New Algorithms for Static Analysis via Dyck Reachability

Andreas Pavlogiannis

4th Inria/EPFL Workshop

February 15, 2018
- Does it crash?
- Is it efficient?
- Does it leak information?
- Is it safe?
- Is it responsive?
Program analysis is hard
All non-trivial problems undecidable
Relax . . .
Program analysis is hard
All non-trivial problems undecidable
Relax . . .
  . . . your ambitions
  . . . your model
Static Analysis

- Program analysis is hard
- All non-trivial problems undecidable
- Relax . . .
  - . . . your ambitions
  - . . . your model

Static analysis
- Lightweight, “clever” scans of the program
- Detect “obvious” bugs
  1. Fast
  2. On demand
  3. Dynamic
Static Analysis via Dyck Reachability
\[ \Sigma = \{ (1,)_1, \ldots , (k,)_k \} \cup \{ \epsilon \} \]

\[ S \rightarrow SS \mid (1S)_1 \mid \ldots \mid (kS)_k \mid \epsilon \]

\[ G = (V, E, \lambda : E \rightarrow \Sigma) \]

\[ P : x \Rightarrow z \text{ with } \lambda(P) = (1(2)2)1 \]

Dyck Reachability

A. Pavlogiannis

New Algorithms for Static Analysis via Dyck Reachability
\[ \Sigma = \{(1,)_1, \cdots (k,)_k\} \cup \{\epsilon\} \]

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

\[ S \to S \; S \; | \; (1 \; S) \;_1 \; | \; \cdots \; |\; (k \; S) \;_k \; | \; \epsilon \]

\[ P : x \rightsquigarrow z \; \text{with} \; \lambda(P) = (1(2)2)_1 \]
\[ \Sigma = \{ (1,)_1, \ldots, (k,)_k \} \cup \{ \epsilon \} \]

\[ G = (V, E, \lambda : E \rightarrow \Sigma) \]

\[ S \rightarrow SS \mid (1 \; S \;)_1 \mid \ldots \mid (k \; S \;)_k \mid \epsilon \]

\[ P : x \leadsto z \; \text{with} \; \lambda(P) = (1(2)_2)_1 \]

- Alias analysis
- Data-dependence analysis
- Data-flow analysis
- Shape analysis
- Impact analysis
- Bloat analysis
- Type-based flow analysis
- Program slicing
Q: Is \( v \) Dyck-reachable from \( u \)?

- Dyck languages are a class of CFL
- Solution similar to CYK for CFL parsing
- \( O(n^3) \) [Yannakakis '90]
- \( O(n^3 / \log n) \) [Chaudhuri '08]
- Often prohibitive for lightweight static analysis
Q: Is \( v \) Dyck-reachable from \( u \)?

- Dyck languages are a class of CFL
- Solution similar to CYK for CFL parsing
- \( O(n^3) \) [Yannakakis '90]
- \( O(n^3 / \log n) \) [Chaudhuri '08]
- Often prohibitive for lightweight static analysis

Can do better if the graph is simple!

1. Low treewidth
2. Bidirected
Static Analysis

- Analyze source code without executing it

Typical paradigm in static analysis: reduce the problem to a graph problem $P$:

Input: A program of methods $M_i$

1. Extract control flow graphs $G_i$
2. Annotate $G_i$
3. Run best graph algorithm for $P$ on $G$

Data-dependence analysis
Which variable depends on which others
Identify Def-Use chains in a program
$LHS \text{ depends on } RHS$

$x \leftarrow y \leftarrow x + 1$

$y \text{ depends on } x$ if $x \Rightarrow y$
Static Analysis

- Analyze source code without executing it

  Typical paradigm in static analysis: reduce the problem to a graph problem $P$:
  Input: A program of methods $M_i$
  1. Extract control flow graphs $G_i$
  2. Annotate $G_i$
  3. Run best graph algorithm for $P$ on $G$

Data-dependence analysis
- Which variable depends on which others
- Identify Def-Use chains in a program
- LHS depends on RHS
Static Analysis

- Analyze source code without executing it

Typical paradigm in static analysis: reduce the problem to a graph problem $P$:
Input: A program of methods $M_i$

1. Extract control flow graphs $G_i$
2. Annotate $G_i$
3. Run best graph algorithm for $P$ on $G$

Data-dependence analysis
- Which variable depends on which others
- Identify Def-Use chains in a program
- LHS depends on RHS
  $y$ depends on $x$ if $x \leadsto y$

\[
\begin{align*}
1 & \quad x \leftarrow 2 \\
2 & \quad y \leftarrow x + 1
\end{align*}
\]
Method 1: \texttt{dot\_vector}

1. \texttt{result} $\leftarrow 0$
2. \texttt{for $i \leftarrow 1$ to $n$ do}
3. \hspace{1em} $z \leftarrow x[i] \cdot y[i]$
4. \hspace{1em} \texttt{result} $\leftarrow \texttt{result} + z$
5. \texttt{end}
6. \texttt{return result}

Method 2: \texttt{dot\_matrix}

1. $C \leftarrow \text{zero matrix of size } n \times m$
2. \texttt{for $i \leftarrow 1$ to $n$ do}
3. \hspace{1em} \texttt{for $j \leftarrow 1$ to $m$ do}
4. \hspace{2em} \text{Call \texttt{dot\_vector}(A[i,:], B[:,j])}
5. \hspace{2em} $C[i,j] \leftarrow$ the value returned by the call of line 4
6. \texttt{end}
7. \texttt{end}
8. \texttt{return } C
Inter-procedural → Recursive Graphs

Method 1: $f_1()$
1 ...  
2 $a \leftarrow g()$  
3 ... 

Method 2: $f_2()$
1 ...  
2 $b \leftarrow g()$  
3 ... 

Method 3: $g()$
1 ... 

Filter out interprocedurally invalid paths
Inter-procedural → Recursive Graphs

Method 1: $f_1()$

1 \[\ldots\]
2 $a \leftarrow g()$
3 \[\ldots\]

Method 3: $g()$

1 \[\ldots\]
2 $b \leftarrow g()$
3 \[\ldots\]

Method 2: $f_2()$

1 \[\ldots\]

Filter out interprocedurally invalid paths
**Method 4: dot_vector**

1. \( \text{result} \leftarrow 0 \)
2. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
3. \( z \leftarrow x[i] \cdot y[i] \)
4. \( \text{result} \leftarrow \text{result} + z \)
5. \( \text{end} \)
6. \( \text{return } \text{result} \)

**Method 5: dot_matrix**

1. \( C \leftarrow \text{zero matrix of size } n \times m \)
2. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
3. \( \text{for } j \leftarrow 1 \text{ to } m \text{ do} \)
4. \( \text{Call } \text{dot_vector}(A[i, :], B[:, j]) \)
5. \( C[i, j] \leftarrow \text{the value returned by the call of line 4} \)
6. \( \text{end} \)
7. \( \text{end} \)
8. \( \text{return } C \)
Control-flow graphs are simple
“Similar to trees”
Precise notion: treewidth
Treewidth
Cops and Robber

- You enter the metro and “forget” to buy a ticket
- \( t \) inspectors are looking for you in the metro stations
- Everyone is aware of everyone else’s position
Cops and Robber

- You enter the metro and “forget” to buy a ticket
- $t$ inspectors are looking for you in the metro stations
- Everyone is aware of everyone else’s position
- In every round
  - Some inspectors move between stations to reach you
  - You use the metro system to move between stations
  - You cannot cross an occupied station
You enter the metro and “forget” to buy a ticket

\( t \) inspectors are looking for you in the metro stations

Everyone is aware of everyone else’s position

In every round

- Some inspectors move between stations to reach you
- You use the metro system to move between stations
- You cannot cross an occupied station

The **treewidth** of the metro graph is the largest \( t \) from which you can always escape
Definition (Tree decomposition)

Given a graph $G = (V, E)$, a **tree-decomposition** $\text{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$. 

$G$

![Graph G](image)

$\text{Tree}(G)$

![Tree of bags](image)
Definition (Treewidth)

The width of $\text{Tree}(G)$ is $\max_i |B_i| - 1$. The treewidth of $G$ is the minimum width of a tree decomposition of $G$. 

![Diagram of G and Tree(G)]
Definition (Treewidth)

The width of $\text{Tree}(G)$ is $\max_i |B_i| - 1$. The treewidth of $G$ is the minimum width of a tree decomposition of $G$.

CFGs of typical imperative programs have small treewidth

- For goto-free programs [Thorup '98]
  - Pascal $\leq 3$
  - C $\leq 6$
- In practice small in imperative programs (e.g. Java $\leq 7$)
**Definition (Treewidth)**

The width of $\text{Tree}(G)$ is $\max_i |B_i| - 1$. The treewidth of $G$ is the minimum width of a tree decomposition of $G$.

**CFGs of typical imperative programs have small treewidth**

- For goto-free programs [Thorup ’98]
  - Pascal $\leq 3$
  - C $\leq 6$
- In practice small in imperative programs (e.g. Java $\leq 7$)

**Theorem (Tree decomposition)**

*For constant treewidth graphs, $\text{Tree}(G)$ can be constructed in $O(n)$ time*
**Algorithmic Principle**

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

Graph:

$$G_h$$

$$G_g$$

$$G_f$$

Nodes: 1, 2, 3, 4, 5, 6, 7, 8

Edges:
- (1) from 1 to 3
- (2) from 2 to 8
- (1) from 3 to 4
- (2) from 4 to 7
- (1) from 5 to 6
- (1) from 6 to 7
- (2) from 7 to 8

Most time spent in (1)
Incremental reachability
Low treewidth
Algorithmic Principle

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

Graphical representation:

$G_h$
$G_g$
$G_f$
1. Reachability ignoring parenthesis edges

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

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

- Most time spent in (1)
- Incremental reachability
- Low treewidth
Reachability on low-treewidth graphs
Given a graph $G = (V, E)$ of $n$ nodes and bounded treewidth

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Space</th>
<th>Pair query time</th>
<th>Single-source query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O \left( \frac{n^2}{\log n} \right)$</td>
<td>$O \left( \frac{n^2}{\log n} \right)$</td>
<td>$O(1)$</td>
<td>$O \left( \frac{n}{\log n} \right)$</td>
</tr>
<tr>
<td>$O(n \cdot \log n)$</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O \left( \frac{n}{\log n} \right)$</td>
</tr>
</tbody>
</table>

- Optimal
- Faster than DFS/BFS after a constant number of queries
Given a graph $G = (V, E)$ of $n$ nodes and bounded treewidth

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Space</th>
<th>Pair query time</th>
<th>Single-source query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
<tr>
<td>$O(n \cdot \log n)$</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
</tbody>
</table>

- Optimal
- Faster than DFS/BFS after a constant number of queries
Given a graph $G = (V, E)$ of $n$ nodes and bounded treewidth.

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Space</th>
<th>Pair query time</th>
<th>Single-source query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
<tr>
<td>$O(n \cdot \log n)$</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
</tbody>
</table>

- Optimal
- Faster than DFS/BFS after a constant number of queries
Given a graph $G = (V, E)$ of $n$ nodes and bounded treewidth

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Space</th>
<th>Pair query time</th>
<th>Single-source query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
<tr>
<td>$O(n \cdot \log n)$</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
</tbody>
</table>

- Optimal
- Faster than DFS/BFS after a constant number of queries
Given a graph $G = (V, E)$ of $n$ nodes and bounded treewidth

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Space</th>
<th>Pair query time</th>
<th>Single-source query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$- O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
<tr>
<td>$O(n \cdot \log n)$</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
</tbody>
</table>

- Optimal
- Faster than DFS/BFS after a constant number of queries
Interprocedural Dataflow Analysis on Recursive Graphs of Low Treewidth
Interprocedural Dataflow Analysis

Input:
- Recursive graph $G$ with $n$ nodes
- Finite graphs $G_i$ of $n_i$ nodes and bounded treewidth
  - Every $G_i$ has a unique entry and exit

Theorem (POPL'15)
For low-treewidth graphs we construct dynamic reachability oracles:
- Preprocess $G_i$ in $O(n_i \cdot |D|^3)$ time
- Query for the distance $d(u, v)$ in $O(|D|^3 \cdot \log n_i)$ time
- Update the weight of an edge $(u, v)$ in $O(|D|^3 \cdot \log n_i)$ time
Interprocedural Dataflow Analysis

Input:
- Recursive graph $G$ with $n$ nodes
- Finite graphs $G_i$ of $n_i$ nodes and bounded treewidth
  - Every $G_i$ has a unique entry and exit
- $D$ data fact set
  - e.g. “variable $x$ is uninitialized”, “variable $y$ is constant”
- Weighted graph
- Semiring distance on lattice over $D$
Interprocedural Dataflow Analysis

Input:
- Recursive graph $G$ with $n$ nodes
- Finite graphs $G_i$ of $n_i$ nodes and bounded treewidth
  - Every $G_i$ has a unique entry and exit
- $D$ data fact set
  - e.g. “variable $x$ is uninitialized”, “variable $y$ is constant”
- Weighted graph
- Semiring distance on lattice over $D$

Theorem (POPL’15)

For low-treewidth graphs we construct dynamic reachability oracles:
- **Preprocess** $G_i$ in $O(n_i \cdot |D|^3)$ time
- **Query** for the distance $d(u, v)$ in $O(|D|^3 \cdot \log n_i)$ time
- **Update** the weight of an edge $(u, v)$ in $O(|D^3| \cdot \log n_i)$ time
Interprocedural Dataflow Analysis
Interprocedural Dataflow Analysis

0 \rightarrow G_1 \rightarrow G_2 \rightarrow \ldots \rightarrow G_k

Preprocess  Preprocess  Preprocess
Interprocedural Dataflow Analysis

0

\[ G_1 \quad G_2 \quad \ldots \quad G_k \]

1

\[ G_1 \quad G_2 \quad \ldots \quad G_k \]
Interprocedural Dataflow Analysis

Query \( d(En_1, Ex_1) \)

Query \( d(En_i, Ex_i) \)
Interprocedural Dataflow Analysis

Query $d(En_1, Ex_1)$

Update $wt_2(C, R)$
Interprocedural Dataflow Analysis

\[ G_1 \quad G_2 \quad \cdots \quad G_k \]

\[ G_1 \quad G_2 \quad \cdots \quad G_k \]

\[ G_1 \quad G_2 \quad \cdots \quad G_k \]
### Interprocedural Dataflow Analysis

<table>
<thead>
<tr>
<th></th>
<th>$G_1$</th>
<th>$G_2$</th>
<th>...</th>
<th>$G_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Query** $d(u, v)$ in $O(\log n_2)$
Interprocedural Dataflow Analysis

A. Pavlogiannis

New Algorithms for Static Analysis via Dyck Reachability 21
Input graph with $n$ nodes, $D$ data fact set, $|D| = \Omega(\log n)$

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing time</th>
<th>Space</th>
<th>Query Single-source</th>
<th>Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>No preprocessing</td>
<td>-</td>
<td>$O(n \cdot</td>
<td>D</td>
<td>^2)$</td>
</tr>
<tr>
<td>Complete preprocessing</td>
<td>$O(n^2 \cdot</td>
<td>D</td>
<td>^3)$</td>
<td>$O(n^2 \cdot</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>$O(n \cdot</td>
<td>D</td>
<td>^3)$</td>
<td>$O(n</td>
</tr>
</tbody>
</table>
Dataflow Analysis [POPL’15]

Input graph with $n$ nodes, $D$ data fact set, $|D| = \Omega(\log n)$

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing time</th>
<th>Space</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single-source</td>
</tr>
<tr>
<td>No preprocessing</td>
<td>-</td>
<td>$O(n \cdot</td>
<td>D</td>
</tr>
<tr>
<td>Complete preprocessing</td>
<td>$O(n^2 \cdot</td>
<td>D</td>
<td>^3)$</td>
</tr>
<tr>
<td>Ours</td>
<td>$O(n \cdot</td>
<td>D</td>
<td>^3)$</td>
</tr>
</tbody>
</table>
Dyck Reachability on Bidirected Graphs
∀u, v ∈ V : λ(u, v) = (i) ⇔ λ(v, u) = )i
Bidirected graphs

\[ \forall u, v \in V : \lambda(u, v) = (i) \iff \lambda(v, u) = )i \]

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

*Dyck reachability on bidirected graphs is an equivalence relation.*
Reachability is Equivalence

Theorem (ZLYS '13)

Dyck reachability on bidirected graphs is an equivalence relation.
Reachability is Equivalence

Theorem (ZLYS '13)

*Dyck reachability on bidirected graphs is an equivalence relation.*
Graphs of $n$ nodes, $\alpha(n)$ the inverse Ackermann

<table>
<thead>
<tr>
<th>Source</th>
<th>Worst-case Time</th>
<th>Average-case Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ZLYS '13]</td>
<td>$O(n^2)$</td>
<td>$O(n \cdot \log n)$</td>
</tr>
<tr>
<td>Our Result</td>
<td>$O(n \cdot \alpha(n))$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Graphs of $n$ nodes, $\alpha(n)$ the inverse Ackermann

<table>
<thead>
<tr>
<th>Source</th>
<th>Worst-case Time</th>
<th>Average-case Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ZLYS '13]</td>
<td>$O(n^2)$</td>
<td>$O(n \cdot \log n)$</td>
</tr>
<tr>
<td><strong>Our Result</strong></td>
<td>$O(n \cdot \alpha(n))$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

optimal! optimal!
Hardness?
$O(n^3)$
- 2NPDA-hard
  - Conditional cubic lower bound
- $O(n^3)$
- 2NPDA-hard
  - Conditional cubic lower bound

**Theorem**

*Dyck reachability is Boolean Matrix Multiplication - hard.*
Complexity of Dyck Reachability [POPL’18]

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

Theorem

*Dyck reachability is Boolean Matrix Multiplication - hard.*
Complexity of Dyck Reachability [POPL’18]

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

Theorem

*Dyck reachability on low-treewidth graphs is Boolean Matrix Multiplication - hard.*

Can we do $O(n^3 / \log^2 n)$?
Theorem

Dyck reachability on constant-treewidth, bidirected graphs requires \( \Omega(n \cdot \alpha(n)) \) time.
Complexity on Bidirected Graphs [POPL’18]

Theorem

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

Compare with

Theorem (ZLYZ ’13)

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

Typically complexity on low-treewidth graphs = complexity on trees
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

- 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
Time-usage comparison (ms)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Our Algorithm</th>
<th>Existing</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>1341</td>
<td>34601</td>
</tr>
<tr>
<td>bloat</td>
<td>34910</td>
<td>34910</td>
</tr>
<tr>
<td>chart</td>
<td>12769</td>
<td>148034</td>
</tr>
<tr>
<td>eclipse</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>fop</td>
<td>55331</td>
<td>636</td>
</tr>
<tr>
<td>hsqldb</td>
<td>1127</td>
<td>1127</td>
</tr>
<tr>
<td>jython</td>
<td>8923</td>
<td>45</td>
</tr>
<tr>
<td>luindex</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>lusearch</td>
<td>1341</td>
<td>34601</td>
</tr>
<tr>
<td>pmd</td>
<td>34910</td>
<td>34910</td>
</tr>
<tr>
<td>xalan</td>
<td>12769</td>
<td>148034</td>
</tr>
</tbody>
</table>

Our Algorithm

Existing
Data-dependence Analysis

Time-usage comparison (ms)

Library (preprocessing)

Client

Our Algorithm
TAL
CFL

A. Pavlogiannis
New Algorithms for Static Analysis via Dyck Reachability

Data-dependence Analysis

Time-usage comparison (ms)

Library (preprocessing)

- helloworld
- check
- compiler
- sample
- crypto
- derby
- mpegaudio
- xml
- mushroom
- btree
- startup
- sunflow
- compress
- parser
- scimark

Our Algorithm
TAL
CFL

Client

- helloworld
- check
- compiler
- sample
- crypto
- derby
- mpegaudio
- xml
- mushroom
- btree
- startup
- sunflow
- compress
- parser
- scimark

Our Algorithm
TAL
Data-dependence Analysis

Space-usage comparison (MB)

Library (preprocessing)

<table>
<thead>
<tr>
<th>Library</th>
<th>Our Algorithm</th>
<th>TAL</th>
<th>CFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelloWorld</td>
<td>321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check</td>
<td>336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sample</td>
<td>232</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crypto</td>
<td>261</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derby</td>
<td>600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MpegAudio</td>
<td>516</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xml</td>
<td>463</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mushroom</td>
<td>230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTree</td>
<td>308</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Startup</td>
<td>345</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sunflow</td>
<td>315</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compress</td>
<td>338</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parser</td>
<td>320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scimark</td>
<td>134</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A. Pavlogiannis
New Algorithms for Static Analysis via Dyck Reachability
## Data-dependence Analysis

### Space-usage comparison (MB)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Library (Lib.)</th>
<th>Clustering (Cl.)</th>
<th>Library (Lib.)</th>
<th>Clustering (Cl.)</th>
<th>Library (Lib.)</th>
<th>Clustering (Cl.)</th>
<th>Library (Lib.)</th>
<th>Clustering (Cl.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>69999</td>
<td>468</td>
<td>130</td>
<td>130</td>
<td>&gt;12000</td>
<td>&gt;12000</td>
<td>3964</td>
<td>4314</td>
</tr>
</tbody>
</table>

### Library (preprocessing)

<table>
<thead>
<tr>
<th>Library</th>
<th>Our Algorithm</th>
<th>TAL</th>
<th>CFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello world</td>
<td>321</td>
<td>336</td>
<td>329</td>
</tr>
<tr>
<td>check</td>
<td>232</td>
<td>600</td>
<td>516</td>
</tr>
<tr>
<td>compiler</td>
<td>261</td>
<td>463</td>
<td>230</td>
</tr>
<tr>
<td>sample</td>
<td>308</td>
<td>345</td>
<td>315</td>
</tr>
<tr>
<td>crypto</td>
<td>338</td>
<td>320</td>
<td>134</td>
</tr>
</tbody>
</table>
Thank you!
Questions?