Next In Line, Please!

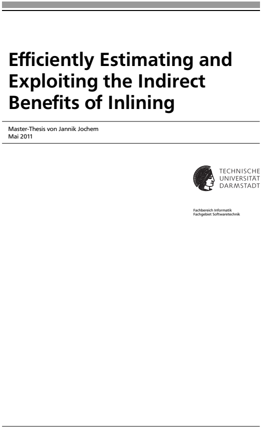
Exploiting the Indirect Benefits of Inlining by Accurately Predicting Further Inlining

TECHNISCHE UNIVERSITÄT DARMSTADT

November 3, 2011 | Andreas Sewe, Jannik Jochem, Mira Mezini | 1

Next In Line, Please!

Exploiting the Indirect Benefits of Inlining by Accurately Predicting Further Inlining



Efficiently Estimating and Exploiting the Indirect Benefits of Inlining

Master-Thesis von Jannik Jochem
Mai 2011

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Fachgebiet Softwaretechnik

Background: The Direct and Indirect Benefits of Inlining

Background: The Direct and Indirect Benefits of Inlining

Direct Benefits

- No stack frame creation
- No call/return overhead
- (Possibly) no dynamic dispatch

Background: The Direct and Indirect Benefits of Inlining

Direct Benefits

- No stack frame creation
- No call/return overhead
- (Possibly) no dynamic dispatch

Indirect Benefits

- Constant folding
- Elimination of type checks
- Elimination of null checks
- Elimination of array bounds checks
- Further, guardless Inlining

Background: The Direct and Indirect Benefits of Inlining

Direct Benefits

- No stack frame creation
- No call/return overhead
- (Possibly) no dynamic dispatch

Indirect Benefits

- Constant folding
- Elimination of type checks
- Elimination of null checks
- Elimination of array bounds checks
- Further, guardless Inlining

Background: When is Further, Guardless Inlining Possible?

TECHNISCHE UNIVERSITÄT DARMSTADT

Precise Arguments

```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}

class D extends C { ... }
```

Background: When is Further, Guardless Inlining Possible?

Precise Arguments

```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}

class D extends C { ... }
```

Extant Arguments

```
class A2 {
  void m(C c) {
    B b = ...
    // Argument exists before call
    // to A2.m(C).
    b.n(c);
  }
}
```

Background: When is Further, Guardless Inlining Possible? And When Obviously Not?

Precise Arguments

```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}
```

Extant Arguments

```
class A2 {
  void m(C c) {
    B b = ...
    // Argument exists before call
    // to A2.m(C).
    b.n(c);
  }
}
```

```
class D extends C { void n(C c) { this.f = c; } }
```

Background: The Inline Oracle of Jikes RVM

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Background: The Inline Oracle of Jikes RVM

1. Reject @NoInline or native methods.

Background: The Inline Oracle of Jikes RVM

1. Reject @NoInline or native methods.
2. Accept trivial callees.

Background: The Inline Oracle of Jikes RVM

1. Reject @NoInline or native methods.
2. Accept trivial callees.
3. Identify targets in dynamic call graph.

Background: The Inline Oracle of Jikes RVM

1. Reject @NoInline or native methods.
2. Accept trivial callees.
3. Identify targets in dynamic call graph.
4. For each dynamic target, estimate its size and decide whether to inline.

Background: The Inline Oracle of Jikes RVM

1. Reject @NoInline or native methods.
2. Accept trivial callees.
3. Identify targets in dynamic call graph.
4. For each dynamic target, estimate its size and decide whether to inline.
5. Choose appropriate guards.

Background: The Inline Oracle of Jikes RVM

1. Reject @NoInline or native methods.
2. Accept trivial callees.
3. Identify targets in dynamic call graph.
4. For each dynamic target, estimate its size and decide whether to inline.
5. Choose appropriate guards.

Problem: Jikes RVM's Assumed Size Reductions

	Reduction
Reference argument of precise type	15 %
Reference argument pre-exists method call	5 %
Non-null object constant	10 %
null constant	10 %
Integer constant	5 %
Array argument of precise type	5 %
No aastore check required	2 %

Problem: Jikes RVM's Assumed Size Reductions

	Reduction
Reference argument of precise type	15 %
Reference argument pre-exists method call	5 %
Non-null object constant	10 %
null constant	10 %
Integer constant	5 %
Array argument of precise type	5 %
No aastore check required	2 %

Proposed Solution: Award Size Reductions Only When Further Inlining Likely

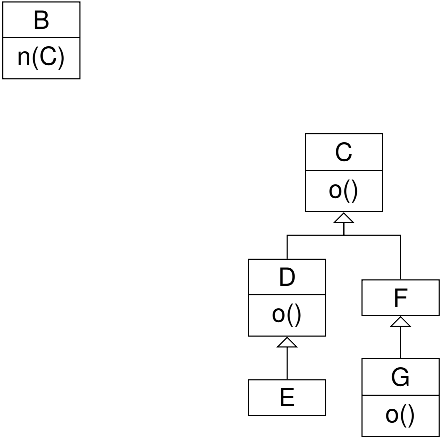
	Reduction
Reference argument of precise type	0 %
Reference argument pre-exists method call	0 %
Non-null object constant	10 %
null constant	10 %
Integer constant	5 %
Array argument of precise type	5 %
No aastore check required	2 %

Proposed Solution: How to Determine Likelihood of Further Inlining?

1. Reject @NoInline or native methods.
2. Accept trivial callees.
3. Identify targets in dynamic call graph.
4. For each dynamic target, estimate its size and decide whether to inline.
5. Choose appropriate guards.

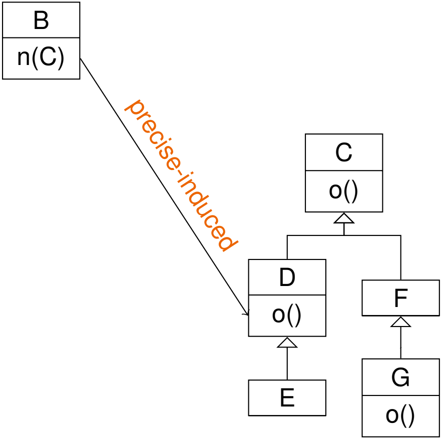
Proposed Solution: Identify Precise- and Extant-Induced Edges

```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}
```



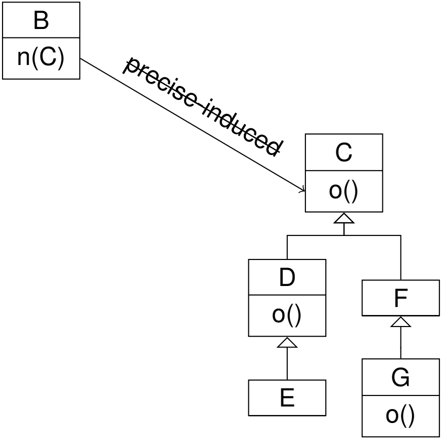
Proposed Solution: Identify Precise- and Extant-Induced Edges

```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}
```

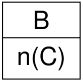


Proposed Solution: Identify Precise- and Extant-Induced Edges

```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}
```

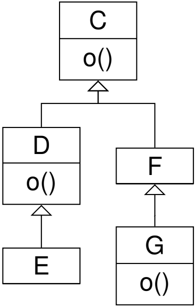


Proposed Solution: Identify Precise- and Extant-Induced Edges



```
class A1 {
  void m() {
    B b = ...
    C c = new D();
    // Precise type of argument is D.
    b.n(c);
  }
}
```

Extant-induced edges work similar.



Evaluation: The Prediction Problem of Further Inlining

Problem

Given the decision to inline B.n(C) into A.m(), predict whether further inlining of C.o() into B.n(C) will occur—and only then award a size reduction.

Evaluation: The Prediction Problem of Further Inlining

Problem

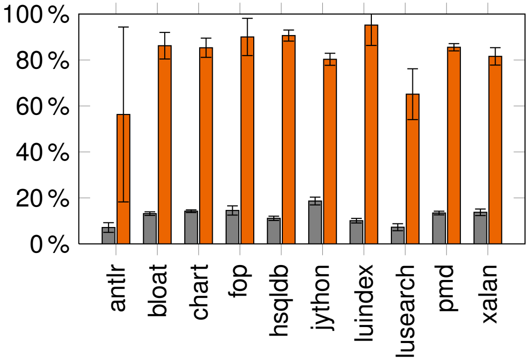
Given the decision to inline B.n(C) into A.m(), predict whether further inlining of C.o() into B.n(C) will occur—and only then award a size reduction.

Evaluation: The Prediction Problem of Further Inlining

Problem

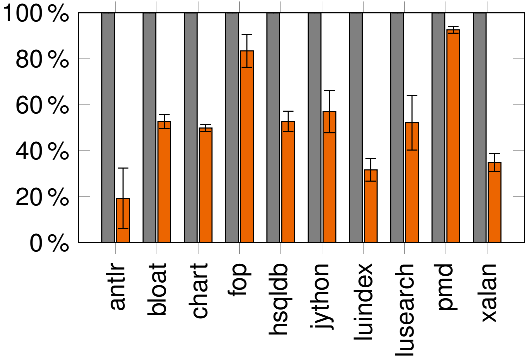
Given the decision to inline B.n(C) into A.m(), predict whether further inlining of C.o() into B.n(C) will occur—and only then award a size reduction.

Evaluation: Per-Decision Quality of Inlining Heuristics (Precision)



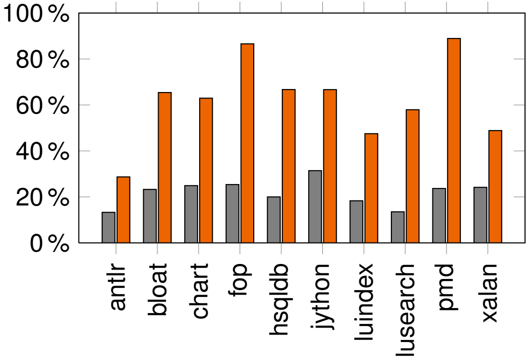
$$\frac{\text{true positives}}{\text{true positives + false positves}}$$

Evaluation: Per-Decision Quality of Inlining Heuristics (Recall)



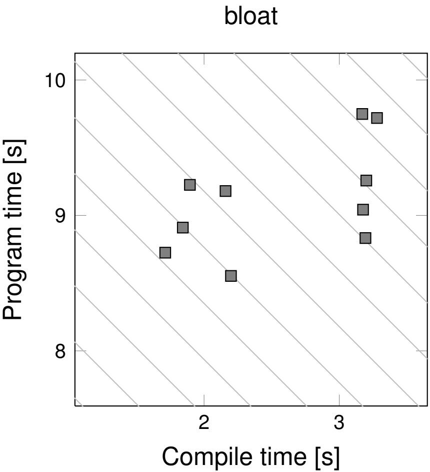
$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Evaluation: Per-Decision Quality of Inlining Heuristics (F1-Measure)



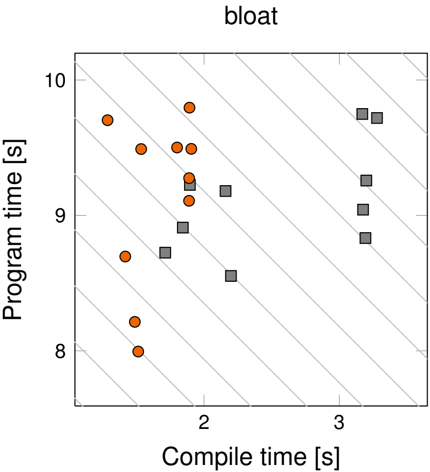
$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Evaluation: Performance Measurements with Replay-Compilation



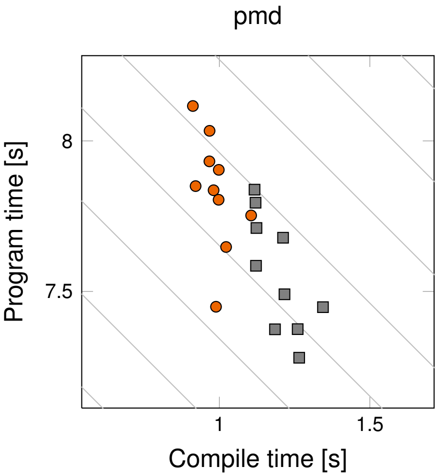
- 10 Compilation Plans

Evaluation: Performance Measurements with Replay-Compilation



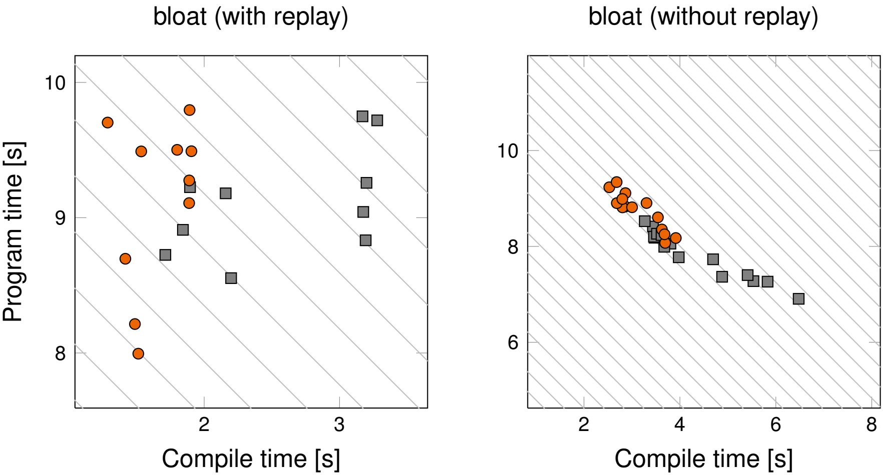
- 10 Compilation Plans
- Proposed Inlining Heuristic saves compile time

Evaluation: Performance Measurements with Replay-Compilation

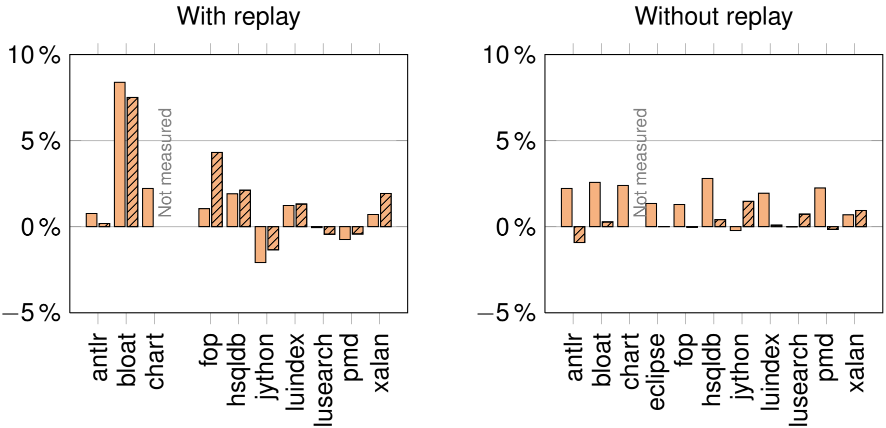


- 10 Compilation Plans
- Proposed Inlining Heuristic saves compile time
- Proposed Inlining Heuristic (sometimes) increases program time

Evaluation: Performance Measurements with and without Replay-Compilation



Evaluation: Speed-up on different architectures (▯▯ AMD Athlon 64, ▨▨ Intel Core i7)



Open Questions and Future Work

Open Questions and Future Work

- Assumption “*x*-induced edge ⇒ call on *x* argument” valid?

Open Questions and Future Work

- Assumption “*x*-induced edge ⇒ call on *x* argument” valid? And is gathering exact information worth it?

Open Questions and Future Work

- Assumption “*x*-induced edge $\Rightarrow$ call on *x* argument” valid? And is gathering exact information worth it?
- Replay compilation right methodology?

Open Questions and Future Work

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Assumption "x-induced edge $\Rightarrow$ call on x argument" valid? And is gathering exact information worth it?
- Replay compilation right methodology?
- How to account for other indirect benefits (checkcast, etc.)

Open Questions and Future Work

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Assumption “*x*-induced edge ⇒ call on *x* argument” valid? And is gathering exact information worth it?
- Replay compilation right methodology?
- How to account for other indirect benefits (checkcast, etc.)
- How to better spend compile time saved?

Open Questions and Future Work

- Assumption “*x*-induced edge ⇒ call on *x* argument” valid? And is gathering exact information worth it?
- Replay compilation right methodology?
- How to account for other indirect benefits (checkcast, etc.)
- How to better spend compile time saved?

```
Collections.sort(list, new Comparator() {
  int compare(Object lhs, Object rhs) {
    ...
  }
});
```