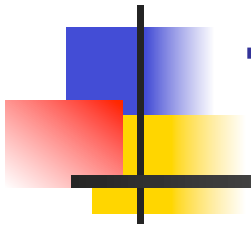


Thin Slicing

Manu Sridharan, Ras Bodík
UC Berkeley

Stephen J. Fink
IBM Research

"Thin-slicing is part of what makes the unconscious so dazzling. But it's also what we find most problematic about rapid cognition. How is it possible to gather the necessary information for a sophisticated judgment in such a short time?"

Malcolm Gladwell,
Blink: The Power of Thinking Without Thinking

Slices Large *By Definition*

Goal: show code "relevant" to *seed* statement

E.g., seed is crash point, cause in relevant code

Slice relevance: all stmts that may affect seed ***s***

- Affect = transitive control + data dependences
- Intuitive: returns executable subset

Problem: slice relevance too broad for user tasks

- Slices often most of the program
- Better analysis won't help!

Thin slicing approach: *Task-centric relevance*

- Focus on direct value flow to seed
- 3.3X, 9.4X reduction in simulated developer effort

A Typical Large Slice

```
String[] readNames(InputStream input) {
  String[] firstNames = new String[100]; int i = 0;
  while (!eof(input)) {
    String fullName = readFullName(input);
    int spaceInd = fullName.indexOf(' ');
    if (spaceInd != -1) {
      // BUG: should pass spaceInd
      String firstName = fullName.substr(0, spaceInd-1);
      firstNames[i++] = firstName; } }      •••
  return firstNames; }
void printNames(String[] firstNames) {
  for (int i = 0; i < firstNames.length; i++) {
    String firstName = firstNames[i];
    print("FIRST NAME: " + firstName);
  }}
void main(String[] args) {
  String[] firstNames = readNames(…);
  SessionState s = getState(); s.setNames(firstNames);
  if (handleRequests()) {
    printNames(getState().getNames()); }}
```

The slice:
Too many statements!

```
FIRST NAME: Man
FIRST NAME: Stephe
FIRST NAME: Rastisla
```

Task-Centric Relevance

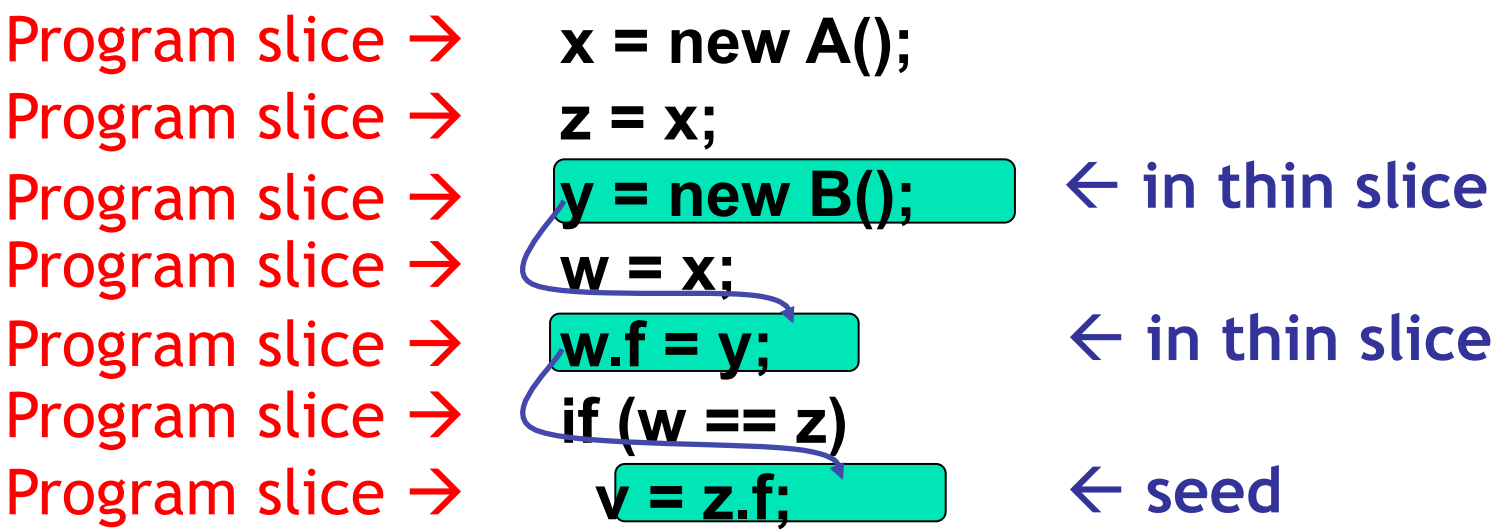
For tasks, value flow often most important

Thin slice relevance: *producers* for seed

- Producer def: flows a "top-level" value to seed

  Top-level: ignoring dereferenced pointers

- Interprocedural def-use chains (including heap)



Thin Slicing in Action

```
String[] readNames(InputStream input) {
  String[] firstNames = new String[100]; int i = 0;
  while (!eof(input)) {
    String fullName = readFullName(input);
    int spaceInd = fullName.indexOf(‘ ‘);
    if (spaceInd != -1) {
      // BUG: should pass spaceInd
      String firstName = fullName.substr(0, spaceInd-1);
      firstNames[i++] = firstName; } }
  return firstNames; }
void printNames(String[] firstNames) {
  for (int i = 0; i < firstNames.length; i++) {
    String firstName = firstNames[i];
    print(“FIRST NAME: “ + firstName);
  }}
void main(String[] args) {
  String[] firstNames = readNames(…);
  SessionState s = getState(); s.setNames(firstNames);
  if (handleRequests()) {
    printNames(getState().getNames()); }}
```

Are We Done?

Tried several debugging, comprehension tasks

For ~50% of tasks, thin slice alone suffices

For other tasks:

- Often need thin slice + a couple statements
- Can we handle these cases?

Thin Slice Expansion

Thin slices exclude *explainers*

Explainer def: shows *why* producer can affect seed

- Why heap accesses read / write same object, or
- Why producer can execute

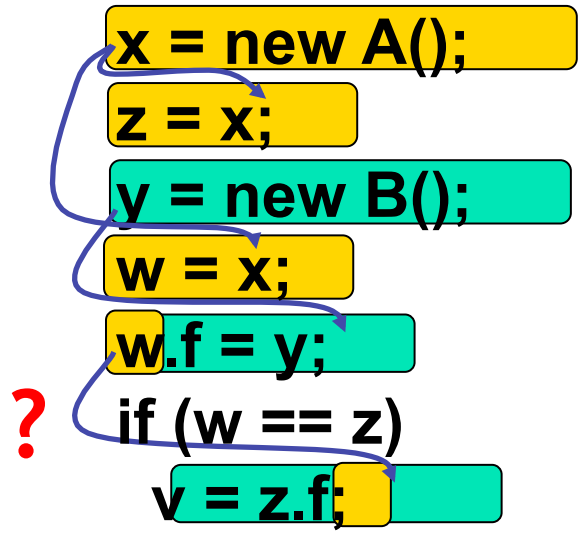
Most explainers not useful for tasks

(Transitive) producers + explainers = whole slice

Expose with incremental expansion

- Guided by user
- Typically, little expansion needed

Explaining Heap-Based Flow



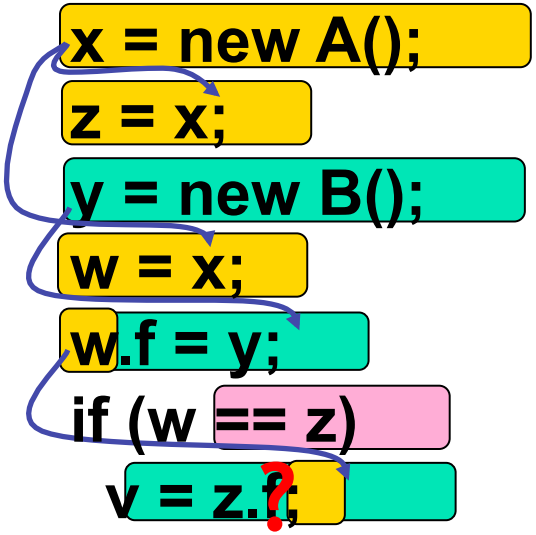
Question: why are base pointers may-aliased?

Answer: two more thin slices!

Shows flow of common object(s)

Incremental: just one level of data structures

Explaining Control Flow



Question: why can producer execute?

Answer: *lexically close* control dependences

- Always sufficient in tested tasks
- Usually, source code navigation enough

Evaluation Methodology

Hypothesis: more effective for developer tasks

E.g., tracking down a bug

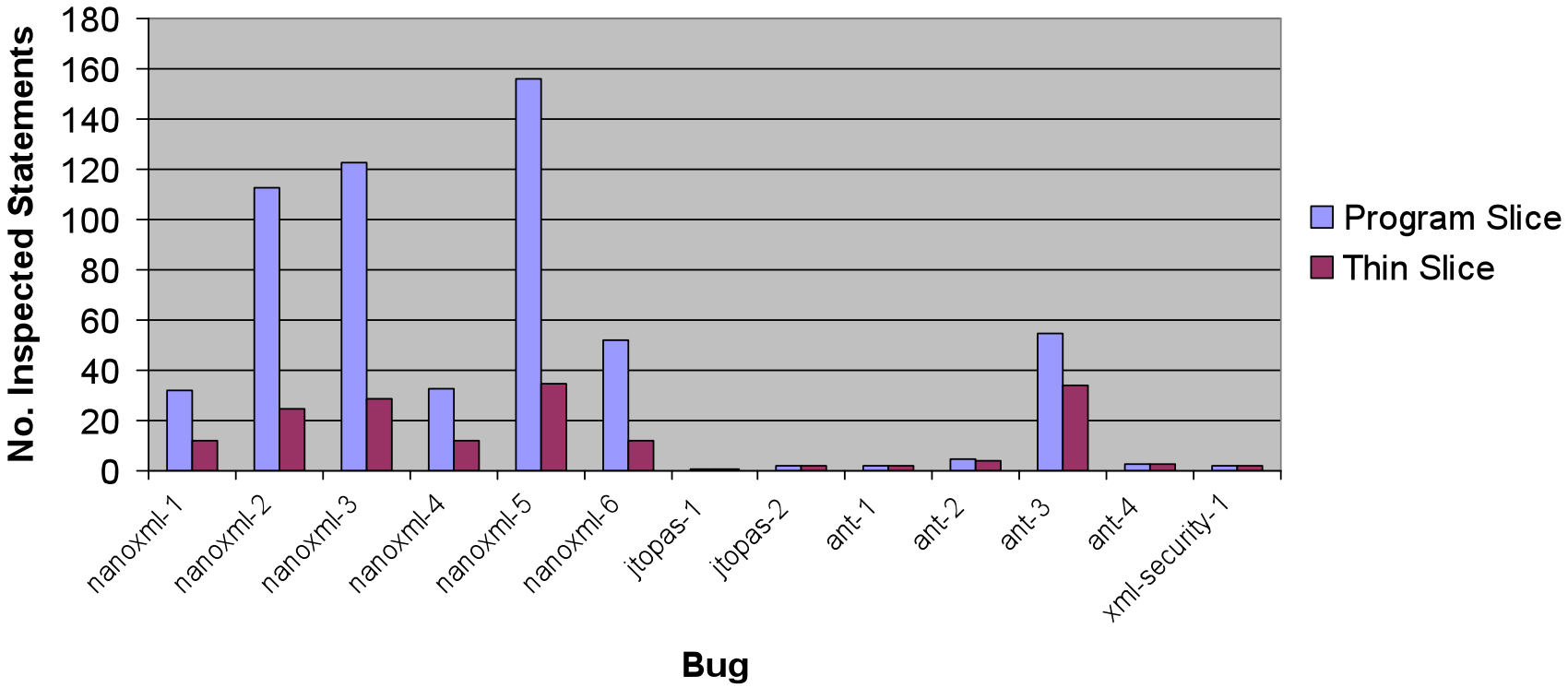
Slice sizes not a good metric

- Developer stops when cause discovered
- Likely to browse dependences, as in Codesurfer

Compare simulated developer effort (Renieris and Reiss, ASE03)

- BFS from crash point (seed) to cause of bug
- Count reached ("inspected") statements
- (Include identical control dependences)

Program Slicing vs. Thin Slicing



Mean of 12 inspected stmts / thin slice

Manageable for a developer

Overall, 3.3X fewer inspected stmts

In an understanding experiment, 9.4X fewer inspected stmts

Scalable and Precise Thin Slicing

Two key computations

- Points-to analysis (call graph, aliasing info)
- Reachability on dependence graph

For precision: Context-*sensitive* points-to analysis

- Used Andersen's + object-sensitive containers
- Just Andersen's) up to 17.2X more inspected stmts

For scalability: Context-*insensitive* reachability

- Context-sensitive bottleneck: heap accesses as parameters
- In tested tasks, no precision loss observed

Conclusions / Future Work

Program slices too large by definition

- Problem: relevance too broad

For thin slicing, only producers relevant

- Sufficient for ~50% of tasks

Expand to show useful explainers

- Usually close to producers

Bottom line: basis for practical slicing tool

- Next steps: Eclipse front end, user study

Get the code! http://wala.sourceforge.net