

Review of  
Efficient Points-to Analysis for  
Whole Program Analysis

---

-Donglin Liang and Mary Jean Harrold

By  
Shweta Behere  
Feb 21, 2006

# What problem does the paper address?

---

## Big Picture Problem

Program Optimization and Error Detection for Software Systems

## Specific Problem

Providing fast and precise alias information to whole program analysis tools

## Why is the problem hard?

Algorithm must be scalable to large programs

Alias algorithm precision is essential for whole program analysis

---

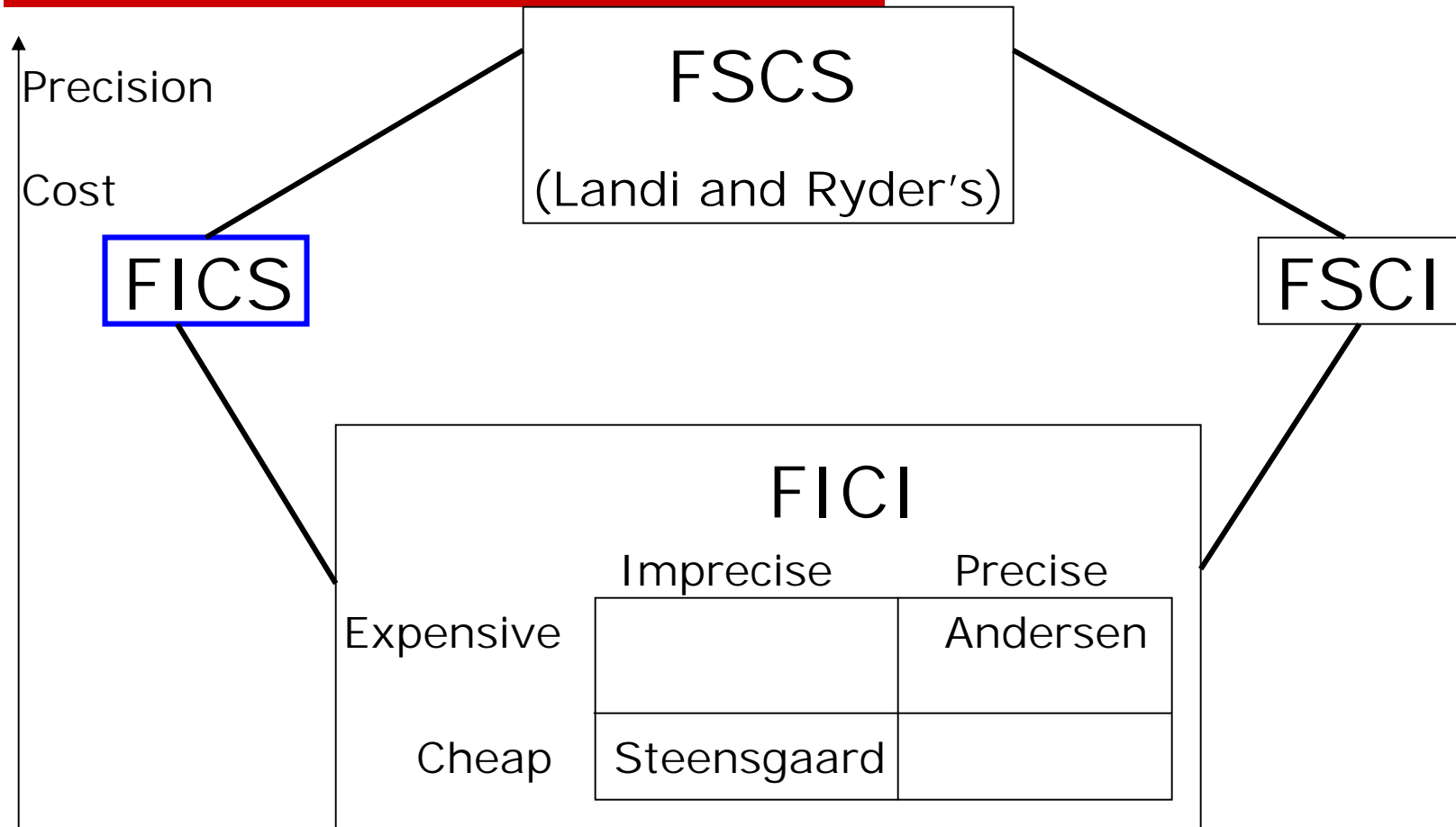
# Why do we care?

---

- ❑ Numerous languages such as C, make extensive use of pointers
  - ❑ Useful for program understanding, enhancement and maintenance
  - ❑ Many existing alias-analysis tools are too imprecise
  - ❑ “Alias analysis is a key ingredient in many compiler optimizations” - [Landi,Ryder97]
-

# Alias-analysis techniques

---



# Approach

---

Three phase approach:

- ❑ Create points-to graphs for individual procedures
  - ❑ Propagates alias information from callee to caller and generates global points-to graph
  - ❑ Propagates alias information from the caller to the callee and from global variables to each procedure
-

# Example: Phase I - Points-to graphs for each procedure

---

```
int *buf1,*buf2;
```

```
main() {
```

```
int *q, *r;
```

```
...
```

```
q = buf1;
```

```
...
```

```
...
```

```
q = buf2;
```

```
r = incr_ptr(q); ptr=q;r=incr_ptr;
```

```
...
```

```
int *incr_ptr(int *ptr) {
```

```
return ptr+1; incr_ptr=ptr;
```

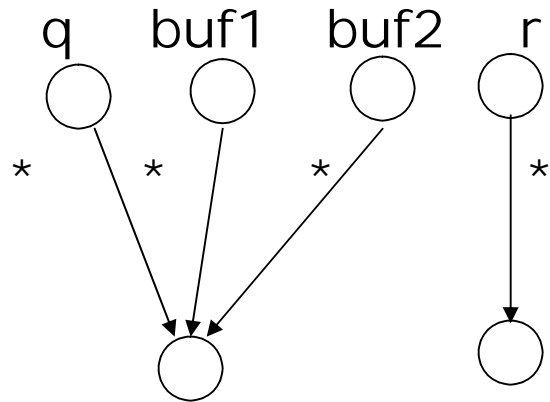
```
}
```

```
}
```

---

# Phase I - Points-to graphs for each procedure

---



`main()`

---

# Phase II - Bottom-up Approach

---

```
int *buf1,*buf2;
```

```
main() {
```

```
int *q, *r;
```

```
...
```

```
q = buf1;
```

```
...
```

```
...
```

```
q = buf2;
```

```
r = incr_ptr(q); ptr=q;r=incr_ptr;
```

```
...
```

```
int *incr_ptr(int *ptr) {
```

```
return ptr+1; incr_ptr=ptr;
```

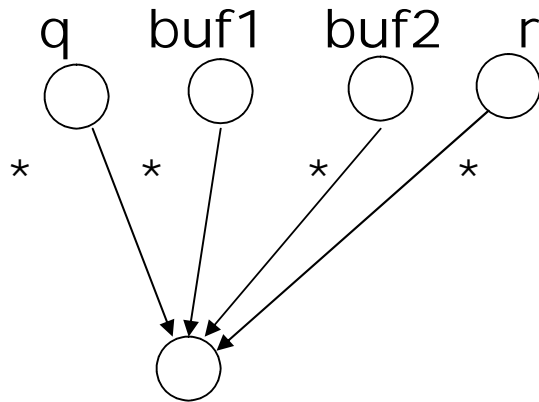
```
}
```

```
}
```

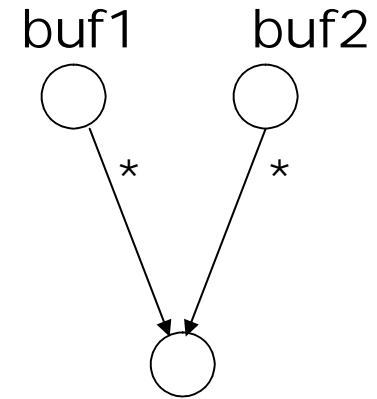
---

# Phase II – callee to caller

---



`main()`



`h_17,h_18`

`global`

---

# Phase III – Top Down Approach

---

```
int *buf1,*buf2;
```

```
main() {
```

```
int *q, *r;
```

```
...
```

```
q = buf1;
```

```
...
```

```
...
```

```
q = buf2;
```

```
r = incr_ptr(q); ptr=q;r=incr_ptr;
```

```
...
```

```
int *incr_ptr(int *ptr) {
```

```
return ptr+1; incr_ptr=ptr;
```

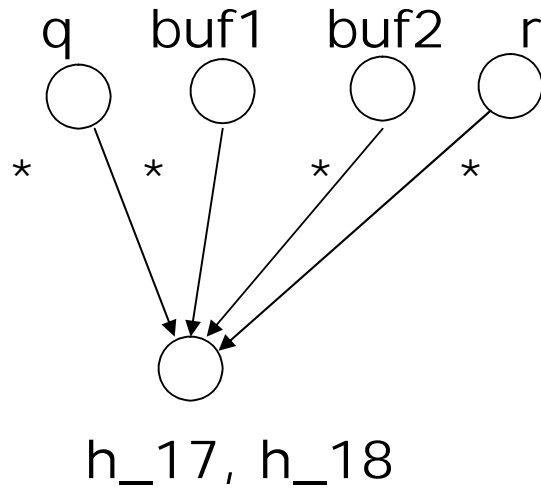
```
}
```

```
}
```

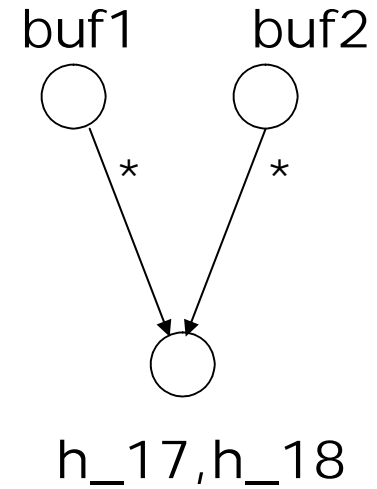
---

# Phase III – caller to callee

---



`main()`



`global`

---

# Evaluation of Approach

---

- Empirical Studies

Studies focused on performance, precision, cost and whole program analysis

- Cites related work and compares similar approaches

- Complexity of the algorithm

---

# Conclusions (mostly justified by empirical results)

---

- ❑ Steengaard's algorithm too imprecise
  - ❑ Andersen's, Landi and Ryder's algorithm too costly
  - ❑ FICS as precise as Andersen's and runtime comparable to Steengaard's
  - ❑ FICS more effective for whole-program analysis
-

# Future Research Areas

---

- ❑ Results on large programs  
( > 25000 LOC)
  - ❑ Effect of imprecision on whole-program analysis
  - ❑ Handling complex program structures such as functions pointers, setjump etc
  - ❑ More work on comparison with similar approaches on real systems
-

# Critique

---

- Effective comparison of the alias-analysis algorithms with a concrete example and empirical studies
  - Landi and Ryder's not handled well
  - No information on criteria used to select the programs for experimental study
  - Examples needed on integration of FICS with whole program analysis
-

# Relation to CS653

---

- Helpful to understand and compare the different alias analysis techniques
  - Realize the importance of pointer analysis in whole-program analysis
  - Good approach for solving the open questions in pointer analysis
-