Reasoning about Reachability

at Tom Rep's 60th Birthday Celebration

Neil Immerman

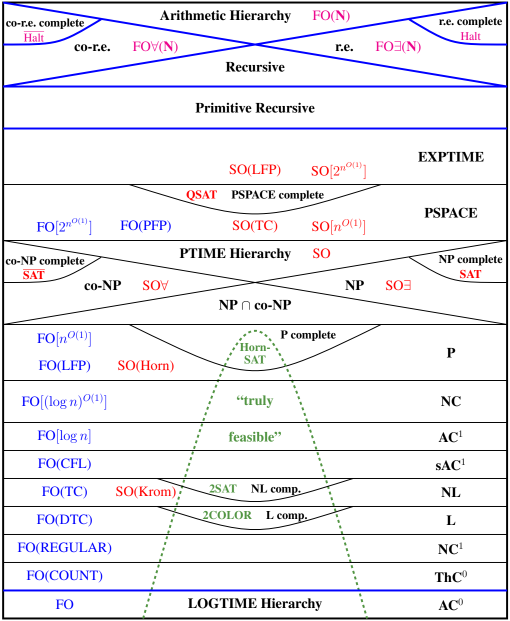
College of Information and Computer Sciences

UMass Amherst

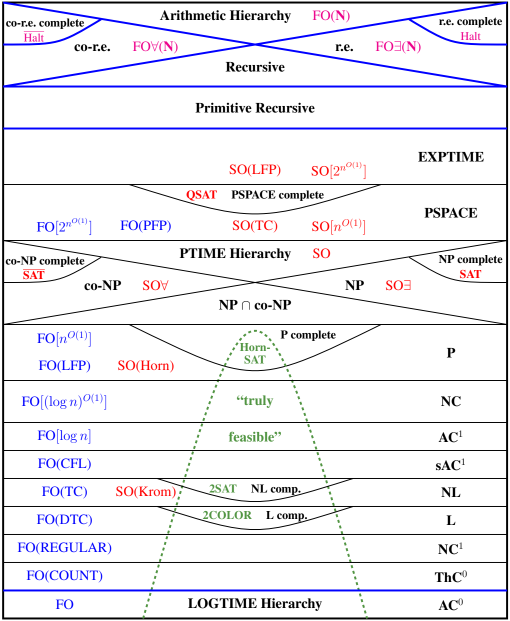
`people.cs.umass.edu/~immerman/`

1978: Tom 22
Neil 24
@Cornell

1978: Tom 22
Neil 24
@Cornell
⋮

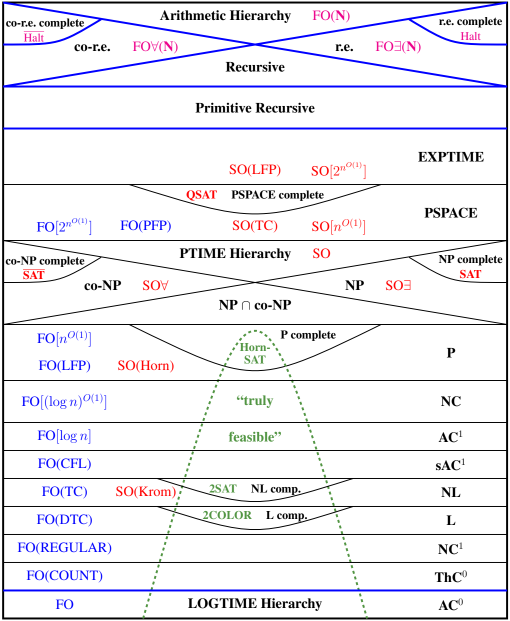


1978: Tom 22
Neil 24
@Cornell
⋮
1999: *Descriptive Complexity*



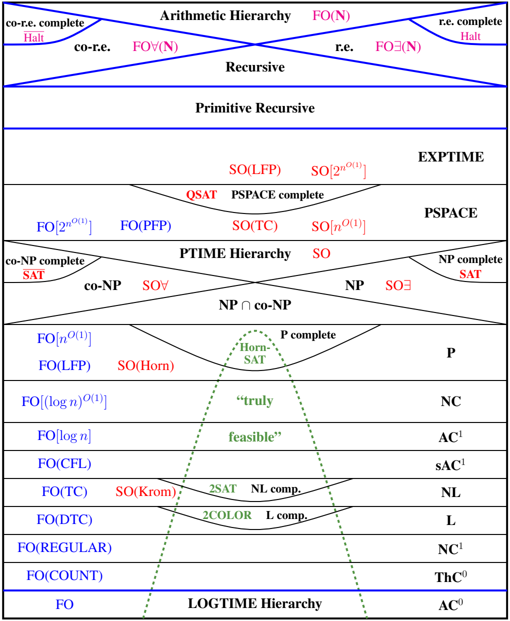
1978: Tom 22
Neil 24
@Cornell
⋮
1999:
Descriptive Complexity

2002: FLoC, Tom told me his idea



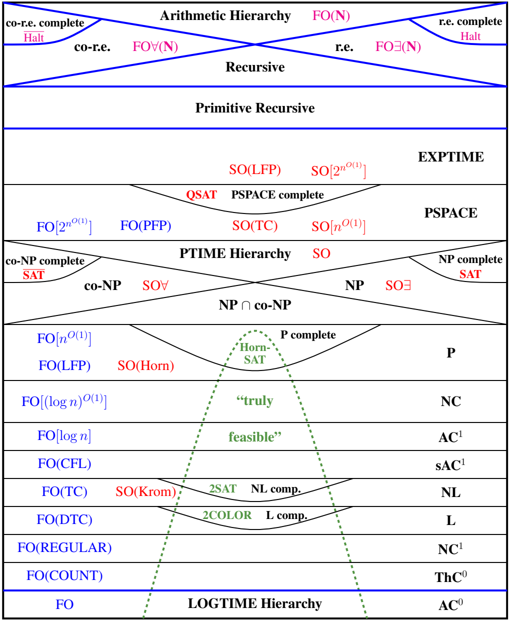
1978: Tom 22
Neil 24
@Cornell
⋮
1999:
Descriptive Complexity

2002: FLoC,
Tom told me his
idea
⋮



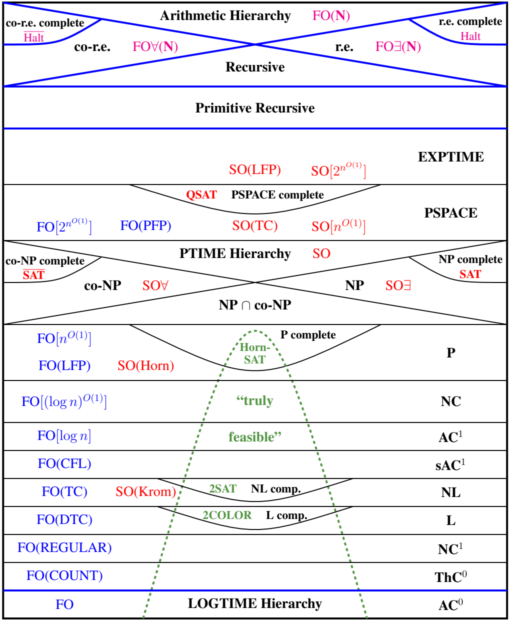
1978: Tom 22
Neil 24
@Cornell
⋮
1999:
Descriptive Complexity

2002: FLoC,
Tom told me his idea
⋮
fun collaboration



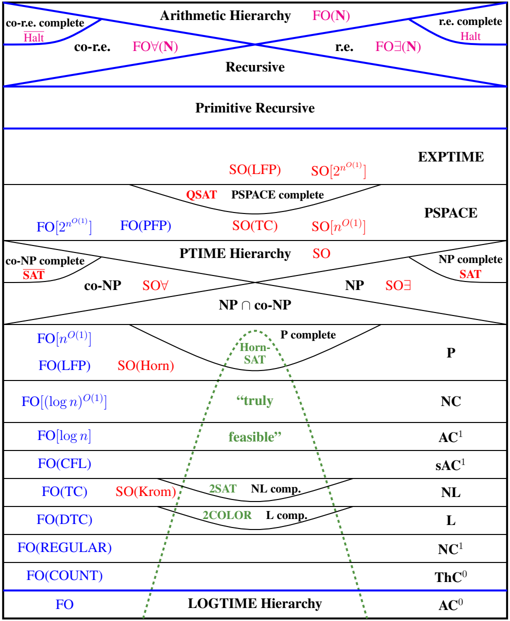
1978: Tom 22
Neil 24
@Cornell
⋮
1999:
Descriptive Complexity

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 collaboration
⋮
2016: still working on it



Background: Dynamic Complexity

Static

1. Read entire input
2. Compute boolean query **Q**(input)
3. Classic Complexity Classes are static: FO, NC, P, NP, ...

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?

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**(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
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

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0
	ins(3,S)	

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0
0010000	**ins**(3,S)	1

Parity

Current Database: *S*	Request	Auxiliary Data: *b*
0000000		0
0010000	**ins**(3,S)	1
	ins(7,S)	

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0
0010000	**ins**(3,S)	1
0010001	**ins**(7,S)	0

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0
0010000	**ins**(3,S)	1
0010001	**ins**(7,S)	0
	del(3,S)	

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0
0010000	**ins**(3,S)	1
0010001	**ins**(7,S)	0
0000001	**del**(3,S)	1

Parity

Current Database: S	Request	Auxiliary Data: b
0000000		0
0010000	**ins**(3,S)	1
0010001	**ins**(7,S)	0
0000001	**del**(3,S)	1

ins(a,S)

$$\begin{aligned} S'(x) &\equiv S(x) \vee x = a \\ b' &\equiv (b \wedge S(a)) \vee \\ &\quad (\neg b \wedge \neg S(a)) \end{aligned}$$

del(a,S)

$$\begin{aligned} S'(x) &\equiv S(x) \wedge x \neq a \\ b' &\equiv (b \wedge \neg S(a)) \vee \\ &\quad (\neg b \wedge S(a)) \end{aligned}$$

Dynamic Examples

Parity

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

Dynamic Examples

Parity

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

REACH$_u$

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

Dynamic Examples

Parity

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

REACH$_u$

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

connectivity,
minimum spanning trees, **in Dyn-FO**
***k*-edge connectivity, . . .**

Tom’s Question at FLoC 2002

- 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.

Tom’s Question at FLoC 2002

- 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.

Tom's Question at FLoC 2002

- 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**,

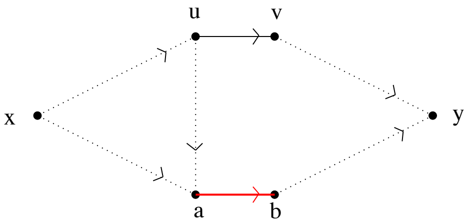
Tom's Question at FLoC 2002

- 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] REACH(acyclic) $\in$ DynFO

ins(a,b,E) : $P'(x,y) \;\equiv\; P(x,y) \vee (P(x,a) \wedge P(b,y))$

del(a,b,E):



$$P'(x,y) \;\equiv\; P(x,y) \wedge \Big[\neg(P(x,a) \wedge P(b,y))$$
$$\vee\ (\exists uv)\big(P(x,u) \wedge E(u,v) \wedge P(v,y)$$
$$\wedge\ P(u,a) \wedge \neg P(v,a) \wedge (a \neq u \vee b \neq v)\big)\Big]$$

Reachability Problems

$$
\begin{array}{rcll}
\text{REACH} & = & \{G \mid G \text{ directed}, s \xrightarrow[G]{\star} t\} & \mathrm{NL} \\
\text{REACH}_d & = & \{G \mid G \text{ directed, outdegree} \leq 1\ s \xrightarrow[G]{\star} t\} & \mathrm{L} \\
\text{REACH}_u & = & \{G \mid G \text{ undirected}, s \xrightarrow[G]{\star} t\} & \mathrm{L} \\
\text{REACH}_a & = & \{G \mid G \text{ alternating}, s \xrightarrow[G]{\star} t\} & \mathrm{P}
\end{array}
$$

Facts about dynamic REACHABILITY Problems:

Dyn-REACH(acyclic)	$\in$	Dyn-FO	[DS]
Dyn-REACH$_d$	$\in$	Dyn-QF	[H]
Dyn-REACH$_u$	$\in$	Dyn-FO	[PI]
Dyn-REACH	$\in$	Dyn-FO(COUNT)	[H]
Dyn-PAD(REACH$_a$)	$\in$	Dyn-FO	[PI]

Exciting New Result

Reachability is in DynFO

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

http://arxiv.org/abs/1502.07467

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)

$$\begin{aligned} E'(x,y) &\equiv E(x,y) \lor (x = i \land y = j) \\ D'(x) &\equiv D(x) \lor x = i \\ E^{*\prime}(x,y) &\equiv E^*(x,y) \lor (E^*(x,i) \land E^*(j,y)) \end{aligned}$$

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)

$$\begin{aligned} E'(x,y) &\equiv E(x,y) \ \vee \ (x = i \wedge y = j) \\ D'(x) &\equiv D(x) \ \vee \ x = i \\ E^{*\prime}(x,y) &\equiv E^*(x,y) \ \vee \ (E^*(x,i) \wedge E^*(j,y)) \end{aligned}$$

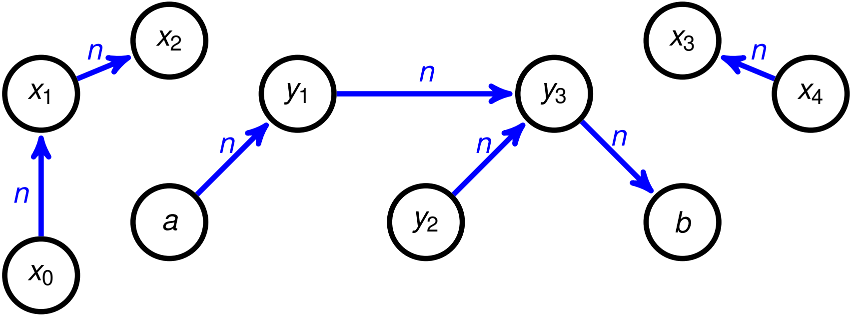
Delete $E(i,j)$:

$$\begin{aligned} E'(x,y) &\equiv E(x,y) \ \wedge \ (x \neq i \vee y \neq j) \\ D'(x) &\equiv D(x) \ \wedge \ (x \neq i \vee \neg E(i,j)) \\ E^{*\prime}(x,y) &\equiv E^*(x,y) \ \wedge \ \neg(E^*(x,i) \wedge E(i,j) \wedge E^*(j,y)) \end{aligned}$$

$\square$

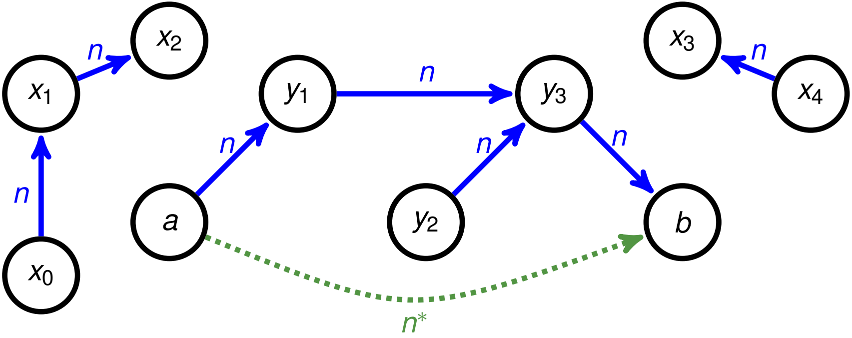
Dynamic Reasoning

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**.



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 $\mathrm{FO}(\mathrm{TC})$ 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.

$$\textbf{acyclic} \quad \equiv \quad \forall xy\,(n^*(x,y) \wedge n^*(y,x) \;\leftrightarrow\; x = y)$$

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.

$$\textbf{acyclic} \equiv \forall xy\,(n^*(x,y) \wedge n^*(y,x) \leftrightarrow x = y)$$

$$\textbf{transitive} \equiv \forall xyz\,(n^*(x,y) \wedge n^*(y,z) \rightarrow n^*(x,z))$$

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.

$$\begin{aligned}
\textbf{acyclic} &\equiv \forall xy\, (n^*(x, y) \land n^*(y, x) \;\leftrightarrow\; x = y) \\
\textbf{transitive} &\equiv \forall xyz\, (n^*(x, y) \land n^*(y, z) \;\rightarrow\; n^*(x, z)) \\
\textbf{linear} &\equiv \forall xyz\, (n^*(x, y) \land n^*(x, z) \;\rightarrow\; n^*(y, z) \lor n^*(z, y))
\end{aligned}$$

Effectively-Propositional Reasoning about Reachability in Linked Data Structures

- 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.

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.
- Using **Hesse's dynQF algorithm** for REACH_d, these $\forall\exists$ formulas are **closed under weakest precondition**.

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.
- 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.

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.
- 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.
- 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^*. $\square$

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^*. $\square$

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

- Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.
- Itzhaky, Banerjee, Immerman, Lahav, Nanevski, Sagiv, "Modular Reasoning about Heap Paths via Effectively Propositional Formulas", POPL 2014

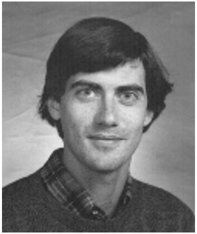
Thank You!

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