Inductive Data Flow Graphs

Azadeh Farzan\textsuperscript{1} \hspace{2cm} \textbf{Zachary Kincaid}\textsuperscript{1} \hspace{1cm} Andreas Podelski\textsuperscript{2}

\textsuperscript{1}University of Toronto \hspace{1cm} \textsuperscript{2}University of Freiburg

January 23, 2013
Algorithmic verification

Goal

Given a (concurrent) program $P$ and a specification $\varphi_{\text{pre}}/\varphi_{\text{post}}$, prove

$$\{ \varphi_{\text{pre}} \} P \{ \varphi_{\text{post}} \}$$

(or provide a counter-example)
Goal

Given a (concurrent) program $P$ and a specification $\varphi_{pre}/\varphi_{post}$, prove

$$\{\varphi_{pre}\} P \{\varphi_{post}\}$$

(or provide a counter-example)

- **Static analysis** for sequential programs
Algorithmic verification

Goal

Given a (concurrent) program $P$ and a specification $\varphi_{\text{pre}}/\varphi_{\text{post}}$, prove

$$\{\varphi_{\text{pre}}\} P \{\varphi_{\text{post}}\}$$

(or provide a counter-example)

- Static analysis for sequential programs
- Model checking for finite-state concurrent protocols
Algorithmic verification

Goal

Given a (concurrent) program \( P \) and a specification \( \varphi_{\text{pre}}/\varphi_{\text{post}} \), prove

\[
\{\varphi_{\text{pre}}\} P \{\varphi_{\text{post}}\}
\]

(or provide a counter-example)

- **Static analysis** for sequential programs
- **Model checking** for finite-state concurrent protocols

This talk presents **Inductive Data Flow Graphs** (iDFGs): a form of correctness proof for (concurrent) programs
Why iDFGs?

There are many proof systems: Floyd/Hoare, Owicki-Gries, Rely/Guarantee. Why do we want a new one?
There are many proof systems: Floyd/Hoare, Owicki-Gries, Rely/Guarantee. Why do we want a new one?

- **Succinct**
  - Present only the *essence* of a proof
  - Polynomial in the data complexity of a program
There are many proof systems: Floyd/Hoare, Owicki-Gries, Rely/Guarantee. Why do we want a new one?

- **Succinct**
  - Present only the *essence* of a proof
  - Polynomial in the data complexity of a program

- Can be **generated and checked automatically**
  - Extend *static analysis* to concurrent control
  - Extend *model checking* to (unbounded) data
“Essence” of a proof

\[ \varphi_{\text{pre}} : x \geq 0 \land y \geq 0 \]

Thread 1

\[ x \, \text{++} \]
\[ y \, \text{++} \]
\[ z = x + y \]

\[ \varphi_{\text{post}} : z \geq 2 \]

Thread 2

\[ x = 2 \]
\[ y \, \text{++} \]
\[ z = x + y \]

\[ \begin{cases} x \geq 0 \land y \geq 0 \\ y \geq 0 \land x \geq 1 \land y \geq 0 \land z \geq 2 \end{cases} \]
“Essence” of a proof

\[ \varphi_{\text{pre}} : x \geq 0 \land y \geq 0 \]

Thread 1

\[
\begin{align*}
x & \text{ ++} \\
y & \text{ ++} \\
z & = x + y
\end{align*}
\]

\[ \varphi_{\text{post}} : z \geq 2 \]

Thread 2

\[
\begin{align*}
x & = 2 \\
x & \text{ ++} \\
\{ x \geq 1 \} & \rightarrow \{ y \geq 0 \}
\end{align*}
\]

\[
\begin{align*}
y & \text{ ++} \\
\{ y \geq 1 \} & \rightarrow \{ x \geq 1 \}
\end{align*}
\]

\[
\begin{align*}
z & = x + y \\
z & \geq 2
\end{align*}
\]

\[ x \geq 0 \land y \geq 0 \]
“Essence” of a proof

$\varphi_{\text{pre}} : x \geq 0 \land y \geq 0$

Thread 1

- $x \mathbin{\mathbf{++}}$
- $y \mathbin{\mathbf{++}}$
- $z = x + y$

$\varphi_{\text{post}} : z \geq 2$

Thread 2

- $x = 2$
- $x \mathbin{\mathbf{++}}$
- $y \mathbin{\mathbf{++}}$

$z = x + y$

$x \geq 0 \land y \geq 0$

Independent conditions
“Essence” of a proof

$\varphi_{\text{pre}} : x \geq 0 \land y \geq 0$

Thread 1

\[ \begin{align*}
  x &\; \mathbf{++} \\
  y &\; \mathbf{++} \\
  z &\; = \; x + y
\end{align*} \]

$\varphi_{\text{post}} : z \geq 2$

Thread 2

\[ \begin{align*}
  x &\; = \; 2 \\
  y &\; \mathbf{++} \\
  z &\; = \; x + y
\end{align*} \]

$\{x \geq 1\}$

Irrelevant

$\{y \geq 0\}$

$\{y \geq 1\}$

Independent conditions
"Essence" of a proof

$\varphi_{\text{pre}}: x \geq 0 \land y \geq 0$

Thread 1

x ++
y ++
z = x + y

$\varphi_{\text{post}}: z \geq 2$

Thread 2

x = 2
y ++
z = x + y

$\varphi_{\text{pre}}: x \geq 0 \land y \geq 0$

Thread 1

x ++
y ++
z = x + y

$\varphi_{\text{post}}: z \geq 2$

Thread 2

x = 2
y ++
z = x + y

“Essence” of a proof

$\varphi_{\text{pre}} : x \geq 0 \land y \geq 0$

Thread 1

$x \; +\!+\!+$
$y \; +\!+\!+$
$z = x + y$

$\varphi_{\text{post}} : z \geq 2$

Thread 2

$x \; = \; 2$

- $x \; +\!+\!+$
- $y \; +\!+\!+$
- $x = 2$

- $z = x + y$

- $z \geq 2$
Inductive Data Flow Graphs (iDFGs)

Inductiveness condition:

\[ \psi_1 \leq \cdots \leq \psi_m \leq \text{cmd}_a \{ \phi_j \} \]

for all \( j \), \( \{ \psi_1 \wedge \cdots \wedge \psi_m \} \text{cmd}_a \{ \phi_j \} \)

\( \{ x \geq 0 \wedge y \geq 0 \} \)

\( \text{init} \)

\( \{ \text{true} \} \)

\( \{ x = 2 \} \)

\( \{ x \geq 1 \} \)

\( \{ y \geq 0 \} \)

\( \{ y \geq 1 \} \)

\( z = x + y \)

\( \{ z \geq 2 \} \)

\( x = 2 \)

\( y ++ \)

\( z = x + y \)

Supress irrelevant details of a partial correctness proof

- Irrelevant ordering constraints
  \( (x = 2; y ++ \text{ vs } y ++; x = 2) \)

- Irrelevant actions \((x ++)\)
Inductive Data Flow Graphs (iDFGs)

Inductiveness condition:

\[ \text{for all } j, \{ \psi_1 \land \cdots \land \psi_m \}\text{cmd}_a\{ \varphi_j \} \]

\[ \{x \geq 0 \land y \geq 0\} \]

Parallelize a partial correctness proof

- Irrelevant ordering constraints
  \[(x = 2; y ++ \text{ vs } y ++; x = 2)\]
- Irrelevant actions \((x ++)\)
Denotation of an iDFG

Data flow graph with inductive assertions (iDFG) proves correctness of traces that obey particular constraints

~ Control flow graph with inductive assertions (Floyd annotation) proves correctness of traces that label paths

The set of such traces is called the denotation of the iDFG, denoted $[G]$. 
Denotation of an iDFG

Data flow graph with inductive assertions (iDFG) proves correctness of traces that obey particular constraints

\[ \varphi_{\text{pre}} : x \geq 0 \land y \geq 0 \]

Thread 1

\[
\begin{align*}
x & \leftarrow x \\
y & \leftarrow y \\
z & \leftarrow x + y
\end{align*}
\]

\[ \varphi_{\text{post}} : z \geq 2 \]

Thread 2

\[
\begin{align*}
x & \leftarrow x \\
y & \leftarrow y
\end{align*}
\]

\[ \{x \geq 0 \land y \geq 0\} \]

Control flow graph with inductive assertions (Floyd annotation) proves correctness of traces that label paths

The set of such traces is called the **denotation** of the iDFG, denoted \([G]\).
Data flow graph with inductive assertions (iDFG) proves correctness of traces that obey particular constraints

~ Control flow graph with inductive assertions (Floyd annotation) proves correctness of traces that label paths

The set of such traces is called the denotation of the iDFG, denoted \( [G] \).

\[
\varphi_{\text{pre}} : x \geq 0 \land y \geq 0
\]

Thread 1
\[
x ++
y ++
z = x + y
\]

\[
\varphi_{\text{post}} : z \geq 2
\]

Thread 2
\[
x = 2
\]

\[
\{x \geq 0 \land y \geq 0\}
\]

\[
\{x \geq 0\}
\]

\[
\{y \geq 0\}
\]

\[
\{x \geq 1\}
\]

\[
\{y \geq 1\}
\]

\[
\{z \geq 2\}
\]
Denotation of an iDFG

Data flow graph with inductive assertions (iDFG) proves correctness of traces that obey particular constraints

~ Control flow graph with inductive assertions (Floyd annotation) proves correctness of traces that label paths

The set of such traces is called the **denotation** of the iDFG, denoted $[G]$.  

\[
\begin{align*}
\varphi_{\text{pre}} & : x \geq 0 \land y \geq 0 \\
\text{Thread 1} & \\
\text{Thread 2} & \\
x \text{ ++} & \\
y \text{ ++} & \\
z = x + y & \\
\varphi_{\text{post}} & : z \geq 2
\end{align*}
\]
**Theorem**

Let \( G = \langle V, E, \varphi_{\text{pre}}, \varphi_{\text{post}}, v_0, V_{\text{final}} \rangle \) be an iDFG. For all \( \tau \in \llbracket G \rrbracket, \)

\[
\{ \varphi_{\text{pre}} \} \tau \{ \varphi_{\text{post}} \}
\]
Theorem

Let $G = \langle V, E, \varphi_{\text{pre}}, \varphi_{\text{post}}, v_0, V_{\text{final}} \rangle$ be an iDFG. For all $\tau \in \llbracket G \rrbracket$,

$$\{ \varphi_{\text{pre}} \} \tau \{ \varphi_{\text{post}} \}$$

Program $P \sim$ finite automaton, $\mathcal{L}(P)$ is the set of traces of $P$. 
Theorem

Let $G = \langle V, E, \varphi_{\text{pre}}, \varphi_{\text{post}}, v_0, V_{\text{final}} \rangle$ be an iDFG. For all $\tau \in \lbrack G \rbrack$,

$$\{\varphi_{\text{pre}}\} \tau \{\varphi_{\text{post}}\}$$

Program $P \sim$ finite automaton, $\mathcal{L}(P)$ is the set of *traces* of $P$.

Proof rule

Program $P$ is correct w.r.t. $\varphi_{\text{pre}} / \varphi_{\text{post}}$ iff $\exists G. \mathcal{L}(P) \subseteq \lbrack G \rbrack$
If there exists a small proof that $P$ is correct (w.r.t. $\phi_{\text{pre}}/\phi_{\text{post}}$), then exists a small iDFG proof.
If there exists a small proof that $P$ is correct (w.r.t. $\varphi_{pre}/\varphi_{post}$), then exists a small iDFG proof

**Theorem**

For any $\{\varphi_{pre}\}P\{\varphi_{post}\}$, there exists a iDFG proof with size polynomial in the data complexity of $P$, $\varphi_{pre}$, $\varphi_{post}$
Data complexity

If there exists a small proof that $P$ is correct (w.r.t. $\varphi_{\text{pre}}/\varphi_{\text{post}}$), then exists a small iDFG proof

**Theorem**

For any $\{\varphi_{\text{pre}}\} P \{\varphi_{\text{post}}\}$, there exists a iDFG proof with size polynomial in the data complexity of $P, \varphi_{\text{pre}}, \varphi_{\text{post}}$

Data complexity measures how difficult a property is to prove.

- Minimum # of assertions in a localized proof that $\{\varphi_{\text{pre}}\} P \{\varphi_{\text{post}}\}$
  - Localized proofs: expose “how compositional” a Floyd proof is.
Pick a program trace $\tau$. Is $\tau$ correct?

If $\mathcal{L}(P) \subseteq \llbracket G \rrbracket$ then construct an iDFG $G_\tau$ from $\tau$. Merge $G_\tau$ into $G$. If yes, program is correct. If no, program is incorrect.
Pick a program trace $\tau$. Is $\tau$ correct? 

- If yes, Construct an iDFG $G_\tau$ from $\tau$. Merge $G_\tau$ into $G$. Program is correct.
- If no, Construct an iDFG $G_\tau$ from $\tau$. Program is incorrect.
**Goal**

Given a trace $\tau \in \mathcal{L}(P)$ with $\{\varphi_{\text{pre}}\} \tau \{\varphi_{\text{post}}\}$, construct an iDFG $G_\tau$ with $\tau \in \llbracket G_\tau \rrbracket$.

\[
\begin{align*}
  x & \quad \text{++} \\
  x & \quad = \quad 2 \\
  y & \quad \text{++} \\
  z & \quad = \quad x + y \\
  \{z \geq 2\}
\end{align*}
\]
Given a trace $\tau \in \mathcal{L}(P)$ with $\{\varphi_{\text{pre}}\} \tau \{\varphi_{\text{post}}\}$, construct an iDFG $G_\tau$ with $\tau \in \left[ G_\tau \right]$. 

\[
\begin{align*}
\text{Goal} \\
x &= 2 \\
\text{z} &= \text{x} + \text{y} \\
\text{y} &= 2 \\
\end{align*}
\]
Goal

Given a trace $\tau \in \mathcal{L}(P)$ with $\{\varphi_{\text{pre}}\} \tau \{\varphi_{\text{post}}\}$, construct an iDFG $G_\tau$ with $\tau \in [G_\tau]$.
Goal

Given a trace $\tau \in \mathcal{L}(P)$ with $\{\varphi_{\text{pre}}\} \tau \{\varphi_{\text{post}}\}$, construct an iDFG $G_\tau$ with $\tau \in \mathcal{F}(G_\tau)$.
Pick a program trace $\tau$. 

Is $\tau$ correct? 

Construct an iDFG $G_\tau$ from $\tau$. 

L$(P) \subseteq [G]$ 

Program is correct. 

Merge $G_\tau$ into $G$. 

Program is incorrect.
Merging iDFGs

Goal

Given iDFGs $G_1, G_2$, construct an iDFG $G_1 \land G_2$ such that

$$[G_1] \cup [G_2] \subseteq [G_1 \land G_2]$$
Merging iDFGs

Goal

Given iDFGs $G_1$, $G_2$, construct an iDFG $G_1 \land G_2$ such that

$$ \lbrack G_1 \rbrack \cup \lbrack G_2 \rbrack \subseteq \lbrack G_1 \land G_2 \rbrack $$
Merging iDFGs

Goal

Given iDFGs $G_1$, $G_2$, construct an iDFG $G_1 \otimes G_2$ such that

$$[G_1] \cup [G_2] \subseteq [G_1 \otimes G_2]$$
Merging iDFGs

Goal

Given iDFGs $G_1$, $G_2$, construct an iDFG $G_1 \bigwedge G_2$ such that

$$[G_1] \cup [G_2] \subseteq [G_1 \bigwedge G_2]$$
Pick a program trace $\tau$. 

Is $\tau$ correct? 

$\mathcal{L}(P) \subseteq \llbracket G \rrbracket$ 

Construct an iDFG $G_\tau$ from $\tau$. 

Merge $G_\tau$ into $G$. 

Program is correct. 

Program is incorrect.
For any iDFG $G$, we can efficiently (linear time, in the size of $G$) construct an alternating finite automaton $A_G$ such that

$$\mathcal{L}(A_G) = \mathcal{L}(G)^{rev}$$

Proof checking: $\mathcal{L}(P)^{rev} \subseteq \mathcal{L}(A_G)$

- Can be solved in PSPACE
- Combinatorial problem (non-reachability)
- Reuse techniques from (finite-state) model checking
Summary

• **Inductive Data Flow Graphs** are a proof method for partial correctness of (concurrent) programs
• (Provably) succinct
• Can be generated automatically
Summary

• **Inductive Data Flow Graphs** are a proof method for partial correctness of (concurrent) programs
• (Provably) **succinct**
• Can be **generated automatically**

Future work

• Can iDFGs be constructed more effectively?
• Efficient proof checking?
• Parameterized programs?
• Weak memory models?
Questions?

Thank you for your attention.