

Software Challenges: Abstraction and Analysis

Martin Bravenboer

University of Massachusetts Amherst

University of Oregon

April 2, 2009

University of Waterloo

the design and implementation of
programming language features and
software engineering tools for solving
software development problems

extensible and domain-specific languages

brief overview

program transformation

no time :(

program analysis

latest work, rest of the talk

examples of poor abstraction and
language integration

```
String userName = getParam("userName");
String password = getParam("password");
String query = "SELECT id FROM users "
               + "WHERE name = '" + userName + "' "
               + "AND password = '" + password + "'";
if (executeQuery(query).size() == 0)
    throw new Exception("bad user/password");
```

```
String userName = getParam("userName");  
String password = getParam("password");  
String query = "SELECT id FROM users "  
               + "WHERE name = '" + userName + "' "  
               + "AND password = '" + password + "'";  
if (executeQuery(query).size() == 0)  
    throw new Exception("bad user/password");
```

password ilovewaterloo

```
SELECT id FROM users  
WHERE name = 'martin' AND  
       password = 'ilovewaterloo'
```



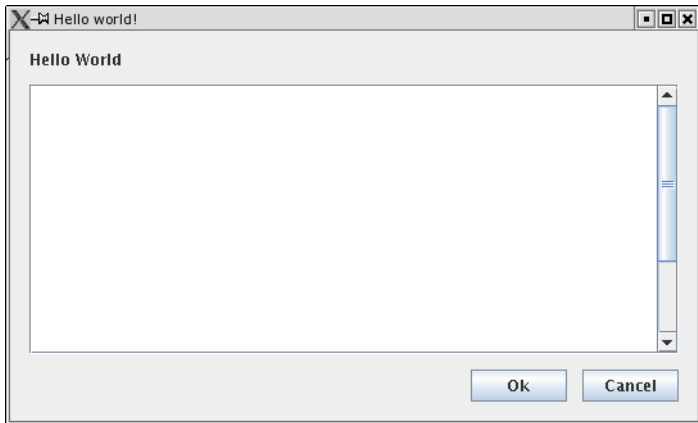
```
String userName = getParam("userName");
String password = getParam("password");
String query = "SELECT id FROM users "
               + "WHERE name = '" + userName + "' "
               + "AND password = '" + password + "'";
if (executeQuery(query).size() == 0)
    throw new Exception("bad user/password");
```

password ' OR 'x' = 'x'

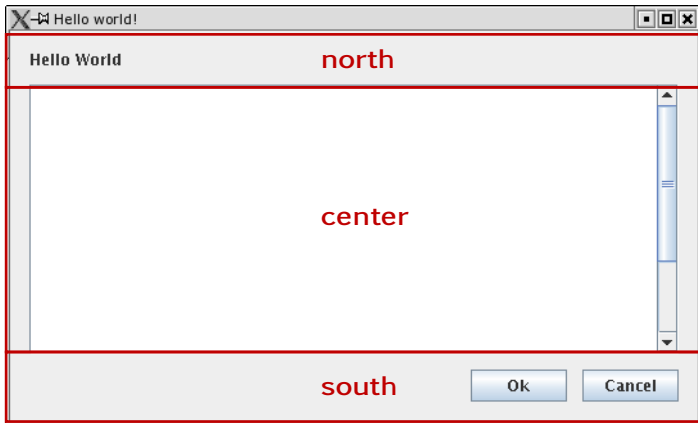
```
SELECT id FROM users
WHERE name = 'dr.evil' AND
      password = '' OR 'x' = 'x'
```



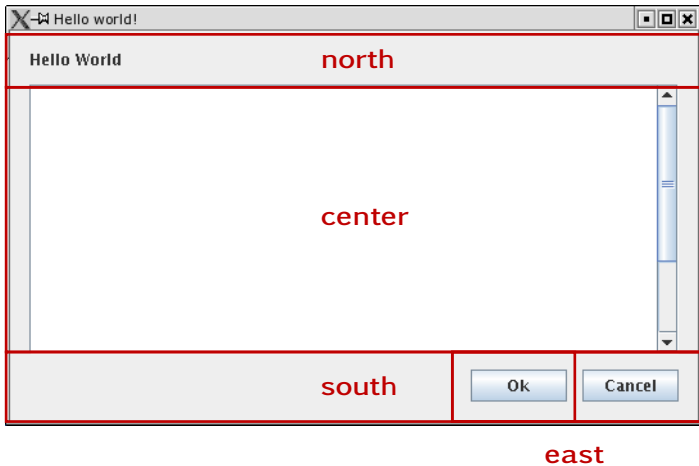
example 2: implement a graphical user-interface 4



example 2: implement a graphical user-interface 4



example 2: implement a graphical user-interface 4



example 2: implement a graphical user-interface 5

```
public class HelloWorld {  
    public static void main(String[] ps) {  
  
        JTextArea text = new JTextArea(20,40);  
  
        JPanel panel = new JPanel(new BorderLayout(12,12));  
        panel.add(BorderLayout.NORTH , new JLabel("Hello World"));  
        panel.add(BorderLayout.CENTER , new JScrollPane(text));  
  
        JPanel south = new JPanel(new BorderLayout(12,12));  
        JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));  
        buttons.add(new JButton("Ok"));  
        buttons.add(new JButton("Cancel"));  
  
        south.add(BorderLayout.EAST, buttons);  
        panel.add(BorderLayout.SOUTH, south);  
        ...  
    }  
}
```

example 2: implement a graphical user-interface 5

```
public class HelloWorld {  
    public static void main(String[] ps) {  
  
        JTextArea text = new JTextArea(20,40);  
  
        JPanel panel = new JPanel(new BorderLayout(12,12));  
        panel.add(BorderLayout.NORTH , new JLabel("Hello World"));  
        panel.add(BorderLayout.CENTER , new JScrollPane(text));  
  
        JPanel south = new JPanel(new BorderLayout(12,12));  
        JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));  
        buttons.add(new JButton("Ok"));  
        buttons.add(new JButton("Cancel"));  
  
        south.add(BorderLayout.EAST, buttons);  
        panel.add(BorderLayout.SOUTH, south);  
        ...  
    }  
}
```

example 2: implement a graphical user-interface 5

```
public class HelloWorld {  
    public static void main(String[] ps) {  
  
        JTextArea text = new JTextArea(20,40);  
  
        JPanel panel = new JPanel(new BorderLayout(12,12));  
        panel.add(BorderLayout.NORTH , new JLabel("Hello World"));  
        panel.add(BorderLayout.CENTER , new JScrollPane(text));  
  
        JPanel south = new JPanel(new BorderLayout(12,12));  
        JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));  
        buttons.add(new JButton("Ok"));  
        buttons.add(new JButton("Cancel"));  
  
        south.add(BorderLayout.EAST, buttons);  
        panel.add(BorderLayout.SOUTH, south);  
        ...  
    }  
}
```

Does not correspond to hierarchical structure of the user-interface.

example 2: implement a graphical user-interface 5

```
public class HelloWorld {  
    public static void main
```

```
    JTextArea text = new JTextArea(20,40);
```

```
    JPanel panel = new JPanel(new BorderLayout(12,12));  
    panel.add(BorderLayout.NORTH , new JLabel("Hello World"));  
    panel.add(BorderLayout.CENTER , new JScrollPane(text));
```

```
    JPanel south = new JPanel(new BorderLayout(12,12));  
    JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));  
    buttons.add(new JButton("Ok"));  
    buttons.add(new JButton("Cancel"));
```

```
    south.add(BorderLayout.EAST, buttons);  
    panel.add(BorderLayout.SOUTH, south);  
    ...
```

General-purpose languages do not give you the means to write coherent sentences over the vocabulary of an API

Does not correspond to hierarchical structure of the user-interface.

```
public void testBuysWhenPriceEqualsThreshold() {  
    mainframe.expects(once())  
        .method("buy").with(eq(QUANTITY))  
        .will(returnValue(TICKET));  
  
    auditing.expects(once())  
        .method("bought").with(same(TICKET));  
  
    agent.onPriceChange(THRESHOLD);  
}
```

```
public void testBuysWhenPriceEqualsThreshold() {  
    mainframe.expects(once())  
        .method("buy").with(eq(QUANTITY))  
        .will(returnValue(TICKET));  
  
    auditing.expects(once())  
        .method("bought").with(same(TICKET));  
  
    agent.onPriceChange(THRESHOLD);  
}
```

general-purpose languages are not
designed for writing coherent sentences
in a domain-specific language

example 4: failed effort for specific combinations 7

```
class Foo {  
  pointcut foo() : call(boolean *.if*());  
}
```

example 4: failed effort for specific combinations 7

```
class Foo {  
    pointcut foo() : call(boolean *.if*());  
}
```

```
$ajc Foo.java
```

```
Foo.java:2 [error] Syntax error on token "if", invalid  
allowable token in pointcut or type pattern
```

example 4: failed effort for specific combinations 7

```
class Foo {  
    pointcut foo() : call(boolean *.if*());  
}
```

\$ajc Foo.java

Foo.java:2 [error] Syntax error on token "if", invalid
allowable token in pointcut or type pattern

```
class Foo {  
    pointcut foo() : call(void *0.Ef());  
}
```

example 4: failed effort for specific combinations 7

```
class Foo {  
    pointcut foo() : call(boolean *.if*());  
}
```

```
$ajc Foo.java
```

```
Foo.java:2 [error] Syntax error on token "if", invalid  
allowable token in pointcut or type pattern
```

```
class Foo {  
    pointcut foo() : call(void *0.Ef());  
}
```

```
$ajc Foo.java
```

```
Foo.java:2 [error] Invalid float literal number  
pointcut foo() : call(void *0.Ef());  
                        ^
```

improved abstraction with embedded
domain-specific languages

java and sql

```
String q = "SELECT id FROM users "  
          + "WHERE name = '" + userName + "'"   
          + "AND password = '" + password + "'";  
if (executeQuery(q.toString()).size() == 0) ...
```

```
SQL q = <| SELECT id FROM users  
          WHERE name = ${userName}  
          AND password = ${password} |>;  
if (executeQuery(q.toString()).size() == 0) ...
```

php and shell commands

```
$command = "svn cat \"file name\" -r" . $rev;  
system($command);
```

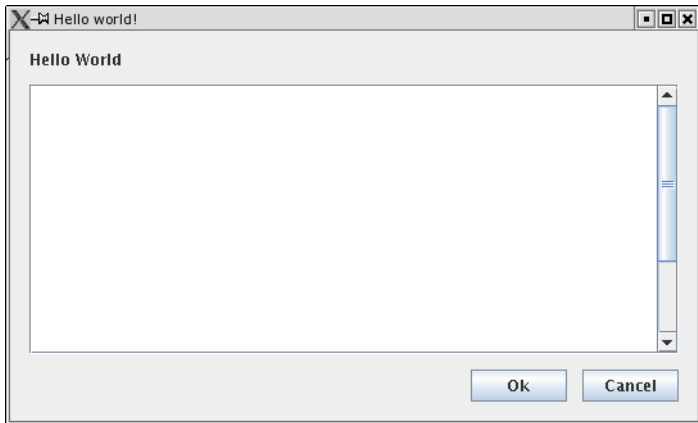
```
$command = "<| svn cat \"file name\" -r${$rev} |>";  
system($command->toString());
```

java and ldap

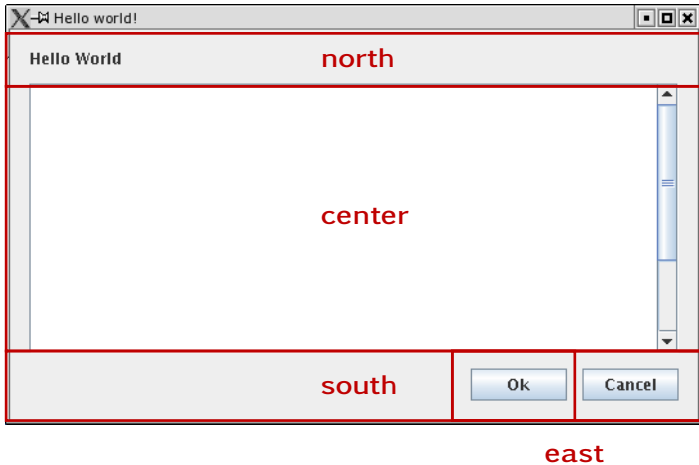
```
String q = "(cn=" + name + ")";
```

```
LDAP q = (| (cn=$(name)) |);
```

example 2: implement a graphical user-interface 10



example 2: implement a graphical user-interface 10



example 2: implement a graphical user-interface 11

```
public static void main(String[] ps) {  
    JPanel panel = panel of border layout {  
        north = label "Hello World"  
  
        center = scrollpane of textarea {  
            rows      = 20  
            columns = 40  
        }  
  
        south = panel of border layout {  
            east = panel of grid layout {  
                row = {  
                    button "Ok"  
                    button "Cancel"  
                }  
            }  
        }  
    }; ...  
}
```

example 2: implement a graphical user-interface 11

```
public static void main(String[] ps) {  
    JPanel panel = panel of border layout {  
        north = label "Hello World"  
  
        center = scrollpane of textarea {  
            rows      = 20  
            columns = 40  
        }  
  
        south = panel of border layout {  
            east = panel of grid layout {  
                row = {  
                    button "Ok"  
                    button "Cancel"  
                }  
            }  
        }  
    }; ...  
}
```

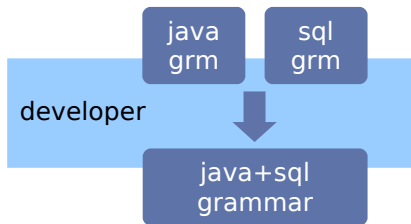
Syntax reflects the hierarchical structure of the user-interface

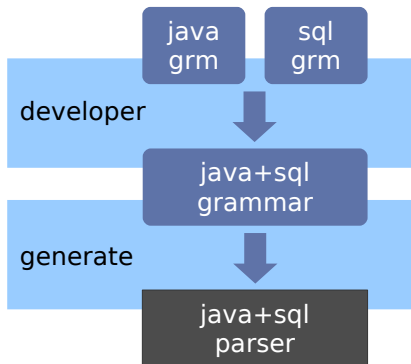
The interaction between the domain-specific and general-purpose code is seamless

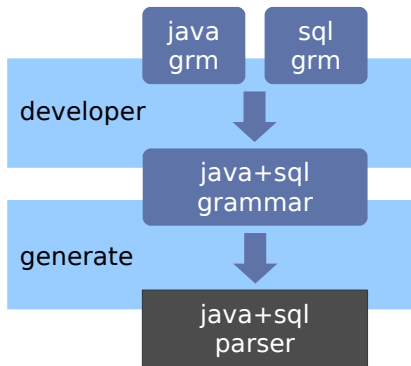
we need to fundamentally change the
way we develop grammars

java
gram

sql
gram

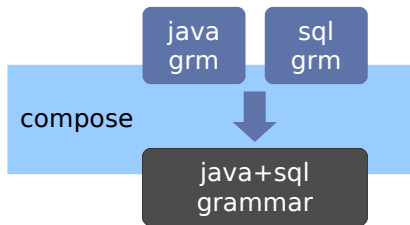
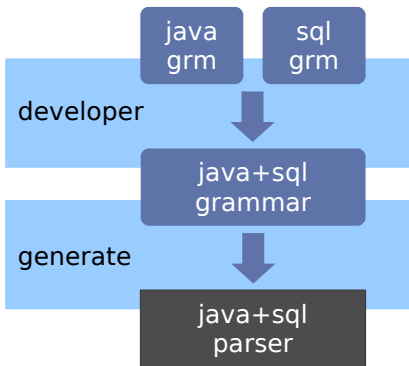


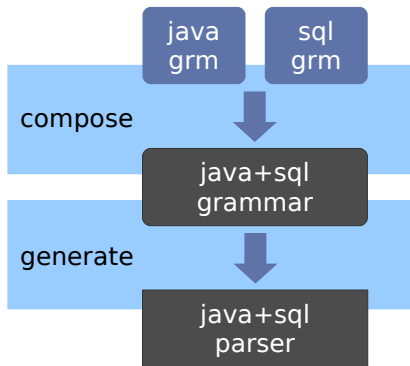
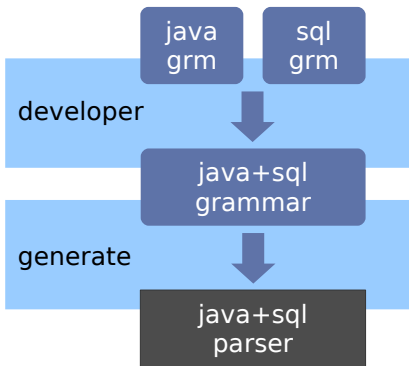


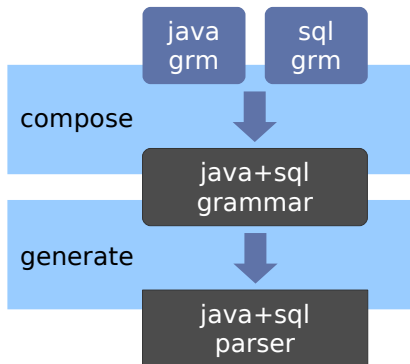
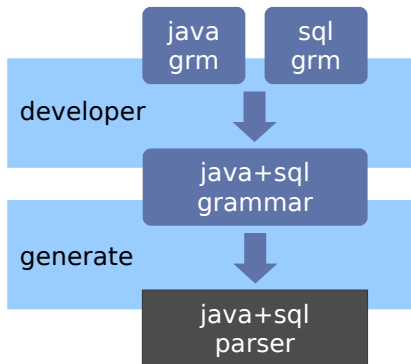


scanners do not compose
→ complex stateful
scanners and/or bugs

lr grammars do not compose
→ rewrite grammar to
resolve conflicts



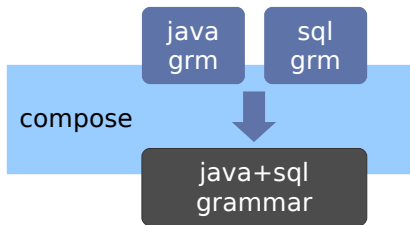


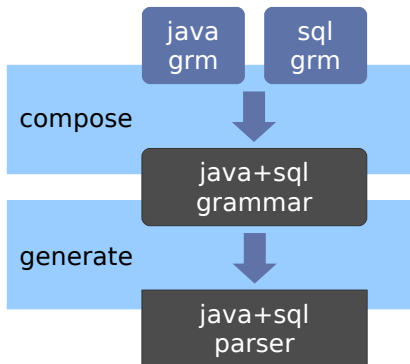


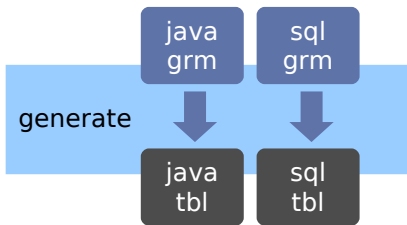
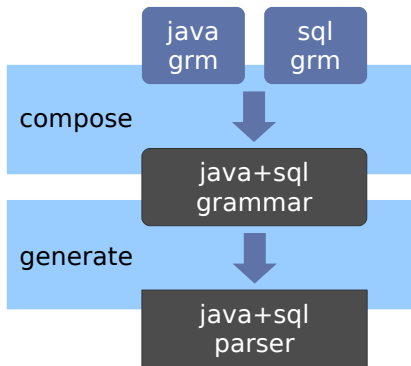
- scanners do not compose → scannerless parser
- lr grammars do not compose → cfg, generalized lr

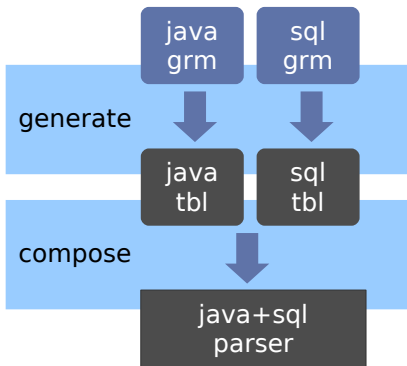
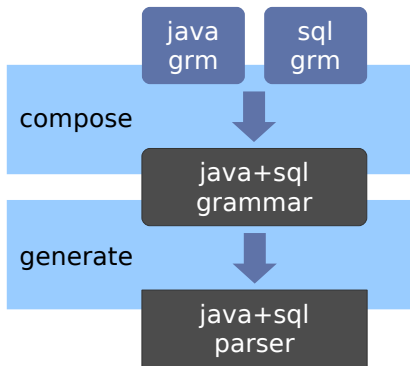
java
gram

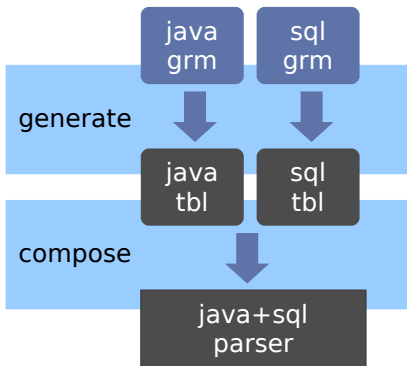
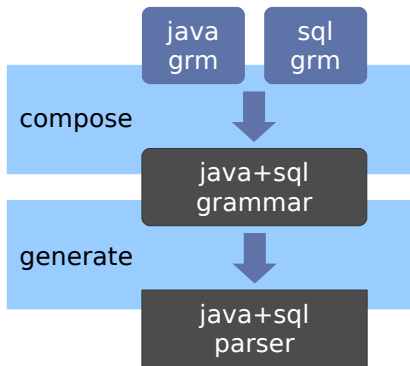
sql
gram











- from source-level extensibility to binary extensibility
- algorithm: minimize DFA reconstruction

extensible and domain-specific languages

program transformation

program analysis

extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

>100 citations

best paper

program transformation

program analysis

extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

preventing injection attacks [GPCE 2007, SCP 2009]

program transformation

program analysis

extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

preventing injection attacks [GPCE 2007, SCP 2009]

parse table composition [SLE 2008]

program transformation

program analysis

extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

preventing injection attacks [GPCE 2007, SCP 2009]

parse table composition [SLE 2008]

program transformation

stratego/xt transformation system [PEPM 2006, FI 2006, SCP 2008]

program analysis

extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

preventing injection attacks [GPCE 2007, SCP 2009]

parse table composition [SLE 2008]

program transformation

stratego/xt transformation system [PEPM 2006, FI 2006, SCP 2008]

grammar engineering [LDTA 2007]

program analysis

extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

preventing injection attacks [GPCE 2007, SCP 2009]

parse table composition [SLE 2008]

program transformation

stratego/xt transformation system [PEPM 2006, FI 2006, SCP 2008]

grammar engineering [LDTA 2007]

program analysis

scalable declarative pointer analysis [ISSTA 2009]

latest and greatest work, rest of the talk

what do we do?

fully declarative pointer analysis

fast, really fast

what do we do?

- fully declarative pointer analysis
- fast, really fast

how do we do it?

- novel, aggressive optimization
- exposition of indexes

what do we do?

- fully declarative pointer analysis
- fast, really fast

how do we do it?

- novel, aggressive optimization
- exposition of indexes

why do you care?

- fast
- sophisticated, simple
- different

what do we do?

- fully declarative pointer analysis
- fast, really fast

how do we do it?

- novel, aggressive optimization
- exposition of indexes

why do you care?

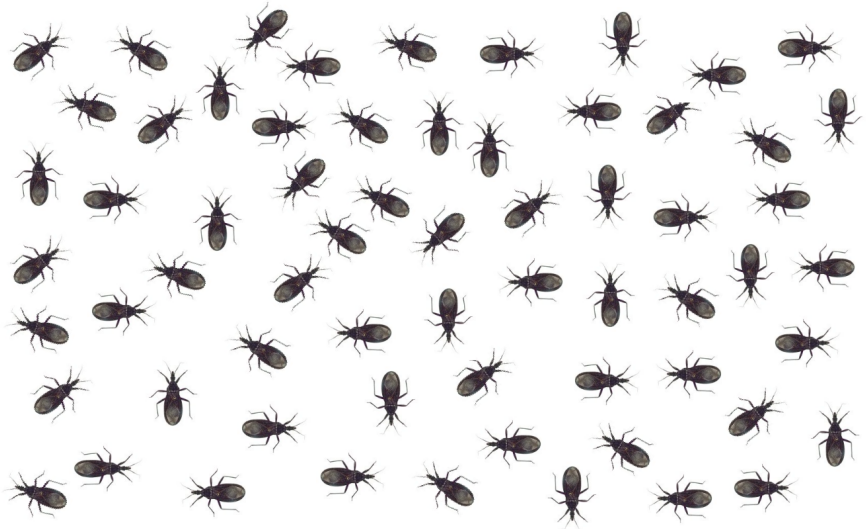
- fast
- sophisticated, simple
- different

why is it relevant?

- optimization
- understanding, find bugs







what objects can a variable point to?

program

```
void foo() {  
    a = new A1();  
    b = id(a);  
}
```

```
void bar() {  
    a = new A2();  
    b = id(a);  
}
```

```
A id(A a) {  
    return a;  
}
```


what objects can a variable point to?

program

```
void foo() {  
    a = new A1();  
    b = id(a);  
}
```

```
void bar() {  
    a = new A2();  
    b = id(a);  
}
```

```
A id(A a) {  
    return a;  
}
```

points-to

foo:a		new A1()
bar:a		new A2()

objects represented
by allocation sites

what objects can a variable point to?

program

- ```
void foo() {
 a = new A1();
 b = id(a);
}
```

- ```
void bar() {  
    a = new A2();  
    b = id(a);  
}
```

- ```
A id(A a) {
 return a;
}
```

## points-to

|       |  |                    |
|-------|--|--------------------|
| foo:a |  | new A1()           |
| bar:a |  | new A2()           |
| id:a  |  | new A1(), new A2() |

## what objects can a variable point to?

## program

```
void foo() {
 a = new A1();
 b = id(a);
}
```

```
void bar() {
 a = new A2();
 b = id(a);
}
```

```
A id(A a) {
 return a;
}
```

## points-to

|       |                    |
|-------|--------------------|
| foo:a | new A1()           |
| bar:a | new A2()           |
| id:a  | new A1(), new A2() |
| foo:b | new A1(), new A2() |
| bar:b | new A1(), new A2() |

## what objects can a variable point to?

## program

```
void foo() {
 a = new A1();
 b = id(a);
}
```

```
void bar() {
 a = new A2();
 b = id(a);
}
```

```
A id(A a) {
 return a;
}
```

## points-to

|       |                    |
|-------|--------------------|
| foo:a | new A1()           |
| bar:a | new A2()           |
| id:a  | new A1(), new A2() |
| foo:b | new A1(), new A2() |
| bar:b | new A1(), new A2() |

## context-sensitive points-to

|            |          |
|------------|----------|
| foo:a      | new A1() |
| bar:a      | new A2() |
| id:a (foo) | new A1() |
| id:a (bar) | new A2() |
| foo:b      | new A1() |
| bar:b      | new A2() |

## what objects can a variable point to?

### program

```
void foo() {
 a = new A1();
 b = id(a);
}
```

```
void bar() {
 a = new A2();
 b = id(a);
}
```

```
A id(A a) {
 return a;
}
```

### points-to

|       |                    |
|-------|--------------------|
| foo:a | new A1()           |
| bar:a | new A2()           |
| id:a  | new A1(), new A2() |
| foo:b | new A1(), new A2() |
| bar:b | new A1(), new A2() |

### context-sensitive points-to

|            |          |
|------------|----------|
| foo:a      | new A1() |
| bar:a      | new A2() |
| id:a (foo) | new A1() |
| id:a (bar) | new A2() |
| foo:b      | new A1() |
| bar:b      | new A2() |

necessarily an  
approximation

full precision  
too expensive

Results 1 - 20 of 2,343

 [Save results to a binder](#)

Sort by  in

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#) [>>](#)

## 1 [Semi-sparse flow-sensitive pointer analysis](#)

 January 2009 **POPL '09**: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages

**Publisher:** ACM

Full text available:  [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** alias analysis, pointer analysis

## 2 [Efficient field-sensitive pointer analysis of C](#)

 [David J. Pearce](#), [Paul H.J. Kelly](#), [Chris Hankin](#)  
November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available:  [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** Set-constraints, pointer analysis

## 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

 [John Whaley](#), [Monica S. Lam](#)  
June 2004 **PLDI '04**: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation

**Publisher:** ACM

Full text available:  [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

## flow-sensitive

Results 1 - 20 of 2,343

 [Save results to a binder](#)

Sort by  in

Result page: **1** 2 3 4 5 6 7 8 9 10 [next](#) [>>](#)

### 1 [Semi-sparse flow-sensitive pointer analysis](#)

 January 2009 **POPL '09**: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages

**Publisher:** ACM

Full text available:  [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** alias analysis, pointer analysis

### 2 [Efficient field-sensitive pointer analysis of C](#)

 [David J. Pearce](#), [Paul H.J. Kelly](#), [Chris Hankin](#)

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available:  [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** Set-constraints, pointer analysis

### 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

 [John Whaley](#), [Monica S. Lam](#)

June 2004 **PLDI '04**: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation

**Publisher:** ACM

Full text available:  [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

## flow-sensitive

Results 1 - 20 of 2,343

[Save results to a binder](#)

Sort by  in

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#) [>>](#)

### 1 [Semi-sparse flow-sensitive pointer analysis](#)

January 2009 **POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

**Publisher:** ACM

Full text available: [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** alias analysis, pointer analysis

### 2 [Efficient field-sensitive pointer analysis of C](#)

David J. Pearce, Paul H.J. Kelly, Chris Hankin

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available: [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** Set-constraints, pointer analysis

### 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

John Whaley, Monica S. Lam

June 2004 **PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**

**Publisher:** ACM

Full text available: [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

## field-sensitive



flow-sensitive

context-sensitive

field-sensitive

Results 1 - 20 of 2,343

 [Save results to a binder](#)

Sort by  in

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#) [>>](#)

## 1 [Semi-sparse flow-sensitive pointer analysis](#)

 January 2009 **POPL '09**: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages

**Publisher:** ACM

Full text available:  [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** [alias analysis](#), [pointer analysis](#)

## 2 [Efficient field-sensitive pointer analysis of C](#)

 [David J. Pearce](#), [Paul H.J. Kelly](#), [Chris Hankin](#)

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available:  [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** [Set constraints](#), [pointer analysis](#)

## 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

 [John Whaley](#), [Monica S. Lam](#)

June 2004 **PLDI '04**: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation

**Publisher:** ACM

Full text available:  [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

flow-sensitive

context-sensitive

field-sensitive

binary decision diagrams

Results 1 - 20 of 2,343

[Save results to a binder](#)

Sort by  in

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#) [>>](#)

## 1 [Semi-sparse flow-sensitive pointer analysis](#)

January 2009 **POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

**Publisher:** ACM

Full text available: [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** [alias analysis](#), [pointer analysis](#)

## 2 [Efficient field-sensitive pointer analysis of C](#)

David J. Pearce, Paul H.J. Kelly, Chris Hankin

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available: [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** [Set constraints](#), [pointer analysis](#)

## 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

John Whaley, Monica S. Lam

June 2004 **PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**

**Publisher:** ACM

Full text available: [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

inclusion-based

unification-based

flow-sensitive

on-the-fly call graph

context-sensitive

k-cfa

object sensitive

field-based

field-sensitive

heap cloning

binary decision diagrams

demand-driven

Results 1 - 20 of 2,343

[Save results to a binder](#)

Sort by  in

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#) [>>](#)

## 1 [Semi-sparse flow-sensitive pointer analysis](#)

January 2009 **POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

**Publisher:** ACM

Full text available: [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** [alias analysis](#), [pointer analysis](#)

## 2 [Efficient field-sensitive pointer analysis of C](#)

David J. Pearce, Paul H.J. Kelly, Chris Hankin

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available: [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** [Set-constraints](#), [pointer analysis](#)

## 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

John Whaley, Monica S. Lam

June 2004 **PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**

**Publisher:** ACM

Full text available: [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

```

procedure exhaustive_aliasing(G)
 G: an interprocedural control flow graph (ICFG);
begin
 /* 1. only performed implicitly */
 1. initialize may_hold with a default value NO;
 create an empty worklist;
 2. for each node N in G
 2.1 if N is a pointer assignment
 aliases_intro_by_assignment(N, YES);
 2.2 else if N is a call node
 aliases_intro_by_call(N, YES);
 3. while worklist is not empty
 3.1 remove (N, AA, PA) from worklist;
 3.2 if N is a call node
 alias_at_call_implies(N, AA, PA, YES);
 3.3 else if N is an exit node
 alias_at_exit_implies(N, AA, PA, YES);
 3.4 else for each M ∈ successor(N)
 3.4.1 if M is a pointer assignment
 alias_implies_thru_assign(M,
 AA, PA, YES);
 3.4.2 else
 make_true(M, AA, PA);
end

```

Figure 1: Exhaustive algorithm for *pointer aliasing*

```

procedure exhaustive_aliasing(G)
 G:
 begin
 1. procedure incremental_aliasing(G,N)
 G: an ICFG;
 N: a statement to be changed;
 begin
 1. falsify the affected aliases, which are either generated
 2. at N, or depend on other affected aliases.
 2. update G to reflect the change to statement N;
 3. worklist = reintroduce_aliases(G);
 4. reiterate_worklist(worklist, YES);
 end
 3. alias_at_call_implies(N, AA, PA, YES);
 3.3 else if N is an exit node
 alias_at_exit_implies(N, AA, PA, YES);
 3.4 else for each M ∈ successor(N)
 3.4.1 if M is a pointer assignment
 alias_implies_thru_assign(M,
 AA, PA, YES);
 3.4.2 else
 make_true(M, AA, PA);
 end

```

Figure 1: Exhaustive algorithm for *pointer aliasing*

Figure 2: Incremental aliasing algorithm for handling addition/deletion of a statement

```

procedure exhaustive_aliasing(G)
 G: a
 begin
 procedure incremental_aliasing(G.N)
 G: a /* Alias falsification corresponding to step 1 in Figure 2 */
 N: a
 begin
 procedure naive_falsification(N)
 N: a statement to be changed;
 begin
 1. fa
 2. at
 3. u
 4. re
 1. if N is marked TOUCHED, return;
 /* Falsify aliases at the changed node N */
 2. set all may_hold(N, AA, PA) to NO;
 3. mark N TOUCHED;
 4. if N is an exit node
 for each call node C which calls the function
 containing N;
 naive_falsification(corresponding return of C);
 5. else if N is a call node
 5.1 disable_aliases(entry of the function called by N);
 5.2 naive_falsification(corresponding return of N);
 6. else for each M ∈ successor(N)
 naive_falsification(M);
 end
 end
 procedure disable_aliases(E)
 E: entry of the function whose aliases will be disabled;
 begin
 1. if E is marked INFLUENCED, return;
 2. set all may_hold(E, AA, AA) to FALSIFIED;
 3. mark E INFLUENCED;
 4. for each call node C in function E;
 disable_aliases(entry of the function called by C);
 end
 end
 end
 end

```

Figure 1: Exhaustive

```

procedure exhaustive_aliasing(G)
 G: a program node in the ICFG;
begin
 1. procedure incremental_aliasing(G, N)
 G: a program node in the ICFG;
 N: a program node in the ICFG;
 begin
 1. procedure reintroduce_aliases(G)
 G: an ICFG;
 begin
 1. if G is a leaf node then
 2. return a worklist for keeping the reintroduced aliases;
 2. else
 3. create an empty worklist;
 4. for each call node C in G
 2.1 if C is TOUCHED or its called function is INFLUENCED,
 2.1.1 aliases_intro_by_call(C, YES);
 2.1.2 repropagate_aliases(C, worklist);
 2.2 repropagate_aliases(C, worklist);
 3. for each TOUCHED node N in G
 3.1 if N is a pointer assignment statement,
 aliases_intro_by_assignment(N, YES);
 3.2 for each M ∈ predecessor(N)
 repropagate_aliases(M, worklist);
 4. return worklist;
 2. end
 2. end
 3. end
 2. end
 3. end
 4. end
end

```

Figure 1: Exhaustive aliasing algorithm

```

procedure exhaustive_aliasing(G)
 G:
 begin
 procedure incremental_aliasing(G.N)
 G: a /* Alias falsification corresponding to step 1 in Figure 2 */
 N: a
 1. begin procedure /* Alias reintroduction corresponding to step 3 in Figure 2 */
 1. fa proced /* Reiteration corresponding to step 4 in Figure 2 */
 2. at G: a procedure reiterate_worklist(worklist,value)
 3. w 1. if return
 2. u /* a wo
 3. w 2. se worklist: a worklist for keeping the aliases to process;
 4. re 3. m value: value that will be given to (N,AA,PA);
 3. end 4. if begin begin
 1. cr 1. while worklist is not empty do
 2. fe 1.1 remove (N,AA,PA) from worklist;
 2. 1.2 if N is a call node
 aliases_propagated_at_call(N, AA, PA,
 value);
 3. 1.3 else if N is an exit node
 alias_at_exit_implies(N, AA, PA, value);
 3. 1.4 else for each M ∈ successor(N)
 1.4.1 if M is a pointer assignment
 alias_implies_thru_assign(M,
 AA, PA, value);
 1.4.2 else if value is YES
 make_true(M, AA, PA);
 1.4.3 else /* value is FALSIFIED */
 make_false(M, AA, PA);
 3. 3. fe
 3. 3.
 3. end
 end
 proced
 E: e
 begin
 1. if
 2. se
 3. m
 4. fe
 end
 end
 end

```

Figure 1: Exhaustive aliasing algorithm

```

Figure 5: Reiteration for the incremental algorithm
worklist: a worklist for keeping the reintroduced aliases;
begin
 for each may_hold(N, AA, PA) = YES
 add (N, AA, PA) to worklist;
end

```



```

procedure exhaustive_aliasing(G)
 G:
 begin
 procedure incremental_aliasing(G.N)
 G: a /* Alias falsification corresponding to step 1 in Figure 2 */
 N: a
 1. begin procedure /* Alias reintroduction corresponding to step 3 in Figure 2 */
 1. begin proced /* Reiteration corresponding to step 4 in Figure 2 */
 1. fa begin proced aliases.propagated.at.call(N, AA, PA, value)
 1. N: a call node;
 2. AA: reaching alias at the entry of the function contain-
 ing N;
 3. PA: possible alias at N;
 4. value: value to set the propagated aliases;
 begin
 1. let E be the entry of the function called by N, and
 R the corresponding return node of N;
 /* aliasing effect propagated to the entry node E */
 2. for each AA' in bind(N, E, PA)
 /* bind uses parameter bindings to map PA to the
 entry E of the called function */
 2.1 if (E, AA', AA') has not been seen before
 make.true(E, AA', AA');
 2.2 else if may.hold(E, AA', AA') ≠ value
 2.2.1 set may.hold(E, AA', AA') to value;
 /* Recursively enable (or disable) all the
 reaching aliases implied at the entry of
 other functions reachable from E */
 2.2.2 inter.proc.propagate(E, AA', value);
 /* aliasing effect propagated to the return node R */
 3. (Same as what is done for propagating aliases to the
 return node in procedure aliases.at.call.implies, except
 it will make.true or make.false the implied aliases,
 depending on what value is)
 end
 end
 end
 end
 E: e
 begin
 1. if
 2. se
 3. m
 4. fo
 end
 proced
 N: a
 worklist: a w
 begin
 for each may
 add (N, A
 end
 end

```

Figure 1: Exh

Figure 2

Figure 3

Figure 4

Figure 5

Figure 6

Figure 7

Figure 8

Figure 9

Figure 10

Figure 11

Figure 12

Figure 13

Figure 14

Figure 15

Figure 16

Figure 17

Figure 18

Figure 19

Figure 20

Figure 21

Figure 22

Figure 23

Figure 24

Figure 25

Figure 26

Figure 27

Figure 28

Figure 29

Figure 30

Figure 31

Figure 32

Figure 33

Figure 34

Figure 35

Figure 36

Figure 37

Figure 38

Figure 39

Figure 40

Figure 41

Figure 42

Figure 43

Figure 44

Figure 45

Figure 46

Figure 47

Figure 48

Figure 49

Figure 50

Figure 51

Figure 52

Figure 53

Figure 54

Figure 55

Figure 56

Figure 57

Figure 58

Figure 59

Figure 60

Figure 61

Figure 62

Figure 63

Figure 64

Figure 65

Figure 66

Figure 67

Figure 68

Figure 69

Figure 70

Figure 71

Figure 72

Figure 73

Figure 74

Figure 75

Figure 76

Figure 77

Figure 78

Figure 79

Figure 80

Figure 81

Figure 82

Figure 83

Figure 84

Figure 85

Figure 86

Figure 87

Figure 88

Figure 89

Figure 90

Figure 91

Figure 92

Figure 93

Figure 94

Figure 95

Figure 96

Figure 97

Figure 98

Figure 99

Figure 100

```

procedure exhaustive_aliasing(G)
 G: a
 begin
 procedure incremental_aliasing(G, N)
 G: a
 N: a
 1. begin
 1. fa
 2. u
 3. w
 4. re
 2. end
 3. Figure
 4. addition
 5. e
 6. e
 7. end
 8. proced
 9. E: e
 10. begin
 1. if
 2. se
 3. m
 4. fe
 11. end
 12. worklist: a w
 13. begin
 1. for each may
 2. add (N, A
 14. end
 1. return wor
 2. end
 3. proced
 4. G: a
 5. proced
 6. N: a call node
 7. AA: read
 8. ing
 9. PA: pos
 10. value: v
 11. begin
 1. let E
 2. R the
 3. /* all
 4. for e
 5. /* bit
 6. en
 7. 2.1
 8. alias.implies.thru.assign(N, AA, PA,
 9. FALSIFIED);
 10. 4. reiterate.worklist(worklist, FALSIFIED);
 11. end
 12. procedure falsify_for_deleting_call(N)
 13. N: a function call to be deleted;
 14. begin
 1. create an empty worklist;
 2. /* Falsify the aliases introduced by the call */
 3. let E and X be the corresponding entry node and
 4. exit node of the function called by N respectively;
 5. aliases.propagated.at.call(N, $\emptyset$, $\emptyset$, FALSIFIED);
 6. for each may.hold(N, AA, PA) = YES
 7. /* If the called function may generate new aliases
 8. from the reaching aliases implied by PA */
 9. if $\exists$ AA' $\in$ bind(N, E, PA), such that some
 10. PA' ($\neq$ AA') is generated from AA' at exit X
 11. aliases.propagated.at.call(N, AA, PA,
 12. FALSIFIED);
 13. 5. if a function becomes unreachable from the main pro-
 14. gram after the call node is deleted, steps 3 and 4
 15. are repeated on those calls within each of the
 16. reachable functions.
 17. 6. reiterate.worklist(worklist, FALSIFIED);
 18. end
 19. end
 20. end
 21. end
 22. end
 23. end
 24. end
 25. end
 26. end
 27. end
 28. end
 29. end
 30. end
 31. end
 32. end
 33. end
 34. end
 35. end
 36. end
 37. end
 38. end
 39. end
 40. end
 41. end
 42. end
 43. end
 44. end
 45. end
 46. end
 47. end
 48. end
 49. end
 50. end
 51. end
 52. end
 53. end
 54. end
 55. end
 56. end
 57. end
 58. end
 59. end
 60. end
 61. end
 62. end
 63. end
 64. end
 65. end
 66. end
 67. end
 68. end
 69. end
 70. end
 71. end
 72. end
 73. end
 74. end
 75. end
 76. end
 77. end
 78. end
 79. end
 80. end
 81. end
 82. end
 83. end
 84. end
 85. end
 86. end
 87. end
 88. end
 89. end
 90. end
 91. end
 92. end
 93. end
 94. end
 95. end
 96. end
 97. end
 98. end
 99. end
 100. end
 101. end
 102. end
 103. end
 104. end
 105. end
 106. end
 107. end
 108. end
 109. end
 110. end
 111. end
 112. end
 113. end
 114. end
 115. end
 116. end
 117. end
 118. end
 119. end
 120. end
 121. end
 122. end
 123. end
 124. end
 125. end
 126. end
 127. end
 128. end
 129. end
 130. end
 131. end
 132. end
 133. end
 134. end
 135. end
 136. end
 137. end
 138. end
 139. end
 140. end
 141. end
 142. end
 143. end
 144. end
 145. end
 146. end
 147. end
 148. end
 149. end
 150. end
 151. end
 152. end
 153. end
 154. end
 155. end
 156. end
 157. end
 158. end
 159. end
 160. end
 161. end
 162. end
 163. end
 164. end
 165. end
 166. end
 167. end
 168. end
 169. end
 170. end
 171. end
 172. end
 173. end
 174. end
 175. end
 176. end
 177. end
 178. end
 179. end
 180. end
 181. end
 182. end
 183. end
 184. end
 185. end
 186. end
 187. end
 188. end
 189. end
 190. end
 191. end
 192. end
 193. end
 194. end
 195. end
 196. end
 197. end
 198. end
 199. end
 200. end
 201. end
 202. end
 203. end
 204. end
 205. end
 206. end
 207. end
 208. end
 209. end
 210. end
 211. end
 212. end
 213. end
 214. end
 215. end
 216. end
 217. end
 218. end
 219. end
 220. end
 221. end
 222. end
 223. end
 224. end
 225. end
 226. end
 227. end
 228. end
 229. end
 230. end
 231. end
 232. end
 233. end
 234. end
 235. end
 236. end
 237. end
 238. end
 239. end
 240. end
 241. end
 242. end
 243. end
 244. end
 245. end
 246. end
 247. end
 248. end
 249. end
 250. end
 251. end
 252. end
 253. end
 254. end
 255. end
 256. end
 257. end
 258. end
 259. end
 260. end
 261. end
 262. end
 263. end
 264. end
 265. end
 266. end
 267. end
 268. end
 269. end
 270. end
 271. end
 272. end
 273. end
 274. end
 275. end
 276. end
 277. end
 278. end
 279. end
 280. end
 281. end
 282. end
 283. end
 284. end
 285. end
 286. end
 287. end
 288. end
 289. end
 290. end
 291. end
 292. end
 293. end
 294. end
 295. end
 296. end
 297. end
 298. end
 299. end
 300. end
 301. end
 302. end
 303. end
 304. end
 305. end
 306. end
 307. end
 308. end
 309. end
 310. end
 311. end
 312. end
 313. end
 314. end
 315. end
 316. end
 317. end
 318. end
 319. end
 320. end
 321. end
 322. end
 323. end
 324. end
 325. end
 326. end
 327. end
 328. end
 329. end
 330. end
 331. end
 332. end
 333. end
 334. end
 335. end
 336. end
 337. end
 338. end
 339. end
 340. end
 341. end
 342. end
 343. end
 344. end
 345. end
 346. end
 347. end
 348. end
 349. end
 350. end
 351. end
 352. end
 353. end
 354. end
 355. end
 356. end
 357. end
 358. end
 359. end
 360. end
 361. end
 362. end
 363. end
 364. end
 365. end
 366. end
 367. end
 368. end
 369. end
 370. end
 371. end
 372. end
 373. end
 374. end
 375. end
 376. end
 377. end
 378. end
 379. end
 380. end
 381. end
 382. end
 383. end
 384. end
 385. end
 386. end
 387. end
 388. end
 389. end
 390. end
 391. end
 392. end
 393. end
 394. end
 395. end
 396. end
 397. end
 398. end
 399. end
 400. end
 401. end
 402. end
 403. end
 404. end
 405. end
 406. end
 407. end
 408. end
 409. end
 410. end
 411. end
 412. end
 413. end
 414. end
 415. end
 416. end
 417. end
 418. end
 419. end
 420. end
 421. end
 422. end
 423. end
 424. end
 425. end
 426. end
 427. end
 428. end
 429. end
 430. end
 431. end
 432. end
 433. end
 434. end
 435. end
 436. end
 437. end
 438. end
 439. end
 440. end
 441. end
 442. end
 443. end
 444. end
 445. end
 446. end
 447. end
 448. end
 449. end
 450. end
 451. end
 452. end
 453. end
 454. end
 455. end
 456. end
 457. end
 458. end
 459. end
 460. end
 461. end
 462. end
 463. end
 464. end
 465. end
 466. end
 467. end
 468. end
 469. end
 470. end
 471. end
 472. end
 473. end
 474. end
 475. end
 476. end
 477. end
 478. end
 479. end
 480. end
 481. end
 482. end
 483. end
 484. end
 485. end
 486. end
 487. end
 488. end
 489. end
 490. end
 491. end
 492. end
 493. end
 494. end
 495. end
 496. end
 497. end
 498. end
 499. end
 500. end
 501. end
 502. end
 503. end
 504. end
 505. end
 506. end
 507. end
 508. end
 509. end
 510. end
 511. end
 512. end
 513. end
 514. end
 515. end
 516. end
 517. end
 518. end
 519. end
 520. end
 521. end
 522. end
 523. end
 524. end
 525. end
 526. end
 527. end
 528. end
 529. end
 530. end
 531. end
 532. end
 533. end
 534. end
 535. end
 536. end
 537. end
 538. end
 539. end
 540. end
 541. end
 542. end
 543. end
 544. end
 545. end
 546. end
 547. end
 548. end
 549. end
 550. end
 551. end
 552. end
 553. end
 554. end
 555. end
 556. end
 557. end
 558. end
 559. end
 560. end
 561. end
 562. end
 563. end
 564. end
 565. end
 566. end
 567. end
 568. end
 569. end
 570. end
 571. end
 572. end
 573. end
 574. end
 575. end
 576. end
 577. end
 578. end
 579. end
 580. end
 581. end
 582. end
 583. end
 584. end
 585. end
 586. end
 587. end
 588. end
 589. end
 590. end
 591. end
 592. end
 593. end
 594. end
 595. end
 596. end
 597. end
 598. end
 599. end
 600. end
 601. end
 602. end
 603. end
 604. end
 605. end
 606. end
 607. end
 608. end
 609. end
 610. end
 611. end
 612. end
 613. end
 614. end
 615. end
 616. end
 617. end
 618. end
 619. end
 620. end
 621. end
 622. end
 623. end
 624. end
 625. end
 626. end
 627. end
 628. end
 629. end
 630. end
 631. end
 632. end
 633. end
 634. end
 635. end
 636. end
 637. end
 638. end
 639. end
 640. end
 641. end
 642. end
 643. end
 644. end
 645. end
 646. end
 647. end
 648. end
 649. end
 650. end
 651. end
 652. end
 653. end
 654. end
 655. end
 656. end
 657. end
 658. end
 659. end
 660. end
 661. end
 662. end
 663. end
 664. end
 665. end
 666. end
 667. end
 668. end
 669. end
 670. end
 671. end
 672. end
 673. end
 674. end
 675. end
 676. end
 677. end
 678. end
 679. end
 680. end
 681. end
 682. end
 683. end
 684. end
 685. end
 686. end
 687. end
 688. end
 689. end
 690. end
 691. end
 692. end
 693. end
 694. end
 695. end
 696. end
 697. end
 698. end
 699. end
 700. end
 701. end
 702. end
 703. end
 704. end
 705. end
 706. end
 707. end
 708. end
 709. end
 710. end
 711. end
 712. end
 713. end
 714. end
 715. end
 716. end
 717. end
 718. end
 719. end
 720. end
 721. end
 722. end
 723. end
 724. end
 725. end
 726. end
 727. end
 728. end
 729. end
 730. end
 731. end
 732. end
 733. end
 734. end
 735. end
 736. end
 737. end
 738. end
 739. end
 740. end
 741. end
 742. end
 743. end
 744. end
 745. end
 746. end
 747. end
 748. end
 749. end
 750. end
 751. end
 752. end
 753. end
 754. end
 755. end
 756. end
 757. end
 758. end
 759. end
 760. end
 761. end
 762. end
 763. end
 764. end
 765. end
 766. end
 767. end
 768. end
 769. end
 770. end
 771. end
 772. end
 773. end
 774. end
 775.
```

```

procedure exhaustive_aliasing(G)
 G:
 begin
 procedure incremental_aliasing(G, N)
 G: a /* Alias falsification corresponding to step 1 in Figure 2 */
 N: a
 1. begin procedure /* Alias reintroduction corresponding to step 3 in Figure 2 */
 1. fa proced /* Reiteration corresponding to step 4 in Figure 2 */
 2. at 1. begin procedure aliases_propagated_at_call(N, AA, PA, value)
 1. if N: a call node
 2. u /* Alias falsification for deleting a pointer assignment
 corresponding to step 2 in Figure 2 */
 3. w /* a wo valu
 4. re 2. se begin begin
 1. c 1. v
 2. fe 1.
 2. 2.
 3. end 4. if 1.
 Figure 1: Exhaustive aliasing
 addition
 3.3 else 5. e
 3.4 else 5. /
 6. e 3. fe
 end 3
 proced E: e
 begin 4. r
 1. if end
 2. se proced
 3. m N: a
 4. fe worklist: a w
 begin
 for each may
 add (N, A
 end
 end
 end
 end
 end

```

Figure 8: Procedure for falsifying aliases that are potentially affected by adding a pointer assignment

```

procedure update_for_adding_assign(N, M)
 N: a pointer assignment to be added;
 M: the statement after which statement N is added;
 begin
 1. make N as a successor of M, and leave N without
 any successors;
 2. create an empty worklist;
 3. aliases_intro_by_assignment(N, YES);
 4. repropagate_aliases(M, worklist);
 5. reiterate_worklist(worklist, YES);
 6. for each may_hold(M, AA, PA = (o1, o2)) = YES,
 and may_hold(N, AA, PA) = NO
 add (M, AA, PA) to worklist;
 7. reiterate_worklist(worklist, FALSIFIED);
 end

```

Figure 1: Exhaustive aliasing

Figure 8: Procedure for falsifying aliases that are potentially affected by adding a pointer assignment

Figure 9: Procedure for falsifying aliases that are potentially affected by adding a pointer assignment

variation points  
unclear

every variant new  
algorithm

correctness  
unclear

incomparable  
in precision

incomparable  
in performance

```

procedure falsify_aliases(N, AA, PA, value)
N: a call node;
AA: real aliasing information corresponding to step 1 in Figure 2 */
/* Alias reintroduction corresponding to step 3 in Figure 2 */
/* Reiteration corresponding to step 4 in Figure 2 */
begin
 procedure aliases_propagated_at_call(N, AA, PA, value)
 N: a pointer assignment to be added;
 M: the statement after which statement N is added;
 begin
 1. make N as a successor of M, and leave N without
 any successors;
 2. create an empty worklist;
 3. aliases_intro_by_assignment(N, YES);
 4. repropagate_aliases(M, worklist);
 5. reiterate_worklist(worklist, YES);
 6. for each may_hold(M, AA, PA = (o1, o2)) = YES,
 and may_hold(N, AA, PA) = NO
 add (M, AA, PA) to worklist;
 7. reiterate_worklist(worklist, FALSIFIED);
 end
 end
 procedure N_is_a_function(N)
 N: a function name;
 begin
 1. create an empty worklist;
 2. let E be the entry point of N;
 3. aliases_intro_by_assignment(E, YES);
 4. for each reachable function F of N
 if F ≠ AA
 if F is generated from AA at exit A
 aliases_propagated_at_call(N, AA, PA, FALSIFIED);
 5. if a function becomes unreachable from the main program
 after the call node is deleted, steps 3 and 4
 are repeated on those calls within each of the
 reachable functions.
 6. reiterate_worklist(worklist, FALSIFIED);
 end
 end
end

```

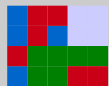
Figure 8: Procedure for falsifying aliases that are potentially affected by adding a pointer assignment

Figure 8: Procedure for falsifying aliases that are potentially affected by adding a pointer assignment

algorithm



algorithm

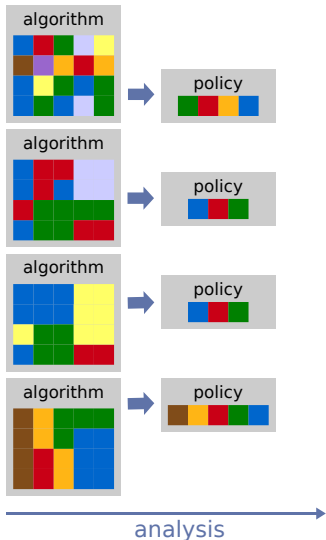


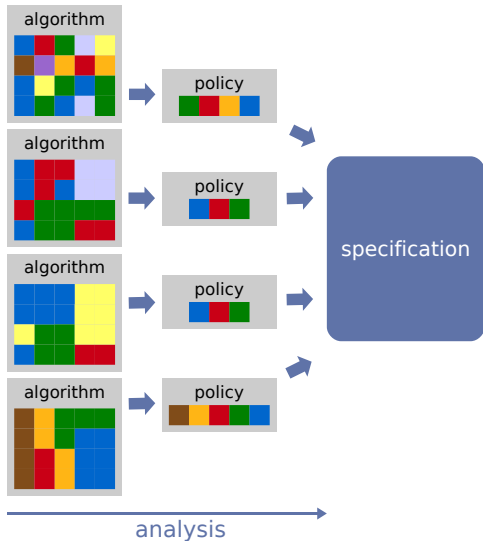
algorithm

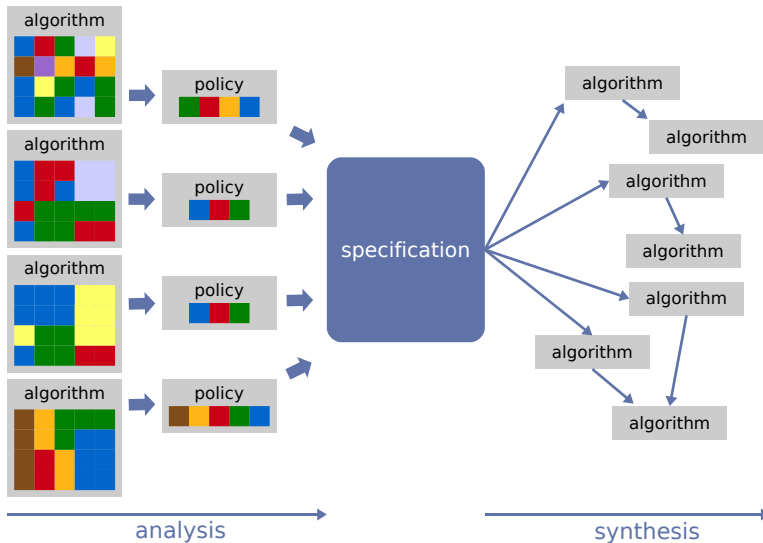


algorithm







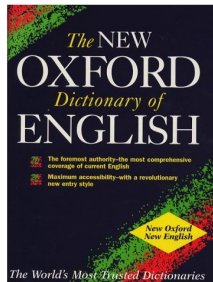




from algorithms to specification

*“denoting high-level programming languages which can be used to solve problems without requiring the programmer to specify an exact procedure to be followed.”*

- high-level
- what, not how
- no control-flow
- no side-effects
- specifications, not programs, not algorithms



- datalog-based pointer analysis framework for java

- datalog-based pointer analysis framework for java
- declarative: what, not how

- datalog-based pointer analysis framework for java
- declarative: what, not how
  - easier to express sophisticated analyses
  - correctness more clear
  - clear variation points
  - eases exploration of approximations
  - enables aggressive optimization

- datalog-based pointer analysis framework for java
- declarative: what, not how
  - easier to express sophisticated analyses
  - correctness more clear
  - clear variation points
  - eases exploration of approximations
  - enables aggressive optimization
- sophisticated
  - subset-based analysis, fully on-the-fly call graph discovery, field-sensitivity, context-sensitivity, call-site sensitive, object sensitive, thread sensitive, context-sensitive heap abstraction, type filtering, precise exception analysis

- datalog-based pointer analysis framework for java
- declarative: what, not how
  - easier to express sophisticated analyses
  - correctness more clear
  - clear variation points
  - eases exploration of approximations
  - enables aggressive optimization
- sophisticated
  - subset-based analysis, fully on-the-fly call graph discovery, field-sensitivity, context-sensitivity, call-site sensitive, object sensitive, thread sensitive, context-sensitive heap abstraction, type filtering, precise exception analysis
- support for full semantic complexity of java
  - jvm initialization, reflection analysis, threads, reference queues, native methods, class initialization, finalization, cast checking, assignment compatibility

- datalog-based pointer analysis framework for java
- declarative: what, not how
  - easier to express sophisticated analyses
  - correctness more clear
  - clear variation points
  - eases exploration of approximations
  - enables aggressive optimization
- sophisticated
  - subset-based analysis, fully on-the-fly call graph discovery, field-sensitivity, context-sensitivity, call-site sensitive, object sensitive, thread sensitive, context-sensitive heap abstraction, type filtering, precise exception analysis
- support for full semantic complexity of java
  - jvm initialization, reflection analysis, threads, reference queues, native methods, class initialization, finalization, cast checking, assignment compatibility
- enables precision and performance comparison



### expressed pointer analysis in full sophistication in datalog

- core specification: ~250 lines of logic
- full specification: ~2500 lines of logic

## expressed pointer analysis in full sophistication in datalog

- core specification: ~250 lines of logic
- full specification: ~2500 lines of logic

## synthesized efficient algorithms from specification

- order of magnitude performance improvement
- support all analyses considered interesting

## expressed pointer analysis in full sophistication in datalog

- core specification: ~250 lines of logic
- full specification: ~2500 lines of logic

## synthesized efficient algorithms from specification

- order of magnitude performance improvement
- support all analyses considered interesting

## approach: heuristics for searching algorithm space

- targeted at recursive problem domains

## expressed pointer analysis in full sophistication in datalog

- core specification: ~250 lines of logic
- full specification: ~2500 lines of logic

## synthesized efficient algorithms from specification

- order of magnitude performance improvement
- support all analyses considered interesting

## approach: heuristics for searching algorithm space

- targeted at recursive problem domains

## demonstrated scalability with explicit representation

from algorithms to specification

pointer analysis and datalog background

## program analysis: a domain of mutual recursion 26

var points-to

## program analysis: a domain of mutual recursion 26

$x = y$

var points-to

## program analysis: a domain of mutual recursion 26

$x = y$





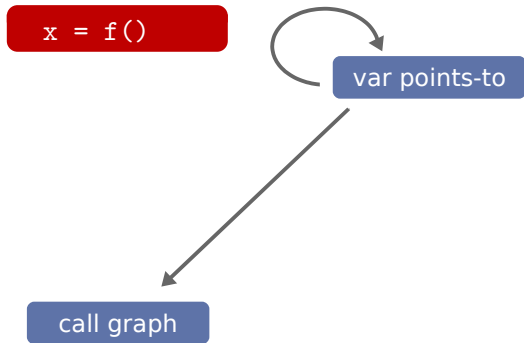
## program analysis: a domain of mutual recursion 26

`x = f()`



call graph

## program analysis: a domain of mutual recursion 26



## program analysis: a domain of mutual recursion 26

`x = y.f()`

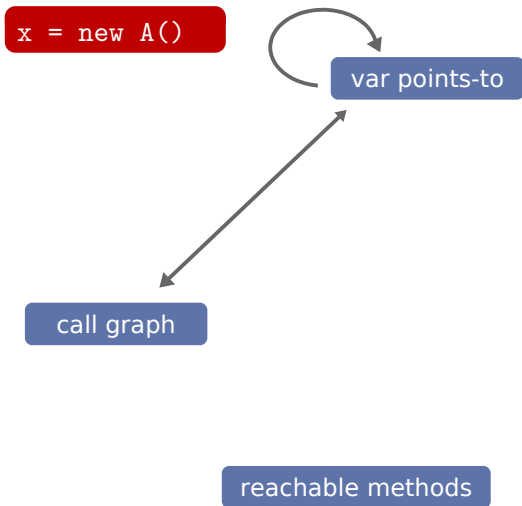
var points-to

```
graph TD; A["x = y.f()"] --> B["var points-to"]; B --> C["call graph"]; B --> B;
```

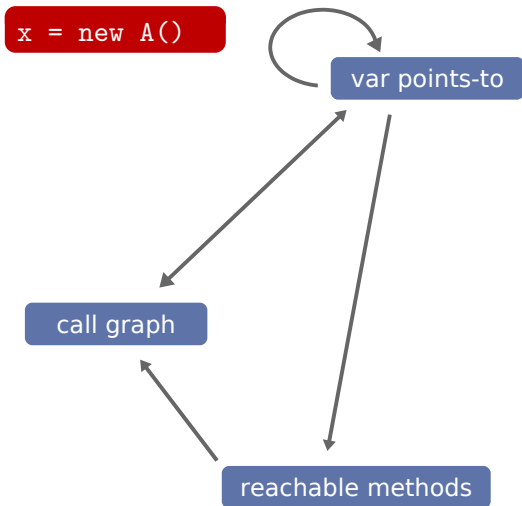
The diagram illustrates the components of a program analysis domain. A red box containing the code snippet `x = y.f()` is connected by a straight arrow to a blue box labeled `var points-to`. This blue box has a self-looping curved arrow and another straight arrow pointing to a second blue box labeled `call graph`.

call graph

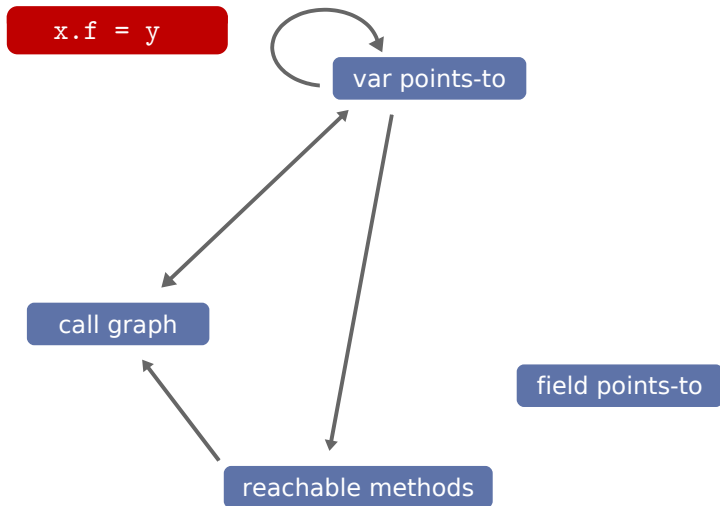
## program analysis: a domain of mutual recursion 26



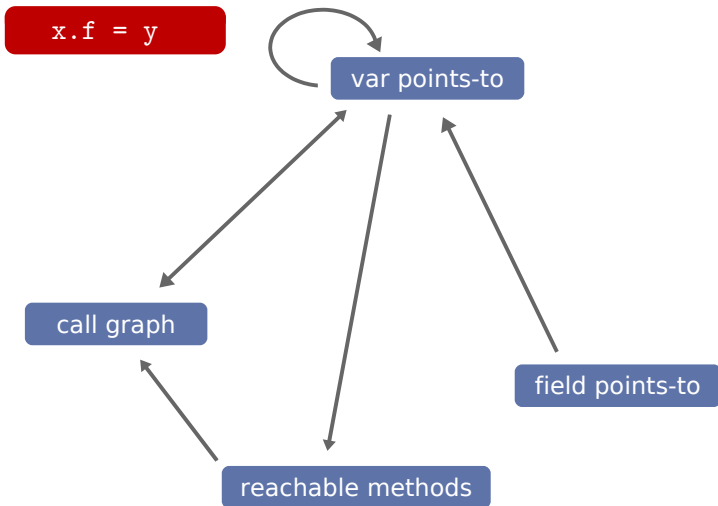
## program analysis: a domain of mutual recursion 26



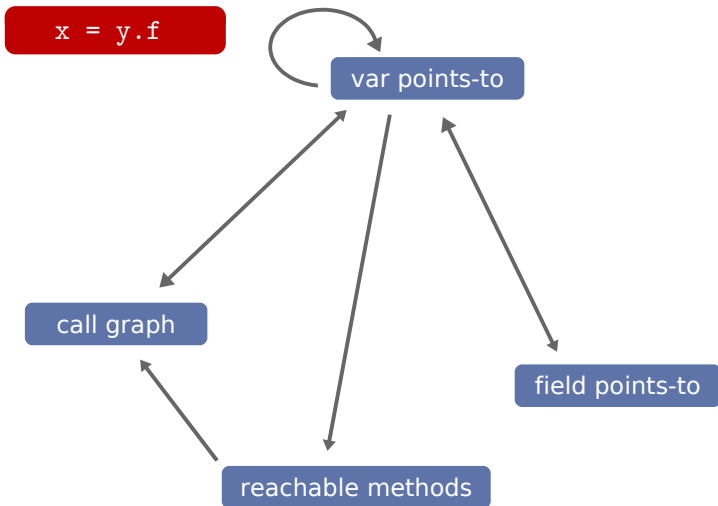
## program analysis: a domain of mutual recursion 26



## program analysis: a domain of mutual recursion 26

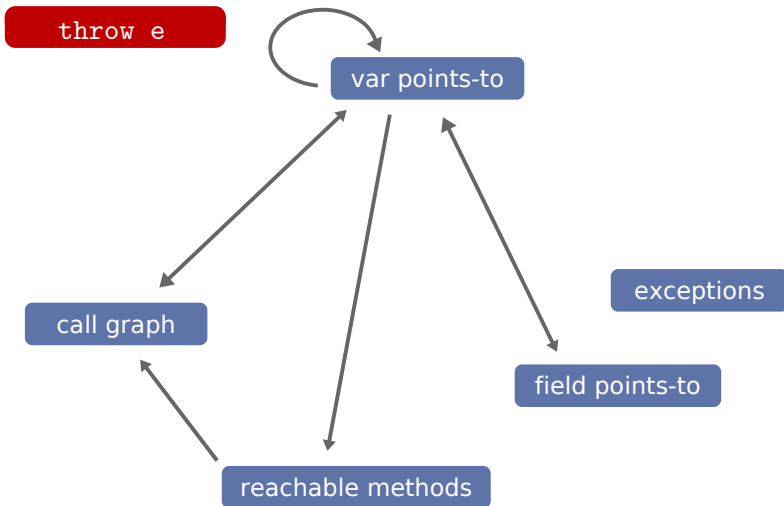


## program analysis: a domain of mutual recursion 26

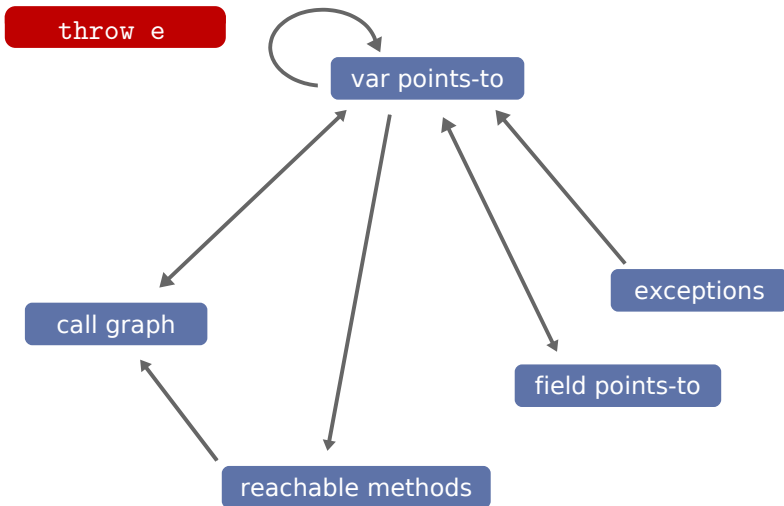




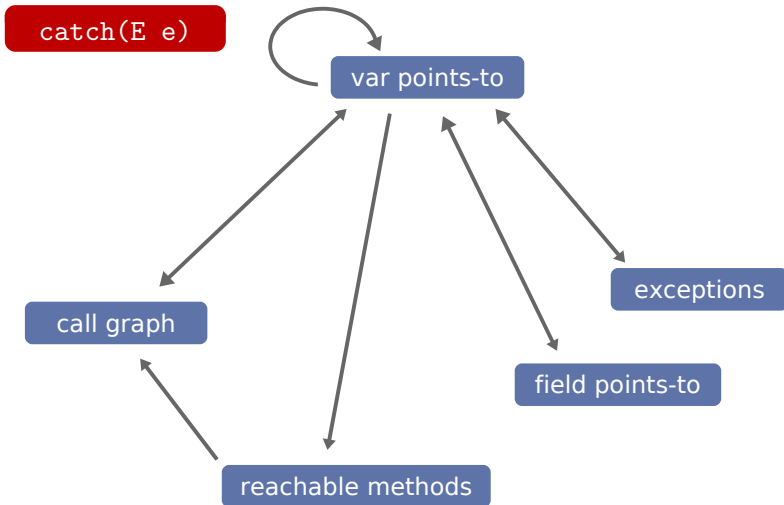
## program analysis: a domain of mutual recursion 26



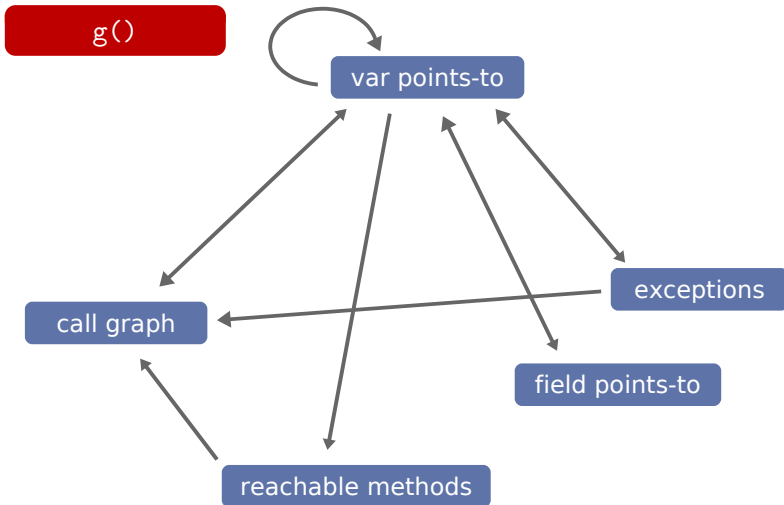
## program analysis: a domain of mutual recursion 26



## program analysis: a domain of mutual recursion 26

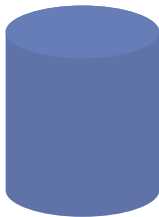


## program analysis: a domain of mutual recursion 26



source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```



## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```



## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```



## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |
| a |  | new B() |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## source

```
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```

## AssignObjectAllocation

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |

## Assign

|   |  |   |
|---|--|---|
| b |  | a |
| a |  | b |
| b |  | c |

## VarPointsTo

|   |  |         |
|---|--|---------|
| a |  | new A() |
| b |  | new B() |
| c |  | new C() |
| a |  | new B() |
| b |  | new A() |
| c |  | new B() |
| c |  | new A() |

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

## limited logic programming

- sql with recursion  
prolog without complex terms (constructors)
- captures PTIME complexity class

## strictly declarative

- as opposed to prolog
  - conjunction commutative
  - rules commutative
- increases algorithm space
  - enables different execution strategies
  - enables more aggressive optimization

writing datalog is less programming, more specification

from algorithms to specification

declarative pointer analysis specification

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
?base . ?field = ?from
```

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

- FieldPointsTo(?baseobj, ?field, ?obj) <-

```
?base . ?field = ?from
```

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

field ?field  
of object ?baseobj  
may point to object ?obj

- FieldPointsTo(?baseobj, ?field, ?obj) <-

?base . ?field = ?from



```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

field ?field  
of object ?baseobj  
may point to object ?obj

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
```

?base . ?field = ?from

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

field ?field  
of object ?baseobj  
may point to object ?obj

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

?base . ?field = ?from

↓  
?baseobj

↓  
?obj

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

`?to = ?base . ?field`

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

● VarPointsTo(?to, ?obj) <-

?to = ?base . ?field

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 LoadField(?base, ?field, ?to),
```

?to = ?base . ?field

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 LoadField(?base, ?field, ?to),
 VarPointsTo(?base, ?baseobj),
```

?baseobj



?to = ?base . ?field

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 LoadField(?base, ?field, ?to),
 VarPointsTo(?base, ?baseobj),
 FieldPointsTo(?baseobj, ?field, ?obj).
```

?baseobj



?to = ?base . ?field

```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?var, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to),
 VarPointsTo(?from, ?obj).
```

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?base, ?baseobj),
 VarPointsTo(?from, ?obj).
```

```
VarPointsTo(?to, ?obj) <-
 LoadField(?base, ?field, ?to),
 VarPointsTo(?base, ?baseobj),
 FieldPointsTo(?baseobj, ?field, ?obj).
```

modularity: introducing  
new mutually recursive  
dependencies does not  
change existing rules



### method invocations: propagated exceptions

```
void f() {
 g();
}
```

### method invocations: caught exceptions

```
void f() {
 try {
 g();
 } catch(E e) {...}
}
```

### method invocations: propagated exceptions

- `ThrowPointsTo(?caller, ?obj) <-`

```
void f() {
 g();
}
```

### method invocations: caught exceptions

- `VarPointsTo(?param, ?obj) <-`

```
void f() {
 try {
 g();
 } catch(E e) {...}
}
```

### method invocations: propagated exceptions

- ```
ThrowPointsTo(?caller, ?obj) <-  
  CallGraphEdge(?invocation, ?tomethod),
```

```
void f() {  
  g();  
}
```

method invocations: caught exceptions

```
VarPointsTo(?param, ?obj) <-
```

```
void f() {  
  try {  
    g();  
  } catch(E e) {...}  
}
```

method invocations: propagated exceptions

ThrowPointsTo(?caller, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),

```
void f() {  
  g();  
}
```

method invocations: caught exceptions

VarPointsTo(?param, ?obj) <-

```
void f() {  
  try {  
    g();  
  } catch(E e) {...}  
}
```

method invocations: propagated exceptions

ThrowPointsTo(?caller, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
 Type[?obj] = ?objtype,

```
void f() {  
  g();  
}
```

method invocations: caught exceptions

VarPointsTo(?param, ?obj) <-

```
void f() {  
  try {  
    g();  
  } catch (E e) {...}  
}
```

method invocations: propagated exceptions

● `ThrowPointsTo(?caller, ?obj) <-`
 `CallGraphEdge(?invocation, ?tomethod),`
 `ThrowPointsTo(?tomethod, ?obj),`
 `Type[?obj] = ?objtype,`
 `not exists ExceptionHandler[?objtype, ?invocation],`

```
void f() {  
    g();  
}
```

method invocations: caught exceptions

`VarPointsTo(?param, ?obj) <-`

```
void f() {  
    try {  
        g();  
    } catch (E e) {...}  
}
```

method invocations: propagated exceptions

ThrowPointsTo(?caller, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
 Type[?obj] = ?objtype,
 not exists ExceptionHandler[?objtype, ?invocation],
 Method[?invocation] = ?caller.

```
void f() {  
  g();  
}
```

method invocations: caught exceptions

VarPointsTo(?param, ?obj) <-

```
void f() {  
  try {  
    g();  
  } catch (E e) {...}  
}
```

method invocations: propagated exceptions

- ThrowPointsTo(?caller, ?obj) <-
- CallGraphEdge(?invocation, ?tomethod),
- ThrowPointsTo(?tomethod, ?obj),
- Type[?obj] = ?objtype,
not exists ExceptionHandler[?objtype, ?invocation],
Method[?invocation] = ?caller.

```
void f() {  
    g();  
}
```

method invocations: caught exceptions

- VarPointsTo(?param, ?obj) <-
- CallGraphEdge(?invocation, ?tomethod),
- ThrowPointsTo(?tomethod, ?obj),
- Type[?obj] = ?objtype,

```
void f() {  
    try {  
        g();  
    } catch (E e) {...}  
}
```


method invocations: propagated exceptions

```
ThrowPointsTo(?caller, ?obj) <-  
  CallGraphEdge(?invocation, ?tomethod),  
  ThrowPointsTo(?tomethod, ?obj),  
  Type[?obj] = ?objtype,  
  not exists ExceptionHandler[?objtype, ?invocation],  
  Method[?invocation] = ?caller.
```

```
void f() {  
  g();  
}
```

method invocations: caught exceptions

- ```
VarPointsTo(?param, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
 Type[?obj] = ?objtype,
 ExceptionHandler[?objtype, ?invocation] = ?handler,
```

```
void f() {
 try {
 g();
 } catch (E e) {...}
}
```

## method invocations: propagated exceptions

```
ThrowPointsTo(?caller, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
 Type[?obj] = ?objtype,
 not exists ExceptionHandler[?objtype, ?invocation],
 Method[?invocation] = ?caller.
```

```
void f() {
 g();
}
```

## method invocations: caught exceptions

```
VarPointsTo(?param, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
 Type[?obj] = ?objtype,
 ExceptionHandler[?objtype, ?invocation] = ?handler,
 ExceptionHandler:FormalParam[?handler] = ?param.
```

```
void f() {
 try {
 g();
 } catch (E e) {...}
}
```

## method invocations: propagated exceptions

```
ThrowPointsTo(?caller, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
 Type[?obj] = ?objtype,
 not exists ExceptionHandler[?objtype, ?invocation],
 Method[?invocation] = ?caller.
```

```
void f() {
 g();
}
```

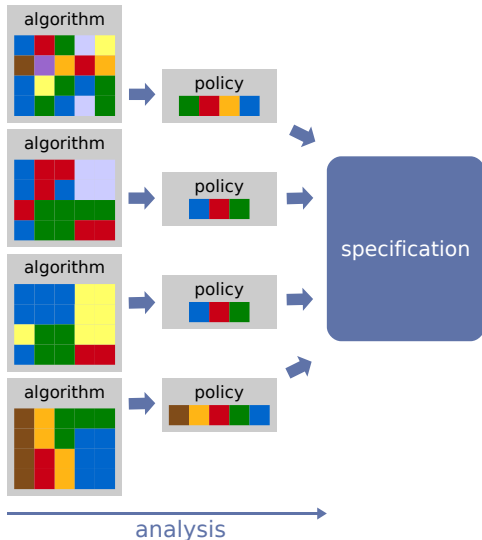
## method invocations: caught exceptions

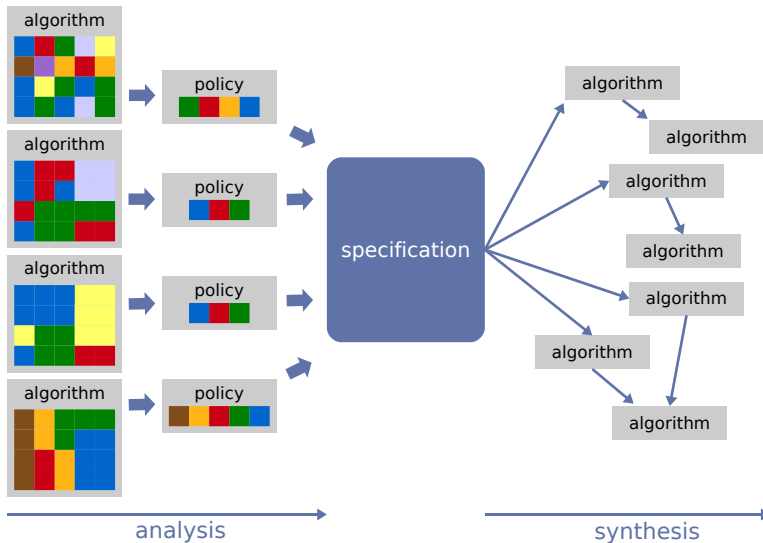
```
VarPointsTo(?param, ?obj) <-
 CallGraphEdge(?invocation, ?tomethod),
 ThrowPointsTo(?tomethod, ?obj),
```

```
void f() {
 try {
 g();
 } catch(E e) {...}
}
```

again, new mutually recursive dependencies do not change existing rules

```
ExceptionHandler[?objtype, ?invocation] = ?handler,
Param[?handler] = ?param.
```

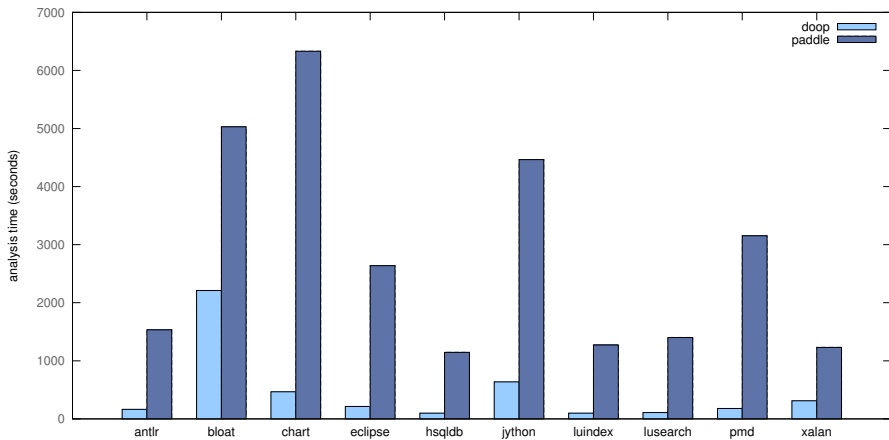




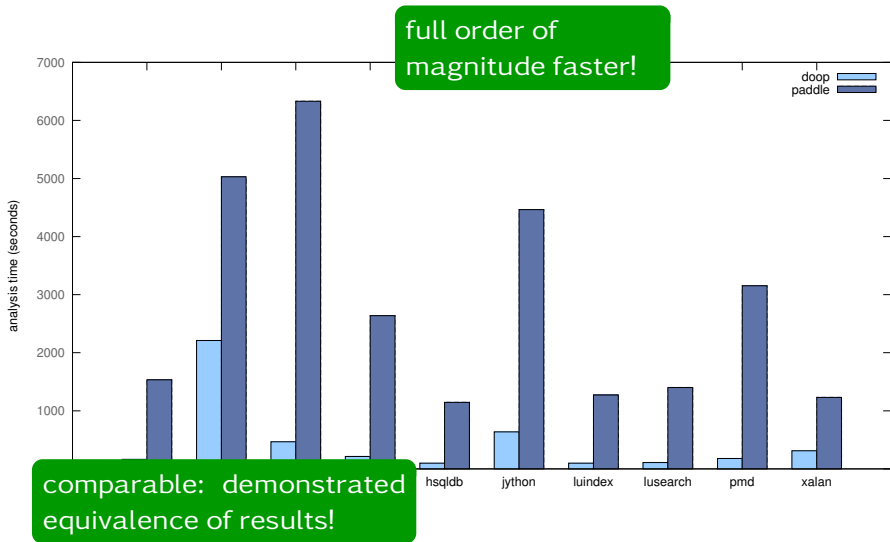
from specification to algorithms

from specification to algorithms

benchmark







strictly declarative

- no coupling of specification and algorithm

efficient architecture for datalog engine

- but, not a magic tool

⇒ novel heuristics for searching the algorithm space

- targeting recursive logic

from specification to algorithms

datalog evaluation background

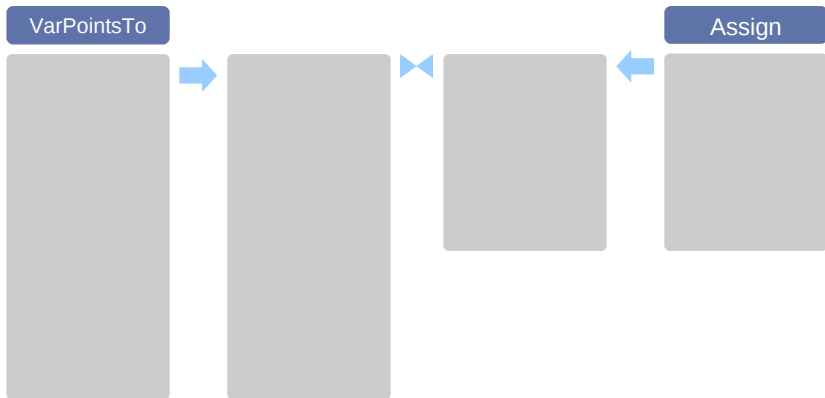
## datalog

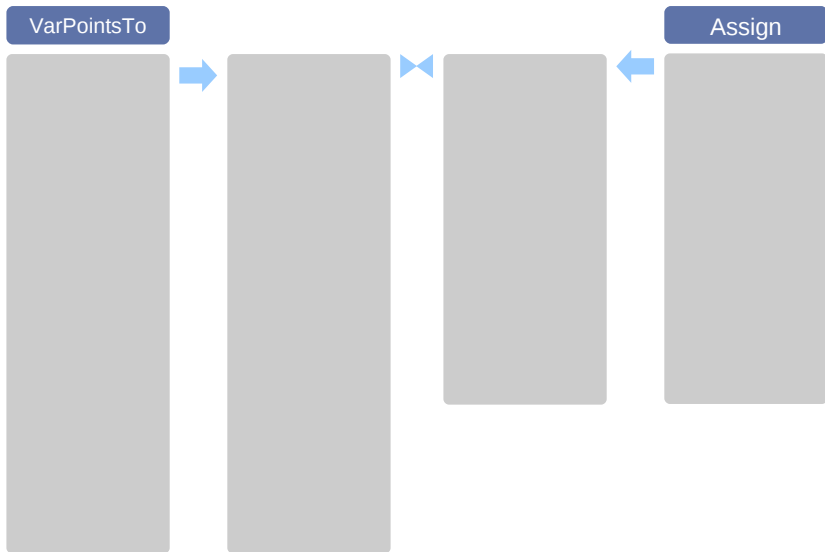
```
VarPointsTo(?var, ?obj) <-
 AssignObjectAllocation(?obj, ?var).
```

```
VarPointsTo(?to, ?obj) <-
 Assign(?from, ?to), VarPointsTo(?from, ?obj).
```

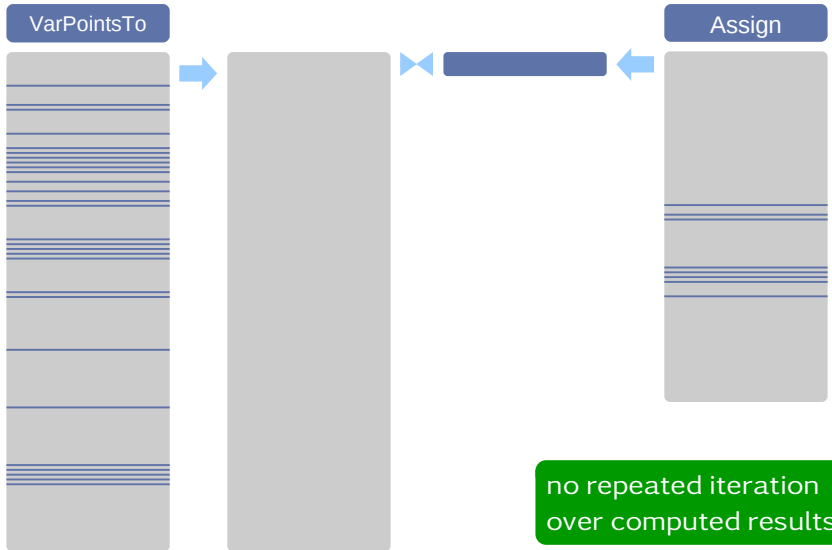
## naive evaluation (relational algebra)

```
VarPointsTo := AssignObjectAllocation
repeat
 tmp := $\pi_{to \rightarrow var, heap}$ (VarPointsTo $\bowtie_{from=var}$ Assign)
 VarPointsTo := VarPointsTo $\cup$ tmp
until fixpoint
```

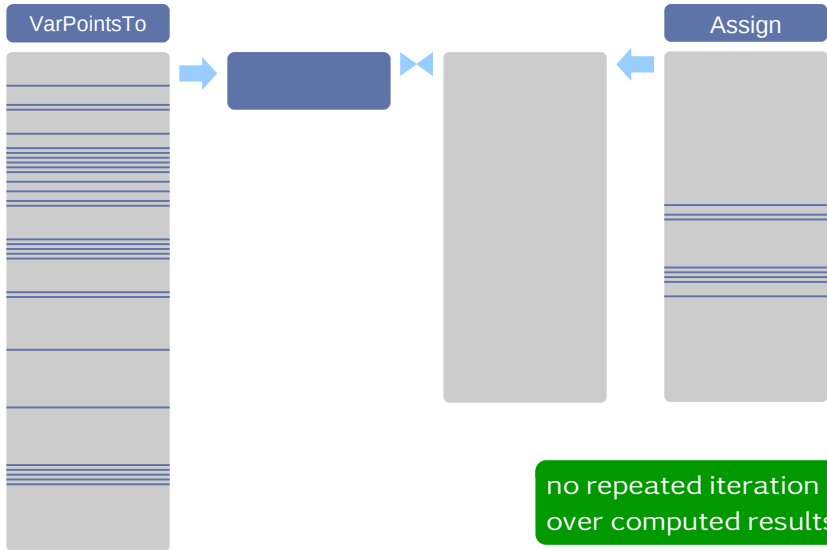




[illegible]







from specification to algorithms

**datalog engine architecture**

## efficient low-level representation

"<java.lang.Thread: void <clinit>()>" → 32-bit int  
call-graph edge: invocation × method → 64-bit int

## relations are multidimensional indexes (b-trees)

'index-organized tables'

## efficient representation of sparse relations

alternative data structures for (partially) dense relations

## indexes exposed in the language

argument order determines index  
enables us to stay in datalog

## VarPointsTo

|   |         |
|---|---------|
| a | new A() |
| b | new B() |
| c | new C() |
| a | new B() |
| b | new A() |
| c | new B() |
| c | new A() |

## VarPointsTo

|   |         |
|---|---------|
| a | new A() |
| b | new B() |
| c | new C() |
| a | new B() |
| b | new A() |
| c | new B() |
| c | new A() |

index organized by  
reverse argument order

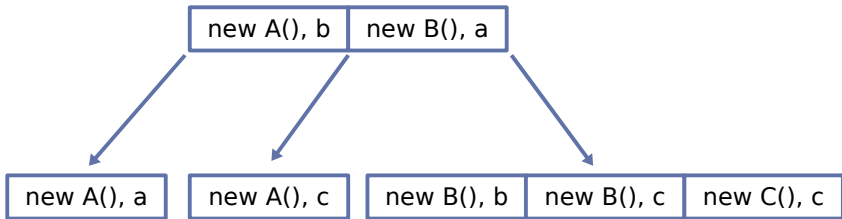
VarPointsTo(?var, ?obj)

## VarPointsTo

|   |         |
|---|---------|
| a | new A() |
| b | new B() |
| c | new C() |
| a | new B() |
| b | new A() |
| c | new B() |
| c | new A() |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)



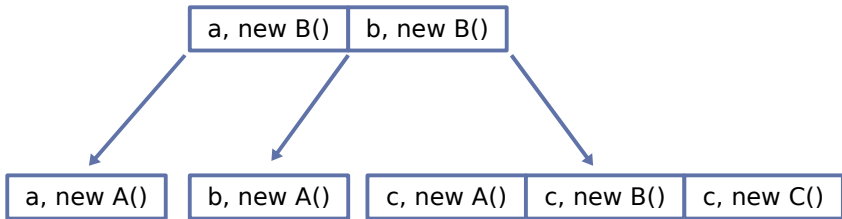
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)



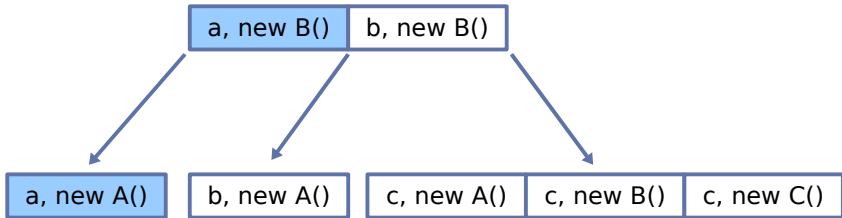
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)





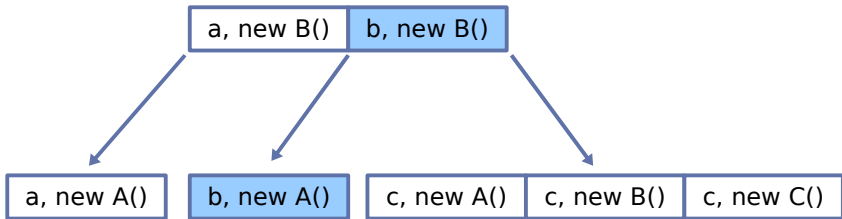
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)



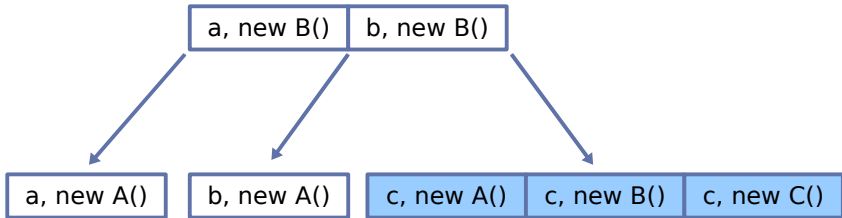
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)



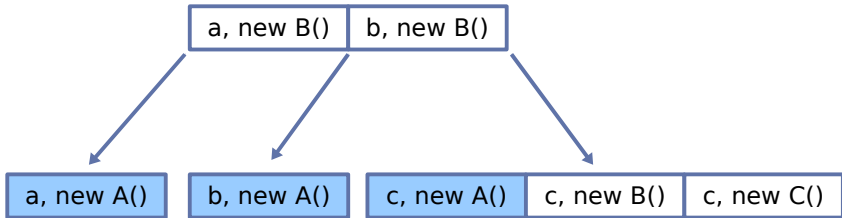
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)



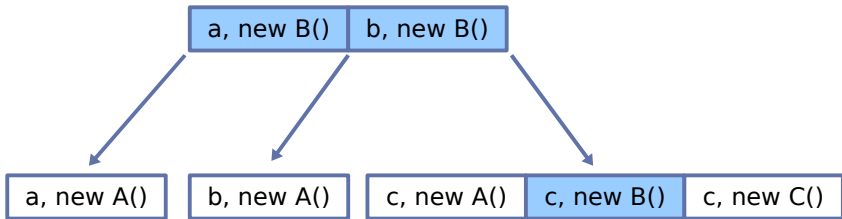
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)



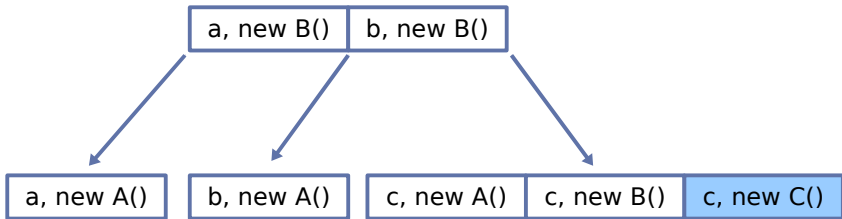
## VarPointsTo

|         |   |
|---------|---|
| new A() | a |
| new B() | b |
| new C() | c |
| new B() | a |
| new A() | b |
| new B() | c |
| new A() | c |

index organized by  
reverse argument order

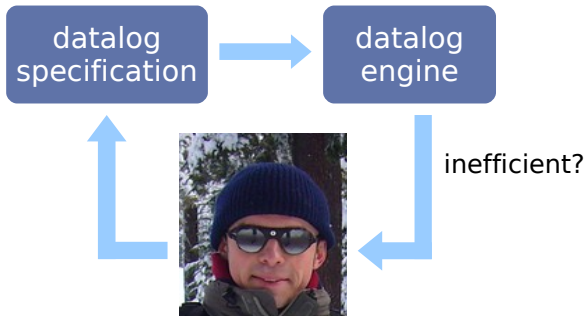
VarPointsTo(?var, ?obj)

- VarPointsTo(?obj, ?var)



from specification to algorithms

heuristics for searching the algorithm space



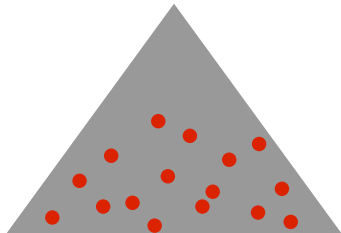
traditional design of single algorithm major human effort!

always use index efficiently



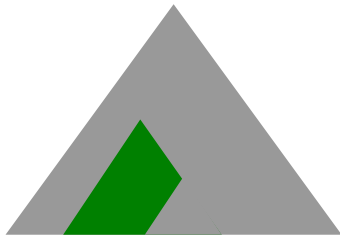
always use index efficiently

- $\Delta\text{VarPointsTo}(\text{?from}, \text{?obj}) \propto \text{Assign}(\text{?from}, \text{?to})$



## always use index efficiently

- $\Delta\text{VarPointsTo}(\text{?from}, \text{?obj}) \bowtie \text{Assign}(\text{?from}, \text{?to})$
- $\Delta\text{VarPointsTo}(\text{?from}, \text{?obj}) \bowtie \text{Assign}(\text{?to}, \text{?from})$



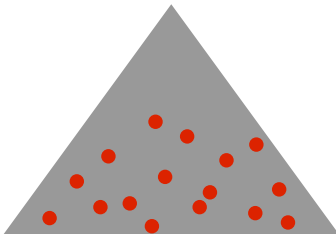
always use index efficiently

never iterate over full views

always use index efficiently

never iterate over full views

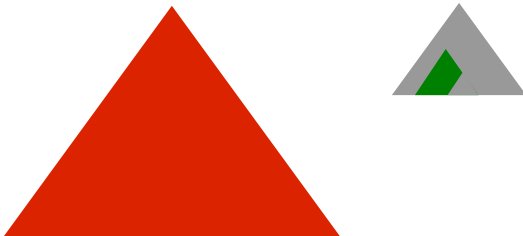
- `ΔAssign(?to, ?from) ⋈ VarPointsTo(?from, ?obj)`



always use index efficiently

never iterate over full views

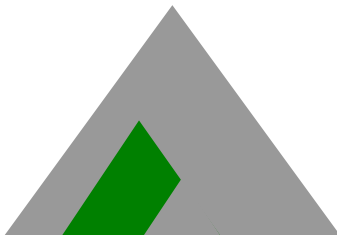
- $\Delta\text{Assign}(\text{?to}, \text{?from}) \bowtie \text{VarPointsTo}(\text{?from}, \text{?obj})$
- $\text{VarPointsTo}(\text{?from}, \text{?obj}) \bowtie \Delta\text{Assign}(\text{?to}, \text{?from})$



always use index efficiently

never iterate over full views

- $\Delta\text{Assign}(\text{?to}, \text{?from}) \bowtie \text{VarPointsTo}(\text{?from}, \text{?obj})$
- $\text{VarPointsTo}(\text{?from}, \text{?obj}) \bowtie \Delta\text{Assign}(\text{?to}, \text{?from})$
- $\Delta\text{Assign}(\text{?to}, \text{?from}) \bowtie \text{VarPointsTo}(\text{?obj}, \text{?from})$



always use index efficiently

never iterate over full views

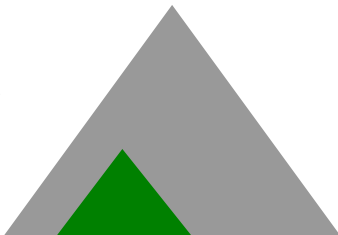
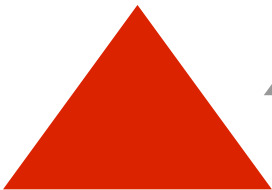
never iterate over big input relations

always use index efficiently

never iterate over full views

never iterate over big input relations

- StoreField(?from, ?field, ?base)
  - ⊗ ΔVarPointsTo(?obj, ?from)
  - ⊗ VarPointsTo(?baseobj, ?base)





from specification to algorithms

folding transformation example

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

## semi-naive executions

```
 Δ FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 Δ VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

```
 Δ FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 Δ VarPointsTo(?obj, ?from).
```

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

## join orderings

```
ΔVarPointsTo(?baseobj, ?base)
⊗ StoreField(?from, ?base, ?field)
⊗ VarPointsTo(?obj, ?from)
```

```
ΔVarPointsTo(?obj, ?from)
⊗ StoreField(?from, ?base, ?field)
⊗ VarPointsTo(?baseobj, ?base)
```

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

there is no efficient  
variable ordering!

## join orderings

- $\Delta$ VarPointsTo(?baseobj, ?base)
  - ⊗ StoreField(?from, ?base, ?field)
  - ⊗ VarPointsTo(?obj, ?from)
- $\Delta$ VarPointsTo(?obj, ?from)
  - ⊗ StoreField(?from, ?base, ?field)
  - ⊗ VarPointsTo(?baseobj, ?base)

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

there is no efficient  
variable ordering!

## join orderings

```
ΔVarPointsTo(?baseobj, ?base)
⊗ StoreField(?from, ?base, ?field)
⊗ VarPointsTo(?obj, ?from)
```

```
ΔVarPointsTo(?obj, ?from)
⊗ StoreField(?from, ?base, ?field)
⊗ VarPointsTo(?baseobj, ?base)
```

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

there is no efficient  
variable ordering!

## join orderings

```
ΔVarPointsTo(?baseobj, ?base)
⊗ StoreField(?from, ?base, ?field)
⊗ VarPointsTo(?obj, ?from)
```

```
ΔVarPointsTo(?obj, ?from)
⊗ StoreField(?from, ?base, ?field)
⊗ VarPointsTo(?baseobj, ?base)
```

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
● StoreField(?from, ?base, ?field),
● VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```



## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
● StoreField(?from, ?base, ?field),
● VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

## optimized rules

```
FieldPointsTo(?obj, ?field, ?baseobj) <-
 VarPointsTo(?obj, ?from).

StoreObjectField(?baseobj, ?field, ?from) <-
● StoreField(?from, ?field, ?base),
● VarPointsTo(?baseobj, ?base).
```

## specification

```
FieldPointsTo(?baseobj, ?field, ?obj) <-
 StoreField(?from, ?base, ?field),
 VarPointsTo(?baseobj, ?base),
 VarPointsTo(?obj, ?from).
```

## optimized rules

- ```
FieldPointsTo(?obj, ?field, ?baseobj) <-  
  StoreObjectField(?baseobj, ?field, ?from),  
  VarPointsTo(?obj, ?from).
```
- ```
StoreObjectField(?baseobj, ?field, ?from) <-
 StoreField(?from, ?field, ?base),
 VarPointsTo(?baseobj, ?base).
```

inclusion-based

unification-based

flow-sensitive

on-the-fly call graph

context-sensitive

k-cfa

object sensitive

field-based

field-sensitive

heap cloning

binary decision diagrams

demand-driven

Results 1 - 20 of 2,343

[Save results to a binder](#)

Sort by  in

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#) [>>](#)

## 1 [Semi-sparsely flow-sensitive pointer analysis](#)

January 2009 **POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

**Publisher:** ACM

Full text available: [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** alias analysis, pointer analysis

## 2 [Efficient field-sensitive pointer analysis of C](#)

David J. Pearce, Paul H.J. Kelly, Chris Hankin

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available: [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** Set-constraints, pointer analysis

## 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)

John Whaley, Monica S. Lam

June 2004 **PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**

**Publisher:** ACM

Full text available: [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

inclusion-based

unification-based

flow-sensitive

on-the-fly call graph

context-sensitive

k-cfa

object sensitive

field-based

field-sensitive

heap cloning

binary decision diagrams

demand-driven

Results 1 - 20 of 2,343

 [Save results to a binder](#)

- 1 [Semi-sparse flow-sensitive pointer analysis](#)  
January 2009 **POPL '09: Proceedings of the ACM SIGPLAN conference on Programming languages**

**Publisher:** ACM

Full text available:  [Pdf](#) (246.09 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0

Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...

**Keywords:** [alias analysis](#), [pointer analysis](#)

- 2 [Efficient field-sensitive pointer analysis of C](#)  
[David J. Pearce](#), [Paul H.J. Kelly](#), [Chris Hankin](#)

November 2007 **Transactions on Programming Languages and Systems (TOPLAS)**, Volume 30 Issue 1

**Publisher:** ACM

Full text available:  [Pdf](#) (924.64 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

**Bibliometrics:** Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1

The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...

**Keywords:** [Set-constraints](#), [pointer analysis](#)

- 3 [Cloning-based context-sensitive pointer alias analysis using binary decision diagrams](#)  
[John Whaley](#), [Monica S. Lam](#)

June 2004 **PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**

**Publisher:** ACM

Full text available:  [Pdf](#) (937.61 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

database community:

~ index selection

~ materialized view selection

from algorithms to specification

from specification to algorithms

## distilled domain of pointer analysis to its essence

- declarative specification
- fully captured the domain: no loss of sophistication
  - even improved: exception, reflection analysis
  - declarativeness is a major benefit here

## developed techniques for deriving efficient algorithms

- effective heuristics for searching the algorithm space
- outperforms current hand-written implementations
  - ⇒ evidence that this is an effective method!

### pointer analysis

- new context abstractions
- selective context-sensitivity
- on-demand analysis
- improve evaluation methods

### pointer analysis

- new context abstractions
- selective context-sensitivity
- on-demand analysis
- improve evaluation methods

### program analysis and applications

- mixed language analysis
- declarative dynamic program analysis
- applications: client analyses



## pointer analysis

- new context abstractions
- selective context-sensitivity
- on-demand analysis
- improve evaluation methods

## program analysis and applications

- mixed language analysis
- declarative dynamic program analysis
- applications: client analyses

## transformation systems, parsing, extensible languages

- too much to list here!

## extensible and domain-specific languages

embedding domain-specific languages

[OOPSLA 2004, GPCE 2005, OOPSLA 2006, OOPSLA 2008, ATEM 2007]

preventing injection attacks [GPCE 2007, SCP 2009]

parse table composition [SLE 2008]

## program transformation systems

stratego/xt transformation system [PEPM 2006, FI 2006, SCP 2008]

grammar engineering [LDTA 2007]

## program analysis

scalable declarative pointer analysis [ISSTA 2009]