Points-To Analysis

Derek Rayside
MIT CSAIL 6.883, Prof Ernst
drayside@mit.edu
November 14, 2005

Contents

1 Overview

Classic Research Challenge Getting precision for large program quickly.

Recently groups from McGill [2, 20] and Stanford [34, 35] have used *binary decision diagrams* (BDDs) to make precise analyses scale to large programs. Raman [28] has a brief overview of BDDs in this context.

New Research Challenges Incremental analyses. Incomplete programs. Demand-driven analyses (eg, [30]). Dynamic class loading (eg, [15, 24]).

Software Engineering Decision How to choose the right points-to analysis for the software engineering problem you're trying to solve. What costs are worth paying?

[13] [14] [23] [10]

Liang et al. [22] found Andersen-style (inclusion) analyses significantly more precise than Steensgaard-style (unification).

Lhoták and Hendren [21] found object-sensitivity Milanova et al. [25] gave the most bang for the buck for Java, vs approaches such as [34, 35].

6.883'05 MIT EECS 6.883 Program Analysis, Prof Ernst

Surveys Hind [12] Raman [28]

Dynamic Analysis Relatively little work done here. Gross [10], Mock et al. [26] show 100x improvement over static analyses, with 100x slowdown in program execution.

Context Sensitivity $\begin{cases} \textit{fun} \\ \textit{string} \end{cases}$

1.1 Complexity

Abstract from Chakaravarthy [3]:

> Given a program and two variables p and q, the goal of points-to analysis is to check if p can point to q in some execution of the program. This well-studied problem plays a crucial role in compiler optimization. The problem is known to be undecidable when dynamic memory is allowed. But the result is known only when variables are allowed to be structures. We extend the result to show that, the problem remains undecidable, even when only scalar variables are allowed. Our second result deals with a version of points-to analysis called flow-insensitive analysis, where one ignores the control flow of the program and assumes that the statements can be executed in any order. The problem is known to be NP-Hard, even when dynamic memory is not allowed and variables are scalar. We show that when the variables are further restricted to have well-defined data types, the problem is in P. The corresponding flow-sensitive version, even with further restrictions, is known to be PSPACE-Complete. Thus, our result gives some theoretical evidence that flow-insensitive analysis is easier than flow-sensitive analysis. Moreover, while most variations of the points-to analysis are known to be computationally hard, our result gives a rare instance of a non-trivial points-to problem solvable in polynomial time.

Ramalingam [27]: Aliasing is undecidable

Landi [17]: PSPACE-complete even with no procedures or memory allocations

Landi and Ryder [18]

Figure 1 A Brief History of Pointer Analysis [33] — focus on scalability and precision

	Equality-based	**Subset-based**	**Flow-sensitive**
Context-insensitive	• Weihl [32] 1980: < 1 KLOC first paper on pointer analysis • Steensgaard [31] 1996: **1+ MLOC** first scalable pointer analysis	• Andersen [1] 1994: 5 KLOC • Fähndrich et al. [7] 1998: 60 KLOC • Heintze and Tardieu [11] 2001: **1 MLOC** • Berndl et al. [2] 2003: **500 KLOC** first to use BDDs	• Choi et al. [5] 1993: 30 KLOC
Context-sensitive	• Fähndrich et al. [8] 2000: **200K**	• Whaley and Lam [35] 2004: **600 KLOC** cloning-based BDDs	• Landi and Ryder [19] 1992: 3 KLOC • Wilson and Lam [37] 1995: 30 KLOC • Whaley and Rinard [36] 1999: 80 KLOC

Horwitz [16]: Even flow-insensitive problem is NP-hard

Chakaravarthy [3]: Cannot even get a good approximation (within a constant factor) unless P=NP

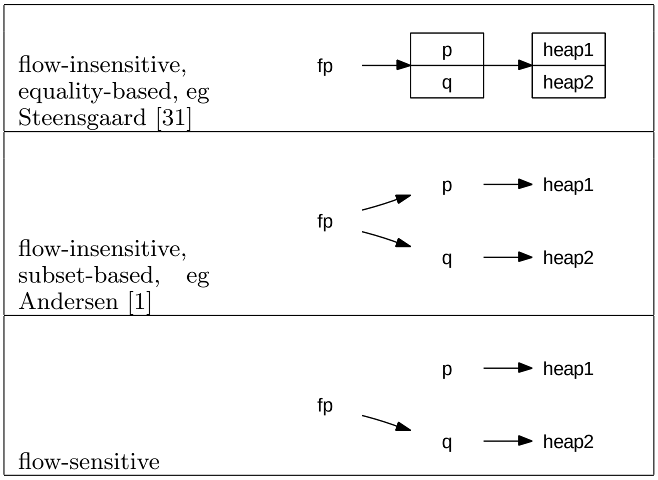
1.2 Axes of Precision

less precise	*more precise*
equivalence	subset/inclusion
flow-insensitive	flow-sensitive
context-insensitive	context-sensitive

Consider the following example [33]:

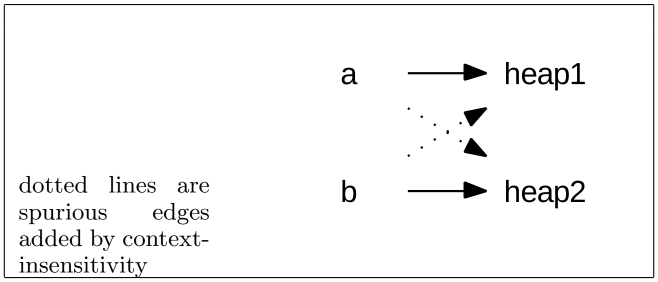
```
p = malloc();
q = malloc();
fp = &p;
fp = &q;
... = *fp;
```

What does the points-to graph look like at the end of the snippet? Depends on what analysis you do:



Another example, for context-sensitivity [33]:

```
id (x) { return x; }
foo() {
    a = malloc();
    a = id(a);
}
bar() {
    b = malloc();
    b = id(b);
}
```



2 Steensgaard [31] Example

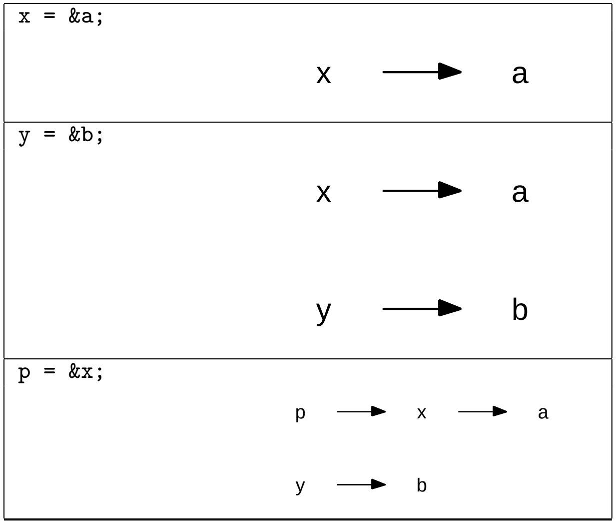
Consider the following program:

1. `x = &a;`
2. `y = &b;`
3. `p = &x;`
4. `p = &y;`

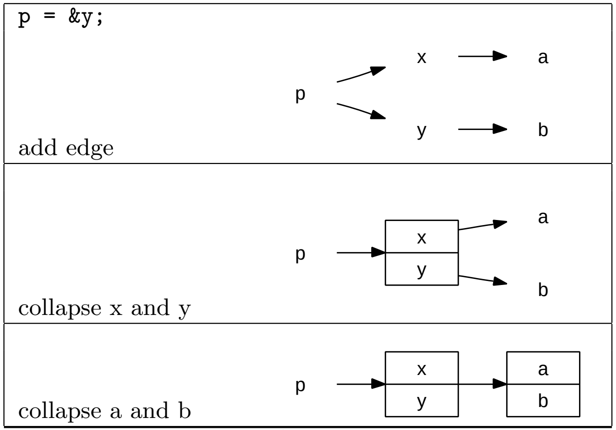
2.1 Intuitive formulation [29]

Now we'll construct the points-to graph for this program using the Steensgaard approach as formulated by Ryder [29].

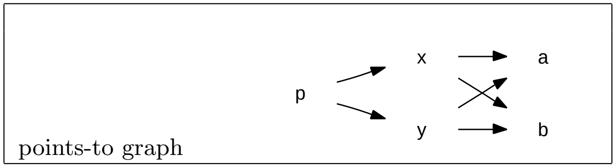
First three statements are easy:



Last statement takes more effort to process:



Resulting points-to graph over-approximates:



Why do we have to do this collapsing? It seems that the analysis would be linear in the size of the program if we didn't do collapsing. The issues is statements like `a=b;` see the example of Andersen's analysis below for why these introduce more complexity.

2.2 Type-based formulation [6]

First we assign each variable its own type:

- `x` : t_1
- `y` : t_2
- `a` : t_3
- `b` : t_4
- `p` : t_5

Then we construct the initial constraints:

1. `x = &a;` $t_1 = \mathsf{ref}(t_3 \times \_)$
2. `y = &b;` $t_2 = \mathsf{ref}(t_4 \times \_)$
3. `p = &x;` $t_5 = \mathsf{ref}(t_1 \times \_)$
4. `p = &y;` $t_5 = \mathsf{ref}(t_2 \times \_)$

Now we solve/unify the constraints. First we see:

$$t_5 = \mathsf{ref}(t_1 \times \_) = \mathsf{ref}(t_2 \times \_)$$

So we merge t_1 and t_2 into t_1. The world looks like this:

- `x` : t_1
- `y` : t_1
- `a` : t_3
- `b` : t_4
- `p` : t_5
- $t_1 = \mathsf{ref}(t_3 \times \_)$
- $t_1 = \mathsf{ref}(t_4 \times \_)$
- $t_5 = \mathsf{ref}(t_1 \times \_)$

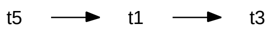
Next we see:

$$t_1 = \mathsf{ref}(t_3 \times \_) = \mathsf{ref}(t_4 \times \_)$$

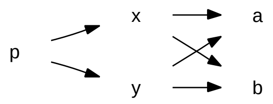
So we merge t_3 and t_4 into t_3. The world looks like this:

- `x` : t_1
- `y` : t_1
- `a` : t_3
- `b` : t_3
- `p` : t_5
- $t_1 = \mathsf{ref}(t_3 \times \_)$
- $t_5 = \mathsf{ref}(t_1 \times \_)$

We're done solving. The storage shape graph is:



If we expand that to a points-to graph we get:

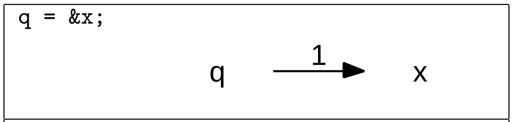


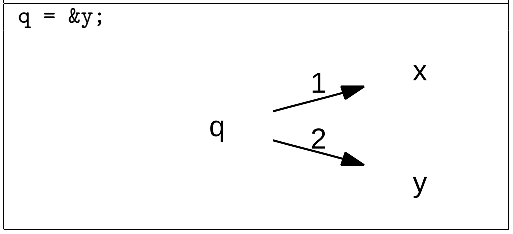
3 Andersen [1] Example [29]

Consider the following program:

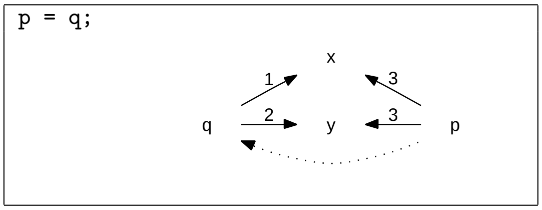
1. `q = &x;`
2. `q = &y;`
3. `p = q;`
3. `q = &z;`

First two statements are easy:

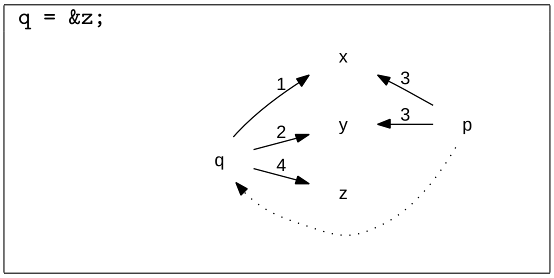




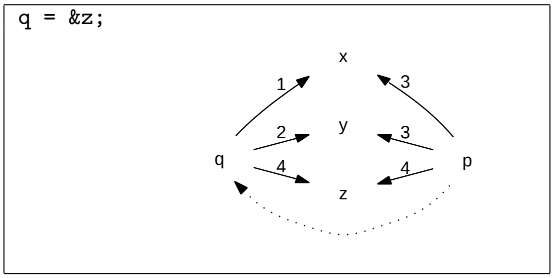
Third statement. See all the things q points to, and make p point to them as well. Add in dotted line, to remind us pts(q) $\subseteq$ pts(p).



Fourth statement. Add in $q \rightarrow z$ edge.



But dotted line reminds us that pts(q) $\subseteq$ pts(p). So we need to add $p \rightarrow z$ edge as well. This is the extra work that makes Andersen's analysis more expensive. In a Steensgaard style analysis we would have collapsed x and y at the second statement, and then we wouldn't have to worry about this extra work (although we would lose precision).



Andersen is $O(n^3)$.

Steensgaard is said to be equality-based, eg: pts(q) = pts(p).

Acknowledgements

Thanks to Greg Dennis and Rob Seater for discussions. Thanks to John Whaley for sending me his slides [33]. Thanks to Michael Ernst for sending me to Dagstuhl where I saw Barbara Ryder's talk [29].

References

[1] Lars O. Andersen. *Program Analysis and Specialization of the C Programming Language.* PhD thesis, DIKU, University of Copenhagen, 1994.

[2] Marc Berndl, Ondřej Lhoták, Feng Qian, Laurie Hendren, and Navindra Umanee. Points-to analysis using BDDs. In Rajiv Gupta, editor, *Proc.PLDI*, pages 103–114, June 2003.

[3] Venkatesan T. Chakaravarthy. New results on the computability and complexity of points-to analysis. In Greg Morrisett, editor, *30th POPL*, pages 115–125, New Orleans, Louisiana, January 2003.

[4] Craig Chambers, editor. *Proc.PLDI*, June 2004. ISBN 1-58113-807-5.

[5] Jong-Deok Choi, Michael G. Burke, and Paul R. Carini. Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects. In *20th POPL*, pages 232–245, Charleston, SC, January 1993.

[6] Amer Diwan. CSCI 5535: Homework 4, 1999. `http://www-plan.cs.colorado.edu/diwan/5535-99/hw4-soln.pdf`.

[7] Manuel Fähndrich, Jeffrey S. Foster, Zhendong Su, and Alexander Aiken. Partial online cycle elimination in inclusion constraint graphs. In *Proc.PLDI*, pages 85–96, Montreal, Canada, May 1998.

[8] Manuel Fähndrich, Jakob Rehof, and Manuvir Das. Scalable context-sensitive flow analysis using instantiation constraints. In *Proc.PLDI*, pages 253–263, Vancouver, British Columbia, Canada, June 2000.

[9] John Field and Gregor Snelting, editors. *Proc.ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, Snowbird, UT, June 2001.

[10] Axel Gross. Evaluation of dynamic points-to analysis, 2004. `http://www.complang.tuwien.ac.at/franz/sem-arbeiten/04w/semWS04_gross_0026934.pdf`.

[11] Nevin Heintze and Olivier Tardieu. Ultra-fast aliasing analysis using CLA: A million lines of C code in a second. In Mary Lou Soffa, editor, *Proc.PLDI*, Snowbird, UT, June 2001.

[12] Michael Hind. Pointer analysis: haven't we solved this problem yet? In Field and Snelting [9], pages 54–61.

[13] Michael Hind and Anthony Pioli. Assessing the effects of flow-sensitivity on pointer alias analyses. In *Proc.International Static Analysis Symposium (SAS)*, pages 57–81, 1998.

[14] ———. Evaluating the effectiveness of pointer alias analyses. *Science of Computer Programming*, 39(1): 31–55, 2001.

[15] Martin Hirzel, Amer Diwan, and Michael Hind. Pointer analysis in the presence of dynamic class loading. In Martin Odersky, editor, *Proc.18th ECOOP*, volume 3344 of *LNCS*, pages 96–122, Oslo, Norway, June 2004. Springer-Verlag. ISBN 3-540-23988-X.

[16] Susan Horwitz. Precise flow-insensitive may-alias analysis is np-hard. *Transactions on Programming Languages and Systems*, 19(1):1–6, 1997. ISSN 0164-0925.

[17] William Landi. Undecidability of static analysis. *ACM Letters on Programming Languages and Systems*, 1(4):323–337, 1992.

[18] William Landi and Barbara G. Ryder. Pointer-induced aliasing: A problem classification. In *18th POPL*, pages 83–103, Orlando, FL, January 1991.

[19] ———. A safe approximate algorithm for interprocedural pointer aliasing. In *Proc.PLDI*, pages 235–248, San Francisco, CA, June 1992.

[20] Ondřej Lhoták and Laurie Hendren. Jedd: A BDD-based relational extension of Java. In Chambers [4]. ISBN 1-58113-807-5.

[21] ———. Context-sensitive points-to analysis: is it worth it? Technical Report 2005-2, McGill University, Sable Research Group, October 2005.

[22] Donglin Liang, Maikel Pennings, and Mary Jean Harrold. Extending and evaluating flow-insensitive and context-insensitive points-to analyses for Java. In Field and Snelting [9], pages 73–79.

[23] ———. Evaluating the impact of context-sensitivity on andersen's algorithm for Java programs. In Michael Ernst and Thomas Jensen, editors, *Proc.6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, Lisbon, Portugal, September 2005.

[24] Benjamin Livshits, John Whaley, and Monica S. Lam. Reflection analysis for java. In *Proceedings of Programming Languages and Systems: Third Asian Symposium, APLAS 2005*, Tsukuba, Japan, November 2005.

[25] Ana Milanova, Atanas Rountev, and Barbara G. Ryder. Parameterized object sensitivity for points-to analysis for Java. *ACM Transactions on Software Engineering and Methodology*, 14(1):1–41, January 2005.

[26] Markus Mock, Manuvir Das, Craig Chambers, and Susan J. Eggers. Dynamic points-to sets: A comparison with static analyses and potential applications in program understanding and optimization. In Field and Snelting [9].

[27] G. Ramalingam. The undecidability of aliasing. *Transactions on Programming Languages and Systems*, 16(5):1467–1471, 1994. ISSN 0164-0925.

[28] Vishwanath Raman. Pointer analysis – a survey. CS203 UC Santa Cruz, 2004. `http://www.soe.ucsc.edu/~vishwa/publications/Pointers.pdf`.

[29] Barbara Ryder. Analysis of object-oriented programming langauges. Dagstuhl, 2003. `http://www.cs.rutgers.edu/~ryder/OOAnalRefacDagstuhl.pdf`.

[30] Manu Sridharan, Denis Gopan, Lexin Shan, and Rastislav Bodík. Demand-driven points-to analysis for Java. In Richard P. Gabriel, editor, *Proc.20th OOPSLA*, pages 59–76, San Diego, CA, October 2005. ISBN 1-59593-031-0.

[31] Bjarne Steensgaard. Points-to analysis in almost linear time. In *23rd POPL*, St. Petersburg Beach, Florida, January 1996. ISBN 0-89791-769-3.

[32] William E. Weihl. Interprocedural data flow analysis in the presence of pointers, procedure variables and label variables. In *7th POPL*, pages 83–94, Las Vegas, NV, January 1980.

[33] John Whaley. Program analysis using BDDs. Talk at MIT, March 2005.

[34] ———. *Program Analysis using Binary Decision Diagrams.* PhD thesis, Stanford University, July 2005.

[35] John Whaley and Monica Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In Chambers [4]. ISBN 1-58113-807-5.

[36] John Whaley and Martin Rinard. Compositional pointer and escape analysis for Java programs. In Linda Northrop, editor, *Proc.14th OOPSLA*, Denver, CO, November 1999.

[37] Robert P. Wilson and Monica S. Lam. Efficient context-sensitive pointer analysis for C programs. In *Proc.PLDI*, pages 1–12, La Jolla, CA, June 1995.