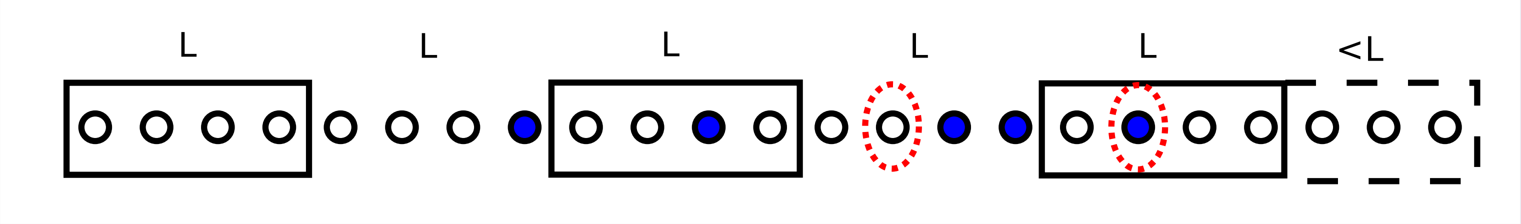

Demand that counting is correct for consecutive sections.

- Proof: hand-waving (but check the paper!)

## Learning to count better

- Good: a single set gives many sections.
- Bad: hard to count how many sections we have. Using induction not good enough.

Idea: count in binary!




We went from $L$ to $L2^L$ using $eq_L$ $O(1)$ times.
$\rightarrow$ each level of exponentiation increases size by a constant factor.
$\rightarrow$ can compare sets of size $n$ with $2^{\log^* n}$ quantifiers.

# Are we done with the math?

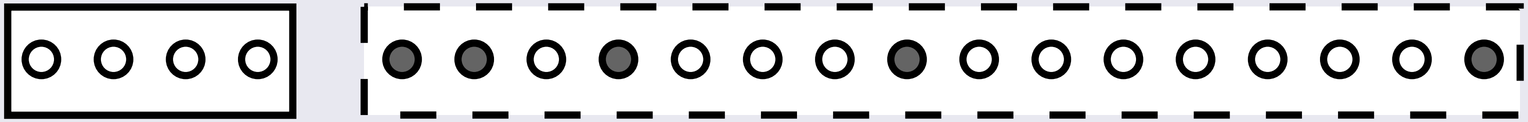
Using $eq_L$ it's easy to do comparisons, div, mod, ...

- We will also need exponentiation. $exp_L(P_1, P_2)$ is true if $|P_2| = 2^{|P_1|}$.

# Are we done with the math?

Using $eq_L$ it's easy to do comparisons, div, mod, ...

- We will also need exponentiation. $exp_L(P_1, P_2)$ is true if $|P_2| = 2^{|P_1|}$.



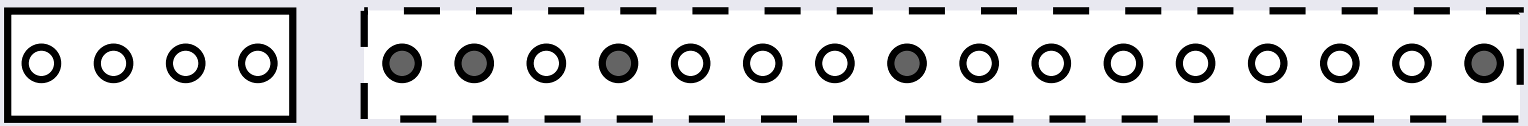
Idea: Find a set in $P_2$ with size $|P_1| + 1$. Ensure that consecutive distances are doubled.

DONE!

# Are we done with the math?

Using $eq_L$ it's easy to do comparisons, div, mod, ...

- We will also need exponentiation. $exp_L(P_1, P_2)$ is true if $|P_2| = 2^{|P_1|}$.



Idea: Find a set in $P_2$ with size $|P_1| + 1$. Ensure that consecutive distances are doubled.

DONE!

The hard part is over!

# Putting things together

- Reduction from NEXP Turing machine acceptance with $n$ input bits.
- Machine runs in $T = 2^{n^c}$ time. Input (read as binary number) is $I \leq 2^n$.
- Construct a path of length $T^2(2I + 1)$.
- Construct a $\phi$ that simulates the machine on the path.

# Putting things together

- Reduction from NEXP Turing machine acceptance with $n$ input bits.
- Machine runs in $T = 2^{n^c}$ time. Input (read as binary number) is $I \leq 2^n$.
- Construct a path of length $T^2(2I + 1)$.
- Construct a $\phi$ that simulates the machine on the path.

The last one is the tricky part. But we now have the right tools.

- Locate a set of length $T^2$. Divide it into sections of size $T$. These will represent snapshots of the machine's tape.
- Locate a set of length $I$. Use $exp$ to "read" input bits from it.
- Guess the contents of the tape.
- Check that the computation is correct and accepting.

## Consequences

Unless EXP=NEXP:

- Max-leaf is hard

# Consequences

Unless EXP=NEXP:

- Max-leaf is hard
- Graph classes closed under edge sub-divisions are hard

# Consequences

Unless EXP=NEXP:

- Max-leaf is hard
- Graph classes closed under edge sub-divisions are hard
- Graph classes closed under induced subgraphs with unbounded (dense)$^*$ diameter are hard

## Consequences

Unless EXP=NEXP:

- Max-leaf is hard
- Graph classes closed under edge sub-divisions are hard
- Graph classes closed under induced subgraphs with unbounded (dense)$^*$ diameter are hard
- MSO$_2$ for cliques is very hard! (not in XP)

The last one was already known. But "easier" proof using that $eq_L$ has constant size on cliques with MSO$_2$.

Dessert:

Trees of bounded height

## Why trees of bounded height?

This class of graphs is important for two recent meta-theorems:

- Shrub-depth in "When trees grow low: Shrubs and fast MSO$_1$" [Ganian et al. MFCS '12]
- Tree-depth in "Faster deciding MSO properties of trees of fixed height, and some consequences" [Gajarský and Hliněný FSTTCS '12]

In both cases the main tool is the following:

MSO model-checking for $q$ quantifiers on trees of height $h$ colored with $t$ colors can be done in $\exp^{(h+1)}(O(q(t+q)))$ time.

# Lower bound

Goal: prove that $h+1$ levels of exponentiation are exactly necessary.

- Start from an $n$-variable 3-SAT instance.
- Construct a tree of height $h$. Use $t = \log^{(h)}(n)$ colors.
- Construct a formula with $q = O(h)$ quantifiers.
- Prove equivalence between instances.

# Lower bound

Goal: prove that $h + 1$ levels of exponentiation are exactly necessary.

- Start from an $n$-variable 3-SAT instance.
- Construct a tree of height $h$. Use $t = \log^{(h)}(n)$ colors.
- Construct a formula with $q = O(h)$ quantifiers.
- Prove equivalence between instances.

Argument: an algorithm running in $\exp^{(h+1)}(o(t))$ would run in $2^{o(n)}$ here, disproving ETH.

# Let's count some more!

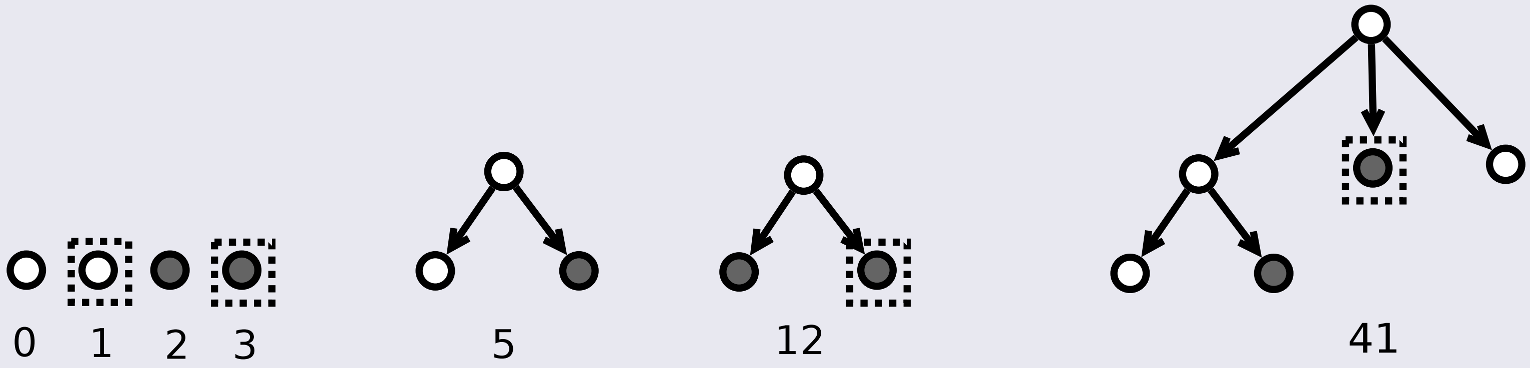
Fix $h$. The main problem is again to count efficiently.

- We have $\log^{(h)}(n)$ colors available. These can represent numbers up to $\log^{(h-1)}(n)$ with a single vertex (and comparisons are propositional!).

# Let's count some more!

Fix $h$. The main problem is again to count efficiently.

- We have $\log^{(h)}(n)$ colors available. These can represent numbers up to $\log^{(h-1)}(n)$ with a single vertex (and comparisons are propositional!).
- Assuming we can do numbers up to $L$ with trees of height $i$. We do numbers up to $2^L$ with trees of height $i+1$ (Frick& Grohe).



# The rest is easy

- Construct a tree of height $h - 1$ for each variable, encoding its index.
- Construct a tree of height $h - 1$ for each clauses, encoding the indices of its three literals.
- Add a root.
- Express satisfiability with a constant quantifier-depth formula.

Essential idea: we are using the proof of Frick and Grohe for $h$ levels.

## The rest is easy

- Construct a tree of height $h-1$ for each variable, encoding its index.
- Construct a tree of height $h-1$ for each clauses, encoding the indices of its three literals.
- Add a root.
- Express satisfiability with a constant quantifier-depth formula.

Essential idea: we are using the proof of Frick and Grohe for $h$ levels.

Thm: There is no $\exp^{(h+1)}(o(t))$ algorithm for MSO logic on $t$-colored trees of height $h$ unless the ETH is false.

# Conclusions - Open problems

- Three natural barriers to future improvements.
- Paths are probably the toughest to work around.

Future work

- (Uncolored) tree-depth?
- Height of tower for paths?

# Conclusions - Open problems

- Three natural barriers to future improvements.
- Paths are probably the toughest to work around.

Future work

- (Uncolored) tree-depth?
- Height of tower for paths?
- Other logics?!?

# Thank you!