Points-to Analysis using BDDs

Marc Berndl, Ondřej Lhoták, Feng Qian, Laurie Hendren and Navindra Umanee
McGill University

Presented by Bruno Dufour
dufour@cs.rutgers.edu
Rutgers University DCS
Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions
Background – Points-to Analysis

- **Goal:** Given a (reference) variable \( v \), find the set of objects to which \( v \) may point at runtime.
  - For each \( v \), keep a set of possible objects (points-to set).
Background – Points-to Analysis

- **Goal:** Given a (reference) variable $v$, find the set of objects to which $v$ may point at runtime.
  - For each $v$, keep a set of possible objects (points-to set).

- **Problems**
Background – Points-to Analysis

- **Goal:** Given a (reference) variable $v$, find the set of objects to which $v$ may point at runtime.
  - For each $v$, keep a set of possible objects (points-to set).

- **Problems**
  - Large points-to sets
Background – Points-to Analysis

- **Goal:** Given a (reference) variable \( v \), find the set of objects to which \( v \) may point at runtime.
  - For each \( v \), keep a set of possible objects (points-to set).

- **Problems**
  - Large points-to sets
  - Large number of points-to sets
**Background – Points-to Analysis**

- **Goal:** Given a (reference) variable $v$, find the set of objects to which $v$ may point at runtime.
  - For each $v$, keep a set of possible objects (points-to set).

- **Problems**
  - Large points-to sets
    - Find efficient set representations
  - Large number of points-to sets
    - Collapse equivalent variables
Points-to Example Code

X: O a = new O();
Y: O b = new O();
Z: O c = new O();
    a = b;
    b = a;
    c = b;

Points-to set: { }
Points-to Example Code

X: O a = new O();
Y: O b = new O();
Z: O c = new O();
    a = b;
    b = a;
    c = b;

Points-to set: { (a,X) (b,Y) (c,Z) }
Points-to Example Code

X: O a = new O();
Y: O b = new O();
Z: O c = new O();
    a = b;
    b = a;
    c = b;

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) }
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
    a = b;
    b = a;
    c = b;

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) (b,X) }
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
   a = b;
   b = a;
   c = b;

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) (b,X) (c,X) (c,Y) }
Background – BDDs

- **Binary Decision Diagrams (BDDs)** are data structures that are used to represent large sets with similarities.
- Introduced in [Bryant86]
- Applications in model checking
- Essentially single-root DAGs with out-degree two for each non-leaf node
- Some possible interpretations:
  - Set of binary strings
  - Representation of a boolean function
    \[ f : \{0, 1\}^n \rightarrow \{0, 1\} \]
  - Finite automaton with accepting state \( \boxed{1} \) and rejecting state \( \boxed{0} \) taking binary strings as input
Example BDD

L = {000, 001, 011, 100}
Example BDD

L = \{000,001,011,100\}
Reducing a BDD

```
A
   0 1
B 0 1
   1 0
D 1
   0
E 1
F 0
G 1
```
Reducing a BDD
Types of BDDs

- Ordered BDDs (OBDDs)
  - variables are *ordered*
  - Each variable appears only in one level of the BDD

- Reduced Ordered BDDs (ROBDDs)
  - OBDDs in reduced form
  - Consistent ordering of nodes ensures uniqueness
BDD Operations

- BDDs support common set operations ($\cap$, $\cup$, $\ldots$)
- Existential quantification: $S = \{ a | \exists b. (a, b) \in X \}$
- Relational product: $\{(a, c) | \exists b. (a, b) \in X \land (b, c) \in Y \}$
  ($\cap$ + existential quantification)

Replace: bit reordering

Operation cost proportional to # of nodes in BDD
- To minimize cost, keep BDDs in reduced form
- Implicitly refer to ROBDDs simply as BDDs
Bit Ordering

- Ordering of bits in BDDs is arbitrary
  - Any permutation is valid
  - Some permutations lead to smaller (reduced) BDDs
BuDDy

- Publicly available BDD package
  - Written in C
  - Supports dynamic variable reordering
  - Features node garbage collection
  - Groups bits into **domains**
Outline

■ Background
  ■ Points-to (reference) analysis
  ■ BDDs
■ Points-to algorithm using BDDs
■ Performance tuning
■ Experimental results
■ Applications
■ Conclusions
Points-to Algorithm

- Java extension of Andersen’s analysis
  - Flow-insensitive
  - Context-insensitive
  - Subset-based constraints
- All constraints generated ahead of time to separate constraint generation from solver
  - Call graph for constraint generation obtained using CHA
Points-to Algorithm

- 4 types of statements
  - Allocation: \( a : l := \text{new } C \)
  - Simple assignment: \( l_2 := l_1 \)
  - Field store: \( q.f := l \)
  - Field load: \( l := p.f \)

- 2 relations
  - Points-to: \( pt \)
    - \( pt(l) \) denotes the set of objects that \( l \) may point to
  - Assignment-edge: \( \rightarrow \)
    - \( a \rightarrow b \) indicates that \( b \) may point to any object that \( a \) may point to
Inference Rules

- Simple assignments

\[ l_1 \rightarrow l_2 \quad o \in pt(l_1) \]
\[ o \in pt(l_2) \]

- Field stores

\[ o_2 \in pt(l) \quad l \rightarrow q.f \quad o_1 \in pt(q) \]
\[ o_2 \in pt(o_1.f) \]

- Field loads

\[ p.f \rightarrow l \quad o_1 \in pt(p) \quad o_2 \in pt(o_1.f) \]
\[ o_2 \in pt(l) \]
PTA Solver Algorithm

init
repeat
  repeat
    Process simple assignments
  until no change
  Process field stores
  Process field loads
until no change
Recall:

X: O a = new O();
Y: O b = new O();
Z: O c = new O();
    a = b;
    b = a;
    c = b;

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) (b,X) (c,X) (c,Y) }
Encoding the Example Points-to Set as a BDD

- Points-to set contains pairs of the form \((v, h)\) where \(v\) is a variable and \(h\) is a heap location.

- Need two domains:
  - \(V = \{a, b, c\}\)
  - \(H = \{X, Y, Z\}\)

- Points-to set \(P \subseteq V \times H\)

- Need \(\lceil \log_2(|V|) \rceil = 2\) bits for each element of \(V\)
  - Represent elements of \(V\) as binary string \(v_1v_0\)

<table>
<thead>
<tr>
<th>(v_1v_0)</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0</td>
<td>0 1</td>
<td>1 0</td>
</tr>
</tbody>
</table>

- Idem for \(H\)
(v, h) ∈ P ⇔ v₁v₀h₁h₀ is mapped to 1 in the BDD
Encoding the Example Points-to Set as a BDD (2)

- \((v, h) \in P \iff v_1v_0h_1h_0\) is mapped to 1 in the BDD

\((a, X) \in P\)
Encoding the Example Points-to Set as a BDD (2)

- \((v, h) \in P \iff v_0 v_1 h_0 h_1\) is mapped to \(1\) in the BDD

\[(b, Z) \notin P\]
BDD Representation

Points-to Analysis using BDDs – p. 22/76
BDD Reduction (2)
Reduced BDD Representation
General BDD Implementation

- Need 5 domains
  - $V_1, V_2$: Reference variables
    - Need two domains to represent pairs in $V \times V$
  - $H_1, H_2$: Allocation sites
    - Need two domains to represent the points-to set of object fields
  - $FD$: field signatures
Propagating points-to sets

X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;

(a,X) (b → a)
(b,Y) (a → b)
(c,Z) (b → c)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td>a b c</td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Propagating points-to sets

X: \( a = \text{new } O(); \)
Y: \( b = \text{new } O(); \)
Z: \( c = \text{new } O(); \)

\[
\begin{align*}
(a,X) & \quad (b \rightarrow a) \\
(b,Y) & \quad (a \rightarrow b) \\
(c,Z) & \quad (b \rightarrow c) \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>a b c</td>
<td>a b c</td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Propagating points-to sets

\[ X: \quad a = \text{new} \quad 0(); \quad a = b; \]
\[ Y: \quad b = \text{new} \quad 0(); \quad b = a; \]
\[ Z: \quad c = \text{new} \quad 0(); \quad c = b; \]

\[
\begin{align*}
(a,X) & \quad (b \rightarrow a) \\
(b,Y) & \quad (a \rightarrow b) \\
(c,Z) & \quad (b \rightarrow c)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td>a b c</td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Propagating points-to sets

X: \( a = \text{new } O(); \)  
Y: \( b = \text{new } O(); \)  
Z: \( c = \text{new } O(); \)

\( a = b; \)  
\( b = a; \)  
\( c = b; \)

\((a,X)\)  
\((b,Y)\)  
\((c,Z)\)

\((b \rightarrow a)\)  
\((a \rightarrow b)\)  
\((b \rightarrow c)\)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td>a b c</td>
<td>b</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Propagating points-to sets

X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;

(a,X)  (b → a)
(b,Y)  (a → b)
(c,Z)  (b → c)

Domains | Points-to | Edges | New points-to
---------|-----------|-------|-----------------|
V1       | a b c     | b a b | b a b           |
V2       | a b c     | a b c | b               |
H1       | X Y Z     | X     | X               |
Propagating points-to sets

X: a = new O();
Y: b = new O();
Z: c = new O();

a = b;
b = a;
c = b;

(a,X) (b → a)
(b,Y) (a → b)
(c,Z) (b → c)

Domains | Points-to | Edges | New points-to
--- | --- | --- | ---
V1 | a b c | b a b | b a c
V2 | a b c | a b c | b a c
H1 | X Y Z | X Y Y |

relprod
### Propagating points-to sets

```java
X: a = new O();  // X: a = b;
Y: b = new O();  // b = a;
Z: c = new O();  // c = b;
```

- (a, X)
- (b, Y)
- (c, Z)
- (b \rightarrow a)
- (a \rightarrow b)
- (b \rightarrow c)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td>a b c</td>
</tr>
<tr>
<td>V2</td>
<td>a b c</td>
<td>a b c</td>
<td>b a c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td>X Y Y</td>
<td>X Y Y</td>
</tr>
</tbody>
</table>
Propagating points-to sets

X: \( a = \text{new} \ O(); \)
Y: \( b = \text{new} \ O(); \)
Z: \( c = \text{new} \ O(); \)

\( a = b; \)
\( b = a; \)
\( c = b; \)

\[(a,X) \quad (b \rightarrow a) \]
\[(b,Y) \quad (a \rightarrow b) \]
\[(c,Z) \quad (b \rightarrow c) \]

Domains | Points-to | Edges | New points-to
---|---|---|---
V1 | a b c | b a b | a b c
V2 | b a b | a b c | b a c
H1 | X Y Z | X Y Y | X Y Y
## Propagating points-to sets

\[
\begin{align*}
X: & \quad a = \text{new } O(); \\
Y: & \quad b = \text{new } O(); \\
Z: & \quad c = \text{new } O(); \\
& \quad a = b; \\
& \quad b = a; \\
& \quad c = b;
\end{align*}
\]

\[
\begin{align*}
(a, X) & \quad \rightarrow (b \rightarrow a) \\
(b, Y) & \quad \rightarrow (a \rightarrow b) \\
(c, Z) & \quad \rightarrow (b \rightarrow c)
\end{align*}
\]

### Domains

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td>b a c</td>
</tr>
<tr>
<td>V2</td>
<td>a b c</td>
<td>a b c</td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td>X Y Y</td>
<td></td>
</tr>
</tbody>
</table>

### Replace

- Replace (b, Y) with (a, X) in the Edges column.
- The New points-to set becomes (b, a, c).
Propagating points-to sets

X: a = new O();
Y: b = new O();
Z: c = new O();

(a,X) (b → a)
(b,Y) (a → b)
(c,Z) (b → c)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c</td>
<td>b a b</td>
<td>b a c</td>
</tr>
<tr>
<td>V2</td>
<td>a b c</td>
<td>a b c</td>
<td>b a c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z</td>
<td></td>
<td>X Y Y</td>
</tr>
</tbody>
</table>
Propagating points-to sets

X: a = new O();
Y: b = new O();
Z: c = new O();

a = b;
b = a;
c = b;

(a,X) (b → a)
(b,Y) (a → b)
(c,Z) (b → c)

Domains | Points-to | Edges | New points-to
---|---|---|---
V1 | a b c | b a b | b a c
V2 | a b c | a b c | b a c
H1 | X Y Z | X Y Y | X Y Y
### Propagating points-to sets

```plaintext
X: a = new O();
Y: b = new O();
Z: c = new O();
```

(a,X)\(\Rightarrow\) (b → a)
(b,Y)\(\Rightarrow\) (a → b)
(c,Z)\(\Rightarrow\) (b → c)

**Domains** | **Points-to** | **Edges** | **New**
--- | --- | --- | ---
V1 | a b c b a c | b a b | 
V2 | a b c | a b c | 
H1 | X Y Z X Y Y | | 
````
## Propagating points-to sets

X: \( a = \text{new } O(); \)  
Y: \( b = \text{new } O(); \)  
Z: \( c = \text{new } O(); \)

\[ a = b; \]
\[ b = a; \]
\[ c = b; \]

\[(a, X)\] \quad \[(b \rightarrow a)\]
\[(b, Y)\] \quad \[(a \rightarrow b)\]
\[(c, Z)\] \quad \[(b \rightarrow c)\]

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>a b c a b c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y</td>
<td>a b c</td>
<td></td>
</tr>
</tbody>
</table>
Propagating points-to sets

\[
\begin{align*}
X: & \quad a = \text{new } O(); \\
Y: & \quad b = \text{new } O(); \\
Z: & \quad c = \text{new } O(); \\
\end{align*}
\]

\[
\begin{align*}
& (a, X) \\
& (b, Y) \\
& (c, Z)
\end{align*}
\]

\[
\begin{align*}
\text{Domains} & \quad \text{Points-to} & \quad \text{Edges} & \quad \text{New} \\
V1 & \quad a & b & c & b & a & c & b & a & b \\
V2 & \quad a & b & c \\
H1 & \quad X & Y & Z & X & Y & Y & a & b & c
\end{align*}
\]
### Propagating points-to sets

X: \( a = \text{new} \ O() \);  
Y: \( b = \text{new} \ O() \);  
Z: \( c = \text{new} \ O() \);  

\( a = b; \)  
\( b = a; \)  
\( c = b; \)

\( (a,X) \)  
\( (b,Y) \)  
\( (c,Z) \)

\( (b \rightarrow a) \)  
\( (a \rightarrow b) \)  
\( (b \rightarrow c) \)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td></td>
<td>a b c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
### Propagating points-to sets

X: \( a = \text{new } O(); \)
Y: \( b = \text{new } O(); \)
Z: \( c = \text{new } O(); \)

\( a = b; \)
\( b = a; \)
\( c = b; \)

\[(a,X)(b,Y)(c,Z)\]

\[(b \to a)(a \to b)(b \to c)\]

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>a b c</td>
<td>a b c</td>
<td>c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Propagating points-to sets

\[
X: \quad a = \text{new } O(); \quad \quad a = b; \\
Y: \quad b = \text{new } O(); \quad \quad b = a; \\
Z: \quad c = \text{new } O(); \quad \quad c = b; \\
\]

\[
(a,X) \quad (b \rightarrow a) \quad \text{replace} \\
(b,Y) \quad (a \rightarrow b) \\
(c,Z) \quad (b \rightarrow c) \\
\]

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c</td>
<td>b a b</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>a b c a b c</td>
<td>a b c</td>
<td>c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Propagating points-to sets

X: \( a = \text{new } O(); \)  
Y: \( b = \text{new } O(); \)  
Z: \( c = \text{new } O(); \)  
\( a = b; \)  
\( b = a; \)  
\( c = b; \)

\[(a,X) \quad (b \to a) \quad \text{replace} \]
\[(b,Y) \quad (a \to b) \]
\[(c,Z) \quad (b \to c) \]

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c</td>
<td>b a b</td>
<td>c</td>
</tr>
<tr>
<td>V2</td>
<td>a b c a b c</td>
<td>a b c</td>
<td>c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Propagating points-to sets

X: \( a = \text{new} \ 0() \);  
Y: \( b = \text{new} \ 0() \);  
Z: \( c = \text{new} \ 0() \);

\( a = b; \)  
\( b = a; \)  
\( c = b; \)

\((a,X)\)  
\((b,Y)\)  
\((c,Z)\)

\((b \rightarrow a)\)  
\((a \rightarrow b)\)  
\((b \rightarrow c)\)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>( a \ b \ c \ b \ a \ c )</td>
<td>( b \ a \ b )</td>
<td>( c )</td>
</tr>
<tr>
<td>V2</td>
<td>( a \ b \ c )</td>
<td>( a \ b \ c )</td>
<td>( c )</td>
</tr>
<tr>
<td>H1</td>
<td>( X \ Y \ Z \ X \ Y \ Y )</td>
<td>( X )</td>
<td></td>
</tr>
</tbody>
</table>
### Propagating points-to sets

X: \( a = \text{new } O() ; \)  
Y: \( b = \text{new } O() ; \)  
Z: \( c = \text{new } O() ; \)

\( a = b ; \)
\( b = a ; \)
\( c = b ; \)

\( (a,X) \)
\( (b,Y) \)
\( (c,Z) \)

(union)

- \( b \rightarrow a \)
- \( a \rightarrow b \)
- \( b \rightarrow c \)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c</td>
<td>b a b</td>
<td>c</td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td>a b c</td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Domains | Points-to | Edges | New
---------|-----------|-------|-----
V1      | a b c b a c | b a b  | c   |
V2      |           | a b c  |     |
H1      | X Y Z X Y Y |       | X   |
### Propagating points-to sets

X: \(a = \text{new } O();\)  
Y: \(b = \text{new } O();\)  
Z: \(c = \text{new } O();\)

\(a = b;\)  
\(b = a;\)  
\(c = b;\)

\((a, X)\)  
\((b, Y)\)  
\((c, Z)\)

\((b \to a)\)  
\((a \to b)\)  
\((b \to c)\)

#### Domains

<table>
<thead>
<tr>
<th>Points-to</th>
<th>Edges</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>a b c b a c c</td>
<td>b a b</td>
</tr>
<tr>
<td>V2</td>
<td>a b c</td>
<td>a b c</td>
</tr>
<tr>
<td>H1</td>
<td>X Y Z X Y Y Y X</td>
<td>X</td>
</tr>
</tbody>
</table>
Important Relations

- \( pointsTo \subseteq V_1 \times H_1 \)
  points-to relation for variables
  \((l \text{ points to } o)\)

- \( fieldPt \subseteq (H_1 \times FD) \times H_2 \)
  points-to relation for object fields
  \((o_1.f \text{ points to } o_2)\)

- \( edgeSet \subseteq V_1 \times V_2 \)
  simple assignments
  \((l_2 := l_1)\)

- \( stores \subseteq V_1 \times (V_2 \times FD) \)
  field stores
  \((l_2.f := l_1)\)

- \( loads \subseteq (V_1 \times FD) \times V_2 \)
  field loads
  \((l_2 := l_1.f)\)

- \( typeFilter \subseteq V_1 \times H_1 \)
Simple assignments ($l_2 := l_1$)

newPt1: $\begin{bmatrix} V2xH1 \end{bmatrix} = \text{relprod}(\text{edgeSet: } \begin{bmatrix} V1xV2 \end{bmatrix}, \text{pointsTo: } \begin{bmatrix} V1xH1 \end{bmatrix}, V1 )$;

newPt2: $\begin{bmatrix} V1xH1 \end{bmatrix} = \text{replace}(\text{newPt1: } \begin{bmatrix} V2xH1 \end{bmatrix}, \text{V2ToV1 } )$;

newPt3: $\begin{bmatrix} V1xH1 \end{bmatrix} = \text{isect} (\text{newPt2: } \begin{bmatrix} V1xH1 \end{bmatrix}, \text{typeFilter: } \begin{bmatrix} V1xH1 \end{bmatrix} )$;

pointsTo: $\begin{bmatrix} V1xH1 \end{bmatrix} = \text{union}(\text{pointsTo: } \begin{bmatrix} V1xH1 \end{bmatrix}, \text{newPt3: } \begin{bmatrix} V1xH1 \end{bmatrix} )$;
Field stores \( q.f := l \)

\[
\text{tmpRel1:} [(V2xFD)xH1] = \text{relprod} ( \text{stores:} [V1x(V2xFD)], \text{pointsTo:} [V1xH1], V1 );
\]

\[
\text{tmpRel2:} [(V1xFD)xH2] = \text{replace} ( \text{tmpRel1:} [(V2xFD)xH1], V2ToV1 \& H1ToH2 );
\]

\[
\text{fieldPt:} [(H1xFD)xH2] = \text{relprod} ( \text{tmpRel2:} [(V1xFD)xH2], \text{pointsTo:} [V1xH1], V1 );
\]
Field loads \( (l := p.f) \)

\[
\text{tmpRel3: } [(H1xFD)xV2] = \relprod( \text{loads: } [(V1xFD)xV2], \text{pointsTo: } [V1xH1], V1 );
\]

\[
\text{newPt4: } [V2xH2] = \relprod( \text{tmpRel3: } [(H1xFD)xV2], \text{fieldPt: } [(H1xFD)xH2], H1xFD );
\]

\[
\text{newPt5: } [V1xH1] = \replace( \text{newPt4: } [V2xH2], V2ToV1 \& H2ToH1 );
\]
Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions
Experimental Setup

- Subset-based constraints generated by SPARK for a field-sensitive analysis
- Call graph constructed using CHA
- Effect of native methods considered (inherited from SOOT)
- 2 kinds of sets of constraints
  - Simplified (s)
  - Non-simplified (ns)
- 2 strategies for handling declared types
  - Type filtering during analysis (t)
  - Type filtering at the end of analysis (nt)
Problem: Using the default configuration, the BDD solver cannot solve most real benchmarks.

Profiling reveals that \texttt{relprod} operation from the inner loop is the bottleneck.

Two factors to consider for performance:
- Relative domain ordering
- Variable interleaving within domains
Variable Interleaving Notation

- Let $FD, V1$ be domains such that $f_0, \ldots, f_n$ are the variables of $FD$ and $v_0, \ldots, v_n$ are the variables of domain $V1$.
  - $FDV1$ denotes the interleaving of variables, i.e. $f_0v_0f_1v_1\ldots f_nv_n$.
  - $FD\_V1$ denotes the concatenation of variables, i.e. $f_0f_1\ldots f_nv_0v_1\ldots v_n$.

- Variables are always order from most to least significant bit to exploit unused high bits.

- BuDDy’s default ordering is $FDV1V2H1H2$
Effect of Domain Arrangement on $relprod$

The diagram shows the execution time in seconds for different iteration numbers. The lines represent different domain arrangements:

- $fd_v1v2_h1_h2$
- $fd_h1_v1v2_h2$

The y-axis represents execution time (s), and the x-axis represents iteration number.
Effect of Interleaving Domains on relprod

![Graph showing the effect of interleaving domains on execution time.](image)
Effect of Interleaving Domains on `pointsTo`
Effect of Variable Ordering on Performance

- Default ordering is good for model checking, but much too slow for PTA
- Investigate other orderings
  - Focus on the domains used in the problematic `relprod` operation, i.e. $V_1, V_2, H_1$.
  - Reason about the impact of certain orderings on BDD size
    - Domains that feature a large amount of similarity between the sets could benefit from preventing interleaving.
    - It is easier to exploit similarities when present at the end of the variable sequence than at the beginning.
Effect of Different Orderings on Performance

![Graph showing effect of different orderings on performance]

- (V1V2H1)
- H1_(V1V2)
- (V1V2)_H1
- V1_V2_H1

Seconds

compress  javac  sablecc  jedit
Effect of Ordering on $\text{edgeSet}$

![Graph showing the effect of ordering on edgeSet. The graph plots the number of nodes against BDD level with three lines representing different orderings: V1_V2 (red solid line), V2_V1 (black dashed line), and (V1V2) (blue dotted line).]
Effect of Ordering on \textit{pointsTo}

![Graph showing effect of ordering on pointsTo](image)

- V1_H1
- H1_V1
- (V1H1)
**Observation:** The `relprod` operation propagates all points-to sets along all edges at every execution.

- Most sets have already been propagated in previous iterations
- `relprod` executes in time proportional to the # of nodes
- The inner `relprod` is very hot
**Performance Tuning – Incrementalization**

- **Observation:** The relprod operation propagates all points-to sets along all edges at every execution.
  - Most sets have already been propagated in previous iterations.
  - $\text{relprod}$ executes in time proportional to the # of nodes.
  - The inner relprod is very hot.
- Only propagate the new part of the points-to set.
  - new $\text{pointsTo}$ relation remains small.
  - relprod executes much faster.
Incremental BDD-PTA Algorithm

newPt1: \[ [V2 \times H1] = \text{relprod}( \text{edgeSet: } [V1 \times V2], \text{pointsTo: } [V1 \times H1], V1 ); \]

newPt2: \[ [V1 \times H1] = \text{replace}( \text{newPt1: } [V2 \times H1], V2ToV1 ); \]

pointsTo: [V1 \times H1] = \text{union}( \text{pointsTo: } [V1 \times H1], \text{newPt2: } [V1 \times H1] );
Performance Tuning – Incrementalization

newPt1: [V2xH1] =
    relprod( edgeSet: [V1xV2],
              newPoint:[V1xH1],
              V1 );

newPt2: [V1xH1] =
    replace( newPt1: [V2xH1],
             V2ToV1 );

newPoint:[V1xH1] =
    setminus( newPt2: [V1xH1],
              pointsTo:[V1xH1] );

pointsTo:[V1xH1] =
    union( pointsTo:[V1xH1],
           newPoint:[V1xH1] );
## Effect of Incrementalization on Performance

<table>
<thead>
<tr>
<th>benchmark</th>
<th>fd_V1V2_H1_2</th>
<th></th>
<th>FD_V1_V2_H1_H2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>non-inc</td>
<td>inc</td>
<td>non-inc</td>
<td>inc</td>
</tr>
<tr>
<td>compress (s/t)</td>
<td>20.63</td>
<td>11.72</td>
<td>19.07</td>
<td>9.80</td>
</tr>
<tr>
<td>compress (ns/t)</td>
<td>54.46</td>
<td>26.83</td>
<td>83.63</td>
<td>19.66</td>
</tr>
<tr>
<td>compress (ns/nt)</td>
<td>145.33</td>
<td>71.55</td>
<td>228.21</td>
<td>58.58</td>
</tr>
<tr>
<td>javac (s/t)</td>
<td>22.62</td>
<td>14.83</td>
<td>23.89</td>
<td>10.83</td>
</tr>
<tr>
<td>javac (ns/t)</td>
<td>62.35</td>
<td>30.55</td>
<td>103.52</td>
<td>23.14</td>
</tr>
<tr>
<td>javac (ns/nt)</td>
<td>166.66</td>
<td>80.04</td>
<td>285.65</td>
<td>65.46</td>
</tr>
<tr>
<td>sablecc-j (s/t)</td>
<td>21.90</td>
<td>14.00</td>
<td>23.10</td>
<td>10.60</td>
</tr>
<tr>
<td>sablecc-j (ns/t)</td>
<td>63.43</td>
<td>30.05</td>
<td>110.87</td>
<td>22.86</td>
</tr>
<tr>
<td>sablecc-j (ns/nt)</td>
<td>158.33</td>
<td>76.53</td>
<td>269.30</td>
<td>63.82</td>
</tr>
<tr>
<td>jedit (s/t)</td>
<td>35.92</td>
<td>20.11</td>
<td>35.43</td>
<td>15.60</td>
</tr>
<tr>
<td>jedit (ns/t)</td>
<td>112.47</td>
<td>47.53</td>
<td>357.97</td>
<td>35.29</td>
</tr>
<tr>
<td>jedit (ns/nt)</td>
<td>336.18</td>
<td>150.72</td>
<td>783.92</td>
<td>120.53</td>
</tr>
</tbody>
</table>
Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions
Overall Performance – Time

The graph illustrates the overall performance over time, comparing BDD and Spark. The x-axis represents the constraints in units of $10^3$, while the y-axis shows the seconds. The data points show that Spark generally outperforms BDD, especially as the number of constraints increases.
Overall Performance – Space

![Graph showing performance metrics for BDD and Spark](image-url)

- **BDD**
- **Spark**
Applications

- Manipulating sets makes it easy to express common problems:
  - “May/Must be aliased” analysis
  - Virtual method resolution (receiver types)
    - Inlining
    - Devirtualization
- Manipulating the solution as a BDD
  - improves performance
  - lowers development cost
Related Work

- A lot...
  - PTA
    - PTA algorithms
    - Efficient set representations
    - Equality-based constraints
    - ...
  - BDDs
    - Model checking
Conclusions

- BDDs are useful in the context of PTA
- It is possible to write efficient solvers using BDD libraries “out of the box”
- Finding a good bit ordering is necessary to obtain good performance