Flow-sensitive Alias Analysis

Last time

– Client-Driven pointer analysis

Today

– Demand DFA paper

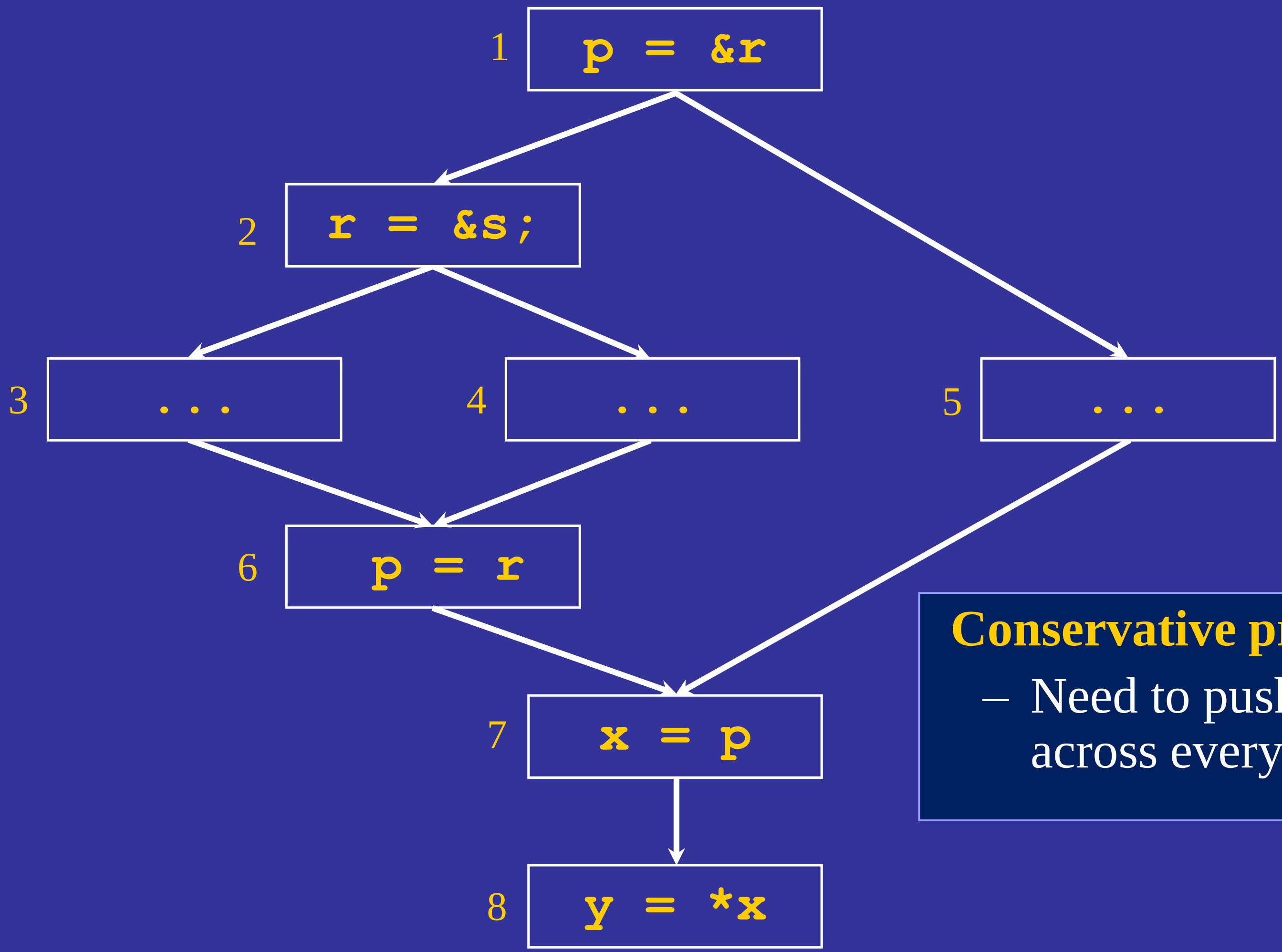
– Scalable flow-sensitive alias analysis

Recall Previous Flow-Sensitive Solution

Iterative data-flow analysis

– We’ve seen how IDFA could be use to compute May Points-to and Must Points-to information
– This solution does not scale– can only analyze C programs with 10’s of thousands of lines of code

The Problem



Conservative propagation

– Need to push data-flow facts across every node

The Problem (cont)

Conservative propagation

- The analysis doesn't know which nodes require pointer information ⇒ must propagate information to all reachable nodes
- We need to store, propagate, and compute transfer functions for all pointer information at all program points

How large can this information be?

- For programs with 100K to 1M LOC
  - 100's of thousands of program points
  - Two points-to graphs per program point (for In and Out sets)
  - Each points-to graph can contains 10's of thousands of pointers (nodes)
  - Each points-to set can contain 100's or 1000's of elements (edges)

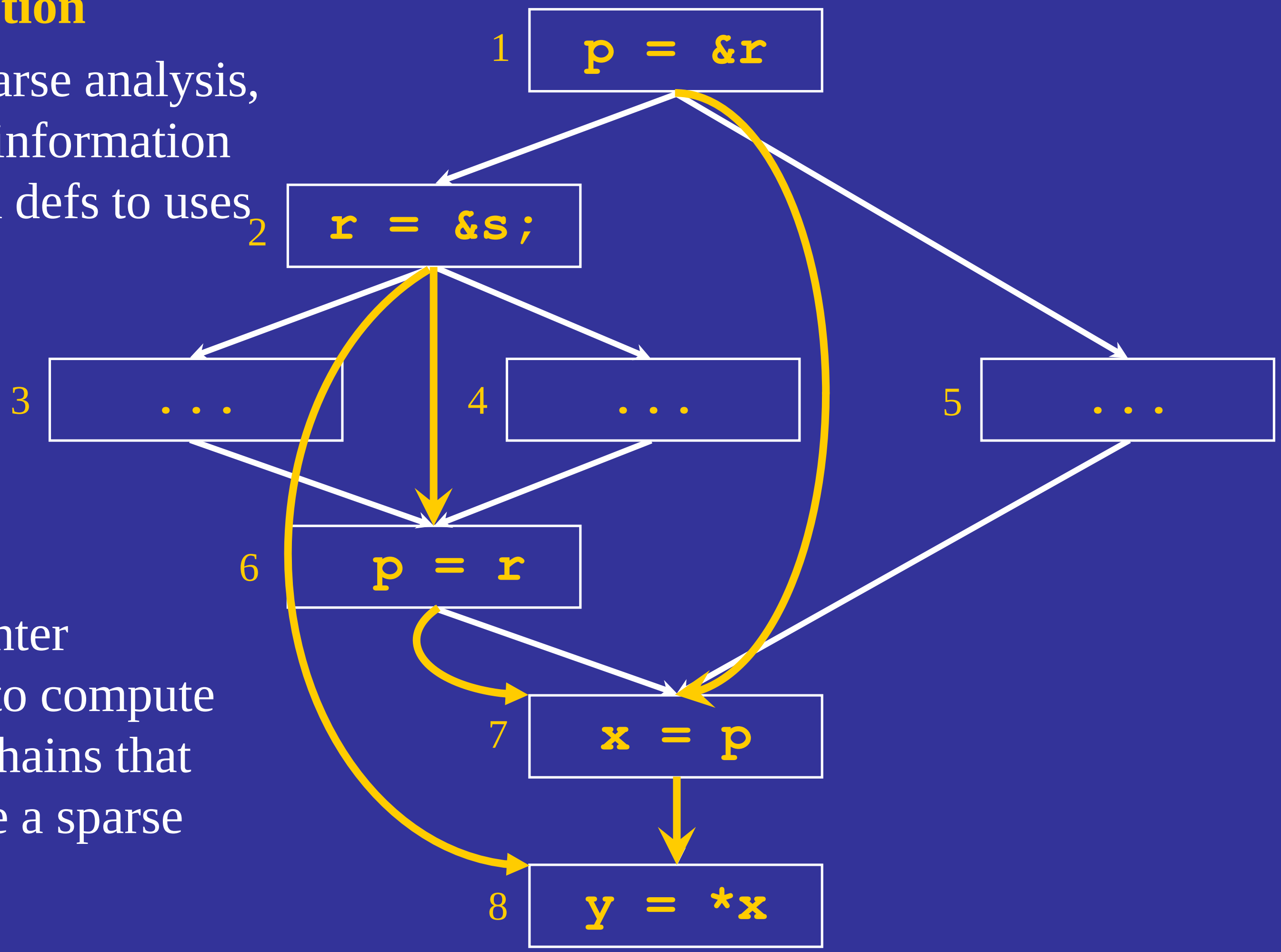
Exploiting Sparsity

Traditional solution

- Employ a sparse analysis, propagating information directly from defs to uses
- What if this code were in a loop?

Catch-22

- We need pointer information to compute the def-use chains that would enable a sparse analysis



Previous Work

Dynamically compute def-use information [Chase et al, ’90, Tok & Lin’06]

– High overhead limits scalability

– Scales to 70K LOC

Semi-Sparse Analysis [Hardekopf & Lin ’09]

– Separate pointers into two groups: Top-level and Address-taken

– Top-level variables

  – Addresses are never taken

  – Can easily put these variables into SSA form

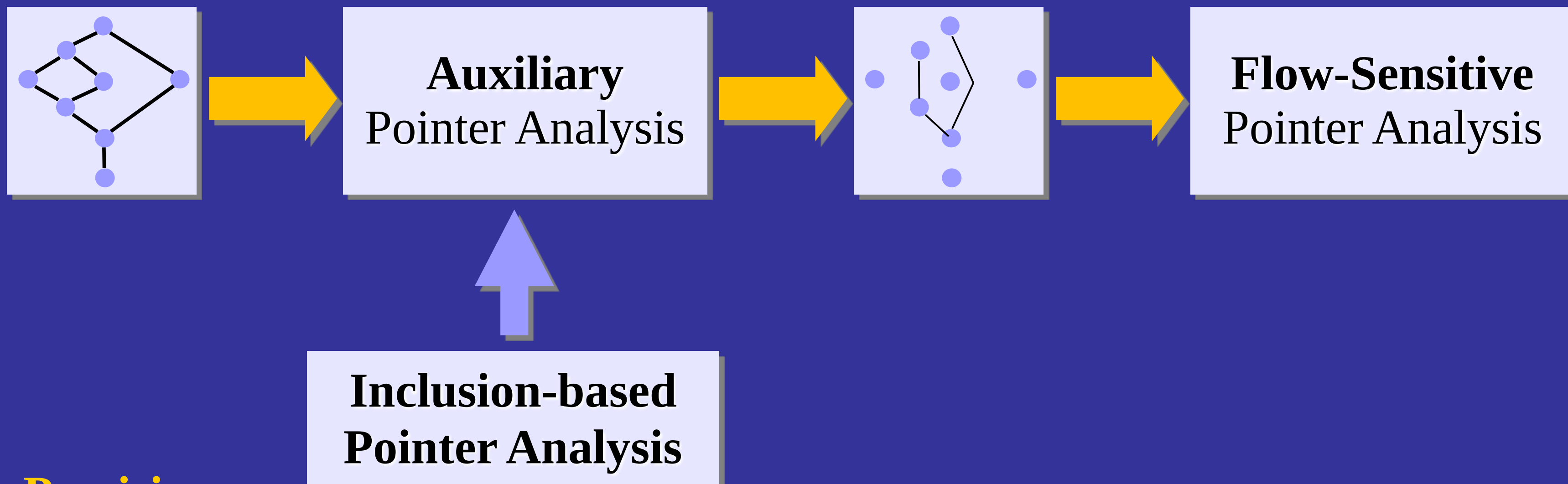
– Address-taken variables

  – Use traditional IDFA

– Scales to 300K LOC

A Better Solution

Staged Flow-Sensitive Analysis [Hardekopf & Lin ’11]



Precision

- The precision of the auxiliary analysis impacts performance but not precision– as long as the auxiliary analysis is sound
- Use inclusion-based pointer analysis as auxiliary analysis, since it’s the most accurate of the flow-insensitive analyses

Staged Analysis– Performance Problem

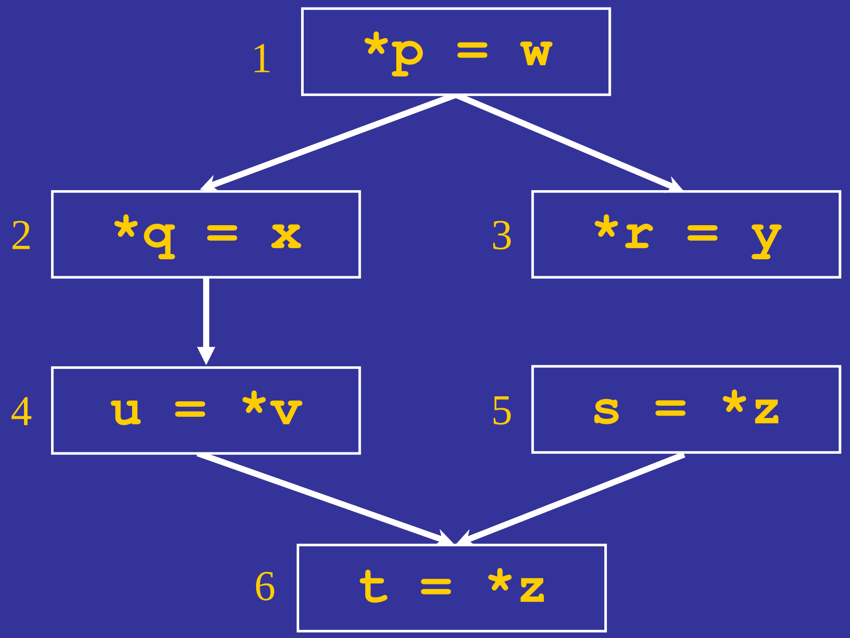
High overhead

- The number of def-use chains computed by inclusion-based analysis can result in 100's of thousands of def-use edges for programs with > 1M LOC

Optimization

- Identify access equivalent variables, those whose def-use chains are identical
- Collapse their def-use chains
- Reduces number of def-use edges by an order of magnitude

Example: CFG



Auxiliary analysis

$p \rightarrow \{a\}$

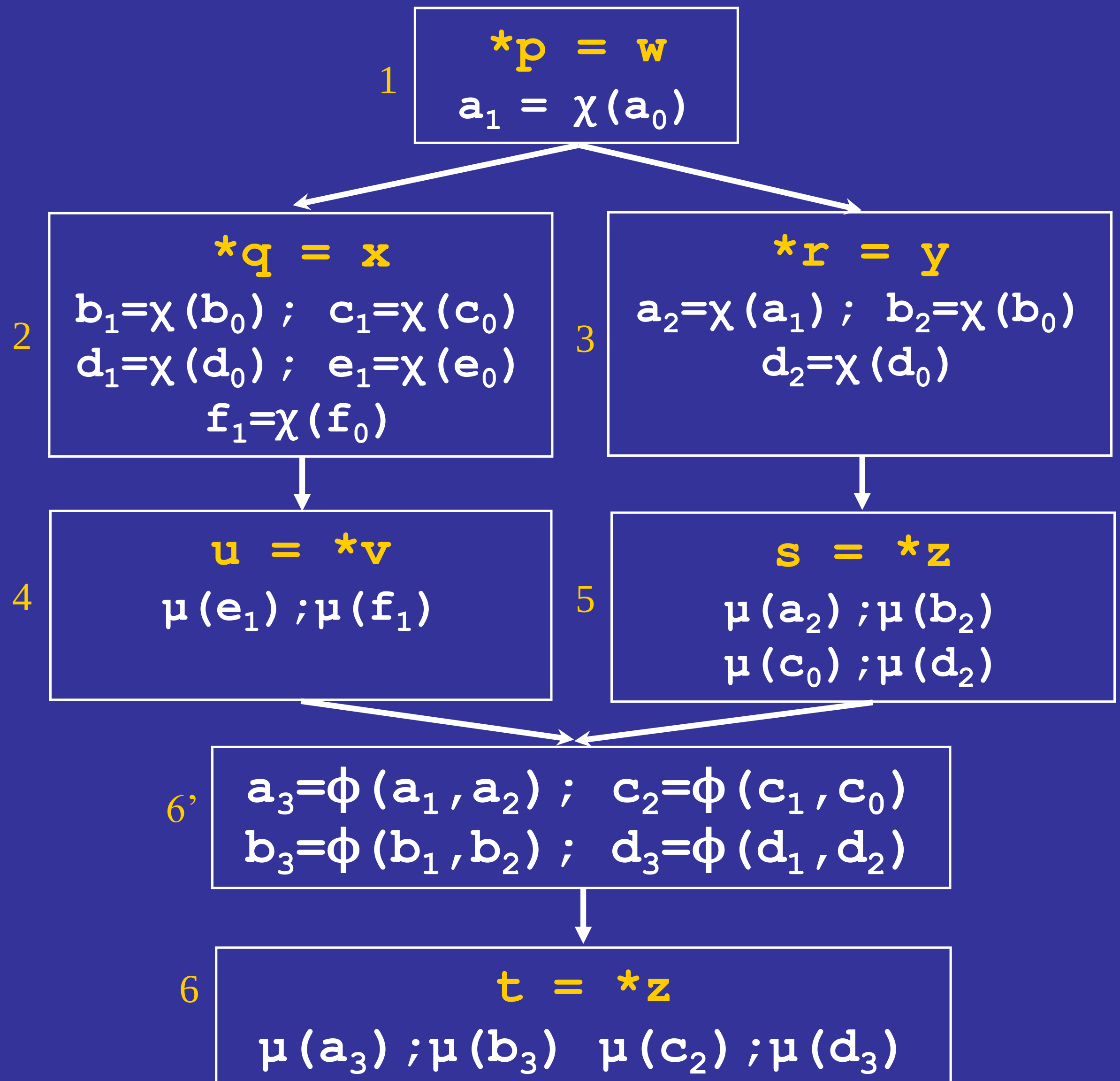
$q \rightarrow \{b, c, d, e, f\}$

$v \rightarrow \{e, f\}$

$r \rightarrow \{a, b, d\}$

$z \rightarrow \{a, b, c, d\}$

Example: SSA Form



Auxiliary analysis

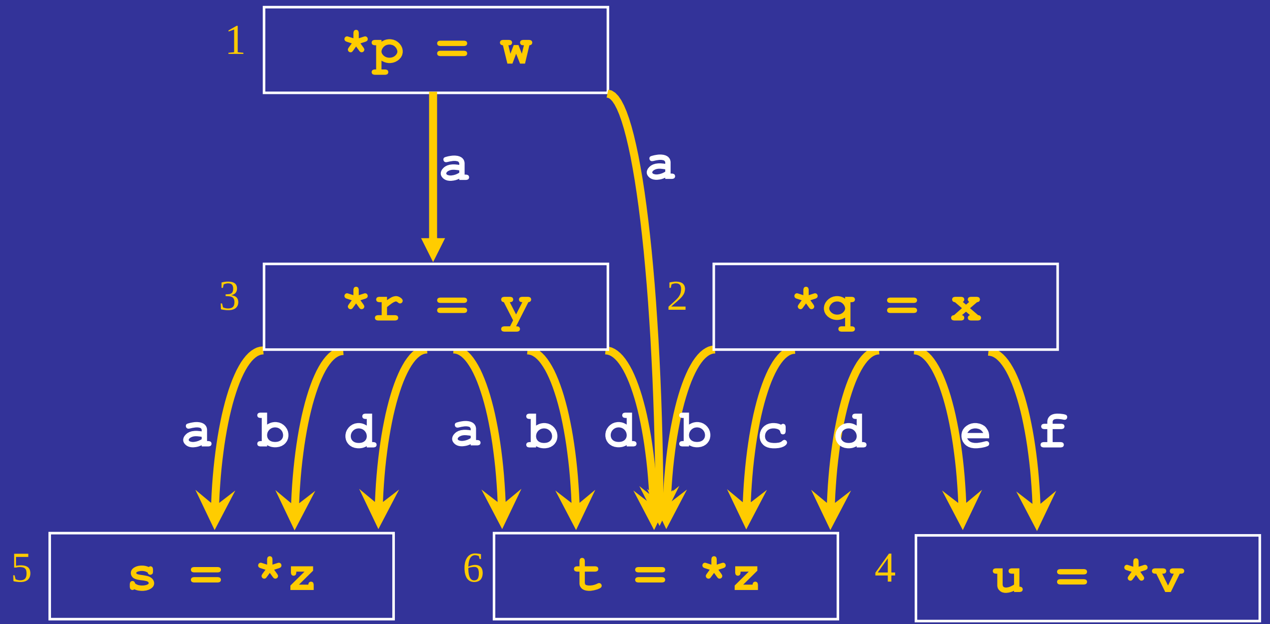
- p → {a}
- q → {b, c, d, e, f}
- v → {e, f}
- r → {a, b, d}
- z → {a, b, c, d}

Terminology

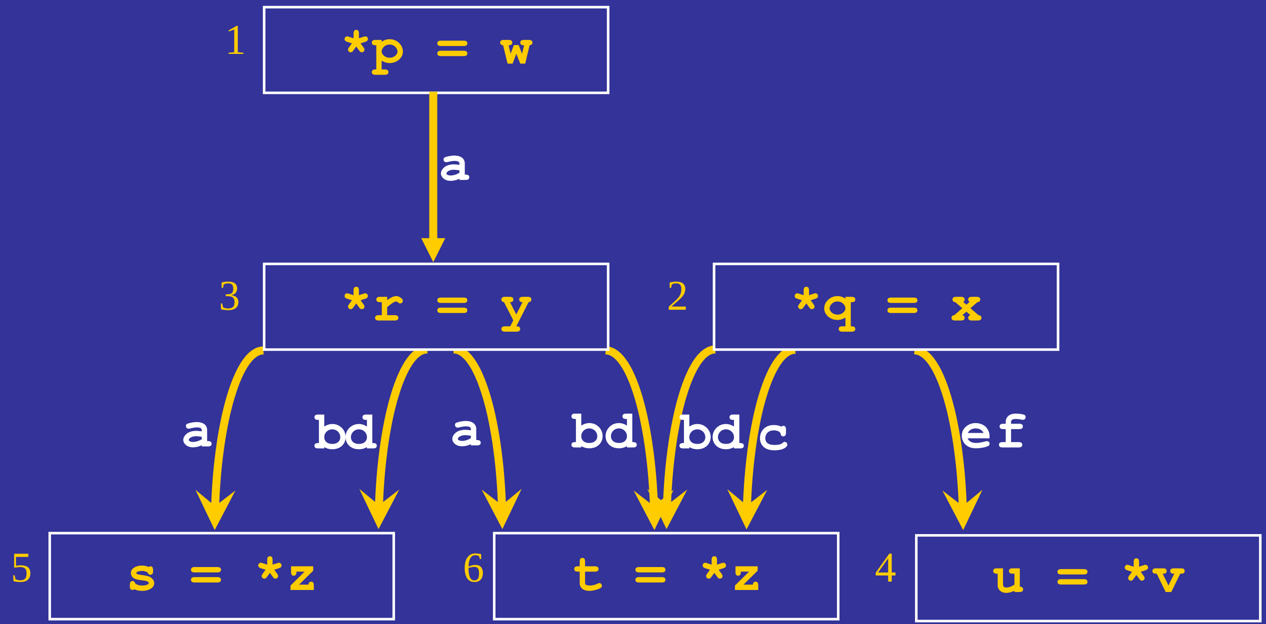
$\chi(a_0)$ is a May Def

$\mu(b_2)$ is a May Use

Example: Def-Use Graph



Example: Optimized Def-Use Graph



Applying the transfer functions

- Use IFDA, but only propagate a variable's points-to sets across edges whose label contains that variable
- Because def-use graph is an over-approximation, we might propagate information unnecessarily
  - Imprecision in a sound Auxiliary analysis affects performance, not precision

Evaluation

Comparison against state-of-the art

- Staged Flow-Sensitive Analysis (SFS) ⇐ New algorithm
- Semi-Sparse Flow-Sensitive Analysis (SSO) ⇐ Prior state-of-the-art

Details

- Implemented in LLVM using shared code base
- Both analyses are field-sensitive
- Both use BDDs to store points-to sets

Benchmarks

Name	Description	LOC
197.parser	parser	11K
300.twolf	place and route simulator	20K
ex	text processor	34K
255.vortex	object-oriented database	67K
254.gap	group theory interpreter	71K
sendmail	email server	74K
253.perlbmk	PERL interpreter	82K
nethack	text-based game	167K
python	interpeter	185K
176.gcc	C language compiler	222K
vim	text processor	268K
pine	e-mail client	342K
svn	source code control	344K
ghostscript	Postscript viewer	354K
gimp	image manipulation tool	877K
tshark	wireless network analyzer	1,946K

Results

Name	SSO (s)	SFS (s)	Speedup
197.parser	0.41	0.37	1.11
300.twolf	0.23	0.41	0.56
ex	0.35	0.40	0.88
255.vortex	0.60	0.62	0.97
254.gap	1.28	1.29	0.99
sendmail	1.21	1.00	1.21
253.perlbmk	2.30	1.57	1.46
nethack	3.16	2.64	1.20
python	120.16	6.62	18.15
176.gcc	3.74	3.46	1.08
vim	61.85	5.53	11.18
pine	347.53	82.00	4.24
svn	185.10	10.69	17.32
ghostscript	OOT	31:56.29	∞
gimp	OOT	20:22.27	∞
tshark	OOT	13:48.47	∞

Conclusions?

Results (cont)

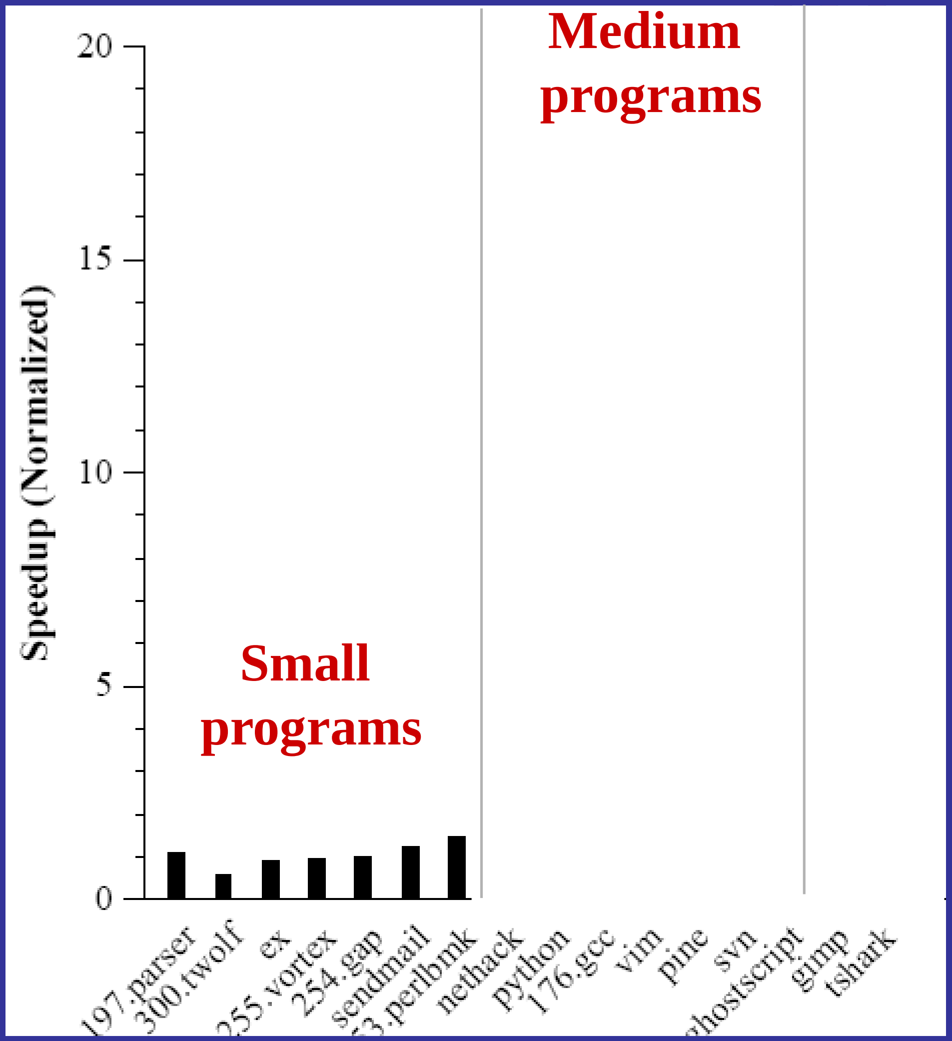
Large programs

Big picture

– Two orders of magnitude improvement

2006: 70K LOC

2011: 2,000K LOC



Related Work

Previous staged pointer analyses

- Auxiliary analysis partitions the program [Kahlon ’08]
- Auxiliary analysis prunes the program [Guyer & Lin ’03, Fink et al ’06]
- Complementary to this solution

Future Work

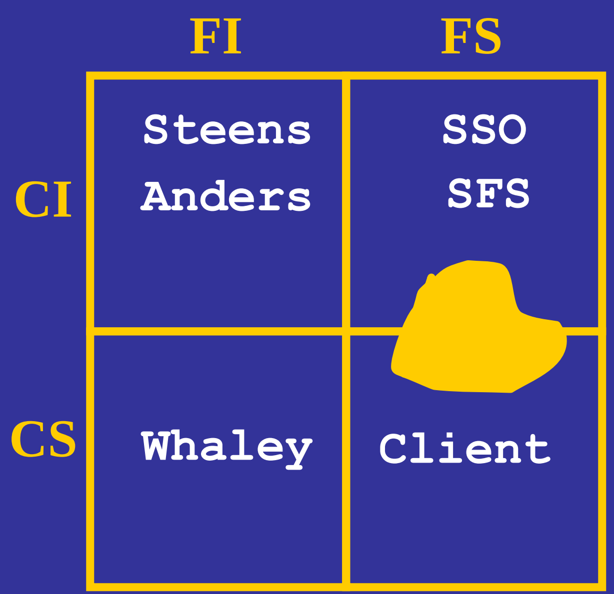
Stage the Client-Driven Pointer Analysis

- A sparse FICI will be much more scalable than the current implementation

Wild Idea

Staged Flow-Sensitive Pointer Analysis: A family of algorithms

- We can select other Auxiliary analyses
  - Instead of inclusion-based (FICI), consider a FICS analysis
  - Resulting analysis would be more precise than a FSCI analysis
  - How scalable?
  - How precise?



The Big Picture

Many dimensions of pointer analysis precision

- Flow-sensitive
- Path-sensitive
- Heap model
- Field-based
- Arrays
- Context-sensitive
- Field-sensitive
- Object-based
- Shape analysis

Language effects

- Different languages have different usage patterns
  - eg. C often passes pointers to functions (why?)
  - Das' Steensgaard's analysis with one-level flow [Das '00]
- Modern languages (Python, Java) add more dynamicism

Cottage Industry

Could churn out endless number of new analyses

- Language × precision dimension (huge space)

The Problem

Practical use

- Pointer analyses are difficult to reuse
- Pointer analyses are difficult to write and debug
  - "Around 20 pointer analyses available in LLVM"
- How much precision do you need?
  - Depends on the client and the program
  - Eg. Cisco's question: To parallelize IOS, which pointer analysis should we use?
- We need to better understand the impact on clients
  - It's hard to do this without already having multiple pointer analyses

The Vision

Turnkey Pointer Analysis

- We need pointer analysis that is so easy to use that everyone can use it
- Should be client-driven
- Needs to be much more adaptive than Guyer's Client-Driven analysis, which only looked at two dimensions of precision
- Requires careful study of multiple clients
- If successful, would be a game changer

A Step Towards a Solution

Tunable Pointer Analysis (TPA)

- Decouples control flow sensitivity from core pointer analysis algorithm

Diagnostic tool:

- TPA can simulate other pointer analysis algorithms
- TPA can be used to learn about the precision needs of client analyses
- TPA can be used to help develop and test new pointer analyses by providing a set of known results

Tunable Pointer Analysis

Useful pointer analysis:

- Sufficiently scalable for clients such as model checking and software verification

Valuable research tool:

- Guide the research community: What precisions are important?
- Identify new techniques for applying adaptive precision

Next Time

Lectures

- Modern uses of compilers
- Traditional uses of compilers

Projects

- You should have received feedback from me
- Submit next iteration of proposals by Friday April 3^{rd}

Assignments

- Assignment 4 due Friday April 10^{th}