my research

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

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");
example 1: embedding in string literals

```java
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

```sql
SELECT id FROM users 
WHERE name = 'martin' AND 
password = 'ilovewaterloo'
```
example 1: embedding in string literals

```java
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
example 2: implement a graphical user-interface

- north
- center
- south
example 2: implement a graphical user-interface

Hello World

north

center

south

Ok    Cancel

east
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

```java
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);
        ...
    }
}
```
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);
        ...
General-purpose languages do not give you the means to write coherent sentences over the vocabulary of an API.
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);
}
example 3: libraries as half-baked languages

```java
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

```java
class Foo {
    pointcut foo() : call(boolean *.if*());
}
```
example 4: failed effort for specific combinations

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

```bash
$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

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

```java
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
```

```java
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
Example 1: preventing injection attacks

Java and SQL

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

```sql
SQL q = <| SELECT id FROM users
WHERE name = ${userName}
AND password = ${password} |>
if (executeQuery(q.toString()).size() == 0) ...
```
example 1: preventing injection attacks

php and shell commands

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

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

java and ldap

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

```java
LDAP q = (| (cn=$(name)) |);
```
example 2: implement a graphical user-interface
example 2: implement a graphical user-interface

![Diagram of a user interface with sections labeled north, center, south, and east.]
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

```java
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
technical challenge: grammar composition

- scanners do not compose
  - scannerless parser
- lr grammars do not compose
  - cfg, generalized lr
technical challenge: grammar composition

- scanners do not compose
- lr grammars do not compose
technical challenge: grammar composition

developer

java grm
sql grm

java+sql grammar

generate

java+sql parser
technical challenge: grammar composition

- Scanners do not compose → complex stateful scanners and/or bugs
- LR grammars do not compose → rewrite grammar to resolve conflicts

Diagram:

- Developer
- Generate
- Java + SQL grammar
- Java + SQL parser
- Java grm
- SQL grm
technical challenge: grammar composition

- developer
  - java grm
  - sql grm
  - java+sql grammar
  - generate
    - java+sql parser

- compose
  - java grm
  - sql grm
  - java+sql grammar

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

developer

java grm  sql grm

java+sql grammar

generate

java+sql parser

compose

java grm  sql grm

java+sql grammar

generate

java+sql parser
technical challenge: grammar composition

- scanners do not compose → scannerless parser
- lr grammars do not compose → cfg, generalized lr
technical challenge: parse table composition

• from source-level extensibility to binary extensibility
• algorithm: minimize DF A reconstruction
technical challenge: parse table composition

• from source-level extensibility to binary extensibility
• algorithm: minimize DF A reconstruction
technical challenge: parse table composition

- compose
  - java
  - sql
  - grammar
- generate
  - java + sql parser

- from source-level extensibility to binary extensibility
- algorithm: minimize DF A reconstruction
technical challenge: parse table composition

compose

java grm
sql grm

java+sql grammar

generate

java+sql parser

generate

java grm
sql grm

java tbl
sql tbl

• from source-level extensibility to binary extensibility
• algorithm: minimize DF A reconstruction
technical challenge: parse table composition

compose

generate

java+sql grammar

java+sql parser

java grm
sql grm

java+sql parser

java grm
sql grm

java tbl
sql tbl

java+sql parser
technical challenge: parse table composition

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

extensible and domain-specific languages

program transformation

program analysis
my research

extensible and domain-specific languages

embedding domain-specific languages

>100 citations  best paper

program transformation

program analysis
my research

**extensible and domain-specific languages**

embedding domain-specific languages


preventing injection attacks [GPCE 2007, SCP 2009]

**program transformation**

**program analysis**
my research

extensible and domain-specific languages
embedding domain-specific languages
preventing injection attacks [GPCE 2007, SCP 2009]
parse table composition [SLE 2008]

program transformation

program analysis
my research

extensible and domain-specific languages

embedding domain-specific languages


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
my research

extensible and domain-specific languages

embedding domain-specific languages
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
<table>
<thead>
<tr>
<th>my research</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>extensible and domain-specific languages</strong></td>
</tr>
<tr>
<td>embedding domain-specific languages</td>
</tr>
<tr>
<td>preventing injection attacks [GPCE 2007, SCP 2009]</td>
</tr>
<tr>
<td>parse table composition [SLE 2008]</td>
</tr>
<tr>
<td><strong>program transformation</strong></td>
</tr>
<tr>
<td>stratego/xt transformation system [PEPM 2006, FI 2006, SCP 2008]</td>
</tr>
<tr>
<td>grammar engineering [LDTA 2007]</td>
</tr>
<tr>
<td><strong>program analysis</strong></td>
</tr>
<tr>
<td>scalable declarative pointer analysis [ISSTA 2009]</td>
</tr>
<tr>
<td>latest and greatest work, rest of the talk</td>
</tr>
</tbody>
</table>
what do we do?

fully declarative pointer analysis
fast, really fast
overview

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 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
program analysis: run faster
program analysis: find bugs
what objects can a variable point to?

```java
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

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

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

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

points-to

```
<table>
<thead>
<tr>
<th>foo:a</th>
<th>new A1()</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar:a</td>
<td>new A2()</td>
</tr>
<tr>
<td>id:a</td>
<td>new A1(), new A2()</td>
</tr>
</tbody>
</table>
```
what objects can a variable point to?

<table>
<thead>
<tr>
<th>program</th>
<th>points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void foo() {</strong></td>
<td><strong>foo:a</strong> new A1()</td>
</tr>
<tr>
<td>a = new A1();</td>
<td>bar:a new A2()</td>
</tr>
<tr>
<td>b = id(a);</td>
<td>id:a new A1(), new A2()</td>
</tr>
<tr>
<td><strong>}</strong></td>
<td>foo:b new A1(), new A2()</td>
</tr>
<tr>
<td><strong>void bar() {</strong></td>
<td>bar:b new A1(), new A2()</td>
</tr>
<tr>
<td>a = new A2();</td>
<td></td>
</tr>
<tr>
<td>b = id(a);</td>
<td></td>
</tr>
<tr>
<td><strong>}</strong></td>
<td></td>
</tr>
<tr>
<td>A id(A a) {</td>
<td></td>
</tr>
<tr>
<td>return a;</td>
<td></td>
</tr>
</tbody>
</table>
**What objects can a variable point to?**

### Program

```java
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**

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

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

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

**points-to**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>foo:a</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:a</td>
<td>new A2()</td>
</tr>
<tr>
<td>id:a</td>
<td>new A1(), new A2()</td>
</tr>
<tr>
<td>foo:b</td>
<td>new A1(), new A2()</td>
</tr>
<tr>
<td>bar:b</td>
<td>new A1(), new A2()</td>
</tr>
</tbody>
</table>

**context-sensitive points-to**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>foo:a</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:a</td>
<td>new A2()</td>
</tr>
<tr>
<td>id:a (foo)</td>
<td>new A1()</td>
</tr>
<tr>
<td>id:a (bar)</td>
<td>new A2()</td>
</tr>
<tr>
<td>foo:b</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:b</td>
<td>new A2()</td>
</tr>
</tbody>
</table>

necessarily an approximation
full precision too expensive
pointer analysis: a complex domain

1. Semi-sparse flow-sensitive pointer analysis
   Publisher: ACM
   Full text available: Pdf (246.08 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 (824.64 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 (257.21 KB)  Additional Information: full citation, abstract, references, index terms
   Bibliometrics: Downloads (6 Weeks): 11, Downloads (12 Months): 72, Citation Count: 1
flow-sensitive

1. Semi-sparse flow-sensitive pointer analysis
   Publisher: ACM
   Full text available: Pdf (246.05 KB)
   Additional Information: full citation, abstract, references, index terms
   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 (824.64 KB)
   Additional Information: full citation, abstract, references, index terms
   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 (247.05 KB)

   The method presented here introduces a novel approach to context-sensitive pointer alias analysis using binary decision diagrams (BDDs). BDDs are an efficient data structure for representing Boolean functions, which makes them well-suited for pointer alias analysis where the goal is to determine whether two pointers may alias in the context of a given program state. The approach presented in this paper uses a custom BDD representation to efficiently store and manipulate the information necessary for context-sensitive alias analysis. The method is evaluated experimentally and compared with existing techniques, demonstrating its effectiveness and efficiency.

   Keywords: context-sensitive pointer alias analysis, binary decision diagrams
pointer analysis: a complex domain

1. Semi-sparse flow-sensitive pointer analysis
   - Publisher: ACM
   - Bibliometrics: Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0
   - Keywords: alias analysis, pointer analysis
   - 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, ...

2. Efficient field-sensitive pointer analysis of C
   - Publisher: ACM
   - Bibliometrics: Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1
   - Keywords: Set-constraints, pointer analysis
   - 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 ...

3. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams
   - Publisher: ACM
   - Bibliometrics: Downloads (6 Weeks): 22, Downloads (12 Months): 33, Citation Count: 0
   - Keywords: pointer analysis
   - The document discusses a method for alias analysis that uses binary decision diagrams for cloning-based context-sensitive analysis.
pointer analysis: a complex domain

flow-sensitive

context-sensitive

field-sensitive
pointer analysis: a complex domain

- flow-sensitive
- context-sensitive
- field-sensitive
- binary decision diagrams
<table>
<thead>
<tr>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-sparse flow-sensitive pointer analysis</td>
<td>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,...</td>
</tr>
<tr>
<td>Publisher: ACM</td>
<td></td>
</tr>
<tr>
<td>Full text available: [Pdf](2456 KB)</td>
<td></td>
</tr>
<tr>
<td>Additional Information: full citation, abstract, references, index terms</td>
<td></td>
</tr>
<tr>
<td>Bibilometrics: Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0</td>
<td></td>
</tr>
<tr>
<td>Keywords: alias analysis, pointer analysis</td>
<td></td>
</tr>
<tr>
<td>Efficient field-sensitive pointer analysis of C</td>
<td>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...</td>
</tr>
<tr>
<td>David J. Pearce, Paul Ged, Kelly, Chris Hankin</td>
<td></td>
</tr>
<tr>
<td>Publisher: ACM</td>
<td></td>
</tr>
<tr>
<td>Full text available: [Pdf](824.64 KB)</td>
<td></td>
</tr>
<tr>
<td>Additional Information: full citation, abstract, references, index terms</td>
<td></td>
</tr>
<tr>
<td>Bibilometrics: Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1</td>
<td></td>
</tr>
<tr>
<td>Keywords: Set-constraints, pointer analysis</td>
<td></td>
</tr>
<tr>
<td>Cloning-based context-sensitive pointer alias analysis using binary decision diagrams</td>
<td></td>
</tr>
<tr>
<td>John Whaley, Monica S. Lam</td>
<td></td>
</tr>
<tr>
<td>Publisher: ACM</td>
<td></td>
</tr>
<tr>
<td>Full text available: [Pdf](2456 KB)</td>
<td></td>
</tr>
<tr>
<td>Additional Information: full citation, abstract, references, index terms</td>
<td></td>
</tr>
</tbody>
</table>
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_int_by_assignment(N,YES);
       2.2 else if N is a call node
           aliases_int_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
Figure 1: Exhaustive algorithm for pointer aliasing

\begin{algorithm}
\SetAlgoLined
\DontPrintSemicolon
\textbf{procedure} exhaustive_aliasing($G$)
\begin{algorithmic}
\State $G$: an ICFG;
\State $N$: a statement to be changed;
\begin{algorithmic}
\State \textbf{procedure} incremental_aliasing($G,N$)
\end{algorithmic}
\State \begin{algorithmic}
\State 1. falsify the affected aliases, which are either generated at $N$, or depend on other affected aliases.
\State 2. update $G$ to reflect the change to statement $N$;
\State 3. worklist$=\text{reintroduce}\_\text{aliases}(G)$;
\State 4. worklist$=\text{reiterate}\_\text{worklist}(\text{worklist},\text{YES})$;
\State end
\end{algorithmic}
\end{algorithmic}
\State \begin{algorithmic}
\State \begin{algorithmic}
\State 3.3 \textbf{else} if $N$ is an exit node
\State \hspace{1em} \textit{alias}\_\textit{at}\_\textit{exit}\_\text{implies}(N,AA,PA,YES);
\State 3.4 \textbf{else} for each $M \in \text{successor}(N)$
\State \hspace{1em} \begin{algorithmic}
\State \hspace{2em} \textbf{if} $M$ is a pointer assignment
\State \hspace{4em} \textit{alias}\_\textit{implies}\_\textit{thru}\_\textit{assign}(M,AA,PA,YES);
\State \hspace{2em} \textbf{else}
\State \hspace{4em} make.true($M,AA,PA$);
\State \hspace{1em} \end{algorithmic}
\State end
\end{algorithmic}
\State \end{algorithmic}
\State \end{algorithmic}
\end{algorithm}


**Figure 1:** Exhaustive alias analysis

```
procedure exhaustive-aliasing(C, N)
begin
1. if N is marked TOUCHEO, return.
2. set all may-hold(N, AA, A) to FALSIFIED;
3. for each call node C in function E;
   disable-aliases(entry of the function called by C);
4. for each call node C in function E;
   disable-aliases(entry of the function called by C);
5.1 if E is marked INFLUENCED, return;
5.2 if N is a call node containing N;
   native-falsification(corresponding return of C);
6. else for each M ∈ successors(N);
   native-falsification(M);
end
```

**Figure 2:** Incremental alias analysis

```
procedure incremental-aliasing(C, N)
begin
1. if N is marked TOUCHEO, return.
2. set all may-hold(N, AA, A) to NO;
3. 1 if N is an exit node;
   native-falsification(corresponding return of N);
4. for each call node C which calls the function
   containing N;
   native-falsification(entry of the function called by N);
5.1 else if N is a call node
   disable-aliases(entry of the function called by N);
6. else for each M ∈ successors(N);
   native-falsification(M);
end
```

* Alias falsification corresponding to step 1 in Figure 2 */
procedure exhaustive_aliasing(G)
begin
  G: a program graph in the ICFG;

begin
  1. if is called function is INFLUENCED, then
  2.1 aliases_intro_by_call(C,YES);
      /* Inter-procedural propagation */
  2.2 repropagate_aliases(C,worklist);
  /* Intra-procedural propagation */
  3. for each TOUCHED node N in G
  3.1 if N is a pointer assignment statement, then
      aliases_intro_by_assignment(M,YES);
  3.2 for each M ∈ predecessor(N)
      repropagate_aliases(M,worklist);
  4. return worklist;
end

procedure repropagate_aliases(N,worklist)
begin
  N: a program node in the ICFG;
  worklist: a worklist for keeping the reintroduced aliases;

begin
  for each may_hold(N,AA,PA) = YES
  add (N,AA,PA) to worklist;
end

/* Aliases falsification corresponding to step 1 in Figure 2 */

procedure incremental_aliasing(G,N)
begin
  G: a program graph in the ICFG;
  N: a node in G;

begin
  if is called function is INFLUENCED then
  1. aliases_intro_by_call(C,YES);
  2.1.1 aliases_intro_by_call(C,YES);
  2.2.1 repropagate_aliases(C,worklist);
  /* Intra-procedural propagation */
  3. for each TOUCHED node N in G
  3.1 if N is a pointer assignment statement, then
      aliases_intro_by_assignment(M,YES);
  3.2 for each M ∈ predecessor(N)
      repropagate_aliases(M,worklist);
  4. return worklist;
end

/* Aliases reintroduction corresponding to step 3 in Figure 2 */

Figure 1: Exhaustive aliasing

Figure 2: Incremental aliasing
algorithms in a 10-page pointer analysis paper

```plaintext
procedure exhaustive_aliasing(G)
begin
G: a 10-page pointer analysis paper
begin
/* Alias falsification corresponding to step 1 in Figure 2 */
procedure incremental_aliasing(G, N)
begin
N: a 10-page pointer analysis paper
begin
/* Alias reintroduction corresponding to step 3 in Figure 2 */
procedure reiterate_worklist(worklist, value)
return
worklist: a worklist for keeping the aliases to process;
value: value that will be given to (N, AA, PA);
begin
begin
1. while worklist is not empty do
2. for each (N, AA, PA) in worklist do
3. for each M in successor(N) do
4. if (N, AA, PA) = YES then
5. add (N, AA, PA) to worklist;
else
6. if (N, AA, PA) = FALSE then
7. remove (N, AA, PA) from worklist;
8. if N is a call node
9. aliases_propagated_at_call(N, AA, PA, value);
10. else if N is an exit node
11. alias_at_exit_implies(N, AA, PA, value);
else
12. if value is YES
13. make_true(M, AA, PA);
else /* value is FALSIFIED */
14. make_false(M, AA, PA);
end
end
end
end
end
```

Figure 1: Exhaustive aliasing

Figure 5: Reiteration for the incremental algorithm
procedure exhaustive_aliasing(N)
begin
1. if
2. else
3. if
4. then
5. else
6. if
7. end
end

Figure 1: Exhaustive aliasing algorithm

procedure incremental_aliasing(N)
begin
1. if
2. then
3. else
4. if
5. then
6. else
7. end
end

procedure aliases_propagated_at_call(N, AA, PA, value)
begin
1. create an empty worklist;
2. for each
3. if
4. then
5. else
6. if
7. then
8. end
9. end

Figure 2: Incremental aliasing algorithm
Figure 1: Exhaustive aliasing algorithm.

Figure 2: Incremental aliasing algorithms.

Figure 3: Alias reinsertion and reiteration algorithms.

Figure 4: Procedure for adding an alias assignment.

Figure 5: Procedure for false-finding alias assignments.

Figure 8: 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

Figure 8: Procedure for falsifying aliases that are potentially affected by adding a pointer assignment
from algorithms to specification and back
from algorithms to specification and back
from algorithms to specification and back
from algorithms to specification and back
from algorithms to specification
what does it mean to be declarative?

“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

- 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
• 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
key contributions

expressed pointer analysis in full sophistication in datalog

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

approach: heuristics for searching algorithm space
- targeted at recursive problem domains

demonstrated scalability with explicit representation
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
key contributions

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
key contributions

expressed pointer analysis in full sophistication in datalog
- core specification: $\sim$250 lines of logic
- full specification: $\sim$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
var points-to
$x = y$

var points-to
program analysis: a domain of mutual recursion 26

x = y

var points-to
x = f()

var points-to

call graph
x = f()

var points-to

call graph
x = y.f()
x = new A()
\texttt{x = new A()}

\textbf{var points-to}

\textbf{call graph}

\textbf{reachable methods}
x.f = y

var points-to

field points-to

reachable methods

call graph
x.f = y
\[ x = y.f \]
throw e

var points-to

call graph

reachable methods

field points-to

exceptions
catch(E e)

var points-to

call graph

reachable methods

exceptions

field points-to
program analysis: a domain of mutual recursion
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

<table>
<thead>
<tr>
<th>a</th>
<th>new A()</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

Assign

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
source

```java
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

<table>
<thead>
<tr>
<th>var</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

Assign

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

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

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

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

AssignObjectAllocation

```plaintext
AssignObjectAllocation(?var, ?obj).
```

Assgin

```plaintext
Assign(?from, ?to),
VarPointsTo(?from, ?obj).
```

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

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

### 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()
```

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(?var, ?obj) <-

AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-

Assign(?from, ?to),
VarPointsTo(?from, ?obj).
### datalog: declarative mutual recursion

#### Source

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

#### AssignObjectAllocation

<table>
<thead>
<tr>
<th><code>a</code></th>
<th><code>new A()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b</code></td>
<td><code>new B()</code></td>
</tr>
<tr>
<td><code>c</code></td>
<td><code>new C()</code></td>
</tr>
</tbody>
</table>

#### Assign

<table>
<thead>
<tr>
<th><code>b</code></th>
<th><code>a</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td><code>b</code></td>
</tr>
<tr>
<td><code>b</code></td>
<td><code>c</code></td>
</tr>
</tbody>
</table>

#### 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

<table>
<thead>
<tr>
<th>var</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

VarPointsTo

VarPointsTo(\text{var}, \text{obj}) \leftarrow
AssignObjectAllocation(\text{var}, \text{obj}).

VarPointsTo(\text{to}, \text{obj}) \leftarrow
Assign(\text{from}, \text{to}),
VarPointsTo(\text{from}, \text{obj}).
AssignObjectAllocation

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

Assign

b  a
a  b
b  c

VarPointsTo(\texttt{?var}, \texttt{?obj}) \leftarrow
AssignObjectAllocation(\texttt{?var}, \texttt{?obj}).

VarPointsTo(\texttt{?to}, \texttt{?obj}) \leftarrow
Assign(\texttt{?from}, \texttt{?to}),
VarPointsTo(\texttt{?from}, \texttt{?obj}).

source

a = \texttt{new A();}
b = \texttt{new B();}
c = \texttt{new C();}
a = b;
b = a;
c = b;
## datalog: declarative mutual recursion

### Source

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

### AssignObjectAllocation

<table>
<thead>
<tr>
<th>var</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

### Assign

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

### VarPointsTo

<table>
<thead>
<tr>
<th>var</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
</tbody>
</table>

1. `VarPointsTo(?var, ?obj) <- AssignObjectAllocation(?var, ?obj).`
2. `VarPointsTo(?to, ?obj) <- Assign(?from, ?to), VarPointsTo(?from, ?obj).`
datalog: declarative mutual recursion

<table>
<thead>
<tr>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = new A();</td>
</tr>
<tr>
<td>b = new B();</td>
</tr>
<tr>
<td>c = new C();</td>
</tr>
<tr>
<td>a = b;</td>
</tr>
<tr>
<td>b = a;</td>
</tr>
<tr>
<td>c = b;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AssignObjectAllocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VarPointsTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>

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

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

\[
\begin{align*}
\text{a} &= \text{new A();} \\
\text{b} &= \text{new B();} \\
\text{c} &= \text{new C();} \\
\text{a} &= \text{b;} \\
\text{b} &= \text{a;} \\
\text{c} &= \text{b;} \\
\end{align*}
\]

AssignObjectAllocation

\[
\begin{align*}
\text{a} &= \text{new A()} \\
\text{b} &= \text{new B()} \\
\text{c} &= \text{new C()} \\
\end{align*}
\]

Assign

\[
\begin{align*}
\text{b} &= \text{a} \\
\text{a} &= \text{b} \\
\text{b} &= \text{c} \\
\end{align*}
\]

VarPointsTo

\[
\begin{align*}
\text{VarPointsTo(?var, ?obj) <- AssignObjectAllocation(?var, ?obj).} \\
\text{VarPointsTo(?to, ?obj) <- Assign(?from, ?to), VarPointsTo(?from, ?obj).} \\
\end{align*}
\]
datalog: properties

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
example 1: introducing fields

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

VarPointsTo(?to, ?obj) <-
    Assign(?from, ?to),
    VarPointsTo(?from, ?obj).
Example 1: introducing fields

\[
\text{VarPointsTo}(\text{?var}, \text{?obj}) \leftarrow \\
\quad \text{AssignObjectAllocation}(\text{?var}, \text{?obj}).
\]

\[
\text{VarPointsTo}(\text{?to}, \text{?obj}) \leftarrow \\
\quad \text{Assign}(\text{?from}, \text{?to}), \\
\quad \text{VarPointsTo}(\text{?from}, \text{?obj}).
\]

\[
\text{FieldPointsTo}(\text{?baseobj}, \text{?field}, \text{?obj}) \leftarrow \\
\quad \text{StoreField}(\text{?from}, \text{?base}, \text{?field}), \\
\quad \text{VarPointsTo}(\text{?base}, \text{?baseobj}), \\
\quad \text{VarPointsTo}(\text{?from}, \text{?obj}).
\]

\[
\text{VarPointsTo}(\text{?to}, \text{?obj}) \leftarrow \\
\quad \text{LoadField}(\text{?base}, \text{?field}, \text{?to}), \\
\quad \text{VarPointsTo}(\text{?base}, \text{?baseobj}), \\
\quad \text{FieldPointsTo}(\text{?baseobj}, \text{?field}, \text{?obj}).
\]
example 1: introducing fields

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).

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

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

?base . ?field = ?from
VarPointsTo(?var, ?obj) <-
   AssignObjectAllocation(?var, ?obj).

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

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

field ?field of object ?baseobj may point to object ?obj
example 1: introducing fields

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(?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).

\[?to = ?base . ?field\]
example 1: introducing fields

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
example 1: introducing fields

VarPointsTo(?var, ?obj) ←
   AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) ←
   Assign(?from, ?to),
   VarPointsTo(?from, ?obj).

FieldPointsTo(?baseobj, ?field, ?obj) ←
   StoreField(?from, ?base, ?field),
   VarPointsTo(?baseobj, ?baseobj),
   VarPointsTo(?from, ?obj).

VarPointsTo(?to, ?obj) ←
   LoadField(?base, ?field, ?to),
   VarPointsTo(?baseobj, ?baseobj),
?to = ?base . ?field
example 1: introducing fields

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).

\[\text{?to} = \text{?base} \cdot \text{?field}\]
example 1: introducing fields

```
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

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

method invocations: caught exceptions

```java
void f() {
    try {
        g();
    } catch(E e) {..}
}
```
example 2: precise exception analysis

method invocations: propagated exceptions

\[ \text{ThrowPointsTo}(?\text{caller}, ?\text{obj}) \leftarrow \]

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

method invocations: caught exceptions

\[ \text{VarPointsTo}(?\text{param}, ?\text{obj}) \leftarrow \]

```java
void f() {
    try {
        g();
    } catch(E e) {..}
}
```
example 2: precise exception analysis

method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}),
\]

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

method invocations: caught exceptions

\[
\text{VarPointsTo}(\text{?param}, \text{?obj}) \leftarrow \\
\]

```java
void f() {
   try {
      g();
   } catch(E e) {
     ...
   }
}
```
method invocations: propagated exceptions

ThrowPointsTo(\text{?caller}, \text{?obj}) \leftarrow
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}),
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}),

\begin{verbatim}
void f() {
  --g();
}
\end{verbatim}

method invocations: caught exceptions

VarPointsTo(\text{?param}, \text{?obj}) \leftarrow

\begin{verbatim}
void f() {
  try {
    g();
  } catch(E e) {..}
}
\end{verbatim}
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) { .. }
}
example 2: precise exception analysis

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],

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) { .. }
}
example 2: precise exception analysis

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

```
void f() {
    try {
        g();
    } catch(E e) {..}
}
```
example 2: precise exception analysis

**method invocations: propagated exceptions**

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

**method invocations: caught exceptions**

```
VarPointsTo(?param, ?obj) <-
   CallGraphEdge(?invocation, ?tomethod),
   ThrowPointsTo(?tomethod, ?obj),
   Type[?obj] = ?objtype,
   ExceptionHandler[?objtype, ?invocation] = ?handler,
```
example 2: precise exception analysis

method invocations: propagated exceptions

\[
\text{ThrowPointsTo}(\text{?caller}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}), \\
\text{Type}[\text{?obj}] = \text{?objtype}, \\
\text{not exists ExceptionHandler}[\text{?objtype, ?invocation}], \\
\text{Method}[\text{?invocation}] = \text{?caller}.
\]

void \text{f}() \{
\text{g}(); 
\}

method invocations: caught exceptions

\[
\text{VarPointsTo}(\text{?param}, \text{?obj}) \leftarrow \\
\text{CallGraphEdge}(\text{?invocation}, \text{?tomethod}), \\
\text{ThrowPointsTo}(\text{?tomethod}, \text{?obj}), \\
\text{Type}[\text{?obj}] = \text{?objtype}, \\
\text{ExceptionHandler}[\text{?objtype, ?invocation}] = \text{?handler}, \\
\text{ExceptionHandler:FormalParam}[\text{?handler}] = \text{?param}.
\]

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

again, new mutually recursive dependencies do not change existing rules
from algorithms to specification and back
from algorithms to specification and back
from specification to algorithms
from specification to algorithms

benchmark
benchmark: 1-call-site-sensitive+heap

full order of magnitude faster!

comparable: demonstrated equivalence of results!
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: naive evaluation

```
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
```
datalog: naive evaluation
datalog: naive evaluation
datalog: semi-naive evaluation
Datalog: semi-naive evaluation

- **VarPointsTo**
- **Assign**

No repeated iteration over computed results
datalog: semi-naive evaluation

- VarPointsTo
- Assign

no repeated iteration over computed results
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
relations as multidimensional indexes

- VarPointsTo

<table>
<thead>
<tr>
<th>a</th>
<th>new A()</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
<tr>
<td>a</td>
<td>new B()</td>
</tr>
<tr>
<td>b</td>
<td>new A()</td>
</tr>
<tr>
<td>c</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new A()</td>
</tr>
</tbody>
</table>
relations as multidimensional indexes

- **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)
```
relations as multidimensional indexes

VarPointsTo

<table>
<thead>
<tr>
<th>a</th>
<th>new A()</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new C()</td>
</tr>
<tr>
<td>a</td>
<td>new B()</td>
</tr>
<tr>
<td>b</td>
<td>new A()</td>
</tr>
<tr>
<td>c</td>
<td>new B()</td>
</tr>
<tr>
<td>c</td>
<td>new A()</td>
</tr>
</tbody>
</table>

index organized by reverse argument order

VarPointsTo(?var, ?obj)

new A(), b  new B(), a

new A(), a  new A(), c  new B(), b  new B(), c  new C(), c
relations as multidimensional indexes

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)

- a, new B()
- b, new B()
- a, new A()
- b, new A()
- c, new A()
- c, new B()
- c, new C()
relations as multidimensional indexes

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)
relations as multidimensional indexes

<table>
<thead>
<tr>
<th>VarPointsTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>new A()</td>
</tr>
<tr>
<td>new B()</td>
</tr>
<tr>
<td>new C()</td>
</tr>
<tr>
<td>new B()</td>
</tr>
<tr>
<td>new A()</td>
</tr>
<tr>
<td>new B()</td>
</tr>
<tr>
<td>new A()</td>
</tr>
</tbody>
</table>

index organized by reverse argument order

\[ \text{VarPointsTo}(\text{?var, ?obj}) \]

\[ \text{VarPointsTo}(\text{?obj, ?var}) \]

```
a, new B()   b, new B()
a, new A()   b, new A()   c, new A()   c, new B()   c, new C()
```
relations as multidimensional indexes

index organized by reverse argument order

VarPointsTo(?var, ?obj)

VarPointsTo(?obj, ?var)

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

a, new B() b, new B()
a, new A() b, new A() c, new A() c, new B() c, new C()
relations as multidimensional indexes

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)

a, new B()  b, new B()

a, new A()  b, new A()  c, new A()  c, new B()  c, new C()
relations as multidimensional indexes

**VarPointsTo**

<table>
<thead>
<tr>
<th>new A()</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>new B()</td>
<td>b</td>
</tr>
<tr>
<td>new C()</td>
<td>c</td>
</tr>
<tr>
<td>new B()</td>
<td>a</td>
</tr>
<tr>
<td>new A()</td>
<td>b</td>
</tr>
<tr>
<td>new B()</td>
<td>c</td>
</tr>
<tr>
<td>new A()</td>
<td>c</td>
</tr>
</tbody>
</table>

**index organized by reverse argument order**

VarPointsTo(?var, ?obj)

VarPointsTo(?obj, ?var)

```
new A() a
new B() b
new C() c
new B() a
new A() b
new B() c
new A() c
```
relations as multidimensional indexes

<table>
<thead>
<tr>
<th>VarPointsTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>new A()</td>
</tr>
<tr>
<td>new B()</td>
</tr>
<tr>
<td>new C()</td>
</tr>
<tr>
<td>new B()</td>
</tr>
<tr>
<td>new A()</td>
</tr>
<tr>
<td>new B()</td>
</tr>
<tr>
<td>new A()</td>
</tr>
</tbody>
</table>

index organized by reverse argument order

- VarPointsTo(?var, ?obj)
- VarPointsTo(?obj, ?var)

- a, new B()
- b, new B()

  - a, new A()
  - b, new A()
  - c, new A()
  - c, new B()
  - c, new C()
from specification to algorithms

heuristics for searching the algorithm space
optimalization approach: heuristics

datalog specification → datalog engine

inefficient?

traditional design of single algorithm major human effort!
heuristics for searching algorithm space

always use index efficiently
always use index efficiently

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

- $\Delta \text{VarPointsTo}(\text{?from}, \text{?obj}) \bowtie \text{Assign(}\text{?from}, \text{?to})$
- $\Delta \text{VarPointsTo}(\text{?from}, \text{?obj}) \bowtie \text{Assign(}\text{?to}, \text{?from})$
<table>
<thead>
<tr>
<th><strong>heuristics for searching algorithm space</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>always use index efficiently</td>
</tr>
<tr>
<td>never iterate over full views</td>
</tr>
</tbody>
</table>
heuristics for searching algorithm space

- always use index efficiently
- never iterate over full views

- \( \Delta \text{Assign}(\text{?to}, \text{?from}) \iff \text{VarPointsTo}(\text{?from}, \text{?obj}) \)
Always use index efficiently

Never iterate over full views

- $\Delta \text{Assign}(\text{?to, ?from}) \bowtie \text{VarPointsTo(} \text{?from, ?obj)}$
- $\text{VarPointsTo(} \text{?from, ?obj)} \bowtie \Delta \text{Assign(} \text{?to, ?from)}$
heuristics for searching algorithm space

always use index efficiently

never iterate over full views

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

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

• \( \Delta \text{Assign}(\text{?to, ?from}) \bowtie \text{VarPointsTo}(\text{?obj, ?from}) \)
<table>
<thead>
<tr>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always use index efficiently</td>
</tr>
<tr>
<td>Never iterate over full views</td>
</tr>
<tr>
<td>Never iterate over big input relations</td>
</tr>
</tbody>
</table>
heuristics for searching algorithm space

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

ΔFieldPointsTo(?baseobj, ?field, ?obj) <-
    StoreField(?from, ?base, ?field),
    ΔVarPointsTo(?baseobj, ?base),
    VarPointsTo(?obj, ?from).

ΔFieldPointsTo(?baseobj, ?field, ?obj) <-
    StoreField(?from, ?base, ?field),
    VarPointsTo(?baseobj, ?base),
    ΔVarPointsTo(?obj, ?from).
specification

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

join orderings

\[ \Delta \text{VarPointsTo}(?\text{baseobj}, ?\text{base}) \]
\[ \bowtie \text{StoreField}(?\text{from}, ?\text{base}, ?\text{field}) \]
\[ \bowtie \text{VarPointsTo}(?\text{obj}, ?\text{from}) \]

\[ \Delta \text{VarPointsTo}(?\text{obj}, ?\text{from}) \]
\[ \bowtie \text{StoreField}(?\text{from}, ?\text{base}, ?\text{field}) \]
\[ \bowtie \text{VarPointsTo}(?\text{baseobj}, ?\text{base}) \]
specification

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

join orderings

△VarPointsTo(?baseobj, ?base)
  ⬤ StoreField(?from, ?base, ?field)
  ⬤ VarPointsTo(?obj, ?from)

△VarPointsTo(?obj, ?from)
  ⬤ StoreField(?from, ?base, ?field)
  ⬤ VarPointsTo(?baseobj, ?base)

there is no efficient variable ordering!
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)

there is no efficient variable ordering!
folding transformation: example

specification

\[ \text{FieldPointsTo}(\text{?baseobj}, \text{?field}, \text{?obj}) \leftarrow \]
\[ \text{StoreField}(\text{?from}, \text{?base}, \text{?field}), \]
\[ \text{VarPointsTo}(\text{?baseobj}, \text{?base}), \]
\[ \text{VarPointsTo}(\text{?obj}, \text{?from}). \]

join orderings

\[ \Delta \text{VarPointsTo}(\text{?baseobj}, \text{?base}) \]
\[ \bowtie \text{StoreField}(\text{?from}, \text{?base}, \text{?field}) \]
\[ \bowtie \text{VarPointsTo}(\text{?obj}, \text{?from}) \]

\[ \Delta \text{VarPointsTo}(\text{?obj}, \text{?from}) \]
\[ \bowtie \text{StoreField}(\text{?from}, \text{?base}, \text{?field}) \]
\[ \bowtie \text{VarPointsTo}(\text{?baseobj}, \text{?base}) \]

there is no efficient variable ordering!
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).
folding transformation: example

**specification**

\[
\text{FieldPointsTo(?baseobj, ?field, ?obj) <- StoreField(?from, ?base, ?field), VarPointsTo(?baseobj, ?base), VarPointsTo(?obj, ?from).}
\]

**optimized rules**

\[
\text{FieldPointsTo(?obj, ?field, ?baseobj) <- StoreObjectField(?baseobj, ?field, ?from), VarPointsTo(?obj, ?from).}
\]

\[
\text{StoreObjectField(?baseobj, ?field, ?from) <- StoreField(?from, ?field, ?base), VarPointsTo(?baseobj, ?base).}
\]
algorithm space is gigantic

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

1. Semi-sparse flow-sensitive pointer analysis
   Publisher: ACM
   Full text available: PDF (246.86 KB)
   Additional Information: full citation, abstract, references, index terms

   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
   Publisher: ACM
   Full text available: PDF (824.64 KB)
   Additional Information: full citation, abstract, references, index terms

   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
   Publisher: ACM
   Full text available: PDF (217.56 KB)

   Cloning-based context-sensitive pointer alias analysis using binary decision diagrams
algorithm space is gigantic

database community:
~ index selection
~ materialized view selection
from algorithms to specification
from specification to algorithms
fundamental new approach to pointer analysis

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!
future work

pointer analysis

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

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
future work

**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!
my research

**extensible and domain-specific languages**

embedding domain-specific languages  

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]