Two Birds with One Stone: Multi-Derivation for Fast Context-Free Language Reachability Analysis

Chenghang Shi, Haofeng Li, Yulei Sui,

Jie Lu, Lian Li, Jingling Xue

Institute of Computing Technology

University of Chinese Academy Sciences

University of New South Wales

CFL (Context-Free Language) Reachability

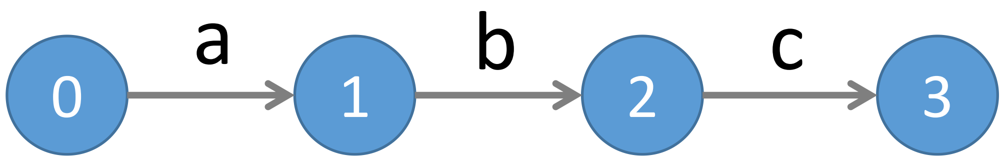
- Fundamental framework for program analysis
  - Taint Analysis
  - Pointer Analysis
  - Bug Detection
  - Program Slicing
  - ...

What is CFL-Reachability?

CFL-reachability extends the conventional graph reachability problem to an edge-labeled graph with a context-free language.

What is CFL-Reachability?

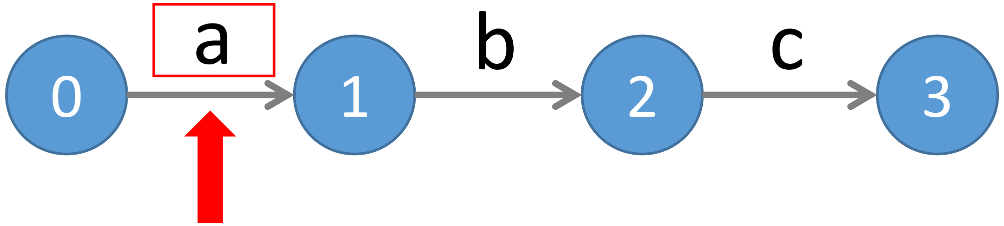
CFL-reachability extends the conventional graph reachability problem to an edge-labeled graph with a context-free language.



Edge-labeled Graph G

What is CFL-Reachability?

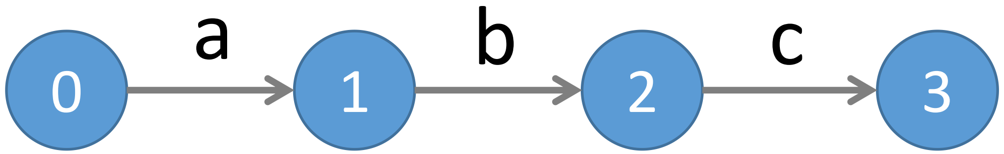
CFL-reachability extends the conventional graph reachability problem to an edge-labeled graph with a context-free language.



Edge-labeled Graph G

What is CFL-Reachability?

CFL-reachability extends the conventional graph reachability problem to an edge-labeled graph with a context-free language.



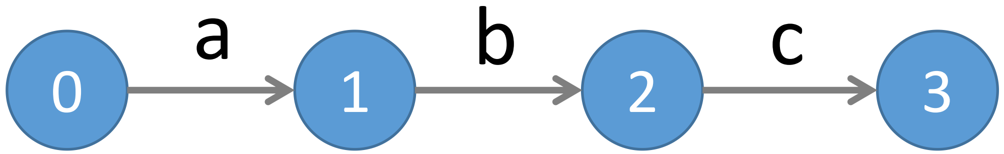
Edge-labeled Graph G

$$X ::= a\ b$$
$$Y ::= X\ c$$

Context-free Grammar of L

What is CFL-Reachability?

CFL-reachability extends the conventional graph reachability problem to an edge-labeled graph with a context-free language.



Edge-labeled Graph G

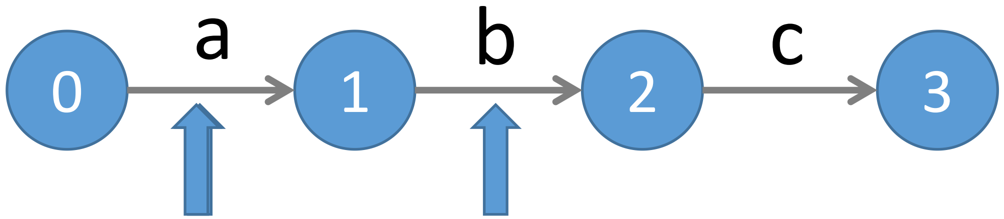
$$X ::= a\ b$$
$$Y ::= X\ c$$

Context-free Grammar of L

CFL solving ≈ Edge Derivation + Edge Insertion

An Example

An X-reachability relation holds between Node 0 and Node 2, i.e., Node 2 is X-reachable from Node 0



Edge-labeled Graph G

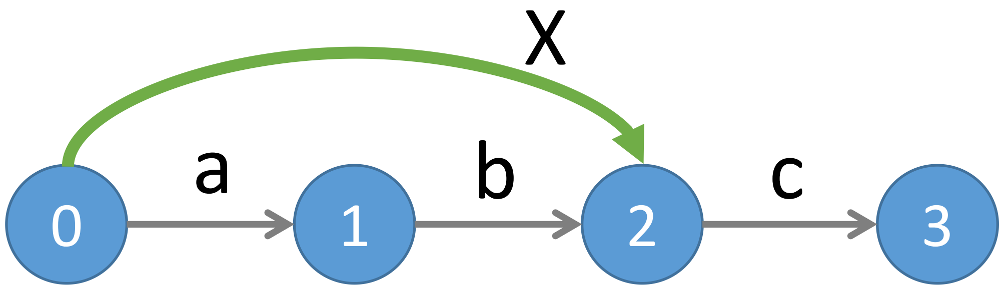
$X ::= a\ b$

$Y ::= X\ c$

Context-free Grammar of L

An Example

Insert an X-edge from Node 0 to Node 2!

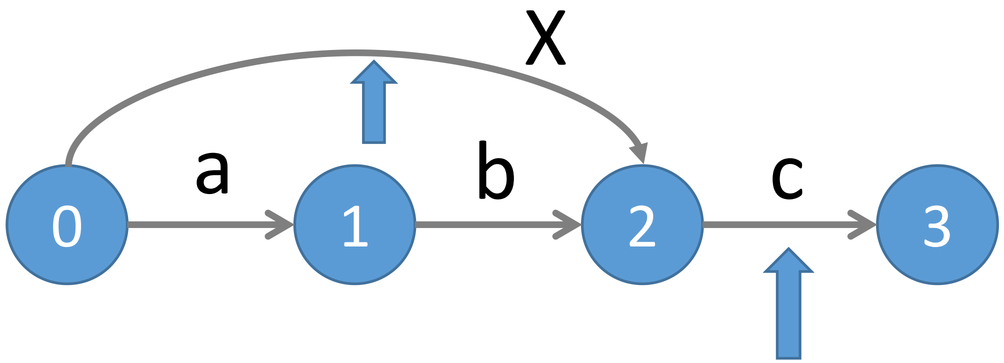


$X ::= a\ b$

$Y ::= X\ c$

An Example

A Y-reachability relation holds between Node 0 and Node 3

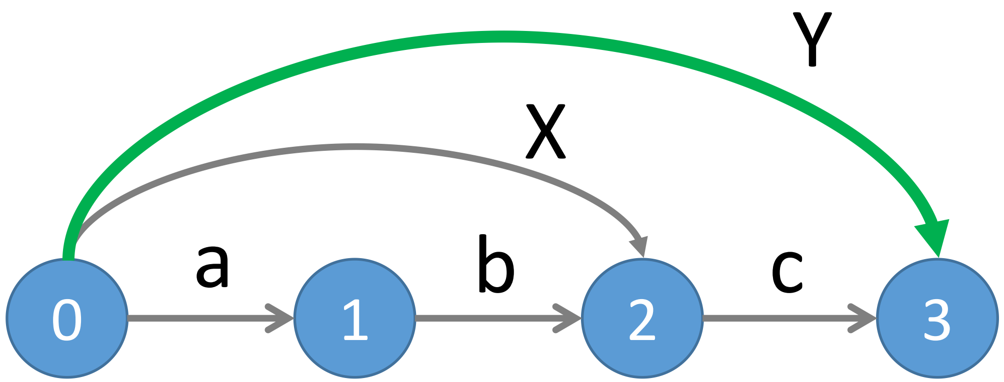


X ::= a b

Y ::= X c

An Example

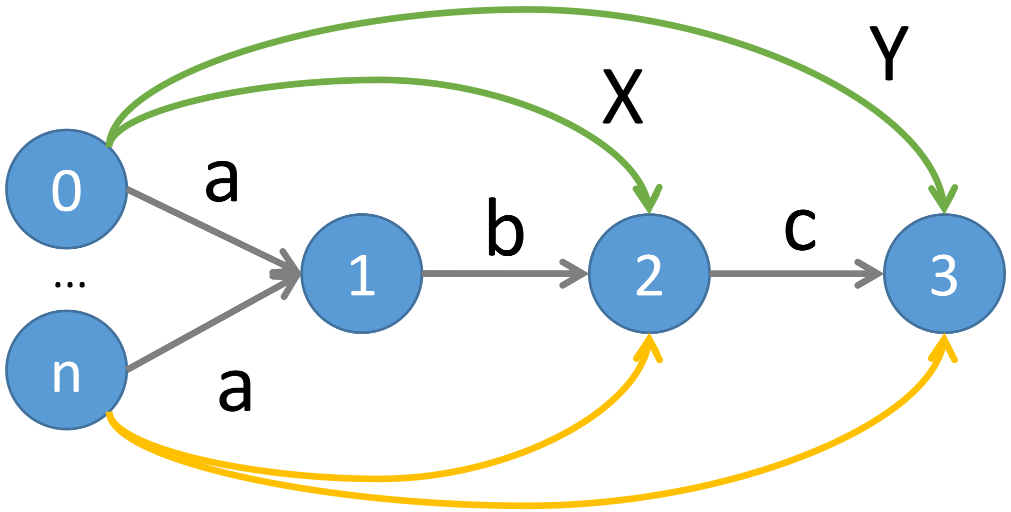
Insert a Y-edge from Node 0 to Node 3!



X ::= a b

Y ::= X c

Limitation of Existing Approaches

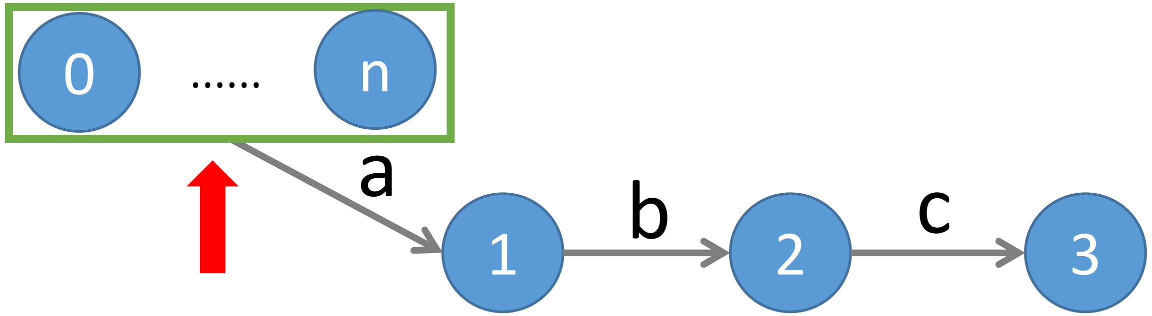


X ::= a b

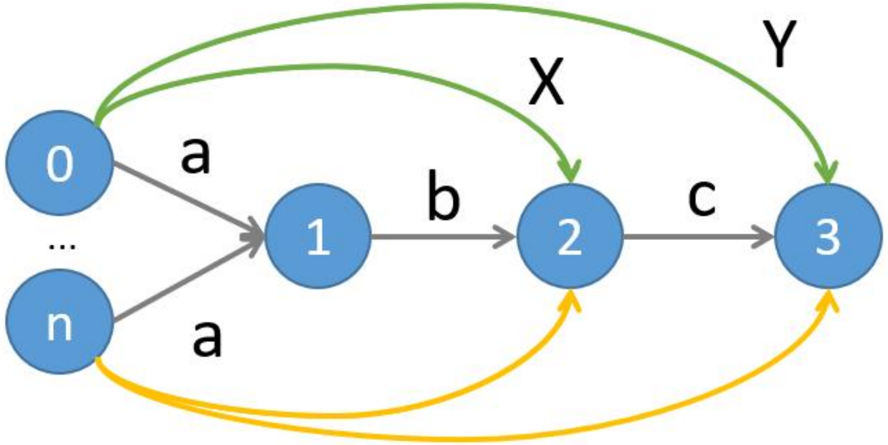
Y ::= X c

*single-reachability derivation: existing CFL algorithms process multiple a-reachability relations **separately**, which causes **redundancy***

Multi-Derivation



Multi-derivation



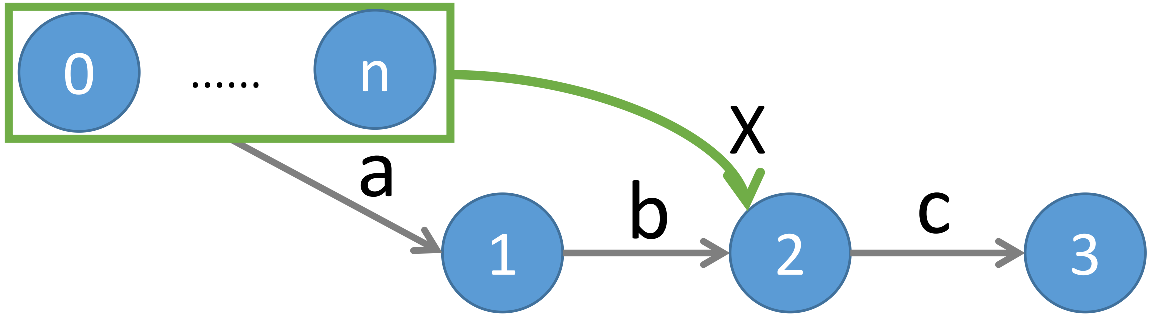
Single-reachability derivation

X ::= a b

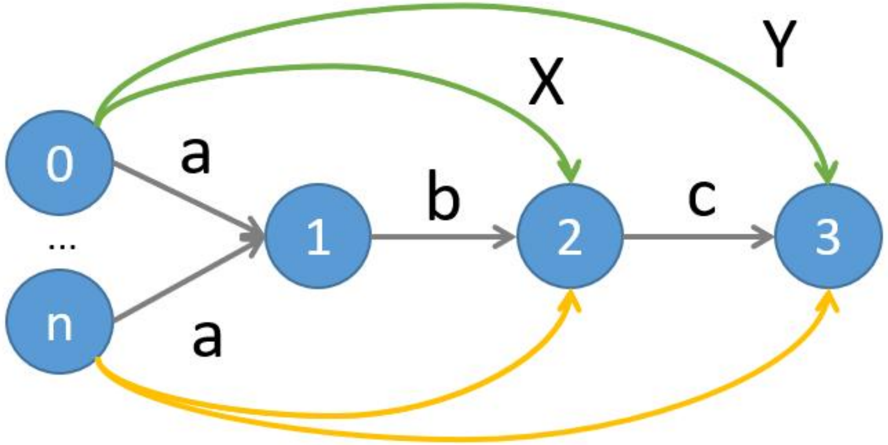
Y ::= X c

Packing multiple a-reachability relations together

Multi-Derivation



Multi-derivation



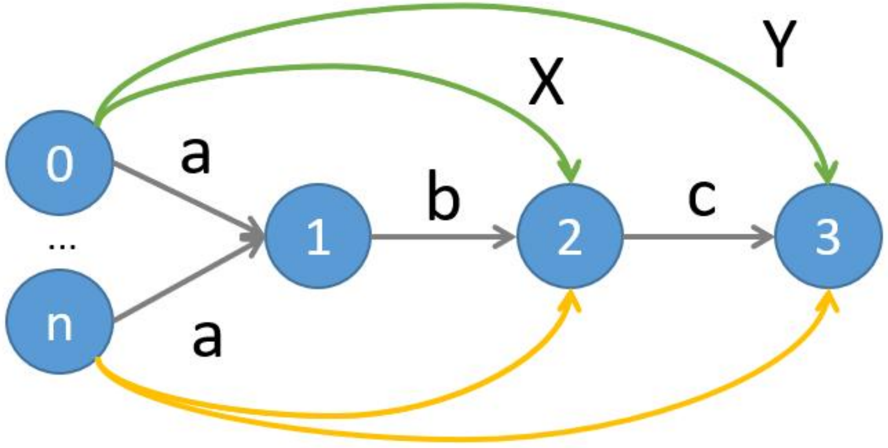
Single-reachability derivation

X ::= a b

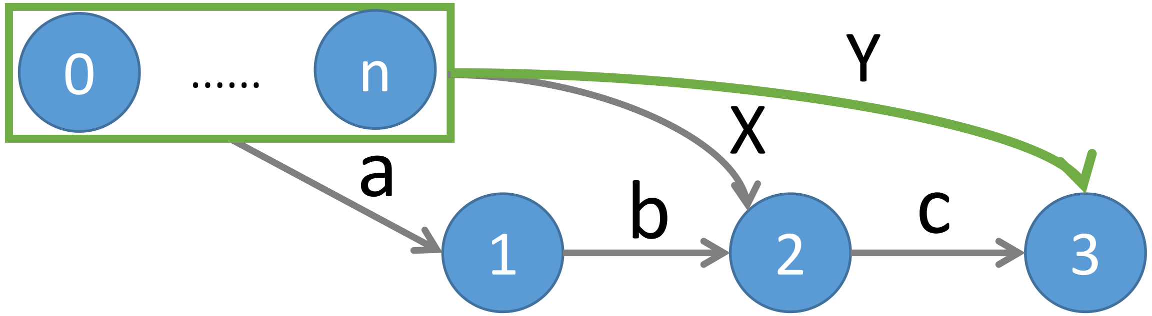
Y ::= X c

Propagating a-reachability relations in ***batch*** *via b-edge -> multiple new X-reachability relations produced*

Multi-Derivation



Single-reachability derivation



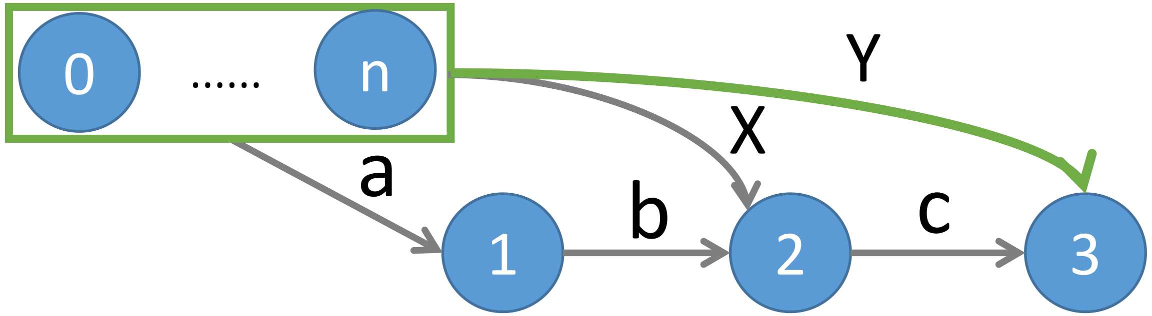
Multi-derivation

X ::= a b

Y ::= X c

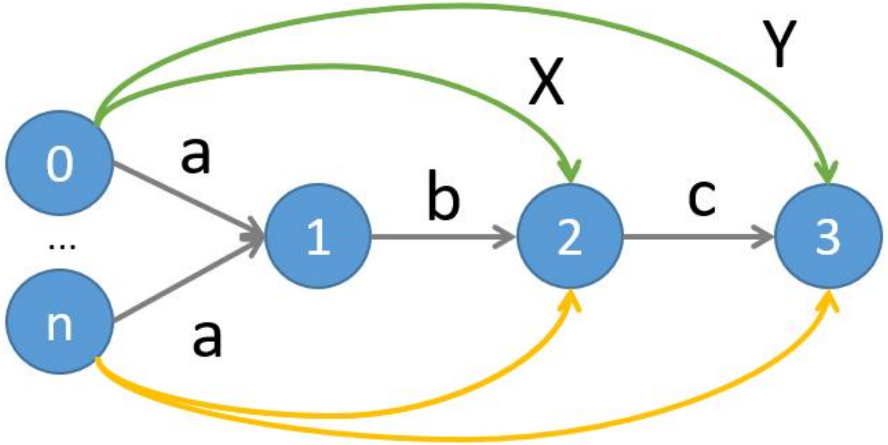
Propagating X-reachability relations in batch via c-edge -> multiple new Y-reachability relations produced

Multi-Derivation



Multi-derivation

Difference propagation!



Single-reachability derivation

$$X ::= a\ b$$
$$Y ::= X\ c$$

Optimize Multi-Derivation for Transitivity

- Transitive relations are ubiquitous in CFL-based program analysis, e.g., data flow, control flow are transitive.

Optimize Multi-Derivation for Transitivity

- Transitive relations are ubiquitous in CFL-based program analysis, e.g., data flow, control flow are transitive.
- Relation A is transitive if we have:

$$A ::= A\ A \text{ or } A ::= A^* \text{ or } A ::= A^+ \ldots$$

Optimize Multi-Derivation for Transitivity

- Transitive relations are ubiquitous in CFL-based program analysis, e.g., data flow, control flow are transitive.
- Relation A is transitive if we have:

$$A ::= A\,A \text{ or } A ::= A^{*} \text{ or } A ::= A^{+} \ldots$$

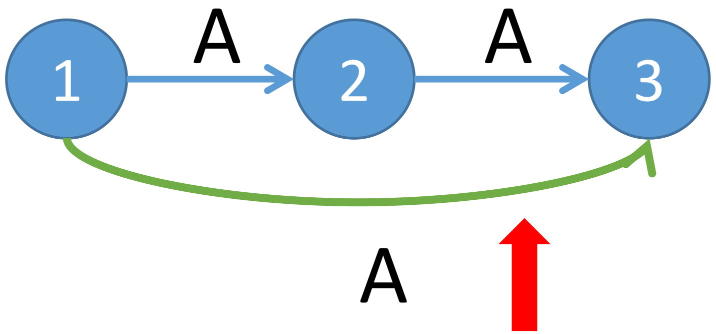
- Property: *two consecutive A-edges form a new A-Edge*.

Optimize Multi-Derivation for Transitivity

- Transitive relations are ubiquitous in CFL-based program analysis, e.g., data flow, control flow are transitive.
- Relation A is transitive if we have:

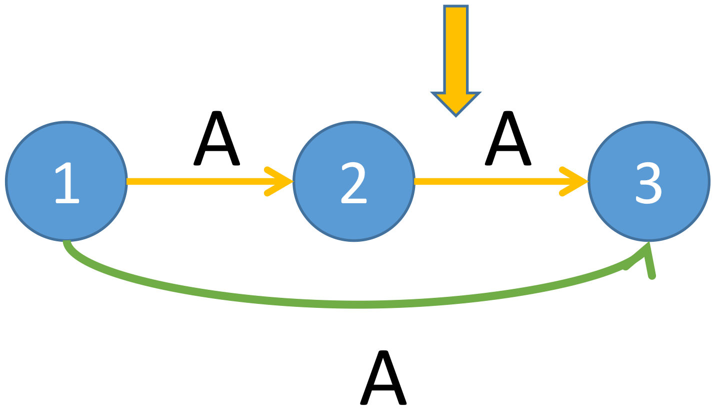
$$A ::= A\ A \text{ or } A ::= A^* \text{ or } A ::= A^+ \ldots$$

- Property: *two consecutive A-edges form a new A-Edge*.



Redundant Propagation due to Transitivity

The X-reachability relations of Node 1 are propagated **twice** from Node 1 to Node 3

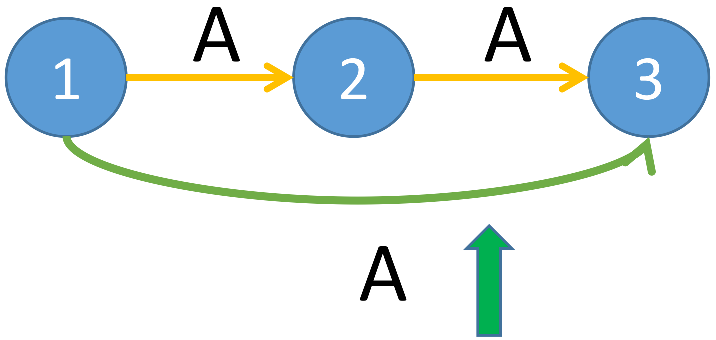


A ::= A A

X ::= X A

Redundant Propagation due to Transitivity

The X-reachability relations of Node 1 are propagated **twice** from Node 1 to Node 3

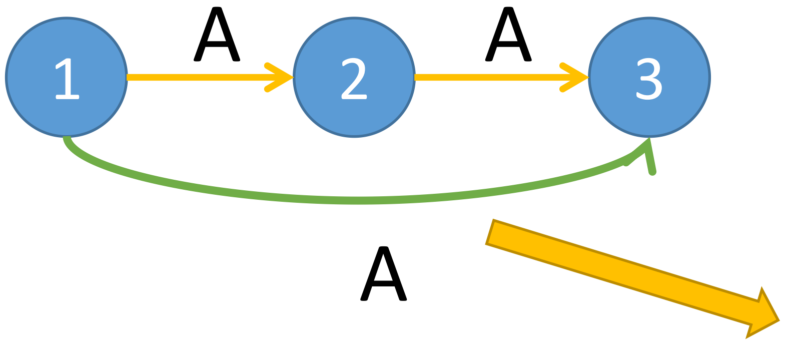


A ::= A A

X ::= X A

Redundant Propagation due to Transitivity

The X-reachability relations of Node 1 are propagated **twice** from Node 1 to Node 3



$A ::= A\ A$

$X ::= X\ A$

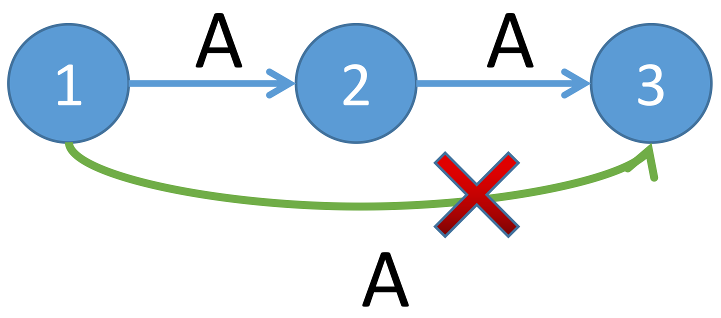
Shortcut path causes redundant propagation!

Propagation Graph

- *A transitivity-aware propagation graph,* **PG(A)** for transitive relation A
  - With redundant edges excluded (partially)

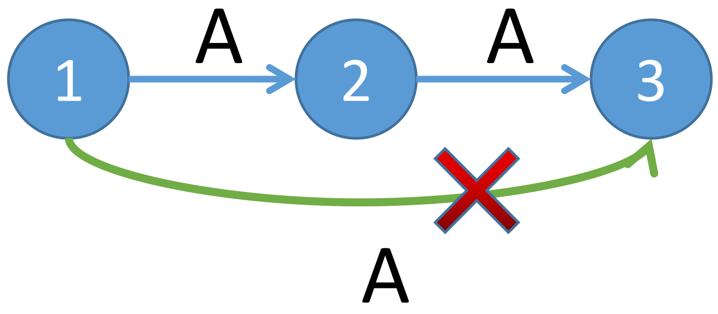
Propagation Graph

- *A <u>transitivity-aware</u> propagation graph,* **PG(A)** for transitive relation A
  - With redundant edges excluded (partially)



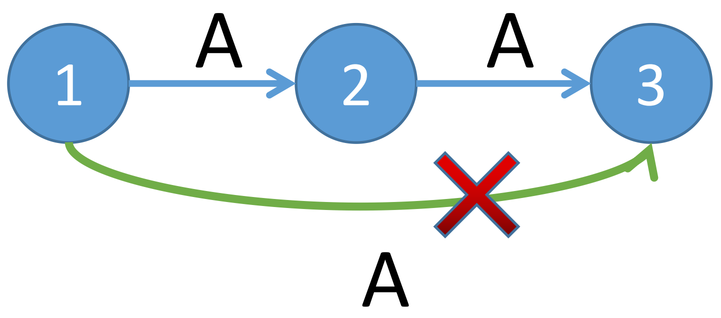
Propagation Graph

- *A <u>transitivity-aware</u> propagation graph,* **PG(A)** for transitive relation A
  - With redundant edges excluded (partially)
- Construction
  - Subgraph induced from the original edge-labeled graph
  - Constructed on the fly



Propagation Graph

- The construction of propagation graph is non-trivial
  - Duplicate A-edges can be introduced by other productions rather than A ::= A A
  - Can be seen as **partial transitive reduction**
  - Please refer to our paper for more details



Solving Transitivity via Multi-Derivation

- *A ::= A A:* propagating A-reachability relations in batch in PG(A)
- *X ::= X A:* propagating X-reachability relations in batch in PG(A)

Solving Transitivity via Multi-Derivation

- *A ::= A A:* propagating A-reachability relations in batch in PG(A)
- *X ::= X A:* propagating X-reachability relations in batch in PG(A)
- PG(A) is constructed on the fly with redundant edges excluded

Solving Transitivity via Multi-Derivation

- *A ::= A A:* propagating A-reachability relations in batch in PG(A)
- *X ::= X A:* propagating X-reachability relations in batch in PG(A)
- PG(A) is constructed on the fly with redundant edges excluded
- *X ::= A X* can be handled similarly as *X ::= X A*

Evaluation

- Implementation: SVF, https://github.com/SVF-tools/SVF
  - CFL-reachability solver: ***Pearl***
- 2 popular clients: Value-Flow Analysis and Alias Analysis for C++
- 10 benchmarks from SPEC CPU2017 C/C++
- Compare with
  - the standard algorithm
  - POCR[OOPSLA'22] for fast transitivity solving

Performance

Field-Sensitive Alias Analysis

Speed up over POCR: **2.4x (avg.), 4.2x(max.)**

id	STD		POCR			PEARL		
	Time	Mem	Time	SPU	Mem	Time	SPU	Mem
cactus	-	-	191.27	-	11.62	96.59	2.0x	9.28
imagick	-	-	554.13	-	42.55	334.76	1.7x	41.41
leela	312.28	0.31	3.40	91.8x	0.39	2.24	1.5x	0.36
nab	7.12	0.10	0.76	9.4x	0.10	0.18	4.2x	0.09
omnetpp	-	-	410.79	-	17.96	195.77	2.1x	17.08
parest	-	-	92.77	-	4.79	42.10	2.2x	4.69
perlbench	-	-	1733.42	-	110.84	978.29	1.8x	80.30
povray	14699.10	3.24	160.97	91.3x	6.60	58.64	2.7x	5.55
x264	1056.20	1.31	11.13	94.9x	1.05	3.39	3.3x	1.00
xz	6.67	0.05	0.42	15.9x	0.07	0.19	2.2x	0.07

Performance

Context-Sensitive Value Flow Analysis

Speed up over POCR: **10.1x (avg.), 29.2x(max.)**

id	STD		POCR			PEARL		
	Time	Mem	Time	SPU	Mem	Time	SPU	Mem
cactus	3408.36	3.46	604.10	5.6x	40.26	28.26	21.4x	4.74
imagick	583.71	0.43	59.13	9.9x	5.87	5.18	11.4x	0.74
leela	1.58	0.02	0.47	3.4x	0.19	0.16	2.9x	0.02
nab	55.51	0.50	16.59	3.3x	4.34	3.27	5.1x	0.32
omnetpp	229.26	1.08	15.49	14.8x	3.99	3.62	4.3x	0.53
parest	2.40	0.07	0.67	3.6x	0.19	0.38	1.8x	0.07
perlbench	16366.80	6.35	1520.19	10.8x	63.57	52.06	29.2x	10.42
povray	5834.13	5.05	655.14	8.9x	55.84	43.91	14.9x	4.72
x264	194.16	0.70	34.77	5.6x	6.46	4.71	7.4x	0.67
xz	0.54	0.01	0.16	3.4x	0.06	0.06	2.7x	0.01

Performance

Context-Sensitive Value Flow Analysis

For *perlbench*, reduce **84% memory usage** over POCR

id	STD		POCR			PEARL		
	Time	Mem	Time	SPU	Mem	Time	SPU	Mem
cactus	3408.36	3.46	604.10	5.6x	40.26	28.26	21.4x	4.74
imagick	583.71	0.43	59.13	9.9x	5.87	5.18	11.4x	0.74
leela	1.58	0.02	0.47	3.4x	0.19	0.16	2.9x	0.02
nab	55.51	0.50	16.59	3.3x	4.34	3.27	5.1x	0.32
omnetpp	229.26	1.08	15.49	14.8x	3.99	3.62	4.3x	0.53
parest	2.40	0.07	0.67	3.6x	0.19	0.38	1.8x	0.07
perlbench	16366.80	6.35	1520.19	10.8x	63.57	52.06	29.2x	10.42
povray	5834.13	5.05	655.14	8.9x	55.84	43.91	14.9x	4.72
x264	194.16	0.70	34.77	5.6x	6.46	4.71	7.4x	0.67
xz	0.54	0.01	0.16	3.4x	0.06	0.06	2.7x	0.01

Ablation Study

- Pwb (**P**earl **w**ithout **b**atch propagation)
- Value-Flow Analysis: Pwb is 7.2x faster than POCR, and 1.3x slower than Pearl
- Alias Analysis: Pwb is comparable with POCR, and 2.2x slower than Pearl

Ablation Study

- Pwb (**P**earl **w**ithout **b**atch propagation)
- Value-Flow Analysis: Pwb is 7.2x faster than POCR, and 1.3x slower than Pearl
- Alias Analysis: Pwb is comparable with POCR, and 2.2x slower than Pearl

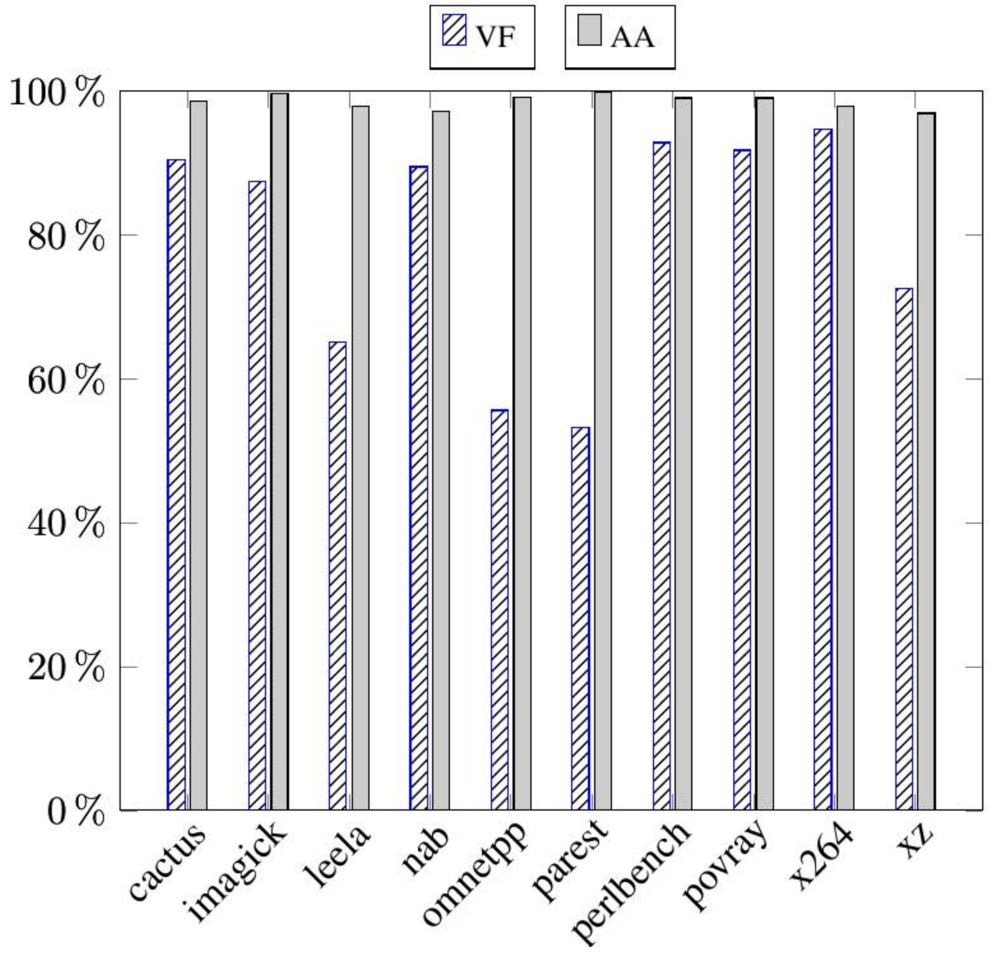
- Reason of different speedups
  - Propagation graph is simple yet effective, especially when transitive relations dominates, e.g., in value-flow analysis.
  - Value-flow analysis has no productions in the form of *X ::= X A*, so the effectiveness of multi-derivation is limited.

Effectiveness of Multi-Derivation

How many propagations of transitive relations can be reduced?

Reduction rates (Pearl v.s. Pwb)

- Value-flow analysis: 79.3%
- Alias analysis: 98.5%



Reduction rates in propagations of transitive relations

Conclusion

- Contributions
  - A multi-derivation approach to CFL-reachability
  - A transitivity-aware propagation graph representation
  - A highly efficient CFL-reachability solver, Pearl

Conclusion

- Contributions
  - A multi-derivation approach to CFL-reachability
  - A transitivity-aware propagation graph representation
  - A highly efficient CFL-reachability solver, Pearl
- Artifact available
  - Docker image and instructions for reproduction
  - https://figshare.com/articles/dataset/ASE_2023_artifact/23702271

Thank you for your listening!

Email: chenghangshi@gmail.com

Backup Slides

Set Constraint

- Set constraint and CFL reachability are interconvertible
- Two steps
  - reduce the CFL problem to a set contraint instance
  - solve it using off-the-shelf constraint solver
- Disadvantages
  - Unawareness of the graph features, e.g., transitivity
  - An extra reduction step