MSO-checking

Fernando Sánchez Villaamil

Theoretical Computer Science

RWTH AACHEN

@NCSU 2014

As seen in previous lectures

Theorem

The languages over trees which are recognizable by DTA are precisely those languages that are definable in MSO_1 *logic.*

Corollary

Every graph property definable in monadic second-order logic can be decided in linear time for trees of bounded degree.

Proof.

1. Convert the given MSO_1 formula to a DTA $\mathcal{A}$ in constant time.
2. Run $\mathcal{A}$ on the given tree.

$\square$

As seen in previous lectures

Theorem

It is possible to convert EMSO*-formulas for labeled trees of bounded degree into automata that recognize the same language over trees.*

Today's menu

1. How hard MSO-checking is on graphs in general?
2. Easy solution for Maximum Independent Set on trees of bounded degree.
3. An unnecessarily complicated algorithm that solves Maximum Independent Set on trees of unbounded degree in linear time.
4. Show that what we did for Maximum Independent Set can be used to solve any MSO_1 formula on trees of unbounded degree.
5. Generalize the proof so that it works on any graph class of bounded treewidth.
6. We are gonna talk about the context of these results.

How hard is MSO_1-checking?

Definition ($\mathbf{PSPACE}$)

The complexity class $\mathbf{PSPACE}$ contains all problems which are solvable using polynomial space.

Theorem

$\mathbf{NP} \subseteq \mathbf{PSPACE}$.

Proof.

For a problem in $\mathbf{NP}$ it is possible to enumerate all its possible polynomially sized certificates and check them in polynomial time. $\square$

(It is assumed, but it has not yet been proven, that $\mathbf{NP} \subsetneq \mathbf{PSPACE}$)

How hard is MSO_1-checking?

Theorem

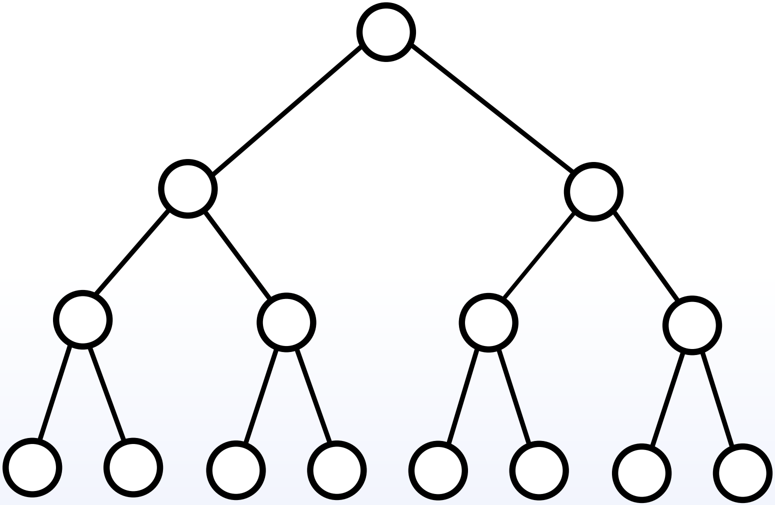
MSO_2*-checking is* $\mathbf{PSPACE}$*-complete.*

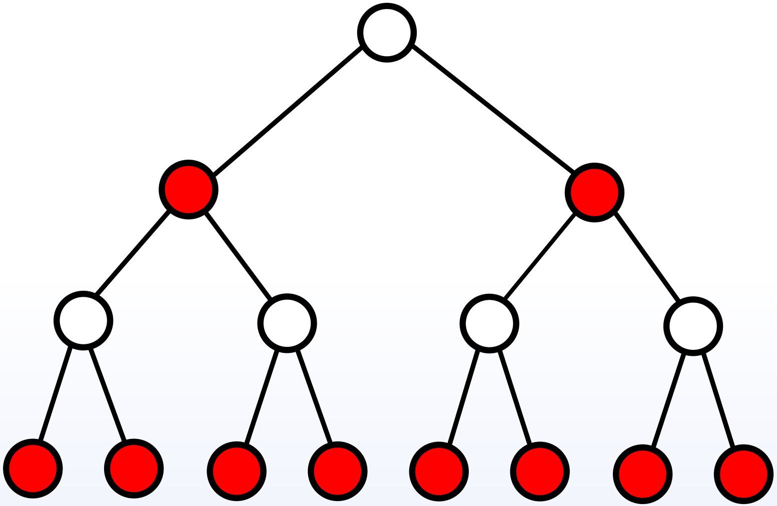
INDEPENDENT SET on trees

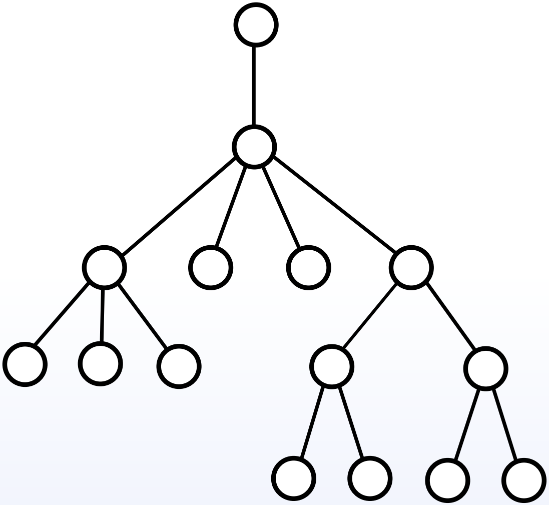
Definition (k-INDEPENDENT SET)

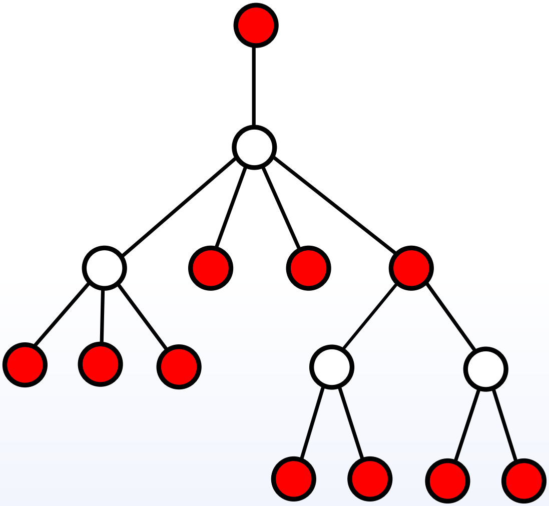
Given a graph G, does there exist a set S of vertices of size k such that the graph induced by S is edgeless.

Does there exists a linear time algorithm that solves MAXIMUM INDEPENDENT SET on trees of bounded degree?









INDEPENDENT SET on trees

For an ordered tree with maximum degree $d + 1$ we can write the following constant sized formula to check if there is an edge between two nodes.

$$\phi_{\textsf{edge}}(x, y) := S_1(x, y) \vee \ldots \vee S_d(x, y) \vee S_1(y, x) \vee \ldots \vee S_d(y, x)$$

We can now write a simple constant sized formula for MAXIMUM INDEPENDENT SET on trees of bounded degree.

$$\max S(\forall x \forall y((x \in S \wedge y \in S) \rightarrow (x = y \vee \neg \phi_{\textsf{edge}}(x, y))))$$

INDEPENDENT SET on trees

Theorem

A tree of bounded degree T has an independent set of size k iff there exists a set $S \subseteq G$ of size k such that

$$(G, S) \models \forall x, \forall y((x \in S \wedge y \in S) \rightarrow (x = y \vee \neg\phi_{\textit{edge}}(x, y))$$

INDEPENDENT SET on trees

Theorem

A tree of bounded degree T has an independent set of size k iff there exists a set $S \subseteq G$ of size k such that

$$(G, S) \models \forall x, \forall y((x \in S \wedge y \in S) \rightarrow (x = y \vee \neg\phi_{\textit{edge}}(x, y))$$

Corollary

It is possible to find an independent set of maximal size in linear time on trees of bounded degree.

Independent Set on trees of unbounded degree

Question

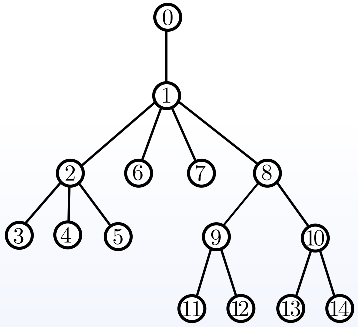
Does there exists a linear time algorithm that solves Maximal Independent Set on trees of **unbounded** degree?

INDEPENDENT SET on trees of unbounded degree

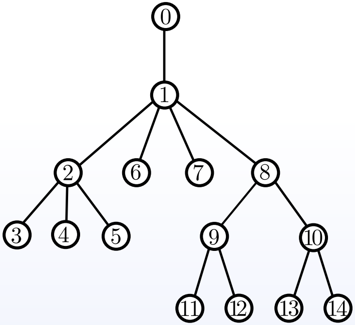
Idea

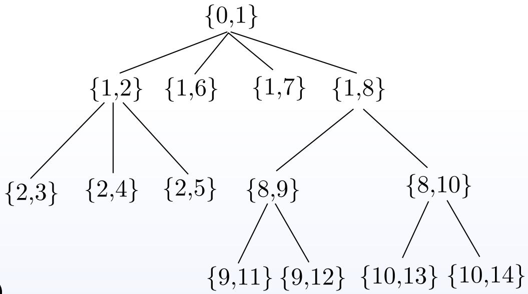
1. Convert the tree into a nice tree-decomposition of width one.
2. Adapt the MAXIMUM INDEPENDENT SET EMSO formula for trees to work on nice tree decompositions of width one.
3. Since a nice tree decomposition is a binary tree, we can now convert the new EMSO formula to a deterministic automata and solve the problem in linear time.

Finding a binary tree decompositions

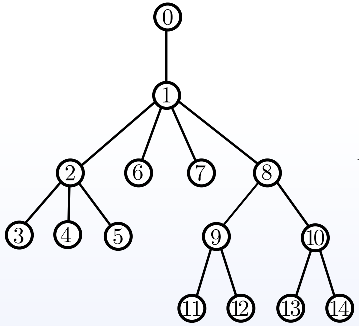


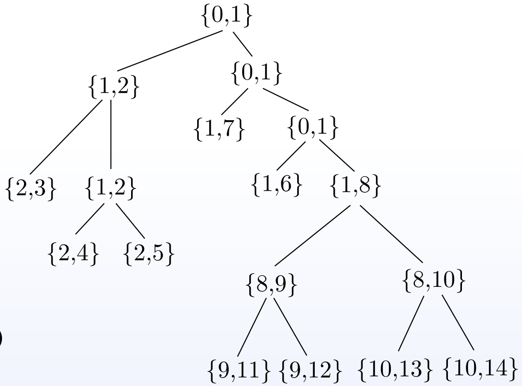
Finding a binary tree decompositions



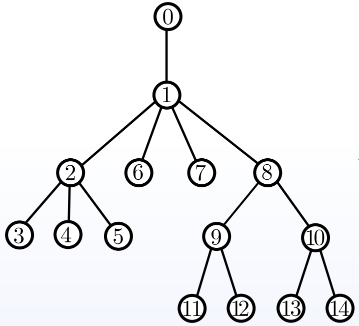


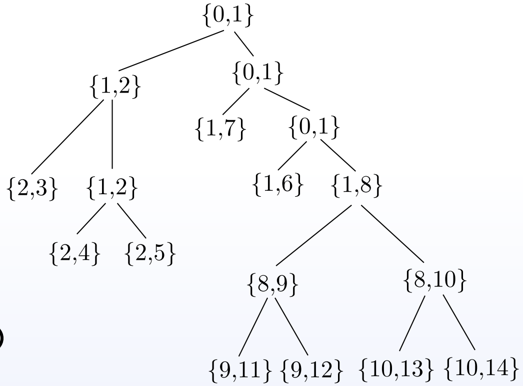
Finding a binary tree decompositions





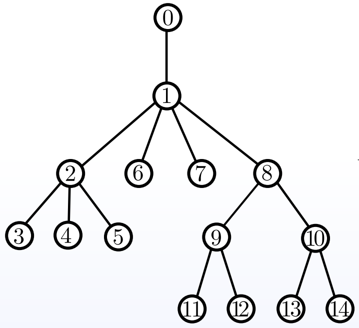
Finding a binary tree decompositions

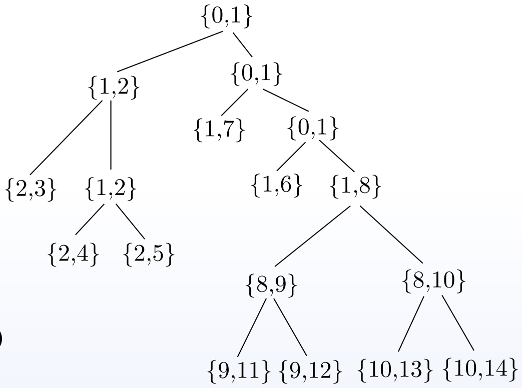




Can we treat the tree decomposition as a labeled graph?

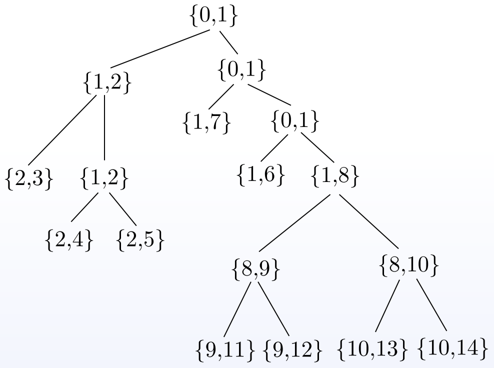
Finding a binary tree decompositions



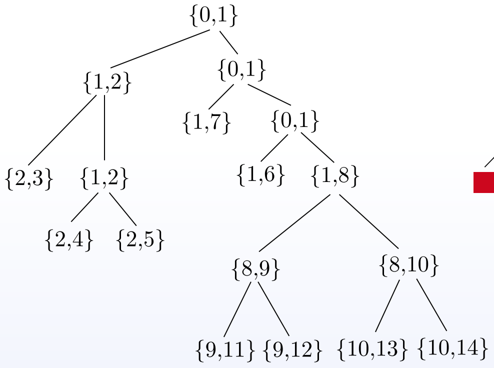


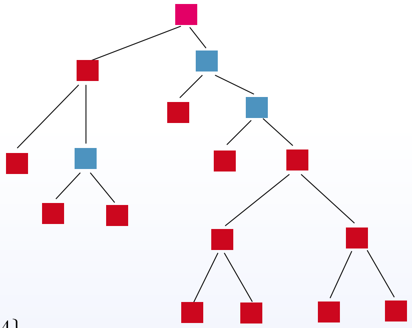
Not like this: The size of the labeling depends on the number of nodes of the original tree which is problematic.

Tree decomposition over finite alphabet

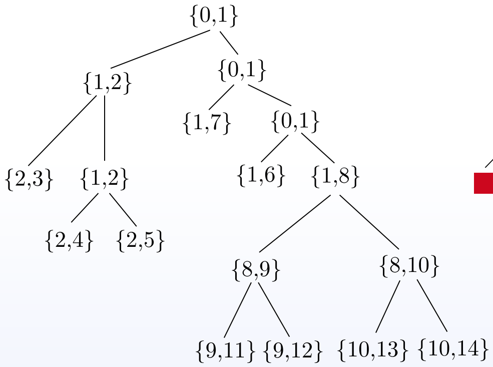


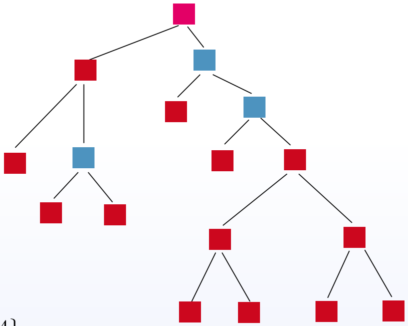
Tree decomposition over finite alphabet





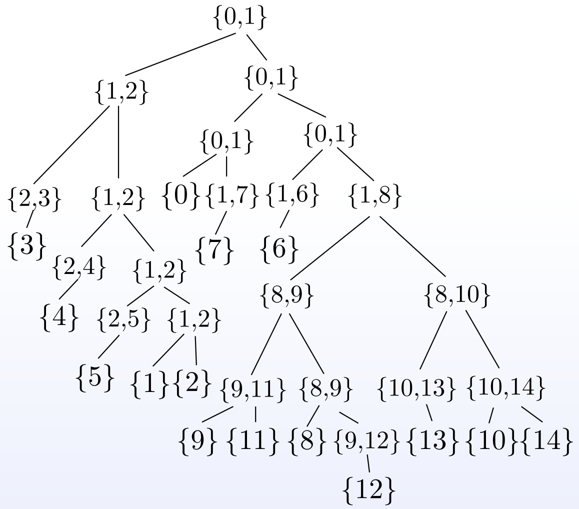
Tree decomposition over finite alphabet



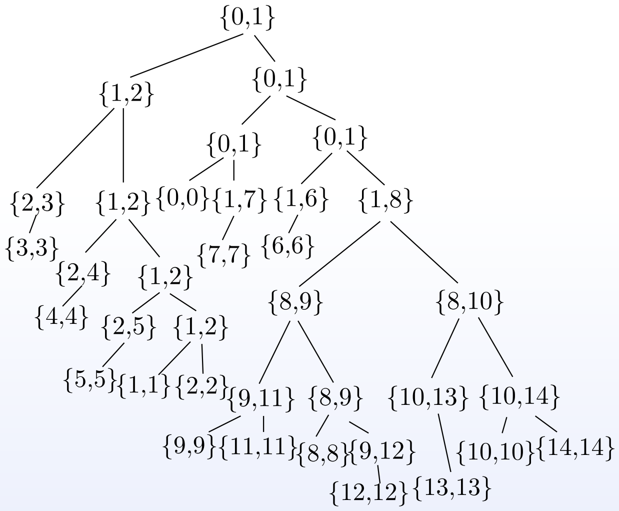


Can we introduce bags that represent nodes?

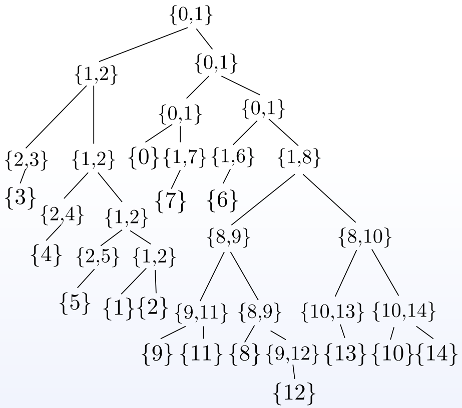
Tree decomposition over finite alphabet



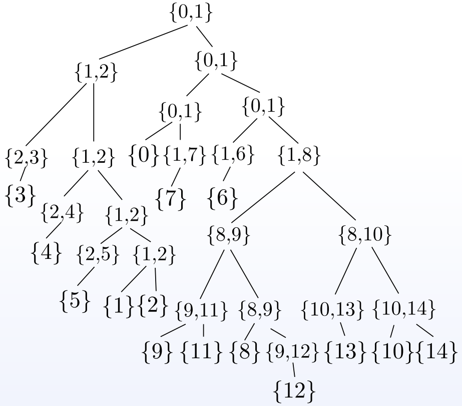
Tree decomposition over finite alphabet

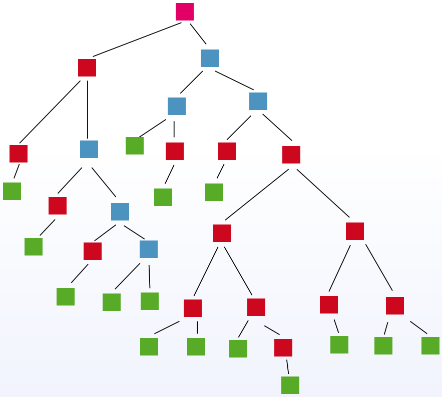


Tree decomposition over finite alphabet

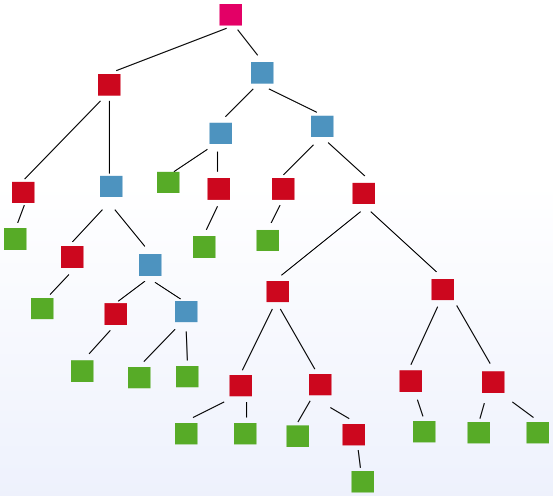


Tree decomposition over finite alphabet

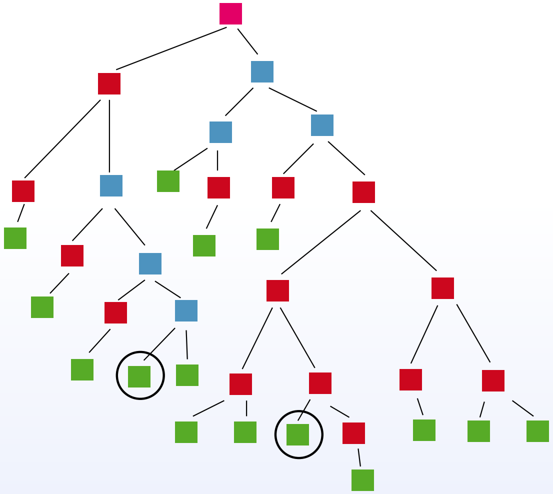




Tree decomposition over finite alphabet

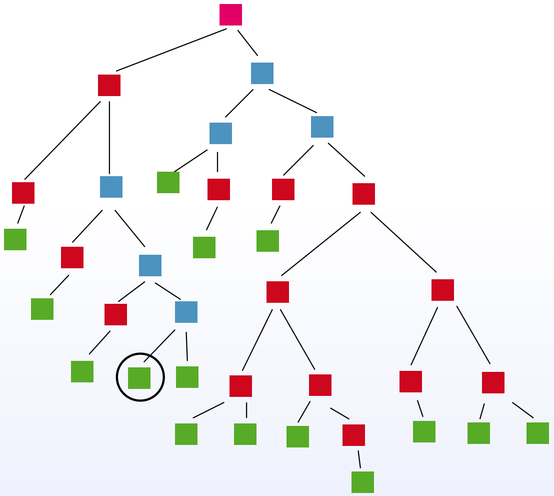


Tree decomposition over finite alphabet



Are the nodes represented by these leaves connected?

Tree decomposition over finite alphabet



Which bags contain the node represented by this leaf?

Label interpretation

- 🟦 The parent bag is exactly the same bag, order included.
- 🟥 The node on the left position appears in the parent bag at the right position.
- 🟩 The bag is a leaf representing a node from the original tree. It can have three further meanings:
    1. If it is a single child, then the node it represents is in the right position in the parent.
    2. If it is not a single child and it is a left child, then it is contained in the left position in the parent.
    3. If it is not a single child and it is a right child, then it is contained in the right position in the parent.

Overlap

Let us write formulas $\phi_{i,j}(s,t)$ for $0 \leq i,j \leq 1$ that are true if s is a parent of t in the tree decomposition and the positions j in s and i in t are the same.

Overlap

Let us write formulas $\phi_{i,j}(s,t)$ for $0 \leq i, j \leq 1$ that are true if s is a parent of t in the tree decomposition and the positions j in s and i in t are the same.

$$\begin{aligned}\phi_{0,0}(s,t) :=& (S_0(s,t) \wedge (\text{🟩}(t) \vee \text{🟦}(t))) \\ & \vee (S_1(s,t) \wedge \text{🟦}(t))\end{aligned}$$

Overlap

Let us write formulas $\phi_{i,j}(s,t)$ for $0 \leq i, j \leq 1$ that are true if s is a parent of t in the tree decomposition and the positions j in s and i in t are the same.

$$\begin{aligned}\phi_{0,0}(s,t) :=& (S_0(s,t) \wedge (\text{🟩}(t) \vee \text{🟦}(t))) \\ & \vee (S_1(s,t) \wedge \text{🟦}(t))\end{aligned}$$

$$\begin{aligned}\phi_{0,1}(s,t) :=& (S_0(s,t) \wedge \text{🟩}(t) \wedge \exists y(S_1(s,y))) \\ & \vee (S_1(s,t) \wedge \text{🟩}(t)) \\ & \vee ((S_0(s,t) \vee S_1(s,t)) \wedge \text{🟥}(t))\end{aligned}$$

Overlap

Let us write formulas $\phi_{i,j}(s,t)$ for $0 \leq i,j \leq 1$ that are true if s is a parent of t in the tree decomposition and the positions j in s and i in t are the same.

$$\begin{aligned}\phi_{0,0}(s,t) :=& (S_0(s,t) \wedge (\text{🟩}(t) \vee \text{🟦}(t))) \\ & \vee (S_1(s,t) \wedge \text{🟦}(t))\end{aligned}$$

$$\begin{aligned}\phi_{0,1}(s,t) :=& (S_0(s,t) \wedge \text{🟩}(t) \wedge \exists y(S_1(s,y))) \\ & \vee (S_1(s,t) \wedge \text{🟩}(t)) \\ & \vee ((S_0(s,t) \vee S_1(s,t)) \wedge \text{🟥}(t))\end{aligned}$$

$$\phi_{1,1}(s,t) := (S_0(s,t) \vee S_1(s,t)) \wedge \text{🟦}(t)$$

All the bags that contain a node

Assume that for any node u the set X_0 contains all bags that contain u at the left position and the set X_1 contains all bags that contain u at the right position. Let us write a formula $\phi_{\mathsf{conn}}(X_0, X_1)$ which is true on these sets, but false for all sets $X_0', X_1' \subsetneq X_0, X_1$.

All the bags that contain a node

Assume that for any node u the set X_0 contains all bags that contain u at the left position and the set X_1 contains all bags that contain u at the right position. Let us write a formula $\phi_{\text{conn}}(X_0, X_1)$ which is true on these sets, but false for all sets $X_0', X_1' \subsetneq X_0, X_1$.

$$\begin{aligned}\phi_{\text{conn}}(X_0, X_1) :={}& X_0 \neq \varnothing \wedge X_1 \neq \varnothing \wedge \\ & \forall s \forall t (((t \in X_0 \rightarrow (\phi_{0,0}(s,t) \rightarrow s \in X_0) \\ & \vee (\phi_{0,0}(t,s) \rightarrow s \in X_0) \\ & \vee (\phi_{0,1}(s,t) \rightarrow s \in X_1)) \\ & \wedge (t \in X_1 \rightarrow (\phi_{1,1}(s,t) \rightarrow s \in X_1) \\ & \vee (\phi_{1,1}(t,s) \rightarrow s \in X_1))))\end{aligned}$$

If X_0 contains all the bags that contain a node u at position zero and X_1 contains all the bags that contain u at position one then $\phi_{\mathsf{conn}}(X_0, X_1)$ is true.

If X_0 contains all the bags that contain a node u at position zero and X_1 contains all the bags that contain u at position one then $\phi_{\mathsf{conn}}(X_0, X_1)$ is true.

Problem: If X_0 and X_1 contain all bags, then $\phi_{\mathsf{conn}}(X_0, X_1)$ is also true.

If X_0 contains all the bags that contain a node u at position zero and X_1 contains all the bags that contain u at position one then $\phi_{\mathsf{conn}}(X_0, X_1)$ is true.

Problem: If X_0 and X_1 contain all bags, then $\phi_{\mathsf{conn}}(X_0, X_1)$ is also true.

Solution: Force exclusion minimality.

$$\begin{aligned}\phi_{\mathsf{exc}}(X_0, X_1) :=& \phi_{\mathsf{conn}}(X_0, X_1) \wedge \\ & \forall X_0' \forall X_1' ((X_0' \subsetneq X_0 \vee X_1' \subsetneq X_1) \\ & \qquad \wedge (X_0' \subseteq X_0 \wedge X_1' \subseteq X_1) \\ & \qquad \rightarrow \neg \phi_{\mathsf{conn}}(X_0', X_1'))\end{aligned}$$

If X_0 contains all the bags that contain a node u at position zero and X_1 contains all the bags that contain u at position one then $\phi_{\textsf{conn}}(X_0, X_1)$ is true.

Problem: If X_0 and X_1 contain all bags, then $\phi_{\textsf{conn}}(X_0, X_1)$ is also true.

Solution: Force exclusion minimality.

$$\begin{aligned}\phi_{\textsf{exc}}(X_0, X_1) :=& \phi_{\textsf{conn}}(X_0, X_1) \wedge \\ & \forall X_0' \forall X_1' ((X_0' \subsetneq X_0 \vee X_1' \subsetneq X_1) \\ & \qquad \wedge (X_0' \subseteq X_0 \wedge X_1' \subseteq X_1) \\ & \qquad \rightarrow \neg \phi_{\textsf{conn}}(X_0', X_1'))\end{aligned}$$

Finally $\phi_{\textsf{exc}}(X_0, X_1)$ is true iff there exists a node u such that the set of bags that contain u is precisely $X_0 \cup X_1$.

Interpretation

MSO-formula for trees of bounded degree.

$$\phi_{\textsf{edge}}(x,y) := S_1(x,y) \vee \ldots \vee S_d(x,y) \vee S_1(y,x) \vee \ldots \vee S_d(y,x)$$

$$\max S(\forall x \forall y((x \in S \wedge y \in S) \rightarrow (x = y \vee \neg \phi_{\textsf{edge}}(x,y))))$$

Interpretation

We can now rewrite the formula for edges with the help of ϕ_{exc}:

$$\begin{aligned}\phi_{\mathsf{edge}}(x,y) :=\ & x \neq y \wedge \exists X_0 \exists X_1 \exists Y_0 \exists Y_1 (x \in X_0 \wedge y \in Y_0 \\ & \wedge \phi_{\mathsf{exc}}(X_0, X_1) \wedge \phi_{\mathsf{exc}}(Y_0, Y_1) \\ & \wedge \exists z((z \in X_0 \vee z \in X_1) \wedge (z \in Y_0 \vee z \in Y_1)))\end{aligned}$$

And finally rewrite the formula for INDEPENDENT SET on trees.

$$\begin{gathered}\max S(\forall x \forall y(\text{🟩}(x) \wedge \text{🟩}(y) \wedge (x \in S \wedge y \in S) \\ \rightarrow (x = y \vee \neg \phi_{\mathsf{edge}}(x,y))))\end{gathered}$$

Linear algorithm for INDEPENDENT SET on trees

1. Compute a tree decomposition $\mathcal{T}$ of the input tree T with the following properties:
   - $\mathcal{T}$ has width one.
   - $\mathcal{T}$ is a binary tree.
   - Every node of T appears as a leaf bag in $\mathcal{T}$.

   This is doable in linear time.
2. Compute a label tree T' that represent $\mathcal{T}$.
3. Since the formula we created for INDEPENDENT SET on tree decompositions of width one has constant size, we can compute a DTA for it.
4. Run the DTA on T' and repeat the output.

Since there exists a rather simple dynamic programming algorithm to compute independent set on trees in linear time our proof via tree-automata has been more of an exercise in mental resilience than anything else…unless we can use these techniques to prove something more general!

MSO_1-checking on trees of unbounded degree

We can use a similar algorithm to solve all problems expressible with a MSO_1 formula ψ of constant size on trees in linear time.

1. Compute a tree decomposition $\mathcal{T}$ of the input tree T with the following properties:
   - $\mathcal{T}$ has width one.
   - $\mathcal{T}$ is a binary tree.
   - Every node of T appears as a leaf bag in $\mathcal{T}$.

   This is doable in linear time.
2. Compute a label tree T' that represent $\mathcal{T}$.
3. Translate ψ into a formula ψ' on tree decompositions, by forcing all variables to be leaves in the tree decomposition and replacing the edge predicate by our MSO_1-formula for the edge relation.
4. We can compute a DTA for ψ' in constant time.
5. Run the DTA on T' and repeat the output.

Courcelle's Theorem

Question

Can we do something similar for any constant treewidth w?

Courcelle's Theorem

Question

Can we do something similar for any constant treewidth w?

Theorem (Courcelle's Theorem)

Every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth.

Courcelle’s Theorem

Question

Can we do something similar for any constant treewidth w?

Theorem (Courcelle’s Theorem)

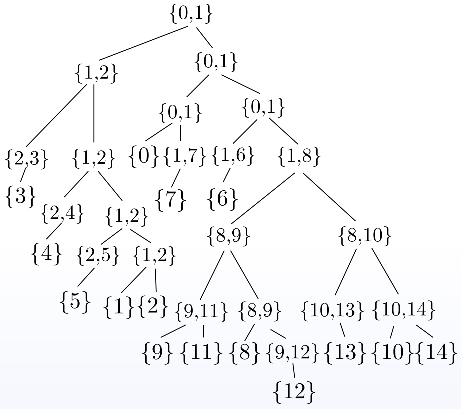
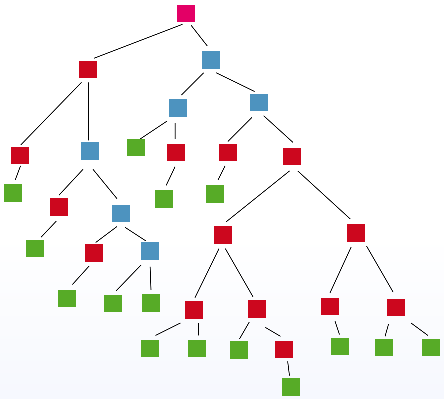
Every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth.

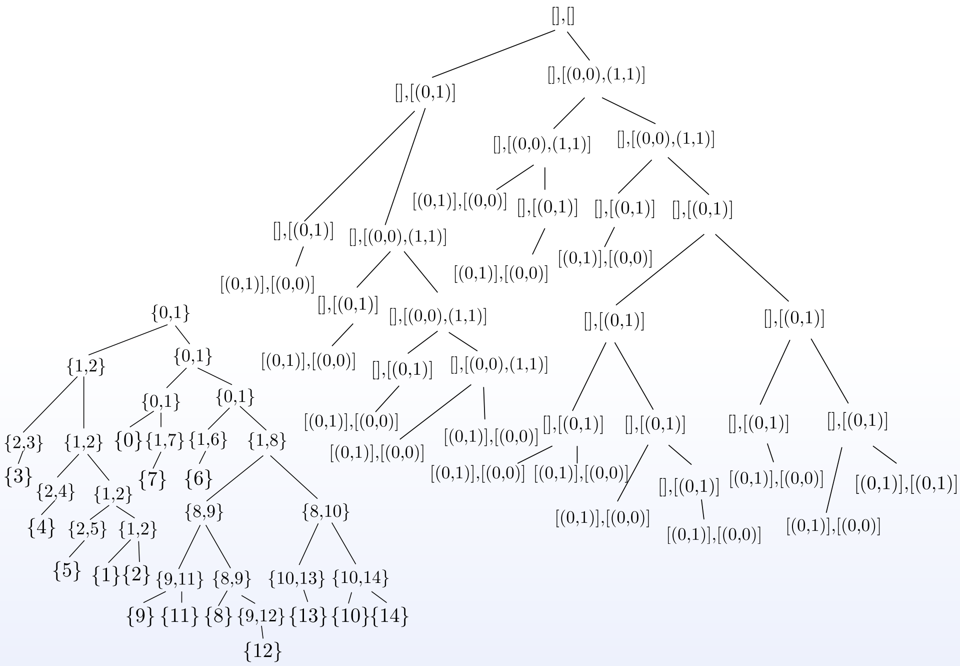
> *[This] result was first proved by Bruno Courcelle in 1990 and is considered the archetype of algorithmic meta-theorems.*
>
> —*Wikipedia*

Bruno Courcelle

Labeled trees for general tree decompositions

Assume you are given a tree decomposition of width w, how can we convert it into a tree with a finite labeling (where the size can only depend on w) that is usable?


{0,1}
{1,2}
{0,1}
{0,1}
{0,1}
{2,3}
{1,2}
{0}
{1,7}
{1,6}
{1,8}
{3}
{2,4}
{1,2}
{7}
{6}
{4}
{2,5}
{1,2}
{8,9}
{8,10}
{5}
{1}
{2}
{9,11}
{8,9}
{10,13}
{10,14}
{9}
{11}
{8}
{9,12}
{13}
{10}
{14}
{12}




[],[]
[],[(0,1)]
[],[(0,0),(1,1)]
[],[(0,0),(1,1)]
[],[(0,0),(1,1)]
[(0,1)],[(0,0)]
[],[(0,1)]
[],[(0,1)]
[],[(0,1)]
[],[(0,1)]
[(0,1)],[(0,0)]
[(0,1)],[(0,0)]
[],[(0,0),(1,1)]
[(0,1)],[(0,0)]
[],[(0,1)]
[],[(0,0),(1,1)]
[(0,1)],[(0,0)]
[],[(0,1)]
[],[(0,0),(1,1)]
[(0,1)],[(0,0)]
[(0,1)],[(0,0)]
[(0,1)],[(0,0)]
[],[(0,1)]
[],[(0,1)]
[],[(0,1)]
[],[(0,1)]
[],[(0,1)]
[],[(0,1)]
[(0,1)],[(0,0)]
[(0,1)],[(0,0)]
[(0,1)],[(0,0)]
[],[(0,1)]
[(0,1)],[(0,0)]
[(0,1)],[(0,1)]
[(0,1)],[(0,0)]
[(0,1)],[(0,0)]
{0,1}
{1,2}
{0,1}
{0,1}
{0,1}
{2,3}
{1,2}
{0}
{1,7}
{1,6}
{1,8}
{3}
{2,4}
{1,2}
{7}
{6}
{8,9}
{8,10}
{4}
{2,5}
{1,2}
{5}
{1}
{2}
{9,11}
{8,9}
{10,13}
{10,14}
{9}
{11}
{8}
{9,12}
{13}
{10}
{14}
{12}

Labeled trees for general tree decompositions

1. Write every bag set as a tuple of size $w + 1$ (repeat vertices if necessary).
2. For every bag t generate a labeling $\lambda(t) := [eq(t)], [overlap(t)], [edge(t)]$ with the following properties:

$eq(t)$	Is a list of pairs of positions in the bag which represent the same vertex of graph.
$overlap(t)$	If there is a parent $p = (p_0, \ldots, p_w)$ of $t = (t_0, \ldots, t_w)$ it contains a pair (i, j) if $t_i = p_j$.
$edge(t)$	Contains a pair (i, j) if for $t = (t_0, \ldots, t_w)$ there is an edge in the original graph between t_i and t_j.

The number of possible words representing bags of the tree decomposition is bounded by $2^{(w+1)^2} \cdot 2^{(w+1)^2} \cdot 2^{(w+1)^2}$, thus the alphabet of the tree has constant size.

Interpretation for width w

We can repeat the same principles we used for labeled trees representing tree decompositions of width one for labeled trees representing trees of width w.

Instead of using formulas like $\phi_{0,0}$, $\phi_{0,1}$ and $\phi_{1,1}$ to write a formula which forces the inclusion of bags when they overlap with bags already contained we use the relation $overlap_{i,j}$ to do the same.

$$\phi_{\text{overlap}}(X_0, \ldots, X_w) := \forall s \forall t (\\ \bigwedge_{i,j} t \in X_i \wedge ((S_0(s,t) \vee S_1(s,t) \wedge overlap_{i,j}(t)) \\ \vee (S_0(t,s) \vee S_1(t,s) \wedge overlap_{j,i}(s)))) \rightarrow s \in X_j$$

Interpretation for width w

We can now force the inclusion of bags that overlap with bags already contained for a set for some position. We have to be careful to also deal with values repeating in a bag.

$$\phi_{\mathsf{eq}}(X_0, \ldots, X_w) := \bigwedge_i \forall t(t \in X_i \rightarrow (\bigwedge_{i \neq j} eq_{i,j}(t) \rightarrow t \in X_j))$$

The above formula forces a bag that contains repeated nodes to be contained in all the appropriate position sets.

We can finally redefine $\phi_{\textsf{conn}}(X_0, \ldots, X_w)$ to be a formula which is true when $\bigcup_i X_i$ contains all bags that contain some node u.

$$\phi_{\textsf{conn}}(X_0, \ldots, X_w) = \phi_{\textsf{overlap}}(X_0, \ldots, X_w) \wedge \phi_{\textsf{eq}}(X_0, \ldots, X_w)$$

We can finally redefine $\phi_{\mathsf{conn}}(X_0, \ldots, X_w)$ to be a formula which is true when $\bigcup_i X_i$ contains all bags that contain some node u.

$$\phi_{\mathsf{conn}}(X_0, \ldots, X_w) = \phi_{\mathsf{overlap}}(X_0, \ldots, X_w) \wedge \phi_{\mathsf{eq}}(X_0, \ldots, X_w)$$

We can use as before exclusion minimality to make sure that we get exactly the bags that contain some node u.

$$\phi_{\mathsf{exc}}(X_0, \ldots, X_w) := \phi_{\mathsf{conn}}(X_0, \ldots, X_w) \wedge (\bigvee_i X_i \neq \varnothing) \wedge$$
$$\forall X'_0, \ldots, X'_w((\bigvee_i X'_i \subsetneq X_i) \wedge (\bigwedge_i X'_i \subseteq X_i)) \rightarrow \neg\phi_{\mathsf{conn}}(X'_0, \ldots, X'_w))$$

We can finally redefine $\phi_{\mathsf{conn}}(X_0, \ldots, X_w)$ to be a formula which is true when $\bigcup_i X_i$ contains all bags that contain some node u.

$$\phi_{\mathsf{conn}}(X_0, \ldots, X_w) = \phi_{\mathsf{overlap}}(X_0, \ldots, X_w) \wedge \phi_{\mathsf{eq}}(X_0, \ldots, X_w)$$

We can use as before exclusion minimality to make sure that we get exactly the bags that contain some node u.

$$\phi_{\mathsf{exc}}(X_0, \ldots, X_w) := \phi_{\mathsf{conn}}(X_0, \ldots, X_w) \wedge (\bigvee_i X_i \neq \varnothing) \wedge$$
$$\forall X'_0, \ldots, X'_w((\bigvee_i X'_i \subsetneq X_i) \wedge (\bigwedge_i X'_i \subseteq X_i)) \rightarrow \neg\phi_{\mathsf{conn}}(X'_0, \ldots, X'_w))$$

And finally we can get the sought after formula for the edge relation.

$$\begin{aligned} \phi_{edge}(x, y) := & x \neq y \wedge \exists X_0 \ldots X_w \exists Y_0 \ldots Y_w (x \in X_0 \wedge y \in Y_0 \\ & \wedge \phi_{\mathsf{exc}}(X_0, \ldots, X_w) \wedge \phi_{\mathsf{exc}}(Y_0, \ldots, Y_w) \\ & \wedge \exists z((\bigvee_{i,j} z \in X_i \wedge z \in Y_j \wedge edge_{i,j}(z)))) \end{aligned}$$

Size of the formula for edges

Question

How big is our new $\phi_{edge}(x, y)$?

Size of the formula for edges

Question

How big is our new $\phi_{edge}(x, y)$?

Answer

Constant, since the size only depends on w.

Almost Courcelle's Theorem

Theorem (Almost Courcelle's Theorem)

Every graph property definable in MSO$_1$ *can be decided in linear time on graphs of bounded treewidth.*

Proof.

The following algorithm achieves linear running time:

1. Compute a tree decomposition of minimal width w for the graph in linear time.
2. Convert the tree decomposition into a leaf decomposition.
3. Convert the leaf decomposition into a labeled tree T' whose labeling is bounded in w.
4. Interpret the given MSO$_1$ formula to work on tree decompositions of width w.
5. Convert this formula into a DTA.
6. Run the DTA on T'.

□

Courcelle's Theorem

Theorem

MSO$_1$ *and* MSO$_2$ *have the same expressive power on graph classes of bounded treewidth.*

Proof.

A graph class with bounded treewidth w is w-degenerate (every subgraph contains a node of maximal degree w). MSO$_1$ and MSO$_2$ have the same expressive power on graph classes with bounded degeneracy. □

Courcelle's Theorem

Theorem

MSO_1 *and* MSO_2 *have the same expressive power on graph classes of bounded treewidth.*

Proof.

A graph class with bounded treewidth w is w-degenerate (every subgraph contains a node of maximal degree w). MSO_1 and MSO_2 have the same expressive power on graph classes with bounded degeneracy. $\square$

We have now proven Courcelle's Theorem.

Theorem (Courcelle's Theorem)

Every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth.

Improvement

Definition ($\mathbf{L}$)

The complexity class $\mathbf{L}$ contains all problems which can be solved in logarithmic space.

Theorem (Elberfeld, Jakoby and Tantau)

Every graph property definable in monadic second-order logic can be decided in ***logarithmic space*** *on graphs of bounded treewidth.*

Running time

The running time of the previous algorithm for a formula ψ on treewidth w is $f(||\psi||, w) \cdot n$ where $f(||\psi||, w)$ is

$$f(||\psi||, w) := 2^{2^{\cdot^{\cdot^{\cdot^{2^{w}}}}}} \Big\}\Theta(||\psi||) \; .$$

Running time

The running time of the previous algorithm for a formula ψ on treewidth w is $f(||\psi||, w) \cdot n$ where $f(||\psi||, w)$ is

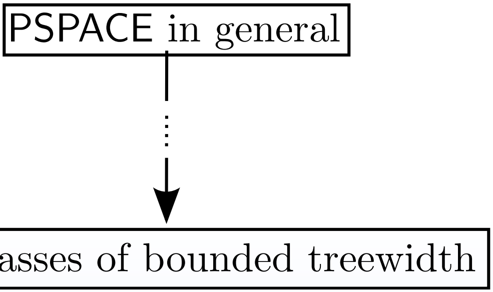
$$f(||\psi||, w) := 2^{2^{\cdot^{\cdot^{\cdot^{2^w}}}}} \Big\} \Theta(||\psi||) \; .$$

This is also basically the best one can do.

Theorem (Frick and Grohe)

Unless $\mathbf{P} = \mathbf{NP}$ the the model-checking problem for monadic second-order logic on finite words is not solvable in time $f(||\psi||) \cdot p(n)$, for any elementary function f and any polynomial p.

The complexity of MSO-checking



Question

Is there something in between which is polynomial?

The complexity of MSO-checking

$$\begin{tikzcd}
\boxed{\textsf{PSPACE} \text{ in general}} \arrow[d, dashed] \\
\boxed{\textsf{L} \text{ in classes of bounded treewidth}}
\end{tikzcd}$$

Question

Is there something in between which is polynomial?

Answer

Not really...

The complexity of MSO-checking

Question

Is there something in between which is polynomial?

Answer

Not really...

Theorem (Kreutzer and Tazari)

Let $\mathcal{C}$ be a graph class that is closed under subgraphs, and has polylogarithmically unbounded treewidth $(\log^c(|G_n|) \leq tw(G_n))$*. Then given* $G \in \mathcal{C}, \psi \in$ MSO$_1$ *with* $|\psi|$ *as parameter, deciding whether* $G \models \psi$ *is not in* $\mathbf{XP}$*, unless the Exponential Time Hypothesis fails.*

The complexity of MSO-checking

Question

Is there something in between which is polynomial?

Answer

Not really…

Theorem (Kreutzer and Tazari)

Let $\mathcal{C}$ be a graph class that is closed under subgraphs, and has polylogarithmically unbounded treewidth $(\log^c(|G_n|) \leq tw(G_n))$. Then given $G \in \mathcal{C}, \psi \in \mathsf{MSO}_1$ with $|\psi|$ ***being a constant****, deciding whether $G \models \psi$ is not* ***solvable in time*** *$O(n^{f(|\psi|)})$, unless* ***something terrible happens****.*

Extension of Courcelle's Theorem

The following logics are also solvable in polynomial time on graph classes of bounded treewidth:

1. EMSO [Arnborg et al.]
2. EMSO-counting [Arnborg et al.]
3. MSO-evaluation [Courcelle and Mosbah]

Takeaway

- MSO_2 and extensions can be model-checked in linear time for graphs of bounded treewidth.
- The running time dependence on the given formula is terrible, and the worst case can not be good.
- This result can not be extended to graph classes where the treewidth is not constant but grows very slowly.
- This gives a certain kind of characterization of MSO_2 model-checking: It is very easy (linear time/logarithmic space) for graphs of bounded treewidth and hard otherwise.

Takeaway

- MSO_2 and extensions can be model-checked in linear time for graphs of bounded treewidth.
- The running time dependence on the given formula is terrible, and the worst case can not be good.
- This result can not be extended to graph classes where the treewidth is not constant but grows very slowly.
- This gives a certain kind of characterization of MSO_2 model-checking: It is very easy (linear time/logarithmic space) for graphs of bounded treewidth and hard otherwise.
- (It has cool implications in the realm of $\mathbf{FPT}$, but more about that on Monday).

EOF