A Tool for Lightweight and Efficient Pointer Analysis in Source Code

Pointer analysis determines what variables can be referenced by which pointers.
In this study, we present a lightweight, fast, and flexible tool for pointer analysis.

Student Undergraduate Research Symposium, 2017
Computer Science Department, Kent State University
Vlas Zyrianov
Advisors: Prof. Jonathan I. Maletic & Christian Newman

What are Pointers?

- Code is a list of instructions that the computer executes to do tasks.
- Variables are a fundamental part of code. They store almost any data: numbers, strings, pictures, etc.
- Pointers are a type of variable that store the address of another variable. Pointers are useful as a way to reference and modify another variable without doing it directly. They *point* to other variables.
- Pointers to pointers are possible!

```
int x;
string y;


x = 10;
y = "Hello World!";


int* ptr;
string* sprt;


ptr = &x;
sprtr = &y;


*ptr = 25;
*sptr = "abc";
```

Declaring:
- A variable named x that stores an integer
- A variable named y that stores a string

Assigning the values:
- 10 to x
- "Hello World" to y

Declaring:
- A variable, of type pointer to an integer, called ptr.
- A variable of type pointer to a string called sptr

Assigning:
- ptr to point to x
- sptr to point to y

Indirectly assigning:
- x to equal 25
- y to equal "abc"

What is Pointer Analysis?

In computer science, *pointer analysis*, or *points-to analysis*, is a static code analysis technique that establishes which pointers can point to which variables, or storage locations. Pointer analysis is important for finding bugs, impact analysis, and optimization. In general it is difficult to solve all cases and is shown to be NP-Hard.

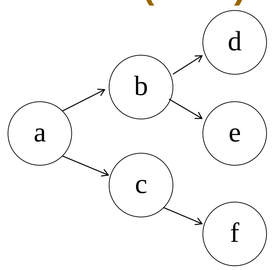
Pointer analysis is hard because transitive closure over the graph is expensive. One of the reasons why Steensgaard's algorithm is faster than Andersen's is because it doesn't compute transitive closure. Andersen's algorithm is $O(n^3)$ while Steensgaard's is nearly linear.

Given the following relationships between variables:

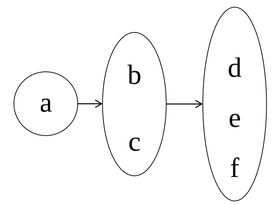
```
a = &b;
a = &c;
b = &d;
b = &e;
c = &f;
```

The algorithms produce the following graphs:

Andersen (1994)

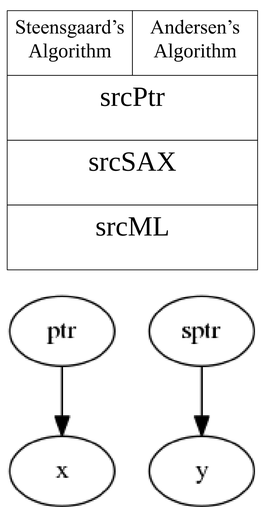


Steensgaard (1996)



srcPtr

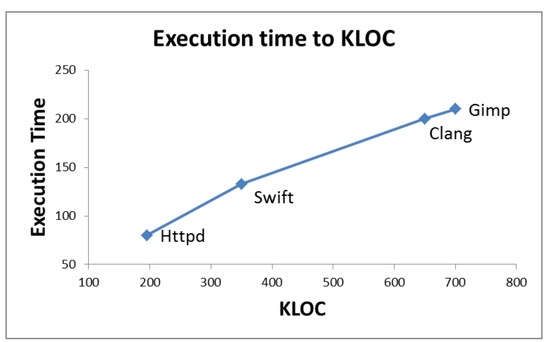
- Developed in C++ using the srcML infrastructure developed here at KSU (see www.srcML.org)
- Uses srcSAX event dispatcher with srcML which provides abstract syntax information of the code
- srcPtr identifies pointers and all points-to relationships
- Analysis algorithms are then implemented on top of this infrastructure
- Currently have Andersen's implemented
- Easy to extend with other algorithms
- Generates a graph output for viewing results with Graphviz



srcPtr will be made open source on GitHub when complete.

Results

When run without any pointer analysis algorithm (just the srcPtr base):
- Httpd (195 KLOC) takes 1 minute and 20 seconds
- Swift (350 KLOC) takes 2 minutes and 13 seconds
- Clang (650 KLOC) takes 3 minutes and 20 seconds
- Gimp (700 KLOC) takes 3 minutes and 30 seconds



Further Work

- Steensgaard's algorithm and other pointer analysis algorithms still need to be implemented into srcPtr
- Finish preperations to make srcPtr open source
- Study trade offs of pointer analysis algorithms in practice