Courcelle's Theorem Made Dynamic

Patricia Bouyer-Decitre[1,2], Vincent Jugé[1,2,3] & Nicolas Markey[1,2,4]

1: CNRS — 2: ENS Paris-Saclay (LSV) — 3: UPEM (LIGM) — 4: Rennes (IRISA)

03/10/2017

Contents

1. Dynamic Complexity of Decision Problems
2. Courcelle's Theorem
3. Making Courcelle's Theorem Dynamic

Dynamic Complexity of Decision Problems

Modulo 3 Decision

- Input: Elements $x_1, x_2, \ldots, x_n$ of $\mathbb{F}_3$
- Output: **Yes** if $x_1 + x_2 + \ldots + x_n = 0$ — **No** otherwise

Dynamic Complexity of Decision Problems

Modulo 3 Decision

- Input: Elements $x_1, x_2, \ldots, x_n$ of $\mathbb{F}_3$
- Output: **Yes** if $x_1 + x_2 + \ldots + x_n = 0$ — **No** otherwise

Solving this problem. . .

- **Static world**: membership in a regular language

Dynamic Complexity of Decision Problems

Modulo 3 Decision

- Input: Elements $x_1, x_2, \ldots, x_n$ of $\mathbb{F}_3$
- Output: **Yes** if $x_1 + x_2 + \ldots + x_n = 0$ — **No** otherwise

Solving this problem. . .

- **Static world**: membership in a regular language
- **Dynamic world**: what if some element x_k changes?
  - Maintain predicates $\mathbf{S}_i \equiv$ "$x_1 + x_2 + \ldots + x_n = i$" for $i \in \mathbb{F}_3$
  - Update the values of $\mathbf{S}_0$, $\mathbf{S}_1$, $\mathbf{S}_2$ when x_k changes
  - Use the new value of $\mathbf{S}_0$ and answer the problem

Dynamic Complexity of Decision Problems

Modulo 3 Decision

- Input: Elements $x_1, x_2, \ldots, x_n$ of $\mathbb{F}_3$
- Output: **Yes** if $x_1 + x_2 + \ldots + x_n = 0$ — **No** otherwise

Solving this problem...

- **Static world**: membership in a regular language
- **Dynamic world**: what if some element x_k changes?
  - Maintain predicates $\mathbf{S}_i \equiv$ "$x_1 + x_2 + \ldots + x_n = i$" for $i \in \mathbb{F}_3$
  - Update the values of $\mathbf{S}_0$, $\mathbf{S}_1$, $\mathbf{S}_2$ when x_k changes
  - Use the new value of $\mathbf{S}_0$ and answer the problem

How complex is it?

- **Static world**: linear time
- **Dynamic world**:
  - **Easy** initial instance $(x_1 = x_2 = \ldots = x_n = 0)$: constant time
  - Each update: constant time

Dynamic Complexity of Decision Problems

Reachability in DAGs

- Input: Directed acyclic graph $G = (V, E)$ & two vertices $s, t \in V$
- Output: **Yes** if $\exists$ path from s to t in G — **No** otherwise

Dynamic Complexity of Decision Problems

Reachability in DAGs

- Input: Directed acyclic graph $G = (V, E)$ & two vertices $s, t \in V$
- Output: **Yes** if $\exists$ path from s to t in G — **No** otherwise

Solving this problem...

- **Static world**: use your favorite graph exploration algorithm
- **Dynamic world**: what if edge $u \to v$ is inserted/deleted?
  - Maintain a predicate $\mathbf{R}(x, y) \equiv (\exists$ path from x to y in $G)$ for $x, y \in V$
  - Update the values of $\mathbf{R}(x, y)$ when $u \to v$ is inserted/deleted
  - Use the new value of $\mathbf{R}(s, t)$ and answer the problem

Dynamic Complexity of Decision Problems

Reachability in DAGs

- Input: Directed acyclic graph $G = (V, E)$ & two vertices $s, t \in V$
- Output: **Yes** if $\exists$ path from s to t in G — **No** otherwise

Solving this problem...

- **Static world**: use your favorite graph exploration algorithm
- **Dynamic world**: what if edge $u \to v$ is inserted/deleted?
  - Maintain a predicate $\mathbf{R}(x, y) \equiv (\exists$ path from x to y in $G)$ for $x, y \in V$
  - Update the values of $\mathbf{R}(x, y)$ when $u \to v$ is inserted/deleted
  - Use the new value of $\mathbf{R}(s, t)$ and answer the problem

How complex is it?

- **Static world**: linear time
- **Dynamic world**:
  - **Easy** initial edgeless instance: FO formulæ
  - Each update: FO formulæ

Dynamic Complexity of Decision Problems

Reachability in DAGs

- Input: Directed acyclic graph $G = (V, E)$ & two vertices $s, t \in V$
- Output: **Yes** if $\exists$ path from s to t in G — **No** otherwise

Solving this problem. . .

- **Static world**: use your favorite graph exploration algorithm
- **Dynamic world**: what if edge $u \to v$ is inserted/deleted?
  - Maintain a predicate $\mathbf{R}(x, y) \equiv (\exists$ path from x to y in $G)$ for $x, y \in V$
  - Update the values of $\mathbf{R}(x, y)$ when $u \to v$ is inserted/deleted
  - Use the new value of $\mathbf{R}(s, t)$ and answer the problem

How complex is it?

- **Static world**: linear time
- **Dynamic world**:
  - **Easy** initial edgeless instance: FO formulæ (**parallel** constant time)
  - Each update: FO formulæ (**parallel** constant time)

FO formulæ $\Rightarrow$ parallel $\approx$constant time

$$\phi = \exists x.\forall y.\psi(x,y) \vee \psi(y,x)$$

FO formulæ ⇒ parallel ≈constant time

$$\phi = \exists x. \forall y. \psi(x, y) \vee \psi(y, x)$$

$\psi(e_1, e_1)$ $\psi(e_1, e_2)$ $\psi(e_2, e_1)$ $\psi(e_2, e_2)$

● ● ● ●

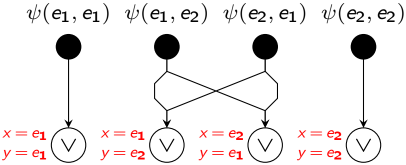
FO formulæ $\Rightarrow$ parallel $\approx$constant time

$$\phi = \exists x. \forall y. \psi(x, y) \vee \psi(y, x)$$

$\psi(e_1, e_1)$ $\psi(e_1, e_2)$ $\psi(e_2, e_1)$ $\psi(e_2, e_2)$

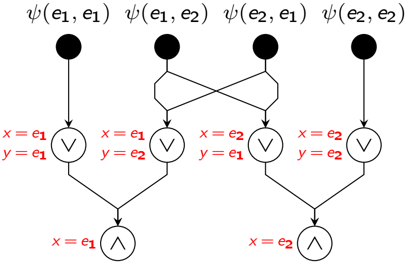
FO formulæ ⇒ parallel ≈constant time

$$\phi = \exists x. \forall y. \psi(x, y) \vee \psi(y, x)$$



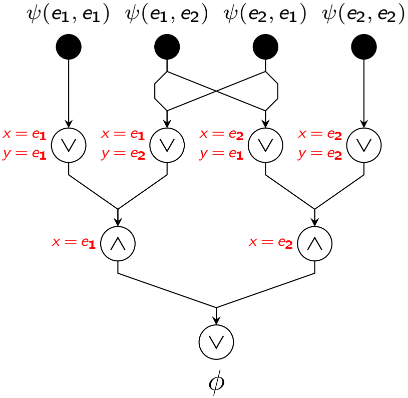
FO formulæ ⇒ parallel ≈constant time

$$\phi = \exists x. \forall y. \psi(x,y) \vee \psi(y,x)$$



FO formulæ ⇒ parallel ≈constant time

$$\phi = \exists x.\forall y.\psi(x,y) \vee \psi(y,x)$$

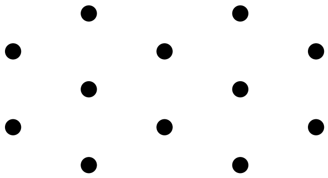


Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓

$\mathbf{R}(x, y) \leftarrow (x = y)$

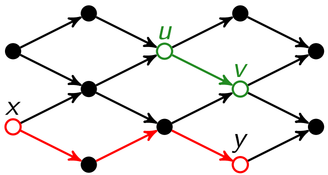


Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \rightarrow v$

$\mathbf{R}(x, y) \leftarrow \mathbf{R}(x, y)$

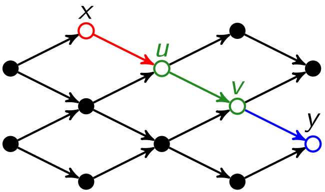


Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \to v$: ✓

$$\mathbf{R}(x,y) \leftarrow \mathbf{R}(x,y) \vee (\mathbf{R}(x,u) \wedge \mathbf{R}(v,y))$$

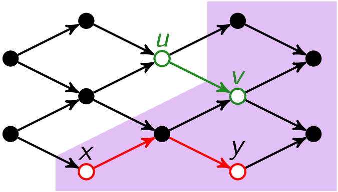


Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \to v$: ✓
- Update after **deleting** the edge $u \to v$

$$\mathbf{R}(x, y) \leftarrow (\mathbf{R}(x, y) \wedge \neg \mathbf{R}(x, u))$$

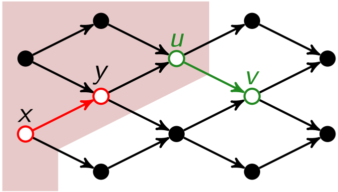


Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \to v$: ✓
- Update after **deleting** the edge $u \to v$

$$\mathbf{R}(x,y) \leftarrow (\mathbf{R}(x,y) \wedge \neg\mathbf{R}(x,u)) \vee (\mathbf{R}(x,y) \wedge \mathbf{R}(y,u))$$

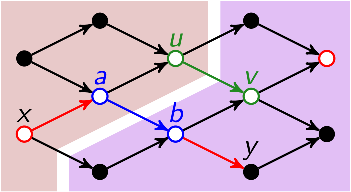


Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \rightarrow v$: ✓
- Update after **deleting** the edge $u \rightarrow v$: ✓

$$\begin{aligned}\mathbf{R}(x,y) \leftarrow\ & (\mathbf{R}(x,y) \wedge \neg\mathbf{R}(x,u)) \vee \\ & (\mathbf{R}(x,y) \wedge \mathbf{R}(y,u)) \vee \\ & (\exists a.\exists b.\mathbf{R}(x,a) \wedge \mathbf{R}(b,y) \wedge \\ & \ (a \rightarrow b) \wedge (a,b) \neq (u,v) \wedge \\ & \ \mathbf{R}(a,u) \wedge \neg\mathbf{R}(b,u))\end{aligned}$$



Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \to v$: ✓
- Update after **deleting** the edge $u \to v$: ✓

$\Rightarrow$ You can even **maintain paths** from s to t!

Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \rightarrow v$: ✓
- Update after **deleting** the edge $u \rightarrow v$: ✓

Definition (Dong & Su & Topor 93 – Patnaik & Immerman 97)

A decision problem with updates is in $\mathcal{C}$-**DynFO** if $\exists$ predicates s.t.:

- every predicate can be initialized in $\mathcal{C}$
- every predicate can be updated in FO
- one predicate is the goal predicate

Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓
- Update after **inserting** the edge $u \to v$: ✓
- Update after **deleting** the edge $u \to v$: ✓

Definition (Dong & Su & Topor 93 – Patnaik & Immerman 97)

A decision problem with updates is in **DynFO** if ∃ predicates s.t.:

- every predicate can be initialized in FO
- every predicate can be updated in FO
- one predicate is the goal predicate

Dynamic Complexity of Decision Problems

Some more problems in DynFO

- Reachability in undirected graphs (Patnaik & Immerman 97)
- Integer multiplication (Patnaik & Immerman 97)
- Context-free language membership (Gelade et al. 08)
- Distance in undirected graphs (Grädel & Siebertz 12)
- Reachability in directed graphs (Datta et al. 15)

Some problems that might be in DynFO

- Distance in directed graphs
- Next hop / path maintenance in directed graphs
- Shortest path maintenance in undirected graphs

Dynamic Complexity of Decision Problems

Some more problems in LogSpace-DynFO

- Reachability in undirected graphs (Patnaik & Immerman 97)
- Integer multiplication (Patnaik & Immerman 97)
- Context-free language membership (Gelade et al. 08)
- Distance in undirected graphs (Grädel & Siebertz 12)
- Reachability in directed graphs (Datta et al. 15)
- MSO model checking on graphs of small tree-width (Bouyer et al. 17 – Datta et al. 17)

Some problems that might be in DynFO

- Distance in directed graphs
- Next hop / path maintenance in directed graphs
- Shortest path maintenance in undirected graphs

Contents

1 Dynamic Complexity of Decision Problems

2 Courcelle's Theorem

3 Making Courcelle's Theorem Dynamic

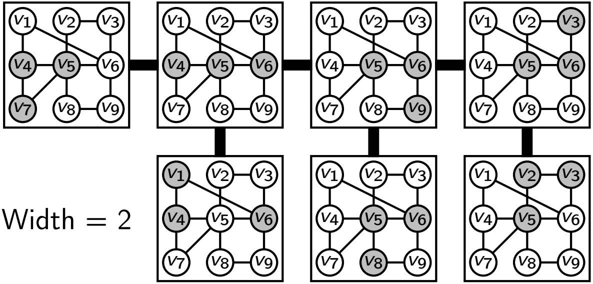
Tree Decompositions and Tree Width

Definition #1 (Halin 76 – Robertson & Seymour 84)

A **tree decomposition** of a graph $G = (V, E)$ is formed of:

- a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$
- a mapping $\mathbf{T} : \mathcal{V} \mapsto 2^V$, such that:
  - for every edge (x, y) of G, we have $\{x, y\} \subseteq \mathbf{T}(v)$ for some node $v \in \mathcal{V}$
  - for every vertex x of G, the set $\{v \in \mathcal{V} \mid x \in \mathbf{T}(v)\}$ is a sub-tree of $\mathcal{T}$

The **width** of the tree decomposition is $\max\{\#\mathbf{T}(v) \mid v \in \mathcal{V}\} - 1$.



Width $= 2$

Tree Decompositions and Tree Width

Definition #2 (Halin 76 – Robertson & Seymour 84)

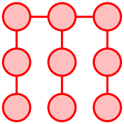
The **tree width** of a graph G is the minimal width of all of G's tree decompositions.

Tree Decompositions and Tree Width

Definition #2 (Halin 76 – Robertson & Seymour 84)

The **tree width** of a graph G is the minimal width of all of G's tree decompositions.

Tree width of some specific graphs



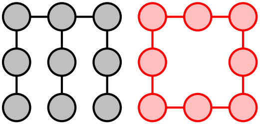
Graph	Width
Tree	1
Cycle	2
K_n	$n-1$
$K_{a,b}$	$\min\{a,b\}$
$\mathbb{Z}_a \times \mathbb{Z}_b$	$\min\{a,b\}$

Tree Decompositions and Tree Width

Definition #2 (Halin 76 – Robertson & Seymour 84)

The **tree width** of a graph G is the minimal width of all of G's tree decompositions.

Tree width of some specific graphs



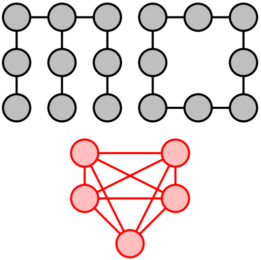
Graph	Width
Tree	1
Cycle	2
K_n	$n-1$
$K_{a,b}$	$\min\{a,b\}$
$\mathbb{Z}_a \times \mathbb{Z}_b$	$\min\{a,b\}$

Tree Decompositions and Tree Width

Definition #2 (Halin 76 – Robertson & Seymour 84)

The **tree width** of a graph G is the minimal width of all of G's tree decompositions.

Tree width of some specific graphs



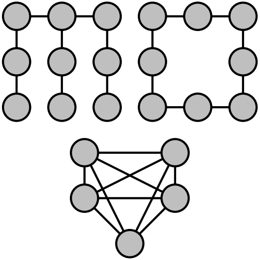
Graph	Width
Tree	1
Cycle	2
K_n	$n-1$
$K_{a,b}$	$\min\{a,b\}$
$\mathbb{Z}_a \times \mathbb{Z}_b$	$\min\{a,b\}$

Tree Decompositions and Tree Width

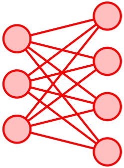
Definition #2 (Halin 76 – Robertson & Seymour 84)

The **tree width** of a graph G is the minimal width of all of G's tree decompositions.

Tree width of some specific graphs



Graph	Width
Tree	1
Cycle	2
K_n	$n-1$
$K_{a,b}$	$\min\{a,b\}$
$\mathbb{Z}_a \times \mathbb{Z}_b$	$\min\{a,b\}$

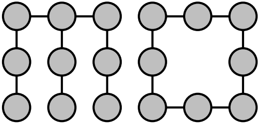


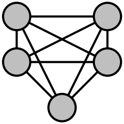
Tree Decompositions and Tree Width

Definition #2 (Halin 76 – Robertson & Seymour 84)

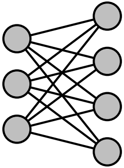
The **tree width** of a graph G is the minimal width of all of G's tree decompositions.

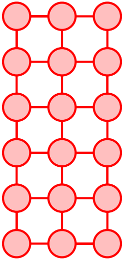
Tree width of some specific graphs





Graph	Width
Tree	1
Cycle	2
K_n	$n-1$
$K_{a,b}$	$\min\{a, b\}$
$\mathbb{Z}_a \times \mathbb{Z}_b$	$\min\{a, b\}$





Monadic Second-Order Formulæ on Directed Graphs

Is the graph $G = (V, E)$

- Undirected? $\forall s. \forall t. (s, t) \in E \Rightarrow (t, s) \in E$

Monadic Second-Order Formulæ on Directed Graphs

Is the graph $G = (V, E)$

- Undirected? $\forall s.\forall t.(s,t) \in E \Rightarrow (t,s) \in E$
- Strongly connected?
  $\forall X.\forall a.\forall b.a \in X \wedge b \notin X \Rightarrow (\exists s.\exists t.s \in X \wedge t \notin X \wedge (s,t) \in E)$
- 3-colorable?
  $\exists V_1.\exists V_2.\exists V_3.V = V_1 \uplus V_2 \uplus V_3 \wedge \forall s.\forall t.\bigwedge_{i=1}^{3}(s \in V_i \wedge t \in V_i) \Rightarrow (s,t) \notin E$

Monadic Second-Order Formulæ on Directed Graphs

Is the **partitioned** graph $G = (V_A \uplus V_B, E)$

- Undirected? $\forall s. \forall t. (s, t) \in E \Rightarrow (t, s) \in E$
- Strongly connected?
  $\forall X. \forall a. \forall b. a \in X \wedge b \notin X \Rightarrow (\exists s. \exists t. s \in X \wedge t \notin X \wedge (s, t) \in E)$
- 3-colorable?
  $\exists V_1. \exists V_2. \exists V_3. V = V_1 \uplus V_2 \uplus V_3 \wedge \forall s. \forall t. \bigwedge_{i=1}^{3} (s \in V_i \wedge t \in V_i) \Rightarrow (s, t) \notin E$
- Properly partitioned? $\forall s. \forall t. (s, t) \in E \Rightarrow (s \in V_A \Leftrightarrow t \in V_B)$
- Winning for Alice (in the reachability game $s \to t$)?
  $\exists$ Alice's strategy s.t. $\forall$ Barbara's strategies, A wins

Monadic Second-Order Formulæ on Directed Graphs

Is the **partitioned** graph $G = (V_A \uplus V_B, E)$

- Undirected? $\forall s.\forall t.(s,t) \in E \Rightarrow (t,s) \in E$
- Strongly connected?
  $\forall X.\forall a.\forall b.a \in X \wedge b \notin X \Rightarrow (\exists s.\exists t.s \in X \wedge t \notin X \wedge (s,t) \in E)$
- 3-colorable?
  $\exists V_1.\exists V_2.\exists V_3.V = V_1 \uplus V_2 \uplus V_3 \wedge \forall s.\forall t. \bigwedge_{i=1}^{3}(s \in V_i \wedge t \in V_i) \Rightarrow (s,t) \notin E$
- Properly partitioned? $\forall s.\forall t.(s,t) \in E \Rightarrow (s \in V_A \Leftrightarrow t \in V_B)$
- Winning for Alice (in the reachability game $s \to t$)?
  $\exists$ Alice's strategy s.t. $\forall$ Barbara's strategies, A wins

Theorem (Karp 72)

Checking a given MSO formula on finite **structures** is NP-hard.

Courcelle's Theorem

Theorem (Courcelle 90, Bodlaender 96 & Eberfeld et al. 10)

For all κ, checking a given MSO formula on n-vertex structures of tree width at most κ is feasible in time $\mathcal{O}(n)$ and space $\mathcal{O}(\log(n))$.

⚠ The constant in the $\mathcal{O}(\cdot)$ may be huge!

Courcelle's Theorem

Theorem (Courcelle 90, Bodlaender 96 & Eberfeld et al. 10)

For all κ, checking a given MSO formula on n-vertex structures of tree width at most κ is feasible in time $\mathcal{O}(n)$ and space $\mathcal{O}(\log(n))$.

⚠ The constant in the $\mathcal{O}(\cdot)$ may be huge!

Proof Idea

1. Compute a tree decomposition of G of width κ
2. Run a **tree** automaton on the tree decomposition

Contents

1 Dynamic Complexity of Decision Problems

2 Courcelle's Theorem

3 Making Courcelle's Theorem Dynamic

Result Framework

Check MSO satisfaction in low dynamic complexity

Result Framework

Check MSO satisfaction in LogSpace-DynFO

Result Framework

Check MSO satisfaction in LogSpace-DynFO

- Too hard in general! Look for restricted cases

Result Framework

Check MSO satisfaction in LogSpace-DynFO

- Too hard in general! — Look for restricted cases
- Use a maximal graph $G_\star = (V, E_\star)$? — Added edges belong to $E_\star$
- Still too hard in general! — Look for further restricted cases

Result Framework

Check MSO satisfaction in LogSpace-DynFO

- Too hard in general! — Look for restricted cases
- Use a maximal graph $G_\star = (V, E_\star)$? — Added edges belong to $E_\star$
- Still too hard in general! — Look for further restricted cases
- Do it for graphs $G_\star$ with tree width at most κ! — Copy Courcelle

Result Framework

Check MSO satisfaction in LogSpace-DynFO

- Too hard in general! — Look for restricted cases
- Use a maximal graph $G_\star = (V, E_\star)$? — Added edges belong to $E_\star$
- Still too hard in general! — Look for further restricted cases
- Do it for graphs $G_\star$ with tree width at most κ! — Copy Courcelle

Result Framework

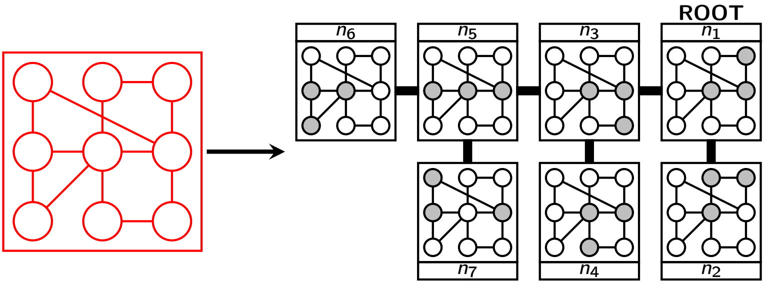
Check MSO satisfaction in LogSpace-DynFO

- Too hard in general! — Look for restricted cases
- Use a maximal graph $G_\star = (V, E_\star)$? — Added edges belong to $E_\star$
- Still too hard in general! — Look for further restricted cases
- Do it for graphs $G_\star$ with tree width at most κ! — Copy Courcelle
- Bonus: Compute witnesses of $\exists$ formulæ

Sketch of Proof

1. Compute a **nice** tree decomposition from G

(linear-size, log-depth binary tree)

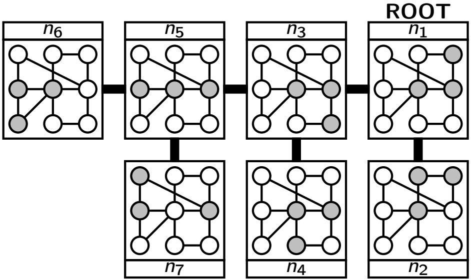


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)

2. Run a (bottom-up, deterministic) automaton

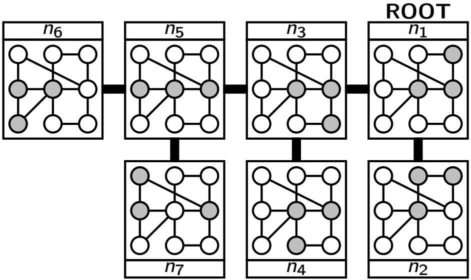


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)

2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

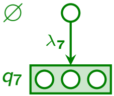


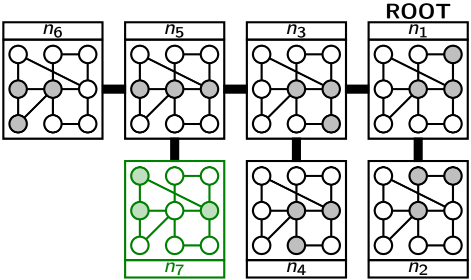
Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)

2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

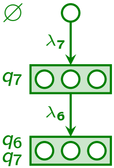


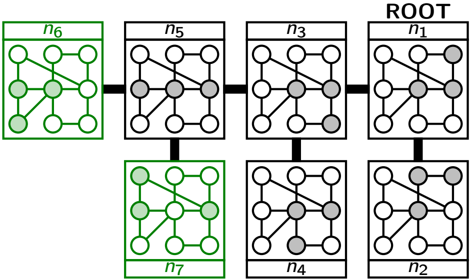


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

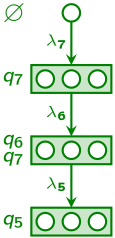


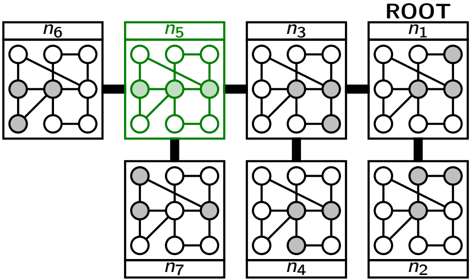


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

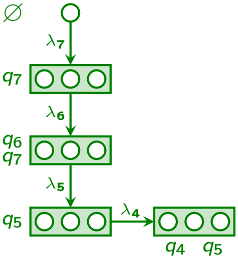


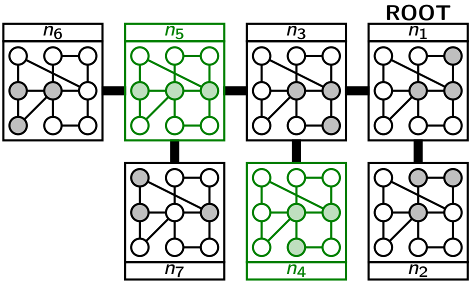


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

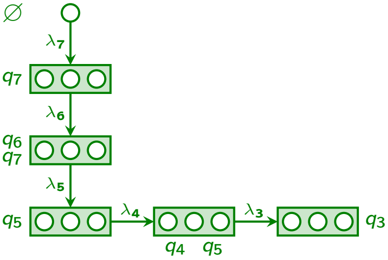


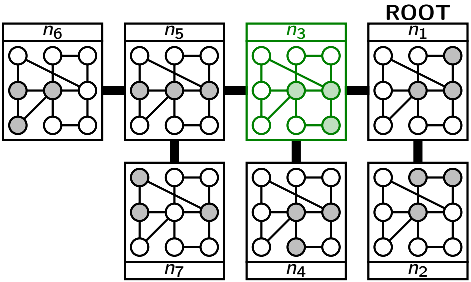


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

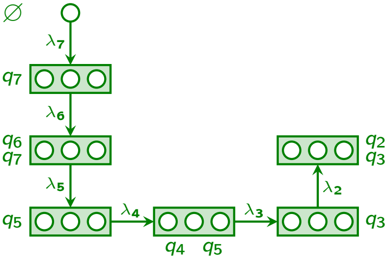


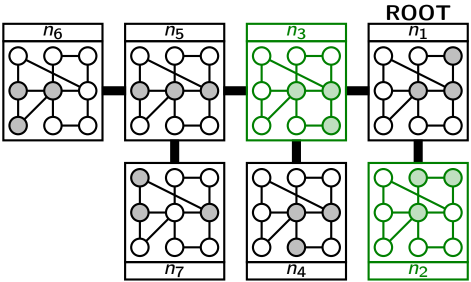


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

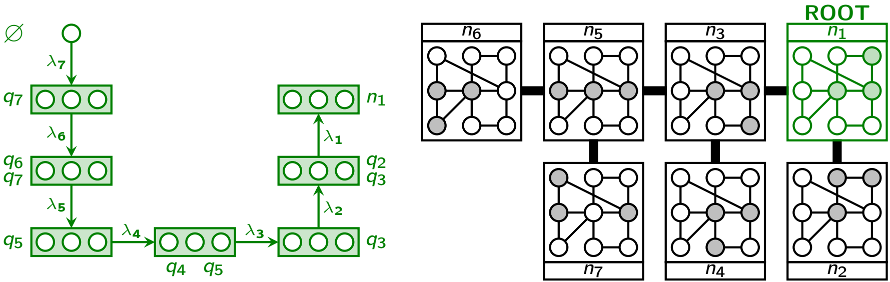




Sketch of Proof

1. Compute a **nice** tree decomposition from G

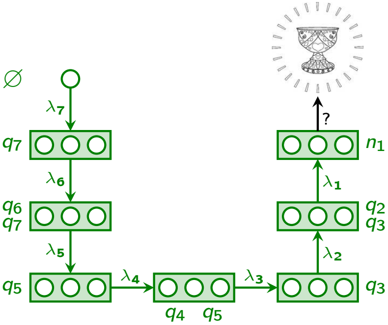
   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'

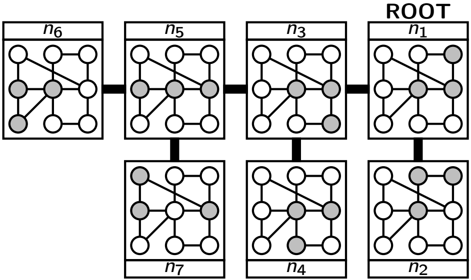


Sketch of Proof

1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)
2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a path in an acyclic graph G'





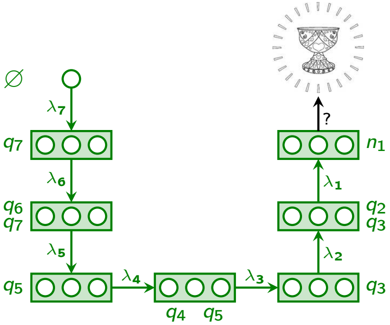
Sketch of Proof

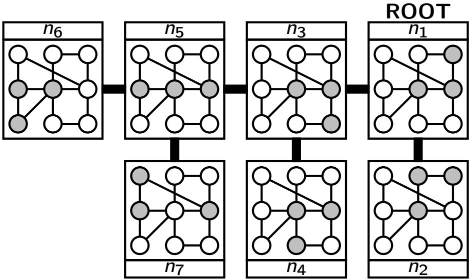
1. Compute a **nice** tree decomposition from G

   (linear-size, log-depth binary tree)

2. Run a (bottom-up, deterministic) automaton **sequentially**
3. Identify its run with a **Dyck** path in an acyclic graph G'

Golden rule: 1 change in G = $\mathcal{O}(1)$ changes in G'





Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( ))
- ([( ]))
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( ))
- ([( ]))
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( ))
- ([( ]))
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( ))
- ([( ]))
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ]))
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ]))
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]

Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

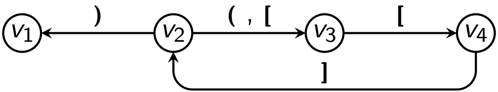
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



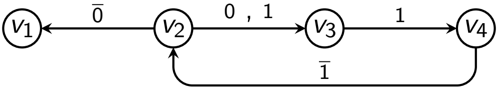
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



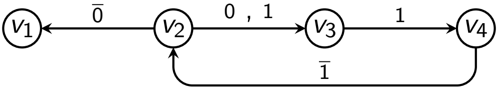
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



v_4

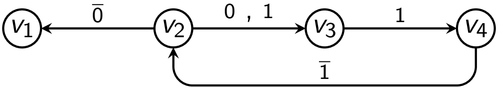
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



$$v_3 \xrightarrow{1} v_4 \xrightarrow{\overline{1}} v_2$$

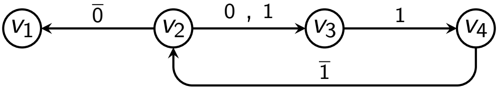
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



$$v_2 \xrightarrow{0} v_3 \xrightarrow{1} v_4 \xrightarrow{\overline{1}} v_2 \xrightarrow{\overline{0}} v_1$$

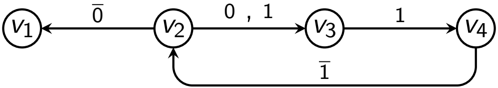
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



$$v_3 \xrightarrow{1} v_4 \xrightarrow{\overline{1}} v_2 \xrightarrow{0} v_3 \xrightarrow{1} v_4 \xrightarrow{\overline{1}} v_2 \xrightarrow{\overline{0}} v_1$$

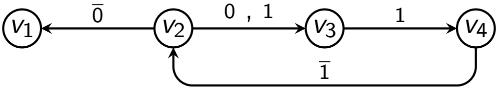
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

- ([( )] ( )): ✓
- ([( ])): ✗
- ([( )] ( )]: ✗

Dyck paths = Paths labeled with Dyck words



$$v_3 \xrightarrow{1} v_4 \xrightarrow{\overline{1}} v_2 \xrightarrow{0} v_3 \xrightarrow{1} v_4 \xrightarrow{\overline{1}} v_2 \xrightarrow{\overline{0}} v_1$$

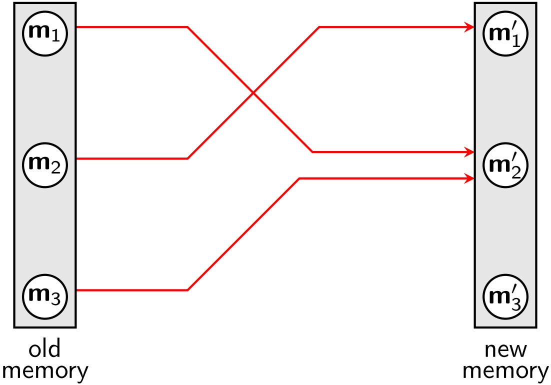
Theorem (Weber & Schwentick 05 – Bouyer et al. 16)

Computing endpoints of Dyck paths in **acyclic** graphs is in DynFO and we can maintain such paths.

Sketch of Proof

Dyck words = Paths on a pushdown graph

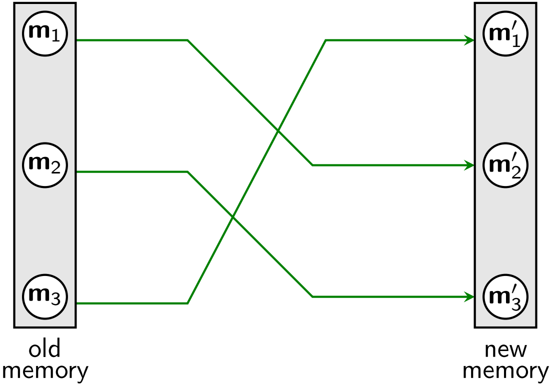
Memory update when reading the symbol ℓ_1



Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol ℓ_2



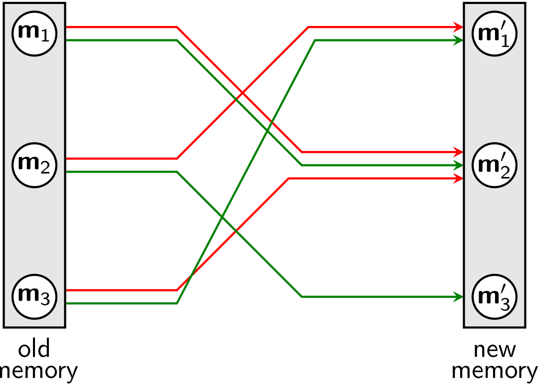
when reading ℓ_1

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_1$
$\mathbf{m}_3 \to \mathbf{m}'_2$

Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol $\ell_?$



when reading ℓ_1

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_1$
$\mathbf{m}_3 \to \mathbf{m}'_2$

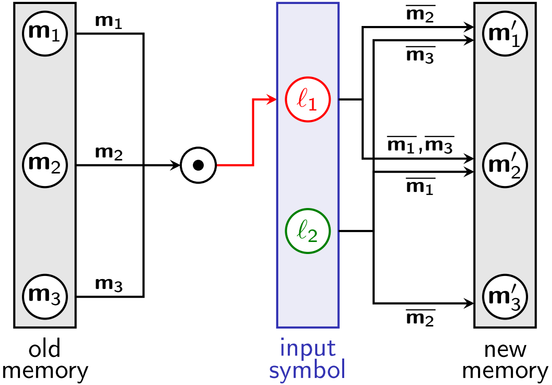
when reading ℓ_2

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_3$
$\mathbf{m}_3 \to \mathbf{m}'_1$

Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol ℓ_1



when reading ℓ_1

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_1$
$\mathbf{m}_3 \to \mathbf{m}'_2$

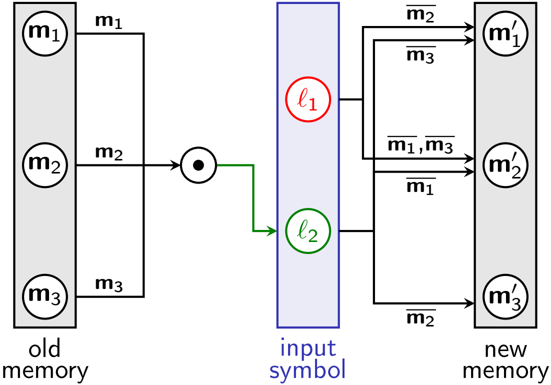
when reading ℓ_2

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_3$
$\mathbf{m}_3 \to \mathbf{m}'_1$

Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol ℓ_2



when reading ℓ_1

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_1$
$\mathbf{m}_3 \to \mathbf{m}'_2$

when reading ℓ_2

$\mathbf{m}_1 \to \mathbf{m}'_2$
$\mathbf{m}_2 \to \mathbf{m}'_3$
$\mathbf{m}_3 \to \mathbf{m}'_1$

Future work

Some problems to investigate:

- Parity games with n priorities ($\approx$ mean-payoff games)
- Nash equilibria with n players

Future work

Some problems to investigate:

- Parity games with n priorities ($\approx$ mean-payoff games)
- Nash equilibria with n players
- Computing good path or tree decompositions in PTIME-DynFO

Future work

Some problems to investigate:

- Parity games with n priorities ($\approx$ mean-payoff games)
- Nash equilibria with n players
- Computing good path or tree decompositions in PTIME-DynFO
- Model checking MSO in **all** graphs of tree width κ (Datta et al. 17)

Future work

Some problems to investigate:

- Parity games with n priorities ($\approx$ mean-payoff games)
- Nash equilibria with n players
- Computing good path or tree decompositions in PTIME-DynFO
- Model checking MSO in **all** graphs of tree width κ (Datta et al. 17)