Incremental Pointer and Escape Analysis

Ondřej Lhoták

October 22, 2001

308−762

References

- J. Whaley and M. Rinard. *Compositional Pointer and Escape Analysis for Java Programs*. OOPSLA 1999
- F. Vivien and M. Rinard. *Incrementalized Pointer and Escape Analysis*. PLDI 2001

Outline

- Overview
- Definitions
- Intraprocedural Analysis
- Interprocedural Analysis
- Incrementalization
- "Analysis Policy"
- Experimental Results
- Conclusion

Overview

- Analysis to generate points−to graph and escape information
- Goal: Remove synchronization and stack−allocate objects (possibly inlining first)
- Flow sensitive
- Context sensitive
- Compositional/Incremental
- Cost−based/Demand−driven

Object Representation (Graph Vertices)

- Node
  - Inside Node `x = new Foo();`
    - Thread Node `x = new Thread();`
  - Outside Node
    - Parameter Node `foo(bar p){}  return p;`
    - Load Node `x = y.f;`
- Variable
  - Local variable `foo x;`
  - Parameter variable `foo(p){}`

Points−To Edges

A points-to escape graph is a pair $\langle O, I \rangle$, where

- $O \subseteq (N \times \mathtt{F}) \times N_L$ is a set of outside edges. We write an edge $\langle\langle n_1, \mathtt{f}\rangle, n_2\rangle$ as $n_1 \xrightarrow{\mathtt{f}} n_2$.
- $I \subseteq ((N \times \mathtt{F}) \times N) \cup (\mathtt{V} \times N)$ is a set of inside edges. We write an edge $\langle \mathtt{v}, n \rangle$ as $\mathtt{v} \rightarrow n$ and an edge $\langle\langle n_1, \mathtt{f}\rangle, n_2\rangle$ as $n_1 \xrightarrow{\mathtt{f}} n_2$.

$$e_{O,I}(n) = \{n' \in N_T \cup N_P . n \text{ is reachable from } n' \text{ in } O \cup I\}$$

- escaped$(\langle O, I \rangle, n)$ if $e_{O,I}(n) \neq \emptyset$
- captured$(\langle O, I \rangle, n)$ if $e_{O,I}(n) = \emptyset$

Intraprocedural Analysis

- Each method analyzed under assumption that parameters not aliased
- Edge sets initialized to

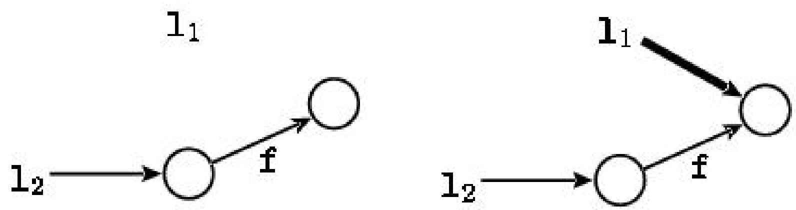
$$\langle \emptyset, \{\langle \mathrm{v}_i, n_{\mathrm{v}_i} \rangle . 1 \leq i \leq n\} \rangle$$

where n_{v_i} is the parameter node for parameter v_i.

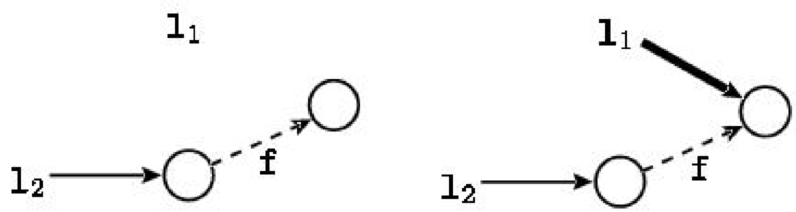
Intraprocedural Analysis

$l = v$

$l_1 = l_2.f$

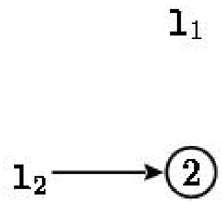


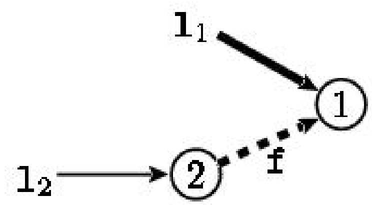
$l_1 = l_2.f$



Intraprocedural Analysis

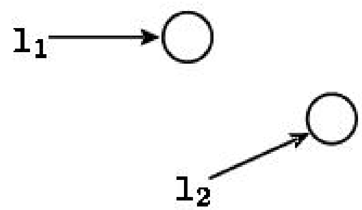
$l_1 = l_2.f$

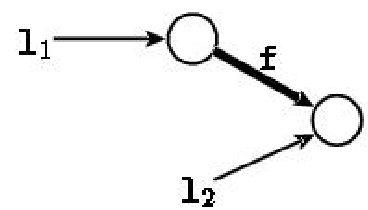




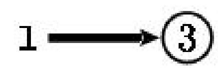
where 2 escaped 1 is the load node for $l_1 = l_2.f$

$l_1.f = l_2$





$l = \mathtt{new\ cl}$



where 3 is the inside node for $l = \mathtt{new\ cl}$

Intraprocedural Analysis

- Assignments to a local variable kill existing edges from the variable.
- Assignments to a field leave existing edges in place.
- At control flow merge points, union of all edges is taken.
- At end of method, all captured nodes, local, and parameter variables discarded

Interprocedural Analysis

We assume a call site of the form $\mathtt{l}_0\mathtt{.op}(\mathtt{l}_1, \ldots, \mathtt{l}_k)$, a potentially invoked method op with formal parameters $\mathtt{v}_0, \ldots, \mathtt{v}_k$, a points-to escape graph $\langle O_1, I_1 \rangle$ at the program point before the call site, and a graph $\langle O_2, I_2 \rangle$ from the end of op.

A map $\mu \subseteq N \times N$ combines the callee graph into the caller graph.

Interprocedural Analysis

$$\hat{\mu}(n) \subseteq \mu(n) \tag{1}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in O_2, n_3 \xrightarrow{\mathtt{f}} n_4 \in O_1 \cup I_1, n_1 \xrightarrow{\mu} n_3}{n_2 \xrightarrow{\mu} n_4} \tag{2}$$

$$\frac{\begin{array}{c} n_1 \xrightarrow{\mu} n_3, n_2 \xrightarrow{\mu} n_3, n_1 \neq n_2, \\ n_1 \xrightarrow{\mathtt{f}} n_4 \in O_2, n_2 \xrightarrow{\mathtt{f}} n_5 \in O_2 \cup I_2 \end{array}}{\mu(n_4) \subseteq \mu(n_5)} \tag{3}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in I_2, n_1 \xrightarrow{\mu} n, n_2 \in N_I}{n_2 \xrightarrow{\mu} n_2} \tag{4}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in O_2, n_1 \xrightarrow{\mu} n, \text{escaped}(\langle O, I \rangle, n)}{n_2 \xrightarrow{\mu} n_2} \tag{5}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in I_2}{(\mu(n_1) \times \{\mathtt{f}\}) \times \mu(n_2) \subseteq I} \tag{6}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in O_2, n_2 \xrightarrow{\mu} n_2}{(\mu(n_1) \times \{\mathtt{f}\}) \times \{n_2\} \subseteq O} \tag{7}$$

Interprocedural Analysis

- Map actual nodes of caller to parameter nodes of callee

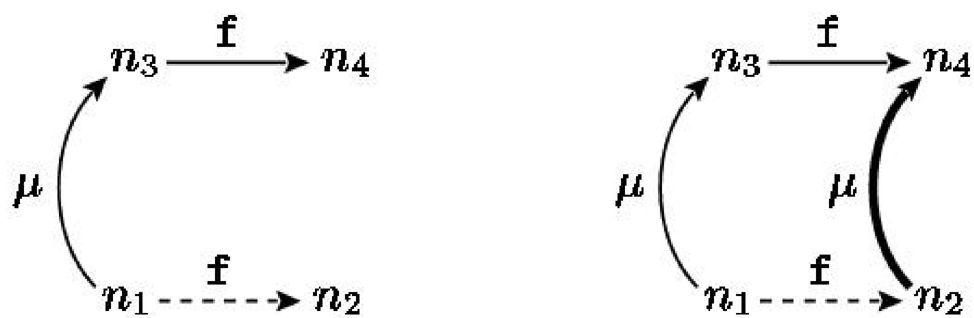
$$\hat{\mu}(n) = \begin{cases} I_1(\mathtt{l}_i) & \text{if } \{n\} = I_2(\mathtt{v}_i) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\hat{\mu}(n) \subseteq \mu(n) \tag{1}$$

Interprocedural Analysis

- Match outside nodes and edges from callee to nodes and edges from caller

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in O_2,\ n_3 \xrightarrow{\mathtt{f}} n_4 \in O_1 \cup I_1,\ n_1 \xrightarrow{\mu} n_3}{n_2 \xrightarrow{\mu} n_4} \tag{2}$$



Interprocedural Analysis

- Map aliases from caller into callee

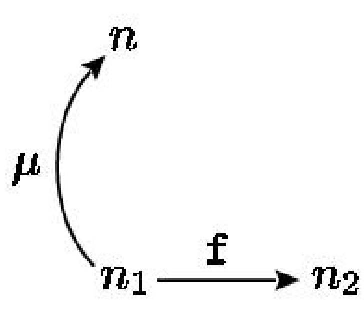
$$\frac{\begin{array}{c} n_1 \xrightarrow{\mu} n_3, n_2 \xrightarrow{\mu} n_3, n_1 \neq n_2, \\ n_1 \xrightarrow{\mathtt{f}} n_4 \in O_2, n_2 \xrightarrow{\mathtt{f}} n_5 \in O_2 \cup I_2 \end{array}}{\mu(n_4) \subseteq \mu(n_5)} \qquad (3)$$

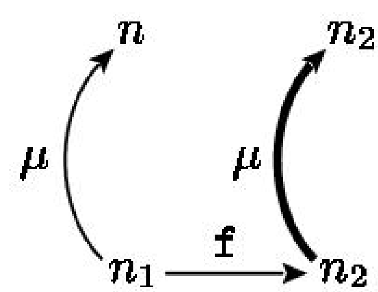
Interprocedural Analysis

- Map nodes escaping from callee into caller

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in I_2,\ n_1 \xrightarrow{\mu} n,\ n_2 \in N_I}{n_2 \xrightarrow{\mu} n_2} \tag{4}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in O_2,\ n_1 \xrightarrow{\mu} n,\ \text{escaped}(\langle O, I\rangle, n)}{n_2 \xrightarrow{\mu} n_2} \tag{5}$$



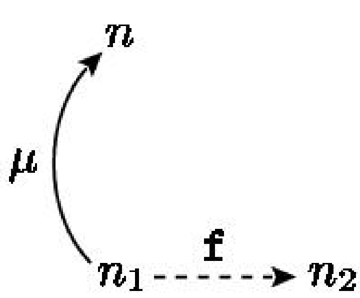


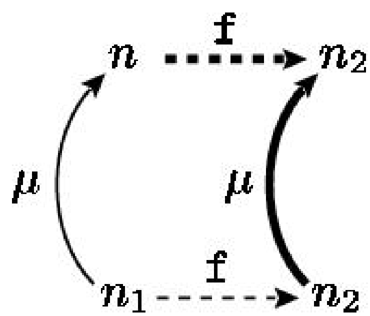
Interprocedural Analysis

- Use map to convert inside and outside edges from callee to caller

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in I_2}{(\mu(n_1) \times \{\mathtt{f}\}) \times \mu(n_2) \subseteq I} \tag{6}$$

$$\frac{n_1 \xrightarrow{\mathtt{f}} n_2 \in O_2, n_2 \xrightarrow{\mu} n_2}{(\mu(n_1) \times \{\mathtt{f}\}) \times \{n_2\} \subseteq O} \tag{7}$$





Interprocedural Analysis

- Because of dynamic dispatch, a call site may invoke multiple target methods.
- Solution is to merge the analyses of all potential targets by taking the union of edges sets, as with an intraprocedural control flow merge.

Incrementalization

- Goal: Delay analysis of call sites
  - Produce conservative result if callee is never analyzed
  - Re−integrate result of analysis of callee into a completed analysis of caller

Incrementalization

- If the callee is never analyzed, simply consider all nodes escaping into it as permanently escaped.
- If the callee is later analyzed, the key obstacle to re−integration is flow−sensitivity.

Incrementalization

- $\omega \subseteq S \times ((N \times \{\mathtt{f}\}) \times N_L)$. For each call site s, $\omega(s) = \{n_1 \xrightarrow{\mathtt{f}} n_2 . \langle s, n_1 \xrightarrow{\mathtt{f}} n_2 \rangle \in \omega\}$ is the set of outside edges that the analysis generates before it skips s.

- $\iota \subseteq S \times ((N \times \{\mathtt{f}\}) \times N)$. For each call site s, $\iota(s) = \{n_1 \xrightarrow{\mathtt{f}} n_2 . \langle s, n_1 \xrightarrow{\mathtt{f}} n_2 \rangle \in \iota\}$ is the set of inside edges that the analysis generates before it skips s.

- $\tau \subseteq S \times ((N \times \{\mathtt{f}\}) \times N_L)$. For each call site s, $\tau(s) = \{n_1 \xrightarrow{\mathtt{f}} n_2 . \langle s, n_1 \xrightarrow{\mathtt{f}} n_2 \rangle \in \tau\}$ is the set of outside edges that the analysis generates after it skips s.

- $\nu \subseteq S \times ((N \times \{\mathtt{f}\}) \times N)$. For each call site s, $\nu(s) = \{n_1 \xrightarrow{\mathtt{f}} n_2 . \langle s, n_1 \xrightarrow{\mathtt{f}} n_2 \rangle \in \nu\}$ is the set of inside edges that the analysis generates after it skips s.

- $\beta \subseteq S \times S$. For each call site s, $\beta(s) = \{s' . \langle s, s' \rangle \in \beta\}$ is the set of call sites that the analysis skips before skipping s.

- $\alpha \subseteq S \times S$. For each call site s, $\alpha(s) = \{s' . \langle s, s' \rangle \in \alpha\}$ is the set of call sites that the analysis skips after skipping s.

Incrementalization

- When computing map to merge callee graph into caller, only use edges generated before call site.

$$\langle O, I, \mu \rangle = \mathrm{map}(\langle \omega_1(s), \iota_1(s) \rangle, \langle O_2, I_2 \rangle, \hat{\mu}_s)$$

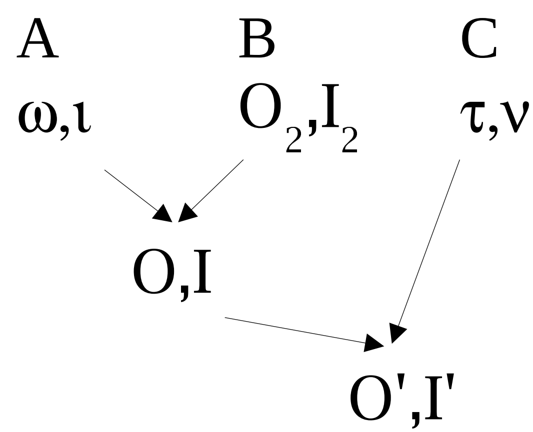
Incrementalization

- Callee may introduce new edges into the call graph, which may in turn cause more edges to be generated.
- BUT, all such edges come from nodes escaping into callee, and therefore will be represented in caller by outside edges. We can therefore reconstruct them.

Incrementalization

- Idea: treat the part of the caller after the call site as a callee

```
CodeA;        CodeA;
CallB();  ->  CallB();
CodeC;        CallC();
```



$$\langle O, I, \mu \rangle = \mathrm{map}(\langle \omega_1(s), \iota_1(s) \rangle, \langle O_2, I_2 \rangle, \hat{\mu}_s)$$

$$\langle O', I', \mu' \rangle = \mathrm{map}(\langle O, I \rangle, \langle \tau_1(s), \nu_1(s) \rangle, \{\langle n, n \rangle . n \in N\})$$

Incrementalization

- The analysis of a call site may add nodes to the formal parameter node mappings at a subsequent site.
- When integrating analysis result from previously skipped call site, parameter maps of all subsequent call sites must be composed with the map from the integration.

Incrementalization

- Similarly, parameter maps for skipped call sites within the callee must be composed with the map from before the original call site.

Incrementalization

- Orders must be recomputed.

$$\begin{aligned}
\omega' &= \omega_1 \cup \omega_2[\mu] \cup (S_2 \times \omega_1(s)) \cup (\alpha_1(s) \times O) \\
\iota' &= \iota_1 \cup \iota_2[\mu] \cup (S_2 \times \iota_1(s)) \cup (\alpha_1(s) \times I) \\
\tau' &= \tau_1 \cup \tau_2[\mu] \cup (S_2 \times \tau_1(s)) \cup (\beta_1(s) \times O) \\
\nu' &= \nu_1 \cup \nu_2[\mu] \cup (S_2 \times \nu_1(s)) \cup (\beta_1(s) \times I) \\
\beta' &= \beta_1 \cup \beta_2 \cup (S_2 \times \beta_1(s)) \cup (\alpha_1(s) \times S_2) \\
\alpha' &= \alpha_1 \cup \alpha_2 \cup (S_2 \times \alpha_1(s)) \cup (\beta_1(s) \times S_2)
\end{aligned}$$

Here $\omega[\mu]$ is the order ω under the map μ, i.e., $\omega[\mu] = \{\langle s, n'_1 \xrightarrow{\mathtt{f}} n'_2 \rangle . \langle s, n_1 \xrightarrow{\mathtt{f}} n_2 \rangle \in \omega, n_1 \xrightarrow{\mu} n'_1, \text{ and } n_2 \xrightarrow{\mu} n'_2\}$, and similarly for ι, τ, and ν.

Incrementalization

- Problem: What if a call site is executed multiple times?
- Solution: Keep track of this, and if it is possible for a call site to be executed multiple times, iterate the integration of the analysis until a fixed point is reached.

Incrementalization

- Problem: Recursion.
- Solution:
  - Base analysis iterates to fixed point.
  - Incrementalized version could also.
  - Implementation does not iterate, leaving it to the "Analysis Policy"

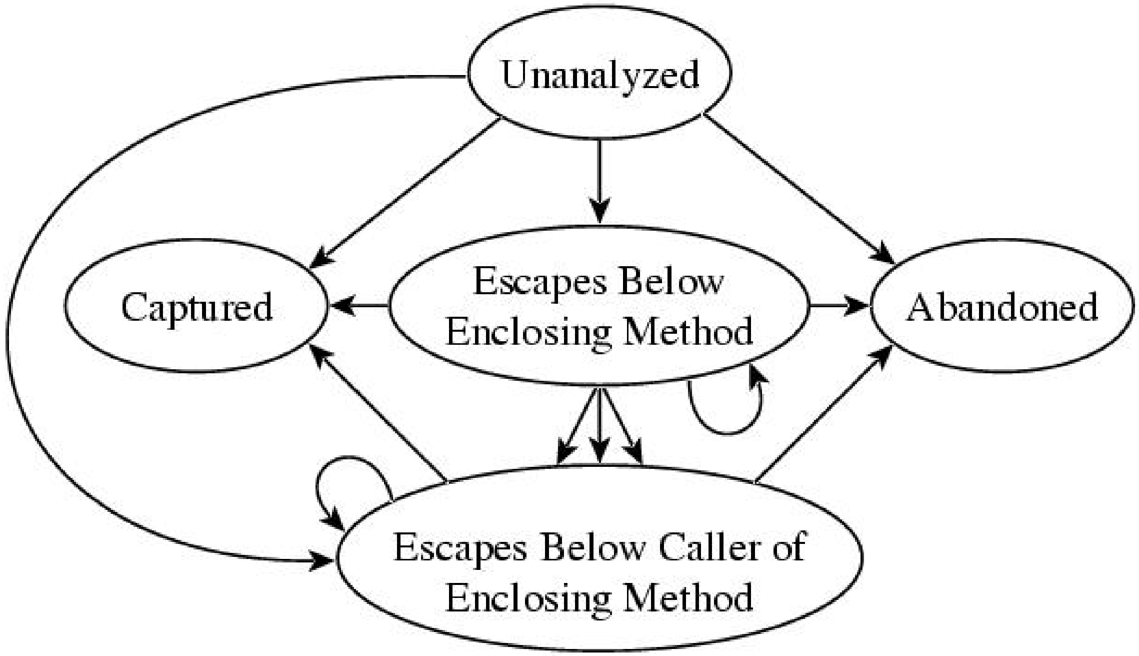
Analysis Policy

- Idea: Pick an allocation site, and analyze only those methods needed to prove that it is captured.
- Trade off predicted analysis time against predicted payoff from stack−allocation

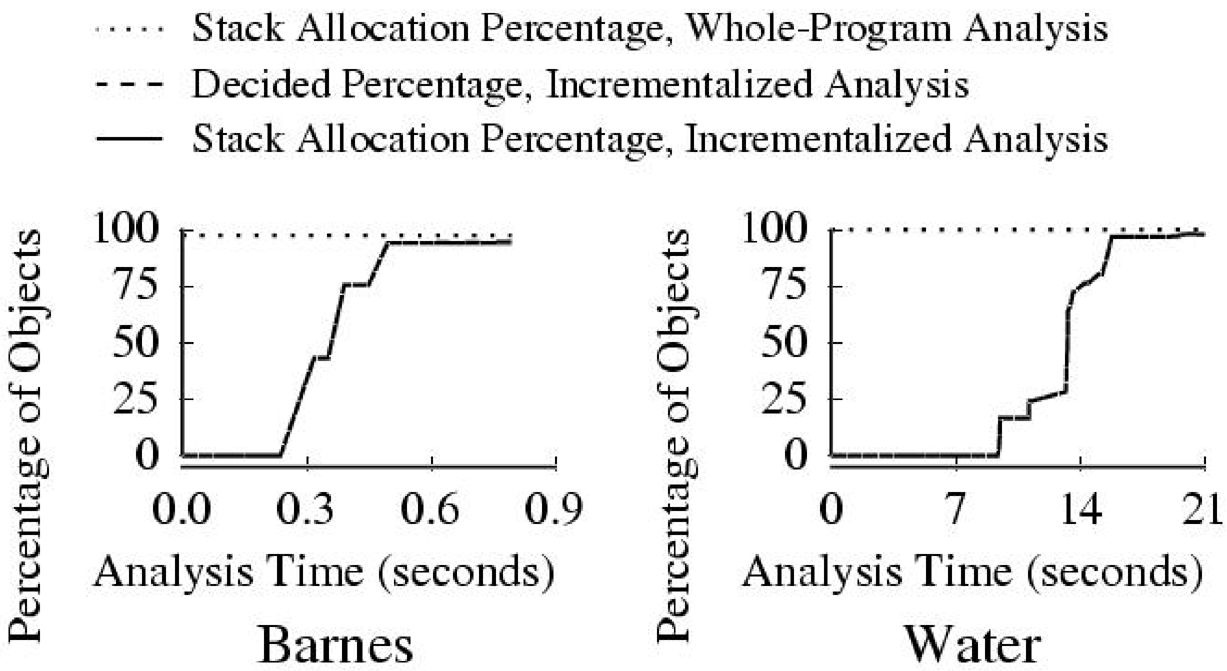
Analysis Policy

- a: candidate allocation site for analysis
- Op: method containing a
- G: current points−to escape graph for a
- p: estimated payoff (from profiling data)
- c: # of skipped call sites where a escapes
- d: call depths of analyzed region
- m: mean cost of analyses for a so far

Analysis Policy



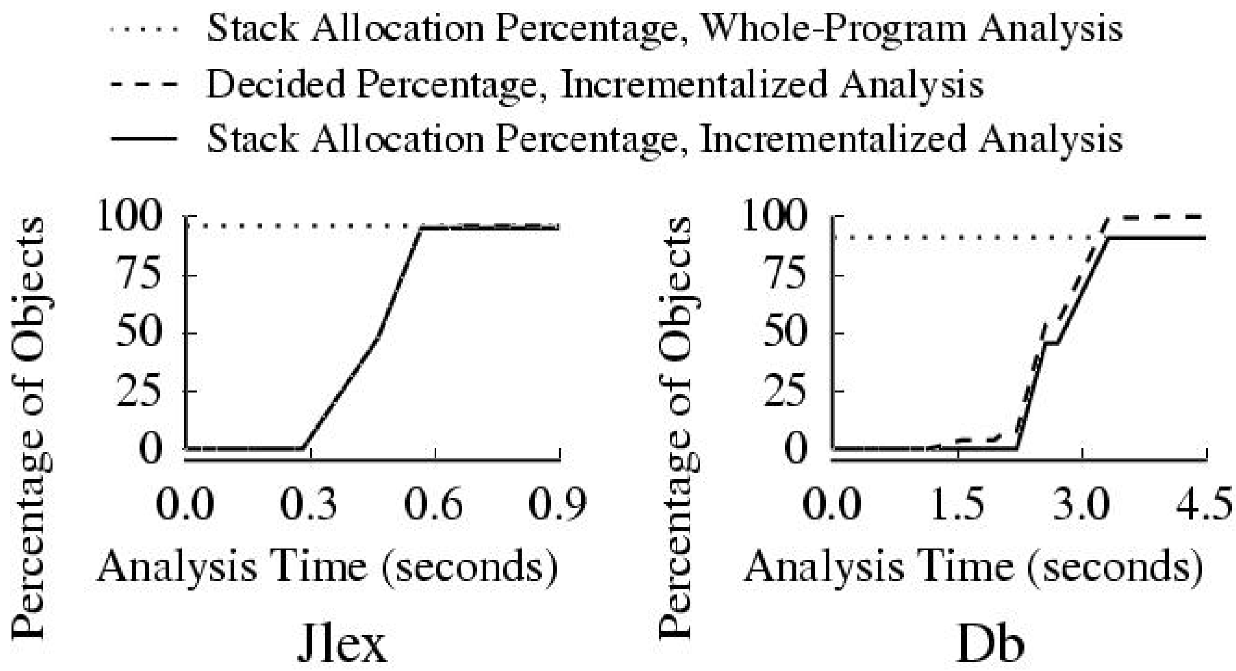
Experimental Results



Whole: 34.3 s

Whole: 38.2 s

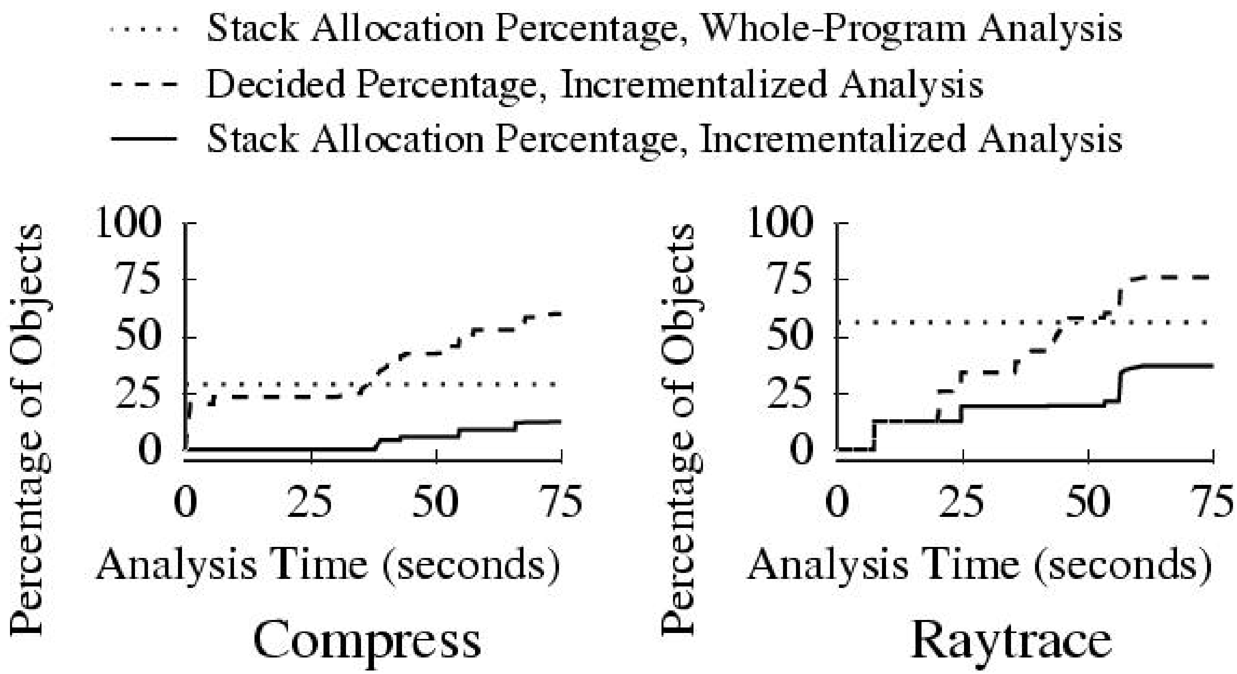
Experimental Results



Whole: 222.8 s

Whole: 126.6

Experimental Results



Whole: 645.1 s

Whole: 102.2 s

Conclusion

- Cost−directed incrementalized analysis produces results almost as accurate as a whole−program flow−sensitive analysis in significantly less time.