Courcelle’s Theorem Made Dynamic

Patricia Bouyer-Decitre\textsuperscript{1,2}, Vincent Jugé\textsuperscript{1,2,3} & Nicolas Markey\textsuperscript{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
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
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 $S_i \equiv "x_1 + x_2 + \ldots + x_n = i"$ for $i \in \mathbb{F}_3$
  - Update the values of $S_0, S_1, S_2$ when $x_k$ changes
  - Use the new value of $S_0$ 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 $S_i \equiv "x_1 + x_2 + \ldots + x_n = i"$ for $i \in \mathbb{F}_3$
  - Update the values of $S_0, S_1, S_2$ when $x_k$ changes
  - Use the new value of $S_0$ 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

---

Solving this problem...

**Static world**: use your favorite graph exploration algorithm

**Dynamic world**: what if edge $u \not\rightarrow v$ is inserted/deleted?

- § Maintain a predicate $R_{p,x,y,q}$
  - § Update the values of $R_{p,s,t,q}$ when $u \not\rightarrow v$ is inserted/deleted
  - § Use the new value of $R_{p,s,t,q}$ 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)
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 \( R(x, y) \equiv (\exists \) path from \( x \) to \( y \) in \( G \)) for \( x, y \in V \)
  - Update the values of \( R(x, y) \) when \( u \to v \) is inserted/deleted
  - Use the new value of \( R(s, t) \) and answer the problem
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 $u \rightarrow v$ is inserted/deleted?
  - Maintain a predicate $R(x, y) \equiv (\exists$ path from $x$ to $y$ in $G)$ for $x, y \in V$
  - Update the values of $R(x, y)$ when $u \rightarrow v$ is inserted/deleted
  - Use the new value of $R(s, t)$ 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 $u \rightarrow v$ is inserted/deleted?
  - Maintain a predicate $R(x, y) \equiv (\exists$ path from $x$ to $y$ in $G)$ for $x, y \in V$
  - Update the values of $R(x, y)$ when $u \rightarrow v$ is inserted/deleted
  - Use the new value of $R(s, t)$ 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

$$\phi = \exists x. \forall y. \psi(x, y) \lor \psi(y, x)$$
FO formulae $\Rightarrow$ parallel $\approx$ constant time

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

\[\begin{array}{ccccc}
\psi(e_1, e_1) & \psi(e_1, e_2) & \psi(e_2, e_1) & \psi(e_2, e_2)
\end{array}\]

\[\begin{array}{cccc}
\bullet & \bullet & \bullet & \bullet
\end{array}\]
FO formulae $\Rightarrow$ parallel $\approx$ constant time

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

$$\psi(e_1, e_1) \quad \psi(e_1, e_2) \quad \psi(e_2, e_1) \quad \psi(e_2, e_2)$$
FO formulæ ⇒ parallel ≈ constant time

\[ \phi = \exists x. \forall y. \psi(x, y) \lor \psi(y, x) \]

\[ \psi(e_1, e_1) \quad \psi(e_1, e_2) \quad \psi(e_2, e_1) \quad \psi(e_2, e_2) \]
FO formulæ ⇒ parallel \cong constant time

\[ \phi = \exists x. \forall y. \psi(x, y) \lor \psi(y, x) \]

\[ \psi(e_1, e_1) \quad \psi(e_1, e_2) \quad \psi(e_2, e_1) \quad \psi(e_2, e_2) \]
FO formulae $\Rightarrow$ parallel $\approx$ constant time

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

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

$\phi$
Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

- Initialization (on the edgeless graph): ✓

\[ R(x, y) \leftarrow (x = y) \]
Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

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

\[ R(x, y) \leftarrow R(x, y) \]
Dynamic Complexity of Decision Problems

Reachability in DAGs with FO formulæ

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

$$R(x, y) \leftarrow R(x, y) \lor (R(x, u) \land R(v, y))$$

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
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 DynFO if

$$R(x, y) \leftarrow (R(x, y) \land \neg R(x, u))$$
Reachability in DAGs with FO formulæ

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

$$R(x, y) \leftarrow (R(x, y) \land \neg R(x, u)) \lor (R(x, y) \land R(y, u))$$
Reachability in DAGs with FO formulæ

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

\[
R(x, y) \leftarrow (R(x, y) \land \neg R(x, u)) \lor \\
(R(x, y) \land R(y, u)) \lor \\
(\exists a. \exists b. R(x, a) \land R(b, y) \land \\
(a \rightarrow b) \land (a, b) \neq (u, v) \land \\
R(a, u) \land \neg R(b, u))
\]
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 $u \rightarrow v$: ✓

⇒ 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 \( \mathcal{C}\text{-DynFO} \) 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 \( u \rightarrow v \): ✓
- Update after **deleting** the edge \( u \rightarrow v \): ✓

### Definition (Dong & Su & Topor 93 – Patnaik & Immerman 97)
A decision problem with updates is in **DynFO** if \( \exists \) 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
**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 \( T = (V, E) \)
- a mapping \( T : V \mapsto 2^V \), such that:
  - for every edge \((x, y)\) of \( G \), we have \( \{x, y\} \subseteq T(v) \) for some node \( v \in V \)
  - for every vertex \( x \) of \( G \), the set \( \{v \in V \mid x \in T(v)\} \) is a sub-tree of \( T \)

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

<table>
<thead>
<tr>
<th>Graph</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>1</td>
</tr>
<tr>
<td>Cycle</td>
<td>2</td>
</tr>
<tr>
<td>$K_n$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$K_{a,b}$</td>
<td>$\min{a, b}$</td>
</tr>
<tr>
<td>$\mathbb{Z}_a \times \mathbb{Z}_b$</td>
<td>$\min{a, b}$</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Graph</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>$1$</td>
</tr>
<tr>
<td>Cycle</td>
<td>$2$</td>
</tr>
<tr>
<td>$K_n$</td>
<td>$n−1$</td>
</tr>
<tr>
<td>$K_{a,b}$</td>
<td>$\min{a, b}$</td>
</tr>
<tr>
<td>$\mathbb{Z}_a \times \mathbb{Z}_b$</td>
<td>$\min{a, b}$</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Graph</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>1</td>
</tr>
<tr>
<td>Cycle</td>
<td>2</td>
</tr>
<tr>
<td>$K_n$</td>
<td>$n-1$</td>
</tr>
<tr>
<td>$K_{a,b}$</td>
<td>$\min{a, b}$</td>
</tr>
<tr>
<td>$\mathbb{Z}_a \times \mathbb{Z}_b$</td>
<td>$\min{a, b}$</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>Graph</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>1</td>
</tr>
<tr>
<td>Cycle</td>
<td>2</td>
</tr>
<tr>
<td>$K_n$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$K_{a,b}$</td>
<td>$\min{a, b}$</td>
</tr>
<tr>
<td>$\mathbb{Z}_a \times \mathbb{Z}_b$</td>
<td>$\min{a, b}$</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Graph</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>1</td>
</tr>
<tr>
<td>Cycle</td>
<td>2</td>
</tr>
<tr>
<td>$K_n$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$K_{a,b}$</td>
<td>$\min{a, b}$</td>
</tr>
<tr>
<td>$\mathbb{Z}_a \times \mathbb{Z}_b$</td>
<td>$\min{a, b}$</td>
</tr>
</tbody>
</table>
Is the graph $G = (V, E)$

- **Undirected?** $\forall s. \forall t. (s, t) \in E \Rightarrow (t, s) \in E$
Monadic Second-Order Formulae 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?**
  \( \forall X. \forall a. \forall b. a \in X \land b \notin X \Rightarrow (\exists s. \exists t. s \in X \land t \notin X \land (s, t) \in E) \)
- **3-colorable?**
  \( \exists V_1. \exists V_2. \exists V_3. V = V_1 \uplus V_2 \uplus V_3 \land \forall s. \forall t. \bigwedge_{i=1}^{3} (s \in V_i \land t \in V_i) \Rightarrow (s, t) \notin E \)
Monadic Second-Order Formulae 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?
  $\forall X.\forall a.\forall b. a \in X \land b \notin X \Rightarrow (\exists s.\exists t. s \in X \land t \notin X \land (s, t) \in E)$
- 3-colorable?
  $\exists V_1.\exists V_2.\exists V_3. V = V_1 \uplus V_2 \uplus V_3 \land \forall s.\forall t. \bigwedge_{i=1}^{3} (s \in V_i \land t \in V_i) \Rightarrow (s, t) \notin E$
- Properly partitioned? $\forall s.\forall t. (s, t) \in E \Rightarrow (s \in V_A \iff t \in V_B)$
- Winning for Alice (in the reachability game $s \rightarrow t$)?
  $\exists$ Alice’s strategy s.t. $\forall$ Barbara’s strategies, A wins
Monadic Second-Order Formulae 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?**
  \( \forall X.\forall a.\forall b.a \in X \land b \notin X \Rightarrow (\exists s.\exists t.s \in X \land t \notin X \land (s, t) \in E) \)
- **3-colorable?**
  \( \exists V_1.\exists V_2.\exists V_3.V = V_1 \uplus V_2 \uplus V_3 \land \forall s.\forall t. \land_{i=1}^{3}(s \in V_i \land t \in V_i) \Rightarrow (s, t) \notin E \)
- **Properly partitioned?** \( \forall s.\forall t.(s, t) \in E \Rightarrow (s \in V_A \iff t \in V_B) \)
- **Winning for Alice (in the reachability game \( s \rightarrow t \))?**
  \( \exists \) 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 $\kappa$, checking a given MSO formula on $n$-vertex structures of tree width at most $\kappa$ is feasible in time $O(n)$ and space $O(\log(n))$.

⚠️ The constant in the $O(\cdot)$ may be huge!
Courcelle’s Theorem

Theorem (Courcelle 90, Bodlaender 96 & Eberfeld et al. 10)
For all $\kappa$, checking a given MSO formula on $n$-vertex structures of tree width at most $\kappa$ is feasible in time $O(n)$ and space $O(\log(n))$.

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

Proof Idea

1. Compute a tree decomposition of $G$ of width $\kappa$
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

\( G \langle V, E \rangle \)

\( \text{Added edges belong to } E \langle \cdot \rangle \)

\( \text{Still too hard in general! Look for further restricted cases} \)

Do it for graphs \( G \langle \cdot \rangle \) with tree width at most \( \kappa \)!

Copy Courcelle Bonus: Compute witnesses of D formulæ
Result Framework

Check MSO satisfaction in LogSpace-DynFO

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

Check MSO satisfaction in LogSpace-DynFO

- Too hard in general!
- Use a maximal graph $G_\star = (V, E_\star)$?
- Still too hard in general!
- Do it for graphs $G_\star$ with tree width at most $\kappa$!

Look for restricted cases

- Added edges belong to $E_\star$
- Look for further restricted cases

Copy Courcelle
Result Framework

Check MSO satisfaction in LogSpace-DynFO

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

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
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 $G'$

\[ \emptyset \quad \emptyset \]
**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 $G'$

---

Golden rule: $1$ change in $G = O(1)$ changes in $G'$
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 $G'$

---

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

---

Golden rule: 1 change in $G = O(1)$ changes in $G'$

---

![Diagram](image)
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 $G'$

---

Golden rule: $1$ change in $G = O(1)$ changes in $G'$

---

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
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 $G'$

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
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 $G'$

---

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
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?

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

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 = 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

\[ \text{v}_1 \rightarrow \text{v}_2 \rightarrow \text{v}_3 \rightarrow \text{v}_4 \]

\[ (, [ ) ] \]
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

\[
\begin{align*}
\text{v}_1 & \rightarrow \text{v}_2 \quad 0 \rightarrow 1 \quad \text{v}_3 \quad 1 \rightarrow \text{v}_4 \\
\text{v}_3 & \rightarrow \text{v}_4 \quad \overline{1} \rightarrow \text{v}_2
\end{align*}
\]
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

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

Dyck paths = Paths labeled with Dyck words

```
\begin{array}{cccc}
  v_1 & \overset{0}{\longrightarrow} & v_2 & \overset{0, 1}{\longrightarrow} v_3 & \overset{1}{\longrightarrow} v_4 \\
\end{array}
```

```
v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_2 \rightarrow v_1
```

---

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

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

Dyck paths = Paths labeled with Dyck words

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.

P. Bouyer-Decitre, V. Jugé & N. Markey
Courcelle’s Theorem Made Dynamic
Sketch of Proof

Dyck words = Well-parenthesized words

Are these words Dyck?

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

Dyck paths = Paths labeled with Dyck words

\[
\begin{align*}
  &v_1 \to v_2 \quad 0 \quad 1 \quad v_3 \quad 0 \quad 1 \quad v_4 \\
  &v_3 \to v_4 \quad \overline{1} \quad v_2 \to v_3 \quad 1 \quad \overline{1} \quad v_2 \to v_1
\end{align*}
\]

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.

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol $\ell_1$

old memory

new memory

P. Bouyer-Decitre, V. Jugé & N. Markey
Courcelle’s Theorem Made Dynamic
Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol $\ell_2$

when reading $\ell_1$

$m_1 \rightarrow m'_2$
$m_2 \rightarrow m'_1$
$m_3 \rightarrow m'_2$
Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol $\ell$?

When reading $\ell_1$
- $m_1 \rightarrow m'_2$
- $m_2 \rightarrow m'_1$
- $m_3 \rightarrow m'_2$

When reading $\ell_2$
- $m_1 \rightarrow m'_1$
- $m_2 \rightarrow m'_3$
- $m_3 \rightarrow m'_1$
Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol $\ell_1$

When reading $\ell_1$
- $m_1 \rightarrow m'_1$
- $m_2 \rightarrow m'_2$
- $m_3 \rightarrow m'_3$

When reading $\ell_2$
- $m_1 \rightarrow m'_2$
- $m_2 \rightarrow m'_3$
- $m_3 \rightarrow m'_1$
Sketch of Proof

Dyck words = Paths on a pushdown graph

Memory update when reading the symbol $\ell_2$

When reading $\ell_1$

$\begin{align*}
m_1 &\rightarrow m_2' \\
m_2 &\rightarrow m_1' \\
m_3 &\rightarrow m_2'
\end{align*}$

When reading $\ell_2$

$\begin{align*}
m_1 &\rightarrow m_2' \\
m_2 &\rightarrow m_3' \\
m_3 &\rightarrow m_1'
\end{align*}$

P. Bouyer-Decitre, V. Jugé & N. Markey

Courcelle’s Theorem Made Dynamic
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 $\kappa$ \hspace{1cm} (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 $\kappa$ (Datta et al. 17)

Thank you