Dynamic Reasoning

Neil Immerman

College of Information and Computer Sciences

UMass Amherst

people.cs.umass.edu/~immerman/
Logic in Computer Science

- Descriptive Complexity
Logic in Computer Science

- Descriptive Complexity
- Dichotomy
Logic in Computer Science

- Descriptive Complexity
- Dichotomy
- Dynamic Complexity
Logic in Computer Science

- Descriptive Complexity
- Dichotomy
- Dynamic Complexity
- SAT Solvers
Logic in Computer Science

- Descriptive Complexity
- Dichotomy
- Dynamic Complexity
- SAT Solvers
- Computer Software: Crisis and Opportunity
Logic in Computer Science

- Descriptive Complexity
- Dichotomy
- Dynamic Complexity
- SAT Solvers

- Computer Software: Crisis and Opportunity

Personal perspective
\[ P = \bigcup_{k=1}^{\infty} \text{DTIME}[n^k] \]

P is a good mathematical wrapper for “truly feasible”.

Dynamic Reasoning

ASL 2015 North American meeting, Urbana-Champaign
NTIME[$t(n)$]: a mathematical fiction

input $w$

$|w| = n$

$\text{Dynamic Reasoning}$

ASL 2015 North American meeting, Urbana-Champaign
Many optimization problems we want to solve are NP complete.

\[
\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}[n^k]
\]
Descriptive Complexity

Query $q_1 q_2 \cdots q_n$ $\mapsto$ Computation $\mapsto$ Answer $a_1 a_2 \cdots a_i \cdots a_m$

Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.

How hard is it to check if input has property $S$?

How rich a language do we need to express property $S$?

There is a constructive isomorphism between these two approaches.
Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.
Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.

How hard is it to check if input has property $S$?
Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.

How hard is it to check if input has property $S$?

How rich a language do we need to express property $S$?
Restrict attention to the complexity of computing individual bits of the output, i.e., **decision problems**.

How hard is it to **check** if input has property $S$?

How rich a language do we need to **express** property $S$?

There is a **constructive isomorphism** between these two approaches.
Interpret Input as Finite Logical Structure

Graph

\[ G = (\{v_1, \ldots, v_n\}, E, s, t) \]

Binary

\[ A_w = (\{p_1, \ldots, p_8\}, S) \]

String

\[ S = \{p_2, p_5, p_7, p_8\} \]

\[ w = 01001011 \]

Vocabularies:

\[ \tau_g = (E^2, s, t), \quad \tau_s = (S^1) \]
First-Order Logic

input symbols: from $\tau$

variables: $x, y, z, \ldots$

boolean connectives: $\land, \lor, \neg$

quantifiers: $\forall, \exists$

numeric symbols: $=, \leq, +, \times, \min, \max$

$\alpha \equiv \forall x \exists y (E(x, y)) \in \mathcal{L}(\tau_g)$

$\beta \equiv \exists x \forall y (x \leq y \land S(x)) \in \mathcal{L}(\tau_s)$

$\beta \equiv S(\min) \in \mathcal{L}(\tau_s)$
Fagin's Theorem:

\[ \text{NP} = \text{SO} \]

\[ \Phi_{3\text{-color}} \equiv \exists R^1 G^1 B^1 \forall x y ((R(x) \lor G(x) \lor B(x)) \land (E(x, y) \rightarrow (\neg (R(x) \land R(y)) \land \neg (G(x) \land G(y)) \land \neg (B(x) \land B(y))))))) \]
Fagin’s Theorem: \( \text{NP} = \text{SO}\exists \)

\[
\Phi_{3\text{-color}} \equiv \exists R^1 G^1 B^1 \forall x y ((R(x) \lor G(x) \lor B(x)) \land (E(x, y) \rightarrow \neg(R(x) \land R(y)) \land \neg(G(x) \land G(y)) \land \neg(B(x) \land B(y))))
\]
Addition is First-Order

\[ Q_+ : \text{STRUC}[\tau_{AB}] \to \text{STRUC}[\tau_s] \]

\[
\begin{array}{cccccc}
A & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
B + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
S & s_1 & s_2 & \ldots & s_{n-1} & s_n \\
\end{array}
\]
Addition is First-Order

\[ Q_+ : \text{STRUC}[\tau_{AB}] \rightarrow \text{STRUC}[\tau_s] \]

\[
\begin{array}{ccccccc}
A & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
B & + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
S & \text{\underline{}} & s_1 & s_2 & \ldots & s_{n-1} & s_n
\end{array}
\]

\[ C(i) \equiv (\exists j > i)((A(j) \land B(j) \land (\forall k. j > k > i)(A(k) \lor B(k)))) \]
Addition is First-Order

\[ Q_+ : \text{STRUC}[\tau_{AB}] \rightarrow \text{STRUC}[\tau_s] \]

\[
\begin{array}{cccccccc}
A & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
B & + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
S & & s_1 & s_2 & \ldots & s_{n-1} & s_n \\
\end{array}
\]

\[
C(i) \equiv (\exists j > i) \left( \bigwedge_{j} (A(j) \land B(j)) \land \bigwedge_{k} (A(k) \lor B(k)) \right)
\]

\[
Q_+(i) \equiv A(i) \oplus B(i) \oplus C(i)
\]
Parallel Machines:

\[ \text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}] \]
Parallel Machines: Quantifiers are Parallel

\[ \text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}] \]

Assume array \( A[x] : x = 1, \ldots, r \) in memory.
Parallel Machines: Quantifiers are Parallel

\[ \text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}] \]

Assume array \( A[x] : x = 1, \ldots, r \) in memory.

\[ \forall x(A(x)) \equiv \text{write}(1); \text{proc } p_i : \text{if } (A[i] = 0) \text{ then } \text{write}(0) \]
$\text{FO} = \text{CRAM}[1] = \text{AC}^0 = \text{Logarithmic-Time Hierarchy}$
CRAM\([t(n)]\) = concurrent parallel random access machine; polynomial hardware, parallel time \(O(t(n))\)

IND\([t(n)]\) = first-order, depth \(t(n)\) inductive definitions

FO\([t(n)]\) = \(t(n)\) repetitions of a block of restricted quantifiers:

\[ QB = [(Q_1x_1.M_1) \cdots (Q_kx_k.M_k)]; \quad M_i \text{ quantifier-free} \]

\[ \varphi_n = \underbrace{[QB][QB] \cdots [QB]}_{t(n)} M_0 \]
parallel time = inductive depth = QB iteration

\textbf{Thm:} For all constructible, polynomially bounded $t(n)$,

$$\text{CRAM}[t(n)] = \text{IND}[t(n)] = \text{FO}[t(n)]$$

\textbf{Thm:} For all $t(n)$, even beyond polynomial,

$$\text{CRAM}[t(n)] = \text{FO}[t(n)]$$
For $t(n)$ poly bdd,

$\text{CRAM}[t(n)] = \text{IND}[t(n)] = \text{FO}[t(n)]$
**Theorem** [Ben Rossman] Any first-order formula with any numeric relations (≤, +, ×, . . .) that means “I have a clique of size \( k \)” must have at least \( k/4 \) variables.

- Creative new proof idea using Håstad’s Switching Lemma gives the essentially optimal bound.
- First lower bound of its kind for number of variables with ordering.
- This lower bound is for a fixed formula, if it were for a sequence of polynomially-sized formulas, it would show that CLIQUE \( \not\in P \) and thus \( P \neq NP \).
**Theorem** [Martin Grohe] Fixed-Point Logic with Counting captures Polynomial Time on all classes of graphs with excluded minors.

Grohe proves that for every class of graphs with excluded minors, there is a constant $k$ such that two graphs of the class are isomorphic iff they agree on all $k$-variable formulas in fixed-point logic with counting.

Thus every class of graphs with excluded minors admits the same general polynomial time canonization algorithm: we’re isomorphic iff we agree on all formulas in $C_k$ and in particular, you are isomorphic to me iff your $C_k$ canonical description is equal to mine.

Dichotomy

- “Natural” Computational Problems Tend to be Complete for Important Complexity Classes
“Natural” Computational Problems Tend to be Complete for Important Complexity Classes

Isomorphism Theorem: only one such problem in each class: small handful of naturally occurring decision problems!
“Natural” Computational Problems Tend to be Complete for Important Complexity Classes

Isomorphism Theorem: only one such problem in each class: small handful of naturally occurring decision problems!

Not true for “unnatural problems”: Ladner’s Delayed Diagonalization
Dichotomy

▶ “Natural” Computational Problems Tend to be Complete for Important Complexity Classes

▶ Isomorphism Theorem: only one such problem in each class: *small handful of naturally occurring decision problems!*

▶ Not true for “unnatural problems”: Ladner’s Delayed Diagonalization

▶ Schaefer; Feder-Vardi: CSP Dichotomy Conjecture
“Natural” Computational Problems Tend to be Complete for Important Complexity Classes

Isomorphism Theorem: only one such problem in each class: small handful of naturally occurring decision problems!

Not true for “unnatural problems”: Ladner’s Delayed Diagonalization

Schaefer; Feder-Vardi: CSP Dichotomy Conjecture

Tremendous progress using Universal Algebra. (Solved for domains of size 2 and 3, and for undirected graphs.) See: “Constraint Satisfaction Problem and Universal Algebra” by Libor Barto in SigLog Newsletter.
**Static**

1. Read entire input
2. Compute boolean query $Q(\text{input})$
3. Classic Complexity Classes are static: FO, NC, P, NP, …

**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 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 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
Dynamic Complexity

**Static**

1. Read entire input
2. Compute boolean query \( Q(\text{input}) \)
3. Classic Complexity Classes are static: FO, NC, P, NP, \ldots
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
- Surviving in the wild
### 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></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></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>0000000</td>
<td>$\text{ins}(3,S)$</td>
<td>0</td>
</tr>
</tbody>
</table>

*Dynamic Reasoning*
### 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><code>ins(3,S)</code></td>
<td>1</td>
</tr>
</tbody>
</table>

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

$b' \equiv (b \land S(a)) \lor (b' \land \neg S(a)) \land \neg b \land \neg S(a)$
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>
<tr>
<td></td>
<td><strong>ins(7,S)</strong></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Current Database: $S$</th>
<th>Request</th>
<th>Auxiliary Data: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0010000</td>
<td>ins(3,S)</td>
<td>1</td>
</tr>
<tr>
<td>0010001</td>
<td>ins(7,S)</td>
<td>0</td>
</tr>
</tbody>
</table>

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

$$b' \equiv (b \land S(a)) \lor (b \land \neg S(a)) \lor (\neg b \land \neg S(a))$$
<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>0010001</td>
<td>ins(7,S)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>del(3,S)</td>
<td></td>
</tr>
</tbody>
</table>

Parity

Dynamic Reasoning

ASL 2015 North American meeting, Urbana-Champaign
### 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>0010001</td>
<td>ins $(7, S)$</td>
<td>0</td>
</tr>
<tr>
<td>0000001</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>

<table>
<thead>
<tr>
<th>$\text{ins}(a, S)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S'(x) \equiv S(x) \lor x = a$</td>
</tr>
<tr>
<td>$b' \equiv (b \land S(a)) \lor (\neg b \land \neg S(a))$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{del}(a, S)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S'(x) \equiv S(x) \land x \neq a$</td>
</tr>
<tr>
<td>$b' \equiv (b \land \neg S(a)) \lor (\neg b \land S(a))$</td>
</tr>
</tbody>
</table>
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 $\text{FO}$, 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 FO, requires \( \text{FO}[\Omega(\log n / \log \log n)] \)
- **Dynamic:** in Dyn-FO \( \quad [\text{Patnaik, I}] \)
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 $\text{Dyn-FO}$  [Patnaik, 1]

Minimum Spanning Trees, $k$-edge connectivity, ...
**Fact:** [Dong & Su]  \( \text{REACH}(\text{acyclic}) \in \text{DynFO} \)

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

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

\[
\begin{align*}
P'(x, y) & \equiv P(x, y) \land \left[ \neg(P(x, a) \land P(b, y)) \right] \\
& \lor (\exists uv)(P(x, u) \land E(u, v) \land P(v, y) \\
& \land P(u, a) \land \neg P(v, a) \land (a \neq u \lor b \neq v)) \right]
\end{align*}
\]
REACHABILITY Problems

\[
\begin{align*}
\text{REACH} &= \{ G \mid G \text{ directed}, \; s \xrightarrow{G}^* t \} \quad \text{NL} \\
\text{REACH}_d &= \{ G \mid G \text{ directed}, \text{ outdegree} \leq 1 \; s \xrightarrow{G}^* t \} \quad \text{L} \\
\text{REACH}_u &= \{ G \mid G \text{ undirected}, \; s \xrightarrow{G}^* t \} \quad \text{L} \\
\text{REACH}_a &= \{ G \mid G \text{ alternating}, \; s \xrightarrow{G}^* t \} \quad \text{P}
\end{align*}
\]
Facts about dynamic REACHABILITY Problems:

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

\[
E'(x, y) \equiv E(x, y) \lor (x = i \land y = j)
\]
\[
D'(x) \equiv D(x) \lor x = i
\]
\[
E^*'(x, y) \equiv E^*(x, y) \lor (E^*(x, i) \land E^*(j, y))
\]
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.
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.
However reasoning about reachability in general is **undecidable**.

**Ideas:**

- 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].
For the time being, let's restrict ourselves to acyclic fields which thus also generate a linear ordering of all points reachable from a given point.

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

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

\[
\text{linear} \equiv \forall xyz \ (n^*(x, y) \land n^*(x, z) \rightarrow n^*(y, z) \lor n^*(z, y))
\]
Effectively-Propositional Reasoning about Reachability in Linked Data Structures

- Automatically transform a program manipulating linked lists to an \( \forall \exists \) correctness condition.
- Using Hesse’s dynQF algorithm for \( \text{REACH}_d \), is that these \( \forall \exists \) formulas are closed under weakest precondition.
- Using acyclic, transitive and linear axioms, the negation of the correctness condition is equi-satisfiable with a propositional formula.
- Use a SAT solver to automatically prove correctness or find counter-example runs, typically in under 3 seconds per program.
**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.

SAT was the first NP Complete problem, thus hard.
SAT was the first NP Complete problem, thus hard.
Through amazing increases of computer speed and memory, plus terrific engineering and algorithmic ideas – clause learning, and good heuristics, SAT solvers are typically incredibly fast – seconds on formulas with a million variables.
SAT was the first NP Complete problem, thus **hard**.

Through amazing increases of computer speed and memory, plus terrific engineering and algorithmic ideas – clause learning, and good heuristics, SAT solvers are typically incredibly fast – seconds on formulas with a million variables.

They provably aren’t good on all instances, but they do extremely well in practice.
SAT Solvers

- SAT was the first NP Complete problem, thus **hard**.
- Through amazing increases of computer speed and memory, plus terrific engineering and algorithmic ideas – clause learning, and good heuristics, SAT solvers are typically incredibly fast – seconds on formulas with a million variables.
- They provably aren’t good on all instances, but they do extremely well in practice.
- Thus we have a general purpose problem solver.
SAT Solvers

- SAT was the first NP Complete problem, thus **hard**.
- Through amazing increases of computer speed and memory, plus terrific engineering and algorithmic ideas – clause learning, and good heuristics, SAT solvers are typically incredibly fast – seconds on formulas with a million variables.
- They provably aren’t good on all instances, but they do extremely well in practice.
- Thus we have a general purpose problem solver.
- Very useful for checking the correctness of programs, automatically finding counter-example runs, and for synthesizing good code from specifications.
Software is everywhere, controlling more and more of our lives.
Software is everywhere, controlling more and more of our lives.

Software is buggy, insecure, brittle, hard to change.
Software Crisis

- Software is everywhere, controlling more and more of our lives.
- Software is buggy, insecure, brittle, hard to change.
- Logic and its application to automatic model checking and synthesis are – in my opinion – our best hope.
Software Crisis

- Software is everywhere, controlling more and more of our lives.
- Software is buggy, insecure, brittle, hard to change.
- Logic and its application to automatic model checking and synthesis are – in my opinion – our best hope.
- Thank you!
Dynamic Reasoning

Arithmetic Hierarchy

FO(N) r.e. complete
Halt
co-r.e. FO¬(N) r.e. complete
Halt
Recursive

FO∃(N) co-r.e. Halt

Primitive Recursive

SO(LFP) EXPTIME

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

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

PSPACE

SO(TC) SO[n^{O(1)}]

PTIME Hierarchy

co-NP complete

SAT

co-NP

SO¬

NP ∩ co-NP

NP

SO¬

NP complete

SAT

FO[n^{O(1)}]

P complete

P

FO[(log n)^{O(1)}] "truly" feasible"

NC

AC^1

sAC^1

FO(CFL)

FO(TC) SO(Krom)

2SAT NL comp.

NL

FO(DTC)

2COLOR L comp.

L

FO(REGULAR)

FO(COUNT)

FO

LOGTIME Hierarchy

AC^0

NC^1

ThC^0

Dynamic Reasoning

ASL 2015 North American meeting, Urbana-Champaign