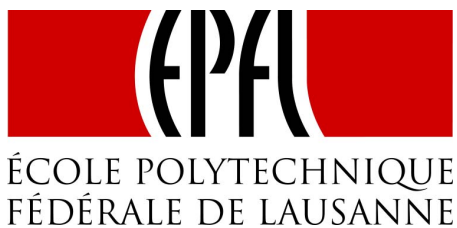
Optimal Dyck Reachability for Data-dependence and Alias Analysis

Krishnendu Chatterjee Bhavya Choudhary
Andreas Pavlogiannis



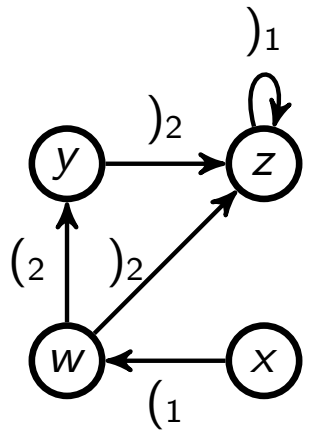
Static Analysis via Dyck Reachability

Dyck Reachability

$$\Sigma = \{(_1,)_1, \ldots (_k,)_k\} \cup \{\epsilon\}$$

$$G = (V, E, \lambda : E \to \Sigma)$$

$$\mathcal{S} \to \mathcal{S}\ \mathcal{S} \mid (_1\ \mathcal{S}\ )_1 \mid \ldots \mid (_k\ \mathcal{S}\ )_k \mid \epsilon$$

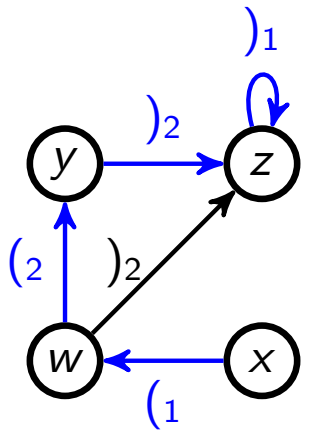


Dyck Reachability

$$\Sigma = \{(_1,)_1, \ldots (_k,)_k\} \cup \{\epsilon\}$$

$$G = (V, E, \lambda : E \to \Sigma)$$

$$\mathcal{S} \to \mathcal{S}\ \mathcal{S} \mid (_1\ \mathcal{S}\ )_1 \mid \ldots \mid (_k\ \mathcal{S}\ )_k \mid \epsilon$$



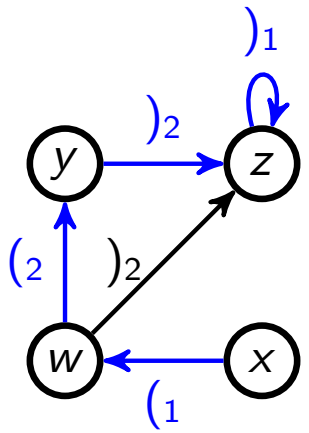
$$P : x \rightsquigarrow z \quad \text{with} \quad \lambda(P) = (_1(_2)_2)_1$$

Dyck Reachability

$$\Sigma = \{(_1,)_1, \ldots (_k,)_k\} \cup \{\epsilon\}$$

$$G = (V, E, \lambda : E \to \Sigma)$$

$$\mathcal{S} \to \mathcal{S}\ \mathcal{S} \mid (_1\ \mathcal{S}\ )_1 \mid \ldots \mid (_k\ \mathcal{S}\ )_k \mid \epsilon$$



$$P : x \rightsquigarrow z \quad \text{with} \quad \lambda(P) = (_1(_2)_2)_1$$

- Alias analysis
- Data-dependence analysis
- Data-flow analysis
- Shape analysis
- Impact analysis
- Bloat analysis
- Type-based flow analysis
- Program slicing

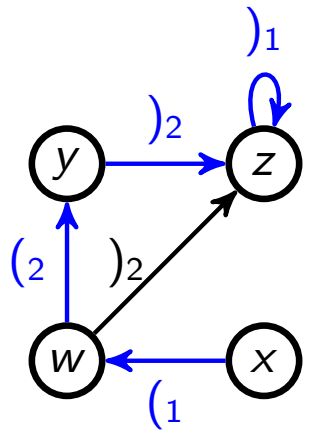
Dyck Reachability

$$\Sigma = \{(_1,)_1, \ldots (_k,)_k\} \cup \{\epsilon\}$$

$$G = (V, E, \lambda : E \to \Sigma)$$

$$\mathcal{S} \to \mathcal{S}\ \mathcal{S} \mid (_1\ \mathcal{S}\ )_1 \mid \ldots \mid (_k\ \mathcal{S}\ )_k \mid \epsilon$$

$$P : x \rightsquigarrow z \quad \text{with} \quad \lambda(P) = (_1(_2)_2)_1$$



- **Alias analysis**
- **Data-dependence analysis**
- Data-flow analysis
- Shape analysis
- Impact analysis
- Bloat analysis
- Type-based flow analysis
- Program slicing

Algorithms

Q: Is v Dyck-reachable from u?

- Dyck languages are a class of CFL
- Solution similar to CYK for CFL parsing
- $O(n^3)$
- $O(n^3/\log n)$ [Chaudhuri, POPL '08]
- Often prohibitive for lightweight static analysis

Algorithms

Q: Is v Dyck-reachable from u?

- Dyck languages are a class of CFL
- Solution similar to CYK for CFL parsing
- $O(n^3)$
- $O(n^3/\log n)$ [Chaudhuri, POPL '08]
- Often prohibitive for lightweight static analysis

Can do better if the graph is simple!

Data-dependence Analysis

Data-dependence analysis in the presence of libraries

- Which variable depends on which others
- Identify Def-Use chains in a program
- LHS depends on RHS

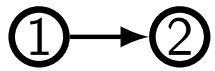
Data-dependence Analysis

Data-dependence analysis in the presence of libraries

- Which variable depends on which others
- Identify Def-Use chains in a program
- LHS depends on RHS

y depends on x if $x \rightsquigarrow y$

1 $x \leftarrow 2$
2 $y \leftarrow x + 1$



Interprocedural analysis

: $h()$

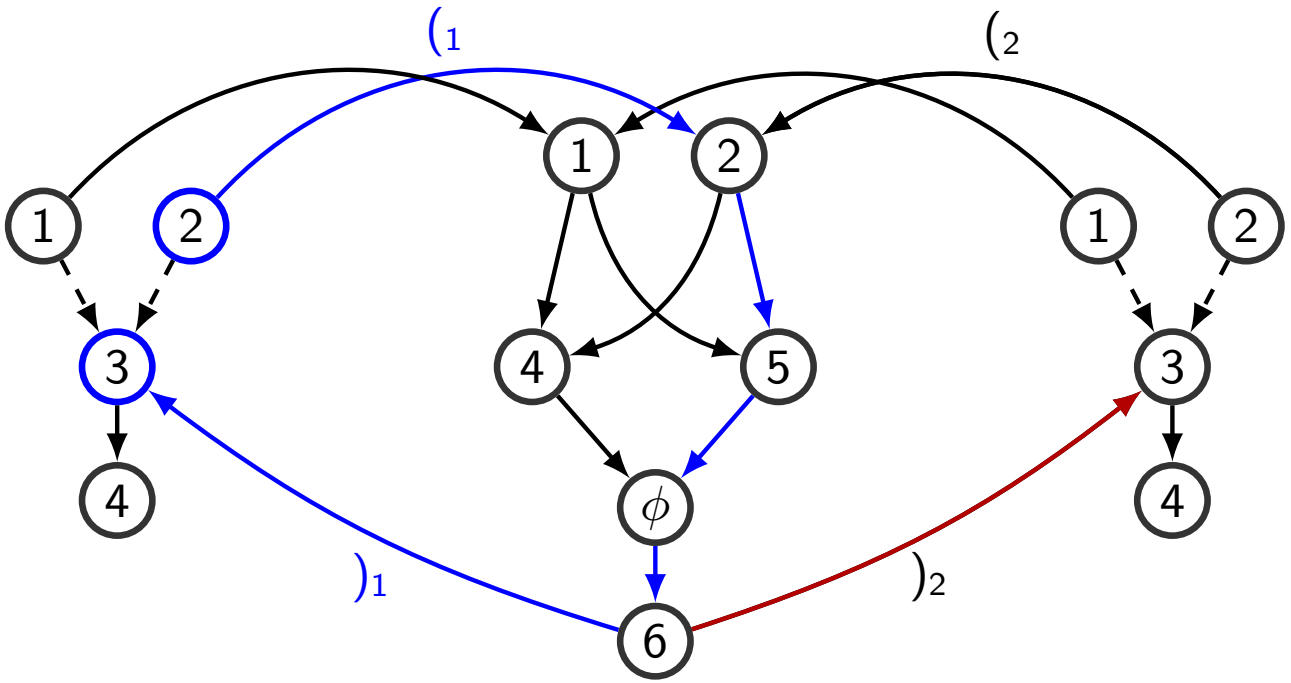
1 $x \leftarrow 2$
2 $y \leftarrow 2$
3 $p \leftarrow f(x, y)$
4 **return** p

: $f(x, y)$

1 **if** $y\%2 = 1$ **then**
2 $\quad z \leftarrow x + y$
3 **else**
4 $\quad z \leftarrow x \cdot y$
5 **end**
6 **return** z

: $g()$

1 $x \leftarrow 2$
2 $y \leftarrow 2$
3 $p \leftarrow f(x, y)$
4 **return** p



Analysis in the Presence of Libraries

- Large library
- Small client
- Goal:
  - Preprocess library once
  - Small cost of client analysis

Solution(?): Dyck reachability on the library once

Callbacks!

: $f_1(x, y)$

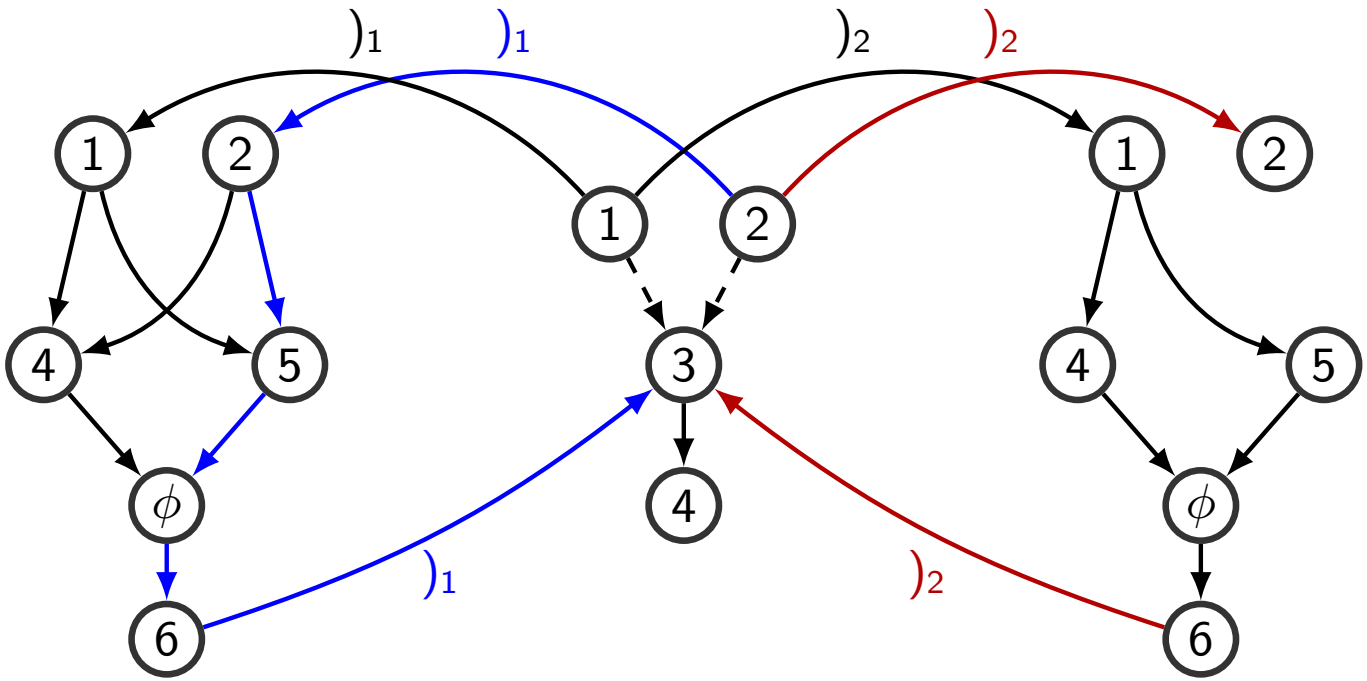
```
1 if y%2 = 1 then
2 |  z ← x + y
3 else
4 |  z ← x · y
5 end
6 return z
```

: $g()$

```
1 x ← 2
2 y ← 2
3 p ← f(x, y)
4 return p
```

: $f_2(x, y)$

```
1 if x%2 = 1 then
2 |  z ← 2 · x
3 else
4 |  z ← 2 · x + 1
5 end
6 return z
```



Results

- Library size n_1
- Library call sites k_1

- Client size n_2
- Client call sites k_2

$n_1 >> n_2$ $\quad$ $n_i >> k_i$

Approach	**Time**	
	Library	Client
CFL	$O\left(n_1^3\right)$	$O\left((n_1+n_2)^3\right)$
TAL*	$O\left(n_1^6\right)$	$O\left((s+n_2)^6\right)$

- * [TWZXZM, POPL '15]

Results

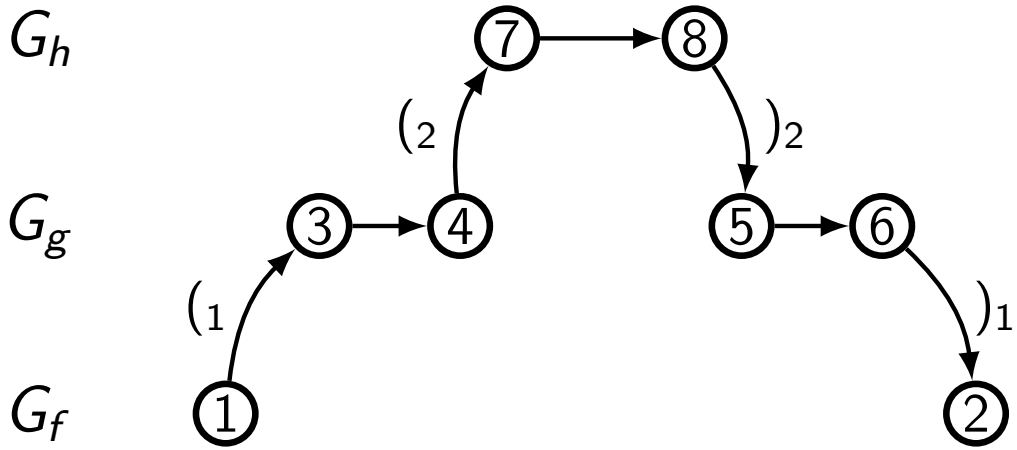
- Library size n_1
- Library call sites k_1
- Client size n_2
- Client call sites k_2

$$n_1 >> n_2 \qquad n_i >> k_i$$

Approach	**Time**	
	Library	Client
CFL	$O\left(n_1^3\right)$	$O\left((n_1+n_2)^3\right)$
TAL*	$O\left(n_1^6\right)$	$O\left((s+n_2)^6\right)$
This Paper	$O\left(n_1 + k_1 \cdot \log n_1\right)$	$O\left(n_2 + k_1 \cdot \log n_1 + k_2 \cdot \log n_2\right)$

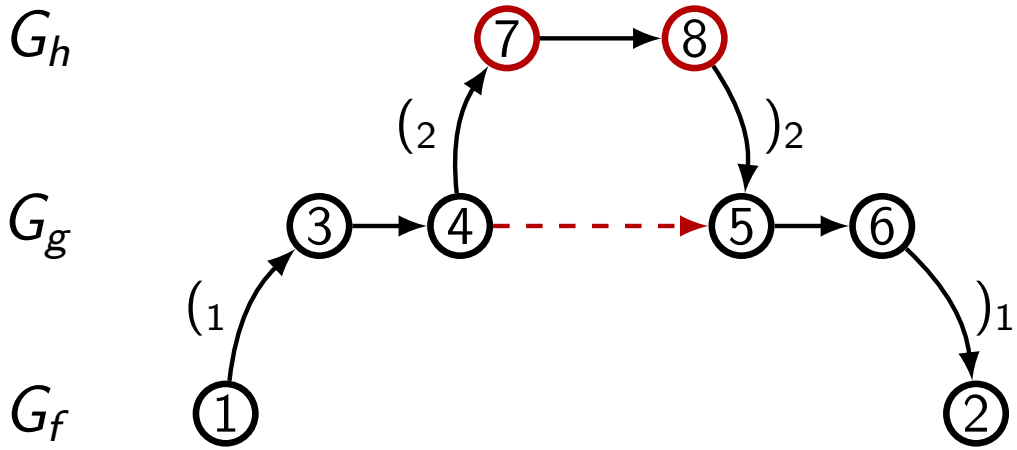
- * [TWZXZM, POPL '15]

Algorithmic Principle



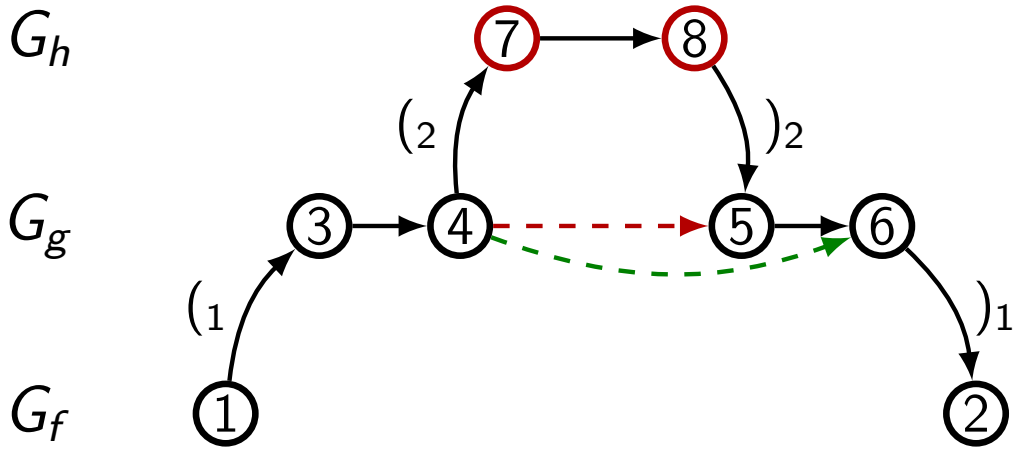
1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

Algorithmic Principle



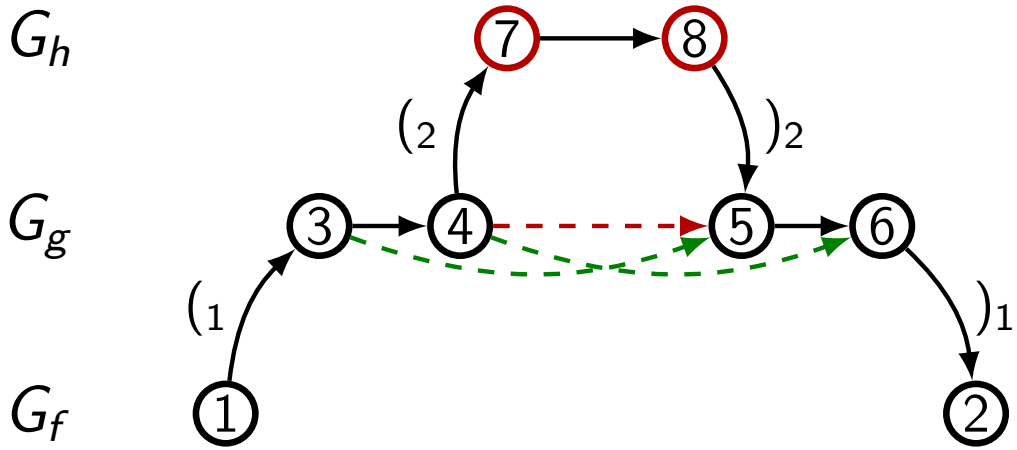
1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

Algorithmic Principle



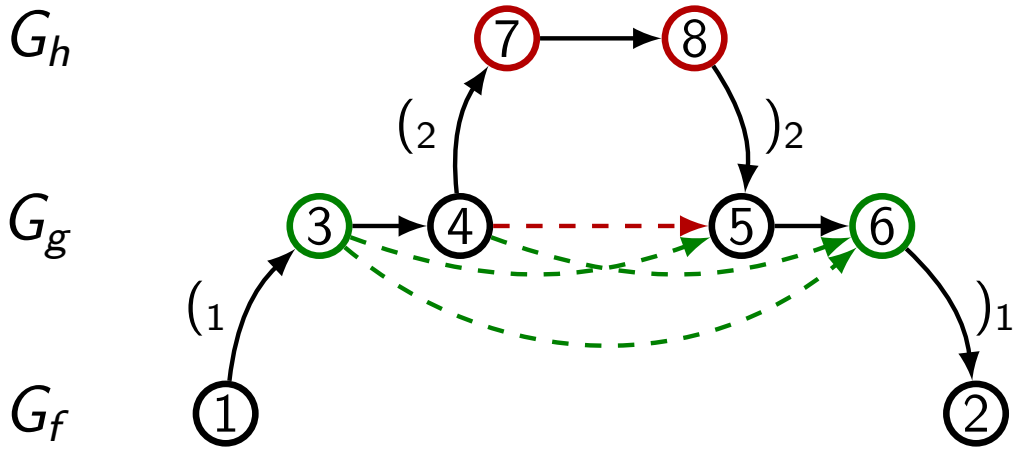
1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

Algorithmic Principle



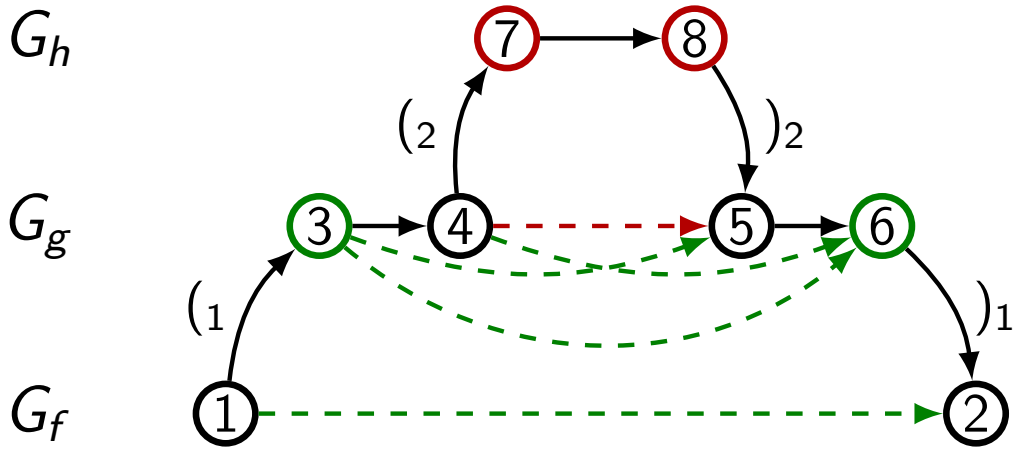
1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

Algorithmic Principle



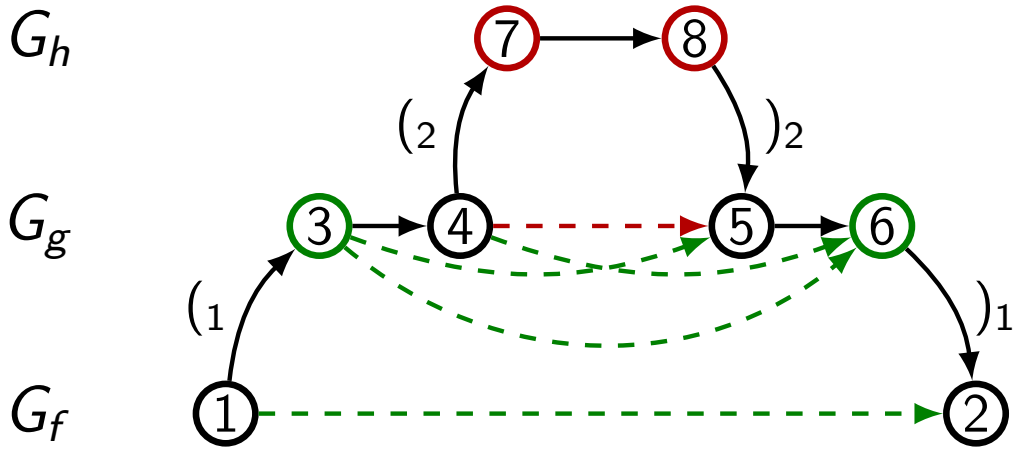
1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

Algorithmic Principle



1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

Algorithmic Principle



1. Reachability ignoring parenthesis edges
2. If Entry $\rightsquigarrow$ Exit
   - Insert summary edge

- Most time spent in (1)
- Each G_i is a "simple graph"
  - Low treewidth

Treewidth

Given a graph G

- An integer parameter $t_G \geqslant 1$
- Measures how "similar" G is to a tree
- G is a tree iff $t_G = 1$
- Trees are very simple to handle
- Low-treewidth graphs are somewhat simple to handle

Treewidth

Given a graph G

- An integer parameter $t_G \geqslant 1$
- Measures how "similar" G is to a tree
- G is a tree iff $t_G = 1$
- Trees are very simple to handle
- Low-treewidth graphs are somewhat simple to handle

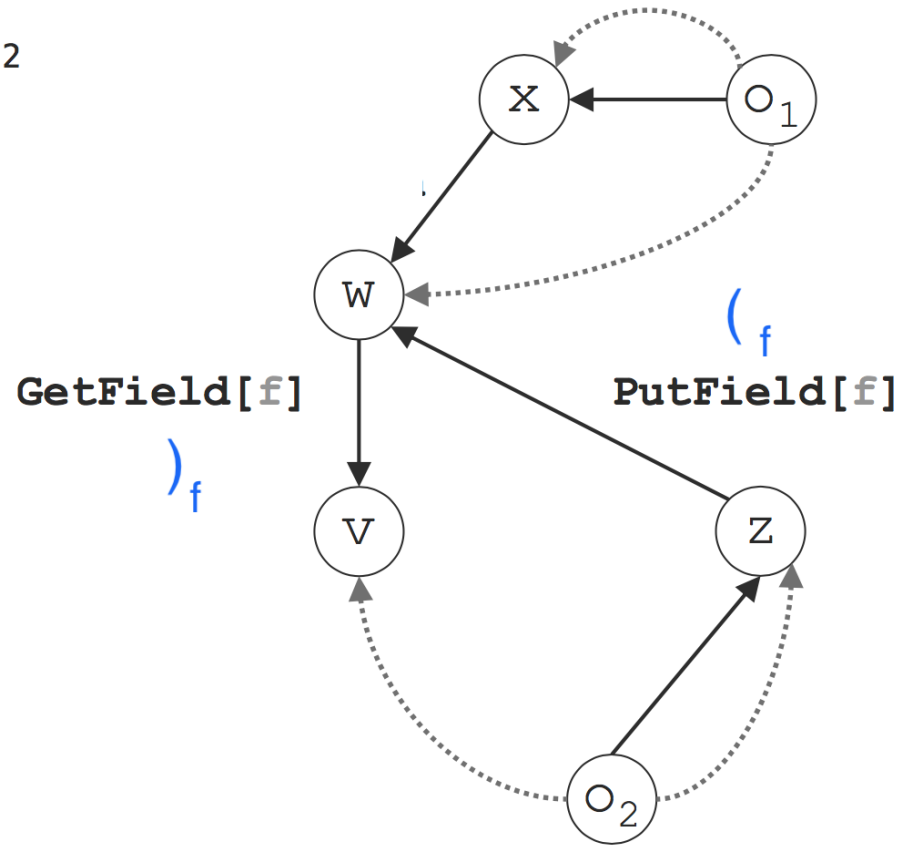
Theorem

For a low-treewidth graph we can construct a dynamic reachability oracle that supports

- *edge insertions in* $O(\log n)$ *time*
- *reachability queries in* $O(\log n)$ *time*

May Alias Analysis

```
x = new Obj(); // o1
z = new Obj(); // o2
w = x;
w.f = z;
v = w.f;
```

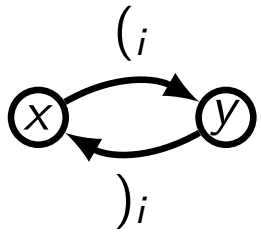


o_1 ⇢ x: object o_1 flows to pointer x

[YE, ESOP '09] ⇔ pointer x points to object o_1

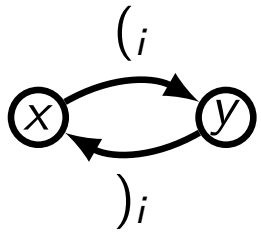
Bidirected graphs

$$\forall u, v \in V : \ \lambda(u, v) = (_i \quad \Longleftrightarrow \quad \lambda(v, u) =)_i$$



Bidirected graphs

$$\forall u, v \in V : \ \lambda(u, v) = (_i \quad \Longleftrightarrow \quad \lambda(v, u) =)_i$$



- Demand-driven field-sensitive alias analysis
- [YXR, ISSTA '11] [ZLYS, PLDI '13]

Results

Graphs of n nodes

Source	**Worst-case Time**	**Average-case Time**
[ZLYS, PLDI '13]	$O\left(n^2\right)$	$O\left(n \cdot \log n\right)$

Results

Graphs of n nodes, $\alpha(n)$ the inverse Ackermann

Source	**Worst-case Time**	**Average-case Time**
[ZLYS, PLDI '13]	$O\left(n^2\right)$	$O\left(m \cdot \log n\right)$
This paper	$O(n \cdot \alpha(n))$	$O(n)$

Results

Graphs of n nodes, $\alpha(n)$ the inverse Ackermann

Source	**Worst-case Time**	**Average-case Time**
[ZLYS, PLDI '13]	$O\left(n^2\right)$	$O\left(m \cdot \log n\right)$
This paper	$O(n \cdot \alpha(n))$	$O(n)$
	optimal!	optimal!

Reachability is Equivalence

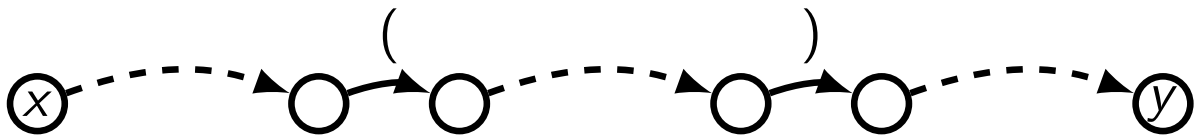
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.

Reachability is Equivalence

Theorem (ZLYS, PLDI '13)

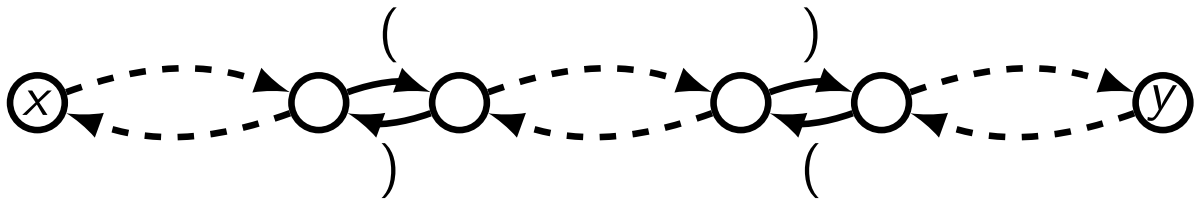
Dyck reachability on bidirected graphs is an equivalence relation.



Reachability is Equivalence

Theorem (ZLYS, PLDI '13)

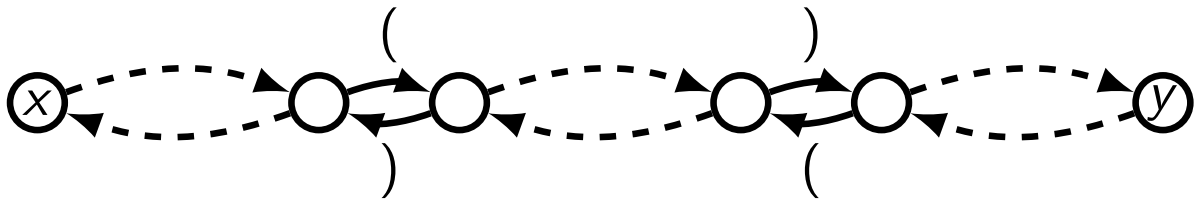
Dyck reachability on bidirected graphs is an equivalence relation.



Reachability is Equivalence

Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.

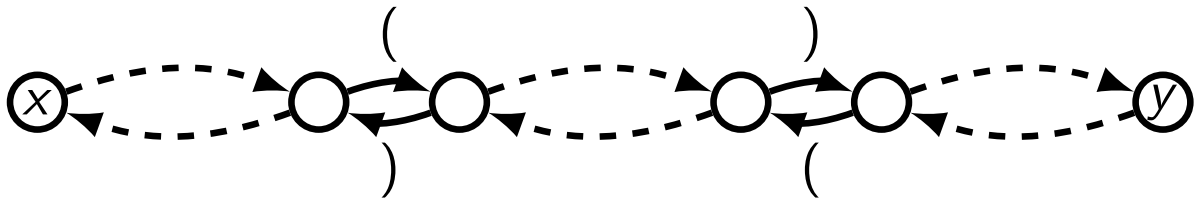


- Compute Strongly Connected Components

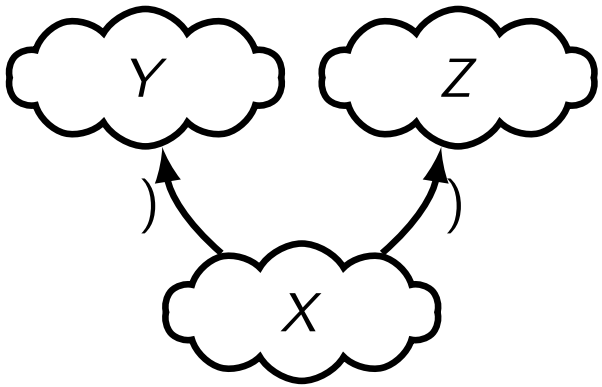
Reachability is Equivalence

Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



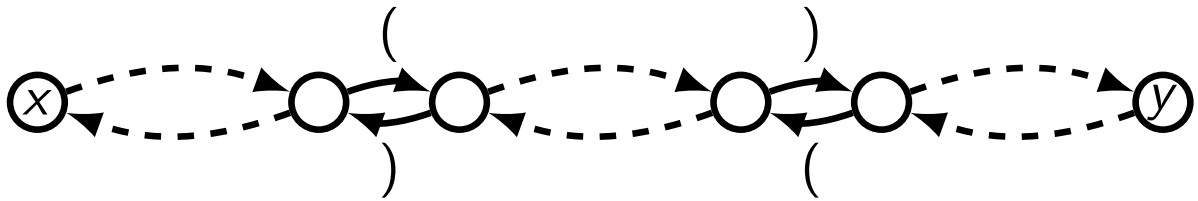
- Compute Strongly Connected Components



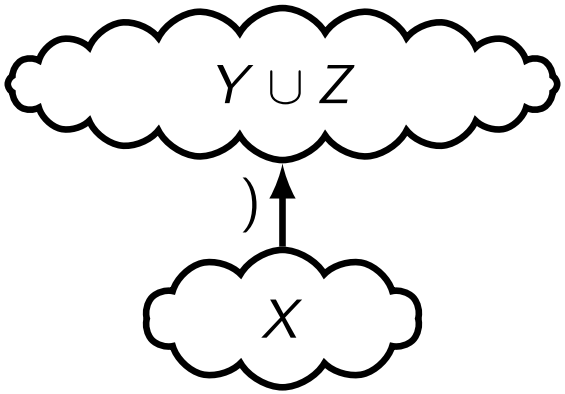
Reachability is Equivalence

Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



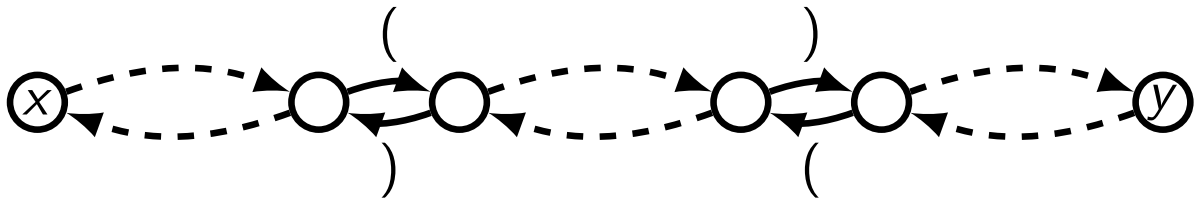
- Compute Strongly Connected Components



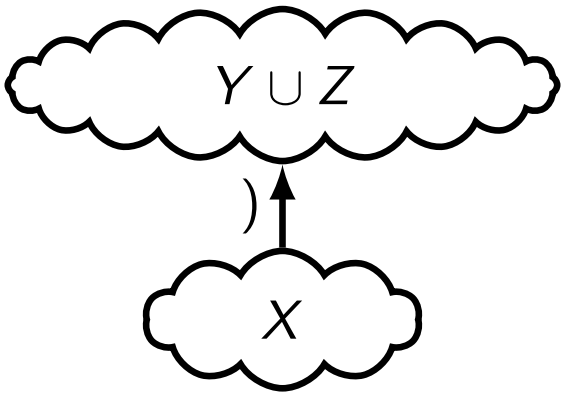
Reachability is Equivalence

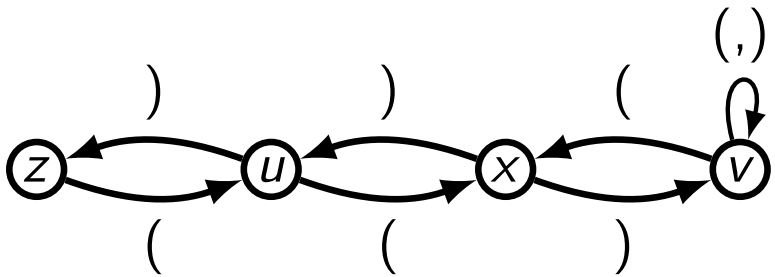
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components

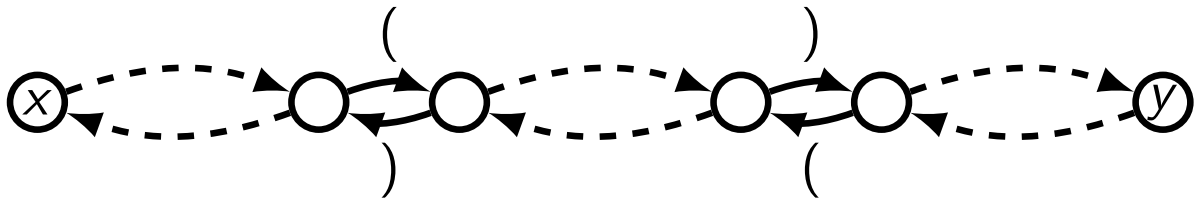




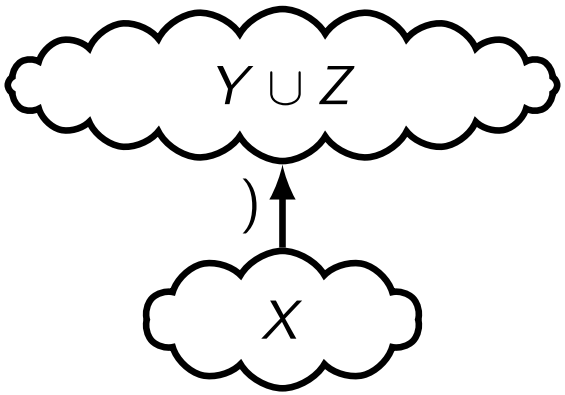
Reachability is Equivalence

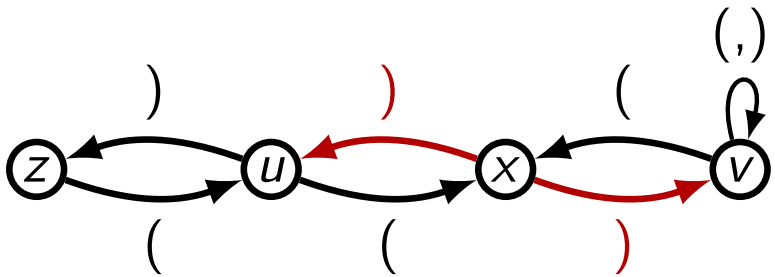
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components

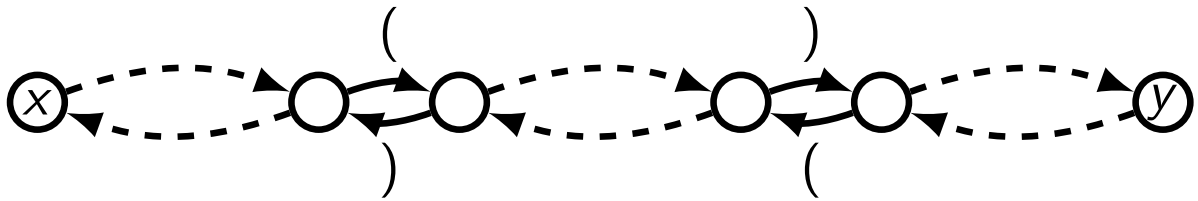




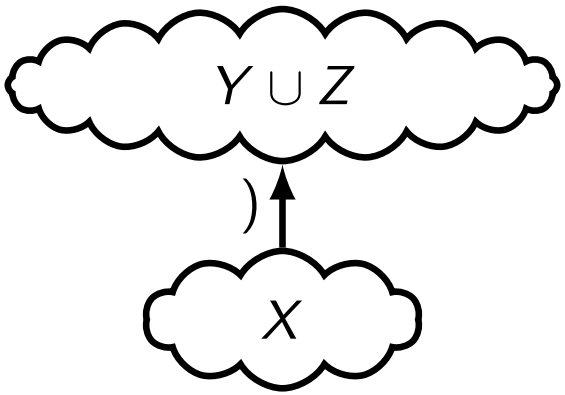
Reachability is Equivalence

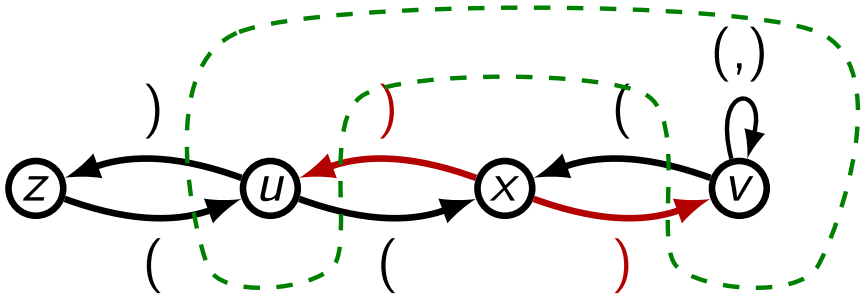
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components

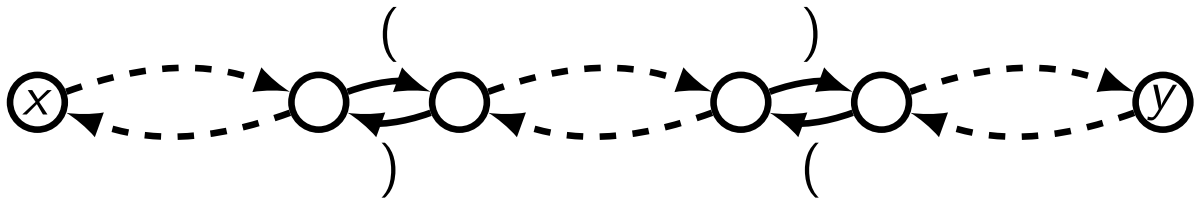




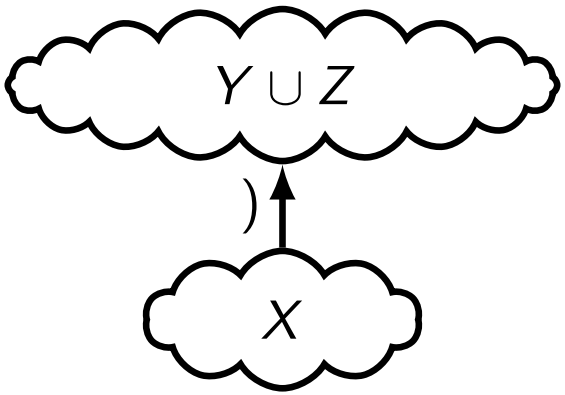
Reachability is Equivalence

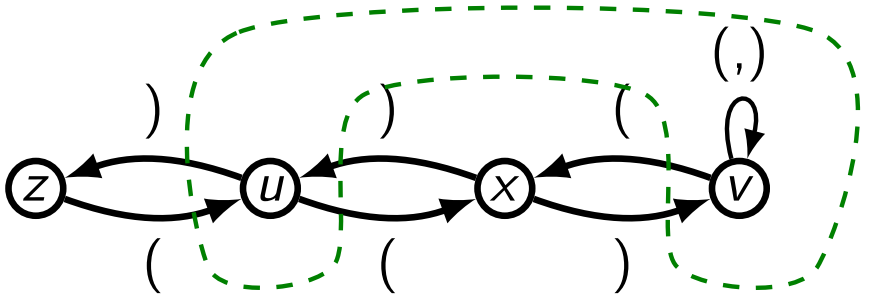
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components

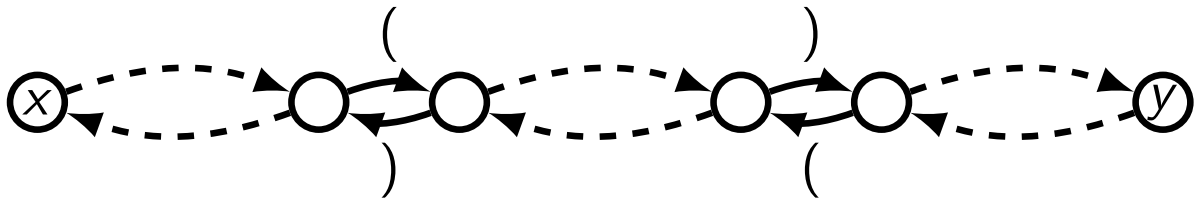




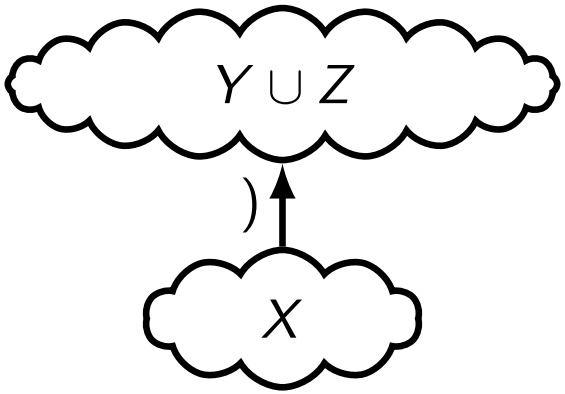
Reachability is Equivalence

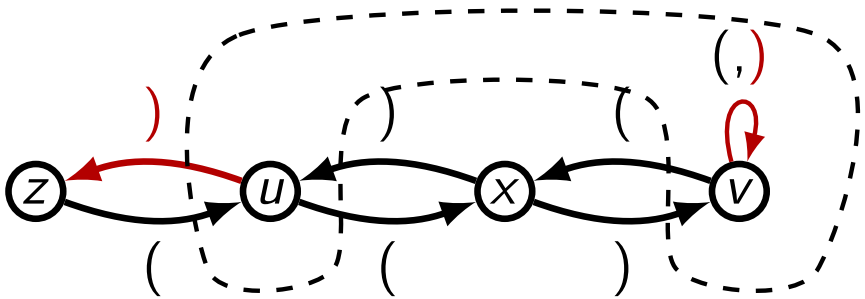
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components

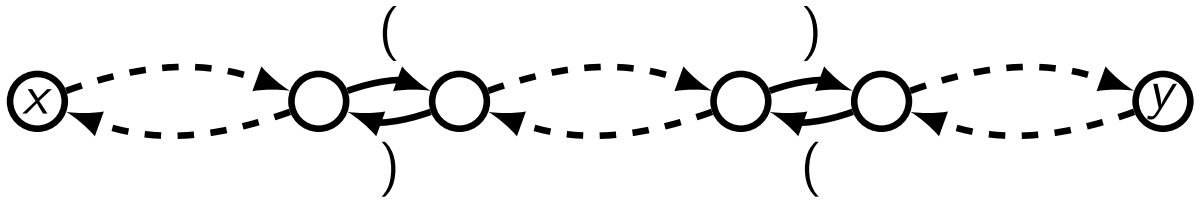




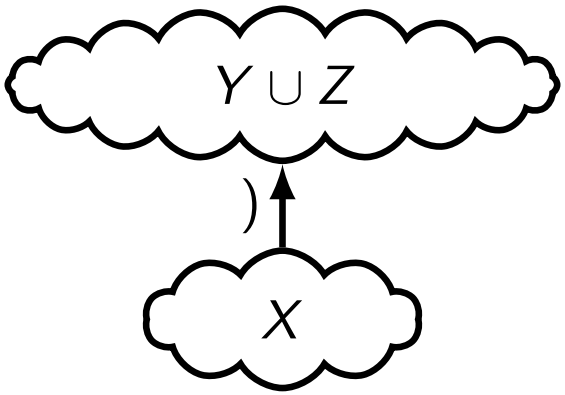
Reachability is Equivalence

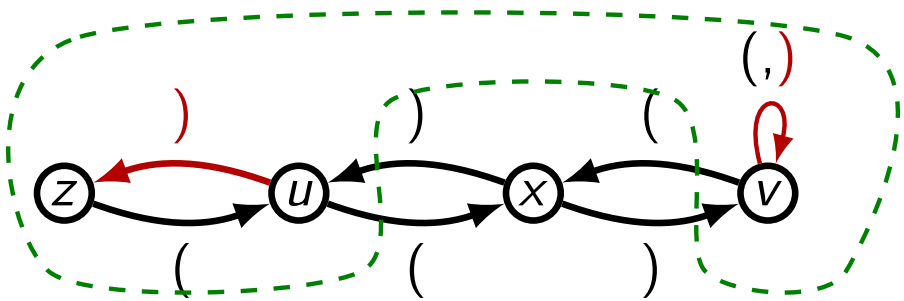
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components

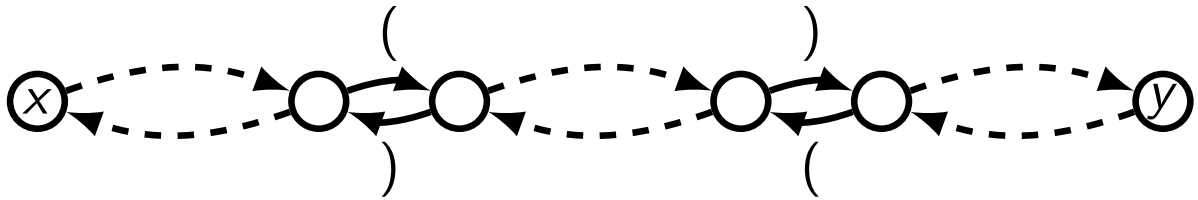




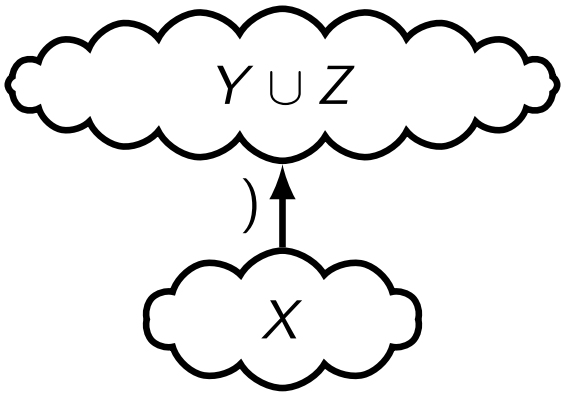
Reachability is Equivalence

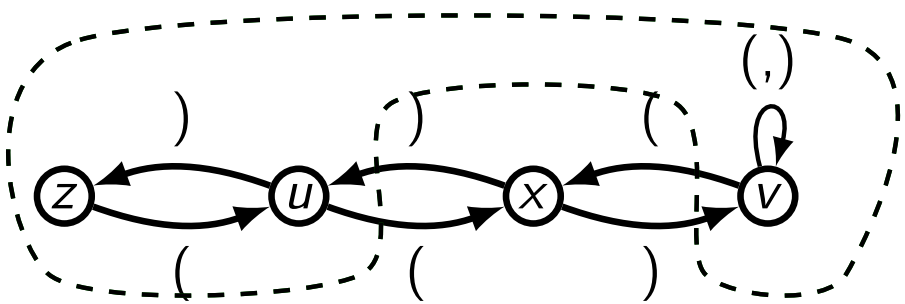
Theorem (ZLYS, PLDI '13)

Dyck reachability on bidirected graphs is an equivalence relation.



- Compute Strongly Connected Components





Implementation & Experiments

Implementation & Experiments

Alias Analysis

- Implementation in C++
- Compared with [ZLYS, PLDI '13]
- DaCapo-2006 benchmarks
- SPGs from [YXR, ISSTA '11]

Implementation & Experiments

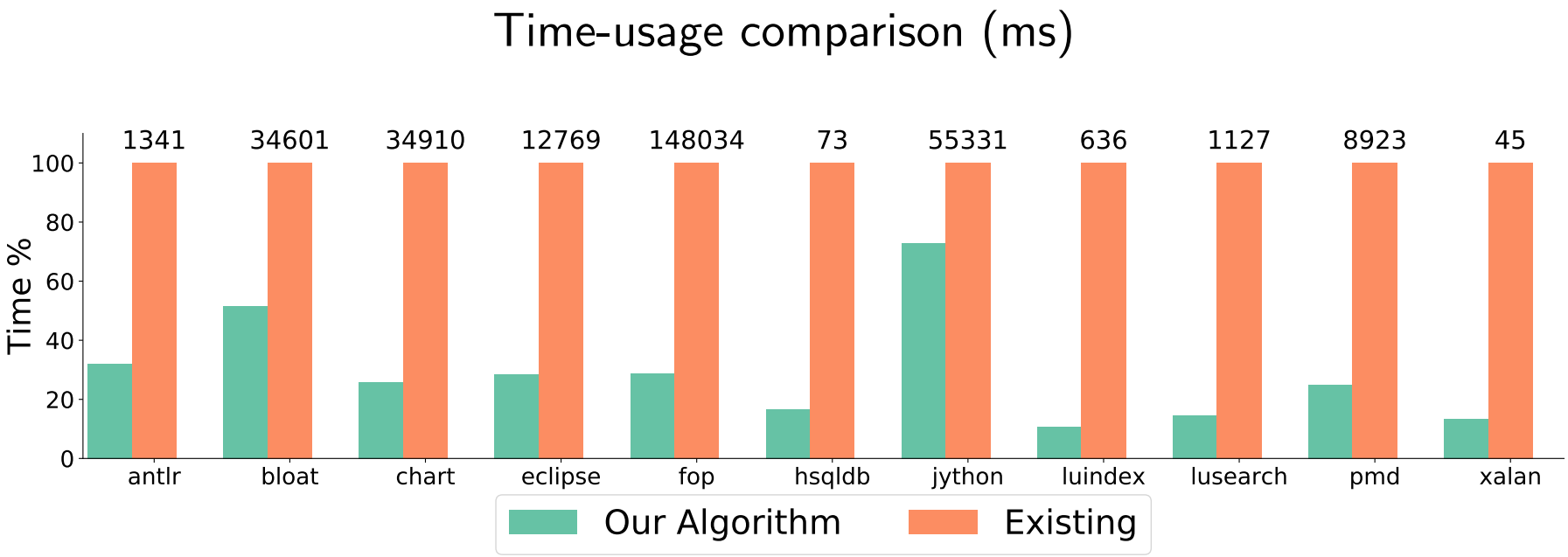
Alias Analysis

- Implementation in C++
- Compared with [ZLYS, PLDI '13]
- DaCapo-2006 benchmarks
- SPGs from [YXR, ISSTA '11]

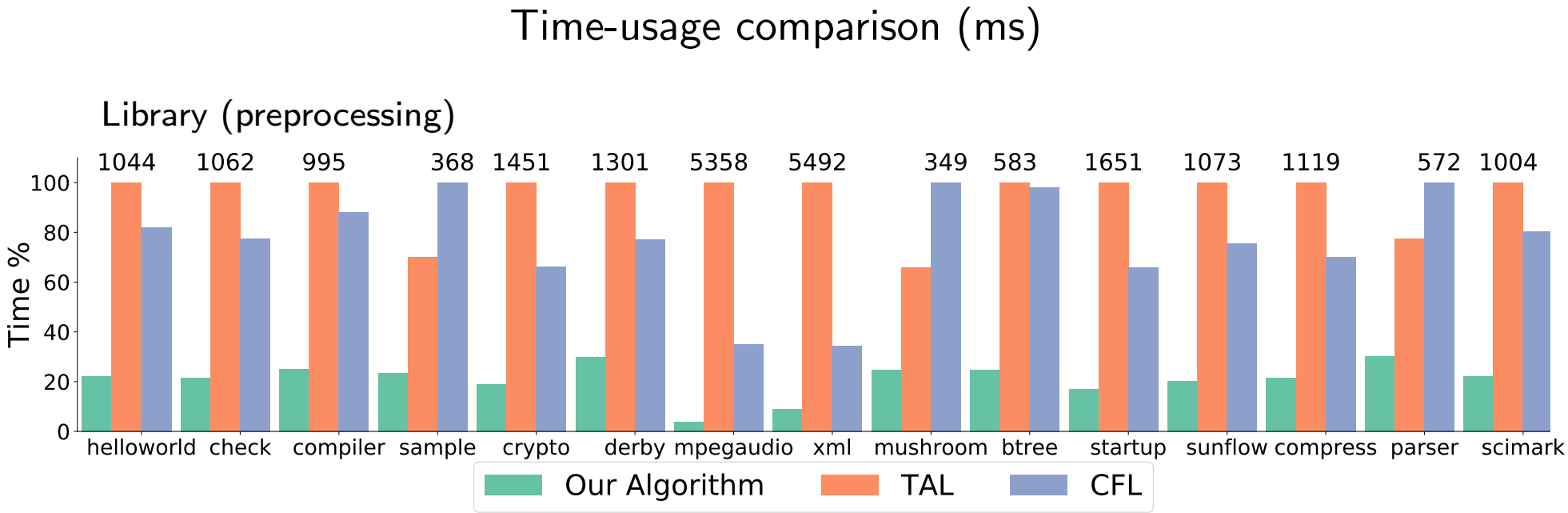
Data-dependence Analysis

- Implementation in Java
- Compared with TAL reachability [TWZXZM, POPL '15]
- Benchmarks:
  - SPECjvm2008
  - 4 randomly chosen GitHub projects

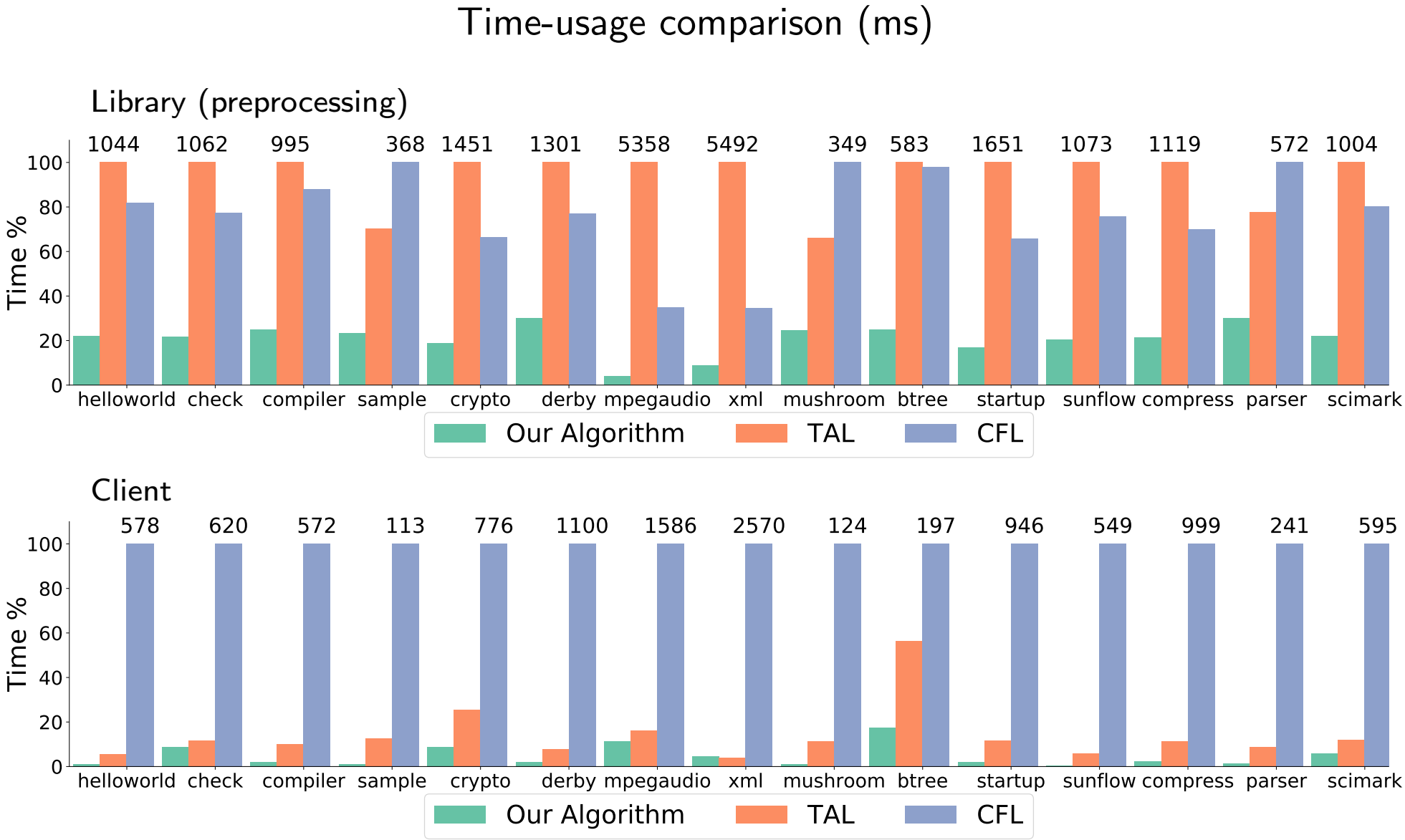
Alias Analysis



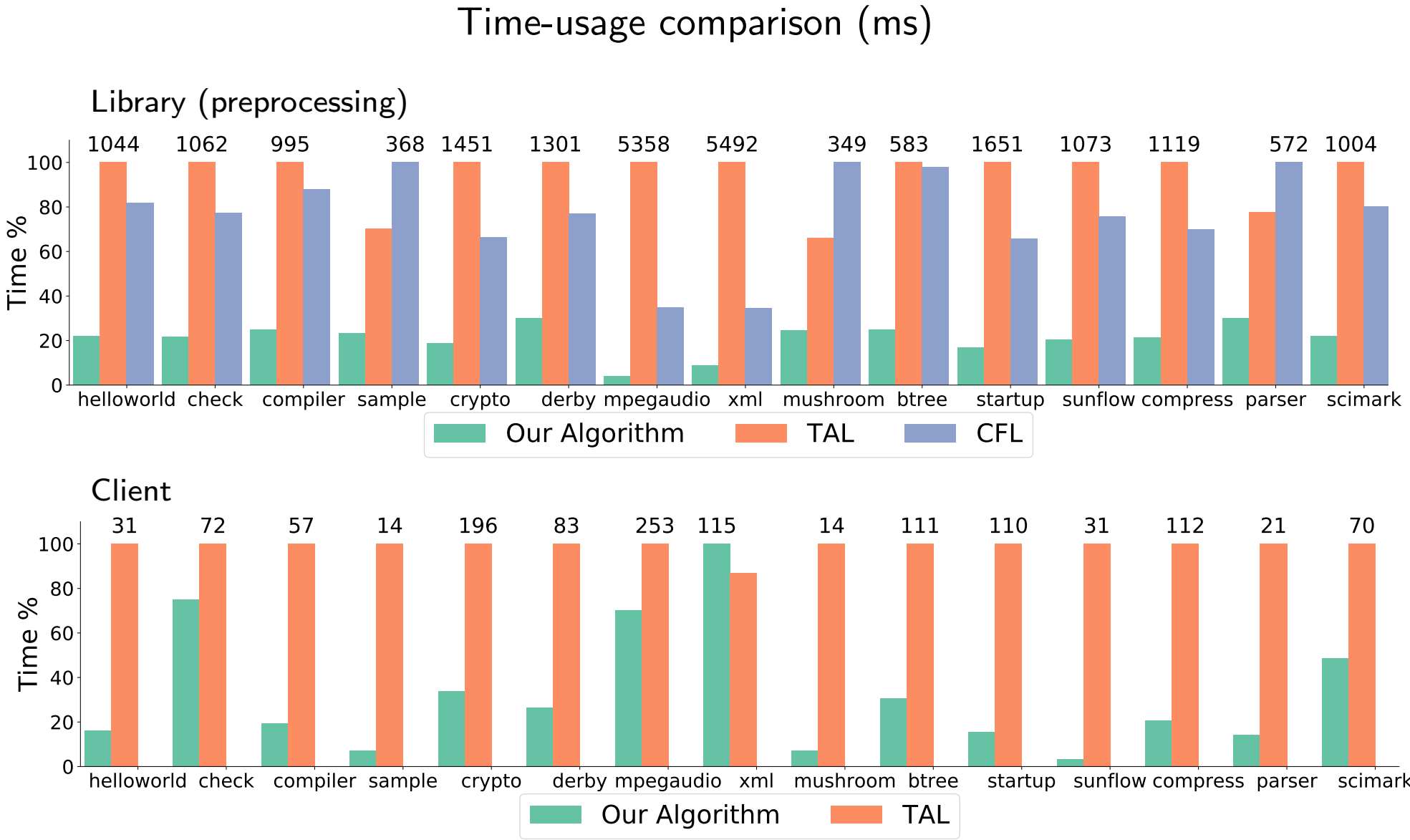
Data-dependence Analysis



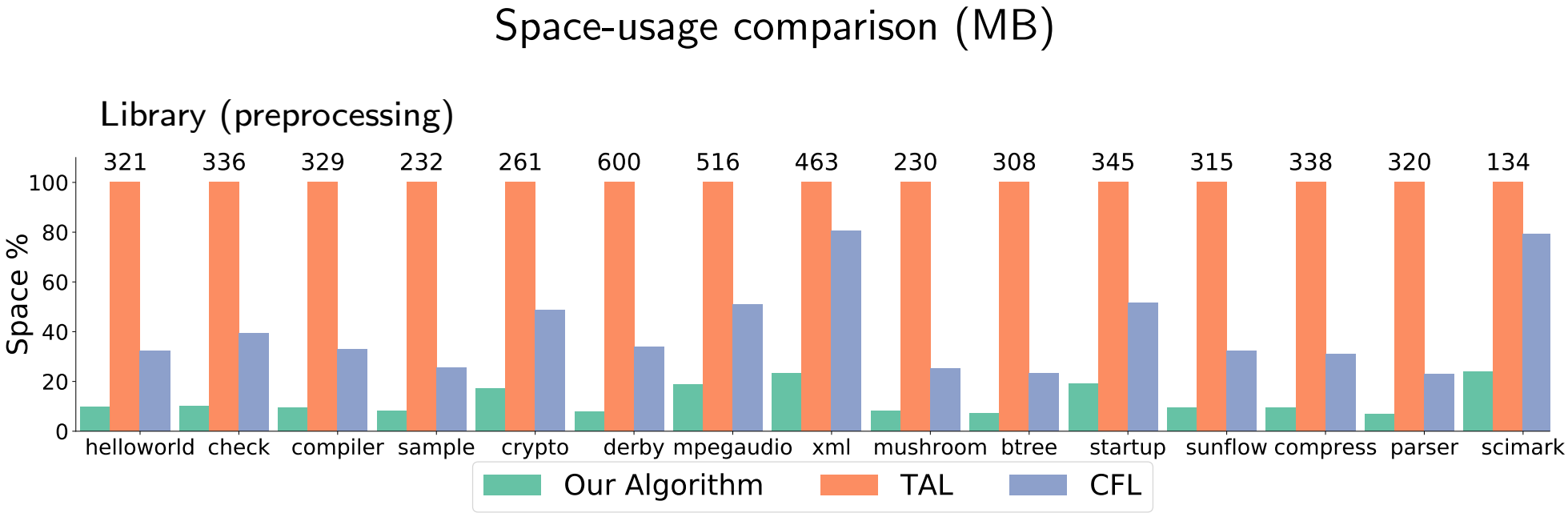
Data-dependence Analysis



Data-dependence Analysis



Data-dependence Analysis

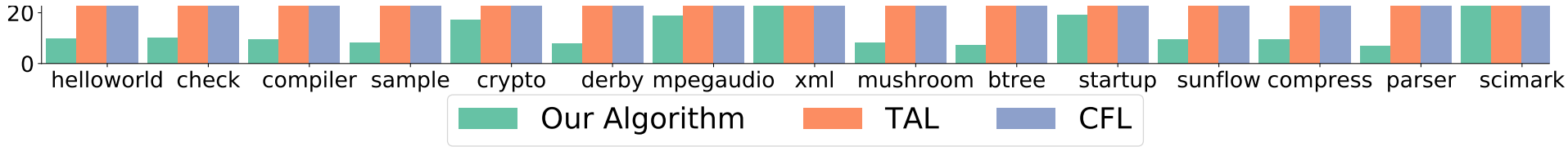


Data-dependence Analysis

Space-usage comparison (MB)

Library (preprocessing)

	Nodes		Our Algo		TAL		CFL	
Benchmark	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Li.	Cl.
serial	69999	468	130	130	>12000	>12000	3964	4314



- Graphs in Alias and Data-dependence analysis have special structure
- Faster, better algorithms for Dyck reachability
- Optimality guarantees

Thank you!
Questions?

Algorithmic Ingredients

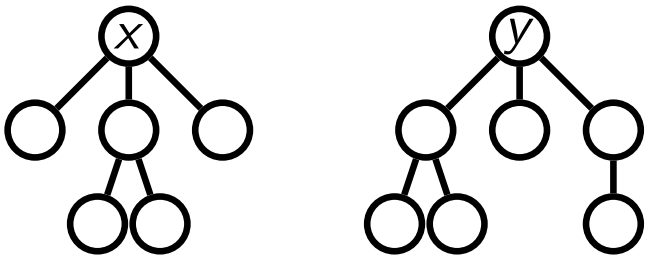
Need to make SCC merging fast

Algorithmic Ingredients

Need to make SCC merging fast

Merging nodes

- Disjoint-sets data structure
- $O(\alpha(n))$ time for merging and querying

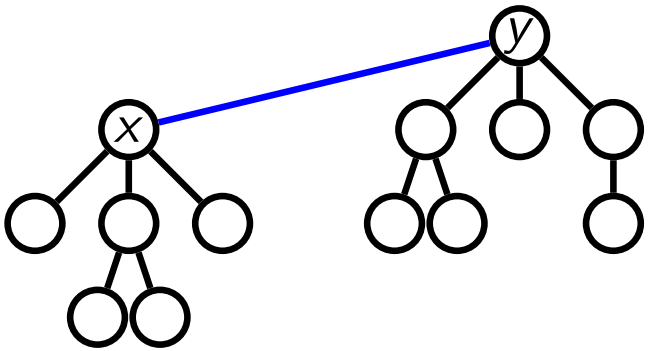


Algorithmic Ingredients

Need to make SCC merging fast

Merging nodes

- Disjoint-sets data structure
- $O(\alpha(n))$ time for merging and querying

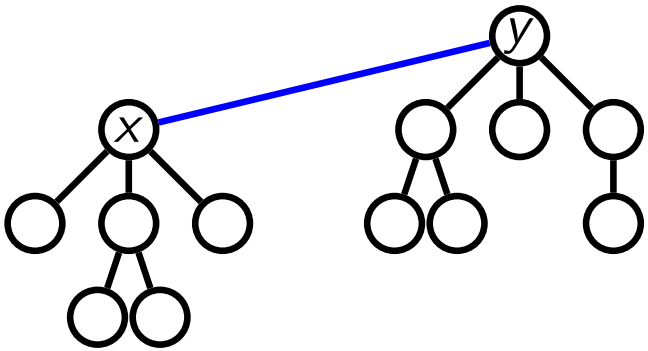


Algorithmic Ingredients

Need to make SCC merging fast

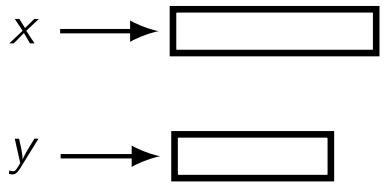
Merging nodes

- Disjoint-sets data structure
- $O(\alpha(n))$ time for merging and querying



Merging edges

- Only outgoing!
- Linked list representation
- $O(1)$ time for merging

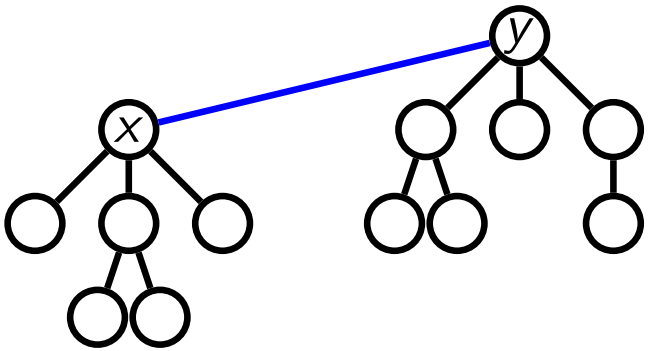


Algorithmic Ingredients

Need to make SCC merging fast

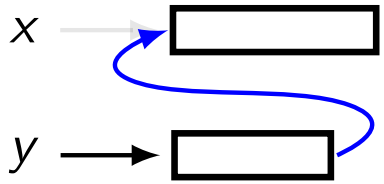
Merging nodes

- Disjoint-sets data structure
- $O(\alpha(n))$ time for merging and querying



Merging edges

- Only outgoing!
- Linked list representation
- $O(1)$ time for merging

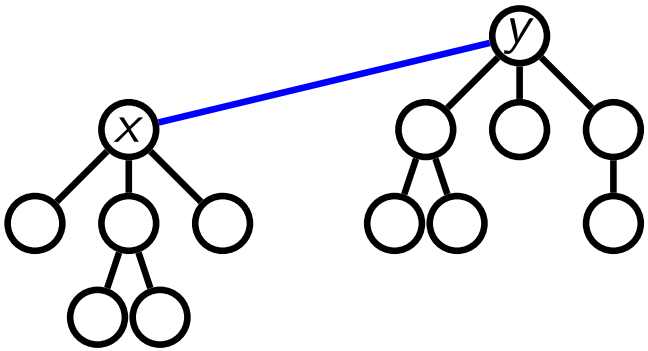


Algorithmic Ingredients

Need to make SCC merging fast

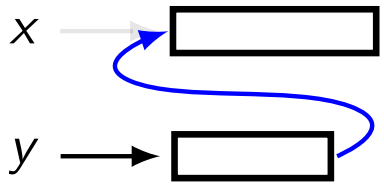
Merging nodes

- Disjoint-sets data structure
- $O(\alpha(n))$ time for merging and querying



Merging edges

- Only outgoing!
- Linked list representation
- $O(1)$ time for merging



Theorem

Dyck reachability on bidirected graphs solvable in

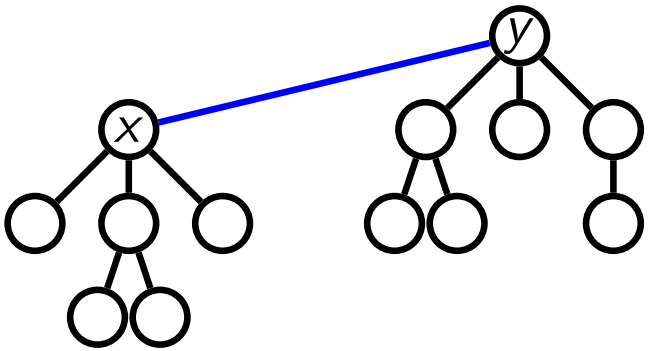
- $O(m + n \cdot \alpha(n))$ *worst-case time*
- $O(m)$ *average-case time*

Algorithmic Ingredients

Need to make SCC merging fast

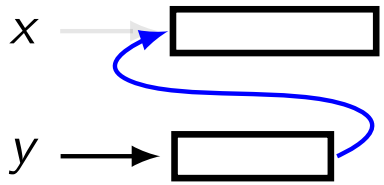
Merging nodes

- Disjoint-sets data structure
- $O(\alpha(n))$ time for merging and querying



Merging edges

- Only outgoing!
- Linked list representation
- $O(1)$ time for merging



Theorem

Dyck reachability on bidirected graphs solvable in

- $O(m + n \cdot \alpha(n))$ *worst-case time, optimally!*
- $O(m)$ *average-case time, optimally!*

Treewidth for Bidirected Graphs

Theorem

Dyck reachability on constant-treewidth, bidirected graphs requires $\Omega(m + n \cdot \alpha(n))$ *time.*

Treewidth for Bidirected Graphs

Theorem

Dyck reachability on constant-treewidth, bidirected graphs requires $\Omega(m + n \cdot \alpha(n))$ *time.*

Compare with

Theorem (ZLYZ, PLDI '13)

Dyck reachability on bidirected trees solvable in $O(n)$ *time.*

Typically complexity on low-treewidth graphs = complexity on trees

Complexity of Dyck reachability

- $O(n^3)$
- 2NPDA-hard
  - Conditional cubic lower bound

Complexity of Dyck reachability

- $O(n^3)$
- 2NPDA-hard
  - Conditional cubic lower bound

Theorem

Dyck reachability is Boolean Matrix Multiplication (BMM) - hard.

Complexity of Dyck reachability

- $O(n^3)$
- 2NPDA-hard
  - Conditional cubic lower bound

Theorem

Dyck reachability is Boolean Matrix Multiplication (BMM) - hard.

Corollary

An $O(n^{3-\delta})$ algorithm for Dyck reachability requires an $O(n^{3-\epsilon})$ combinatorial algorithm for BMM.

- $\delta, \epsilon > 0$

Existing Dyck-reachability algorithms conditionally optimal (s.t. BMM-hardness)

Complexity of Dyck reachability

- $O(n^3)$
- 2NPDA-hard
  - Conditional cubic lower bound

Theorem

Dyck reachability on low-treewidth graphs is Boolean Matrix Multiplication (BMM) - hard.

Corollary

An $O(n^{3-\delta})$ algorithm for Dyck reachability on low-treewidth graphs requires an $O(n^{3-\epsilon})$ combinatorial algorithm for BMM.

- $\delta, \epsilon > 0$

Existing Dyck-reachability algorithms conditionally optimal (s.t. BMM-hardness)

Alias Analysis, Time

Time-usage comparison (s)

Benchmark	Fields	Nodes	Edges	Our Algo	Existing Algo
antlr	172	13708	23547	0.428783	1.34152
bloat	316	43671	103361	17.7888	34.6012
chart	711	53500	91869	8.99378	34.9101
eclipse	439	34594	52011	3.62835	12.7697
fop	1064	101507	178338	42.5447	148.034
hsqldb	43	3048	4134	0.012899	0.073863
jython	338	56336	167040	40.239	55.3311
luindex	167	9931	14671	0.068013	0.636346
lusearch	200	12837	21010	0.163561	1.12788
pmd	357	31648	58025	2.21662	8.92306
xalan	41	2342	2979	0.006626	0.045144

Data-dependence Analysis, Time

Time-usage comparison (ms)

	Nodes		TW		**Our Algo**		**TAL**		**CFL**	
Benchmark	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Li.	Cl.
helloworld	16003	296	5	3	229	5	1044	31	855	578
check	16604	3347	5	4	228	54	1062	72	821	620
compiler	16190	536	5	3	248	11	995	57	876	572
sample	3941	28	4	1	86	1	258	14	368	113
crypto	20094	3216	5	5	273	66	1451	196	961	776
derby	23407	1106	6	3	389	22	1301	83	1003	1100
mpegaudio	28917	27576	5	24	204	177	5358	253	1864	1586
xml	71474	2312	5	3	489	115	5492	100	1891	2570
mushroom	3858	7	4	1	86	1	230	14	349	124
btree	6710	1103	4	4	144	34	583	111	571	197
startup	19312	621	5	3	279	17	1651	110	1087	946
sunflow	15615	85	5	2	217	1	1073	31	811	549
compress	16157	1483	5	3	240	23	1119	112	783	999
parser	7856	112	4	1	172	3	443	21	572	241
scimark	16270	2027	5	5	220	34	1004	70	805	595
serial	69999	468	8	3	440	9	MEMOUT	MEMOUT	117147	165958

Data-dependence Analysis, Space

Space-usage comparison (MB)

	Nodes		TW		**Our Algo**		**TAL**		**CFL**	
Benchmark	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Li.	Cl.
helloworld	16003	296	5	3	31	27	321	44	104	126
check	16604	3347	5	4	34	31	336	89	132	184
compiler	16190	536	5	3	31	28	329	44	108	137
sample	3941	28	4	1	19	16	232	59	59	64
crypto	20094	3216	5	5	45	45	261	61	127	188
derby	23407	1106	6	3	46	41	600	88	204	265
mpegaudio	28917	27576	5	24	96	96	516	219	262	397
xml	71474	2312	5	3	108	108	463	153	373	480
mushroom	3858	7	4	1	19	16	230	59	58	58
btree	6710	1103	4	4	22	19	308	65	72	89
startup	19312	621	5	3	66	66	345	92	178	230
sunflow	15615	85	5	2	30	27	315	43	102	124
compress	16157	1483	5	3	32	29	338	50	105	131
parser	7856	112	4	1	22	19	320	64	73	83
scimark	16270	2027	5	5	32	29	134	49	106	140
serial	69999	468	8	3	130	130	MEMOUT	MEMOUT	3964	4314

Data-dependence Analysis, Space

Space-usage comparison (MB)

	Nodes		TW		**Our Algo**		**TAL**		**CFL**	
Benchmark	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Lib.	Cl.	Li.	Cl.
helloworld	16003	296	5	3	31	27	321	44	104	126
check	16604	3347	5	4	34	31	336	89	132	184
compiler	16190	536	5	3	31	28	329	44	108	137
sample	3941	28	4	1	19	16	232	59	59	64
crypto	20094	3216	5	5	45	45	261	61	127	188
derby	23407	1106	6	3	46	41	600	88	204	265
mpegaudio	28917	27576	5	24	96	96	516	219	262	397
xml	71474	2312	5	3	108	108	463	153	373	480
mushroom	3858	7	4	1	19	16	230	59	58	58
btree	6710	1103	4	4	22	19	308	65	72	89
startup	19312	621	5	3	66	66	345	92	178	230
sunflow	15615	85	5	2	30	27	315	43	102	124
compress	16157	1483	5	3	32	29	338	50	105	131
parser	7856	112	4	1	22	19	320	64	73	83
scimark	16270	2027	5	5	32	29	134	49	106	140
serial	69999	468	8	3	130	130	MEMOUT	MEMOUT	3964	4314
							$>$ 12GB	$>$ 12GB		