Reasoning about Reachability
at Tom Rep’s 60th Birthday Celebration

Neil Immerman
College of Information and Computer Sciences
UMass Amherst

people.cs.umass.edu/~imimerman/
1978: Tom 22
   Neil 24
   @Cornell
1978: Tom 22
Neil 24
@Cornell

1999: Descriptive Complexity
2002: FLoC, Tom told me his idea...fun collaboration...
2016: still working on it

Arithmetic Hierarchy
FO(N)
- r.e. complete
- co-r.e.
- Halt

Recursive
FO(\exists(N))
- r.e.
- Halt

Primitive Recursive

EXPTIME
SO(LFP)
- SO[2^{n^{O(1)}}]

PSPACE
QSAT
- PSPACE complete

PTIME Hierarchy
NP complete
- SAT
- co-NP

FO[2^{n^{O(1)}}]
- FO(PFP)
- SO(TC)
- SO[2^{n^{O(1)}}]

FO[\exists^{n^{O(1)}}]
- FO(Horn)
- SO(Horn)

P complete
P

NC
FO[log n^{O(1)}]
- “truly feasible”

AC^1
FO[n^{O(1)}]
- FO(CFL)

sAC^1
FO(\exists^{O(1)})
- FO(TC)
- FO(DTC)

NL
FO(\exists^{n^{O(1)}})
- 2SAT
- NL comp.

L
FO(\exists^{log n})
- FO(REGULAR)
- FO(COUNT)

ThC^0
FO
- LOGTIME Hierarchy
- AC^0

2COLOR
- L comp.
1978: Tom 22
Neil 24
@Cornell

1999: Descriptive Complexity

2002: FLoC,
Tom told me his idea
1978: Tom 22
Neil 24
@Cornell

1999: Descriptive Complexity

2002: FLoC,
Tom told me his idea
1978: Tom 22
Neil 24
@Cornell

1999:
Descriptive
Complexity

2002: FLoC,
Tom told me his idea

fun collaboration
1978: Tom 22
Neil 24
@Cornell
:
1999:
Descriptive Complexity
2002: FLoC,
Tom told me his idea
:
fun collaboration
:
1978: Tom 22
Neil 24
@Cornell

1999:
Descriptive
Complexity

2002: FLoC,
Tom told me his
idea

fun

2016: still
working on it
Static
1. Read entire input
2. Compute boolean query $Q(input)$
3. Classic Complexity Classes are static: FO, NC, P, NP, ...
**Static**
1. Read entire input
2. Compute boolean query $Q(\text{input})$
3. Classic Complexity Classes are static: FO, NC, P, NP, …
4. What is the fastest way upon reading the entire input, to compute the query?
Background: Dynamic Complexity

**Static**
1. Read entire input
2. Compute boolean query $Q(input)$
3. Classic Complexity Classes are static: FO, NC, P, NP, …
4. What is the fastest way *upon reading the entire input*, to compute the query?

**Dynamic**
1. Long series of Inserts, Deletes, Changes, and, Queries
2. On *query*, *very quickly* compute $Q(current\ database)$
3. Dynamic Complexity Classes: Dyn-FO, Dyn-NC
Background: Dynamic Complexity

**Static**
1. Read entire input
2. Compute boolean query $Q(\text{input})$
3. Classic Complexity Classes are static: FO, NC, P, NP, …
4. What is the fastest way **upon reading the entire input**, to compute the query?

**Dynamic**
1. Long series of Inserts, Deletes, Changes, and, Queries
2. On **query**, **very quickly** compute $Q(\text{current database})$
3. Dynamic Complexity Classes: Dyn-FO, Dyn-NC
4. What **additional information** should we maintain? — **auxiliary data structure**
Dynamic (Incremental) Applications

- Databases
- LaTexing a file
- Performing a calculation
- Processing a visual scene
- Understanding a natural language
- Verifying a circuit
- Verifying and compiling a program
Dynamic (Incremental) Applications

- Databases
- LaTeXing a file
- Performing a calculation
- Processing a visual scene
- Understanding a natural language
- Verifying a circuit
- Verifying and compiling a program
- Surviving in the wild
<table>
<thead>
<tr>
<th>Current Database: S</th>
<th>Request</th>
<th>Auxiliary Data: b</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parity

<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>$\text{ins}(3, S)$</td>
<td>0</td>
</tr>
</tbody>
</table>

$$S'(x) \equiv S(x) \lor x = a$$
$$b' \equiv (b \land S(a)) \lor (b' \land \neg S(a)) \lor (\neg b \land \neg S(a))$$
### Parity

<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>00100000</td>
<td><strong>ins(3,S)</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

$$S'(x) \equiv S(x) \lor x = a$$

$$b' \equiv (b \land S(a)) \lor (b' \land \neg S(a)) \lor (\neg b' \land \neg S(a))$$
<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>00100000</td>
<td>ins(3,S)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ins(7,S)</td>
<td></td>
</tr>
<tr>
<td>Current Database: S</td>
<td>Request</td>
<td>Auxiliary Data: b</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>00100000</td>
<td>ins(3,S)</td>
<td>1</td>
</tr>
<tr>
<td>00100001</td>
<td>ins(7,S)</td>
<td>0</td>
</tr>
</tbody>
</table>
### Parity

<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>00100000</td>
<td><strong>ins</strong>(3,$S$)</td>
<td>1</td>
</tr>
<tr>
<td>0010001</td>
<td><strong>ins</strong>(7,$S$)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td><strong>del</strong>(3,$S$)</td>
<td></td>
</tr>
</tbody>
</table>
## Parity

<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>00100000</td>
<td>ins$(3,S)$</td>
<td>1</td>
</tr>
<tr>
<td>00100001</td>
<td>ins$(7,S)$</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>del$(3,S)$</td>
<td>1</td>
</tr>
</tbody>
</table>
### Parity

<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>00100000</td>
<td>ins(3,S)</td>
<td>1</td>
</tr>
<tr>
<td>00100001</td>
<td>ins(7,S)</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>del(3,S)</td>
<td>1</td>
</tr>
</tbody>
</table>

**ins(a,S)**

$$S'(x) \equiv S(x) \lor x = a$$

$$b' \equiv (b \land S(a)) \lor (\neg b \land \neg S(a))$$

**del(a,S)**

$$S'(x) \equiv S(x) \land x \neq a$$

$$b' \equiv (b \land \neg S(a)) \lor (\neg b \land S(a))$$
Dynamic Examples

Parity

- Does binary string $w$ have an odd number of 1’s?
- **Static:** $\text{TIME}[n]$, $\text{FO}[\Omega(\log n / \log \log n)]$
- **Dynamic:** $\text{Dyn-TIME}[1]$, $\text{Dyn-FO}$
Dynamic Examples

Parity

- Does binary string $w$ have an odd number of 1’s?
- **Static:** $\text{TIME}[n]$, $\text{FO}[\Omega(\log n / \log \log n)]$
- **Dynamic:** $\text{Dyn-TIME}[1]$, $\text{Dyn-FO}$

$\text{REACH}_u$

- Is $t$ reachable from $s$ in undirected graph $G$?
- **Static:** not in FO, requires $\text{FO}[\Omega(\log n / \log \log n)]$
- **Dynamic:** in $\text{Dyn-FO}$  [Patnaik, I]
Dynamic Examples

Parity

- Does binary string \(w\) have an odd number of 1's?
- **Static:** \(\text{TIME}[n], \text{FO}[\Omega(\log n / \log \log n)]\)
- **Dynamic:** \(\text{Dyn-TIME}[1], \text{Dyn-FO}\)

REACH\(_u\)

- Is \(t\) reachable from \(s\) in undirected graph \(G\)?
- **Static:** not in \(\text{FO}\), requires \(\text{FO}[\Omega(\log n / \log \log n)]\)
- **Dynamic:** in Dyn-FO [Patnaik, I]

connectivity,
minimum spanning trees,
\(k\)-edge connectivity, ...
In TVLA we build a bounded-size summary of an unbounded data structure, updating it after each program step until we reach a fixed point.
In TVLA we build a bounded-size summary of an unbounded data structure, updating it after each program step until we reach a fixed point.

We want to maintain accurate information in that summary concerning pointer reachability.
In TVLA we build a bounded-size summary of an unbounded data structure, updating it after each program step until we reach a fixed point.

We want to maintain accurate information in that summary concerning pointer reachability.

Can some of your ideas for maintaining auxiliary information about a dynamic graph in order to compute reachability information more efficiently?
In TVLA we build a bounded-size summary of an unbounded data structure, updating it after each program step until we reach a fixed point.

We want to maintain accurate information in that summary concerning pointer reachability.

Can some of your ideas for maintaining auxiliary information about a dynamic graph in order to compute reachability information more efficiently, instead be used in TVLA to keep auxiliary information that allows us to maintain reachability information more accurately?
Fact: [Dong & Su] \( \text{REACH(acyclic)} \in \text{DynFO} \)

\( \text{ins}(a, b, E): P'(x, y) \equiv P(x, y) \lor (P(x, a) \land P(b, y)) \)

\( \text{del}(a, b, E): \)

\[
P'(x, y) \equiv P(x, y) \land \left[ \neg(P(x, a) \land P(b, y)) \lor (\exists uv)(P(x, u) \land E(u, v) \land P(v, y) \land P(u, a) \land \neg P(v, a) \land (a \neq u \lor b \neq v)) \right]
\]
Reachability Problems

\[
\text{REACH} = \left\{ G \mid G \text{ directed, } s \overset{*}{\rightarrow}_G t \right\}
\]

\[
\text{REACH}_d = \left\{ G \mid G \text{ directed, outdegree } \leq 1, s \overset{*}{\rightarrow}_G t \right\}
\]

\[
\text{REACH}_u = \left\{ G \mid G \text{ undirected, } s \overset{*}{\rightarrow}_G t \right\}
\]

\[
\text{REACH}_a = \left\{ G \mid G \text{ alternating, } s \overset{*}{\rightarrow}_G t \right\}
\]
Facts about dynamic REACHABILITY Problems:

\[
\begin{align*}
\text{Dyn-REACH}(\text{acyclic}) & \in \text{Dyn-FO} \\
\text{Dyn-REACH}_d & \in \text{Dyn-QF} \\
\text{Dyn-REACH}_u & \in \text{Dyn-FO} \\
\text{Dyn-REACH} & \in \text{Dyn-FO(\text{COUNT})} \\
\text{Dyn-PAD(\text{REACH}_a)} & \in \text{Dyn-FO}
\end{align*}
\]
Exciting New Result

**Reachability is in DynFO**

by Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick and Thomas Zeume


They show that Matrix Rank is in DynFO and REACH reduces to Matrix Rank.
**Thm. 1** [Hesse] Reachability of functional DAG is in DynQF.

**proof:** Maintain $E$, $E^*$, $D$ (outdegree = 1).

**Insert $E(i, j)$:** (ignore if adding edge violates outdegree or acyclicity)

\[
\begin{align*}
E'(x, y) & \equiv E(x, y) \lor (x = i \land y = j) \\
D'(x) & \equiv D(x) \lor x = i \\
E^*(x, y) & \equiv E^*(x, y) \lor (E^*(x, i) \land E^*(j, y))
\end{align*}
\]
**Thm. 1** [Hesse] Reachability of functional DAG is in DynQF.

**proof:** Maintain $E$, $E^*$, $D$ (outdegree = 1).

**Insert $E(i, j)$:** (ignore if adding edge violates outdegree or acyclicity)

\[
E'(x, y) \equiv E(x, y) \lor (x = i \land y = j)
\]
\[
D'(x) \equiv D(x) \lor x = i
\]
\[
E^{*'}(x, y) \equiv E^*(x, y) \lor (E^*(x, i) \land E^*(j, y))
\]

**Delete $E(i, j)$:**

\[
E'(x, y) \equiv E(x, y) \land (x \neq i \lor y \neq j)
\]
\[
D'(x) \equiv D(x) \land (x \neq i \lor \neg E(i, j))
\]
\[
E^{*'}(x, y) \equiv E^*(x, y) \land \neg(E^*(x, i) \land E(i, j) \land E^*(j, y))
\]
Reasoning About reachability – can we get to $b$ from $a$ by following a sequence of pointers – is crucial for proving that programs meet their specifications.
**Dynamic Reasoning**

**Reasoning About reachability** – can we get to $b$ from $a$ by following a sequence of pointers – is **crucial for proving that programs meet their specifications**.
In general, reasoning about reachability is undecidable.

- Can express tilings and thus runs of Turing Machines.
In general, reasoning about reachability is **undecidable**.

- Can express tilings and thus runs of Turing Machines.
- Even worse, can express **finite path** and thus **finite** and thus **standard natural numbers**. Thus $\text{FO(TC)}$ is as hard as the Arithmetic Hierarchy [Avron].
Much is still decidable.

[Itzhaky et. al.]
Much is still decidable.

[Itzhaky et. al.]

For now, restrict to acyclic fields.
Much is still decidable.

[Itzhaky et. al.]

For now, restrict to acyclic fields.

$n(x, y)$ means that $x$ points to $y$. 
Much is still decidable.

[Itzhaky et. al.] For now, restrict to acyclic fields.

$n(x, y)$ means that $x$ points to $y$.

Use predicate symbol, $n^*$, but not $n$. 
Much is still decidable.

[Itzhaky et. al.]

For now, restrict to acyclic fields.

$n(x, y)$ means that $x$ points to $y$.

Use predicate symbol, $n^*$, but not $n$.

The following axioms assure that $n^*$ is the reflexive transitive closure of some acyclic, functional $n$. 
Much is still decidable.

[Itzhaky et. al.]

For now, **restrict** to **acyclic** fields.

\( n(x, y) \) means that \( x \) points to \( y \).

Use predicate symbol, \( n^* \), **but not** \( n \).

The following axioms assure that \( n^* \) is the reflexive transitive closure of some acyclic, functional \( n \).

\[
\text{acyclic} \equiv \forall xy \ (n^*(x, y) \land n^*(y, x) \iff x = y)
\]
Much is still decidable.

[Itzhaky et. al.]

For now, restrict to **acyclic** fields.

$n(x, y)$ means that $x$ points to $y$.

Use predicate symbol, $n^*$, **but not** $n$.

The following axioms assure that $n^*$ is the reflexive transitive closure of some acyclic, functional $n$.

\[
\text{acyclic} \equiv \forall xy (n^*(x, y) \land n^*(y, x) \iff x = y)
\]

\[
\text{transitive} \equiv \forall xyz (n^*(x, y) \land n^*(y, z) \rightarrow n^*(x, z))
\]
Much is still decidable.

[Itzhaky et. al.]

For now, restrict to acyclic fields.

$n(x, y)$ means that $x$ points to $y$.

Use predicate symbol, $n^*$, but not $n$.

The following axioms assure that $n^*$ is the reflexive transitive closure of some acyclic, functional $n$.

\[
\text{acyclic} \equiv \forall xy \left( n^*(x, y) \land n^*(y, x) \iff x = y \right)
\]

\[
\text{transitive} \equiv \forall xyz \left( n^*(x, y) \land n^*(y, z) \rightarrow n^*(x, z) \right)
\]

\[
\text{linear} \equiv \forall xyz \left( n^*(x, y) \land n^*(x, z) \rightarrow n^*(y, z) \lor n^*(z, y) \right)
\]
Assume acyclic, transitive and linear axioms, as integrity constraints.
Effectively-Propositional Reasoning about Reachability in Linked Data Structures

- Assume acyclic, transitive and linear axioms, as integrity constraints.
- Automatically transform a program manipulating linked lists to an $\forall\exists$ correctness condition.
Assume acyclic, transitive and linear axioms, as integrity constraints.

Automatically transform a program manipulating linked lists to an $\forall \exists$ correctness condition.

Using Hesse’s dynQF algorithm for $\text{REACH}_d$, these $\forall \exists$ formulas are closed under weakest precondition.
Assume acyclic, transitive and linear axioms, as integrity constraints.

Automatically transform a program manipulating linked lists to an $\forall \exists$ correctness condition.

Using Hesse’s dynQF algorithm for REACH$_d$, these $\forall \exists$ formulas are closed under weakest precondition.

The negation of the correctness condition is $\exists \forall$, thus equi-satisfiable with a propositional formula.
Assume acyclic, transitive and linear axioms, as integrity constraints.

Automatically transform a program manipulating linked lists to an $\forall\exists$ correctness condition.

Using Hesse’s dynQF algorithm for $\text{REACH}_d$, these $\forall\exists$ formulas are closed under weakest precondition.

The negation of the correctness condition is $\exists\forall$, thus equi-satisfiable with a propositional formula.

Use a SAT solver to automatically prove correctness or find counter-example runs, typically in only a few seconds.
Thm. 2 [Hesse] Reachability of functional graphs is in DynQF.

**proof idea:** If adding an edge, $e$, would create a cycle, then we maintain relation $p^*$ – the path relation without the edge completing the cycle – as well as $E^*$, $E$ and $D$.

Surprisingly this can all be maintained via quantifier-free formulas, **without remembering which edges we are leaving out** in computing $p^*$.
**Thm. 2** [Hesse] Reachability of functional graphs is in DynQF.

**proof idea:** If adding an edge, $e$, would create a cycle, then we maintain relation $p^*$ – the path relation without the edge completing the cycle – as well as $E^*$, $E$ and $D$.

Surprisingly this can all be maintained via quantifier-free formulas, *without remembering which edges we are leaving out* in computing $p^*$. □

Using Thm. 2, the above methodology has been extended to cyclic deterministic graphs.

Thank You!

Anindya Banerjee, Sumit Gulwani, Bill Hesse, Shachar Itzhaky, Aleksandr Karbyshev, Ori Lahav, Tal Lev-Ami, Aleksandar Nanevski, Oded Padon, Sushant Patnaik, Alex Rabinovich, Mooly Sagiv, Sharon Shoham, Siddharth Srivastava, Greta Yorsh