

Precise Shape Analysis Using Field Sensitivity

Sandeep Dasgupta

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

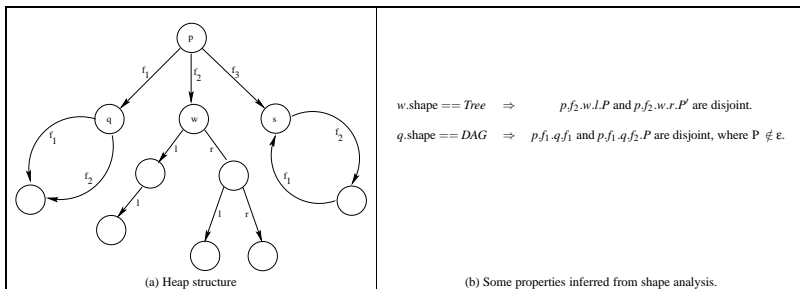
June 24, 2011

Outline

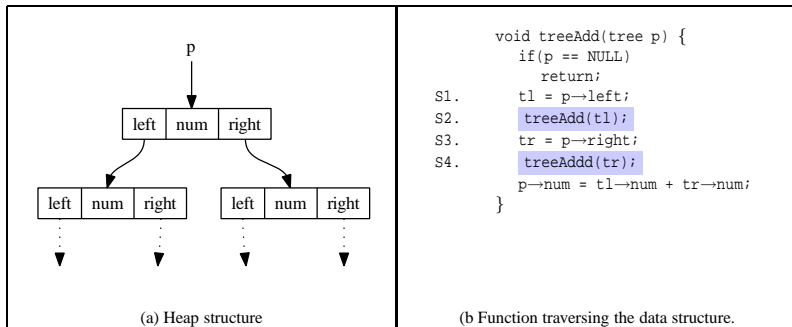
- 1 Introduction
- 2 Related Work
- 3 Motivation
- 4 Our Analysis
- 5 Properties
- 6 Future Work

Introduction

Shape Analysis : Class of techniques that statically approximate the run-time structures created on the heap.



Application




S2 and S4 can be executed in parallel.

Outline

- 1 Introduction
- 2 **Related Work**
- 3 Motivation
- 4 Our Analysis
- 5 Properties
- 6 Future Work

Related Work

- Work done by Ghiya et. al¹
 - Keeps interference and direction matrices between any two pointer variables.
 - Infer the shape of the structures as Tree, DAG or Cycle
 - Cannot handle kill information.
 - Conservatively identify the shapes.

¹Rakesh Ghiya and Laurie J. Hendren. Is it a Tree, a DAG, or a Cyclic graph? a shape analysis for heap-directed pointers in C. In Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 1-15, January 1996. 

Related Work

- Work done by Sagiv et. al.²
 - Introduce the concepts of abstraction and re-materialization.
 - Potentially exponential run-time in the number of predicates.

²Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. POPL 1999, pages 105-118, 1999.

Related Work

- Work done by Marron et. al³
 - Presents a data flow framework that uses heap graphs to model data flow values.
 - The analysis uses technique similar to re-materialization, but the re-materialization is approximate and may result in loss of precision.

³Mark Marron, Deepak Kapur, Darko Stefanovic, and Manuel Hermenegildo. A static heap analysis for shape and connectivity: unified memory analysis: the base framework. LCPC'06, pages 345-363, 2006.

Outline

- 1 Introduction
- 2 Related Work
- 3 Motivation**
- 4 Our Analysis
- 5 Properties
- 6 Future Work


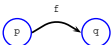

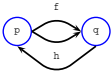
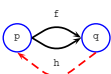
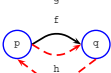
Motivation

For each pointer variable, our analysis computes the shape attribute of the data structure pointed to by the variable. We define the shape attribute $p.\text{shape}$ for a pointer p as follows:

$$p.\text{shape} = \begin{cases} \text{Cycle} & \text{If a cycle can be reached from } p \\ \text{Dag} & \text{Else if a DAG can be reached from } p \\ \text{Tree} & \text{Otherwise} \end{cases}$$

where the heap is visualized as a directed graph, and cycle and DAG have there natural graph-theoretic meanings.

Motivation

Statements	Heap Structure	Actual Shape		Field Insensitive	
		p.shape	q.shape	p.shape	q.shape
S1. $p = \text{malloc}()$ S2. $q = \text{malloc}()$		Tree	Tree	Tree	Tree
S3. $p \rightarrow f = q$		Tree	Tree	Tree	Tree
S4. $p \rightarrow h = q$		Dag	Tree	Dag	Tree
S5. $q \rightarrow g = p$		Cycle	Cycle	Cycle	Cycle
S6. $q \rightarrow g = \text{null}$		Dag	Tree	Cycle	Cycle
S7. $p \rightarrow h = \text{null}$		Tree	Tree	Cycle	Cycle

Motivation

Field insensitive shape analysis algorithms use conservative kill information and hence they are, in general, unable to infer the shape transition from cycle to DAG or from DAG to Tree.

```
S1. p = malloc();
S2. q = malloc();
S3. p→f = q;
S4. p→h = q;
S5. q→g = p;
S6. q→g = NULL;
S7. p→h = NULL;
```

(a) A code fragment

After Stmt	<i>D</i>			<i>I</i>		
S1		<i>p</i>	<i>q</i>		<i>p</i>	<i>q</i>
	<i>p</i>	1	0	<i>p</i>	1	0
	<i>q</i>	0	0	<i>q</i>	0	0
S2		<i>p</i>	<i>q</i>		<i>p</i>	<i>q</i>
	<i>p</i>	1	0	<i>p</i>	1	0
	<i>q</i>	0	1	<i>q</i>	0	1
S3		<i>p</i>	<i>q</i>		<i>p</i>	<i>q</i>
	<i>p</i>	1	1	<i>p</i>	1	1
	<i>q</i>	0	1	<i>q</i>	1	1
S4		<i>p</i>	<i>q</i>		<i>p</i>	<i>q</i>
	<i>p</i>	1	1	<i>p</i>	1	1
	<i>q</i>	0	1	<i>q</i>	1	1
S5, S6, S7		<i>p</i>	<i>q</i>		<i>p</i>	<i>q</i>
	<i>p</i>	1	1	<i>p</i>	1	1
	<i>q</i>	1	1	<i>q</i>	1	1

(b) Direction (*D*) and Interference (*I*) matrices as computed by [?]

Figure: A motivating example

Outline

- 1 Introduction
- 2 Related Work
- 3 Motivation
- 4 Our Analysis**
- 5 Properties
- 6 Future Work

Our Analysis

- Our analysis approximates any path between two variables by the first field that is dereferenced on the path.
- We use k-limiting to approximate the number of paths starting at a given field.

Our Analysis

Our analysis remembers the path information using the following:

- D_F : Modified direction matrix that stores the first fields of the paths between two pointers.
- I_F : Modified interference matrix that stores the pairs of first fields corresponding to the pairs of interfering paths.
- Boolean Variables that remember the fields directly connecting two pointer variables.

Notion of Path

- A path from p to q is the sequence of pointer fields that need to be traversed in the heap to reach from p to q .
- Path length may be unbounded, so we consider only the first field of a path.
- For a path of length one (direct path) (f_D); For a path of length greater than one (indirect path) (f_I).
- Possible to have multiple indirect paths starting with the same field : k -limiting, beyond k treat the number of paths to be ∞ .

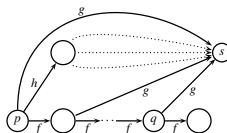
Notion of Path

```

S1.  q = p;
S2.  while (...) {
S3.    q → g = s;
S4.    q = q → f;
S5.  }

```

(a) A code fragment



(b) A possible heap graph for code in (a). Solid edges are the direct paths, dotted edges are the indirect paths.

Figure: Paths in a heap graph

Assuming the limit $k \geq 3$, the path information between p and s can be represented by the set: $\{g^D, f^{I\infty}, h^{I3}\}$.
 If $k < 3$, then the set becomes: $\{g^D, f^{I\infty}, h^{I\infty}\}$

Direction Matrix : D_F

Definition

Field sensitive Direction matrix D_F is a matrix that stores information about paths between two pointer variables. Given $p, q \in \mathcal{H}, f \in \mathcal{F}$:

ε	$\in D_F[p, p]$	where ε denotes the empty path.
f^D	$\in D_F[p, q]$	if there is a direct path f from p to q .
f^{lm}	$\in D_F[p, q]$	if there are m indirect paths starting with field f from p to q and $m \leq k$.
$f^{l\infty}$	$\in D_F[p, q]$	if there are m indirect paths starting with field f from p to q and $m > k$.

Abstraction : D_F

Let \mathcal{N} denote the set of natural numbers. We define the following partial order for approximate paths used by our analysis. For $f \in \mathcal{F}$, $m, n \in \mathcal{N}$, $n \leq m$:

$$\varepsilon \sqsubseteq \varepsilon, \quad f^D \sqsubseteq f^D, \quad f^{I^\infty} \sqsubseteq f^{I^\infty}, \quad f^{Im} \sqsubseteq f^{I^\infty}, \quad f^{In} \sqsubseteq f^{Im}.$$

The partial order is extended to set of paths S_{P_1}, S_{P_2} as:

$$S_{P_1} \sqsubseteq S_{P_2} \quad \Leftrightarrow \quad \forall \alpha \in S_{P_1}, \exists \beta \in S_{P_2} \text{ s.t. } \alpha \sqsubseteq \beta.$$

Interference Matrix : I_F

Two pointers $p, q \in \mathcal{H}$ are said to interfere if there exists $s \in \mathcal{H}$ such that both p and q have paths reaching s . Note that s could be p (or q) itself, in which case the path from p (from q) is ϵ .

Definition

Field sensitive Interference matrix I_F between two pointers captures the ways in which these pointers are interfering. For $p, q, s \in \mathcal{H}, p \neq q$, the following relation holds for D_F and I_F :

$$D_F[p, s] \times D_F[q, s] \subseteq I_F[p, q] .$$

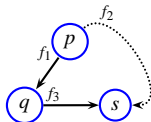
Abstraction : I_F

For pair of paths:

$$(\alpha, \beta) \sqsubseteq (\alpha', \beta') \Leftrightarrow (\alpha \sqsubseteq \alpha') \wedge (\beta \sqsubseteq \beta')$$

For set of pairs of paths R_{P_1}, R_{P_2} :

$$R_{P_1} \sqsubseteq R_{P_2} \Leftrightarrow \forall (\alpha, \beta) \in R_{P_1}, \exists (\alpha', \beta') \in R_{P_2} \text{ s.t. } (\alpha, \beta) \sqsubseteq (\alpha', \beta')$$

Example : D_F and I_F 

(a) Heap graph

D_F	p	q	s
p	$\{\epsilon\}$	$\{f_1^D\}$	$\{f_1^{I1}, f_2^{I1}\}$
q	\emptyset	$\{\epsilon\}$	$\{f_3^D\}$
s	\emptyset	\emptyset	$\{\epsilon\}$

(b) Direction Matrix

I_F	p	q	s
p	$\{\epsilon, \epsilon\}$	$\{(f_1^D, \epsilon), (f_2^{I1}, f_3^D)\}$	$\{(f_1^{I1}, \epsilon), (f_2^{I1}, \epsilon)\}$
q	$\{(\epsilon, f_1^D), (f_3^D, f_2^{I1})\}$	$\{\epsilon, \epsilon\}$	$\{(f_3^D, \epsilon)\}$
s	$\{(\epsilon, f_1^{I1}), (\epsilon, f_2^{I1})\}$	$\{(\epsilon, f_3^D)\}$	$\{\epsilon, \epsilon\}$

(c) Interference Matrix

Boolean Functions

- For each variable $p \in \mathcal{H}$, our analysis stores boolean functions p_{Dag} and p_{Cycle} to infer whether p can reach a DAG or cycle respectively in the heap.
- The boolean functions consist of the values from:
 - D_F
 - I_F
 - Field connectivity information: For $f \in \mathcal{F}, p, q \in \mathcal{H}$, field connectivity is captured by boolean variables of the form f_{pq} , which is true when f field of p points directly to q .

Boolean Functions

- The shape of p , $p.\text{shape}$, can be obtained by evaluating the functions for the attributes p_{Cycle} and p_{Dag} , and using following Table.







Table: Determining shape from boolean attributes

p_{Cycle}	p_{Dag}	$p.\text{shape}$
True	Don't Care	Cycle
False	True	DAG
False	False	Tree

Analysis Framework

- We use forward data flow analysis framework, where the data flow values are the D_F , I_F and the boolean variables.
- We do not evaluate the boolean functions immediately, but associate the unevaluated functions with the statements. When we want to find out the shape at a given statement, only then we evaluate the function using the D_F and I_F matrices, and the values of boolean variables at that statement.
- Field connectivity information is updated directly by the statement.

Motivational Example Revisited

Statements	Heap Structure	Boolean Functions	Shape Inference	
			p.shape	q.shape
S1. p = malloc() S2. q = malloc()		$p_{\text{cycle}} = \text{False}$ $p_{\text{dag}} = \text{False}$ $q_{\text{cycle}} = \text{False}$ $q_{\text{dag}} = \text{False}$	Tree	Tree
S3. p → f = q		$p_{\text{dag}} = f_{pq} \wedge I_F[p, q] > 1$ $f_{pq} = \text{True}$	Tree	Tree
S4. p → h = q		$p_{\text{dag}} = (f_{pq} \wedge I_F[p, q] > 1) \vee (h_{pq} \wedge I_F[p, q] > 1)$ $f_{pq} = \text{True}$ $h_{pq} = \text{True}$	Dag	Tree
S5. q → g = p		$p_{\text{cycle}} = (g_{qp} \wedge D_F[p, q] \geq 1)$ $p_{\text{dag}} = (f_{pq} \wedge I_F[p, q] > 1) \vee (h_{pq} \wedge I_F[p, q] > 1)$ $f_{pq} = \text{True}$ $h_{pq} = \text{True}$ $g_{qp} = \text{True}$	Cycle	Cycle
S6. q → g = null		$p_{\text{cycle}} = (g_{qp} \wedge D_F[p, q] \geq 1)$ $p_{\text{dag}} = (f_{pq} \wedge I_F[p, q] > 1) \vee (h_{pq} \wedge I_F[p, q] > 1)$ $f_{pq} = \text{True}$ $h_{pq} = \text{True}$ $g_{qp} = \text{False}$	Dag	Tree
S7. p → h = null		$p_{\text{cycle}} = (g_{qp} \wedge D_F[p, q] \geq 1)$ $p_{\text{dag}} = (f_{pq} \wedge I_F[p, q] > 1) \vee (h_{pq} \wedge I_F[p, q] > 1)$ $f_{pq} = \text{True}$ $h_{pq} = \text{False}$ $g_{qp} = \text{False}$	Tree	Tree

Outline

- 1 Introduction
- 2 Related Work
- 3 Motivation
- 4 Our Analysis
- 5 Properties**
- 6 Future Work

Need for Boolean Variables

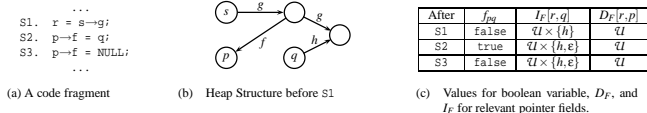


Figure: Using boolean variables to improve precision

After S2,

$$r_{\text{Dag}} = (f_{pq} \wedge (|I_F[r, q]| > 1))$$

$$f_{pq} = \mathbf{True}$$

Storage Requirement

Space requirement for D_F : $O(n^2 * m)$

Space requirement for I_F : $O(n^2 * m^2)$

Height and width of the expression tree : Polynomial

Outline

- 1 Introduction
- 2 Related Work
- 3 Motivation
- 4 Our Analysis
- 5 Properties
- 6 Future Work**

Future Work

- We use a very simple inter procedural framework to handle function calls, that computes safe approximate summaries to reach fix point . Our next challenge is to develop a better inter procedural analysis to handle function calls more precisely.
- To extend our shape analysis technique to handle more of frequently occurring programming patterns.
- To develop a prototype model using GCC framework to show the effectiveness on large benchmarks.

Thank You !!!