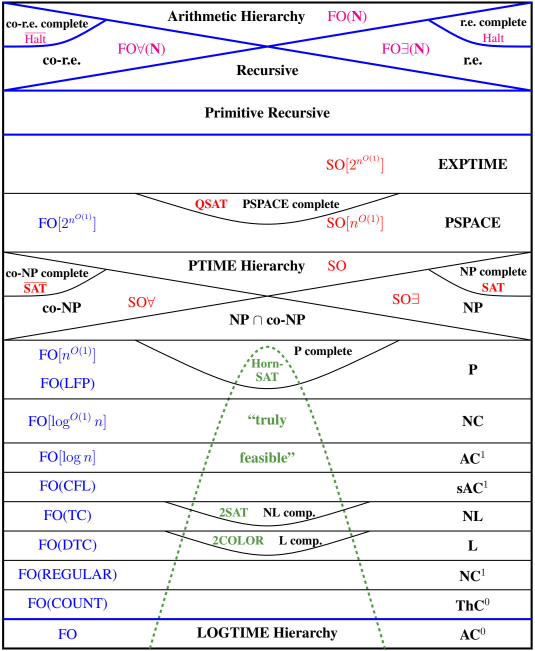
Efficiently Reasoning about Programs

Neil Immerman

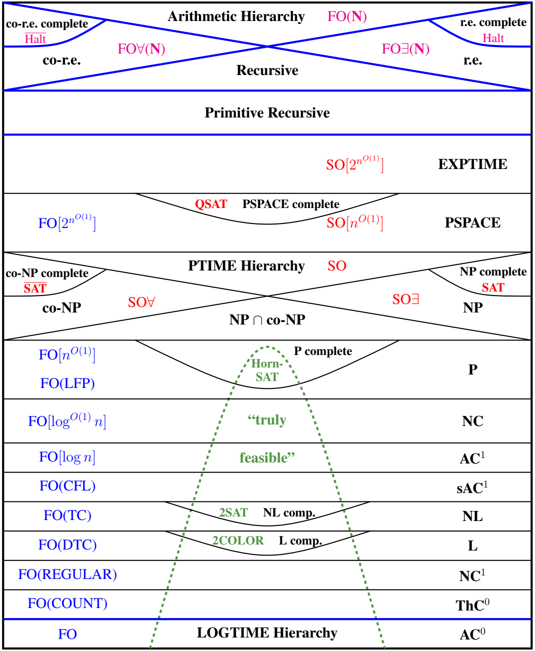
College of Computer and Information Sciences
University of Massachusetts, Amherst
Amherst, MA, USA

people.cs.umass.edu/~immerman



Arithmetic Hierarchy
FO(N)
co-r.e. complete
$\overline{\text{Halt}}$
r.e. complete
Halt
FO∀(N)
FO∃(N)
co-r.e.
r.e.
Recursive
Primitive Recursive
$\text{SO}[2^{n^{O(1)}}]$
EXPTIME
QSAT
PSPACE complete
$\text{FO}[2^{n^{O(1)}}]$
$\text{SO}[n^{O(1)}]$
PSPACE
PTIME Hierarchy
SO
co-NP complete
$\overline{\text{SAT}}$
NP complete
SAT
SO∀
SO∃
co-NP
NP
NP ∩ co-NP
$\text{FO}[n^{O(1)}]$
P complete
Horn-
SAT
FO(LFP)
P
$\text{FO}[\log^{O(1)} n]$
“truly
NC
$\text{FO}[\log n]$
feasible”
AC^1
FO(CFL)
sAC^1
FO(TC)
2SAT
NL comp.
NL
FO(DTC)
2COLOR
L comp.
L
FO(REGULAR)
NC^1
FO(COUNT)
ThC^0
FO
LOGTIME Hierarchy
AC^0

Thm. [Turing 1936] Halt undecidable.



- Halt is r.e. complete

- Halt is r.e. complete
- $\exists w \in \Sigma^* \; (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in \text{Halt}$

- Halt is r.e. complete
- $\exists w \in \Sigma^* \, (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in \text{Halt}$
- Any arbitrary search problem can be translated to Halt.

- Halt is r.e. complete
- $\exists w \in \Sigma^* \; (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in \text{Halt}$
- Any arbitrary search problem can be translated to Halt.
- **Cannot check correctness** of **arbitrary input program**.

- Halt is r.e. complete
- $\exists w \in \Sigma^* \; (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in \text{Halt}$
- Any arbitrary search problem can be translated to Halt.
- **Cannot check correctness** of **arbitrary input program**.

- **Long-Term Societal Goal:**

- Halt is r.e. complete
- $\exists w \in \Sigma^* \, (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in$ Halt
- Any arbitrary search problem can be translated to Halt.
- **Cannot check correctness** of **arbitrary input program**.

- **Long-Term Societal Goal:**
- **Automatic help** to produce programs that are

- Halt is r.e. complete
- $\exists w \in \Sigma^* \; (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in \text{Halt}$
- Any arbitrary search problem can be translated to Halt.
- **Cannot check correctness** of **arbitrary input program**.

- **Long-Term Societal Goal:**
- **Automatic help** to produce programs that are
- **certified** to **safely** and **faithfully**

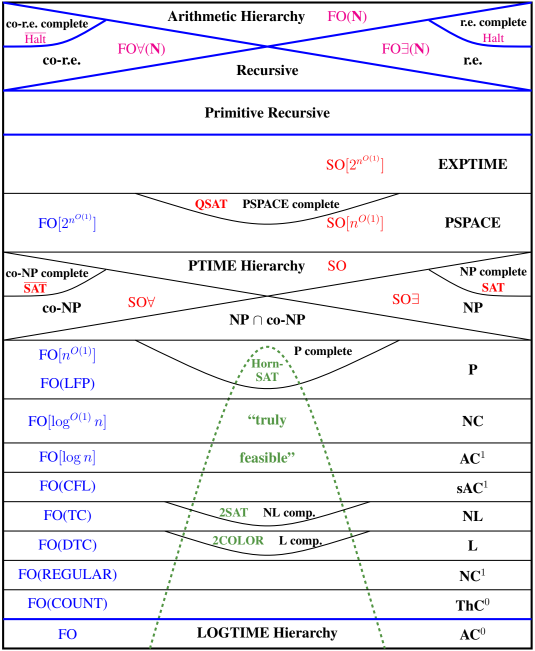
- Halt is r.e. complete
- $\exists w \in \Sigma^* \ (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in \text{Halt}$
- Any arbitrary search problem can be translated to Halt.
- **Cannot check correctness** of **arbitrary input program**.

- **Long-Term Societal Goal:**
- **Automatic help** to produce programs that are
- **certified** to **safely** and **faithfully**
- **do** what they **should do**

- Halt is r.e. complete
- $\exists w \in \Sigma^* \ (\alpha(w)) \quad \Leftrightarrow \quad M_\alpha \in$ Halt
- Any arbitrary search problem can be translated to Halt.
- **Cannot check correctness** of **arbitrary input program**.

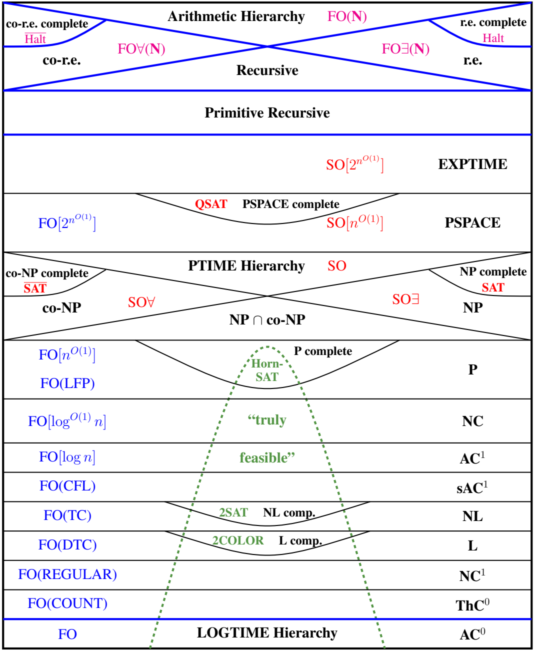
- **Long-Term Societal Goal:**
- **Automatic help** to produce programs that are
- **certified** to **safely** and **faithfully**
- **do** what they **should do**
- and **not** do what they **should not do**.

Thm. [Turing 1936] Halt undecidable.



Thm. [Turing 1936] Halt undecidable.

Thm. [Cook 1971] SAT is NP complete.



- **SAT** is NP complete.

- **SAT** is NP complete.
- $\exists w \in \Sigma^{n^{O(1)}} \, (\alpha(w)) \quad \Leftrightarrow \quad \varphi_\alpha \in$ **SAT**

- **SAT** is NP complete.
- $\exists w \in \Sigma^{n^{O(1)}} (\alpha(w)) \quad \Leftrightarrow \quad \varphi_\alpha \in$ **SAT**
- Arbitrary exponential search problem is translated to **SAT**.

- **SAT** is NP complete.
- $\exists w \in \Sigma^{n^{O(1)}} (\alpha(w)) \quad \Leftrightarrow \quad \varphi_\alpha \in$ **SAT**
- Arbitrary exponential search problem is translated to **SAT**.
- **SAT** is not **feasible** in the **worst case**.

- **SAT** is NP complete.
- $\exists w \in \Sigma^{n^{O(1)}} \, (\alpha(w)) \quad \Leftrightarrow \quad \varphi_\alpha \in$ **SAT**
- Arbitrary exponential search problem is translated to **SAT**.
- **SAT** is not **feasible** in the **worst case**.
- Every **reasonable search problem** can be encoded as an instance of **SAT**.

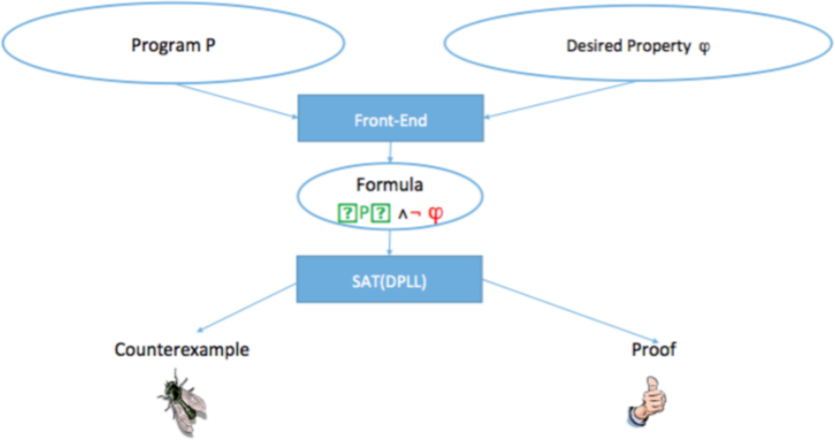
- **SAT** is NP complete.
- $\exists w \in \Sigma^{n^{O(1)}} (\alpha(w)) \quad \Leftrightarrow \quad \varphi_\alpha \in$ **SAT**
- Arbitrary exponential search problem is translated to **SAT**.
- **SAT** is not **feasible** in the **worst case**.
- Every **reasonable search problem** can be encoded as an instance of **SAT**.

- Great progress in design of **SAT Solvers**.

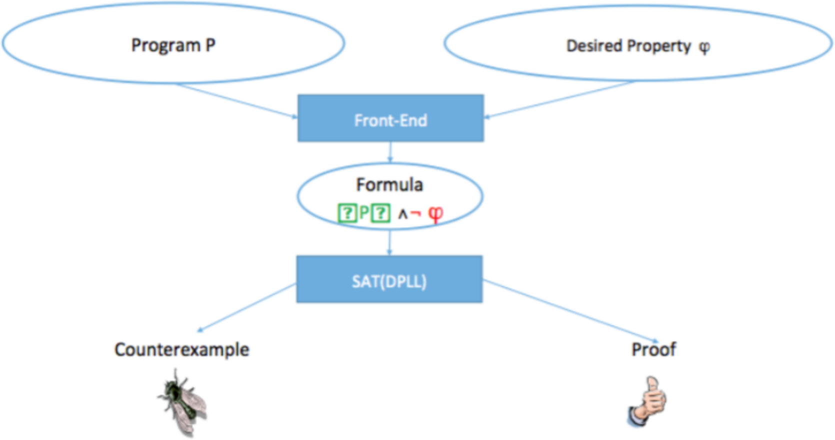
- **SAT** is NP complete.
- $\exists w \in \Sigma^{n^{O(1)}} (\alpha(w)) \quad \Leftrightarrow \quad \varphi_\alpha \in$ **SAT**
- Arbitrary exponential search problem is translated to **SAT**.
- **SAT** is not **feasible** in the **worst case**.
- Every **reasonable search problem** can be encoded as an instance of **SAT**.

- Great progress in design of **SAT Solvers**.
- **Fast**, **general-purpose problem solvers**.

Verification by Reduction to SAT

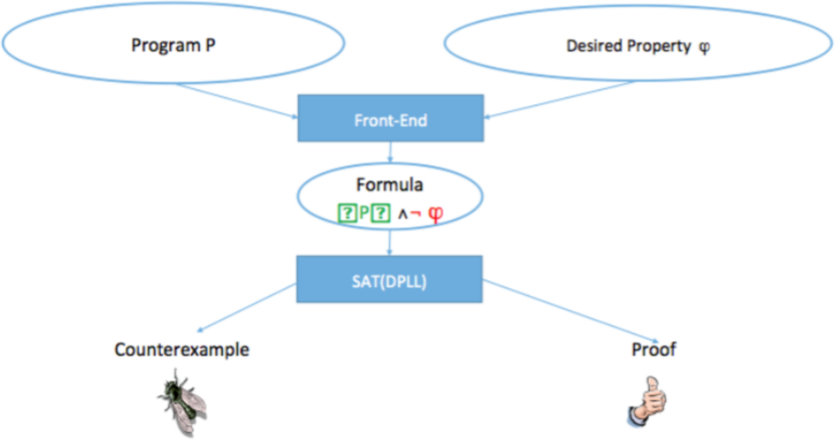


Verification by Reduction to SAT



- When and why does this work?

Verification by Reduction to SAT



- When and why does this work?
- How general and powerful can we make 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 $\in$ Dyn-FO

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)

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

$$b' \equiv (b \wedge S(a)) \vee (\neg b \wedge \neg S(a))$$

del(a,S)

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

$$b' \equiv (b \wedge \neg S(a)) \vee (\neg b \wedge S(a))$$

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

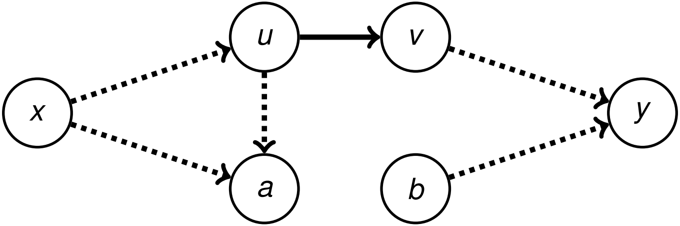
- Does binary string w have an odd number of 1's?
- **Static:** TIME[n], FO[$\Omega(\log n / \log\log n)$]
- **Dynamic:** Dyn-TIME[1], Dyn-FO

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]

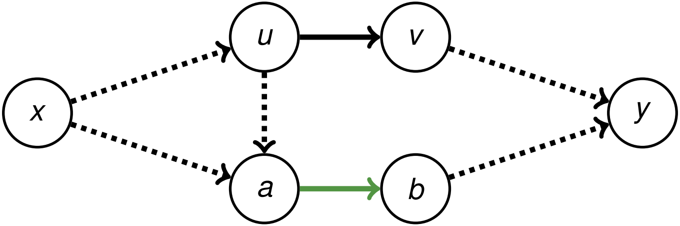
connectivity,
minimum spanning trees, **in Dyn-FO**
***k*-edge connectivity, . . .**

Fact: [Dong & Su] REACH(acyclic) $\in$ DynFO



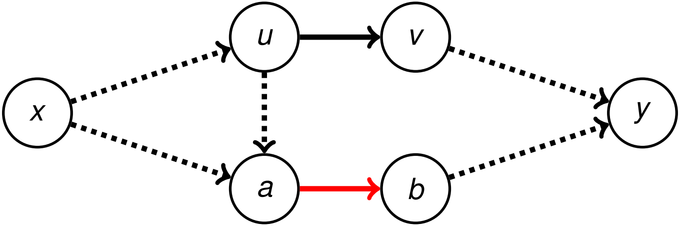
Fact: [Dong & Su] REACH(acyclic) $\in$ DynFO

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



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):

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

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:

REACH(acyclic)	$\in$	Dyn-FO	[DS]
REACH_d	$\in$	Dyn-QF	[H]
REACH_u	$\in$	Dyn-FO	[PI]
REACH	$\in$	Dyn-FO(COUNT)	[H]
$\mathrm{PAD}(\mathrm{REACH}_a)$	$\in$	Dyn-FO	[PI]

Exciting New Result

Thm. REACH $\in$ Dyn-FO

[Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, Thomas Zeume]

`http://arxiv.org/abs/1502.07467`

REACH $\leq$ **Matrix Rank** $\in$ Dyn-FO

Thm. 1 [Hesse] $\quad \text{REACH}_d(\textit{acyclic}) \in \text{Dyn-FO}$

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

ins(a, b, E): (ignore if outdegree or acyclicity violated)

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

Thm. 1 [Hesse] REACH$_d$(*acyclic*) $\in$ Dyn-FO

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

ins(a, b, E): (ignore if outdegree or acyclicity violated)

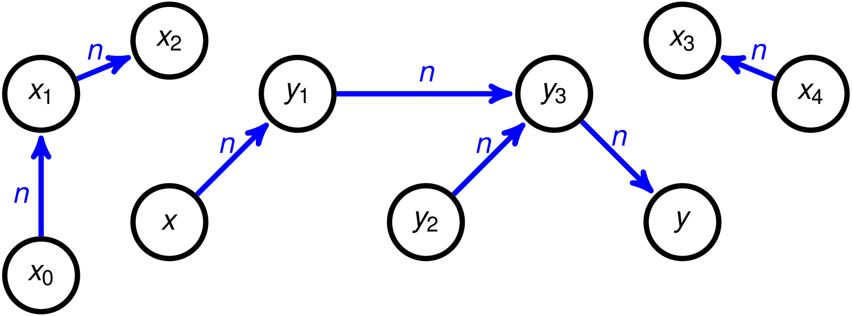
$$\begin{aligned} E'(x,y) &\equiv E(x,y) \lor (x = a \land y = b) \\ D'(x) &\equiv D(x) \lor x = a \\ E^{*\prime}(x,y) &\equiv E^*(x,y) \lor (E^*(x,a) \land E^*(b,y)) \end{aligned}$$

del(a, b, E):

$$\begin{aligned} E'(x,y) &\equiv E(x,y) \land (x \neq a \lor y \neq b) \\ D'(x) &\equiv D(x) \land x \neq a \\ E^{*\prime}(x,y) &\equiv E^*(x,y) \land \neg(E^*(x,a) \land E(a,b) \land E^*(b,y)) \end{aligned}$$

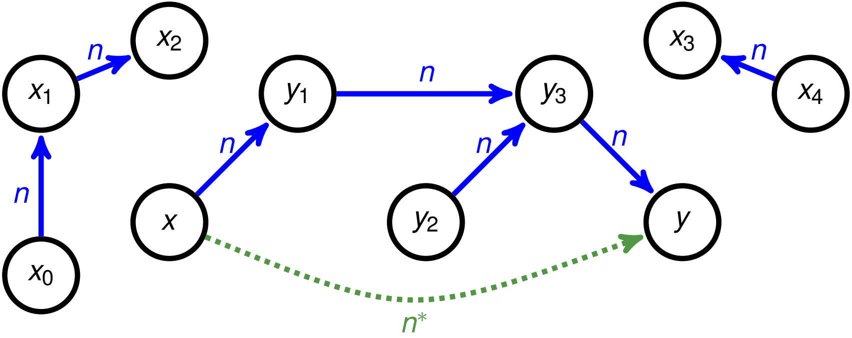
$\square$

Reasoning About reachability – can we get to y from x by following a sequence of pointers –



Dynamic Reasoning

Reasoning About reachability – can we get to y from x by following a sequence of pointers – is **crucial** for **understanding** programs and **proving** that they 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 satisfiablity of $\mathrm{FO}(\mathrm{TC})$ is as hard as the Arithmetic Hierarchy [Avron].

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.

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

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.

- For now, **restrict** to **acyclic** fields.
- $n(x, y)$ means that x points to y.

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.

- For now, **restrict** to **acyclic** fields.
- $n(x, y)$ means that x points to y.
- Use predicate symbol, n^*, **but not** n.

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.

- 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.

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.

- 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.

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

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, “Effectively-Propositional Reasoning About Reachability in Linked Data Structures” CAV 2013.

- 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.

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

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

Itzhaky, Banerjee, Immerman, Aleks Nanevski, Sagiv, "Effectively-Propositional Reasoning About Reachability in Linked Data Structures" CAV 2013.

- 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.

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

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

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

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 (**EPR**).

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 (**EPR**).
- Use a SAT solver to automatically prove correctness or find counter-example runs, typically in **only a few seconds**.

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.
- constant symbols: $c_1, \ldots, c_k$

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.
- constant symbols: $c_1, \ldots, c_k$
- $\varphi = \exists x_1 \ldots x_s \forall y_1 \ldots y_t \; (\alpha(\overline{x}, \overline{t}, \overline{c}))$

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.
- constant symbols: $c_1, \ldots, c_k$
- $\varphi = \exists x_1 \ldots x_s \forall y_1 \ldots y_t \ (\alpha(\overline{x}, \overline{t}, \overline{c}))$
- **small model**: $\varphi \in$ **FO-SAT** iff has model size $\leq k + s$.

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.
- constant symbols: $c_1, \ldots, c_k$
- $\varphi = \exists x_1 \ldots x_s \forall y_1 \ldots y_t \; (\alpha(\overline{x}, \overline{t}, \overline{c}))$
- **small model**: $\varphi \in$ **FO-SAT** iff has model size $\leq k + s$.
- **EPR-SAT** $\in \Sigma_2^p$ (2nd level polynomial-time hierarchy)

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.
- constant symbols: $c_1, \ldots, c_k$
- $\varphi = \exists x_1 \ldots x_s \forall y_1 \ldots y_t \ (\alpha(\overline{x}, \overline{t}, \overline{c}))$
- **small model**: $\varphi \in$ **FO-SAT** iff has model size $\leq k + s$.
- **EPR-SAT** $\in \Sigma_2^p$ (2nd level polynomial-time hierarchy)
- If t is fixed, then reducible to **SAT**.

Effectively-Propositional Reasoning (EPR)

- **FO-SAT** is **undecidable** (co-r.e. complete).
- **EPR**: $\exists\forall$ formulas; no function symbols.
- constant symbols: $c_1, \ldots, c_k$
- $\varphi = \exists x_1 \ldots x_s \forall y_1 \ldots y_t \ (\alpha(\overline{x}, \overline{t}, \overline{c}))$
- **small model**: $\varphi \in$ **FO-SAT** iff has model size $\leq k + s$.
- **EPR-SAT** $\in \Sigma_2^p$ (2nd level polynomial-time hierarchy)
- If t is fixed, then reducible to **SAT**.
- Z3 seems to do very well for us on **EPR-SAT**.

Benchmark	*Formula Size*						*Solving time (Z3)*
	P,Q		Inv		VC		
	#	∀	#	∀	#	∀	
SLL: reverse	2	2	11	2	133	3	57ms
SLL: filter	5	1	14	1	280	4	39ms
SLL: create	1	0	1	0	36	3	13ms
SLL: delete	5	0	12	1	152	3	23ms
SLL: deleteAll	3	2	7	2	106	3	32ms
SLL: insert	8	1	6	1	178	3	17ms
SLL: find	7	1	7	1	64	3	15ms
SLL: last	3	0	5	0	74	3	15ms
SLL: merge	14	2	31	2	2255	3	226ms
SLL: rotate	6	1	-	-	73	3	22ms
SLL: swap	14	2	-	-	965	5	26ms
DLL: fix	5	2	11	2	121	3	32ms
DLL: splice	10	2	-	-	167	4	27ms

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, 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

Extensions

- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.

Extensions

- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.
- What data structures can we handle: lists, doubly linked lists, cyclic lists; binary trees, ...

Extensions

- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.
- What data structures can we handle: lists, doubly linked lists, cyclic lists; binary trees, ...
- The [CAV13] and [POPL14] papers assume that correct invariants are given for each loop. On-going work to automatically generate and prove loop invariants:

Extensions

- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.
- What data structures can we handle: lists, doubly linked lists, cyclic lists; binary trees, ...
- The [CAV13] and [POPL14] papers assume that correct invariants are given for each loop. On-going work to automatically generate and prove loop invariants:
- Feldman, Padon, I, Sagiv, Shoham, “Bounded Quantifier Instantiation for Checking Inductive Invariants” [TACAS17]

Extensions

- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.
- What data structures can we handle: lists, doubly linked lists, cyclic lists; binary trees, ...
- The [CAV13] and [POPL14] papers assume that correct invariants are given for each loop. On-going work to automatically generate and prove loop invariants:
- Feldman, Padon, I, Sagiv, Shoham, “Bounded Quantifier Instantiation for Checking Inductive Invariants” [TACAS17]
- Padon, I, Karbyshev, Sagiv, Shoham, “Decidability of Inferring Inductive Invariants” [POPL16].

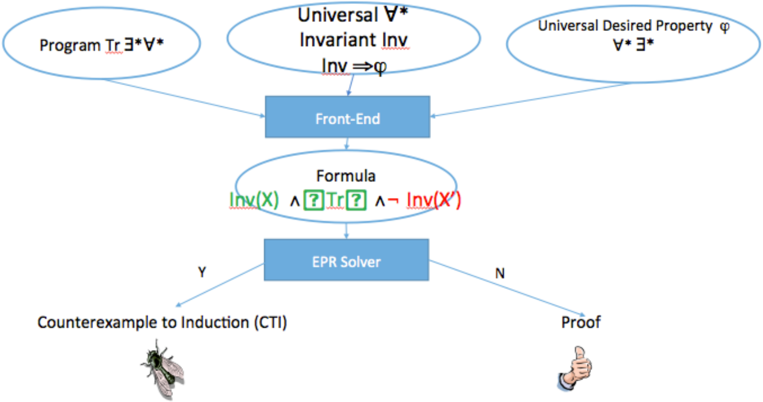
Extensions

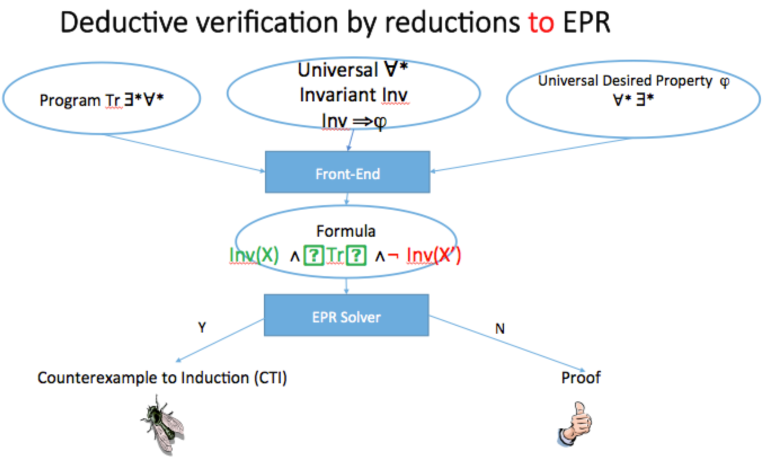
- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.
- What data structures can we handle: lists, doubly linked lists, cyclic lists; binary trees, ...
- The [CAV13] and [POPL14] papers assume that correct invariants are given for each loop. On-going work to automatically generate and prove loop invariants:
- Feldman, Padon, I, Sagiv, Shoham, “Bounded Quantifier Instantiation for Checking Inductive Invariants” [TACAS17]
- Padon, I, Karbyshev, Sagiv, Shoham, “Decidability of Inferring Inductive Invariants” [POPL16].
- Padon, McMillan, Panda, Sagiv, Shoham, “Ivy: Safety Verification by Interactive Generalization” [PLDI16].

Extensions

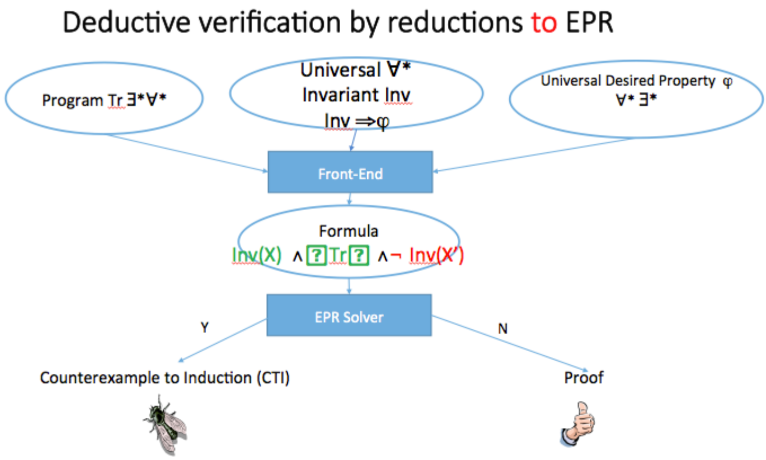
- Extensions to EPR: we can have functions symbols, as long as we can guarantee the the closure of the function symbols on any finite set remains finite.
- What data structures can we handle: lists, doubly linked lists, cyclic lists; binary trees, ...
- The [CAV13] and [POPL14] papers assume that correct invariants are given for each loop. On-going work to automatically generate and prove loop invariants:
- Feldman, Padon, I, Sagiv, Shoham, “Bounded Quantifier Instantiation for Checking Inductive Invariants” [TACAS17]
- Padon, I, Karbyshev, Sagiv, Shoham, “Decidability of Inferring Inductive Invariants” [POPL16].
- Padon, McMillan, Panda, Sagiv, Shoham, “Ivy: Safety Verification by Interactive Generalization” [PLDI16].
- Karbyshev, Bjorner, Itzhaky, Rinetzky, Shoham, “Property-Directed Inference of Universal Invariants or Proving Their Absence” [CAV15].

Deductive verification by reductions to EPR



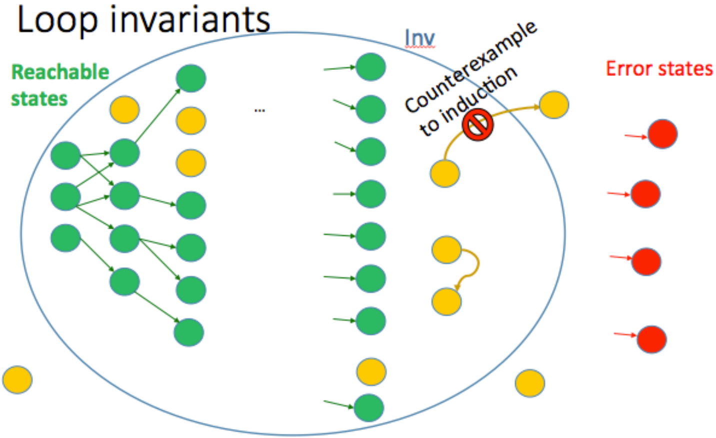


- When does this work?



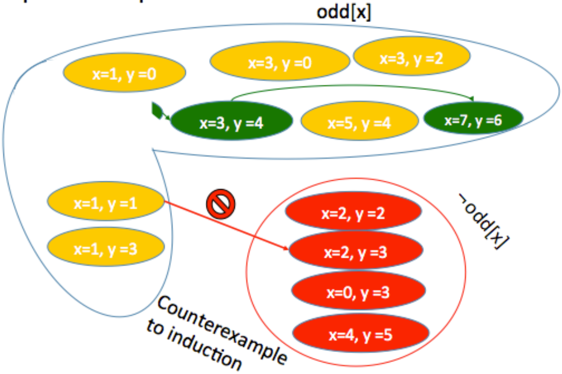
- When does this work?
- When doesn't this work?

- Init $\rightarrow$ Inv; Inv $\wedge$Tr $\rightarrow$ Inv$'$; Inv $\rightarrow$ Safe



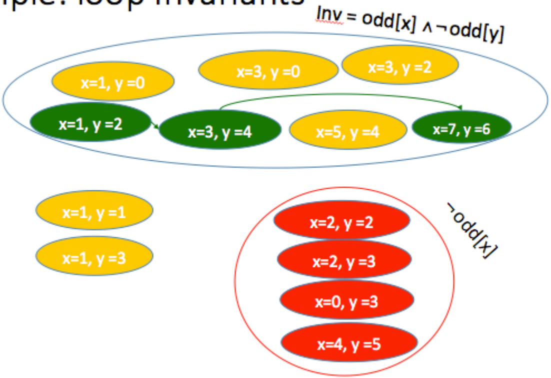
Simple Example: loop Invariants

```
1: x := 1;
2: y := 2;
while * do {
  3: assert odd[x];
  4:  x:= x + y;
  5:  y := y + 2
}
6:
```



Simple Example: loop Invariants

```
1: x := 1;
2: y := 2;
while * do {
  3: assert odd[x];
  4:  x:= x + y;
  5:  y := y + 2
}
6:
```



- **Herbrand Thm.** φ universal $\Rightarrow$

  $$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Herbrand Thm.** φ universal $\Rightarrow$

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Cor.** Complete FO-UNSATmethodology:

- **Herbrand Thm.** φ universal $\Rightarrow$

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Cor.** Complete FO-UNSATmethodology:
- Skolemize φ: φ_S is universal: $\quad \varphi_S = \forall \overline{x}\ (\alpha(\overline{x}))$;

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi_S \in \text{FO-SAT}$$

- **Herbrand Thm.** φ universal $\Rightarrow$

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Cor.** Complete FO-UNSATmethodology:
- Skolemize φ: φ_S is universal: $\quad \varphi_S = \forall \overline{x}\ (\alpha(\overline{x}))$;

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi_S \in \text{FO-SAT}$$

- $\text{grnd}(\alpha) \stackrel{\text{def}}{=} \{\alpha(\bar{t}) \mid \bar{t} \in |\mathcal{H}|\}$

- **Herbrand Thm.** φ universal $\Rightarrow$

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Cor.** Complete FO-UNSATmethodology:
- Skolemize φ: φ_S is universal: $\varphi_S = \forall \overline{x}\ (\alpha(\overline{x}))$;

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi_S \in \text{FO-SAT}$$

- $\text{grnd}(\alpha) \stackrel{\text{def}}{=} \{\alpha(\bar{t}) \mid \bar{t} \in |\mathcal{H}|\}$
- $\varphi \in \text{FO-UNSAT} \quad \Leftrightarrow \quad \text{grnd}(\alpha) \in \text{UNSAT}$

- **Herbrand Thm.** φ universal $\Rightarrow$

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Cor.** Complete FO-UNSATmethodology:
- Skolemize φ: φ_S is universal: $\varphi_S = \forall \bar{x}\ (\alpha(\bar{x}))$;

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi_S \in \text{FO-SAT}$$

- $\text{grnd}(\alpha) \stackrel{\text{def}}{=} \{\alpha(\bar{t}) \mid \bar{t} \in |\mathcal{H}|\}$
- $\varphi \in \text{FO-UNSAT} \quad \Leftrightarrow \quad \text{grnd}(\alpha) \in \text{UNSAT}$
- Feldman, Padon, I, Sagiv, Shoham, “Bounded Quantifier Instantiation for Checking Inductive Invariants” [TACAS17]

- **Herbrand Thm.** φ universal $\Rightarrow$

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi \text{ has Herbrand model, } \mathcal{H} \models \varphi$$

- **Cor.** Complete FO-UNSATmethodology:
- Skolemize φ: φ_S is universal: $\quad \varphi_S = \forall \bar{x}\ (\alpha(\bar{x}))$;

$$\varphi \in \text{FO-SAT} \quad \Leftrightarrow \quad \varphi_S \in \text{FO-SAT}$$

- $\text{grnd}(\alpha) \stackrel{\text{def}}{=} \{\alpha(\bar{t}) \mid \bar{t} \in |\mathcal{H}|\}$
- $\varphi \in \text{FO-UNSAT} \quad \Leftrightarrow \quad \text{grnd}(\alpha) \in \text{UNSAT}$
- Feldman, Padon, I, Sagiv, Shoham, "Bounded Quantifier Instantiation for Checking Inductive Invariants" [TACAS17]
- Can Understand Decidability of Checking FO Inductive Invariants, via bounded depth of nesting of functions in $\bar{t}$ needed for unsatisfiability.

Thank You!

Anindya Banerjee, Bill Hesse,

Yotam Feldman, Shachar Itzhaky,

Aleksandr Karbyshev, Ori Lahav,

Aleksandar Nanevski, Oded Padon,

Sushant Patnaik, Mooly Sagiv,

Sharon Shoham