Inductive Data Flow Graphs

Azadeh Farzan[1] **Zachary Kincaid**[1] Andreas Podelski[2]

[1]University of Toronto
[2]University of Freiburg

January 23, 2013

Algorithmic verification

Goal

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

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

(or provide a counter-example)

Algorithmic verification

Goal

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

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

(or provide a counter-example)

- *Static analysis* for sequential programs

Algorithmic verification

Goal

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

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

(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 $\varphi_{\mathbf{pre}}/\varphi_{\mathbf{post}}$, prove

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

(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?

Why iDFGs?

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

Why iDFGs?

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

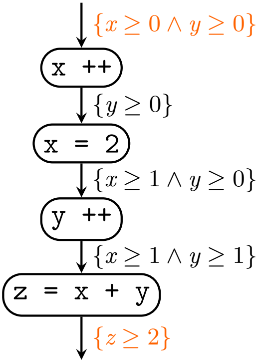
“Essence” of a proof

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

Thread 1		Thread 2

`x ++`
`y ++`
`z = x + y`
	‖	`x = 2`

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

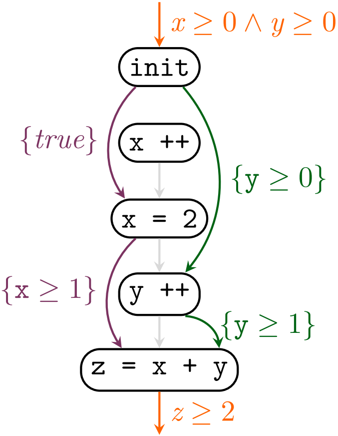


“Essence” of a proof

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

Thread 1	Thread 2
`x ++`	`x = 2`
`y ++`	
`z = x + y`	

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

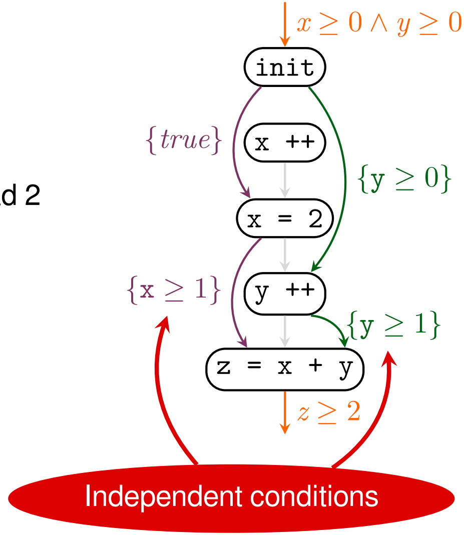


“Essence” of a proof

$\varphi_{\mathbf{pre}} : \mathbf{x} \geq 0 \wedge \mathbf{y} \geq 0$

Thread 1	Thread 2
`x ++`	`x = 2`
`y ++`	
`z = x + y`	

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

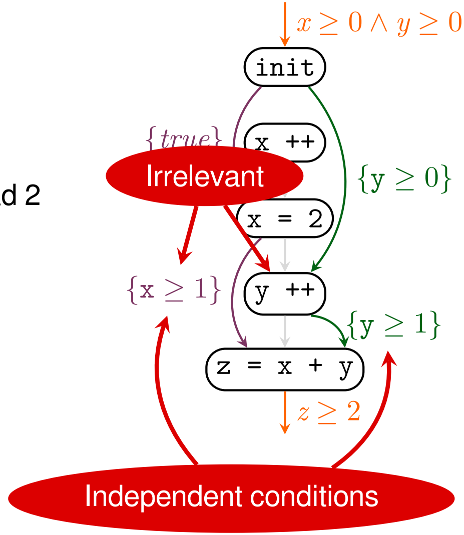


“Essence” of a proof

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

Thread 1	Thread 2
`x ++`	`x = 2`
`y ++`	
`z = x + y`	

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



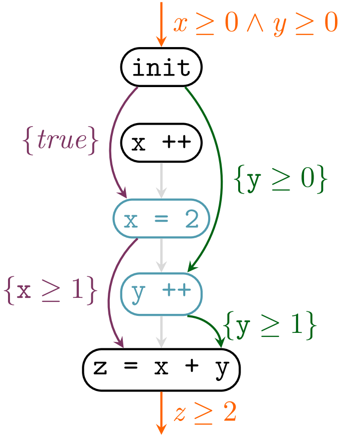
"Essence" of a proof

$\varphi_{\mathbf{pre}} : \mathbf{x} \geq 0 \wedge \mathbf{y} \geq 0$

Thread 1	Thread 2

`x ++`
`y ++`
`z = x + y`
	`x = 2`

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

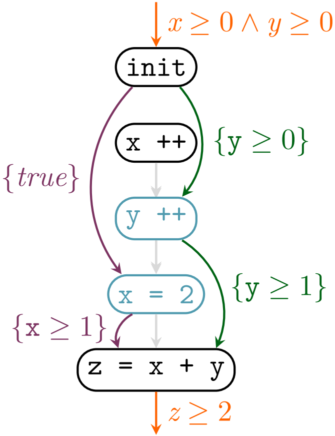


"Essence" of a proof

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

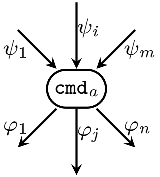
Thread 1	Thread 2
`x ++`	`x = 2`
`y ++`	
`z = x + y`	

$\varphi_{\mathbf{post}} : \mathtt{z} \geq 2$

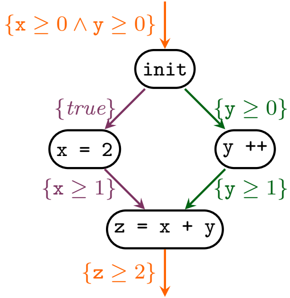


Inductive Data Flow Graphs (iDFGs)

Inductiveness condition:



for all j, $\{\psi_1 \land \cdots \land \psi_m\}\mathtt{cmd}_a\{\varphi_j\}$



Suppress irrelevant details of a partial correctness proof

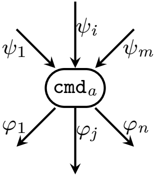
- Irrelevant ordering constraints

  (`x = 2;y ++` vs `y ++;x = 2`)

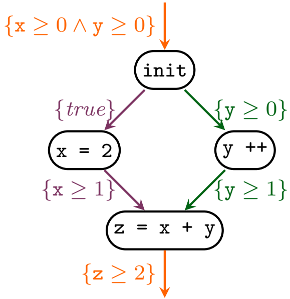
- Irrelevant actions (`x ++`)

Inductive Data Flow Graphs (iDFGs)

Inductiveness condition:



for all j, $\{\psi_1 \wedge \cdots \wedge \psi_m\}\mathtt{cmd}_a\{\varphi_j\}$



Parallelize a partial correctness proof

- Irrelevant ordering constraints

  (`x = 2;y ++` 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

$\sim$ 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

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

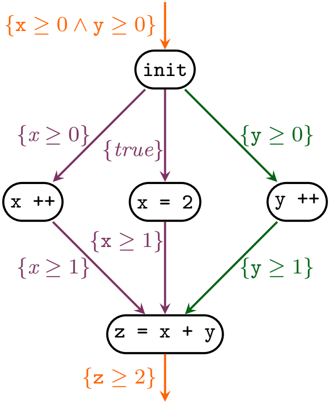
The set of such traces is called the **denotation** of the iDFG, denoted $[\![G]\!]$.

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

Thread 1	Thread 2

`x ++`
`y ++`
`z = x + y`
	`x = 2`

$\varphi_{\mathbf{post}} : \mathtt{z} \geq 2$



Denotation of an iDFG

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

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

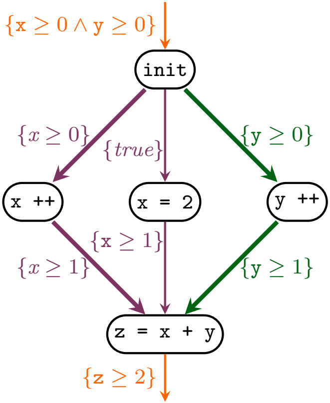
The set of such traces is called the **denotation** of the iDFG, denoted $[\![G]\!]$.

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

Thread 1	Thread 2

`x ++`
`y ++`
`z = x + y`
	`x = 2`

$\varphi_{\mathbf{post}} : \mathtt{z} \geq 2$



Denotation of an iDFG

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

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

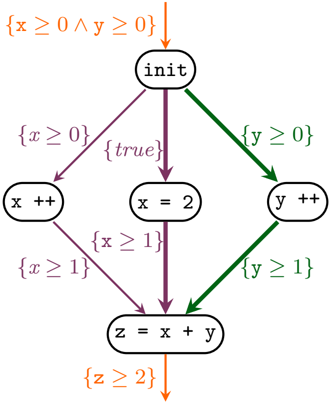
The set of such traces is called the **denotation** of the iDFG, denoted $[\![G]\!]$.

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

Thread 1	Thread 2

`x ++`
`y ++`
`z = x + y`
	`x = 2`

$\varphi_{\mathbf{post}} : \mathtt{z} \geq 2$



iDFGs as proof objects

Theorem

Let $G = \langle V, E, \varphi_{\mathbf{pre}}, \varphi_{\mathbf{post}}, v_o, V_{final} \rangle$ be an iDFG. For all $\tau \in [\![G]\!]$,

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

iDFGs as proof objects

Theorem

Let $G = \langle V, E, \varphi_{\mathbf{pre}}, \varphi_{\mathbf{post}}, v_o, V_{final} \rangle$ be an iDFG. For all $\tau \in [\![G]\!]$,

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

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

iDFGs as proof objects

Theorem

Let $G = \langle V, E, \varphi_{\mathbf{pre}}, \varphi_{\mathbf{post}}, v_o, V_{final} \rangle$ be an iDFG. For all $\tau \in [\![G]\!]$,

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

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

Proof rule

Program P is correct w.r.t. $\varphi_{\mathbf{pre}}/\varphi_{\mathbf{post}}$ iff $\exists G.\mathcal{L}(P) \subseteq [\![G]\!]$

Data complexity

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

Data complexity

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

Theorem

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

Data complexity

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

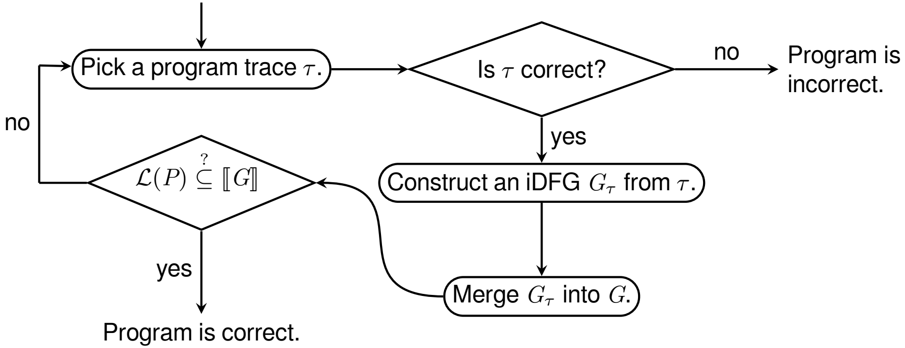
Theorem

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

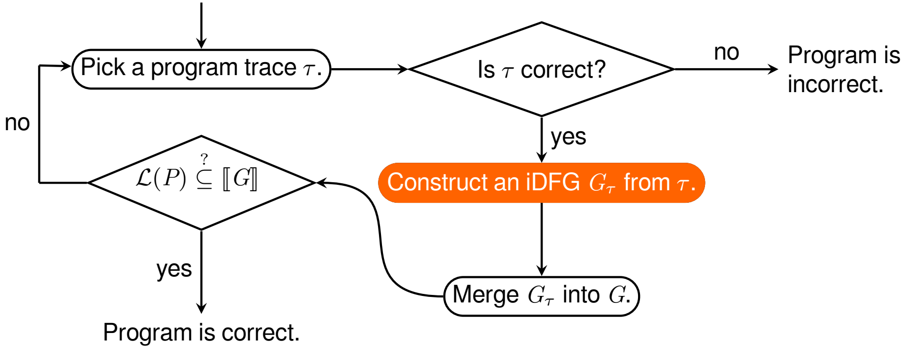
Data complexity measures how difficult a property is to prove.

- Minimum # of assertions in a *localized proof* that $\{\varphi_{\mathbf{pre}}\}P\{\varphi_{\mathbf{post}}\}$
  - Localized proofs: expose *"how compositional"* a Floyd proof is.

Automation


Pick a program trace τ.
Is τ correct?
no
Program is incorrect.
yes
Construct an iDFG G_τ from τ.
Merge G_τ into G.
$\mathcal{L}(P) \overset{?}{\subseteq} [\![G]\!]$
no
yes
Program is correct.

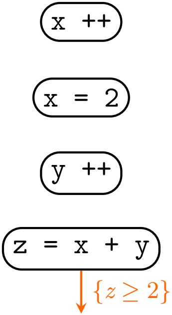
Automation


Pick a program trace τ.
Is τ correct?
no
Program is incorrect.
yes
Construct an iDFG G_τ from τ.
Merge G_τ into G.
$\mathcal{L}(P) \overset{?}{\subseteq} [\![G]\!]$
no
yes
Program is correct.

iDFG construction

Goal

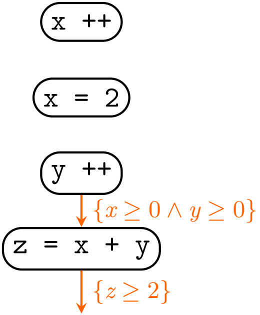
Given a trace $\tau \in \mathcal{L}(P)$ with $\{\varphi_{\mathbf{pre}}\}\tau\{\varphi_{\mathbf{post}}\}$, construct an iDFG G_τ with $\tau \in [\![G_\tau]\!]$.



iDFG construction

Goal

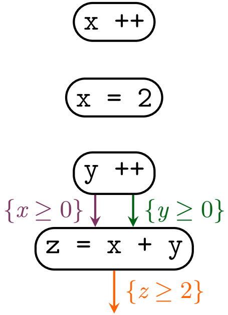
Given a trace $\tau \in \mathcal{L}(P)$ with $\{\varphi_{\mathbf{pre}}\}\tau\{\varphi_{\mathbf{post}}\}$, construct an iDFG G_τ with $\tau \in [\![G_\tau]\!]$.



iDFG construction

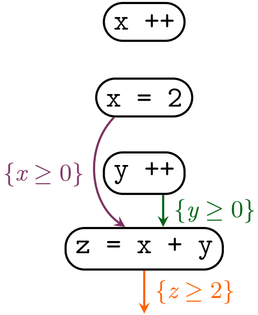
Goal

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

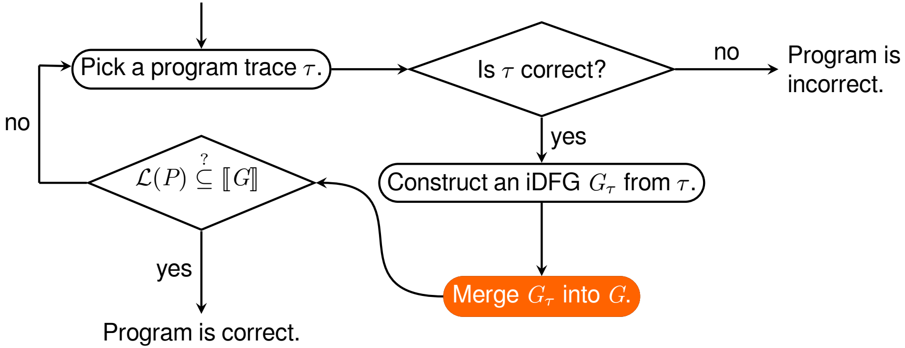


Goal

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



Automation



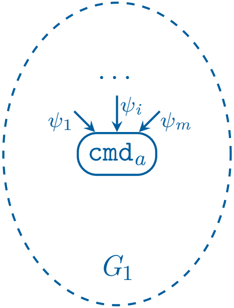
Pick a program trace τ.
Is τ correct?
no
Program is incorrect.
yes
Construct an iDFG G_τ from τ.
Merge G_τ into G.
$\mathcal{L}(P) \overset{?}{\subseteq} [\![G]\!]$
no
yes
Program is correct.

Merging iDFGs

Goal

Given iDFGs G_1, G_2, construct an iDFG $G_1 \mathbin{\bigwedge\!\!\!\bigwedge} G_2$ such that

$$[\![G_1]\!] \cup [\![G_2]\!] \subseteq [\![G_1 \mathbin{\bigwedge\!\!\!\bigwedge} G_2]\!]$$

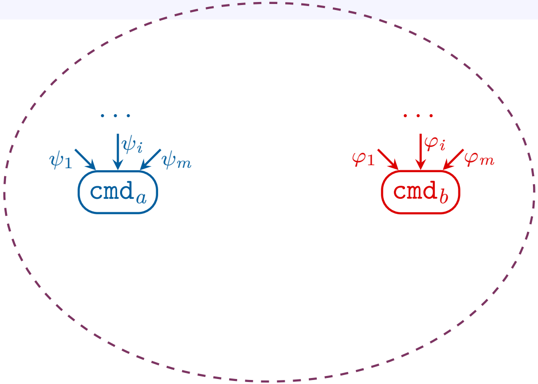


Merging iDFGs

Goal

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

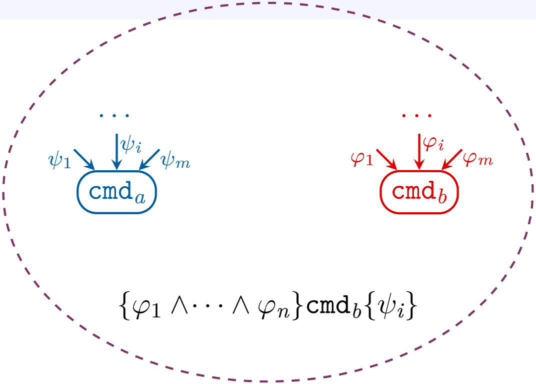
$$[\![G_1]\!] \cup [\![G_2]\!] \subseteq [\![G_1 \curlywedge G_2]\!]$$



Goal

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

$$[\![G_1]\!] \cup [\![G_2]\!] \subseteq [\![G_1 \wedge\!\!\wedge G_2]\!]$$

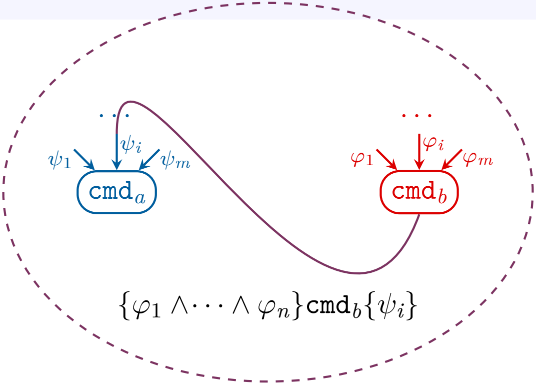


Merging iDFGs

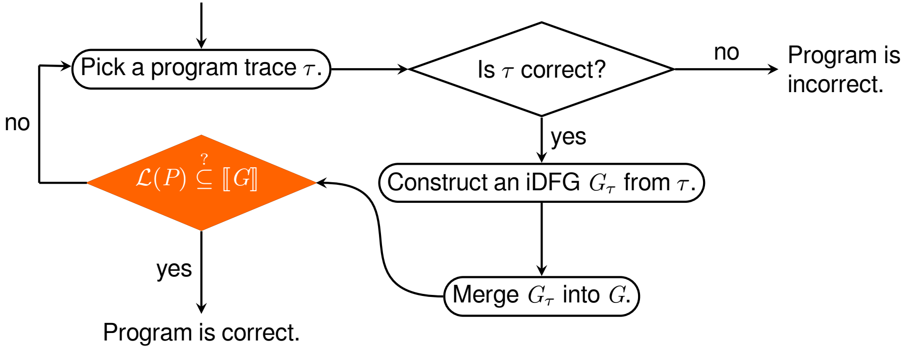
Goal

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

$$[\![G_1]\!] \cup [\![G_2]\!] \subseteq [\![G_1 \curlywedge G_2]\!]$$



Automation



Pick a program trace τ.
Is τ correct?
no
Program is incorrect.
yes
Construct an iDFG G_τ from τ.
Merge G_τ into G.
$\mathcal{L}(P) \overset{?}{\subseteq} [\![G]\!]$
no
yes
Program is correct.

Proof checking

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

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

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

- 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.